Package 'semPower'

October 15, 2026

Type Package

Title Power Analyses for SEM

Version 2.2.0

Description Provides a-priori, post-hoc, and compromise power-analyses for structural equation models (SEM).

License LGPL

URL https://github.com/moshagen/semPower

BugReports https://github.com/moshagen/semPower/issues

Imports graphics, grDevices, stats, utils

Suggests bookdown, covsim, doFuture, foreach, future, knitr, lavaan, mnonr, progressr, rmarkdown,

VignetteBuilder knitr

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Morten Moshagen [aut, cre] (ORCID: <https://orcid.org/0000-0002-2929-7288>), Martina Bader [aut] (ORCID: <https://orcid.org/0000-0002-5706-8933>)

Maintainer Morten Moshagen <morten.moshagen@uni-ulm.de>

Repository CRAN

Date/Publication 2026-10-15 09:20:02 UTC

Contents

checkBounded 3
checkComparisonModel 4
checkDataGenerationTypes 4
checkEllipsis 5
checkMissingTypes 5
checkNullEffect 6

checkPositive . . . 6
checkPositiveDefinite . . . 7
checkPowerTypes . . . 7
checkSquare . . . 8
checkSymmetricSquare . . . 8
doSim . . . 9
genData . . . 9
genData.IG . . . 10
genData.mnonr . . . 11
genData.normal . . . 12
genData.RK . . . 13
genData.VM . . . 14
genLambda . . . 15
genModelString . . . 16
getAGFI.F . . . 17
getBetadiff . . . 17
getCFI.Sigma . . . 18
getCFI.Sigma.mgroups . . . 19
getChiSquare.F . . . 20
getChiSquare.NCP . . . 20
getDiscrepancyFunctionFromFittingFunction . . . 21
getErrorDiff . . . 21
getF . . . 22
getF.AGFI . . . 23
getF.GFI . . . 23
getF.Mc . . . 24
getF.RMSEA . . . 24
getF.Sigma . . . 25
getFittingFunctionFromEstimator . . . 25
getFormattedResults . . . 26
getFormattedSimulationResults . . . 26
getGFI.F . . . 27
getIndices.F . . . 27
getKSdistance . . . 28
getLavOptions . . . 29
getMc.F . . . 29
getNCP . . . 30
getPhi.B . . . 30
getPsi.B . . . 31
getRMSEA.F . . . 32
getSRMR.Sigma . . . 33
getSRMR.Sigma.mgroups . . . 33
getWLSv . . . 34
makeRestrictionsLavFriendly . . . 34
orderLavCov . . . 35
orderLavMu . . . 35
orderLavResults . . . 36
powerPrepare . . . 36

semPower 38
semPower.aPriori 41
semPower.compromise 43
semPower.genSigma 45
semPower.getDf 49
semPower.postHoc 50
semPower.powerARMA 53
semPower.powerAutoreg 63
semPower.powerBifactor 72
semPower.powerCFA 80
semPower.powerCLPM 86
semPower.powerLav 96
semPower.powerLGCM 101
semPower.powerLI 110
semPower.powerMediation 119
semPower.powerMI 126
semPower.powerPath 134
semPower.powerPlot 139
semPower.powerPlot.byEffect 140
semPower.powerPlot.byN 141
semPower.powerRegression 142
semPower.powerRICLPM 150
semPower.showPlot 163
simulate 164
summary.semPower.aPriori 167
summary.semPower.compromise 168
summary.semPower.postHoc 168
validateInput 169

Index **171**

`checkBounded` *checkBounded*

Description

Checks whether x is defined and lies within the specified bound, stop otherwise.

Usage

```
checkBounded(x, message = NULL, bound = c(0, 1), inclusive = FALSE)
```

Arguments

`x`	x
`message`	identifier for x
`bound`	the boundaries, array of size two
`inclusive`	whether x might lie on boundary

checkComparisonModel *checkComparisonModel*

Description

Checks whether comparison is one of 'restricted' or 'saturated' (or respective shortcuts), stop otherwise.

Usage

```
checkComparisonModel(comparison)
```

Arguments

comparison	comparison

Value

Returns cleaned comparison

checkDataGenerationTypes *checkDataGenerationTypes*

Description

Checks whether data generation type is one of 'normal', 'IG', 'mnonr', 'RK', or 'VM', stop otherwise.

Usage

```
checkDataGenerationTypes(type)
```

Arguments

type	type

Value

Returns cleaned data generation type

checkEllipsis *checkEllipsis*

Description

Checks whether ... contains arguments related to loadings and to power, stop otherwise.

Usage

```
checkEllipsis(...)
```

Arguments

...	the parameters to search.

checkMissingTypes *checkMissingTypes*

Description

Checks whether missing generation type is one of `'mcar'`, `'mar'`, or `'nmar'`, stop otherwise.

Usage

```
checkMissingTypes(type)
```

Arguments

type	type

Value

Returns cleaned data generation type

checkNullEffect *checkNullEffect*

Description

Checks whether nullEffect is one of the valid effects, stop otherwise.

Usage

```
checkNullEffect(nullEffect, valid, message = NULL)
```

Arguments

nullEffect	nullEffect
valid	vector of valid effects
message	message

Value

Returns cleaned nullEffect

checkPositive *checkPositive*

Description

Checks whether x is defined and a positive number, stop otherwise.

Usage

```
checkPositive(x, message = NULL)
```

Arguments

x	x
message	identifier for x

checkPositiveDefinite *checkPositiveDefinite*

Description

Checks whether x is positive definite, stop otherwise.

Usage

```
checkPositiveDefinite(x, message = NULL, stop = TRUE)
```

Arguments

x	x
message	identifier for x
stop	whether to stop or to throw a warning

checkPowerTypes *checkPowerTypes*

Description

Checks whether type is one of 'a-priori', 'post-hoc', or 'compromise' (or respective shortcuts), stop otherwise.

Usage

```
checkPowerTypes(type)
```

Arguments

type	type

Value

Returns cleaned type

checkSquare *checkSquare*

Description

Checks whether x is a square matrix, stop otherwise.

Usage

```
checkSquare(x, message = NULL)
```

Arguments

x	x
message	identifier for x

checkSymmetricSquare *checkSymmetricSquare*

Description

Checks whether x is a symmetric square matrix, stop otherwise.

Usage

```
checkSymmetricSquare(x, message = NULL)
```

Arguments

x	x
message	identifier for x

doSim *doSim*

Description

Performs simulation on single data set

Usage

```
doSim(
  r,
  simData,
  isMultigroup = FALSE,
  modelH0,
  modelH1,
  lavOptions,
  lavOptionsH1
)
```

Arguments

r	replication id
simData	list of datafiles
isMultigroup	multigroup flag
modelH0	lavaan model string defining the (incorrect) analysis model.
modelH1	lavaan model string defining the comparison model. If omitted, the saturated model is the comparison model.
lavOptions	a list of additional options passed to lavaan, e. g., list(estimator = 'mlm') to request robust ML estimation
lavOptionsH1	lavoptions when fitting modelH1. If NULL, the same as lavOptions.

Value

list

genData *genData*

Description

Generates random data from population variance-covariance matrix and population means, either from a multivariate normal distribution, or using one of various approaches to generate non-normal data.

Usage

```
genData(
  N = NULL,
  Sigma = NULL,
  mu = NULL,
  nSets = 1,
  gIdx = NULL,
  modelH0 = NULL,
  simOptions = NULL
)
```

Arguments

N	sample size.
Sigma	population covariance matrix.
mu	population means.
nSets	number of data sets to generate
gIdx	if not NULL, add gIdx as numeric group index as additional variable to generated data
modelH0	a lavaan model string, only used to determine the number of factors when type = 'RK'
simOptions	additional arguments specifying the data generation routine

Value

Returns the generated data

Examples

```
## Not run:
gen <- semPower.genSigma(Phi = .2, loadings = list(rep(.5, 3), rep(.7, 3)))
data <- genData(N = 500, Sigma = gen$Sigma)

## End(Not run)
```

---

genData.IG *genData.IG*

Description

Generates random data conforming to a population variance-covariance matrix using the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals.

Usage

```
genData.IG(N = NULL, Sigma = NULL, nSets = 1, skewness = NULL, kurtosis = NULL)
```

Arguments

N	sample size.
Sigma	population covariance matrix.
nSets	number of data sets to generate
skewness	vector specifying skewness for each variable
kurtosis	vector specifying excess kurtosis for each variable

Details

This function is a wrapper for the respective function of the covsim package.

For details, see Foldnes, N. & Olsson, U. H. (2016) A Simple Simulation Technique for Nonnormal Data with Prespecified Skewness, Kurtosis, and Covariance Matrix. *Multivariate Behavioral Research, 51*, 207-219. 10.1080/00273171.2015.1133274

Value

Returns the generated data

---

genData.mnonr *genData.mnonr*

Description

Generates random data conforming to a population variance-covariance matrix using the approach by Qu, Liu, & Zhang (2020) specifying Mardia's multivariate skewness and kurtosis.

Usage

```
genData.mnonr(
  N = NULL,
  Sigma = NULL,
  nSets = 1,
  skewness = NULL,
  kurtosis = NULL
)
```

Arguments

N	sample size.
Sigma	population covariance matrix.
nSets	number of data sets to generate
skewness	multivariate skewness. May not be negative.
kurtosis	multivariate kurtosis. Must be >= 1.641 skewness + p (p + 0.774), where p is the number of variables.

Details

This function is a wrapper for the respective function of the mnonr package.

For details, see Qu, W., Liu, H., & Zhang, Z. (2020). A method of generating multivariate non-normal random numbers with desired multivariate skewness and kurtosis. *Behavior Research Methods, 52*, 939-946. doi: 10.3758/s13428-019-01291-5

Value

Returns the generated data

---

genData.normal *genData.normal*

Description

Generates multivariate normal random data conforming to a population variance-covariance matrix.

Usage

```
genData.normal(N = NULL, Sigma = NULL, nSets = 1)
```

Arguments

N	sample size.
Sigma	population covariance matrix.
nSets	number of data sets to generate

Value

Returns the generated data

genData.RK *genData.RK*

Description

Generates random data conforming to a population variance-covariance matrix using the approach by Ruscio & Kaczetow (2008) specifying distributions for the marginals.

Usage

```
genData.RK(
  N = NULL,
  Sigma = NULL,
  nSets = 1,
  distributions = NULL,
  modelH0 = NULL,
  maxIter = 10
)
```

Arguments

N	sample size.
Sigma	population covariance matrix.
nSets	number of data sets to generate
distributions	a list specifying the population distribution and additional arguments in a list either to apply to all variables (e.g. `list(rchisq, list(df = 2))`) or a list of lists specifying the distributions for each variable. See examples.
modelH0	a `lavaan` model string, only used to determine the number of factors.
maxIter	maximum number of iterations, defaults to 10.

Details

This function is based on the implementation by Ruscio & Kaczetow (2008).

For details, see Ruscio, J., & Kaczetow, W. (2008). Simulating multivariate nonnormal data using an iterative algorithm. *Multivariate Behavioral Research, 43*, 355-381.

Value

Returns the generated data

Examples

```
## Not run:
distributions <- list(
  list('rchisq', list(df = 2)),
  list('runif', list(min = 0, max = 1)),
  list('rexp', list(rate = 1))
)
data <- genData.ruscio(N = 100, Sigma = diag(3),
                       distributions = distributions,
                       modelH0 = 'f =~ x1 + x2 + x3')

distributions <- list(
  list('rnorm', list(mean = 0, sd = 10)),
  list('runif', list(min = 0, max = 1)),
  list('rbeta', list(shape1 = 1, shape2 = 2)),
  list('rexp', list(rate = 1)),
  list('rpois', list(lambda = 4)),
  list('rbinom', list(size = 1, prob = .5))
)
data <- genData.ruscio(N = 100, Sigma = diag(6),
                       distributions = distributions,
                       modelH0 = 'f1=~x1+x2+x3\nf2=~x4+x5+x6')


## End(Not run)
```

---

genData.VM *genData.VM*

Description

Generates random data conforming to a population variance-covariance matrix using the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals.

Usage

```
genData.VM(N = NULL, Sigma = NULL, nSets = 1, skewness = NULL, kurtosis = NULL)
```

Arguments

N	sample size.
Sigma	population covariance matrix.
nSets	number of data sets to generate
skewness	vector specifying skewness for each variable
kurtosis	vector specifying excess kurtosis for each variable

Details

This function is a slightly adapted copy of `lavaan`'s ValeMaurelli1983 implementation that avoids computing the intermediate correlation for each data sets and uses Sigma as input.

For details, see Vale, C. & Maurelli, V. (1983). Simulating multivariate nonnormal distributions. *Psychometrika, 48*, 465-471.

Value

Returns the generated data

`genLambda` *genLambda*

Description

Generate a loading matrix Lambda from various shortcuts, each assuming a simple structure. Either define `loadings`, or define `nIndicator` and `loadM` (and optionally `loadSD`), or define `nIndicator` and `loadMinMax`.

Usage

```
genLambda(
  loadings = NULL,
  nIndicator = NULL,
  loadM = NULL,
  loadSD = NULL,
  loadMinMax = NULL
)
```

Arguments

`loadings`	A list providing the loadings by factor, e. g. `list(c(.4, .5, .6), c(7, .8, .8))` to define two factors with three indicators each with the specified loadings. The vectors must not contain secondary loadings.
`nIndicator`	Vector indicating the number of indicators for each factor, e. g. `c(4, 6)` to define two factors with 4 and 6 indicators, respectively
`loadM`	Either a vector giving the mean loadings for each factor or a single number to use for every loading.
`loadSD`	Either a vector giving the standard deviation of loadings for each factor or a single number, for use in conjunction with `loadM`. If `NULL`, SDs are set to zero. Otherwise, loadings are sampled from a normal distribution.
`loadMinMax`	A list giving the minimum and maximum loading for each factor or a vector to apply to all factors. If set, loadings are sampled from a uniform distribution.

Value

The loading matrix Lambda.

genModelString *genModelString*

Description

Creates lavaan model strings from model matrices.

Usage

```
genModelString(
  Lambda = NULL,
  Phi = NULL,
  Beta = NULL,
  Psi = NULL,
  Theta = NULL,
  tau = NULL,
  Alpha = NULL,
  useReferenceIndicator = !is.null(Beta),
  metricInvariance = NULL,
  nGroups = 1
)
```

Arguments

Lambda	Factor loading matrix.
Phi	Factor correlation (or covariance) matrix. If NULL, all factors are orthogonal.
Beta	Regression slopes between latent variables (all-y notation).
Psi	Variance-covariance matrix of latent residuals when Beta is specified. If NULL, a diagonal matrix is assumed.
Theta	Variance-covariance matrix of manifest residuals. If NULL and Lambda is not a square matrix, Theta is diagonal so that the manifest variances are 1. If NULL and Lambda is square, Theta is 0.
tau	Intercepts. If NULL and Alpha is set, these are assumed to be zero.
Alpha	Factor means. If NULL and tau is set, these are assumed to be zero.
useReferenceIndicator	Whether to identify factors in accompanying model strings by a reference indicator (TRUE) or by setting their variance to 1 (FALSE). When Beta is defined, a reference indicator is used by default, otherwise the variance approach.
metricInvariance	A list containing the factor indices for which the accompanying model strings should apply metric invariance labels, e.g. list(c(1, 2), c(3, 4)) to assume invariance for f1 and f2 as well as f3 and f4.
nGroups	(defaults to 1) If > 1 and metricInvariance = TRUE, group specific labels will be used in the measurement model.

Value

A list containing the following `lavaan` model strings:

`modelPop`	population model
`modelTrue`	"true" analysis model freely estimating all non-zero parameters.
`modelTrueCFA`	similar to `modelTrue`, but purely CFA based and thus omitting any regression relationships.

`getAGFI.F` *getAGFI.F*

Description

Computes AGFI from the minimum of the ML-fit-function.

Usage

```
getAGFI.F(Fmin, df, p)
```

Arguments

`Fmin`	minimum of the ML-fit-function
`df`	model degrees of freedom
`p`	number of observed variables

Value

Returns AGFI

`getBetadiff` *getBetadiff*

Description

get squared difference between requested and achieved beta on a logscale

Usage

```
getBetadiff(
  cN,
  critChi,
  logBetaTarget,
  fmin,
  df,
  weights = NULL,
  simulatedPower = FALSE,
  pkgEnv = NULL,
  ...
)
```

Arguments

cN	current N
critChi	critical chi-square associated with chosen alpha error
logBetaTarget	log(desired beta)
fmin	minimum of the ML fit function
df	the model degrees of freedom
weights	sample weights for multiple group models
simulatedPower	whether to perform a simulated (TRUE) (rather than analytical, FALSE) power analysis.
pkgEnv	local pkgEnv containing iterationCounter.
...	other parameter passed to simulate()

Value

squared difference requested and achieved beta on a log scale

---

getCFI.Sigma *getCFI.Sigma*

Description

Computes CFI given the model-implied and the observed (or population) covariance matrix: `CFI = (F_null - F_hyp) / F_null`.

Usage

```
getCFI.Sigma(SigmaHat, S, muHat = NULL, mu = NULL, fittingFunction = "ML")
```

Arguments

SigmaHat	model implied covariance matrix
S	observed (or population) covariance matrix
muHat	model implied mean vector
mu	observed (or population) mean vector
fittingFunction	whether to use ML or WLS

Value

Returns CFI

---

getCFI.Sigma.mgroups *getCFI.Sigma.mgroups*

Description

Computes CFI given the model-implied and the observed (or population) covariance matrix for multiple group models. CFI = (F_null - F_hyp) / F_null applying multiple group sampling weights to F_hyp and F_null.

Usage

```
getCFI.Sigma.mgroups(
  SigmaHat,
  S,
  muHat = NULL,
  mu = NULL,
  N,
  fittingFunction = "ML"
)
```

Arguments

SigmaHat	a list of model implied covariance matrix
S	a list of observed (or population) covariance matrix
muHat	model implied mean vector
mu	observed (or population) mean vector
N	a list of group weights
fittingFunction	whether to use ML or WLS

Value

Returns CFI

getChiSquare.F *getChiSquare.F*

Description

Computes the (Wishart-) chi-square from the population minimum of the fit-function: `chi-square = (N - 1) * F0 + df = ncp + df`. Note that F0 is the population minimum. Using F_hat would give `chi-square = (N - 1) * F_hat`.

Usage

```
getChiSquare.F(Fmin, n, df)
```

Arguments

`Fmin`	population minimum of the fit-function (can be a list for multiple group models).
`n`	number of observations (can be a list for multiple group models).
`df`	model degrees of freedom

Value

Returns chi-square

getChiSquare.NCP *getChiSquare.NCP*

Description

Computes chi-square from the non-centrality parameter: `chi-square = ncp + df`.

Usage

```
getChiSquare.NCP(NCP, df)
```

Arguments

`NCP`	non-centrality parameter
`df`	model degrees of freedom

Value

Returns chi-square

getDiscrepancyFunctionFromFittingFunction *getDiscrepancyFunctionFromFittingFunction*

Description

get proper discrepancy function (to measure F0) from fitting function (to obtain SigmaHat)

Usage

```
getDiscrepancyFunctionFromFittingFunction(fittingFunction)
```

Arguments

fittingFunction	fittingFunction

Value

discrepancy function

getErrorDiff *getErrorDiff*

Description

Determine the squared log-difference between alpha and beta error given a certain chi-square value from central chi-square(df) and a non-central chi-square(df, ncp) distribution.

Usage

```
getErrorDiff(critChiSquare, df, ncp, log.abratio)
```

Arguments

critChiSquare	evaluated chi-squared value
df	the model degrees of freedom
ncp	the non-centrality parameter
log.abratio	log(alpha/beta)

Value

squared difference between alpha and beta on a log scale

getF *getF*

Description

Computes the minimum of the ML-fit-function from known fit indices.

Usage

```
getF(
  effect,
  effect.measure,
  df = NULL,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  fittingFunction = "ML"
)
```

Arguments

effect	magnitude of effect
effect.measure	measure of effect, one of 'fmin', 'rmsea', 'agfi', 'gfi', 'mc'
df	model degrees of freedom
p	number of observed variables
SigmaHat	model implied covariance matrix
Sigma	observed (or population) covariance matrix
muHat	model implied mean vector
mu	observed (or population) mean vector
fittingFunction	one of ML (default), WLS, DWLS, ULS

Value

Returns Fmin

`getF.AGFI` *getF.AGFI*

Description

Computes the minimum of the ML-fit-function from AGFI.

Usage

```
getF.AGFI(AGFI, df, p)
```

Arguments

`AGFI`	AGFI
`df`	model degrees of freedom
`p`	number of observed variables

Value

Returns Fmin

`getF.GFI` *getF.GFI*

Description

Computes the minimum of the ML-fit-function from GFI.

Usage

```
getF.GFI(GFI, p)
```

Arguments

`GFI`	GFI
`p`	number of observed variables

Value

Returns Fmin

getF.Mc *getF.Mc*

Description

Computes the minimum of the ML-fit-function from Mc.

Usage

```
getF.Mc(Mc)
```

Arguments

Mc	Mc

Value

Returns Fmin

getF.RMSEA *getF.RMSEA*

Description

Computes the minimum of the ML-fit-function from RMSEA: `F_min = rmsea^2 * df`.

Usage

```
getF.RMSEA(RMSEA, df)
```

Arguments

RMSEA	RMSEA
df	model degrees of freedom

Value

Returns Fmin

getF.Sigma *getF.Sigma*

Description

Computes the minimum of the chosen fitting-function given the model-implied and the observed (or population) covariance matrix. The ML fitting function is: `F_min = tr(S %*% SigmaHat^-1) - p + ln(det(SigmaHat)) - ln(det(S))`. When a meanstructure is included, `(mu - muHat)' SigmaHat^-1 (mu - muHat)` is added. The WLS fitting function is: `F_min = (Sij - SijHat)' V (Sij - SijHat)` where V is the inverse of N times the asymptotic covariance matrix of the sample statistics (Gamma; N x ACOV(mu, vech(S))). For DWLS, V is the diagonal of the inverse of diag(NACOV), i.e. diag(solve(diag(Gamma))). For ULS, V = I. ULS has an unknown asymptotic distribution, so it is actually irrelevant, but provided for the sake of completeness.

Usage

```
getF.Sigma(SigmaHat, S, muHat = NULL, mu = NULL, fittingFunction = "ML")
```

Arguments

`SigmaHat`	model implied covariance matrix
`S`	observed (or population) covariance matrix
`muHat`	model implied mean vector
`mu`	observed (or population) mean vector
`fittingFunction`	one of `ML` (default), `WLS`, `DWLS`, `ULS`

Value

Returns Fmin

---

getFittingFunctionFromEstimator *getFittingFunctionFromEstimator*

Description

get proper fitting function (to obtain sigmaHat) for chosen estimator

Usage

```
getFittingFunctionFromEstimator(lavOptions)
```

Arguments

lavOptions	lavOptions

Value

fitting function

---

getFormattedResults *getFormattedResults*

Description

Return data.frame containing formatted results.

Usage

```
getFormattedResults(type, result, digits = 6)
```

Arguments

type	type of power analysis
result	result object (list)
digits	number of significant digits

Value

data.frame

---

getFormattedSimulationResults *getFormattedResults*

Description

Return data.frame containing formatted results.

Usage

```
getFormattedSimulationResults(object, digits = 2)
```

Arguments

object	result object (list)
digits	number of significant digits

Value

data.frame

getGFI.F *getGFI.F*

Description

Computes GFI from the minimum of the ML-fit-function.

Usage

```
getGFI.F(Fmin, p)
```

Arguments

Fmin	minimum of the ML-fit-function
p	number of observed variables

Value

Returns GFI

getIndices.F *getIndices.F*

Description

Computes known indices from the minimum of the ML-fit-function.

Usage

```
getIndices.F(
  fmin,
  df,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  N = NULL
)
```

Arguments

fmin	minimum of the ML-fit-function
df	model degrees of freedom
p	number of observed variables
SigmaHat	model implied covariance matrix
Sigma	population covariance matrix
muHat	model implied means
mu	population means
N	list of sample weights

Value

list of indices

---

getKSdistance *getKSdistance*

Description

computes average absulute KS-distance between empirical and asympotic chi-square reference distribution.

Usage

```
getKSdistance(chi, df, ncp = 0)
```

Arguments

chi	empirical distribution
df	df of reference distribution
ncp	ncp of reference distribution

Value

average absolute distance

getLavOptions *getLavOptions*

Description

returns lavaan options including defaults as set in sem() as a list to be passed to lavaan()

Usage

```
getLavOptions(lavOptions = NULL, isCovarianceMatrix = TRUE, nGroups = 1)
```

Arguments

lavOptions	additional options to be added to (or overwriting) the defaults
isCovarianceMatrix	if TRUE, also adds sample.nobs = 1000 and sample.cov.rescale = FALSE to lavoptions
nGroups	the number of groups, 1 by default

Value

a list of lavaan defaults

getMc.F *getMc.F*

Description

Computes Mc from the minimum of the ML-fit-function.

Usage

```
getMc.F(Fmin)
```

Arguments

Fmin	minimum of the ML-fit-function

Value

Returns Mc

getNCP *getNCP*

Description

Computes the non-centrality parameter from the population minimum of the fit-function (dividing by N - 1 following the Wishart likelihood): `ncp = (N - 1) * F0`.

Usage

```
getNCP(Fmin, n)
```

Arguments

`Fmin`	population minimum of the fit-function (can be a list for multiple group models).
`n`	number of observations (can be a list for multiple group models).

Value

Returns the implied NCP.

getPhi.B *getPhi.B*

Description

Computes implied correlations (completely standardized) from Beta matrix, disallowing recursive paths.

Usage

```
getPhi.B(B, lPsi = NULL)
```

Arguments

`B`	matrix of regression coefficients (all-y notation). Must only contain non-zero lower-triangular elements, so the first row only includes zeros.
`lPsi`	(lesser) matrix of residual correlations. This is not the Psi matrix, but a lesser version ignoring all variances and containing correlations off the diagonal. Can be omitted for no correlations beyond those implied by B.

Value

Returns the implied correlation matrix

Examples

```
## Not run:
# mediation model
B <- matrix(c(
  c(.00, .00, .00),
  c(.10, .00, .00),
  c(.20, .30, .00)
), byrow = TRUE, ncol = 3)
Phi <- getPhi.B(B)

# CLPM with residual correlations
B <- matrix(c(
  c(.00, .00, .00, .00),
  c(.30, .00, .00, .00),
  c(.70, .10, .00, .00),
  c(.20, .70, .00, .00)
), byrow = TRUE, ncol = 4)
lPsi <- matrix(c(
  c(.00, .00, .00, .00),
  c(.00, .00, .00, .00),
  c(.00, .00, .00, .30),
  c(.00, .00, .30, .00)
), byrow = TRUE, ncol = 4)
Phi <- getPhi.B(B, lPsi)

## End(Not run)
```

---

getPsi.B *getPsi.B*

Description

Computes the implied Psi matrix from Beta, when all coefficients in Beta should be standardized.

Usage

```
getPsi.B(B, sPsi = NULL, standResCov = TRUE)
```

Arguments

B	matrix of regression coefficients (all-y notation). May only contain non-zero values either above or below the diagonal.
sPsi	matrix of (residual) correlations/covariances. This is not the Psi matrix, but defines the desired correlations/covariances beyond those implied by B. Can be NULL for no correlations. Standardized and unstandardized residual covariances (between endogenous variables) cannot have the same value, so `standResCov` defines whether to treat these as unstandardized or as standardized.
standResCov	whether elements in `sPsi` referring to residual covariances (between endogenous variables) shall treated as correlation or as covariance.

Value

Psi

Examples

```
## Not run:
# mediation model
B <- matrix(c(
  c(.00, .00, .00),
  c(.10, .00, .00),
  c(.20, .30, .00)
), byrow = TRUE, ncol = 3)
Psi <- getPsi.B(B)

# CLPM with residual correlations
B <- matrix(c(
  c(.00, .00, .00, .00),
  c(.30, .00, .00, .00),
  c(.70, .10, .00, .00),
  c(.20, .70, .00, .00)
), byrow = TRUE, ncol = 4)
sPsi <- matrix(c(
  c(1, .00, .00, .00),
  c(.00, 1, .00, .00),
  c(.00, .00, 1, .30),
  c(.00, .00, .30, 1)
), byrow = TRUE, ncol = 4)
# so that residual cor is std
Psi <- getPsi.B(B, sPsi, standResCov = TRUE)
# so that residual cor is unsstd
Psi <- getPsi.B(B, sPsi, standResCov = FALSE)

## End(Not run)
```

`getRMSEA.F` *getRMSEA.F*

Description

Computes RMSEA from the minimum of the ML-fit-function `F_min = rmsea^2 * df`.

Usage

```
getRMSEA.F(Fmin, df, nGroups = 1)
```

Arguments

`Fmin`	minimum of the ML-fit-function
`df`	model degrees of freedom
`nGroups`	the number of groups

Value

Returns RMSEA

`getSRMR.Sigma` *getSRMR.Sigma*

Description

Computes SRMR given the model-implied and the observed (or population) covariance matrix, using the Hu & Bentler approach to standardization.

Usage

```
getSRMR.Sigma(SigmaHat, S, muHat = NULL, mu = NULL)
```

Arguments

`SigmaHat`	model implied covariance matrix
`S`	observed (or population) covariance matrix
`muHat`	model implied mean vector
`mu`	observed (or population) mean vector

Value

Returns SRMR

`getSRMR.Sigma.mgroups` *getSRMR.Sigma.mgroups*

Description

Computes SRMR given the model-implied and the observed (or population) covariance matrix for multiple group models using the Hu & Bentler approach to standardization and the MPlus approach to multiple group sampling weights (weight squared sums of residuals).

Usage

```
getSRMR.Sigma.mgroups(SigmaHat, S, muHat = NULL, mu = NULL, N)
```

Arguments

`SigmaHat`	a list of model implied covariance matrices
`S`	a list of observed (or population) covariance matrices
`muHat`	model implied mean vector
`mu`	observed (or population) mean vector
`N`	a list of group weights

Value

Returns SRMR

`getWLSv` *getWLSv*

Description

Computes the WLS weight matrix as the asymptotic covariance matrix of the sample statistics

Usage

```
getWLSv(S, mu = NULL, diag = FALSE)
```

Arguments

`S`	observed (or population) covariance matrix
`mu`	observed (or population) mean vector
`diag`	weight matrix for DWLS

Value

Returns V

`makeRestrictionsLavFriendly` *makeRestrictionsLavFriendly*

Description

This function is currently orphaned, but we keep it just in case.

Usage

```
makeRestrictionsLavFriendly(model)
```

Arguments

`model`	`lavaan` model string

Details

This function transforms a `lavaan` model string into a model string that works reliably when both equality constrains and value constrains are imposed on the same parameters. `lavaan` cannot reliably handle this case, e. g., `"a == b \\n a == 0"` will not always work. The solution is to drop the equality constraint and rather apply the value constraint on each equality constrained parameter, e. g. `"a == 0 \n b == 0"` will work.

Value

model with lavaan-friendly constrains

`orderLavCov` *orderLavCov*

Description

returns lavaan implied covariance matrix in correct order.

Usage

```
orderLavCov(lavCov = NULL)
```

Arguments

lavCov	model implied covariance matrix

Value

cov in correct order

`orderLavMu` *orderLavMu*

Description

returns lavaan implied means in correct order.

Usage

```
orderLavMu(lavMu = NULL)
```

Arguments

lavMu	model implied means

Value

mu in correct order

orderLavResults *orderLavResults*

Description

returns lavaan implied covariance matrix or mean vector in correct order.

Usage

```
orderLavResults(lavCov = NULL, lavMu = NULL)
```

Arguments

lavCov	model implied covariance matrix
lavMu	model implied means

Value

either cov or mu in correct order

powerPrepare *powerPrepare*

Description

Performs some preparations common to all types of power analyses.

Usage

```
powerPrepare(
  type = NULL,
  effect = NULL,
  effect.measure = NULL,
  alpha = NULL,
  beta = NULL,
  power = NULL,
  abratio = NULL,
  N = NULL,
  df = NULL,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  fittingFunction = "ML",
```

```
  simulatedPower = FALSE,
  modelH0 = NULL,
  nReplications = NULL,
  minConvergenceRate = NULL,
  lavOptions = NULL
)
```

Arguments

type	type of power analysis
effect	effect size specifying the discrepancy between H0 and H1 (a list for multiple group models; a vector of length 2 for effect-size differences)
effect.measure	type of effect, one of "F0", "RMSEA", "Mc", "GFI", "AGFI"
alpha	alpha error
beta	beta error; set either beta or power
power	power (=1 - beta); set either beta or power
abratio	the ratio of alpha to beta
N	the number of observations (a list for multiple group models)
df	the model degrees of freedom
p	the number of observed variables, required for effect.measure = "GFI" and effect.measure = "AGFI"
SigmaHat	model implied covariance matrix (a list for multiple group models). Use in conjunction with Sigma to define effect and effect.measure.
Sigma	observed (or population) covariance matrix (a list for multiple group models). Use in conjunction with SigmaHat to define effect and effect.measure.
muHat	model implied mean vector
mu	observed (or population) mean vector
fittingFunction	one of 'ML' (default), 'WLS', 'DWLS', 'ULS'. Defines the discrepancy function used to obtain Fmin.
simulatedPower	whether to perform a simulated (TRUE) (rather than analytical, FALSE) power analysis.
modelH0	for simulated power: lavaan model string defining the (incorrect) analysis model.
nReplications	for simulated power: number of random samples drawn.
minConvergenceRate	for simulated power: the minimum convergence rate required
lavOptions	for simulated power: a list of additional options passed to lavaan, e. g., list(estimator = 'mlm') to request robust ML estimation.

Value

list

`semPower` *semPower: Power analyses for structural equation models (SEM).*

Description

semPower allows for performing a-priori, post-hoc, and compromise power-analyses for structural equation models (SEM).

Perform a power analysis. This is a wrapper function for a-priori, post-hoc, and compromise power analyses.

Usage

```
semPower(type, ...)
```

Arguments

`type`	type of power analysis, one of `'a-priori'`, `'post-hoc'`, `'compromise'`.
`...`	other parameters related to the specific type of power analysis requested.

Details

- A-priori power analysis `semPower.aPriori` computes the required N, given an effect, alpha, power, and the model df
- Post-hoc power analysis `semPower.postHoc` computes the achieved power, given an effect, alpha, N, and the model df
- Compromise power analysis `semPower.compromise` computes the implied alpha and power, given an effect, the alpha/beta ratio, N, and the model df

In SEM, the discrepancy between H0 and H1 (the magnitude of effect) refers to the difference in fit between two models. If only one model is defined (which is the default), power refers to the global chi-square test. If both models are explicitly defined, power is computed for nested model tests. semPower allows for expressing the magnitude of effect by one of the following measures: F0, RMSEA, Mc, GFI, or AGFI.

Alternatively, the implied effect can also be computed from the discrepancy between the population (or a certain model-implied) covariance matrix defining H0 and the hypothesized (model-implied) covariance matrix from a nested model defining H1. See the examples below how to use this feature in conjunction with lavaan.

Value

list

Author(s)

Morten Moshagen `<morten.moshagen@uni-ulm.de>`

See Also

Useful links:

- https://github.com/moshagen/semPower
- Report bugs at https://github.com/moshagen/semPower/issues

semPower.aPriori(), semPower.postHoc(), semPower.compromise()

Examples

```
# a-priori power analyses using rmsea = .05 a target power (1-beta) of .80
ap1 <- semPower.aPriori(0.05, 'RMSEA', alpha = .05, beta = .20, df = 200)
summary(ap1)
# generic version
gap1 <- semPower(type = 'a-priori', 0.05, 'RMSEA', alpha = .05, beta = .20, df = 200)
summary(gap1)
# a-priori power analyses using f0 = .75 and a target power of .95
ap2 <- semPower.aPriori(0.75, 'F0', alpha = .05, power = .95, df = 200)
summary(ap2)
# create a plot showing how power varies by N (given a certain effect)
semPower.powerPlot.byN(.05, 'RMSEA', alpha=.05, df=200, power.min=.05, power.max=.99)
# post-hoc power analyses using rmsea = .08
ph <- semPower.postHoc(.08, 'RMSEA', alpha = .05, N = 250, df = 50)
summary(ph)
# generic version
gph1 <- semPower(type = 'post-hoc', .08, 'RMSEA', alpha = .05, N = 250, df = 50)
summary(gph1)
# create a plot showing how power varies by the magnitude of effect (given a certain N)
semPower.powerPlot.byEffect('RMSEA', alpha=.05, N = 100, df=200, effect.min=.001, effect.max=.10)
# compromise power analyses using rmsea = .08 and an abratio of 2
cp <- semPower.compromise(.08, 'RMSEA', abratio = 2, N = 1000, df = 200)
summary(cp)
# generic version
gcp <- semPower(type = 'compromise', .08, 'RMSEA', abratio = 2, N = 1000, df = 200)
summary(gcp)

# use lavaan to define effect through covariance matrices:
## Not run:
library(lavaan)

# define population model (= H1)
model.pop <- '
f1 =~ .8*x1 + .7*x2 + .6*x3
f2 =~ .7*x4 + .6*x5 + .5*x6
f1 ~~ 1*f1
f2 ~~ 1*f2
f1 ~~ 0.5*f2
'
# define (wrong) H0 model
model.h0 <- '
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
```

```
f1 ~~ 0*f2
'

# get population covariance matrix; equivalent to a perfectly fitting H1 model
cov.h1 <- fitted(sem(model.pop))$cov
# get covariance matrix as implied by H0 model
res.h0 <- sem(model.h0, sample.cov = cov.h1, sample.nobs = 1000,
              likelihood='wishart', sample.cov.rescale = F)
df <- res.h0@test[[1]]$df
cov.h0 <- fitted(res.h0)$cov

## do power analyses

# post-hoc
ph <- semPower.postHoc(SigmaHat = cov.h0, Sigma = cov.h1, alpha = .05, N = 1000, df = df)
summary(ph)
# => Power to reject the H1 model is > .9999 (1-beta = 1-1.347826e-08) with N = 1000 at alpha=.05

# compare:
ph$fmin * (ph$N-1)
fitmeasures(res.h1, 'chisq')
# => expected chi-square matches empirical chi-square

# a-priori
ap <- semPower.aPriori(SigmaHat = cov.h0, Sigma = cov.h1, alpha = .05, power = .80, df = df)
summary(ap)
# => N = 194 gives a power of ~80% to reject the H1 model at alpha = .05

# compromise
cp <- semPower.compromise(SigmaHat = cov.h0, Sigma = cov.h1, abratio = 1, N = 1000, df = df)
summary(cp)
# => A critical Chi-Squared of 33.999 gives balanced alpha-beta
#    error probabilities of alpha=beta=0.000089 with N = 1000.


## End(Not run)

## Not run:

ap <- semPower(type = 'a-priori',
               effect = .08, effect.measure = "RMSEA",
               alpha = .05, beta = .05, df = 200)
summary(ph)

ph <- semPower(type = 'post-hoc',
               effect = .08, effect.measure = "RMSEA",
               alpha = .05, N = 250, df = 200)
summary(ph)

cp <- semPower(type = 'compromise',
               effect = .08, effect.measure = "RMSEA",
               abratio = 1, N = 250, df = 200)
summary(ph)
```

```
## End(Not run)
```

---

semPower.aPriori *semPower.aPriori*

Description

Performs an a-priori power analysis, i. e., determines the required sample size given alpha, beta (or power: 1 - beta), df, and a measure of effect.

Usage

```
semPower.aPriori(
  effect = NULL,
  effect.measure = NULL,
  alpha,
  beta = NULL,
  power = NULL,
  N = NULL,
  df = NULL,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  fittingFunction = "ML",
  simulatedPower = FALSE,
  modelH0 = NULL,
  modelH1 = NULL,
  simOptions = NULL,
  lavOptions = NULL,
  lavOptionsH1 = lavOptions,
  ...
)
```

Arguments

effect	effect size specifying the discrepancy between the null hypothesis (H0) and the alternative hypothesis (H1). A list for multiple group models; a vector of length 2 for effect-size differences. Can be NULL if Sigma and SigmaHat are set.
effect.measure	type of effect, one of "F0", "RMSEA", "Mc", "GFI", "AGFI". Can be NULL if Sigma and SigmaHat are set.
alpha	alpha error
beta	beta error; set either beta or power.
power	power (= 1 - beta); set either beta or power.

N	a list of sample weights for multiple group power analyses, e.g. `list(1, 2)` to make the second group twice as large as the first one.
df	the model degrees of freedom. See `semPower.getDf()` for a way to obtain the df of a specific model.
p	the number of observed variables, only required for `effect.measure = "GFI"` and `effect.measure = "AGFI"`.
SigmaHat	can be used instead of `effect` and `effect.measure`: model implied covariance matrix (a list for multiple group models). Used in conjunction with `Sigma` to define the effect.
Sigma	can be used instead of `effect` and `effect.measure`: population covariance matrix (a list for multiple group models). Used in conjunction with `SigmaHat` to define effect.
muHat	can be used instead of `effect` and `effect.measure`: model implied mean vector. Used in conjunction with `mu`. If `NULL`, no meanstructure is involved.
mu	can be used instead of `effect` and `effect.measure`: observed (or population) mean vector. Use in conjunction with `muHat`. If `NULL`, no meanstructure is involved.
fittingFunction	one of `'ML'` (default), `'WLS'`, `'DWLS'`, `'ULS'`. Defines the discrepancy function used to obtain Fmin.
simulatedPower	whether to perform a simulated (`TRUE`, rather than analytical, `FALSE`) power analysis. Only available if `Sigma` and `modelH0` are defined.
modelH0	for simulated power: `lavaan` model string defining the (incorrect) analysis model.
modelH1	for simulated power: `lavaan` model string defining the comparison model. If omitted, the saturated model is the comparison model.
simOptions	a list of additional options specifying simulation details, see `simulate()` for details.
lavOptions	a list of additional options passed to `lavaan`, e. g., `list(estimator = 'mlm')` to request robust ML estimation.
lavOptionsH1	alternative options passed to `lavaan` that are only used for the H1 model. If `NULL`, identical to `lavOptions`. Probably only useful for multigroup models.
...	other parameters related to plots, notably `plotShow`, `plotShowLabels`, and `plotLinewidth`.

Value

Returns a list. Use `summary()` to obtain formatted results.

See Also

`semPower.postHoc()` `semPower.compromise()`

Examples

```
## Not run:
# determine the required sample size to reject a model showing misspecifications
# amounting to RMSEA >= .05 on 200 df with a power of 95 % on alpha = .05
ap <- semPower.aPriori(effect = .05, effect.measure = "RMSEA",
                       alpha = .05, beta = .05, df = 200)
summary(ap)

# use f0 as effect size metric
ap <- semPower.aPriori(effect = .15, effect.measure = "F0",
                       alpha = .05, power = .80, df = 200)
summary(ap)

# power analysis for to detect the difference between a model (with df = 200) exhibiting RMSEA = .05
# and a model (with df = 210) exhibiting RMSEA = .06.
ap <- semPower.aPriori(effect = c(.05, .06), effect.measure = "RMSEA",
                       alpha = .05, power = .80, df = c(200, 210))
summary(ap)

# power analysis based on SigmaHat and Sigma (nonsense example)
ap <- semPower.aPriori(alpha = .05, beta = .05, df = 5,
                       SigmaHat = diag(4), Sigma = cov(matrix(rnorm(4*1000),  ncol=4)))
summary(ap)

# multiple group example
ap <- semPower.aPriori(effect = list(.05, .10), effect.measure = "F0",
                       alpha = .05, power = .80, df = 100,
                       N = list(1, 1))
summary(ap)

# simulated power analysis (nonsense example)
ap <- semPower.aPriori(alpha = .05, beta = .05, df = 200,
                       SigmaHat = list(diag(4), diag(4)),
                       Sigma = list(cov(matrix(rnorm(4*1000), ncol=4)),
                              cov(matrix(rnorm(4*1000), ncol=4))),
                       simulatedPower = TRUE, nReplications = 100)
summary(ap)

## End(Not run)
```

`semPower.compromise` *semPower.compromise*

Description

Performs a compromise power analysis, i. e., determines the critical chi-square along with the implied alpha error and beta error , given the alpha/beta ratio, a measure of effect, N, and df

Usage

```
semPower.compromise(
  effect = NULL,
  effect.measure = NULL,
  abratio = 1,
  N,
  df = NULL,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  fittingFunction = "ML",
  ...
)
```

Arguments

effect	effect size specifying the discrepancy between the null hypothesis (H0) and the alternative hypothesis (H1). A list for multiple group models; a vector of length 2 for effect-size differences. Can be NULL if Sigma and SigmaHat are set.
effect.measure	type of effect, one of "F0", "RMSEA", "Mc", "GFI", "AGFI". Can be NULL if Sigma and SigmaHat are set.
abratio	the ratio of alpha to beta
N	the number of observations (a list for multiple group models)
df	the model degrees of freedom. See semPower.getDf() for a way to obtain the df of a specific model.
p	the number of observed variables, only required for effect.measure = "GFI" and effect.measure = "AGFI".
SigmaHat	can be used instead of effect and effect.measure: model implied covariance matrix (a list for multiple group models). Used in conjunction with Sigma to define the effect.
Sigma	can be used instead of effect and effect.measure: population covariance matrix (a list for multiple group models). Used in conjunction with SigmaHat to define effect.
muHat	can be used instead of effect and effect.measure: model implied mean vector. Used in conjunction with mu. If NULL, no meanstructure is involved.
mu	can be used instead of effect and effect.measure: observed (or population) mean vector. Use in conjunction with muHat. If NULL, no meanstructure is involved.
fittingFunction	one of 'ML' (default), 'WLS', 'DWLS', 'ULS'. Defines the discrepancy function used to obtain Fmin.
...	other parameters related to plots, notably plotShow, plotShowLabels, and plotLinewidth.

Value

Returns a list. Use `summary()` to obtain formatted results.

See Also

`semPower.aPriori()` `semPower.postHoc()`

Examples

```
## Not run:

# determine the critical value such that alpha = beta when distinguishing a model
# involving 200 df exhibiting an RMSEA >= .08 from a perfectly fitting model.
cp <- semPower.compromise(effect = .08, effect.measure = "RMSEA",
                          abratio = 1, N = 250, df = 200)
summary(cp)


## End(Not run)
```

`semPower.genSigma` *semPower.genSigma*

Description

Generate a covariance matrix (and a mean vector) and associated `lavaan` model strings based on CFA or SEM model matrices.

Usage

```
semPower.genSigma(
  Lambda = NULL,
  Phi = NULL,
  Beta = NULL,
  Psi = NULL,
  Theta = NULL,
  tau = NULL,
  Alpha = NULL,
  ...
)
```

Arguments

`Lambda`	factor loading matrix. A list for multiple group models. Can also be specified using various shortcuts, see `genLambda()`.
`Phi`	for CFA models, factor correlation (or covariance) matrix or single number giving the correlation between all factors or `NULL` for uncorrelated factors. A list for multiple group models.

Beta	for SEM models, matrix of regression slopes between latent variables (all-y notation). A list for multiple group models.
Psi	for SEM models, variance-covariance matrix of latent residuals when Beta is specified. If NULL, a diagonal matrix is assumed. A list for multiple group models.
Theta	variance-covariance matrix between manifest residuals. If NULL and Lambda is not a square matrix, Theta is diagonal so that the manifest variances are 1. If NULL and Lambda is square, Theta is 0. A list for multiple group models.
tau	vector of intercepts. If NULL and Alpha is set, these are assumed to be zero. If both Alpha and tau are NULL, no means are returned. A list for multiple group models.
Alpha	vector of factor means. If NULL and tau is set, these are assumed to be zero. If both Alpha and tau are NULL, no means are returned. A list for multiple group models.
...	other

Details

This function generates the variance-covariance matrix of the p observed variables Σ and their means μ via a confirmatory factor (CFA) model or a more general structural equation model.

In the CFA model,

$$\Sigma = \Lambda\Phi\Lambda' + \Theta$$

where Λ is the $p \cdot m$ loading matrix, Φ is the variance-covariance matrix of the m factors, and Θ is the residual variance-covariance matrix of the observed variables. The means are

$$\mu = \tau + \Lambda\alpha$$

with the p indicator intercepts τ and the m factor means α.

In the structural equation model,

$$\Sigma = \Lambda(I - B)^{-1}\Psi[(I - B)^{-1}]'\Lambda' + \Theta$$

where B is the $m \cdot m$ matrix containing the regression slopes and Ψ is the (residual) variance-covariance matrix of the m factors. The means are

$$\mu = \tau + \Lambda(I - B)^{-1}\alpha$$

In either model, the meanstructure can be omitted, leading to factors with zero means and zero intercepts.

When $\Lambda = I$, the models above do not contain any factors and reduce to ordinary regression or path models.

If Phi is defined, a CFA model is used, if Beta is defined, a structural equation model. When both Phi and Beta are NULL, a CFA model is used with $\Phi = I$, i. e., uncorrelated factors. When Phi is a single number, all factor correlations are equal to this number.

When Beta is defined and Psi is NULL, $\Psi = I$.

When Theta is NULL, Θ is a diagonal matrix with all elements such that the variances of the observed variables are 1. When there is only a single observed indicator for a factor, the corresponding element in Θ is set to zero.

Instead of providing the loading matrix Λ via Lambda, there are several shortcuts (see genLambda()):

- loadings: defines the primary loadings for each factor in a list structure, e. g. loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4)) defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading in the second factor by .8, .6, .6, and .4.
- nIndicator: used in conjunction with loadM or loadMinmax, defines the number of indicators by factor, e. g., nIndicator = c(3, 4) defines a two factor model with three and four indicators for the first and second factor, respectively. nIndicator can also be a single number to define the same number of indicators for each factor.
- loadM: defines the mean loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. loadM = c(.5, .6) defines the mean loadings of the first factor to equal .5 and those of the second factor do equal .6
- loadSD: used in conjunction with loadM, defines the standard deviations of the loadings. If omitted or NULL, the standard deviations are zero. Otherwise, the loadings are sampled from a normal distribution with N(loadM, loadSD) for each factor.
- loadMinMax: defines the minimum and maximum loading either for all factors or separately for each factor (as a list). The loadings are then sampled from a uniform distribution. For example, loadMinMax = list(c(.4, .6), c(.4, .8)) defines the loadings for the first factor lying between .4 and .6, and those for the second factor between .4 and .8.

Value

Returns a list (or list of lists for multiple group models) containing the following components:

Sigma	implied variance-covariance matrix.
mu	implied means
Lambda	loading matrix
Phi	covariance matrix of latent variables
Beta	matrix of regression slopes
Psi	residual covariance matrix of latent variables
Theta	residual covariance matrix of observed variables
tau	intercepts
Alpha	factor means
modelPop	lavaan model string defining the population model
modelTrue	lavaan model string defining the "true" analysis model freely estimating all non-zero parameters.
modelTrueCFA	lavaan model string defining a model similar to modelTrue, but purely CFA based and thus omitting any regression relationships.

Examples

```
## Not run:
# single factor model with five indicators each loading by .5
gen <- semPower.genSigma(nIndicator = 5, loadM = .5)
gen$Sigma     # implied covariance matrix
gen$modelTrue # analysis model string
gen$modelPop  # population model string

# orthogonal two factor model with four and five indicators each loading by .5
gen <- semPower.genSigma(nIndicator = c(4, 5), loadM = .5)

# correlated (r = .25) two factor model with
# four indicators loading by .7 on the first factor
# and five indicators loading by .6 on the second factor
gen <- semPower.genSigma(Phi = .25, nIndicator = c(4, 5), loadM = c(.7, .6))

# correlated three factor model with variying indicators and loadings,
# factor correlations according to Phi
Phi <- matrix(c(
  c(1.0, 0.2, 0.5),
  c(0.2, 1.0, 0.3),
  c(0.5, 0.3, 1.0)
), byrow = TRUE, ncol = 3)
gen <- semPower.genSigma(Phi = Phi, nIndicator = c(3, 4, 5),
                         loadM = c(.7, .6, .5))

# same as above, but using a factor loadings matrix
Lambda <- matrix(c(
  c(0.8, 0.0, 0.0),
  c(0.7, 0.0, 0.0),
  c(0.6, 0.0, 0.0),
  c(0.0, 0.7, 0.0),
  c(0.0, 0.8, 0.0),
  c(0.0, 0.5, 0.0),
  c(0.0, 0.4, 0.0),
  c(0.0, 0.0, 0.5),
  c(0.0, 0.0, 0.4),
  c(0.0, 0.0, 0.6),
  c(0.0, 0.0, 0.4),
  c(0.0, 0.0, 0.5)
), byrow = TRUE, ncol = 3)
gen <- semPower.genSigma(Phi = Phi, Lambda = Lambda)

# same as above, but using a reduced loading matrix, i. e.
# only define the primary loadings for each factor
loadings <- list(
  c(0.8, 0.7, 0.6),
  c(0.7, 0.8, 0.5, 0.4),
  c(0.5, 0.4, 0.6, 0.4, 0.5)
)
gen <- semPower.genSigma(Phi = Phi, loadings = loadings)
```

```
# Provide Beta for a three factor model
# with 3, 4, and 5 indicators
# loading by .6, 5, and .4, respectively.
Beta <- matrix(c(
                c(0.0, 0.0, 0.0),
                c(0.3, 0.0, 0.0),  # f2 = .3*f1
                c(0.2, 0.4, 0.0)   # f3 = .2*f1 + .4*f2
               ), byrow = TRUE, ncol = 3)
gen <- semPower.genSigma(Beta = Beta, nIndicator = c(3, 4, 5),
                         loadM = c(.6, .5, .4))

# two group example:
# correlated two factor model (r = .25 and .35 in the first and second group,
# respectively)
# the first factor is indicated by four indicators loading by .7 in the first
# and .5 in the second group,
# the second factor is indicated by five indicators loading by .6 in the first
# and .8 in the second group,
# all item intercepts are zero in both groups,
# the latent means are zero in the first group
# and .25 and .10 in the second group.
gen <- semPower.genSigma(Phi = list(.25, .35),
                         nIndicator = list(c(4, 5), c(4, 5)),
                         loadM = list(c(.7, .6), c(.5, .8)),
                         tau = list(rep(0, 9), rep(0, 9)),
                         Alpha = list(c(0, 0), c(.25, .10))
                         )
gen[[1]]$Sigma  # implied covariance matrix group 1
gen[[2]]$Sigma  # implied covariance matrix group 2
gen[[1]]$mu     # implied means group 1
gen[[2]]$mu     # implied means group 2

## End(Not run)
```

`semPower.getDf` *semPower.getDf*

Description

Determines the degrees of freedom of a given model provided as `lavaan` model string. This only returns the regular df and does not account for approaches using scaled df. This requires the `lavaan` package.

Usage

```
semPower.getDf(lavModel, nGroups = NULL, group.equal = NULL)
```

Arguments

lavModel	the lavaan model string. Can also include (restrictions on) defined parameters.
nGroups	for multigroup models: the number of groups.
group.equal	for multigroup models: vector defining the type(s) of cross-group equality constraints following the lavaan conventions (loadings, intercepts, means, residuals, residual.covariances, lv.variances, lv.covariances, regressions).

Value

Returns the df of the model.

Examples

```
## Not run:
lavModel <- '
f1 =~ x1 + x2 + x3 + x4
f2 =~ x5 + x6 + x7 + x8
f3 =~ y1 + y2 + y3
f3 ~ f1 + f2
'
semPower.getDf(lavModel)

# multigroup version
semPower.getDf(lavModel, nGroups = 3)
semPower.getDf(lavModel, nGroups = 3, group.equal = c('loadings'))
semPower.getDf(lavModel, nGroups = 3, group.equal = c('loadings', 'intercepts'))

## End(Not run)
```

---

semPower.postHoc *semPower.postHoc*

Description

Performs a post-hoc power analysis, i. e., determines power (= 1 - beta) given alpha, df, and and a measure of effect.

Usage

```
semPower.postHoc(
  effect = NULL,
  effect.measure = NULL,
  alpha,
  N,
  df = NULL,
  p = NULL,
  SigmaHat = NULL,
```

```
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  fittingFunction = "ML",
  simulatedPower = FALSE,
  modelH0 = NULL,
  modelH1 = NULL,
  simOptions = NULL,
  lavOptions = NULL,
  lavOptionsH1 = lavOptions,
  ...
)
```

Arguments

effect	effect size specifying the discrepancy between the null hypothesis (H0) and the alternative hypothesis (H1). A list for multiple group models; a vector of length 2 for effect-size differences. Can be NULL if Sigma and SigmaHat are set.
effect.measure	type of effect, one of "F0", "RMSEA", "Mc", "GFI", "AGFI". Can be NULL if Sigma and SigmaHat are set.
alpha	alpha error
N	the number of observations (a list for multiple group models)
df	the model degrees of freedom. See semPower.getDf() for a way to obtain the df of a specific model.
p	the number of observed variables, only required for effect.measure = "GFI" and effect.measure = "AGFI".
SigmaHat	can be used instead of effect and effect.measure: model implied covariance matrix (a list for multiple group models). Used in conjunction with Sigma to define the effect.
Sigma	can be used instead of effect and effect.measure: population covariance matrix (a list for multiple group models). Used in conjunction with SigmaHat to define effect.
muHat	can be used instead of effect and effect.measure: model implied mean vector. Used in conjunction with mu. If NULL, no meanstructure is involved.
mu	can be used instead of effect and effect.measure: observed (or population) mean vector. Use in conjunction with muHat. If NULL, no meanstructure is involved.
fittingFunction	one of 'ML' (default), 'WLS', 'DWLS', 'ULS'. Defines the discrepancy function used to obtain Fmin.
simulatedPower	whether to perform a simulated (TRUE, rather than analytical, FALSE) power analysis. Only available if Sigma and modelH0 are defined.
modelH0	for simulated power: lavaan model string defining the (incorrect) analysis model.
modelH1	for simulated power: lavaan model string defining the comparison model. If omitted, the saturated model is the comparison model.

simOptions	a list of additional options specifying simulation details, see simulate() for details.
lavOptions	a list of additional options passed to lavaan, e. g., list(estimator = 'mlm') to request robust ML estimation.
lavOptionsH1	alternative options passed to lavaan that are only used for the H1 model. If NULL, identical to lavOptions. Probably only useful for multigroup models.
...	other parameters related to plots, notably plotShow, plotShowLabels, and plotLinewidth.

Value

Returns a list. Use summary() to obtain formatted results.

See Also

semPower.aPriori() semPower.compromise()

Examples

```
## Not run:
# achieved power with a sample of N = 250 to detect misspecifications corresponding
# to RMSEA >= .05 on 200 df on alpha = .05.
ph <- semPower.postHoc(effect = .05, effect.measure = "RMSEA",
                       alpha = .05, N = 250, df = 200)
summary(ph)

# power analysis for to detect the difference between a model (with df = 200) exhibiting RMSEA = .05
# and a model (with df = 210) exhibiting RMSEA = .06.
ph <- semPower.postHoc(effect = c(.05, .06), effect.measure = "RMSEA",
                       alpha = .05, N = 500, df = c(200, 210))
summary(ph)

# multigroup example
ph <- semPower.postHoc(effect = list(.02, .01), effect.measure = "F0",
                        alpha = .05, N = list(250, 350), df = 200)
summary(ph)

# power analysis based on SigmaHat and Sigma (nonsense example)
ph <- semPower.postHoc(alpha = .05, N = 1000, df = 5,
                       SigmaHat = diag(4),
                       Sigma = cov(matrix(rnorm(4*1000),  ncol=4)))
summary(ph)

# simulated power analysis (nonsense example)
ph <- semPower.aPriori(alpha = .05, N = 500, df = 200,
                       SigmaHat = list(diag(4), diag(4)),
                       Sigma = list(cov(matrix(rnorm(4*1000), ncol=4)),
                                    cov(matrix(rnorm(4*1000), ncol=4))),
                       simulatedPower = TRUE, nReplications = 100)
summary(ph)

## End(Not run)
```

semPower.powerARMA *semPower.powerARMA*

Description

Convenience function for performing power analysis on effects in an ARMA model. This requires the lavaan package.

Usage

```
semPower.powerARMA(
  type,
  comparison = "restricted",
  nWaves = NULL,
  autoregEffects = NULL,
  autoregLag1 = autoregEffects,
  autoregLag2 = NULL,
  autoregLag3 = NULL,
  mvAvgLag1 = NULL,
  mvAvgLag2 = NULL,
  mvAvgLag3 = NULL,
  means = NULL,
  variances = NULL,
  waveEqual = NULL,
  groupEqual = NULL,
  nullEffect = NULL,
  nullWhich = NULL,
  nullWhichGroups = NULL,
  invariance = TRUE,
  autocorResiduals = TRUE,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
nWaves	number of waves, must be >= 2.
autoregEffects	vector of the lag-1 autoregressive effects, e.g. c(.7, .6) for autoregressive effects of .7 for X1 -> X2 and .6 for X2 -> X3. Must be a list for multiple groups models.

autoregLag1 alternative name for autoregEffects.

autoregLag2 vector of lag-2 effects, e.g. `c(.2, .1)` for lag-2 effects of .2 for `X1 -> X3` and .1 for `X2 -> X4`.

autoregLag3 vector of lag-3 effects, e.g. `c(.2)` for a lag-3 effect of .2 for `X1 -> X4`.

mvAvgLag1 vector of the lag-1 moving average parameters, e.g. `c(.4, .3)` for moving average parameters of .4 for `N1 -> X2` and .3 for `N2 -> X3`. Must be a list for multiple groups models.

mvAvgLag2 vector of the lag-2 moving average parameters, e.g. `c(.3, .2)` for moving average parameters effects of .2 for `N1 -> X3` and .2 for `N2 -> X4`. Must be a list for multiple groups models.

mvAvgLag3 vector of the lag-3 moving average parameters, e.g. `c(.2)` for a moving average parameter of .2 for `N1 -> X4`. Must be a list for multiple groups models.

means vector of means of `X`. May be `NULL` for no meanstructure.

variances vector of variances of the noise factors `N` (= residual variances of `X`).

waveEqual parameters that are assumed to be equal across waves in both the H0 and the H1 model. Because ARMA models are likely not identified when no such constraints are imposed, this may not be empty. Valid are `'autoreg'`, `'autoregLag2'`, and `'autoregLag3'` for autoregressive effects, `'mvAvg'`, `'mvAvgLag2'`, and `'mvAvgLag3'` for moving average effects, `var` for the variance of the noise factors (starting at wave 2), `mean` for the conditional means of X (starting at wave 2).

groupEqual parameters that are restricted across groups in both the H0 and the H1 model, when `nullEffect` implies a multiple group model. Valid are `autoreg`, `mvAvg`, `var`, `mean`.

nullEffect defines the hypothesis of interest. Valid are the same arguments as in `waveEqual` and additionally `'autoreg = 0'`, `'autoregLag2 = 0'`, `'autoregLag3 = 0'`, `'mvAvg = 0'`, `'mvAvgLag2 = 0'`, `'mvAvgLag3 = 0'`, to constrain the autoregressive or moving average effects to zero, and `'autoregA = autoregB'`, `'mvAvgA = mvAvgB'`, `'varA = varB'`, `'meanA = meanB'` to constrain the autoregressive (lag-1) effects, moving average (lag-1) effects, variances of the noise factors, or means of the X to be equal across groups.

nullWhich used in conjunction with `nullEffect` to identify which parameter to constrain when there are multiple waves and parameters are not constant across waves. For example, `nullEffect = 'autoreg = 0'` with `nullWhich = 2` would constrain the second autoregressive effect for X to zero.

nullWhichGroups for hypothesis involving cross-groups comparisons, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

invariance whether metric invariance over waves is assumed (`TRUE`, the default) or not (`FALSE`). When means are part of the model, invariant intercepts are also assumed. This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

`autocorResiduals`
: whether the residuals of the indicators of latent variables are autocorrelated over waves (`TRUE`, the default) or not (`FALSE`). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

`...`
: mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. The order of factors is (X1, X2, ..., X_nWaves). See details.

Details

This function performs a power analysis to reject various hypotheses arising in models with autoregressive and moving average parameters (ARMA models), where one variable `X` is repeatedly assessed at different time points (`nWaves`), and autoregressive (lag-1 effects; `X1 -> X2 -> X3`, and optionally lag-2 and lag-3) effects, and moving average parameters (`N1 -> X2`, or equivalently for lag-2 and lag-3 effects) are assumed.

Relevant hypotheses in arising in an ARMA model are:

- `autoreg`: Tests the hypothesis that the autoregressive lag-1 effects are equal across waves (stationarity of autoregressive lag-1 effects).
- `autoregLag2`: Tests the hypothesis that the autoregressive lag-2 effects are equal across waves (stationarity of autoregressive lag-2 effects).
- `autoregLag3`: Tests the hypothesis that the autoregressive lag-3 effects are equal across waves (stationarity of autoregressive lag-3 effects).
- `mvAvg`: Tests the hypothesis that the moving average lag-1 parameters are equal across waves (stationarity of moving average lag-1 effects).
- `mvAvgLag2`: Tests the hypothesis that the moving average lag-2 parameters are equal across waves (stationarity of moving average lag-2 effects).
- `mvAvgLag3`: Tests the hypothesis that the moving average lag-3 parameters are equal across waves (stationarity of moving average lag-3 effects).
- `var`: Tests the hypothesis that the variances of the noise factors N (= the residual variances of X) are equal across waves 2 to nWaves (stationarity of variance).
- `mean`: Tests the hypothesis that the conditional means of X are equal across waves 2 to nWaves (stationarity of means).
- `autoreg = 0`, `autoregLag2 = 0`, `autoregLag3 = 0`: Tests the hypothesis that the autoregressive effects of the specified lag is zero.
- `mvAvg = 0`, `mvAvgLag2 = 0`, `mvAvgLag3 = 0`: Tests the hypothesis that the moving average parameter of the specified lag is zero.
- `autoregA = autoregB`: Tests the hypothesis that the autoregressive lag-1 effect is equal across groups.
- `mvAvgA = mvAvgB`: Tests the hypothesis that the moving average lag-1 parameter is equal across groups.
- `varA = varB`: Tests the hypothesis that the variance of the noise factors are equal across groups.
- `meanA = meanB`: Tests the hypothesis that latent means are equal across groups.

For hypotheses regarding a simple autoregression, see `semPower.powerAutoreg()`. For hypotheses regarding a CLPM structure, see `semPower.powerCLPM()`. For hypotheses regarding longitudinal measurement invariance, see `semPower.powerLI()`.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.
- `loadM`: Can be used instead of `Lambda`: Used in conjunction with `nIndicator`. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. `loadM = c(.5, .6)` defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either `Lambda`, or `loadings`, or `nIndicator` and `loadM` need to be defined. Neither may contain the noise factors. In a multigroup structure, only provide the loadings for a single group. If the model contains observed variables only, use `Lambda = diag(x)` where `x` is the number of variables.

The order of the factors is (X1, X2, ..., X_nWaves).

Additional arguments related to the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- `nReplications`: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- `minConvergenceRate`: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- `type`: specifies whether the data should be generated from a population assuming multivariate normality (`'normal'`; the default), or based on an approach generating non-normal data (`'IG'`, `'mnonr'`, `'RC'`, or `'VM'`). The approaches generating non-normal data require additional arguments detailed below.

- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# Determine required N in a 10-wave ARMA model
# to detect that the autoregressive effects differ across waves
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave), and
# the autoregressive effects vary between .5 and .7, and
# the moving average parameters are .3 at each wave and
# are assumed to be constant across waves (in both the H0 and the H1 model) and
# there are no lagged effects, and
# metric invariance and autocorrelated residuals are assumed
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = c(.5, .7, .6, .5, .7, .6, .6, .5, .6),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)

# show summary
summary(powerARMA)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerARMA$modelH1, sample.cov = powerARMA$Sigma,
            sample.nobs = powerARMA$requiredN,
            sample.cov.rescale = FALSE)
lavaan::sem(powerARMA$modelH0, sample.cov = powerARMA$Sigma,
            sample.nobs = powerARMA$requiredN,
            sample.cov.rescale = FALSE)


# same as above, but determine power with N = 250 on alpha = .05
powerARMA <- semPower.powerARMA(
  'post-hoc', alpha = .05, N = 250,
  nWaves = 10,
  autoregLag1 = c(.5, .7, .6, .5, .7, .6, .6, .5, .6),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)
```

```
# same as above, but determine the critical chi-square with N = 250 so that alpha = beta
powerARMA <- semPower.powerARMA(
  'compromise', abratio = 1, N = 250,
  nWaves = 10,
  autoregLag1 = c(.5, .7, .6, .5, .7, .6, .6, .5, .6),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80, comparison = 'saturated',
  nWaves = 10,
  autoregLag1 = c(.5, .7, .6, .5, .7, .6, .6, .5, .6),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)


# same as above, but assume only observed variables
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = c(.5, .7, .6, .5, .7, .6, .6, .5, .6),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  Lambda = diag(1, 10),
  invariance = TRUE,
  autocorResiduals = TRUE
)

# same as above, but provide reduced loadings matrix to define that
# X is measured by 3 indicators each loading by .5, .6, .4 (at each wave)
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = c(.5, .7, .6, .5, .7, .6, .6, .5, .6),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
```

```
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  loadings = list(
    c(.5, .6, .4),  # X1
    c(.5, .6, .4),  # X2
    c(.5, .6, .4),  # X3
    c(.5, .6, .4),  # X4
    c(.5, .6, .4),  # X5
    c(.5, .6, .4),  # X6
    c(.5, .6, .4),  # X7
    c(.5, .6, .4),  # X8
    c(.5, .6, .4),  # X9
    c(.5, .6, .4)   # X10
  ),
  invariance = TRUE,
  autocorResiduals = TRUE
)

# same as above, but detect that the moving average parameters differ across waves
# with a power of 80% on alpha = 5%, where
# the moving average parameters vary between .05 and .4, and
# the autoregressive effects are .5 at each wave and
# are assumed to be constant across waves (in both the H0 and the H1 model)
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = rep(.5, 9),
  mvAvgLag1 = c(.1, .05, .2, .1, .1, .3, .4, .4, .4),
  variances = rep(1, 10),
  waveEqual = c('autoreg'),
  nullEffect = 'mvAvg',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)


# same as above, but detect that the (noise) variances differ across waves
# with a power of 80% on alpha = 5%, where
# the variances vary between 0.5 and 2, and
# the autoregressive effects are .5 at each wave and
# the moving average parameters are .3 at each wave and
# bothj are assumed to be constant across waves (in both the H0 and the H1 model)
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = rep(.5, 9),
  mvAvgLag1 = rep(.3, 9),
  variances = c(1, .5, .7, .6, .7, .9, 1.2, 1.7, 2.0, 1.5),
  waveEqual = c('autoreg', 'mvAvg'),
  nullEffect = 'var',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
```

```
  autocorResiduals = TRUE
)


# same as above, but include a meanstructure and
# detect that the means differ across waves
# with a power of 80% on alpha = 5%, where
# the means vary between 0 and .5, and
# the autoregressive effects are .5 at each wave and
# the moving average parameters are .3 at each wave and
# the variances are 1 at each wave and
# all are assumed to be constant across waves (in both the H0 and the H1 model) and
# metric and scalar invariance is assumed
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = rep(.5, 9),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  means = c(0, .1, .2, .3, .4, .5, .3, .4, .5, .5),
  waveEqual = c('autoreg', 'mvAvg', 'var'),
  nullEffect = 'mean',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)

# Determine required N in a 10-wave ARMA model
# to detect that the autoregressive lag-2 effects differ from zero
# with a power of 80% on alpha = 5%, where
# the lag-2 autoregressive effects are .2 at each wave and
# the lag-2 autoregressive effects are .1 at each wave and
# the autoregressive effects are .5 at each wave and
# the moving average parameters are .3 at each wave and
# the noise variances are equal to 1 in each wave,
# and all are assumed to be constant across waves (in both the H0 and the H1 model) and
# metric invariance and autocorrelated residuals are assumed, and
# the autoregressive lag2- and lag3-effects are estimated
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = rep(.5, 9),
  autoregLag2 = rep(.2, 8),
  autoregLag3 = rep(.1, 7),
  mvAvgLag1 = rep(.3, 9),
  variances = rep(1, 10),
  waveEqual = c('mvAvg', 'autoreg', 'var', 'autoreglag2', 'autoreglag3'),
  nullEffect = 'autoreglag2 = 0',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)
```

```
# similar as above, but get required N to detect that
# lag-2 moving average parameters are constant across waves
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80,
  nWaves = 10,
  autoregLag1 = rep(.5, 9),
  autoregLag2 = rep(.2, 8),
  mvAvgLag1 = rep(.3, 9),
  mvAvgLag2 = c(.1, .2, .3, .1, .2, .3, .1, .1),
  variances = rep(1, 10),
  waveEqual = c('mvAvg', 'autoreg', 'var', 'autoreglag2'),
  nullEffect = 'mvAvgLag2',
  nIndicator = rep(3, 10), loadM = .5,
  invariance = TRUE
)


# Determine required N in a 5-wave ARMA model
# to detect that the autoregressive effects in group 1
# differ from the ones in group 2, where
# both groups are equal-sized
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave and in each group), and
# the autoregressive effects in group 1 are .5 (constant across waves) and
# the autoregressive effects in group 2 are .6 (constant across waves) and
# the moving average parameters are .25 at each wave and in both groups and
# the variances are 1 at each wave and in both groups and
# all are assumed to be constant across waves (in both the H0 and the H1 model)
# metric invariance (across both waves and groups) and
# autocorrelated residuals are assumed
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80, N = list(1, 1),
  nWaves = 5,
  autoregLag1 = list(
    c(.5, .5, .5, .5),   # group 1
    c(.6, .6, .6, .6)),  # group 2
  mvAvgLag1 = rep(.25, 4),
  variances = rep(1, 5),
  waveEqual = c('autoreg', 'var', 'mvavg'),
  nullEffect = 'autoregA = autoregB',
  nIndicator = rep(3, 5), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)


# Determine required N in a 5-wave ARMA model
# to detect that the means in group 1
# differ from the means in group 2, where
# both groups are equal-sized
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave and in each group), and
# the autoregressive effects are .5 at each wave and in both groups and
```

```
# the moving average parameters are .25 at each wave and in both groups and
# the variances are 1 at each wave and in both groups and
# all are assumed to be constant across waves (in both the H0 and the H1 model) and
# invariance of variances, autoregressive effects, and moving average parameters
# across groups as well as
# metric and scalar invariance (across both waves and groups) and
# autocorrelated residuals are assumed
powerARMA <- semPower.powerARMA(
  'a-priori', alpha = .05, power = .80, N = list(1, 1),
  nWaves = 5,
  autoregLag1 = list(
    c(.5, .5, .5, .5),   # group 1
    c(.5, .5, .5, .5)),  # group 2
  mvAvgLag1 = rep(.25, 4),
  variances = rep(1, 5),
  means = list(
    c(0, .1, .1, .1, .1),  # group 1
    c(0, .4, .4, .4, .4)   # group 2
  ),
  waveEqual = c('autoreg', 'var', 'mvavg', 'mean'),
  groupEqual = c('var', 'autoreg', 'mvavg'),
  nullEffect = 'meanA = meanB',
  nIndicator = rep(3, 5), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE
)

# perform a simulated post-hoc power analysis
# with 250 replications
set.seed(300121)
powerARMA <- semPower.powerARMA(
  'post-hoc', alpha = .05, N = 500,
  nWaves = 5,
  autoregLag1 = c(.3, .7, .6, .3),
  mvAvgLag1 = rep(.3, 4),
  variances = rep(1, 5),
  waveEqual = c('mvAvg'),
  nullEffect = 'autoreg',
  nIndicator = rep(3, 5), loadM = .5,
  invariance = TRUE,
  autocorResiduals = TRUE,
  simulatedPower = TRUE,
  simOptions = list(nReplications = 250)
)

## End(Not run)
```

`semPower.powerAutoreg` *semPower.powerAutoreg*

Description

Convenience function for performing power analysis on effects in an autoregressive model. This requires the lavaan package.

Usage

```
semPower.powerAutoreg(
  type,
  comparison = "restricted",
  nWaves = NULL,
  autoregEffects = NULL,
  lag1Effects = autoregEffects,
  lag2Effects = NULL,
  lag3Effects = NULL,
  means = NULL,
  variances = NULL,
  waveEqual = NULL,
  nullEffect = NULL,
  nullWhich = NULL,
  nullWhichGroups = NULL,
  standardized = TRUE,
  invariance = TRUE,
  autocorResiduals = TRUE,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
nWaves	number of waves, must be >= 2.
autoregEffects	vector of the autoregressive effects, e.g. c(.7, .6) for autoregressive effects of .7 for X1 -> X2 and .6 for X2 -> X3. Must be a list for multiple groups models.
lag1Effects	alternative name for autoregEffects.
lag2Effects	vector of lag-2 effects, e.g. c(.2, .1) for lag-2 effects of .2 for X1 -> X3 and .1 for X2 -> X4.
lag3Effects	vector of lag-3 effects, e.g. c(.2) for a lag-3 effect of .2 for X1 -> X4.
means	vector of means for X. Can be omitted for no meanstructure.
variances	vector of (residual-)variances for X. When omitted and standardized = FALSE, all (residual-)variances are equal to 1. When omitted and standardized = TRUE, the (residual-)variances are determined so that all variances are 1, and will thus typically differ from each other. When provided, standardized must be FALSE.

waveEqual — parameters that are assumed to be equal across waves in both the H0 and the H1 model. Valid are 'lag1' (or equivalently 'autoreg'), 'lag2', and 'lag3', or NULL for none (so that all parameters are freely estimated, subject to the constraints defined in nullEffect).

nullEffect — defines the hypothesis of interest. Valid are the same arguments as in waveEqual and additionally 'lag1 = 0' (or equivalently 'autoregX = 0') 'lag2 = 0', 'lag3 = 0' to constrain the autoregressive, lag-2, or lag-3 effects to zero, and 'autoregA = autoregB' to the autoregressive effects be equal across groups.

nullWhich — used in conjunction with nullEffect to identify which parameter to constrain when there are > 2 waves and parameters are not constant across waves. For example, nullEffect = 'lag1 = 0' with nullWhich = 2 would constrain the second autoregressive effect for X to zero.

nullWhichGroups — for hypothesis involving cross-groups comparisons, vector indicating the groups for which equality constrains should be applied, e.g. c(1, 3) to constrain the relevant parameters of the first and the third group. If NULL, all groups are constrained to equality.

standardized — whether all parameters should be standardized (TRUE, the default). If FALSE, all regression relations are unstandardized.

invariance — whether metric invariance over waves is assumed (TRUE, the default) or not (FALSE). When means are part of the model, invariant intercepts are also assumed. This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

autocorResiduals — whether the residuals of the indicators of latent variables are autocorrelated over waves (TRUE, the default) or not (FALSE). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

... — mandatory further parameters related to the specific type of power analysis requested, see semPower.aPriori(), semPower.postHoc(), and semPower.compromise(), and parameters specifying the factor model. The order of factors is (X1, X2, ..., X_nWaves). See details.

Details

This function performs a power analysis to reject various hypotheses arising in simple autoregressive (simplex) models, where one variable is repeatedly assessed at two or more different time points (nWaves), yielding autoregressive effects (aka lag-1 effects or stabilities, ; X1 -> X2 -> X3), and optionally lagged effects (X1 -> X3), variances, and means.

Relevant hypotheses in arising in an autogressive model are:

- autoreg or lag1: Tests the hypothesis that the autoregressive (lag-1) effects are equal across waves (stationarity of autoregressive parameters).
- lag2: Tests the hypothesis that the lag-2 effects are equal across waves (stationarity of lag-2 effects).
- lag3: Tests the hypothesis that the lag-3 effects are equal across waves (stationarity of lag-3 effects).

- `var`: Tests the hypothesis that the residual-variances of X (i.e., X_2, ..., X_nWaves) are equal across waves (stationarity of variance).
- `mean`: Tests the hypothesis that the conditional means of X (i.e., X_2, ..., X_nWaves) are equal across waves (stationarity of means).
- `autoreg = 0` or `lag1 = 0`: Tests the hypothesis that the autoregressive (lag-1) effect of X is zero.
- `lag2 = 0` and `lag3 = 0`: Tests the hypothesis that a lag-2 or a lag-3 effect is zero.
- `autoregA = autoregB` or `lag1A = lag1B`: : Tests the hypothesis that the autoregressive effect of X is equal across groups.

For hypotheses in an ARMA model, see `semPower.powerARMA()`. For hypotheses regarding a CLPM structure, see `semPower.powerCLPM()`. For hypotheses regarding longitudinal measurement invariance, see `semPower.powerLI()`.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.
- `loadM`: Can be used instead of `Lambda`: Used in conjunction with `nIndicator`. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. `loadM = c(.5, .6)` defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either `Lambda`, or `loadings`, or `nIndicator` and `loadM` need to be defined. If the model contains observed variables only, use `Lambda = diag(x)` where `x` is the number of variables.

Note that the order of the factors is (X1, X2, ..., X_nWaves).

Additional arguments related to the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.

SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# Determine required N in a 4-wave autoregressive model
# to detect an autoregressive effect between X1 -> X2 of >= .5
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave), and
# the autoregressive effecst are .5 (X1 -> X2), .7 (X2 -> X3), and .6 (X3 -> X4), and
# there are no lagged effects, and
# metric invariance and autocorrelated residuals are assumed
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.5, .7, .6),
  nullEffect = 'autoreg=0',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)

# show summary
summary(powerAutoreg)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerAutoreg$modelH1, sample.cov = powerAutoreg$Sigma,
            sample.nobs = powerAutoreg$requiredN,
            sample.cov.rescale = FALSE)
lavaan::sem(powerAutoreg$modelH0, sample.cov = powerAutoreg$Sigma,
            sample.nobs = powerAutoreg$requiredN,
            sample.cov.rescale = FALSE)


# same as above, but determine power with N = 250 on alpha = .05
powerAutoreg <- semPower.powerAutoreg(
  'post-hoc', alpha = .05, N = 250,
  nWaves = 4,
  autoregEffects = c(.5, .7, .6),
  nullEffect = 'autoreg=0',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
```

```
  invariance = TRUE, autocorResiduals = TRUE)

# same as above, but determine the critical chi-square with N = 250 so that alpha = beta
powerAutoreg <- semPower.powerAutoreg(
  'compromise', abratio = 1, N = 250,
  nWaves = 4,
  autoregEffects = c(.5, .7, .6),
  nullEffect = 'autoreg=0',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerAutoreg <- semPower.powerAutoreg(
  'post-hoc', alpha = .05, N = 250,
  comparison = 'saturated',
  nWaves = 4,
  autoregEffects = c(.5, .7, .6),
  nullEffect = 'autoreg=0',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)

# same as above, but assume only observed variables
powerAutoreg <- semPower.powerAutoreg(
  'post-hoc', alpha = .05, N = 250,
  nWaves = 4,
  autoregEffects = c(.5, .7, .6),
  nullEffect = 'autoreg=0',
  nullWhich = 1,
  Lambda = diag(4))

# same as above, but provide reduced loadings matrix to define that
# X is measured by 3 indicators each loading by .8, .6, .7 (at each wave)
powerAutoreg <- semPower.powerAutoreg(
  'post-hoc', alpha = .05, N = 250,
  nWaves = 4,
  autoregEffects = c(.5, .7, .6),
  nullEffect = 'autoreg=0',
  nullWhich = 1,
  loadings = list(
    c(.8, .6, .7),   # X1
    c(.8, .6, .7),   # X2
    c(.8, .6, .7),   # X3
    c(.8, .6, .7)    # X4
  ),
  invariance = TRUE, autocorResiduals = TRUE)

# same as above, but assume wave-constant autoregressive effects
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
```

```
  autoregEffects = c(.6, .6, .6),
  waveEqual = c('autoreg'),
  nullEffect = 'autoreg=0',
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)


# same as above, but detect that autoregressive effects are not wave-constant
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .7, .8),
  nullEffect = 'autoreg',
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)

# same as above, but include lag-2 and lag-3 effects
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .6, .6),
  lag2Effects = c(.25, .20),
  lag3Effects = c(.15),
  waveEqual = c('autoreg'),
  nullEffect = 'autoreg=0',
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)


# same as above, but detect that first lag-2 effect differs from zero
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .6, .6),
  lag2Effects = c(.25, .20),
  lag3Effects = c(.15),
  waveEqual = c('autoreg'),
  nullEffect = 'lag2=0',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)


# same as above, but assume wave-constant lag2 effects
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .6, .6),
  lag2Effects = c(.25, .25),
  lag3Effects = c(.15),
  waveEqual = c('autoreg', 'lag2'),
  nullEffect = 'lag2=0',
  nIndicator = rep(3, 4), loadM = .5,
```

```
  invariance = TRUE, autocorResiduals = TRUE)


# same as above, but detect that lag3 effect differs from zero
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .6, .6),
  lag2Effects = c(.25, .25),
  lag3Effects = c(.15),
  waveEqual = c('autoreg', 'lag2'),
  nullEffect = 'lag3=0',
  nIndicator = rep(3, 4), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)


# Determine required N in a 3-wave autoregressive model
# assuming wave-constant autoregressive effects
# that the autoregressive effects in group 1
# differ from those in group 2
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave and in each group), and
# the autoregressive effect is .7 in group 1 and
# the autoregressive effect is .5 in group 2 and
# there are no lagged effects, and
# metric invariance over both time and groups and autocorrelated residuals are assumed and
# the groups are equal-sized
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80, N = list(1, 1),
  nWaves = 3,
  autoregEffects = list(
    c(.7, .7),
    c(.5, .5)
  ),
  waveEqual = c('autoreg'),
  nullEffect = 'autoregA = autoregB',
  nullWhich = 1,
  nIndicator = rep(3, 3), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE)

# Determine required N in a 4-wave autoregressive model
# to detect that the factor residual-variances (X2, X3, X4) differ
# with a power of 80% on alpha = 5%, where
# the (residual-)variances are 1, .5, 1.5, and 1, respectively,
# X is measured by 3 indicators loading by .5 each (at each wave), and
# the autoregressive effects are .6, and
# both the H0 and the H1 assume wave-constant autoregressive effects, and
# there are no lagged effects, and
# metric invariance and autocorrelated residuals are assumed
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .6, .6),
```

```
  variances = c(1, .5, 1.5, 1),
  waveEqual = c('autoreg'),
  nullEffect = 'var',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
  standardized = FALSE,
  invariance = TRUE,
  autocorResiduals = TRUE)

# same as above, but
# include latent means and
# detect that latent means differ and
# assume wave-constant variances and autoregressive parameters for both H0 and H1
powerAutoreg <- semPower.powerAutoreg(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  autoregEffects = c(.6, .6, .6),
  variances = c(1, 1, 1, 1),
  means = c(0, .5, 1, .7),
  waveEqual = c('autoreg', 'var'),
  nullEffect = 'mean',
  nullWhich = 1,
  nIndicator = rep(3, 4), loadM = .5,
  standardized = FALSE,
  invariance = TRUE,
  autocorResiduals = TRUE)

# request a simulated post-hoc power analysis with 500 replications
set.seed(300121)
powerAutoreg <- semPower.powerAutoreg(
  'post-hoc', alpha = .05, N = 500,
  nWaves = 3,
  autoregEffects = c(.7, .7),
  waveEqual = c('autoreg'),
  nullEffect = 'autoreg = 0',
  nullWhich = 1,
  nIndicator = rep(3, 3), loadM = .5,
  invariance = TRUE, autocorResiduals = TRUE,
  simulatedPower = TRUE,
  simOptions = list(nReplications = 500)
  )


## End(Not run)
```

`semPower.powerBifactor`

semPower.powerBifactor

Description

Perform a power analysis for models including one or more bifactors to reject one of the following hypotheses: (a) a zero correlation between two factors, (b) the equality of two correlations between factors, or (c) the equality of a correlation between two factors across two or more groups. This requires the lavaan package.

Usage

```
semPower.powerBifactor(
  type,
  comparison = "restricted",
  bfLoadings = NULL,
  bfWhichFactors = NULL,
  Phi = NULL,
  nullEffect = "cor = 0",
  nullWhich = NULL,
  nullWhichGroups = NULL,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
bfLoadings	a single vector or a list containing one or more vectors giving the loadings on each bifactor. For example, list(rep(.6, 10), rep(.6, 10)) defines two bifactors with 10 indicators each, loading by .6 each. Can be a list of lists for multiple group models.
bfWhichFactors	a list containing one or more vectors defining which (specific) factors defined in the respective arguments in ... are part of the bifactor structure. See details.
Phi	either a single number defining the correlation between exactly two factors or the factor correlation matrix. Must only contain the bifactor(s) and the covariate(s). Must be a list for multiple group models. Phi assumes the following order (bifactor_1, bifactor_2, ..., bifactor_j, covariate_1, covariate_2, ..., covariate_k). See details.
nullEffect	defines the hypothesis of interest, must be one of 'cor = 0' (the default) to test whether a correlation is zero, 'corX = corZ' to test for the equality of correlations, and 'corA = corB' to test for the equality of a correlation across groups. Define the correlations to be set to equality in nullWhich and the groups in nullWhichGroups.
nullWhich	vector of size 2 indicating which factor correlation in Phi is hypothesized to equal zero when nullEffect = 'cor = 0', or to restrict to equality across groups

when `nullEffect = 'corA = corB'`, or list of vectors defining which correlations to restrict to equality when `nullEffect = 'corX = corZ'`. Can also contain more than two correlations, e.g., `list(c(1, 2), c(1, 3), c(2, 3))` to set `Phi[1, 2] = Phi[1, 3] = Phi[2, 3]`.

`nullWhichGroups`
: for `nullEffect = 'corA = corB'`, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

`...`
: mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model concerning the specific factors and the covariate(s). See details.

Details

This function performs a power analysis to reject various hypotheses arising in a model including a bifactor structure:

- `nullEffect = 'cor = 0'`: Tests the hypothesis that the correlation between a bifactor and another factor (which can also be a bifactor) is zero.
- `nullEffect = 'corX = corZ'`: Tests the hypothesis that two or more correlations involving one or more bifactors are equal to each other.
- `nullEffect = 'corA = corB'`: Tests the hypothesis that the correlation between the bifactor and another factor (which can also be a bifactor) is equal in two or more groups (always assuming metric invariance).

A bifactor structure is defined by specifying the loadings on the general factor in `bfLoadings`, the comprised specific factors in `bfWhichFactors`, and the loadings on the specific factors in either `Lambda`, or `loadings`, or `nIndicator` and `loadM`. The latter arguments also include the loadings defining the covariate(s).

The correlations betwen the bifactor(s) and the covariate(s) are defined in `Phi`, which must omit the specific factors and only includes the bifactor(s) and the covariate(s) assuming the following order: (bifactor_1, bifactor_2, ..., bifactor_j, covariate_1, covariate_2, ..., covariate_k).

For example, the following defines a single bifactor with 10 indicators loading by .5 each. The bifactor structure involves 3 specific factors measured by 3 indicators each, each loading by .3, .2, and .1 on the respective specific factor (in addition to the loadings on the bifactor). Furthermore, two covariate with 5 indicators each, all loading by .7, are defined. The correlation between the covariates is .5, the one between the bifactor and the first and second covariate are .3 and .2, respectively.

```
bfLoadings <- rep(.5, 10)
bfWhichFactors <- c(1, 2, 3)
loadings <- list(
  rep(.3, 3),   # specific factor 1
  rep(.2, 3),   # specific factor 2
  rep(.1, 3),   # specific factor 3
  rep(.7, 5),   # covariate 1
  rep(.7, 5)    # covariate 2
```

```
)
Phi <- matrix(c(
  c(1, .3, .2),   # bifactor
  c(.3, 1, .5),   # covariate 1
  c(.2, .5, 1)   # covariate 2
), ncol = 3, byrow = TRUE)
```

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model** concerning the specific factors and the covariate(s). The loadings on the bifactor must be provided via `bfLoadings`.

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of specific factors and covariates).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.
- `loadM`: Can be used instead of `Lambda`: Used in conjunction with `nIndicator`. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. `loadM = c(.5, .6)` defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

Additional arguments related to the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- `nReplications`: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- `minConvergenceRate`: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- `type`: specifies whether the data should be generated from a population assuming multivariate normality (`'normal'`; the default), or based on an approach generating non-normal data (`'IG'`, `'mnonr'`, `'RC'`, or `'VM'`). The approaches generating non-normal data require additional arguments detailed below.

- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# get required N to detect a correlation of >= .3 between
# a single bifactor with 11 indicators all loadings by .6
# spanning the indicators of 3 specific factors
# with three indicators each, loading by .2, .15, and .25, respectively
# and a covariate measured by 4 indicators loading by .7 each,
# with a power of 95% on alpha = 5%
bfLoadings <- rep(.6, 11)
bfWhichFactors <- c(1, 2, 3)
loadings <- list(
  # specific factors
  rep(.2, 3),
  rep(.15, 3),
  rep(.25, 3),
  # covariate
  rep(.7, 4)
)
Phi <- .3    # bifactor - covariate
powerbifactor <- semPower.powerBifactor(type = 'a-priori',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
                                        alpha = .05, beta = .05)
# show summary
summary(powerbifactor)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerbifactor$modelH1, sample.cov = powerbifactor$Sigma,
            sample.nobs = powerbifactor$requiredN,
            sample.cov.rescale = FALSE)
lavaan::sem(powerbifactor$modelH0, sample.cov = powerbifactor$Sigma,
            sample.nobs = powerbifactor$requiredN,
            sample.cov.rescale = FALSE)

# same as above, but determine power with N = 500 on alpha = .05
powerbifactor <- semPower.powerBifactor(type = 'post-hoc',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
                                        alpha = .05, N = 500)

# same as above, but determine the critical chi-square with N = 500 so that alpha = beta
powerbifactor <- semPower.powerBifactor(type = 'compromise',
                                        bfLoadings = bfLoadings,
```

```
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
                                        abratio = 1, N = 500)

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerbifactor <- semPower.powerBifactor(type = 'a-priori',
                                        comparison = 'saturated',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
                                        alpha = .05, beta = .05)

# define two bifactors with 10 indicators each, where
# all loadings are .6 on the first and .5 on the second bifactor.
# the first bifactor spans the indicators of specific factors 1-3,
# the second bifactor spans the indicators of specific factors 4-6,
# all specific factors are measured by three indicators each,
# loadings are .2, .15, .25, .1, .15., and.2, respectively.
# define an additional  covariate measured by 4 indicators loading by .6 each.
# get required N to detect a correlation of >= .3 between the bifactors
# with a power of 95% on alpha = 5%
bfLoadings <- list(rep(.6, 10),
                   rep(.6, 10))
bfWhichFactors <- list(c(1, 2, 3),
                       c(4, 5, 6))
loadings <- list(
  # specific factors for bf1
  rep(.2, 3),
  rep(.15, 3),
  rep(.25, 3),
  # specific factors bf2
  rep(.1, 3),
  rep(.15, 3),
  rep(.2, 3),
  # covariate
  rep(.6, 4)
)
Phi <- diag(3)
Phi[1, 2] <- Phi[2, 1] <- .3    # bifactor1 - bifactor2
Phi[1, 3] <- Phi[3, 1] <- .5    # bifactor1 - covariate
Phi[2, 3] <- Phi[3, 2] <- .1    # bifactor2 - covariate

powerbifactor <- semPower.powerBifactor(type = 'a-priori',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
```

```
                                        alpha = .05, beta = .05)

# same as above, but get required N to detect that
# the correlation between the first bifactor and the covariate (of r=.5) differs from
# the correlation between the second bifactor and the covariate (of r=.1)
powerbifactor <- semPower.powerBifactor(type = 'a-priori',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullEffect = 'corx = corz',
                                        nullWhich = list(c(1, 3), c(2, 3)),
                                        loadings = loadings,
                                        alpha = .05, beta = .05)

# multiple group example: get required N to detect that
# the correlation of a bifactor with 10 indicators
# spanning three specific factors with 3 indicators each
# to a covariate in group 1  (of r = .3)
# differs from the one in group 2 (of r = .1)
bfLoadings <- rep(.6, 10)
bfWhichFactors <- c(1, 2, 3)
loadings <- list(
  # specific factors
  rep(.2, 3),
  rep(.15, 3),
  rep(.25, 3),
  # covariate
  rep(.7, 4)
)
Phi1 <- Phi2 <- diag(2)
Phi1[1, 2] <- Phi1[2, 1] <- .3    # bifactor - covariate
Phi2[1, 2] <- Phi2[2, 1] <- .1    # bifactor - covariate
Phi <- list(Phi1, Phi2)
powerbifactor <- semPower.powerBifactor(type = 'a-priori',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullEffect = 'corA = corB',
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
                                        alpha = .05, beta = .05,
                                        N = list(1, 1))

# request a simulated post-hoc power analysis with 500 replications.
bfLoadings <- rep(.6, 11)
bfWhichFactors <- c(1, 2, 3)
loadings <- list(
  # specific factors
  rep(.2, 3),
  rep(.15, 3),
  rep(.1, 3),
  # covariate
  rep(.7, 5)
```

```
)
Phi <- .2
set.seed(300121)
powerbifactor <- semPower.powerBifactor(type = 'post-hoc',
                                        bfLoadings = bfLoadings,
                                        bfWhichFactors = bfWhichFactors,
                                        Phi = Phi,
                                        nullWhich = c(1, 2),
                                        loadings = loadings,
                                        alpha = .05, N = 500,
                                        simulatedPower = TRUE,
                                        simOptions = list(nReplications = 500)
                                        )

## End(Not run)
```

`semPower.powerCFA` *semPower.powerCFA*

Description

Convenience function for performing power analyses for CFA models to reject one of the following hypotheses: (a) a zero correlation between two factors, (b) the equality of two correlations between factors, or (c) the equality of a correlation between two factors across two or more groups. This requires the lavaan package.

Usage

```
semPower.powerCFA(
  type,
  comparison = "restricted",
  Phi = NULL,
  nullEffect = "cor = 0",
  nullWhich = NULL,
  nullWhichGroups = NULL,
  ...
)
```

Arguments

`type`	type of power analysis, one of `'a-priori'`, `'post-hoc'`, `'compromise'`.
`comparison`	comparison model, one of `'saturated'` or `'restricted'` (the default). This determines the df for power analyses. `'saturated'` provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. `'restricted'` provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in `nullEffect`, so the df equal the number of restrictions.

`Phi` either a single number defining the correlation between exactly two factors or the factor correlation matrix. A list for multiple group models.

`nullEffect` defines the hypothesis of interest, must be one of `'cor = 0'` (the default) to test whether a correlation is zero, `'corX = corZ'` to test for the equality of correlations, `'corA = corB'` to test for the equality of a correlation across groups, and `loading = 0` to test whether a loading is zero. Define the correlations to be set to equality in `nullWhich` and the groups in `nullWhichGroups`.

`nullWhich` vector of size 2 indicating which element in Lambda should equal zero when `nullEffect = 'loading = 0'`, or which factor correlation in `Phi` is hypothesized to equal zero when `nullEffect = 'cor = 0'`, or to restrict to equality across groups when `nullEffect = 'corA = corB'`, or list of vectors defining which correlations to restrict to equality when `nullEffect = 'corX = corZ'`. Can also contain more than two correlations, e.g., `list(c(1, 2), c(1, 3), c(2, 3))` to set `Phi[1, 2] = Phi[1, 3] = Phi[2, 3]`. If omitted, the correlation between the first and the second factor is targeted, i. e., `nullWhich = c(1, 2)`.

`nullWhichGroups` for `nullEffect = 'corA = corB'`, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

`...` mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. See details.

Details

This function performs a power analysis to reject various hypotheses arising in standard CFA models:

- `nullEffect = 'loading = 0'`: Tests the hypothesis that a loading is zero.
- `nullEffect = 'cor = 0'`: Tests the hypothesis that the correlation between two factors is zero.
- `nullEffect = 'corX = corZ'`: Tests the hypothesis that two or more correlations between three or more factors are equal to each other.
- `nullEffect = 'corA = corB'`: Tests the hypothesis that the correlation between two factors is equal in two or more groups (always assuming metric invariance).

For hypotheses regarding regression relationships between factors, see `semPower.powerRegression()`. For hypotheses regarding mediation effects, see `semPower.powerMediation()`. For hypotheses regarding measurement invariance, see `semPower.powerMI()`.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).

- loadings: Can be used instead of Lambda: Defines the primary loadings for each factor in a list structure, e. g. loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4)) defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- nIndicator: Can be used instead of Lambda: Used in conjunction with loadM. Defines the number of indicators by factor, e. g., nIndicator = c(3, 4) defines a two factor model with three and four indicators for the first and second factor, respectively. nIndicator can also be a single number to define the same number of indicators for each factor.
- loadM: Can be used instead of Lambda: Used in conjunction with nIndicator. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. loadM = c(.5, .6) defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either Lambda, or loadings, or nIndicator and loadM need to be defined. If the model contains observed variables only, use Lambda = diag(x) where x is the number of variables.

Additional arguments related to the requested type of **power analysis**:

- alpha: The alpha error probability. Required for type = 'a-priori' and type = 'post-hoc'.
- Either beta or power: The beta error probability and the statistical power (1 - beta), respectively. Only for type = 'a-priori'.
- N: The sample size. Always required for type = 'post-hoc' and type = 'compromise'. For type = 'a-priori' and multiple group analysis, N is a list of group weights.
- abratio: The ratio of alpha to beta. Only for type = 'compromise'.

If a **simulated power analysis** (simulatedPower = TRUE) is requested, optional arguments can be provided as a list to simOptions:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM').
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR. The approaches generating non-normal data require additional arguments detailed below.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# get required N to detect a correlation of >= .2 between two factors
# with a power of 95% on alpha = 5%, where the factors are
# measured by 5 and 6 indicators, respectively, and all loadings are equal to .5
powercfa <- semPower.powerCFA(type = 'a-priori',
                              Phi = .2,
                              nIndicator = c(5, 6), loadM = .5,
```

```
                              alpha = .05, beta = .05)
# show summary
summary(powercfa)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powercfa$modelH1, sample.cov = powercfa$Sigma,
sample.nobs = powercfa$requiredN, sample.cov.rescale = FALSE)
lavaan::sem(powercfa$modelH0, sample.cov = powercfa$Sigma,
sample.nobs = powercfa$requiredN, sample.cov.rescale = FALSE)

# same as above, but determine power with N = 500 on alpha = .05
powercfa <- semPower.powerCFA(type = 'post-hoc',
                              Phi = .2,
                              nIndicator = c(5, 6), loadM = .5,
                              alpha = .05, N = 500)

# same as above, but determine the critical chi-square with N = 500 so that alpha = beta
powercfa <- semPower.powerCFA(type = 'compromise',
                              Phi = .2,
                              nIndicator = c(5, 6), loadM = .5,
                              abratio = 1, N = 500)

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powercfa <- semPower.powerCFA(type = 'a-priori',
                              comparison = 'saturated',
                              Phi = .2,
                              nIndicator = c(5, 6), loadM = .5,
                              alpha = .05, beta = .05)

# same as above, but provide a reduced loading matrix defining
# three indicators with loadings of .7, .6, and .5 on the first factor and
# four indicators with loadings of .5, .6, .4, .8 on the second factor
powercfa <- semPower.powerCFA(type = 'a-priori',
                              Phi = .2,
                              loadings = list(c(.7, .6, .5),
                                              c(.5, .6, .4, .8)),
                              alpha = .05, beta = .05)

# detect that the loading of indicator 4 on the first factor differs from zero
Lambda <- matrix(c(
  c(.8, 0),
  c(.4, 0),
  c(.6, 0),
  c(.1, .5),
  c(0, .6),
  c(0, .7)
), ncol = 2, byrow = TRUE)
powercfa <- semPower.powerCFA(type = 'a-priori',
                              Phi = .2,
                              nullEffect = 'loading = 0',
                              nullWhich = c(4, 1),
                              Lambda = Lambda,
                              alpha = .05, beta = .05)
```

```
# get required N to detect a correlation of >= .3 between factors 1 and 3
# in a three factor model. Factors are measured by 3 indicators each, and all loadings
# on the first, second, and third factor are .5, .6, and .7, respectively.
Phi <- matrix(c(
  c(1.00, 0.20, 0.30),
  c(0.20, 1.00, 0.10),
  c(0.30, 0.10, 1.00)
), ncol = 3,byrow = TRUE)

powercfa <- semPower.powerCFA(type = 'a-priori',
                              Phi = Phi,
                              nullWhich = c(1, 3),
                              nIndicator = c(3, 3, 3), loadM = c(.5, .6, .7),
                              alpha = .05, beta = .05)

# same as above, but ask for N to detect that
# the correlation between factors 1 and 2 (of r = .2) differs from
# the correlation between factors 2 and 3 (of r = .3).
powercfa <- semPower.powerCFA(type = 'a-priori',
                              Phi = Phi,
                              nullEffect = 'corX = corZ',
                              nullWhich = list(c(1, 2), c(1, 3)),
                              nIndicator = c(3, 3, 3), loadM = c(.5, .6, .7),
                              alpha = .05, beta = .05)

# same as above, but ask for N to detect that all three correlations are unequal
powercfa <- semPower.powerCFA(type = 'a-priori',
                              Phi = Phi,
                              nullEffect = 'corX = corZ',
                              nullWhich = list(c(1, 2), c(1, 3), c(2, 3)),
                              nIndicator = c(3, 3, 3), loadM = c(.5, .6, .7),
                              alpha = .05, beta = .05)

# get required N to detect that the correlation between two factors
# in group 1 (of r = .2) differs from the one in group 2 (of r = .4).
# The measurement model is identical for both groups:
# The first factor is measured by 3 indicators loading by .7 each,
# the second factor is measured by 6 indicators loading by .5 each.
# Both groups are sized equally (N = list(1, 1)).
powercfa <- semPower.powerCFA(type = 'a-priori',
                              nullEffect = 'corA = corB',
                              Phi = list(.2, .4),
                              loadM = c(.7, .5),
                              nIndicator = c(3, 6),
                              alpha = .05, beta = .05, N = list(1, 1))

# request a simulated post-hoc power analysis with 500 replications.
set.seed(300121)
powercfa <- semPower.powerCFA(type = 'post-hoc',
                              Phi = .2,
                              nIndicator = c(5, 6), loadM = .5,
```

```
                              alpha = .05, N = 500,
                              simulatedPower = TRUE,
                              simOptions = list(nReplications = 500))


## End(Not run)
```

`semPower.powerCLPM` *semPower.powerCLPM*

Description

Convenience function for performing power analysis on effects in a cross-lagged panel model (CLPM). This requires the lavaan package.

Usage

```
semPower.powerCLPM(
  type,
  comparison = "restricted",
  nWaves = NULL,
  autoregEffects = NULL,
  crossedEffects = NULL,
  rXY = NULL,
  waveEqual = NULL,
  nullEffect = NULL,
  nullWhich = NULL,
  nullWhichGroups = NULL,
  standardized = TRUE,
  standardizedResidualCovariances = TRUE,
  metricInvariance = TRUE,
  autocorResiduals = TRUE,
  ...
)
```

Arguments

`type`	type of power analysis, one of `'a-priori'`, `'post-hoc'`, `'compromise'`.
`comparison`	comparison model, one of `'saturated'` or `'restricted'` (the default). This determines the df for power analyses. `'saturated'` provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. `'restricted'` provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in `nullEffect`, so the df equal the number of restrictions.
`nWaves`	number of waves, must be >= 2.

autoregEffects vector of the autoregressive effects of X and Y (constant across waves), or a list of vectors of autoregressive effects for X and Y from wave to wave, e.g. `list(c(.7, .6), c(.5, .5))` for a autoregressive effect of .7 for `X1 -> X2` and .6 for `X2 -> X3` and autoregressive effects of .5 for `Y1 -> Y2` and `Y2 -> Y3`. Must be a list of lists for multiple groups models. If the list structure is omitted, no group differences are assumed.

crossedEffects vector of crossed effects of X on Y (`X -> Y`) and vice versa (both constant across waves), or a list of vectors of crossed effects giving the crossed effect of X on Y (and vice versa) for each wave, e.g. `list(c(.2, .3), c(.1, .1))` for `X1 - > Y2` = .2, `X2 -> Y3` = .3, `Y1 -> Y2` = .1, and `Y2 -> Y3` = .1. Must be a list of lists for multiple groups models. If the list structure is omitted, no group differences are assumed.

rXY vector of (residual-)correlations between X and Y for each wave. If `NULL`, all (residual-)correlations are zero. Can be a list for multiple groups models, otherwise no group differences are assumed.

waveEqual parameters that are assumed to be equal across waves in both the H0 and the H1 model. Valid are `'autoregX'` and `'autoregY'` for autoregressive effects, `'crossedX'` and `'crossedY'` for crossed effects, `'corXY'` for residual correlations, or `NULL` for none (so that all parameters are freely estimated, subject to the constraints defined in `nullEffect`).

nullEffect defines the hypothesis of interest. Valid are the same arguments as in `waveEqual` and additionally `'autoregX = 0'`, `'autoregY = 0'`, `'crossedX = 0'`, `'crossedY = 0'` to constrain the X or Y autoregressive effects or the crossed effects to zero, `'autoregX = autoregY'` and `'crossedX = crossedY'` to constrain them to be equal for X and Y, and `'autoregXA = autoregXB'`, `'autoregYA = autoregYB'`, `'crossedXA = crossedXB'`, `'crossedYA = crossedYB'` to constrain them to be equal across groups.

nullWhich used in conjunction with `nullEffect` to identify which parameter to constrain when there are > 2 waves and parameters are not constant across waves. For example, `nullEffect = 'autoregX = 0'` with `nullWhich = 2` would constrain the second autoregressive effect for X to zero.

nullWhichGroups
for hypothesis involving cross-groups comparisons, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

standardized whether all parameters should be treated as standardized (`TRUE`, the default), implying that unstandardized and standardized regression relations have the same value. If `FALSE`, all regression relations are unstandardized.

standardizedResidualCovariances
whether the residual covariances provided in `rXY` should be interpreted as correlations. When `TRUE` (the default) the unstandardized residual covariances differ from the those provided in `rXY`. When `FALSE`, the values provided in `rXY` are the unstandardized residual covariances, and the standardized residual correlations differ.

`metricInvariance`
: whether metric invariance over waves is assumed (`TRUE`, the default) or not (`FALSE`). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

`autocorResiduals`
: whether the residuals of the indicators of latent variables are autocorrelated over waves (`TRUE`, the default) or not (`FALSE`). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

`...`
: mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. The order of factors is (X1, Y1, X2, Y2, ..., X_nWaves, Y_nWaves). See details.

Details

This function performs a power analysis to reject various hypotheses arising in crossed-lagged panel models (CLPM). In a standard CLPM implemented here, two variables X and Y are repeatedly assessed at two or more different time points (`nWaves`), yielding autoregressive effects (stabilities; X1 -> X2 and Y1 -> Y2), synchronous effects (X1 <-> Y1, X2 <-> Y2), and cross-lagged effects (X1 -> Y2 and Y1 -> X2). CLPM including more than two waves are typically implemented assuming that the parameters are constant across waves (`waveEqual`), and usually omit lag-2 effects (e.g., X1 -> Y3). CLPM based on latent factors usually assume at least metric invariance of the factors over waves (`metricInvariance`).

Relevant hypotheses in arising in a CLPM are:

- `autoregX = 0` and `autoregY = 0`: Tests the hypothesis that the autoregressive effect of X and Y, respectively, is zero.
- `crossedX = 0` and `crossedY = 0`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) and of Y on X (`crossedY`), respectively, is zero.
- `autoregX = autoregY`: Tests the hypothesis that the autoregressive effect of X and Y are equal.
- `crossedX = crossedY`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) and of Y on X (`crossedY`) are equal.
- `autoregX` and `autoregY`: Tests the hypothesis that the autoregressive effect of X and Y, respectively, is equal across waves.
- `crossedX` and `crossedY`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) and of Y on X (`crossedY`), respectively, is equal across waves.
- `corXY`: Tests the hypothesis that the (residual-)correlations between X and Y are equal across waves.
- `autoregXA = autoregXB` and `autoregYA = autoregYB`: Tests the hypothesis that the autoregressive effect of either X or Y are equal across groups.
- `crossedXA = crossedXB` and `crossedYA = crossedYB`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) or of Y on X (`crossedY`), respectively, is equal across groups.

For hypotheses regarding the random-intercept CLPM, see `semPower.powerRICLPM()`. For hypothesis in autoregressive models, see `semPower.powerAutoreg()`.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.
- `loadM`: Can be used instead of `Lambda`: Used in conjunction with `nIndicator`. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. `loadM = c(.5, .6)` defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either `Lambda`, or `loadings`, or `nIndicator` and `loadM` need to be defined. If the model contains observed variables only, use `Lambda = diag(x)` where `x` is the number of variables.

Note that the order of the factors is (X1, Y1, X2, Y2, ..., X_nWaves, Y_nWaves), i. e., the first factor is treated as the first measurement of X, the second as the first measurement of Y, the third as the second measurement of X, etc..

Additional arguments related to the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- `nReplications`: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- `minConvergenceRate`: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- `type`: specifies whether the data should be generated from a population assuming multivariate normality (`'normal'`; the default), or based on an approach generating non-normal data (`'IG'`, `'mnonr'`, `'RC'`, or `'VM'`). The approaches generating non-normal data require additional arguments detailed below.

- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# Determine required N in a 2-wave CLPM
# to detect a crossed-effect of X (X1 -> Y2) of >= .2
# with a power of 95% on alpha = 5%, where
# X1 and X2 are measured by 5 indicators loading by .5 each, and
# Y1 and Y2 are measured by 3 indicators loading by .6 each, and
# there is no synchronous correlation between X and Y (rXY = NULL),
# the stability of X is .8,
# the stability of Y is .7, and
# the crossed-effect of Y (Y1 -> X2) is .1.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# show summary
summary(powerCLPM)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerCLPM$modelH1, sample.cov = powerCLPM$Sigma,
            sample.nobs = powerCLPM$requiredN,
            sample.cov.rescale = FALSE)
lavaan::sem(powerCLPM$modelH0, sample.cov = powerCLPM$Sigma,
            sample.nobs = powerCLPM$requiredN,
            sample.cov.rescale = FALSE)

# same as above, but determine power with N = 500 on alpha = .05
powerCLPM <- semPower.powerCLPM(type = 'post-hoc',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, N = 500)

# same as above, but determine the critical chi-square with N = 500 so that alpha = beta
powerCLPM <- semPower.powerCLPM(type = 'compromise',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
```

```
                                nullEffect = 'crossedX = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                abratio = 1, N = 500)

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerCLPM <- semPower.powerCLPM(type = 'compromise',
                                comparison = 'saturated',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                abratio = 1, N = 500)

# same as above, but assume only observed variables
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                Lambda = diag(4),
                                alpha = .05, beta = .05)

# same as above, but provide reduced loadings matrix to define that
# X1 and X2 are measured by 5 indicators each loading by .4, .5, .6, .5, .4
# Y1 and Y2 are measured by 3 indicators each loading by .8, .6, .7
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                loadings = list(
                                  c(.4, .5, .6, .5, .4),    # X1
                                  c(.8, .6, .7),            # Y1
                                  c(.4, .5, .6, .5, .4),    # X2
                                  c(.8, .6, .7)             # Y2
                                ),
                                alpha = .05, beta = .05)

# same as above, but do not assume metric invariance across waves
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                nIndicator = c(5, 3, 5, 3),
```

```
                                loadM = c(.5, .6, .5, .6),
                                metricInvariance = FALSE,
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that the crossed-effect of Y (Y1 -> X2) is >= .1.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedY = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that the stability of X (X1 -> X2) is >= .8.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'autoregX = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that the stability of Y (Y1 -> Y2) is >= .7.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'autoregY = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that
# the crossed effect of X (X1 -> Y2) of .2 differs from
# the crossed effect of Y (Y1 -> X2) of .1
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = crossedY',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that
# the autoregressive effect of X (X1 -> X2) of .8 differs from
# the autoregressive effect of Y (Y1 -> Y2) of .7
```

```
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'autoregX = autoregY',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but assume that the synchronous correlation between X and Y
# is .3 at the first wave, and the respective residual correlation is .2 at the second wave,
# and determine N to detect that synchronous residual correlation (at wave 2) is => .2.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = c(.3, .2),
                                nullEffect = 'corXY = 0',
                                nIndicator = c(5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6),
                                alpha = .05, beta = .05)

# Determine required N in a 3-wave CLPM
# to detect a crossed-effect of X (X1 -> Y2 and X2 -> Y3) of >= .2
# with a power of 95% on alpha = 5%, where
# the crossed, autoregressive, and synchronous effects of X and Y are equal over waves,
# X1, X2, and X3 are measured by 5 indicators loading by .5 each, and
# Y1, Y2, and Y3 are measured by 3 indicators loading by .6 each, and
# the synchronous correlation between X and Y are .2 across all three waves, and
# the stability of X is .8 across all three waves,
# the stability of Y is .7 across all three waves, and
# the crossed-effect of Y (Y1 -> X2, and Y2 -> Y3) is .1.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 3,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = c(.2, .2, .2),
                                waveEqual = c('autoregX', 'autoregY',
                                              'crossedX', 'crossedY'),
                                nullEffect = 'crossedX = 0',
                                nIndicator = c(5, 3, 5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6, .5, .6),
                                alpha = .05, beta = .05)

# Determine required N in a 3-wave CLPM to detect that
# the crossed-effect of X in wave 1 (X1 -> Y2) of .20 is equal to the
# the crossed-effect of X in wave 2 (X2 -> Y3) of .10
# with a power of 95% on alpha = 5%, where
# the autoregressive effects of X and Y are equal over waves,
# X1, X2, and X3 are measured by 5 indicators loading by .5 each, and
# Y1, Y2, and Y3 are measured by 3 indicators loading by .6 each, and
# the synchronous correlation between X and Y are .2, .3, and .4 at the first,
```

```
# second, and third wave, and
# the stability of X is .8 across all three waves,
# the stability of Y is .7 across all three waves, and
# the crossed-effects of Y (Y1 -> X2, and Y2 -> X3) are both .1
# (but freely estimated for each wave).
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 3,
                                autoregEffects = c(.8, .7),
                                crossedEffects = list(
                                  c(.20, .10),   # X1 -> Y2, X2 -> Y3
                                  c(.05, .10)),  # Y1 -> X2, Y2 -> X3
                                rXY = c(.2, .3, .4),
                                nullEffect = 'crossedX',
                                waveEqual = c('autoregX', 'autoregY'),
                                nIndicator = c(5, 3, 5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that
# the crossed-effect of X at wave 2 is >= .10.
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 3,
                                autoregEffects = c(.8, .7),
                                crossedEffects = list(
                                  c(.20, .10),   # X1 -> Y2, X2 -> Y3
                                  c(.05, .10)),  # Y1 -> X2, Y2 -> X3
                                rXY = c(.2, .3, .4),
                                nullEffect = 'crossedX',
                                nullWhich = 2,
                                waveEqual = c('autoregX', 'autoregY'),
                                nIndicator = c(5, 3, 5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6, .5, .6),
                                alpha = .05, beta = .05)

# same as above, but determine N to detect that
# the residual correlation between X and Y at wave 2 (of .3) differs from
# the residual correlation between X and Y at wave 3 (of .4)
# and define unstandardized parameters
powerCLPM <- semPower.powerCLPM(type = 'a-priori',
                                nWaves = 3,
                                autoregEffects = c(.8, .7),
                                crossedEffects = list(
                                  c(.20, .10),   # X1 -> Y2, X2 -> Y3
                                  c(.05, .10)),  # Y1 -> X2, Y2 -> X3
                                rXY = c(.2, .3, .4),
                                nullEffect = 'corXY',
                                waveEqual = c('autoregX', 'autoregY'),
                                standardized = FALSE,
                                nIndicator = c(5, 3, 5, 3, 5, 3),
                                loadM = c(.5, .6, .5, .6, .5, .6),
                                alpha = .05, beta = .05)
```

```
# multiple group example
# determine power in a 3-wave CLPM to detect that
# the autoregressive effect of X in group 1 (of .8) differs from the
# autoregressive effect of X in group 2 (of .6)
# with a 500 observations in both groups on alpha = 5%, where
# the autoregressive effects of X and Y are equal over waves (but not across groups),
# the cross-lagged effects of X and Y are equal over waves (and also across groups),
# X1, X2, and X3 are measured by 5 indicators loading by .5 each, and
# Y1, Y2, and Y3 are measured by 3 indicators loading by .4 each, and
# there are no synchronous correlation between X and Y.
powerCLPM <- semPower.powerCLPM(type = 'post-hoc', alpha = .05, N = list(500, 500),
                                nWaves = 3,
                                autoregEffects = list(
                                # group 1
                                list(c(.8, .8),    # X1 -> X2, X2 -> X3
                                     c(.7, .7)),   # Y1 -> Y2, Y2 -> Y3
                                # group 2
                                list(c(.6, .6),    # X1 -> X2, X2 -> X3
                                     c(.7, .7))    # Y1 -> Y2, Y2 -> Y3
                                ),
                                crossedEffects = c(.2, .1),
                           waveEqual = c('autoregX', 'autoregY', 'crossedX', 'crossedY'),
                                rXY = NULL,
                                nullEffect = 'autoregxa=autoregxb',
                                nIndicator = c(5, 3, 5, 3, 5, 3),
                                loadM = c(.5, .4, .5, .4, .5, .4))


# request a simulated post-hoc power analysis with 500 replications.
set.seed(300121)
powerCLPM <- semPower.powerCLPM(type = 'post-hoc',
                                nWaves = 2,
                                autoregEffects = c(.8, .7),
                                crossedEffects = c(.2, .1),
                                rXY = NULL,
                                nullEffect = 'crossedX = 0',
                                Lambda = diag(4),
                                alpha = .05, N = 500,
                                simulatedPower = TRUE,
                                simOptions = list(nReplications = 500))

## End(Not run)
```

`semPower.powerLav` *semPower.powerLav*

Description

Perform a power analysis given `lavaan` model strings defining the H0 and the H1 model based on either a `lavaan` model string defining the population model or the population covariance matrix Sigma and the population means mu. This requires the `lavaan` package.

Usage

```
semPower.powerLav(
  type,
  modelPop = NULL,
  modelH0 = NULL,
  modelH1 = NULL,
  fitH1model = TRUE,
  Sigma = NULL,
  mu = NULL,
  fittingFunction = "ML",
  simulatedPower = FALSE,
  lavOptions = NULL,
  lavOptionsH1 = lavOptions,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
modelPop	lavaan model string defining the true model. Can be omitted when Sigma is set.
modelH0	lavaan model string defining the (incorrect) analysis model.
modelH1	lavaan model string defining the comparison model. If omitted, the saturated model is the comparison model.
fitH1model	whether to fit the H1 model. If FALSE, the H1 model is assumed to show the same fit as the saturated model, and only the delta df are computed.
Sigma	can be used instead of modelPop: population covariance matrix. A list for multiple group models.
mu	can be used instead of modelPop: vector of population means. Can be omitted for no meanstructure. A list for multiple group models.
fittingFunction	one of 'ML' (default), 'WLS', 'DWLS', 'ULS'. Defines the fitting function used to obtain SigmaHat in analytical power analyses. This also implies a certain discrepancy function used to obtain Fmin.
simulatedPower	whether to perform a simulated (TRUE, rather than analytical, FALSE) power analysis. See simulate() for additional options.
lavOptions	a list of additional options passed to lavaan, e. g., list(estimator = 'mlm') to request robust ML estimation. Mostly useful in conjunction with simulatedPower.
lavOptionsH1	alternative options passed to lavaan that are only used for the H1 model. If NULL, identical to lavOptions. Probably only useful for multigroup models.
...	mandatory further parameters related to the specific type of power analysis requested, see semPower.aPriori(), semPower.postHoc(), and semPower.compromise(). See details.

Details

Generic function to perform a power analysis based on a true population covariance matrix Sigma and a model implied covariance matrix SigmaHat (and optionally the associated mean vectors), where SigmaHat (and muHat) is determined by fitting a respective H0 model using `lavaan`, and Sigma (and mu) can also be provided through a corresponding `lavaan` model string.

All `semPower` convenience functions internally call this function.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- `nReplications`: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- `minConvergenceRate`: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- `type`: specifies whether the data should be generated from a population assuming multivariate normality (`'normal'`; the default), or based on an approach generating non-normal data (`'IG'`, `'mnonr'`, `'RC'`, or `'VM'`). The approaches generating non-normal data require additional arguments detailed below.
- `missingVars`: vector specifying the variables containing missing data (defaults to NULL).
- `missingVarProp`: can be used instead of `missingVars`: The proportion of variables containing missing data (defaults to zero).
- `missingProp`: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- `missingMechanism`: The missing data mechanism, one of `MCAR` (the default), `MAR`, or `NMAR`.
- `nCores`: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the `doFuture` package.
- `futureStrategy`: A string specifying the strategy used in parallel processing (when `nCores` > 1). Defaults to `'multisession'`. This is passed to the `plan` method of the `doFuture` package. See the `doFuture` package for valid options.

`type = 'IG'` implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (`skewness`) and excess kurtosis (`kurtosis`) for each variable are provided as vectors. This requires the `covsim` package.

`type = 'mnonr'` implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (`skewness`) and kurtosis (`kurtosis`), where skewness

must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the `mnonr` package.

`type = 'RK'` implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (`distributions`). `distributions` must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. `rchisq`) and additional arguments (`list(df = 2)`).

`type = 'VM'` implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (`skewness`) and excess kurtosis (`kurtosis`) for each variable are provided as vectors.

Value

a list. Use the `summary` method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

`Sigma`	the population covariance matrix. A list for multiple group models.
`mu`	the population mean vector or `NULL` when no meanstructure is involved. A list for multiple group models.
`SigmaHat`	the H0 model implied covariance matrix. A list for multiple group models.
`muHat`	the H0 model implied mean vector or `NULL` when no meanstructure is involved. A list for multiple group models.
`modelH0`	`lavaan` H0 model string.
`modelH1`	`lavaan` H1 model string or `NULL` when the comparison refers to the saturated model.
`simRes`	detailed simulation results when a simulated power analysis (`simulatedPower = TRUE`) was performed.

See Also

`semPower.aPriori()` `semPower.postHoc()` `semPower.compromise()`

Examples

```
## Not run:
# set up two CFA factors with a true correlation of .2
mPop <- '
  f1 =~ .5*x1 + .6*x2 + .4*x3
  f2 =~ .7*x4 + .8*x5 + .3*x6
  x1 ~~ .75*x1
  x2 ~~ .64*x2
  x3 ~~ .84*x3
  x4 ~~ .51*x4
  x5 ~~ .36*x5
  x6 ~~ .91*x6
  f1 ~~ 1*f1
  f2 ~~ 1*f2
  f1 ~~ .2*f2
```

```
'
# define the H0 analysis model (restricting the factor correlation to zero)
mH0 <- '
  f1 =~ x1 + x2 + x3
  f2 =~ x4 + x5 + x6
  f1 ~~ 0*f2
'
# determine N to reject the H0 that the correlation is zero
# with a power of 95% on alpha = .05
powerLav <- semPower.powerLav(type = 'a-priori',
                              modelPop = mPop, modelH0 = mH0,
                              alpha = .05, beta = .05)
summary(powerLav)

# same as above, but also define an H1 comparison model
mH1 <- '
  f1 =~ x1 + x2 + x3
  f2 =~ x4 + x5 + x6
  f1 ~~ f2
'
powerLav <- semPower.powerLav(type = 'a-priori',
                              modelPop = mPop, modelH0 = mH0, modelH1 = mH1,
                              alpha = .05, beta = .05)

# same as above, but use covariance matrix input instead of modelPop
gen <- semPower.genSigma(Phi = .2,
                         loadings = list(c(.5, .6, .4), c(.7, .8, .3)))
Sigma <- gen$Sigma
powerLav <- semPower.powerLav(type = 'a-priori',
                              Sigma = Sigma, modelH0 = mH0,
                              alpha = .05, beta = .05)

# note all of the above is identical to the output provided by the semPower.powerCFA function
powerCFA <- semPower.powerCFA(type = 'a-priori',
                              comparison = 'saturated',
                              Phi = .2,
                              loadings = list(c(.5, .6, .4), c(.7, .8, .3)),
                              alpha = .05, beta = .05)

# same as above, but perform simulated power analysis
# with 250 replications using a robust ML test-statistic
set.seed(300121)
powerLav <- semPower.powerLav(type = 'a-priori',
                              Sigma = Sigma, modelH0 = mH0,
                              alpha = .05, beta = .05,
                              simulatedPower = TRUE,
                              simOptions = list(nReplications = 250)
                              lavOptions = list(estimator = 'MLM'))

## End(Not run)
```

semPower.powerLGCM *semPower.powerLGCM*

Description

Convenience function for performing power analysis on effects in a latent growth curve model (LGCM). This requires the lavaan package.

Usage

```
semPower.powerLGCM(
  type,
  comparison = "restricted",
  nWaves = NULL,
  means = NULL,
  variances = NULL,
  covariances = NULL,
  quadratic = FALSE,
  timeCodes = NULL,
  ticExogSlopes = NULL,
  ticEndogSlopes = NULL,
  groupEqual = NULL,
  nullEffect = NULL,
  nullWhichGroups = NULL,
  autocorResiduals = TRUE,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
nWaves	number of waves, must be >= 3 for linear and > 3 for quadratic trends.
means	vector providing the means of the intercept and the linear slope factor (and the quadratic slope factor, if quadratic = TRUE). A list for multiple group models.
variances	vector providing the variances of the intercept and the linear slope factor (and the quadratic slope factor, if quadratic = TRUE). Can be omitted, if a matrix is provided to covariances. Takes precedence over the diagonal in covariances when both are defined. A list for multiple group models.

covariances	either the variance-covariance matrix between the intercept and the slope (and the quadratic slope factor, if quadratic = TRUE), or a single number giving the covariance between intercept and slope factor, or NULL for orthogonal factors. A list for multiple group models.
quadratic	whether to include a quadratic slope factor in addition to a linear slope factor. Defaults to FALSE for no quadratic slope factor.
timeCodes	vector of length nWaves defining the loadings on the slope factor. If omitted, the time codes default to (0, 1, ..., (nWaves - 1)).
ticExogSlopes	vector defining the slopes for an exogenous time-invariant covariate in the prediction of the intercept and slope factors (and the quadratic slope factor, if quadratic = TRUE). Can be omitted for no covariate.
ticEndogSlopes	vector defining the slopes for the intercept and slope factors (and the quadratic slope factor, if quadratic = TRUE) in the prediction of an endogenous time-invariant covariate. Can be omitted for no covariate.
groupEqual	parameters that are restricted across groups in both the H0 and the H1 model, when nullEffect implies a multiple group model. Valid are 'imean', 'smean', 's2mean' to restrict the means of the intercept, linear slope, and quadratic slope factors, and 'ivar', 'svar', 's2var' for their variances, and 'iscov', 'is2cov', 'ss2cov' for the covariances between intercept and slope, intercept and quadratic slope, and linear and quadratic slope.
nullEffect	defines the hypothesis of interest. See details for valid arguments.
nullWhichGroups	for hypothesis involving cross-groups comparisons, vector indicating the groups for which equality constrains should be applied, e.g. c(1, 3) to constrain the relevant parameters of the first and the third group. If NULL, all groups are constrained to equality.
autocorResiduals	whether the residuals of the indicators of latent variables are autocorrelated over waves (TRUE, the default) or not (FALSE). This is only applied to second order LGCMs. This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).
...	mandatory further parameters related to the specific type of power analysis requested, see semPower.aPriori(), semPower.postHoc(), and semPower.compromise(), and parameters specifying the factor model. The order of factors is (X1, X2, ..., X_nWaves, ticExogSlopes, ticEndogSlopes). See details.

Details

This function performs a power analysis to reject various hypotheses arising in latent growth curve models (LGCM), where one variable X is repeatedly assessed at different time points (nWaves), and a latent intercept and a linear (and optionally a quadratic) latent slope factor is assumed.

Relevant hypotheses in arising in a LCGM are:

- iMean = 0, sMean = 0, s2Mean = 0: Tests the hypothesis that the mean of the intercept, linear slope, and quadratic slope factors, respectively, is zero.

- `iVar = 0`, `sVar = 0`, `s2Var = 0`: Tests the hypothesis that the variance of the intercept, linear slope, and quadratic slope factors, respectively, is zero.
- `isCov = 0`: Tests the hypothesis that covariance between the intercept and linear slope factor is zero.
- `is2Cov = 0`: Tests the hypothesis that covariance between the intercept and quadratic slope factor is zero.
- `ss2Cov = 0`: Tests the hypothesis that covariance between the linear and the quadratic slope factor is zero.
- `betaIT = 0`, `betaST = 0`, `betaS2T = 0`: Tests the hypothesis that the slope for an exogenous time-invariant covariate in the prediction of the intercept, the linear slope, and the quadratic slope factor, respectively, is zero (`TIC -> I, S, S2`).
- `betaTI = 0`, `betaTS = 0`, `betaTS2 = 0`: Tests the hypothesis that the slope the intercept, the linear slope, and the quadratic slope factor, respectively, in the prediction of an endogenous time-invariant covariate is zero (`I, S, S2 -> TIC`).
- `iMeanA = iMeanB`, `sMeanA = sMeanB`, `s2MeanA = s2MeanB`: Tests the hypothesis that the means of the intercept, linear slope, and quadratic slope factors, respectively, are equal across groups.
- `iVarA = iVarB`, `sVarA = sVarB`, `s2VarA = s2VarB`: Tests the hypothesis that the variances of the intercept, linear slope, and quadratic slope factors, respectively, are equal across groups.
- `isCovA = isCovA`: Tests the hypothesis that covariance between the intercept and linear slope factor is equal across groups.
- `is2CovA = is2CovA`: Tests the hypothesis that the covariance between the intercept and quadratic slope factor is equal across groups.
- `ss2CovA = ss2CovA`: Tests the hypothesis that the covariance between the linear and quadratic slope factor is equal across groups.
- `betaITA = betaITB`, `betaSTA = betaSTB`, `betaS2TA = betaS2TB`: Tests the hypothesis that the slopes for the time-invariant covariate in the prediction of the intercept, the linear slope, and the quadratic slope factor, respectively, are equal across groups (`TIC -> I, S, S2`).
- `betaTIA = betaTIB`, `betaTSA = betaTSB`, `betaTS2A = betaTS2B`: Tests the hypothesis that the slope the intercept, the linear slope, and the quadratic slope factor, respectively, in the prediction of the time-invariant covariate are equal across groups (`I, S, S2 -> TIC`).

For hypotheses regarding longitudinal invariance, see `semPower.powerLI()`. For hypotheses regarding a simple autoregression, see `semPower.powerAutoreg()`. For hypotheses in an ARMA model, see `semPower.powerARMA()`.

Note that power analyses concerning the hypotheses `iVar = 0`, `sVar = 0`, and `s2Var = 0` are only approximate, because the H0 model involves a parameter constraint on the boundary of the parameter space (a variance of zero), so that the correct limiting distribution is a mixture of non-central χ^2 distributions (see Stoel et al., 2006). In effect, power is (slightly) underestimated.

Stoel, R. D., Garre, F. G., Dolan, C., & Van Den Wittenboer, G. (2006). On the likelihood ratio test in structural equation modeling when parameters are subject to boundary constraints. *Psychological Methods, 11*, 439-455.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- Lambda: The factor loading matrix (with the number of columns equaling the number of factors).
- loadings: Can be used instead of Lambda: Defines the primary loadings for each factor in a list structure, e. g. loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4)) defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- nIndicator: Can be used instead of Lambda: Used in conjunction with loadM. Defines the number of indicators by factor, e. g., nIndicator = c(3, 4) defines a two factor model with three and four indicators for the first and second factor, respectively. nIndicator can also be a single number to define the same number of indicators for each factor.
- loadM: Can be used instead of Lambda: Used in conjunction with nIndicator. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. loadM = c(.5, .6) defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either Lambda, or loadings, or nIndicator and loadM need to be defined. Neither may contain entries referring to the intercept and slope factors. In a multigroup structure, only provide the loadings for a single group. If the model contains observed variables only, use Lambda = diag(x) where x is the number of variables.

The order of the factors is (X1, X2, ..., X_nWaves, ticExogenous, ticEndogenous). If ticExogenous is undefined, ticEndogenous takes its place.

Additional arguments related to the requested type of **power analysis**:

- alpha: The alpha error probability. Required for type = 'a-priori' and type = 'post-hoc'.
- Either beta or power: The beta error probability and the statistical power (1 - beta), respectively. Only for type = 'a-priori'.
- N: The sample size. Always required for type = 'post-hoc' and type = 'compromise'. For type = 'a-priori' and multiple group analysis, N is a list of group weights.
- abratio: The ratio of alpha to beta. Only for type = 'compromise'.

If a **simulated power analysis** (simulatedPower = TRUE) is requested, optional arguments can be provided as a list to simOptions:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.

- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# Determine required N in a 3-wave LGCM model
# to detect that the mean of the slope factor differs from zero
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave), and
# the mean of the intercept factor is .5 and
# the mean of the slope factor is .2 and
# the variance of the intercept factor is 1 and
# the variance of the slope factor is .5 and
# the intercept-slope covariance is .25 and
# autocorrelated residuals are assumed
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80,
  nWaves = 3,
  means = c(.5, .2),     # i, s
  variances = c(1, .5),  # i, s
  covariances = .25,
  nullEffect = 'sMean = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# show summary
summary(powerLGCM)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerLGCM$modelH1, sample.cov = powerLGCM$Sigma,
            sample.mean = powerLGCM$mu,
            sample.nobs = powerLGCM$requiredN,
            sample.cov.rescale = FALSE)
lavaan::sem(powerLGCM$modelH0, sample.cov = powerLGCM$Sigma,
            sample.mean = powerLGCM$mu,
            sample.nobs = powerLGCM$requiredN,
            sample.cov.rescale = FALSE)


# same as above, but determine power with N = 250 on alpha = .05
powerLGCM <- semPower.powerLGCM(
  'post-hoc', alpha = .05, N = 250,
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'sMean = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# same as above, but determine the critical chi-square with N = 250 so that alpha = beta
powerLGCM <- semPower.powerLGCM(
  'compromise', abratio = 1, N = 250,
  nWaves = 3,
```

```
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'sMean = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80, comparison = 'saturated',
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'sMean = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# same as above, but assume only observed variables
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80, comparison = 'saturated',
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'sMean = 0',
  Lambda = diag(3),
  autocorResiduals = TRUE
)

# same as above, but provide reduced loadings matrix to define that
# X is measured by 3 indicators each loading by .5, .6, .4 (at each wave)
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80, comparison = 'saturated',
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'sMean = 0',
  loadings = list(
    c(.5, .6, .4),  # X1
    c(.5, .6, .4),  # X2
    c(.5, .6, .4)   # X3
  ),
  autocorResiduals = TRUE
)

# same as above, but get required N to detect that
# the variance of the intercept factor differs from zero
powerLGCM <- semPower.powerLGCM(
```

```
  'a-priori', alpha = .05, power = .80,
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'iVar = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# same as above, but get required N to detect that
# the intercept-slope covariance differs from zero
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80,
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'iscov = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)


# include a quadratic slope factor
# and get required N to detect that
# its variance differs from zero.
# provide the variance-covariance matrix
# between intercept, slope, and quadratic slope factors
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80,
  nWaves = 4,
  quadratic = TRUE,
  means = c(.5, .2, .1),
  covariances = matrix(c(
    # i  s   s2
    c(1, .3, .2),
    c(.3, .5, .01),
    c(.2, .01, .1)
  ), ncol = 3, byrow = TRUE),
  nullEffect = 's2var = 0',
  nIndicator = rep(3, 4), loadM = .5,
  autocorResiduals = TRUE
)

# Determine required N in a 3-wave LGCM model
# to detect that the slope of an time-invariant covariate (TIC)
# on the slope factor differs from zero.
# The TIC is measured by 4 indicators loading
# by .7, .7, .5, and .8. The slope of the TIC in the prediction of
# the intercept factor is .5, and in the prediction of the slope factor is .4.
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80,
```

```
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  ticExogSlopes = c(.5, .4),  # i s
  nullEffect = 'betaST = 0',
  loadings = list(
    c(.5, .6, .4),      # X1
    c(.5, .6, .4),      # X2
    c(.5, .6, .4),      # X3
    c(.7, .7, .5, .8)   # TIC
  ),
  autocorResiduals = TRUE
)

# Determine required N in a 3-wave LGCM model
# to detect that the slope of the slope factor in
# the prediction of a time-invariant covariate (TIC) differs from zero.
# The TIC is measured by 4 indicators loading
# by .7, .7, .5, and .8. The slopes of the intercept and the slope factors in
# the prediction of the TIC are .1 and .3, respectively.
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80,
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = .25,
  ticEndogSlopes = c(.1, .3),  # i s
  nullEffect = 'betaTS = 0',
  loadings = list(
    c(.5, .6, .4),      # X1
    c(.5, .6, .4),      # X2
    c(.5, .6, .4),      # X3
    c(.7, .7, .5, .8)   # TIC
  ),
  autocorResiduals = TRUE
)

# Determine required N in a 3-wave LGCM model
# to detect that the mean of the slope factor in group 1
# differs from the mean of the slope factor in group 2
# with a power of 80% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each (at each wave and in each group), and
# the means of the intercept factor in group 1 and 2 are .5 and .25
# the means of the slope factor in group 1 and 2 are .25 and .4
# the variance of the intercept factor is 1 in both groups and
# the variance of the slope factor is .5in both groups and
# the intercept-slope covariance is .25 in both groups and
# autocorrelated residuals are assumed and
# the groups are equal-sized
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80, N = list(1, 1),
  nWaves = 3,
```

```
  means = list(
    # i, s
    c(.5, .2),     # group 1
    c(.25, .4)),   # group 2
  variances = c(1, .5),
  covariances = .25,
  nullEffect = 'sMeanA = sMeanB',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# similar as above, but get required N to detect that
# the intercept-slope covariance differs across groups,
# assuming that intercept and slope variances are equal across groups.
powerLGCM <- semPower.powerLGCM(
  'a-priori', alpha = .05, power = .80, N = list(1, 1),
  nWaves = 3,
  means = c(.5, .2),
  variances = c(1, .5),
  covariances = list(
    c(.25), # group 1
    c(.1)), # group 2
  nullEffect = 'isCovA = isCovB',
  groupEqual = c('ivar', 'svar'),
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE
)

# perform a simulated post-hoc power analysis
# with 250 replications
set.seed(300121)
powerLGCM <- semPower.powerLGCM(
  'post-hoc', alpha = .05, N = 500,
  nWaves = 3,
  means = c(.5, .2),     # i, s
  variances = c(1, .5),  # i, s
  covariances = .25,
  nullEffect = 'sMean = 0',
  nIndicator = rep(3, 3), loadM = .5,
  autocorResiduals = TRUE,
  simulatedPower = TRUE,
  simOptions = list(nReplications = 250)
)

## End(Not run)
```

`semPower.powerLI` *semPower.powerLI*

Description

Convenience function for performing power analyses for hypothesis arising in longitudinal measurement invariance models concerning a specific level of invariance. This requires the lavaan package.

Usage

```
semPower.powerLI(
  type,
  comparison = NULL,
  nullEffect = NULL,
  autocorResiduals = TRUE,
  Phi = NULL,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = !useReferenceIndicator,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, either 'saturated' or one of 'configural', 'metric', 'scalar', 'residual', 'covariances', 'means', or a vector of restrictions in lavaan format (with 'none' for no restrictions). See details.
nullEffect	defines the hypothesis (i.e., level of invariance) of interest. Accepts the same arguments as comparison. See details.
autocorResiduals	whether the residuals of the indicators of latent variables are autocorrelated over waves (TRUE, the default) or not (FALSE). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).
Phi	the factor covariance matrix. Can be NULL for uncorrelated factors and unit variances.
useReferenceIndicator	whether to identify the factor variances and factor means using a referent indicator (TRUE) or using a standardization approach setting the means to 0 and the variances to 1 (FALSE, the default). See details.
singleOccasionIdent	when useReferenceIndicator = FALSE, whether to maintain the identification constraints only for a single measurement (TRUE) or whether to maintain all scaling constraints (FALSE). See details.
...	mandatory further parameters related to the specific type of power analysis requested, see semPower.aPriori(), semPower.postHoc(), and semPower.compromise(), and parameters specifying the factor model. See details.

Details

This function performs a power analysis to reject various hypotheses arising in the context of longitudinal measurement invariance, where a single attribute is measured repeatedly. The typical - but not in all parts necessary - sequence concerning the measurement part is (a) configural, (b) metric, (c) scalar, (d) residual invariance, and concerning the structural part (e) latent variances, (f) latent covariances, and (g) latent means, where each level of invariance is compared against the previous level (e.g., scalar vs. metric). Power analysis provides the power (or the required N) to reject a particular level of invariance.

For hypotheses regarding multiple group invariance, see `semPower.powerMI()`. For hypotheses regarding autoregressive models, see `semPower.powerAutoreg()`. For hypotheses in an ARMA model, see `semPower.powerARMA()`.

The general model structure posits at least one factor measured at two ore more occasions. The invariance constraints over occasions are always placed separately for each factor. For example, if two factors are measured twice, equality constraints on loadings are placed over the measurements of each factor separately. There are two ways to specify the models defined in the `comparison` and the `nullEffect` arguments. Either, one may specify a specific level of invariance that includes all previous levels:

- `'configural'`: no invariance constraints. Shows the same fit as the saturated model, so only the delta df differ.
- `'metric'`: all loadings (of a particular attribute) are restricted to equality over measurement occasions. Note that if reference scaling is used, the first indicator should be invariant.
- `'scalar'`: all loadings and (indicator-)intercepts are restricted to equality. Note that if reference scaling is used, the first indicator should be invariant.
- `'residual'`: all loadings, (indicator-)intercepts, and (indicator-)residuals are restricted to equality.
- `'variances'`: all loadings, (indicator-)intercepts, (indicator-)residuals, and latent variances are restricted to equality.
- `'covariances'`: all loadings, (indicator-)intercepts, (indicator-)residuals, latent variances, and latent covariances are restricted to equality. This only refers to the factor covariance(s) at each measurement occasion, but does not involve cross-temporal relations such as stabilities.
- `'means'`: all loadings, (indicator-)intercepts, (indicator-)residuals, latent variances, latent covariances, and latent means are restricted to equality.

For example, setting `comparison = 'metric'` and `nullEffect = 'scalar'` determines power to reject the hypothesis that the constraints placed in the scalar invariance model (restricting loadings and intercepts over measurement occasions) over the metric invariance model (restricting only the loadings over measurement occasions) are defensible.

For greater flexibility, the models can also be defined using `lavaan` style restrictions as a vector, namely `'none'` (no restrictions), `'loadings'` (loadings), `'intercepts'` (intercepts), `'residuals'` (residuals), `'lv.variances'` (latent variances), `'lv.covariances'` (latent covariances), `'means'` (latent means). For instance:

- `'none'`: no invariance constraints and thus representing a configural invariance model. Shows the same fit as the saturated model, so only the delta df differ.

- c('loadings'): all loadings (of a particular attribute) are restricted to equality. Note that if reference scaling is used, the first indicator should be invariant.
- c('loadings', 'intercepts'): all loadings and (indicator-)intercepts are restricted to equality. Note that if reference scaling is used, the first indicator should be invariant.
- c('loadings', 'intercepts', 'residuals'): all loadings, (indicator-)intercepts, and (indicator-)residuals are restricted to equality.
- c('loadings', 'residuals'): all loadings and (indicator-)residuals are restricted to equality.
- c('loadings', 'intercepts', 'means'): all loadings, (indicator-)intercepts, and latent factor means are restricted to equality.
- c('loadings', 'residuals', 'lv.variances'): all loadings, (indicator-)residuals, and latent factor variances are restricted to equality.

For example, setting comparison = c('loadings') and nullEffect = 'c('loadings', 'intercepts')' determines power to reject the hypothesis that the constraints placed in the scalar invariance model (restricting loadings and intercepts) over the metric invariance model (restricting only the loadings) are defensible.

By default, the factors are identified by setting their means to 0 and their variances to 1 (useReferenceIndicator = FALSE), so that all loadings and indicator intercepts are freely estimated. In models involving constraints on loadings or intercepts, a single occasion identification approach is used by default (singleOccasionIdent = TRUE), so that the factor variances and means are constrained at only a single occasion, but are freely estimated at all remaining occasions. This mimics the lavaan approach to invariance testing. If setting singleOccasionIdent = FALSE, the identification constraints are maintained in models assuming metric or scalar invariance, so that the factor means and variances are constrained to 0 and 1, respectively, across all occasions (so invariance of factor variances and factor means is always met). Concerning metric invariance, this was the default behavior in semPower versions prior 2.1.4. A referent scaling approach can be be defined using useReferenceIndicator = TRUE, so that the loading and the intercept of the first indicator of each factor are constrained at all occasions, and the factor variances and means are freely estimated. Reference scaling is always maintained in all invariance models, so singleOccasionIdent has no effect (i.e. is always FALSE)

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- Lambda: The factor loading matrix, where the order is F1_T1, F2_T1, ..., F1_T2, F2_T2, ..., F1_TZ, ..., FX_TZ
- loadings: Can be used instead of Lambda: Defines the primary loadings for each factor in a list structure, e. g. loadings = list(c(.5, .4, .6), c(.5, .7, .7)) defines two factors with three indicators loading on the first factor by .5, , 4., and .6, and three indicators loading on the second factor by .5, .7, .7.
- nIndicator: Can be used instead of Lambda: Used in conjunction with loadM. Defines the number of indicators by factor, e. g., nIndicator = c(3, 3) defines a two factor model with three indicators each, respectively. nIndicator can also be a single number to define the same number of indicators for each factor.

- loadM: Can be used instead of Lambda: Used in conjunction with nIndicator. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. loadM = c(.5, .6) defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.
- Theta: Variance-covariance matrix of the indicator residuals, which should be a diagonal matrix. Required when residual non-invariance is to be detected. When NULL, Theta is a diagonal matrix with elements such that all variances are 1.
- tau: Defines the indicator intercepts, required whenever a model involves hypotheses about means (e.g., scalar invariance). If NULL and Alpha is set, all intercepts are assumed to equal zero.
- Alpha: Defines the latent means, required whenever a model involves hypotheses about latent means (e.g., latent mean invariance). If NULL and tau is set, all latent means are assumed to equal zero. If variance scaling is used so that all factor variances are 1, latent mean differences can be interpreted akin to Cohen's d as standardized mean differences.

So either Lambda, or loadings, or nIndicator and loadM always need to be defined, and Theta, tau and Alpha need to be defined for particular levels of invariance.

If longitudinal invariance for a single factor measured at T occasions is of interest, loadings can be a single list with T elements, and nIndicator, loadM, tau and Alpha can be single vectors. If longitudinal invariance for a multiple factor measured at T occasions is of interest, loadings, nIndicator, loadM, tau and Alpha must be wrapped in a list with T elements, where each element refers to a single measurement occasion. Phi and Theta always must be a single matrix. The order of Phi must be F1_T1, F2_T1, ..., F1_T2, F2_T2, ..., F1_TZ, F2_TZ, ..., FX_TZ. The order of Theta must match the order of Phi and thus first contains the respective indicators of F1 at T1, F2 at T1, ..., F1 at T2, F2 at T2, and so on.

Additional arguments related to the requested type of **power analysis**:

- alpha: The alpha error probability. Required for type = 'a-priori' and type = 'post-hoc'.
- Either beta or power: The beta error probability and the statistical power (1 - beta), respectively. Only for type = 'a-priori'.
- N: The sample size. Always required for type = 'post-hoc' and type = 'compromise'. For type = 'a-priori' and multiple group analysis, N is a list of group weights.
- abratio: The ratio of alpha to beta. Only for type = 'compromise'.

If a **simulated power analysis** (simulatedPower = TRUE) is requested, optional arguments can be provided as a list to simOptions:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).

- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:


# obtain the required N to reject the hypothesis of metric invariance
# in comparison to the configural invariance model
# with a power of 80% on alpha = 5%
# for amodel involving a two factors (= two measurements) which
# is measured by 5 indicators
# loading by .5 each at the first measurement occasion
# loading by .6 each in the second measurement occasion,
# and assuming autocorrelated residuals
powerLI <- semPower.powerLI(
  type = 'a-priori', alpha = .05, power = .80,
  comparison = 'configural',
  nullEffect = 'metric',
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.4, .4, .4, .4)    # factor 1 at T2
  ),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)

# show summary
summary(powerLI)

# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerLI$modelH1, sample.cov = powerLI$Sigma,
            sample.nobs = 1000, sample.cov.rescale = FALSE)
lavaan::sem(powerLI$modelH0, sample.cov = powerLI$Sigma,
            sample.nobs = 1000, sample.cov.rescale = FALSE)



# same as above, but determine power with N = 500 on alpha = .05
powerLI <- semPower.powerLI(
  type = 'post-hoc', alpha = .05, N = 500,
  comparison = 'configural',
  nullEffect = 'metric',
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.4, .4, .4, .4)    # factor 1 at T2
  ),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
```

```
)


# same as above, but determine the critical chi-square with N = 500 in each
# group so that alpha = beta
powerLI <- semPower.powerLI(
  type = 'compromise', abratio = 1, N = 500,
  comparison = 'configural',
  nullEffect = 'metric',
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.4, .4, .4, .4)    # factor 1 at T2
  ),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)


# same as above, but compare to the saturated model
# (rather than to the configural invariance model)
powerLI <- semPower.powerLI(
  type = 'a-priori', alpha = .05, power = .80,
  comparison = 'saturated',
  nullEffect = 'metric',
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.4, .4, .4, .4)    # factor 1 at T2
  ),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)


# define two factors measured at three occasions using a
# reduced loading matrix, and provide factor covariance matrix
Phi <- matrix(c(
  # F1T1 F2T1 F1T2 F2T2 F1T3 F2T3
  c(1,   .1,  .7,   0,  .5,   0),     # F1_T1
  c(.1,   1,  .0,  .6,   0,  .3),     # F2_T1
  c(.7,  .0,   1,  .1,  .7,   0),     # F1_T2
  c(.0,  .6,  .1,   1,   0,  .6),     # F2_T2
  c(.5,  .0,  .7,   0,   1,  .1),     # F1_T3
  c(.0,  .3,   0,  .6,  .1,   1)      # F2_T3
), nrow=6, ncol=6, byrow = TRUE)
powerLI <- semPower.powerLI(
  type = 'a-priori', alpha = .05, power = .80,
  comparison = 'configural',
  nullEffect = 'metric',
  loadings = list(
    list(
      c(.8, .8, .8),   # factor 1 at T1
```

```
      c(.5, .5, .5)    # factor 2 at T1
    ),
    list(
      c(.7, .6, .7),   # factor 1 at T2
      c(.4, .6, .4)    # factor 2 at T2
    ),
    list(
      c(.6, .8, .4),   # factor 1 at T3
      c(.5, .5, .7)    # factor 2 at T3
    )
  ),
  Phi = Phi,
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)

# obtain the required N to reject the hypothesis of scalar invariance
# in comparison to the metric invariance model
# with a power of 80% on alpha = 5%
# all intercepts are 0.0 at the first measurement occasion, but
# all intercepts are 0.2 at the second measurement occasion and
powerLI <- semPower.powerLI(
  type = 'a-priori', alpha = .05, power = .80,
  comparison = 'metric',
  nullEffect = 'scalar',
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.7, .6, .5, .7)    # factor 1 at T2
  ),
  tau = c(0, 0, 0, 0,
          .1, .2, .3, .2),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)

# same as above, but use lavaan strings to define invariance models
powerLI <- semPower.powerLI(
  type = 'a-priori', alpha = .05, power = .80,
  comparison = c('loadings'),
  nullEffect = c('loadings', 'intercepts'),
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.7, .6, .5, .7)    # factor 1 at T2
  ),
  tau = c(0, 0, 0, 0,
          .1, .2, .3, .2),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)
```

```
# obtain the required N to reject the hypothesis of equal latent means
# in comparison to the scalar invariance model;
# all intercepts are zero in both groups,
# at the first measurement occasion, the latent mean is 0.0,
# at the first measurement occasion, the latent mean is 0.5
powerLI <- semPower.powerLI(
  type = 'a-priori', alpha = .05, power = .80,
  comparison = c('loadings', 'intercepts'),
  nullEffect = c('loadings', 'intercepts', 'means'),
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.7, .6, .5, .7)    # factor 1 at T2
  ),
  tau = c(0, 0, 0, 0,
          0, 0, 0, 0),
  Alpha = c(0, .5),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE
)

# request a simulated post-hoc power analysis with 250 replications
# to reject the hypothesis of equal latent means.
set.seed(300121)
powerLI <- semPower.powerLI(
  type = 'post-hoc', alpha = .05, N = 500,
  comparison = 'configural',
  nullEffect = 'metric',
  loadings = list(
    c(.7, .6, .5, .7),   # factor 1 at T1
    c(.4, .4, .4, .4)    # factor 1 at T2
  ),
  autocorResiduals = TRUE,
  useReferenceIndicator = FALSE,
  singleOccasionIdent = TRUE,
  simulatedPower = TRUE,
  simOptions = list(nReplications = 250)
)


## End(Not run)
```

`semPower.powerMediation` *semPower.powerMediation*

Description

Convenience function for performing power analysis concerning indirect effect(s) in a mediation model. This requires the lavaan package.

Usage

```
semPower.powerMediation(
  type,
  comparison = "restricted",
  bYX = NULL,
  bMX = NULL,
  bYM = NULL,
  Beta = NULL,
  indirect = NULL,
  estimateDirectEffects = TRUE,
  nullEffect = "ind = 0",
  nullWhichGroups = NULL,
  standardized = TRUE,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
bYX	the slope (direct effect) for X -> Y. A list for multiple group models. Can be NULL if Beta is set.
bMX	the slope for X -> M. A list for multiple group models. Can be NULL if Beta is set.
bYM	the slope for M -> Y. A list for multiple group models. Can be NULL if Beta is set.
Beta	can be used instead of bYX, bMX, and bYM: matrix of regression weights connecting the latent factors (all-Y notation). Exogenous variables must be in the first row(s), so the upper triangular of Beta must be zero. A list for multiple group models.
indirect	NULL unless Beta is set. Otherwise a list of vectors of size 2 indicating the elements of Beta that define the indirect effect of interest, e.g. list(c(2, 1), c(3, 2)). See details.
estimateDirectEffects	Whether to estimate all direct effects (TRUE, the default). If FALSE, only direct effects unequal zero in the population are estimated.
nullEffect	defines the hypothesis of interest, must be one of 'ind = 0' (the default) to test whether the indirect effect is zero or 'indA = indB' to test for the equality of indirect effects across groups. See details.

`nullWhichGroups`
: for `nullEffect = 'indA = indB'`, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

`standardized`
: whether all parameters should be standardized (`TRUE`, the default). If `FALSE`, all regression relations are unstandardized.

`...`
: mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. In case of a simple mediation, the order of factors is X, M, Y. See details.

Details

This function performs a power analysis to reject various hypotheses arising in the context of mediation:

- `nullEffect = 'ind = 0'`: Tests the hypothesis that an indirect effect is zero.
- `nullEffect = 'indA = indB'`: Tests the hypothesis that an indirect effect is equal in two or more groups. This is currently only possible for models without latent variables.

The indirect effect of interest can be specified in two ways:

- If a simple mediation involving three variables of the form `X -> M -> Y` is assumed, the arguments `bYX`, `bMX`, and `bYM` are used to define the respective slopes, e. g. `bYX = .4`, `bMX = .5`, and `bYM = .3` translates to `X -- .5 --> M -- .3 --> Y` and `X -- .4 --> Y`.
- More complex mediation structures can be defined by providing the `Beta` matrix along with `indirect` specifying which paths define the indirect effect. See examples below. To specify residual correlations, use `semPower.powerPath()` in conjunction with `semPower.powerLav()`.

Notes on implementation:

- For models without latent variables, `nullEffect = 'ind = 0'` and `nullEffect = 'indA = indB'` constrain the indirect effect to zero and to equality, respectively, yielding the test described in Tofighi & Kelley (2020).
- For models with latent variables and `nullEffect = 'ind = 0'`, power is (sometimes roughly) approximated by constraining the smallest slope contained in the indirect effect to zero.
- For models with latent variables multiple groups (i. e., `nullEffect = 'indA = indB'`), there is currently no way to determine power.

Tofighi, D., & Kelley, K. (2020). Improved inference in mediation analysis: Introducing the model-based constrained optimization procedure. *Psychological Methods, 25(4)*, 496–515. https://doi.org/10.1037/met0000259

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).

- loadings: Can be used instead of Lambda: Defines the primary loadings for each factor in a list structure, e. g. loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4)) defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- nIndicator: Can be used instead of Lambda: Used in conjunction with loadM. Defines the number of indicators by factor, e. g., nIndicator = c(3, 4) defines a two factor model with three and four indicators for the first and second factor, respectively. nIndicator can also be a single number to define the same number of indicators for each factor.
- loadM: Can be used instead of Lambda: Used in conjunction with nIndicator. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. loadM = c(.5, .6) defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either Lambda, or loadings, or nIndicator and loadM need to be defined. If the model contains observed variables only, use Lambda = diag(x) where x is the number of variables.

Note that in case of a simple mediation model involving three variables, the order of the factors is X, M, Y, i. e., the first factor is treated as X, the second as M, and the third as Y. In case of a more complex mediation defined via the Beta matrix, the order of factors matches the order of Beta.

Additional arguments related to the requested type of **power analysis**:

- alpha: The alpha error probability. Required for type = 'a-priori' and type = 'post-hoc'.
- Either beta or power: The beta error probability and the statistical power (1 - beta), respectively. Only for type = 'a-priori'.
- N: The sample size. Always required for type = 'post-hoc' and type = 'compromise'. For type = 'a-priori' and multiple group analysis, N is a list of group weights.
- abratio: The ratio of alpha to beta. Only for type = 'compromise'.

If a **simulated power analysis** (simulatedPower = TRUE) is requested, optional arguments can be provided as a list to simOptions:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.

- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# simple case of X -> M -> Y mediation in the form of
# X -- .30 --> M -- .40 --> Y
```

```
# X --------- .25 --------> Y
# determine the required N to detect the indirect effect of >= .12 (= .3 * .4)
# with a power of 95% on alpha = 5%, where
# X is measured by 3 indicators loading by .5 each,
# M is measured by 5 indicators loading by .6 each,
# Y is measured by 4 indicators loading by .7 each.
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05,
                                    bYX = .25, bMX = .3, bYM = .4,
                                    nIndicator = c(3, 5, 4),
                                    loadM = c(.5, .6, .7)
                                    )
# show summary
summary(powerMed)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerMed$modelH1, sample.cov = powerMed$Sigma,
sample.nobs = powerMed$requiredN, sample.cov.rescale = FALSE)
lavaan::sem(powerMed$modelH0, sample.cov = powerMed$Sigma,
sample.nobs = powerMed$requiredN, sample.cov.rescale = FALSE)

# same as above, but determine power with N = 500 on alpha = .05
powerMed <- semPower.powerMediation(type = 'post-hoc',
                                    alpha = .05, N = 500,
                                    bYX = .25, bMX = .3, bYM = .4,
                                    nIndicator = c(3, 5, 4),
                                    loadM = c(.5, .6, .7))


# same as above, but determine the critical chi-square with N = 500 so that alpha = beta
powerMed <- semPower.powerMediation(type = 'compromise',
                                    abratio = 1, N = 500
                                    bYX = .25, bMX = .3, bYM = .4,
                                    nIndicator = c(3, 5, 4),
                                    loadM = c(.5, .6, .7)
                                    )


# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05,
                                    comparison = 'saturated',
                                    bYX = .25, bMX = .3, bYM = .4,
                                    nIndicator = c(3, 5, 4),
                                    loadM = c(.5, .6, .7)
                                    )


# same as above, but assuming observed variables only (Lambda = diag(3))
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05,
                                    bYX = .25, bMX = .3, bYM = .4,
                                    Lambda = diag(3)
```

```
                                    )


# same mediation model as above, but specifying Beta and indirect
Beta <- matrix(c(
  #   X    M    Y
  c(.00, .00, .00),    # X
  c(.30, .00, .00),    # M
  c(.25, .40, .00)     # Y
), byrow = TRUE, ncol = 3)
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05
                                    Beta = Beta,
                                    indirect = list(c(2, 1), c(3, 2)),
                                    nIndicator = c(3, 5, 4),
                                    loadM = c(.5, .6, .7)
                                    )

# Beta for a more complex mediation hypothesis
# of the form X -- .2 --> M1 -- .3 --> M2 -- .40 -> Y
# (and all other effects being zero)
# using a reduced loading matrix to define that
# X is measured by 3 indicators loading by .4, .5, .8
# M1 is measured by 4 indicators loading by .7, .6, .5, .8
# M2 is measured by 5 indicators loading by .5, .6, .3, .4, .6
# Y is measured by 4 indicators loading by .6, .7, .8
Beta <- matrix(c(
  c(.00, .00, .00, .00),       # X
  c(.20, .00, .00, .00),       # M1
  c(.00, .30, .00, .00),       # M2
  c(.00, .00, .40, .00)        # Y
), byrow = TRUE, ncol = 4)
loadings <- list(
  c(0.4, 0.5, 0.8),           # X
  c(0.7, 0.6, 0.5, 0.8),      # M1
  c(0.5, 0.6, 0.3, 0.4, 0.6), # M2
  c(0.6, 0.7, 0.8)            # Y
)
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05,
                                    Beta = Beta,
                                    indirect = list(c(2, 1),
                                                    c(3, 2),
                                                    c(4, 3)),
                                    loadings = loadings)


# Determine required N to detect that the indirect effect
# in group 1 (of .2 * .3 = .09) differs from the indirect effect
# in group 2 (of .3 * .5 = .15).
# The direct effect of X on Y is .25 in both groups.
# The model is based on observed variables only (Lambda = diag(3))
```

```
# Both groups are sized equally (N = list(1, 1)).
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05, N = list(1, 1),
                                    nullEffect = 'indA = indB',
                                    bYX = list(.25, .25),
                                    bMX = list(.2, .3),
                                    bYM = list(.3, .5),
                                    Lambda = diag(3)
                                    )

# same as above, but specifying Beta
Beta1 <- matrix(c(
  c(.00, .00, .00),    # X
  c(.20, .00, .00),    # M
  c(.25, .30, .00)     # Y
), byrow = TRUE, ncol = 3)
Beta2 <- matrix(c(
  c(.00, .00, .00),    # X
  c(.30, .00, .00),    # M
  c(.25, .50, .00)     # Y
), byrow = TRUE, ncol = 3)
powerMed <- semPower.powerMediation(type = 'a-priori',
                                    alpha = .05, beta = .05, N = list(1, 1),
                                    nullEffect = 'indA = indB',
                                    Beta = list(Beta1, Beta2),
                                    indirect = list(c(2, 1), c(3, 2)),
                                    Lambda = diag(3)
                                    )

# request a simulated post-hoc power analysis with 500 replications.
set.seed(300121)
powerMed <- semPower.powerMediation(type = 'post-hoc',
                                    alpha = .05, N = 500,
                                    bYX = .25, bMX = .3, bYM = .4,
                                    nIndicator = c(3, 5, 4),
                                    loadM = c(.5, .6, .7),
                                    simulatedPower = TRUE,
                                    simOptions = list(nReplications = 500))

## End(Not run)
```

`semPower.powerMI` *semPower.powerMI*

Description

Convenience function for performing power analyses for hypothesis arising in multigroup measurement invariance models concerning a specific level of invariance. This requires the lavaan package.

Usage

```
semPower.powerMI(
  type,
  comparison = NULL,
  nullEffect = NULL,
  useReferenceIndicator = FALSE,
  singleGroupIdent = !useReferenceIndicator,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, either 'saturated' or one of 'configural', 'metric', 'scalar', 'variances', 'covariances', or a vector of restrictions in lavaan format (with 'none' for no restrictions). See details.
nullEffect	defines the hypothesis (i.e., level of invariance) of interest. One of 'metric', 'scalar', 'residual', 'variances', 'covariances', 'means' or a vector of restrictions in lavaan format. See details.
useReferenceIndicator	whether to identify the factor variances and factor means using a referent indicator (TRUE) or using a standardization approach setting the means to 0 and the variances to 1 (FALSE, the default). See details.
singleGroupIdent	when useReferenceIndicator = FALSE, whether to maintain the identification constraints only for a single group (TRUE) or whether to maintain all scaling constraints (FALSE). See details.
...	mandatory further parameters related to the specific type of power analysis requested, see semPower.aPriori(), semPower.postHoc(), and semPower.compromise(), and parameters specifying the factor model. See details.

Details

This function performs a power analysis to reject various hypotheses arising in the context of multigroup measurement invariance. Multigroup invariance models fit the specified model simultaneously to various groups and place increasingly restrictive cross-group equality constrains on the model parameters. The typical - but not in all parts necessary - sequence is (a) configural, (b) metric, (c) scalar, and (d) residual invariance, where each level of invariance is compared against the previous level (e.g., scalar vs. metric). Power analysis provides the power (or the required N) to reject a particular level of invariance.

For hypotheses regarding longitudinal invariance, see semPower.powerLI().

The models defined in the comparison and the nullEffect arguments can be specified in two ways. Either specify a specific level of invariance that includes all previous levels:

- 'configural': no invariance constraints. Shows the same fit as the saturated model, so only the delta df differ.
- 'metric': all loadings are restricted to equality.

- 'scalar': all loadings and (indicator-)intercepts are restricted to equality.
- 'residual': all loadings, (indicator-)intercepts, and (indicator-)residuals are restricted to equality.
- 'variances': all loadings, (indicator-)intercepts, (indicator-)residuals, and latent variances are restricted to equality.
- 'covariances': all loadings, (indicator-)intercepts, (indicator-)residuals, latent variances, and latent covariances are restricted to equality.
- 'means': all loadings, (indicator-)intercepts, (indicator-)residuals, latent variances, latent covariances, and latent means are restricted to equality.

For example, setting comparison = 'metric' and nullEffect = 'scalar' determines power to reject the hypothesis that the constraints placed in the scalar invariance model (restricting loadings and intercepts) over the metric invariance model (restricting only the loadings) are defensible.

For greater flexibility, the models can also be defined using lavaan style group.equal restrictions as a vector:

- 'none': no invariance constraints and thus representing a configural invariance model. Shows the same fit as the saturated model, so only the delta df differ.
- c('loadings'): all loadings are restricted to equality.
- c('loadings', 'intercepts'): all loadings and (indicator-)intercepts are restricted to equality.
- c('loadings', 'intercepts', 'residuals'): all loadings, (indicator-)intercepts, and (indicator-)residuals are restricted to equality.
- c('loadings', 'residuals'): all loadings and (indicator-)residuals are restricted to equality.
- c('loadings', 'intercepts', 'means'): all loadings, (indicator-)intercepts, and latent factor means are restricted to equality.

For example, setting comparison = c('loadings') and nullEffect = 'c('loadings', 'intercepts')' determines power to reject the hypothesis that the constraints placed in the scalar invariance model (restricting loadings and intercepts) over the metric invariance model (restricting only the loadings) are defensible.

By default, the factors are identified by setting their means to 0 and their variances to 1 (useReferenceIndicator = FALSE), so that all loadings and indicator intercepts are freely estimated. In models involving constraints on loadings or intercepts, a single group identification approach is used by default (singleGroupIdent = TRUE), so that the factor variances and means are constrained in only a single group, but are freely estimated in all remaining groups. This mimics the lavaan approach to invariance testing. If setting singleGroupIdent = FALSE, the identification constraints are maintained in models assuming metric or scalar invariance, so that the factor means and variances are constrained to 0 and 1, respectively, in all groups. Concerning metric invariance, this was the default behavior in semPower versions prior 2.1.4. A referent scaling approach can be be defined using useReferenceIndicator = TRUE, so that the loading and the intercept of the first indicator of each factor are constrained in each group, and the factor variances and means are freely estimated. Reference scaling is always maintained in all invariance models, so singleGroupIdent has no effect (i.e. is always FALSE)

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.
- `loadM`: Can be used instead of `Lambda`: Used in conjunction with `nIndicator`. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. `loadM = c(.5, .6)` defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.
- `Theta`: Variance-covariance matrix of the indicator residuals, which should be a diagonal matrix. Required when residual non-invariance is to be detected. When `NULL`, Theta is a diagonal matrix with elements such that all variances are 1.
- `tau`: Defines the item intercepts, required whenever a model involves hypotheses about means (e.g., scalar invariance). If `NULL` and `Alpha` is set, all intercepts are assumed to equal zero.
- `Alpha`: Defines the latent means, required whenever a model involves hypotheses about latent means (e.g., latent mean invariance). If `NULL` and `tau` is set, all latent means are assumed to equal zero. Because variance scaling is used so that all factor variances are 1, latent mean differences can be interpreted akin to Cohen's d as standardized mean differences.

So either `Lambda`, or `loadings`, or `nIndicator` and `loadM` always need to be defined, and `Theta`, `tau` and `Alpha` need to be defined for particular levels of invariance. As this function operates on multiple groups, either argument is a list whenever there are group differences in the respective parameters. When no list is provided, the same parameter values are assumed for all groups.

Additional arguments related to the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- `nReplications`: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.

- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.

modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# obtain the required N to reject the hypothesis of metric invariance
# in comparison to the configural invariance model
# with a power of 80% on alpha = 5%
# assuming equally sized groups (N = list(1, 1))
# for a factor model involving a single factor which
# is measured by 4 indicators (in both groups)
# loading by .9, .8, .7, .6 in the first group and
# loading by .7, .7, .8, .7 in the second group.
powerMI <- semPower.powerMI(type = 'a-priori',
                            alpha = .05, power = .80, N = list(1, 1),
                            comparison = 'configural',
                            nullEffect = 'metric',
                            loadings = list(
                              list(c(.9, .8, .7, .6)),
                              list(c(.7, .7, .8, .7)))
)

# show summary
summary(powerMI)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerMI$modelH1, sample.cov = list(powerMI$Sigma[[1]], powerMI$Sigma[[2]]),
            sample.nobs = as.list(powerMI$requiredN.g), sample.cov.rescale = FALSE)

# same as above, but determine power with N = 500 in each group on alpha = .05
powerMI <- semPower.powerMI(type = 'post-hoc',
                            alpha = .05, N = list(500, 500),
                            comparison = 'configural',
                            nullEffect = 'metric',
                            loadings = list(
                              list(c(.9, .8, .7, .6)),
                              list(c(.7, .7, .8, .7)))
                            )

# same as above, but determine the critical chi-square with N = 500 in each
# group so that alpha = beta
powerMI <- semPower.powerMI(type = 'compromise',
                            abratio = 1, N = list(500, 500),
                            comparison = 'configural',
                            nullEffect = 'metric',
```

```
                            loadings = list(
                              list(c(.9, .8, .7, .6)),
                              list(c(.7, .7, .8, .7)))
                            )

# same as above, but compare to the saturated model
# (rather than to the configural invariance model)
powerMI <- semPower.powerMI(type = 'a-priori',
                            alpha = .05, beta = .05, N = list(1, 1),
                            comparison = 'saturated',
                            nullEffect = 'metric',
                            loadings = list(
                              list(c(.9, .8, .7, .6)),
                              list(c(.7, .7, .8, .7)))
                            )

# same as above, but make first group twice as large as the second group
powerMI <- semPower.powerMI(type = 'a-priori',
                            alpha = .05, beta = .05, N = list(2, 1),
                            comparison = 'saturated',
                            nullEffect = 'metric',
                            loadings = list(
                              list(c(.9, .8, .7, .6)),
                              list(c(.7, .7, .8, .7)))
                            )

# obtain the required N to reject the hypothesis of scalar invariance
# in comparison to the metric invariance model
# with a power of 95% on alpha = 5%
# assuming equally sized groups (N = list(1, 1))
# for a two factor model, where both factors are
# measured by 3 indicators each and all loadings equal .5 (in both groups),
# the factor correlation is .3 in both groups, and the
# all intercepts are 0.0 in the first group, but
# the intercepts are .1, .2, .3, .4, .5, .6 in the second group
powerMI <- semPower.powerMI(type = 'a-priori',
                            alpha = .05, beta = .05, N = list(1, 1)
                            comparison = 'metric',
                            nullEffect = 'scalar',
                            Phi = list(.3, .3),
                            nIndicator = list(
                              c(3, 3),
                              c(3, 3)),
                            loadM = .5,
                            tau = list(
                              rep(0.0, 6),
                              seq(.1, .6, .1)
                            ))

# same as above, but use lavaan group.equal strings
powerMI <- semPower.powerMI(type = 'a-priori',
                            alpha = .05, beta = .05, N = list(1, 1)
                            comparison = c('loadings'),
```

```
                            nullEffect = c('loadings', 'intercepts'),
                            Phi = list(.3, .3),
                            nIndicator = list(
                              c(3, 3),
                              c(3, 3)),
                            loadM = .5,
                            tau = list(
                              rep(0.0, 6),
                              seq(.1, .6, .1)
                            ))

# same as above, but
# obtain the required N to reject the hypothesis of equal latent means
# in comparison to the scalar invariance model;
# all intercepts are zero in both groups,
# in the first group, the latent means equal 0.0,
# in the second group, the latent mean of the factors are 0.0 and 0.5
powerMI <- semPower.powerMI(type = 'a-priori',
                            alpha = .05, beta = .05, N = list(1, 1),
                            comparison = c('loadings', 'intercepts'),
                            nullEffect = c('loadings', 'intercepts', 'means'),
                            Phi = list(.3, .3),
                            nIndicator = list(
                              c(3, 3),
                              c(3, 3)),
                            loadM = .5,
                            tau = list(
                              rep(0.0, 6),
                              rep(0.0, 6)
                            ),
                            Alpha = list(
                              c(0.0, 0.0),
                              c(0.0, 0.5)
                            ))

# request a simulated post-hoc power analysis with 500 replications
# to reject the hypothesis of metric invariance.
set.seed(300121)
powerMI <- semPower.powerMI(type = 'post-hoc',
                            alpha = .05, N = list(500, 500),
                            comparison = 'configural',
                            nullEffect = 'metric',
                            loadings = list(
                              list(c(.9, .8, .7, .6)),
                              list(c(.7, .7, .8, .7))),
                            simulatedPower = TRUE,
                            simOptions = list(nReplications = 500))


## End(Not run)
```

semPower.powerPath *semPower.powerPath*

Description

Convenience function for performing power analyses for hypothesis arising in a generic path model. This requires the lavaan package.

Usage

```
semPower.powerPath(
  type,
  comparison = "restricted",
  Beta,
  Psi = NULL,
  nullEffect = "beta = 0",
  nullWhich = NULL,
  nullWhichGroups = NULL,
  standardized = TRUE,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
Beta	matrix of regression slopes between latent variables (all-Y notation). A list for multiple group models. Exogenous variables must occupy the first rows in Beta when standardized = TRUE. See details.
Psi	variance-covariance matrix of latent (residual) factors. If standardized = TRUE, the diagonal is ignored and all off-diagonal elements are treated as correlations. If NULL, an identity matrix is assumed. A list for multiple group models. See details.
nullEffect	defines the hypothesis of interest, must be one of 'beta = 0' (the default) to test whether a regression slope is zero, 'betaX = betaZ' to test for the equality of slopes, and 'betaX = betaZ' to test for the equality of a slope across groups. Define the slopes to be set to equality in nullWhich and the groups in nullWhichGroups.
nullWhich	vector of size 2 indicating which slope in Beta is hypothesized to equal zero when nullEffect = 'beta = 0', or to restrict to equality across groups when

`nullEffect = 'betaA = betaB'`, or list of vectors defining which correlations to restrict to equality when `nullEffect = 'betaX = betaZ'`. Can also contain more than two slopes, e.g., `list(c(2, 1), c(3, 1), c(3, 2))` to set `Beta[2, 1] = Beta[3, 1] = Beta[3, 2]`.

`nullWhichGroups`
: for `nullEffect = 'betaA = betaB'`, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

`standardized`
: whether all parameters should be standardized (`TRUE`, the default). If `FALSE`, all regression relations are unstandardized.

`...`
: mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. See details.

Details

This function performs a power analysis to reject a hypothesis arising in a generic structural equation model specifying regression relations between the factors via the Beta matrix:

- `nullEffect = 'beta = 0'`: Tests the hypothesis that a slope is zero.
- `nullEffect = 'betaX = betaZ'`: Tests the hypothesis that two or more slopes are equal to each other.
- `nullEffect = 'betaA = betaB'`: Tests the hypothesis that a slope is equal in two or more groups (always assuming metric invariance).

This function provides a generic way to perform power analyses (as compared to other functions covering special cases in a more accessible manner).

A specific hypothesis is defined by setting `nullEffect` to define the hypothesis type, `nullWhich` to define the slope(s) that are targeted, and by providing the `Beta` (and optionally the `Psi`) matrix to define the population structure.

To understand the structure of `Beta` and `Psi`, consider the general structural equation model,

$$\Sigma = \Lambda(I - B)^{-1}\Psi[(I - B)^{-1}]'\Lambda' + \Theta$$

where B is the $m \cdot m$ matrix containing the regression slopes and Ψ is the (residual) variance-covariance matrix of the m factors.

As an example, suppose there are four factors (X1, X2, X3, X4), and Beta is defined as follows:

	X_1	X_2	X_3	X_4
X_1	0.0	0.0	0.0	0.0
X_2	0.0	0.0	0.0	0.0
X_3	0.2	0.3	0.0	0.0
X_4	0.3	0.5	0.0	0.0

Each row specifies how a particular factor is predicted by the available factors, so the above implies the following regression relations:

$X_1 = 0.0 \cdot X_1 + 0.0 \cdot X_2 + 0.0 \cdot X_3 + 0.0 \cdot X_4$
$X_2 = 0.0 \cdot X_1 + 0.0 \cdot X_2 + 0.0 \cdot X_3 + 0.0 \cdot X_4$

$X_3 = 0.2 \cdot X_1 + 0.3 \cdot X_2 + 0.0 \cdot X_3 + 0.0 \cdot X_4$
$X_4 = 0.3 \cdot X_1 + 0.5 \cdot X_2 + 0.0 \cdot X_3 + 0.0 \cdot X_4$

which simplifies to

$X_3 = 0.2 \cdot X_1 + 0.3 \cdot X_2$
$X_4 = 0.3 \cdot X_1 + 0.5 \cdot X_2$

Further suppose that Psi is

	X_1	X_2	X_3	X_4
X_1	1.0	0.3	0.0	0.0
X_2	0.3	1.0	0.0	0.0
X_3	0.0	0.0	1.0	0.2
X_4	0.0	0.0	0.2	1.0

which implies a correlation between X1 and X2 of .3 and a residual correlation between X3 and X4 of .2.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.
- `loadM`: Can be used instead of `Lambda`: Used in conjunction with `nIndicator`. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. `loadM = c(.5, .6)` defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either `Lambda`, or `loadings`, or `nIndicator` and `loadM` always need to be defined.

Additional arguments related to the requested type of **power analysis**:

- `alpha`: The alpha error probability. Required for `type = 'a-priori'` and `type = 'post-hoc'`.
- Either `beta` or `power`: The beta error probability and the statistical power (1 - beta), respectively. Only for `type = 'a-priori'`.
- `N`: The sample size. Always required for `type = 'post-hoc'` and `type = 'compromise'`. For `type = 'a-priori'` and multiple group analysis, `N` is a list of group weights.
- `abratio`: The ratio of alpha to beta. Only for `type = 'compromise'`.

If a **simulated power analysis** (`simulatedPower = TRUE`) is requested, optional arguments can be provided as a list to `simOptions`:

- `nReplications`: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.

- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.

modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# set up pathmodel in the form of
# f2 = .2*f1
# f3 = .3*f2
# f4 = .1*f1 + .4*f3
# obtain the required N to detect that the
# slope f1 -> f4 is >= .10
# with a power of 95% on alpha = 5%
# where f1 is measured by 3, f2 by 4, f3 by 5, and f4 by 6 indicators,
# and all loadings are .5
Beta <- matrix(c(
  c(.00, .00, .00, .00),       # f1
  c(.20, .00, .00, .00),       # f2
  c(.00, .30, .00, .00),       # f3
  c(.10, .00, .40, .00)        # f4
), byrow = TRUE, ncol = 4)
powerPath <- semPower.powerPath(type = 'a-priori',
                                Beta = Beta,
                                nullWhich = c(4, 1),
                                nIndicator = c(3, 4, 5, 6),
                                loadM = .5,
                                alpha = .05, beta = .05)
# show summary
summary(powerPath)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerPath$modelH1, sample.cov = powerPath$Sigma,
sample.nobs = powerPath$requiredN, sample.cov.rescale = FALSE)
lavaan::sem(powerPath$modelH0, sample.cov = powerPath$Sigma,
sample.nobs = powerPath$requiredN, sample.cov.rescale = FALSE)

# same as above, but detect that the slope f3 -> f4 is >= .30
powerPath <- semPower.powerPath(type = 'a-priori',
                                Beta = Beta,
                                nullWhich = c(4, 3),
                                nIndicator = c(3, 4, 5, 6),
                                loadM = .5,
                                alpha = .05, beta = .05)

# same as above, but detect that
# the slope f1 -> f2 (of .20) differs from the slope f2 -> f3 (of .30)
```

```
powerPath <- semPower.powerPath(type = 'a-priori',
                                Beta = Beta,
                                nullEffect = 'betaX = betaZ',
                                nullWhich = list(c(2, 1), c(3, 2)),
                                nIndicator = c(3, 4, 5, 6),
                                loadM = .5,
                                alpha = .05, beta = .05)

# same as above, but consider a multiple group model with equally sized groups,
# and obtain the required N to detect that the slope
# in group 1 (of .20) differs from the one in group 2 (of .40)
Beta1 <- Beta2 <- matrix(c(
  c(.00, .00, .00, .00),       # f1
  c(.20, .00, .00, .00),       # f2
  c(.00, .30, .00, .00),       # f3
  c(.10, .00, .40, .00)        # f4
), byrow = TRUE, ncol = 4)
Beta2[2, 1] <- .40
Beta <- list(Beta1, Beta2)
powerPath <- semPower.powerPath(type = 'a-priori',
                                Beta = Beta,
                                nullEffect = 'betaA = betaB',
                                nullWhich = c(2, 1),
                                nIndicator = c(3, 4, 5, 6),
                                loadM = .5,
                                alpha = .05, beta = .05, N = list(1, 1))

## End(Not run)
```

`semPower.powerPlot` *semPower.powerPlot*

Description

Shows a plot showing power as function of N given the output of a power analysis.

Usage

```
semPower.powerPlot(semPowerRes, ...)
```

Arguments

`semPowerRes`	results of a semPower analysis
`...`	other parameters passed to `semPower.powerPlot.byN()`

Value

powerplot

Examples

```
## Not run:
# perform a power analysis
powerCFA <- semPower.powerCFA(type = 'post-hoc', alpha = .05, N = 300,
                              Phi = .15, nIndicator = c(5, 4), loadM = c(.5, .6))
# show plot
semPower.powerPlot(powerCFA)

## End(Not run)
```

`semPower.powerPlot.byEffect`

semPower.powerPlot.byEffect

Description

Shows a plot showing power as function of the effect for a given N and alpha.

Usage

```
semPower.powerPlot.byEffect(
  effect.measure = NULL,
  alpha,
  N,
  df,
  p = NULL,
  effect.min = NULL,
  effect.max = NULL,
  steps = 50,
  linewidth = 1
)
```

Arguments

`effect.measure`	type of effect, one of `"F0"`, `"RMSEA"`, `"Mc"`, `"GFI"`, `"AGFI"`
`alpha`	alpha error
`N`	the number of observations
`df`	the model degrees of freedom
`p`	the number of observed variables, required for `effect.measure = "GFI"` and `effect.measure = "AGFI"`
`effect.min`	minimum effect
`effect.max`	maximum effect
`steps`	number of steps
`linewidth`	linewidth

Value

powerplot

Examples

```
## Not run:
semPower.powerPlot.byEffect(effect.measure = "RMSEA", alpha = .05,
                            N = 500, effect.min = .01, effect.max = .15, df = 200)

## End(Not run)
```

`semPower.powerPlot.byN`

semPower.powerPlot.byN

Description

Shows a plot showing power as function of N for a given effect and alpha.

Usage

```
semPower.powerPlot.byN(
  effect = NULL,
  effect.measure = NULL,
  alpha,
  df,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  power.min = alpha,
  power.max = 0.99,
  steps = 50,
  linewidth = 1
)
```

Arguments

`effect`	effect size specifying the discrepancy between H0 and H1
`effect.measure`	type of effect, one of `"F0"`, `"RMSEA"`, `"Mc"`, `"GFI"`, `"AGFI"`
`alpha`	alpha error
`df`	the model degrees of freedom
`p`	the number of observed variables, required for `effect.measure = "GFI"` and `effect.measure = "AGFI"`
`SigmaHat`	model implied covariance matrix. Use in conjunction with `Sigma`` to define effectandeffect.measure‘.

Sigma	population covariance matrix. Use in conjunction with SigmaHat to define effect and effect.measure.
power.min	minimum power, must not be smaller than alpha.
power.max	maximum power
steps	number of steps
linewidth	linewidth

Value

powerplot

Examples

```
## Not run:
semPower.powerPlot.byN(effect = .05, effect.measure = "RMSEA",
                       alpha = .05, power.min = .05, power.max = .99, df = 200)

## End(Not run)
```

---

semPower.powerRegression

semPower.powerRegression

Description

Convenience function for performing power analysis on slope(s) in a latent regression of the form Y = XB. This requires the lavaan package.

Usage

```
semPower.powerRegression(
  type,
  comparison = "restricted",
  slopes = NULL,
  corXX = NULL,
  corYX = NULL,
  R2base = NULL,
  R2inc = NULL,
  nullEffect = "slope = 0",
  nullWhich = NULL,
  nullWhichGroups = NULL,
  standardized = TRUE,
  ...
)
```

Arguments

type	type of power analysis, one of 'a-priori', 'post-hoc', 'compromise'.
comparison	comparison model, one of 'saturated' or 'restricted' (the default). This determines the df for power analyses. 'saturated' provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. 'restricted' provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in nullEffect, so the df equal the number of restrictions.
slopes	vector of slopes (or a single number for a single slope) of the k predictors for Y. A list of slopes for multigroup models.
corXX	correlation(s) between the k predictors (X). Either NULL for uncorrelated predictors, a single number (for k = 2 predictors), or a matrix. Can also be a list for multigroup models providing the correlations by group of matrices (otherwise, the same correlations are used in all groups).
corYX	can be provided instead of slopes. Correlation(s) between the k predictors (X) and the criterion (Y). A list for multigroup models.
R2base	can be provided instead of slopes, to be used in conjunction with R2inc. Multiple R-squared yielded by a set of k1 predictors (X). A list for multigroup models.
R2inc	can be provided instead of slopes, to be used in conjunction with R2base. Incremental multiple R-squared yielded by a set of k2 predictors over the k1 predictors (so that R^2 = R_base^2 + R_inc^2) . A list for multigroup models.
nullEffect	defines the hypothesis of interest, must be one of 'slope = 0' (the default) to test whether a slope is zero, 'slopeX = slopeZ' to test for the equality of slopes, 'slopeA = slopeB' to test for the equality of slopes across groups, or 'R2inc = 0' to test whether the incremental variance explained is zero, or 'R2incA = R2incB' to test whether the incremental variance explained is equal across groups. Define the slopes to set to equality or the relevant predictors in nullWhich.
nullWhich	single number indicating which slope is hypothesized to equal zero when nullEffect = 'slope = 0', or indicating which slope to restrict to equality across groups when nullEffect = 'slopeA = slopeB', or vector defining the slopes to restrict to equality when nullEffect = 'slopeX = slopeZ' (can also contain more than two slopes, e.g. c(1, 2, 3) to constrain the first three slopes to equality), or vector defining the k additional predictors yielding the incremental R-squared when nullEffect = 'R2inc = 0' or nullEffect = 'R2incA = R2incB' (e.g. c(2, 3) to test the incremental variance explained by the including the second and third predictor in addition to the first predictors). In the latter case, the maximum value in nullWhich defines the total number of predictors, i.e. when there are 5 predictors in total, nullWhich must include predictor 5.
nullWhichGroups	for nullEffect = 'slopeA = slopeB' or nullEffect = 'r2IncA = r2IncB', vector indicating the groups for which equality constrains should be applied, e.g. c(1, 3) to constrain the relevant parameters of the first and the third group. If NULL, all groups are constrained to equality.
standardized	whether all parameters should be standardized (TRUE, the default). If FALSE, all regression relations are unstandardized.

... mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. The first factor is treated as Y and the subsequent factors as the predictors X_k. See details.

Details

This function performs a power analysis to reject various hypotheses arising in SEM models involving a simple regression relation of the form `Y = b_1*X_1 + ... + b_k*X_k` between the factors:

- `nullEffect = 'slope = 0'`: Tests the hypothesis that the slope for a predictor is zero.
- `nullEffect = 'slopeX = slopeZ'`: Tests the hypothesis that two or more slopes are equal to each other.
- `nullEffect = 'slopeA = slopeB'`: Tests the hypothesis that the slope for a predictor is equal in two or more groups (always assuming metric invariance).
- `nullEffect = 'R2inc = 0'`: Tests the hypothesis that the increase in the variance by one or more predictors explained is zero.
- `nullEffect = 'R2incA = R2incB'`: Tests the hypothesis that the increase in the variance explained by one or more predictors is equal in two or more groups (always assuming metric invariance).

For hypotheses regarding mediation effects, see `semPower.powerMediation()`. For hypothesis in autoregressive models, see `semPower.powerAutoreg()`.

When the regression structure is defined in terms of `corYX` (rather than providing the individual slopes via `slopes`), the implied slopes are computed according to $b = R_{xx}^{-1} r_{yx}$.

The purpose of `nullEffect = 'R2inc = 0'` and `nullEffect = 'R2incA = R2incB'` is to state the effect in terms of variance explained rather than in terms of individual values of the regression slopes. The H0 model also constrains the respective slopes to zero (or equality across groups), so that the same tests is performed as the ones involving slopes. In this case, slopes are computed that satisfy the specified baseline and incremental R-squared values by solving

$$b = R_{xx}^{-1} r_{yx} R^2 = \sum b \times r_{yx}$$

Note that this is only approximate (but still accurate to about 5 decimal places).

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling the number of factors).
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure, e. g. `loadings = list(c(.5, .4, .6), c(.8, .6, .6, .4))` defines a two factor model with three indicators loading on the first factor by .5, , 4., and .6, and four indicators loading on the second factor by .8, .6, .6, and .4.
- `nIndicator`: Can be used instead of `Lambda`: Used in conjunction with `loadM`. Defines the number of indicators by factor, e. g., `nIndicator = c(3, 4)` defines a two factor model with three and four indicators for the first and second factor, respectively. `nIndicator` can also be a single number to define the same number of indicators for each factor.

- loadM: Can be used instead of Lambda: Used in conjunction with nIndicator. Defines the loading either for all indicators (if a single number is provided) or separately for each factor (if a vector is provided), e. g. loadM = c(.5, .6) defines the loadings of the first factor to equal .5 and those of the second factor do equal .6.

So either Lambda, or loadings, or nIndicator and loadM need to be defined. If the model contains observed variables only, use Lambda = diag(x) where x is the number of variables.

Note that the first factor acts as the criterion Y, the subsequent factors as predictors X_1 to X_k.

Additional arguments related to the requested type of **power analysis**:

- alpha: The alpha error probability. Required for type = 'a-priori' and type = 'post-hoc'.
- Either beta or power: The beta error probability and the statistical power (1 - beta), respectively. Only for type = 'a-priori'.
- N: The sample size. Always required for type = 'post-hoc' and type = 'compromise'. For type = 'a-priori' and multiple group analysis, N is a list of group weights.
- abratio: The ratio of alpha to beta. Only for type = 'compromise'.

If a **simulated power analysis** (simulatedPower = TRUE) is requested, optional arguments can be provided as a list to simOptions:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness

must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the `mnonr` package.

`type = 'RK'` implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (`distributions`). `distributions` must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. `rchisq`) and additional arguments (`list(df = 2)`).

`type = 'VM'` implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (`skewness`) and excess kurtosis (`kurtosis`) for each variable are provided as vectors.

Value

a list. Use the `summary` method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

`Sigma`	the population covariance matrix. A list for multiple group models.
`mu`	the population mean vector or `NULL` when no meanstructure is involved. A list for multiple group models.
`SigmaHat`	the H0 model implied covariance matrix. A list for multiple group models.
`muHat`	the H0 model implied mean vector or `NULL` when no meanstructure is involved. A list for multiple group models.
`modelH0`	`lavaan` H0 model string.
`modelH1`	`lavaan` H1 model string or `NULL` when the comparison refers to the saturated model.
`simRes`	detailed simulation results when a simulated power analysis (`simulatedPower = TRUE`) was performed.

See Also

`semPower.genSigma()` `semPower.aPriori()` `semPower.postHoc()` `semPower.compromise()`

Examples

```
## Not run:
# latent regression of the form `Y = .2*X1 + .3*X2`, where X1 and X2 correlate by .4
# obtain required N to reject the hypothesis that the slope of X1 is zero
# with a power of 95% on alpha = 5%,
# where Y is measured by 3 indicators loading by .5 each,
# X1 by 5 indicators loading by .6 each, and
# X2 by 4 indicators loading by .7 each.
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  slopes = c(.2, .3),         # b1, b2
  corXX = .4,
  nullWhich = 1,              # b1
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2
```

```
# show summary
summary(powerReg)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerReg$modelH1, sample.cov = powerReg$Sigma,
  sample.nobs = powerReg$requiredN, sample.cov.rescale = FALSE)
lavaan::sem(powerReg$modelH0, sample.cov = powerReg$Sigma,
  sample.nobs = powerReg$requiredN, sample.cov.rescale = FALSE)

# same as above, but determine power with N = 500 on alpha = .05
powerReg <- semPower.powerRegression(
  type = 'post-hoc',alpha = .05, N = 500,
  slopes = c(.2, .3),         # b1, b2
  corXX = .4,
  nullWhich = 1,              # b1
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2

# same as above, but determine the critical chi-square with N = 500 so that alpha = beta
powerReg <- semPower.powerRegression(
  type = 'compromise', abratio = .05, N = 500,
  slopes = c(.2, .3),         # b1, b2
  corXX = .4,
  nullWhich = 1,              # b1
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2

# same as above, but ask for the required N to detect that the slope of X2 is zero
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  slopes = c(.2, .3),         # b1, b2
  corXX = .4,
  nullWhich = 2,              # b2
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2

# same as above, but define unstandardized slopes
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  slopes = c(.2, .3),         # b1, b2
  standardized = FALSE,
  corXX = .4,
  nullWhich = 2,              # b2
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2

# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  comparison = 'saturated',
  slopes = c(.2, .3),         # b1, b2
  corXX = .4,
  nullWhich = 2,              # b2
```

```
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2

# same as above, but provide a reduced loading matrix defining
# three indicators with loadings of .7, .6, .5 on the first factor (Y),
# four indicators with loadings of .5, .6, .4, .8 on the second factor (X1), and
# three indicators with loadings of .8, .7, .8 on the third factor (X2).
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  comparison = 'saturated',
  slopes = c(.2, .3),         # b1, b2
  corXX = .4,
  nullWhich = 2,              # b2
  loadings = list(
    c(.7, .6, .5),            # Y
    c(.5, .6, .4, .8),        # X1
    c(.8, .7, .8)             # X2
  ))

# same as above, but define effect in terms
# of predictor-criterion correlations instead of slopes
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  comparison = 'saturated',
  corYX = c(.32, .38),        # r_Y,X1, r_Y,X2
  corXX = .4,                 # r_X1,X2
  nullWhich = 2,              # b2
  nIndicator = c(3, 5, 4),    # Y, X1, X2
  loadM = c(.5, .6, .7))      # Y, X1, X2

# latent regression of the form `Y = .2*X1 + .3*X2 + .4*X3`,
# providing the predictor intercorrelation matrix,
# and ask for the required N to detect that the first slope differs from zero.
corXX <- matrix(c(
  #   X1    X2    X3
  c(1.00, 0.20, 0.30),  # X1
  c(0.20, 1.00, 0.10),  # X2
  c(0.30, 0.10, 1.00)   # X3
), ncol = 3,byrow = TRUE)
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  slopes = c(.2, .3, .4),     # b1, b2, b3
  corXX = corXX,
  nullWhich = 1,              # b1
  nIndicator = c(4, 3, 5, 4), # Y, X1, X2, X3
  loadM = c(.5, .5, .6, .7))  # Y, X1, X2, X3

# same as above, but ask for the required N to detect that
# the slope for X1 (b = .2) and the slope for X2 (b = .3) differ from each other
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  slopes = c(.2, .3, .4),     # b1, b2, b3
  corXX = corXX,
```

```
  nullEffect = 'slopeX = slopeZ',
  nullWhich = c(1, 2),        # b1 = b2
  nIndicator = c(4, 3, 5, 4), # Y, X1, X2, X3
  loadM = c(.5, .5, .6, .7))  # Y, X1, X2, X3

# same as above, but ask for the required N to reject the hypothesis that
# all three slopes are equal to each other
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  slopes = c(.2, .3, .4),     # b1, b2, b3
  corXX = corXX,
  nullEffect = 'slopeX = slopeZ',
  nullWhich = c(1, 2, 3),     # b1 = b2 = b2
  nIndicator = c(4, 3, 5, 4), # Y, X1, X2, X3
  loadM = c(.5, .5, .6, .7))  # Y, X1, X2, X3


# get required N to detect that
# the slope for X2 group 1 (of b2 = .3) differs from the slope for X2 in group 2 (of b = .0).
# The remaining slopes are equal in both groups (b1 = .2, b3 = .4).
# The measurement model is identical in both groups:
# The criterion (Y) is measured by 4 indicators loading by .5 each,
# Predictors X1 and X3 are both measured by 5 indicators loading by .6 each,
# Predictor X2 is measured by 3 indicators loading by .7 each.
# Both groups are sized equally (N = list(1, 1)).
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, beta = .05,
  N = list(1, 1),             # group weights
  slopes = list(
    c(.2, .3, .4),            # group 1: b1, b2, b3
    c(.2, .0, .4)             # group 2: b1, b2, b3
    ),
  corXX = corXX,
  nullEffect = 'slopeA = slopeB',
  nullWhich = 2,              # g1b2 = g2b2
  nIndicator = c(4, 5, 3, 5),  # Y, X1, X2, X3 (same across groups)
  loadM = c(.5, .6, .7, .6))   # Y, X1, X2, X3 (same across groups)


# determine required N to detect that the third predictor incrementally
# accounts for 10% of the variance explained in the criterion
# above predictors 1 and 2 which account for 5% of the variance.
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, power = .80,
  R2base =  .05,
  R2inc =  .1,
  corXX = corXX,
  nullEffect = 'R2inc = 0',
  nullWhich = c(3),             # b3
  nIndicator = c(4, 3, 5, 4),    # Y, X1, X2, X3
  loadM = c(.5, .5, .6, .7)      # Y, X1, X2, X3
)
```

```
# same as above, but determine required N to detect that
# all three predictors jointly account for 15% of the variance explained
# in the criterion
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, power = .80,
  R2base =  0,
  R2inc =  .15,
  corXX = corXX,
  nullEffect = 'R2inc = 0',
  nullWhich = c(1, 2, 3),      # b1, b2, b3
  nIndicator = c(4, 3, 5, 4),  # Y, X1, X2, X3
  loadM = c(.5, .5, .6, .7)    # Y, X1, X2, X3
)


# determine required N to detect that the incremental variance
# explained by the third predictor in group 1 of .025 differs from
# that in group 2 of .10.
powerReg <- semPower.powerRegression(
  type = 'a-priori', alpha = .05, power = .80,
  N = list(1, 1),
  R2base =  list(.05, .05),    # group1, group2
  R2inc =  list(.025, .10),    # group1, group2
  corXX = corXX,
  nullEffect = 'R2incA = R2incB',
  nullWhich = c(3),            # b3
  nIndicator = c(4, 3, 5, 4),  # Y, X1, X2, X3
  loadM = c(.5, .5, .6, .7)    # Y, X1, X2, X3
)


# request a simulated post-hoc power analysis with 500 replications
# to detect that the slope of X1 differs from zero.
set.seed(300121)
powerReg <- semPower.powerRegression(
  type = 'post-hoc', alpha = .05, N = 500,
  slopes = c(.2, .1),
  nullWhich = 1,
  nIndicator = c(4, 3, 3),
  loadM = .5,
  simulatedPower = TRUE,
  simOptions = list(
    nReplications = 500
    )
)

## End(Not run)
```

---

semPower.powerRICLPM *semPower.powerRICLPM*

Description

Convenience function for performing power analysis on effects in a random intercept cross-lagged panel model (RI-CLPM). This requires the lavaan package.

Usage

```
semPower.powerRICLPM(
  type,
  comparison = "restricted",
  nWaves = NULL,
  autoregEffects = NULL,
  crossedEffects = NULL,
  rXY = NULL,
  rBXBY = NULL,
  waveEqual = NULL,
  nullEffect = NULL,
  nullWhichGroups = NULL,
  nullWhich = NULL,
  standardized = TRUE,
  metricInvariance = TRUE,
  autocorResiduals = TRUE,
  ...
)
```

Arguments

`type`	type of power analysis, one of `'a-priori'`, `'post-hoc'`, `'compromise'`.
`comparison`	comparison model, one of `'saturated'` or `'restricted'` (the default). This determines the df for power analyses. `'saturated'` provides power to reject the model when compared to the saturated model, so the df equal the one of the hypothesized model. `'restricted'` provides power to reject the hypothesized model when compared to an otherwise identical model that just omits the restrictions defined in `nullEffect`, so the df equal the number of restrictions.
`nWaves`	number of waves, must be >= 3.
`autoregEffects`	vector of the autoregressive effects of X and Y (constant across waves), or a list of vectors of autoregressive effects for X and Y from wave to wave, e.g. `list(c(.7, .6), c(.5, .5))` for an autoregressive effect of .7 for X1->X2 and .6 for X2->X3 and autoregressive effects of .5 for Y1->Y2 and Y2->Y3. Must be a list of lists for multiple groups models. If the list structure is omitted, no group differences are assumed.
`crossedEffects`	vector of crossed effects of X on Y (X -> Y) and vice versa (both constant across waves), or a list of vectors of crossed effects giving the crossed effect of X on Y (and vice versa) for each wave, e.g. `list(c(.2, .3), c(.1, .1))` for X1->Y2 = .2, X2->Y3 = .3, Y1->Y2 = .1, and Y2->Y3 = .1. Must be a list of lists for multiple groups models. If the list structure is omitted, no group differences are assumed.

`rXY` vector of (residual-)correlations between X and Y for each wave. If `NULL`, all (residual-)correlations are zero. Can be a list for multiple groups models, otherwise no group differences are assumed.

`rBXBY` correlation between random intercept factors. If `NULL`, the correlation is zero. Must be a list of lists for multiple groups models. If the list structure is omitted, no group differences are assumed.

`waveEqual` parameters that are assumed to be equal across waves in both the H0 and the H1 model. Valid are `'autoregX'` and `'autoregY'` for autoregressive effects, `'crossedX'` and `'crossedY'` for crossed effects, `'corXY'` for residual correlations, or `NULL` for none (so that all parameters are freely estimated, subject to the constraints defined in `nullEffect`).

`nullEffect` defines the hypothesis of interest. Valid are the same arguments as in `waveEqual` and additionally `'autoregX = 0'`, `'autoregY = 0'`, `'crossedX = 0'`, `'crossedY = 0'` to constrain the X or Y autoregressive effects or the crossed effects to zero, `'corBXBY = 0'` to constrain the correlation between the random intercepts to zero, and `'autoregX = autoregY'` and `'crossedX = crossedY'` to constrain them to be equal for X and Y, and `'autoregXA = autoregXB'`, `'autoregYA = autoregYB'`, `'crossedXA = crossedXB'`, `'crossedYA = crossedYB'`, and `corBXBYA = corBXBYB` to constrain them to be equal across groups.

`nullWhichGroups` for hypothesis involving cross-groups comparisons, vector indicating the groups for which equality constrains should be applied, e.g. `c(1, 3)` to constrain the relevant parameters of the first and the third group. If `NULL`, all groups are constrained to equality.

`nullWhich` used in conjunction with `nullEffect` to identify which parameter to constrain when there are > 2 waves and parameters are not constant across waves. For example, `nullEffect = 'autoregX = 0'` with `nullWhich = 2` would constrain the second autoregressive effect for X to zero.

`standardized` whether the autoregressive and cross-lagged parameters should be treated as standardized (`TRUE`, the default), implying that unstandardized and standardized regression relations have the same value. If `FALSE`, all regression relations are unstandardized.

`metricInvariance` whether metric invariance over waves is assumed (`TRUE`, the default) or not (`FALSE`). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model, where the df are not affected by invariance constraints).

`autocorResiduals` whether the residuals of the indicators of latent variables are autocorrelated over waves (`TRUE`, the default) or not (`FALSE`). This affects the df when the comparison model is the saturated model and generally affects power (also for comparisons to the restricted model).

`...` mandatory further parameters related to the specific type of power analysis requested, see `semPower.aPriori()`, `semPower.postHoc()`, and `semPower.compromise()`, and parameters specifying the factor model. The order of factors is (X1, Y1, X2, Y2, ..., X_nWaves, Y_nWaves). See details.

Details

This function performs a power analysis to reject various hypotheses arising in a random intercept crossed-lagged panel model (RI-CLPM). In a standard RI-CLPM implemented here, two variables X and Y are repeatedly assessed at three or more different time points (`nWaves`), yielding autoregressive effects (`X1 -> X2`, `X2 -> X3`, `Y1 -> Y2`, `Y2 -> Y3`), synchronous effects (`X1 <-> Y1`, `X2 <-> Y2`, `X3 <-> Y3`), and cross-lagged effects (`X1 -> Y2`, `X2 -> Y3`, `Y1 -> X2`, `Y2 -> X3`). RI-CLPMs are typically implemented assuming that the parameters are constant across waves (`waveEqual`), and usually omit lag-2 effects (e.g., `X1 -> Y3`). RI-CLPMs based on latent factors usually assume at least metric invariance of the factors over waves (`metricInvariance`).

Relevant hypotheses in arising in a RI-CLPM are:

- `autoregX = 0` and `autoregY = 0`: Tests the hypothesis that the autoregressive effect of X and Y, respectively, is zero.
- `crossedX = 0` and `crossedY = 0`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) and Y on X (`crossedY`), respectively, is zero.
- `autoregX = autoregY`: Tests the hypothesis that the autoregressive effect of X and Y are equal.
- `crossedX = crossedY`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) and Y on X (`crossedY`) are equal.
- `autoregX` and `autoregY`: Tests the hypothesis that the autoregressive effect of X and Y, respectively, is equal across waves.
- `crossedX` and `crossedY`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) and Y on X (`crossedY`), respectively, is equal across waves.
- `corXY`: Tests the hypothesis that the (residual-)correlations between X and Y are equal across waves.
- `corBXBY = 0`: Tests the hypothesis that the correlation between the random intercept factors of X and Y is zero.
- `autoregXA = autoregXB` and `autoregYA = autoregYB`: Tests the hypothesis that the autoregressive effect of either X or Y are equal across groups.
- `crossedXA = crossedXB` and `crossedYA = crossedYB`: Tests the hypothesis that the crossed effect of X on Y (`crossedX`) or of Y on X (`crossedY`), respectively, is equal across groups.
- `corBXBYA = corBXBYB`: Tests the hypothesis that the correlation between the random intercept factors is equal across groups.

For hypotheses regarding the traditional CLPM, see `semPower.powerCLPM()`.

Beyond the arguments explicitly contained in the function call, additional arguments are required specifying the factor model and the requested type of power analysis.

Additional arguments related to the **definition of the factor model**:

- `Lambda`: The factor loading matrix (with the number of columns equaling 2 times the number of waves). Columns should be in order X1, Y1, X2, Y2, ..., X_nWaves, Y_nWaves.
- `loadings`: Can be used instead of `Lambda`: Defines the primary loadings for each factor in a list structure ordered by wave, e.g., list(c(.2, .2, .2), c(.4, .4, .4, .4), c(.2, .2, .2), c(.4, .4, .4, .4), c(.2, .2, .2), c(.4, .4, .4, .4)) defines loadings of .2 for the three indicators of X at waves 1-3 and loadings of .4 for the four indicators of Y at waves 1-3. Must not contain secondary loadings.

- nIndicator: Can be used instead of Lambda: Used in conjunction with loadM. Defines the number of indicators for each factor ordered by wave, e.g. c(3, 4, 3, 4, 3, 4) defines three indicators for X at waves 1-3 and four indicators for Y at waves 1-3.
- loadM: Can be used instead of Lambda: Used in conjunction with nIndicator. Defines the loading either for all indicators (if a single number is provided) or separately for each factor at each wave (if a vector is provided), e. g. loadM = c(.5, .6, .5, .6, .5, .6) defines mean loadings of .5 for X at waves 1-3 and mean loadings of .6 for Y at waves 1-3.

So either Lambda, or loadings, or nIndicator and loadM need to be defined. If the model contains observed variables only, use Lambda = diag(x) where x is the number of variables.

Note that the order of the factors is (X1, Y1, X2, Y2, ..., X_nWaves, Y_nWaves), i. e., the first factor is treated as the first measurement of X, the second as the first measurement of Y, the third as the second measurement of X, etc..

Additional arguments related to the requested type of **power analysis**:

- alpha: The alpha error probability. Required for type = 'a-priori' and type = 'post-hoc'.
- Either beta or power: The beta error probability and the statistical power (1 - beta), respectively. Only for type = 'a-priori'.
- N: The sample size. Always required for type = 'post-hoc' and type = 'compromise'. For type = 'a-priori' and multiple group analysis, N is a list of group weights.
- abratio: The ratio of alpha to beta. Only for type = 'compromise'.

If a **simulated power analysis** (simulatedPower = TRUE) is requested, optional arguments can be provided as a list to simOptions:

- nReplications: The targeted number of simulation runs. Defaults to 250, but larger numbers greatly improve accuracy at the expense of increased computation time.
- minConvergenceRate: The minimum convergence rate required, defaults to .5. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RC', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of MCAR (the default), MAR, or NMAR.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy used in parallel processing (when nCores > 1). Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid options.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Value

a list. Use the summary method to obtain formatted results. Beyond the results of the power analysis and a number of effect size measures, the list contains the following components:

Sigma	the population covariance matrix. A list for multiple group models.
mu	the population mean vector or NULL when no meanstructure is involved. A list for multiple group models.
SigmaHat	the H0 model implied covariance matrix. A list for multiple group models.
muHat	the H0 model implied mean vector or NULL when no meanstructure is involved. A list for multiple group models.
modelH0	lavaan H0 model string.
modelH1	lavaan H1 model string or NULL when the comparison refers to the saturated model.
simRes	detailed simulation results when a simulated power analysis (simulatedPower = TRUE) was performed.

See Also

semPower.genSigma() semPower.aPriori() semPower.postHoc() semPower.compromise()

Examples

```
## Not run:
# Determine required N in a 3-wave RI-CLPM
# to detect crossed effects of X (X1 -> Y2 and X2 -> Y3) of >= .2
# with a power of 95% on alpha = 5%, where
# X1, X2, and X3 are measured by 5 indicators loading by .5 each, and
# Y1, Y2, and Y3 are measured by 3 indicators loading by .4 each, and
# there is no synchronous correlation between X and Y (rXY = NULL),
```

```
# the correlation between the random intercept factors of X and Y (rBXBY) is .1,
# the autoregressive effects of X are .8 (equal across waves),
# the autoregressive effects of Y are .7 (equal across waves), and
# the crossed effects of Y (Y1 -> X2 and Y2 -> X3) are .1 (equal across waves).

powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)

# show summary
summary(powerRICLPM)
# optionally use lavaan to verify the model was set-up as intended
lavaan::sem(powerRICLPM$modelH1, sample.cov = powerRICLPM$Sigma,
            sample.nobs = powerRICLPM$requiredN, sample.cov.rescale = FALSE)
lavaan::sem(powerRICLPM$modelH0, sample.cov = powerRICLPM$Sigma,
            sample.nobs = powerRICLPM$requiredN, sample.cov.rescale = FALSE)


# same as above, but determine power with N = 500 on alpha = .05
powerRICLPM <- semPower.powerRICLPM(type = 'post-hoc',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, N = 500)


# same as above, but determine the critical chi-square with N = 500 so that alpha = beta
powerRICLPM <- semPower.powerRICLPM(type = 'compromise',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
```

```
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    abratio = 1, N = 500)


# same as above, but compare to the saturated model
# (rather than to the less restricted model)
powerRICLPM <- semPower.powerRICLPM(type = 'compromise',
                                    comparison = 'saturated',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    abratio = 1, N = 500)


# same as above, but assume only observed variables
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    Lambda = diag(6),
                                    alpha = .05, beta = .05)


# same as above, but provide reduced loadings matrix to define that
# X1, X2, and X3 are measured by 5 indicators each loading by .5, .4, .5, .4, .3
# Y1, Y2, and Y3 are measured by 3 indicators each loading by .4, .3, .2
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    loadings = list(
                                      c(.5, .4, .5, .4, .3),    # X1
                                      c(.4, .3, .2),            # Y1
                                      c(.5, .4, .5, .4, .3),    # X2
                                      c(.4, .3, .2),            # Y2
                                      c(.5, .4, .5, .4, .3),    # X3
```

```
                                    c(.4, .3, .2)             # Y3
                                  ),
                                  alpha = .05, beta = .05)


# same as above, but do not assume metric invariance across waves
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                  nWaves = 3,
                                  autoregEffects = c(.8, .7),
                                  crossedEffects = c(.2, .1),
                                  waveEqual = c('autoregX', 'autoregY',
                                                'crossedX', 'crossedY'),
                                  rXY = NULL,
                                  rBXBY = .1,
                                  nullEffect = 'crossedX = 0',
                                  nIndicator = c(5, 3, 5, 3, 5, 3),
                                  loadM = c(.5, .4, .5, .4, .5, .4),
                                  metricInvariance = FALSE,
                                  alpha = .05, beta = .05)


# same as above, but determine N to detect that the crossed effect of Y
# (Y1 -> X2 and Y2 -> X3) is >= .1.
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                  nWaves = 3,
                                  autoregEffects = c(.8, .7),
                                  crossedEffects = c(.2, .1),
                                  waveEqual = c('autoregX', 'autoregY',
                                                'crossedX', 'crossedY'),
                                  rXY = NULL,
                                  rBXBY = .1,
                                  nullEffect = 'crossedY = 0',
                                  nIndicator = c(5, 3, 5, 3, 5, 3),
                                  loadM = c(.5, .4, .5, .4, .5, .4),
                                  alpha = .05, beta = .05)


# same as above, but determine N to detect that the autoregressive effect
# of X (X1 -> X2 and X2 -> X3) is >= .8.
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                  nWaves = 3,
                                  autoregEffects = c(.8, .7),
                                  crossedEffects = c(.2, .1),
                                  waveEqual = c('autoregX', 'autoregY',
                                                'crossedX', 'crossedY'),
                                  rXY = NULL,
                                  rBXBY = .1,
                                  nullEffect = 'autoregX = 0',
                                  nIndicator = c(5, 3, 5, 3, 5, 3),
                                  loadM = c(.5, .4, .5, .4, .5, .4),
                                  alpha = .05, beta = .05)
```

```
# same as above, but determine N to detect that the autoregressive effect
# of Y (Y1 -> Y2) is >= .7.
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'autoregY = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but determine N to detect that
# the crossed effect of X (X1 -> Y2) of .2 differs from
# the crossed effect of Y (Y1 -> X2) of .1
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = crossedY',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but determine N to detect that
# the autoregressive effect of X (X1 -> X2) of .8 differs from
# the autoregressive effect of Y (Y1 -> Y2) of .7
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'autoregX = autoregY',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but determine N to detect that the correlation between the
# random intercept factors is >= .1
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
```

```
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = .1,
                                    nullEffect = 'corBXBY = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but assume that the synchronous (residual-)correlations between
#  X and Y are equal across waves,
# namely a synchronous correlation of .05 at the first wave and residual correlations
# of .05 at the second and third wave,
# and determine N to detect a crossed effect of X (X1 -> Y2 and X2 -> Y3) of >= .2
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY',
                                                  'corXY'),
                                    rXY = c(.05, .05, .05),
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but assume that the synchronous correlation between X and Y
# is .3 at the first wave, and the respective residual correlations are .2 at
# the second wave and .3 at the third wave,
# and determine N to detect that the synchronous residual correlation at wave 2 is => .2.
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = c(.3, .2, .3),
                                    rBXBY = .1,
                                    nullEffect = 'corXY = 0',
                                    nullWhich = 2,
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# Determine required N in a 3-wave RI-CLPM to detect that
```

```
# the crossed effect of X at wave 1 (X1 -> Y2) of .20 is equal to the
# the crossed effect of X at wave 2 (X2 -> Y3) of .05
# with a power of 95% on alpha = 5%, where
# the autoregressive effects of X and Y are equal over waves,
# X1, X2, and X3 are measured by 5 indicators loading by .5 each, and
# Y1, Y2, and Y3 are measured by 3 indicators loading by .4 each, and
# the synchronous correlation between X and Y are .2, .3, and .4 at the first,
# second, and third wave,
# the correlation between the random intercept factors of X and Y is .1, and
# the autoregressive effect of X is .8 across all three waves,
# the autoregressive effect of Y is .7 across all three waves, and
# the crossed effects of Y (Y1 -> X2, and Y2 -> Y3) are both .1
# (but freely estimated for each wave).
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = list(
                                      # X   Y
                                      c(.20, .10),  # wave 1 -> wave 2
                                      c(.05, .10)), # wave 2 -> wave 3
                                    waveEqual = c('autoregX', 'autoregY'),
                                    rXY = c(.2, .3, .4),
                                    rBXBY = .1,
                                    nullEffect = 'crossedX',
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but determine N to detect that
# the crossed effect of X at wave 2 is >= .05.
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = list(
                                      # X   Y
                                      c(.20, .10),  # wave 1 -> wave 2
                                      c(.05, .10)), # wave 2 -> wave 3
                                    waveEqual = c('autoregX', 'autoregY'),
                                    rXY = c(.2, .3, .4),
                                    rBXBY = .1,
                                    nullEffect = 'crossedX = 0',
                                    nullWhich = 2,
                                    nIndicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# same as above, but determine N to detect that
# the residual correlation between X and Y at wave 2 (of .3) differs from
# the residual correlation between X and Y at wave 3 (of .4).
powerRICLPM <- semPower.powerRICLPM(type = 'a-priori',
                                    nWaves = 3,
```

```
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = list(
                                      # X   Y
                                      c(.20, .10),  # wave 1 -> wave 2
                                      c(.05, .10)), # wave 2 -> wave 3
                                    waveEqual = c('autoregX', 'autoregY'),
                                    rXY = c(.2, .3, .4),
                                    rBXBY = .1,
                                    nullEffect = 'corXY',
                                    nInindicator = c(5, 3, 5, 3, 5, 3),
                                    loadM = c(.5, .4, .5, .4, .5, .4),
                                    alpha = .05, beta = .05)


# multigroup example
# Determine the achieved power N in a 3-wave RI-CLPM to detect that
# the crossed effect of X at wave 1 (X1 -> Y2) in group 1 of .25 differs
# from the crossed effect of X at wave 1 (X1 -> Y2) in group 2 of .15,
# where both groups comprise 500 observations and alpha = 5%, and
# the measurement model is equal for both groups, and
# the crossed effects of X (X1 -> Y2, and X2 -> Y3) are .25 and .10 in the first group,
# the crossed effects of X (X1 -> Y2, and X2 -> Y3) are .15 and .05 in the second group,
# the crossed effects of Y (Y1 -> X2, and Y2 -> X3) are .05 and .15 in the first group,
# the crossed effects of Y (Y1 -> X2, and Y2 -> X3) are .01 and .10 in the second group, and
# the autoregressive effects of X (of .5) and Y (of .4) are equal over waves and over groups
# (but freely estimated in each group).
powerRICLPM <- semPower.powerRICLPM(type = 'post-hoc', alpha = .05, N = list(500, 500),
                                    nWaves = 3,
                                    autoregEffects = c(.5, .4), # group and wave constant
                                    crossedEffects = list(
                                      # group 1
                                      list(
                                        c(.25, .10),   # X
                                        c(.05, .15)    # Y
                                      ),
                                      # group 2
                                      list(
                                        c(.15, .05),   # X
                                        c(.01, .10)    # Y
                                      )
                                    ),
                                    rXY = NULL,        # identity
                                    rBXBY = NULL,      # identity
                                    nullEffect = 'crossedXA = crossedXB',
                                    nullWhich = 1,
                                    nInindicator = rep(3, 6),
                                    loadM = c(.5, .6, .5, .6, .5, .6),
                                    metricInvariance = TRUE,
                                    waveEqual = c('autoregX', 'autoregY')
                                    )


# Request a simulated post-hoc power analysis with 500 replications
```

```
# to detect crossed effects of X (X1 -> Y2 and X2 -> Y3) of >= .2
# with a power of 95% on alpha = 5% in a RI-CLPM with 3 waves,
# where there are only observed variables and
# there is no synchronous correlation between X and Y (rXY = NULL),
# and no correlation between the random intercept factors of X and Y (rBXBY = NULL),
# the autoregressive effects of X are .8 (equal across waves),
# the autoregressive effects of Y are .7 (equal across waves), and
# the crossed effects of Y (Y1 -> X2 and Y2 -> X3) are .1 (equal across waves).
set.seed(300121)
powerRICLPM <- semPower.powerRICLPM(type = 'post-hoc',
                                    nWaves = 3,
                                    autoregEffects = c(.8, .7),
                                    crossedEffects = c(.2, .1),
                                    waveEqual = c('autoregX', 'autoregY',
                                                  'crossedX', 'crossedY'),
                                    rXY = NULL,
                                    rBXBY = NULL,
                                    nullEffect = 'crossedX = 0',
                                    Lambda = diag(6),
                                    alpha = .05, N = 500,
                                    simulatedPower = TRUE,
                                    simOptions = list(nReplications = 500))

## End(Not run)
```

`semPower.showPlot` *semPower.showPlot*

Description

Shows a plot showing central and non-central chi-square distribution.

Usage

```
semPower.showPlot(chiCrit, ncp, df, linewidth = 1, showLabels = TRUE)
```

Arguments

`chiCrit`	critical chi-square, e. g. `qchisq(alpha, df, ncp = 0, lower.tail = FALSE)`
`ncp`	non-centrality parameter under H1
`df`	degrees of freedom
`linewidth`	linewidth
`showLabels`	whether to add labels

simulate *simulate*

Description

Estimates empirical power using a simulation approach.

Usage

```
simulate(
  modelH0 = NULL,
  modelH1 = NULL,
  Sigma = NULL,
  mu = NULL,
  N = NULL,
  alpha = NULL,
 simOptions = list(nReplications = 500, minConvergenceRate = 0.75, type = "normal",
  missingVars = NULL, missingVarProp = 0, missingProp = 0, missingMechanism = "MCAR",
    nCores = 1, futureStrategy = "multisession"),
  lavOptions = NULL,
  lavOptionsH1 = lavOptions,
  returnFmin = TRUE
)
```

Arguments

modelH0	lavaan model string defining the (incorrect) analysis model.
modelH1	lavaan model string defining the comparison model. If omitted, the saturated model is the comparison model.
Sigma	population covariance matrix.
mu	population means.
N	sample size
alpha	alpha error probability
simOptions	a list of additional options specifying simulation details, see details.
lavOptions	a list of additional options passed to lavaan, e. g., list(estimator = 'mlm') to request robust ML estimation
lavOptionsH1	lavoptions when fitting modelH1. If NULL, the same as lavOptions.
returnFmin	whether to return the mean unbiased Fmin over replications (i. e., fmin_0 = fmin_hat - df/N)

Details

The details of the simulation are specified in simOptions, which is a list that may have the following components:

- nReplications: The targeted number of valid simulation runs, defaults to 500.
- minConvergenceRate: The minimum convergence rate required, defaults to .75. The maximum actual simulation runs are increased by a factor of 1/minConvergenceRate.
- type: specifies whether the data should be generated from a population assuming multivariate normality ('normal'; the default), or based on an approach generating non-normal data ('IG', 'mnonr', 'RK', or 'VM'). The approaches generating non-normal data require additional arguments detailed below.
- missingVars: vector specifying the variables containing missing data (defaults to NULL).
- missingVarProp: can be used instead of missingVars: The proportion of variables containing missing data (defaults to zero).
- missingProp: The proportion of missingness for variables containing missing data (defaults to zero), either a single value or a vector giving the probabilities for each variable.
- missingMechanism: The missing data mechanism, one of 'MCAR' (the default), 'MAR', or 'NMAR'.
- nCores: The number of cores to use for parallel processing. Defaults to 1 (= no parallel processing). This requires the doFuture package.
- futureStrategy: A string specifying the strategy how to resolve a future when nCores is larger than 1. Defaults to 'multisession'. This is passed to the plan method of the doFuture package. See the doFuture package for valid strategies.

type = 'IG' implements the independent generator approach (IG, Foldnes & Olsson, 2016) approach specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors. This requires the covsim package.

type = 'mnonr' implements the approach suggested by Qu, Liu, & Zhang (2020) and requires provision of Mardia's multivariate skewness (skewness) and kurtosis (kurtosis), where skewness must be non-negative and kurtosis must be at least 1.641 skewness + p (p + 0.774), where p is the number of variables. This requires the mnonr package.

type = 'RK' implements the approach suggested by Ruscio & Kaczetow (2008) and requires provision of the population distributions of each variable (distributions). distributions must be a list (if all variables shall be based on the same population distribution) or a list of lists. Each component must specify the population distribution (e.g. rchisq) and additional arguments (list(df = 2)).

type = 'VM' implements the third-order polynomial method (Vale & Maurelli, 1983) specifying third and fourth moments of the marginals, and thus requires that skewness (skewness) and excess kurtosis (kurtosis) for each variable are provided as vectors.

Foldnes, N. & Olsson, U. H. (2016) A Simple Simulation Technique for Nonnormal Data with Prespecified Skewness, Kurtosis, and Covariance Matrix. *Multivariate Behavioral Research, 51*, 207-219. doi: 10.1080/00273171.2015.1133274

Qu, W., Liu, H., & Zhang, Z. (2020). A method of generating multivariate non-normal random numbers with desired multivariate skewness and kurtosis. *Behavior Research Methods, 52*, 939-946. doi: 10.3758/s13428-019-01291-5

Ruscio, J., & Kaczetow, W. (2008). Simulating multivariate nonnormal data using an iterative algorithm. *Multivariate Behavioral Research, 43*, 355-381. doi: 10.1080/00273170802285693

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

Value

Returns empirical power: `sum(p < alpha) / nReplications` or a list (if `returnFmin = TRUE`) with the following components:

`ePower`	the empirical power.
`meanFmin`	the estimated mean unbiased Fmin over replications (i. e., `fmin_0 = fmin_hat - df/N`).
`meanFminGroups`	the estimated mean unbiased Fmin by groups given as a vector, assuming the df spread equally over groups. Therefore, `meanFmin != sum(meanFminGroups)`
`df`	the model df.
`nrep`	the number of successful replications.
`convergenceRate`	the convergence rate of the H0 model.
`bChiSq`	median chi-square bias of the H1 model
`bLambda`	average median bias in lambda in the H1 model
`bPhi`	average median bias in phi in the H1 model
`bPsi`	average median bias in psi in the H1 model
`bBeta`	average median bias in beta in the H1 model

Examples

```
## Not run:
# create Sigma and modelH0 using powerCFA
powerCFA <- semPower.powerCFA(type = 'a-priori', alpha = .05, beta = .05,
                              comparison = 'saturated',
                              Phi = .2, loadings = list(rep(.5, 3), rep(.7, 3)))

# perform simulated power analysis using defaults
simulate(modelH0 = powerCFA$modelH0,
         Sigma = powerCFA$Sigma,
         N = powerCFA$requiredN,
         alpha = .05,
         simulatedPower = TRUE)


# same with additional options
simulate(modelH0 = powerCFA$modelH0,
         Sigma = powerCFA$Sigma,
         N = powerCFA$requiredN,
         alpha = .05,
         simulatedPower = TRUE,
         simOptions = list(nReplications = 500,
                           minConvergenceRate = .80,
                           nCores = 8))


# same with IG as data generation routine
simulate(modelH0 = powerCFA$modelH0,
```

```
          Sigma = powerCFA$Sigma,
          N = powerCFA$requiredN,
          alpha = .05,
          simulatedPower = TRUE,
          simOptions = list(type = 'IG',
                            skewness = c(0, 1, -2, 6, 5, 4),
                            kurtosis = c(-3, 6, 9, 0, 2, -2)))


# same with mnonr as data generation routine
simulate(modelH0 = powerCFA$modelH0,
         Sigma = powerCFA$Sigma,
         N = powerCFA$requiredN,
         alpha = .05,
         simulatedPower = TRUE,
         simOptions = list(type = 'mnonr',
                           skewness = 1,
                           kurtosis = 50))


# same with RK as data generation routine
distributions <- list(
  list('rnorm', list(mean = 0, sd = 10)),
  list('runif', list(min = 0, max = 1)),
  list('rbeta', list(shape1 = 1, shape2 = 2)),
  list('rexp', list(rate = 1)),
  list('rpois', list(lambda = 4)),
  list('rbinom', list(size = 1, prob = .5))
)
simulate(modelH0 = powerCFA$modelH0,
         Sigma = powerCFA$Sigma,
         N = powerCFA$requiredN,
         alpha = .05,
         simulatedPower = TRUE,
         simOptions = list(type = 'RK',
                           distributions = distributions))

## End(Not run)
```

---

summary.semPower.aPriori *summary.semPower.aPriori*

Description

provide summary of a-priori power analyses

Usage

```
## S3 method for class 'semPower.aPriori'
summary(object, ...)
```

Arguments

object	result object from semPower.aPriori
...	other

---

summary.semPower.compromise *summary.sempower.compromise*

Description

provide summary of compromise post-hoc power analyses

Usage

```
## S3 method for class 'semPower.compromise'
summary(object, ...)
```

Arguments

object	result object from semPower.compromise
...	other

---

summary.semPower.postHoc *semPower.postHoc.summary*

Description

provide summary of post-hoc power analyses

Usage

```
## S3 method for class 'semPower.postHoc'
summary(object, ...)
```

Arguments

object	result object from semPower.posthoc
...	other

validateInput *validateInput*

Description

Validates input for power functions.

Usage

```
validateInput(
  power.type = NULL,
  effect = NULL,
  effect.measure = NULL,
  alpha = NULL,
  beta = NULL,
  power = NULL,
  abratio = NULL,
  N = NULL,
  df = NULL,
  p = NULL,
  SigmaHat = NULL,
  Sigma = NULL,
  muHat = NULL,
  mu = NULL,
  fittingFunction = "ML",
  simulatedPower = FALSE,
  modelH0 = NULL,
  power.min = alpha,
  power.max = 0.999,
  effect.min = NULL,
  effect.max = NULL,
  steps = 50,
  linewidth = 1
)
```

Arguments

power.type	type of power analyses, one of "a-priori", "post-hoc", "compromise", "powerplot.byN", "powerplot.byEffect"
effect	effect size specifying the discrepancy between H0 and H1
effect.measure	type of effect, one of "F0", "RMSEA", "Mc", "GFI", "AGFI"
alpha	alpha error
beta	beta error
power	power (= 1 - beta)
abratio	ratio alpha/beta

N	the number of observations
df	the model degrees of freedom
p	the number of observed variables, required for effect.measure = "GFI" and effect.measure = "AGFI"
SigmaHat	model implied covariance matrix
Sigma	observed (or population) covariance matrix
muHat	model implied mean vector
mu	observed (or population) mean vector
fittingFunction	whether to use ML (the default) or WLS
simulatedPower	whether to perform a simulated (TRUE) (rather than analytical, FALSE) power analysis.
modelH0	for simulated power: lavaan model string defining the (incorrect) analysis model.
power.min	for plotting: minimum power
power.max	for plotting: maximum power
effect.min	for plotting: minimum effect
effect.max	for plotting: maximum effect
steps	for plotting: number of sampled points
linewidth	for plotting: linewidth

Index

checkBounded, 3
checkComparisonModel, 4
checkDataGenerationTypes, 4
checkEllipsis, 5
checkMissingTypes, 5
checkNullEffect, 6
checkPositive, 6
checkPositiveDefinite, 7
checkPowerTypes, 7
checkSquare, 8
checkSymmetricSquare, 8

doSim, 9

genData, 9
genData.IG, 10
genData.mnonr, 11
genData.normal, 12
genData.RK, 13
genData.VM, 14
genLambda, 15
genLambda(), *45*, *47*
genModelString, 16
getAGFI.F, 17
getBetadiff, 17
getCFI.Sigma, 18
getCFI.Sigma.mgroups, 19
getChiSquare.F, 20
getChiSquare.NCP, 20
getDiscrepancyFunctionFromFittingFunction, 21
getErrorDiff, 21
getF, 22
getF.AGFI, 23
getF.GFI, 23
getF.Mc, 24
getF.RMSEA, 24
getF.Sigma, 25
getFittingFunctionFromEstimator, 25
getFormattedResults, 26
getFormattedSimulationResults, 26
getGFI.F, 27
getIndices.F, 27
getKSdistance, 28
getLavOptions, 29
getMc.F, 29
getNCP, 30
getPhi.B, 30
getPsi.B, 31
getRMSEA.F, 32
getSRMR.Sigma, 33
getSRMR.Sigma.mgroups, 33
getWLSv, 34

makeRestrictionsLavFriendly, 34

orderLavCov, 35
orderLavMu, 35
orderLavResults, 36

powerPrepare, 36

semPower, 38
semPower-package *(*semPower*)*, 38
semPower.aPriori, *38*, 41
semPower.aPriori(), *39*, *45*, *52*, *55*, *58*, *65*, *68*, *74*, *77*, *81*, *83*, *88*, *91*, *97*, *99*, *102*, *105*, *111*, *116*, *121*, *123*, *127*, *131*, *135*, *138*, *144*, *146*, *152*, *155*
semPower.compromise, *38*, 43
semPower.compromise(), *39*, *42*, *52*, *55*, *58*, *65*, *68*, *74*, *77*, *81*, *83*, *88*, *91*, *97*, *99*, *102*, *105*, *111*, *116*, *121*, *123*, *127*, *131*, *135*, *138*, *144*, *146*, *152*, *155*
semPower.genSigma, 45
semPower.genSigma(), *58*, *68*, *77*, *83*, *91*, *105*, *116*, *123*, *131*, *146*, *155*
semPower.getDf, 49
semPower.getDf(), *42*, *44*, *51*
semPower.postHoc, *38*, 50

semPower.postHoc(), *39*, *42*, *45*, *55*, *58*, *65*, *68*, *74*, *77*, *81*, *83*, *88*, *91*, *97*, *99*, *102*, *105*, *111*, *116*, *121*, *123*, *127*, *131*, *135*, *138*, *144*, *146*, *152*, *155*
semPower.powerARMA, 53
semPower.powerARMA(), *66*, *103*, *112*
semPower.powerAutoreg, 63
semPower.powerAutoreg(), *56*, *89*, *103*, *112*, *144*
semPower.powerBifactor, 72
semPower.powerCFA, 80
semPower.powerCLPM, 86
semPower.powerCLPM(), *56*, *66*, *153*
semPower.powerLav, 96
semPower.powerLav(), *121*
semPower.powerLGCM, 101
semPower.powerLI, 110
semPower.powerLI(), *56*, *66*, *103*, *127*
semPower.powerMediation, 119
semPower.powerMediation(), *81*, *144*
semPower.powerMI, 126
semPower.powerMI(), *81*, *112*
semPower.powerPath, 134
semPower.powerPath(), *121*
semPower.powerPlot, 139
semPower.powerPlot.byEffect, 140
semPower.powerPlot.byN, 141
semPower.powerPlot.byN(), *139*
semPower.powerRegression, 142
semPower.powerRegression(), *81*
semPower.powerRICLPM, 150
semPower.powerRICLPM(), *89*
semPower.showPlot, 163
simulate, 164
simulate(), *42*, *52*, *97*
summary.semPower.aPriori, 167
summary.semPower.compromise, 168
summary.semPower.postHoc, 168

validateInput, 169