Package ‘semTools’

February 14, 2026

Encoding UTF-8

Version 0.5-8

Title Useful Tools for Structural Equation Modeling

Description Provides miscellaneous tools for structural equation modeling, many of which extend the 'lavaan' package. For example, latent interactions can be estimated using product indicators (Lin et al., 2010, <doi:10.1080/10705511.2010.488999>) and simple effects probed; analytical power analyses can be conducted (Jak et al., 2021, <doi:10.3758/s13428-020-01479-0>); and scale reliability can be estimated based on estimated factor-model parameters.

Depends R(>= 4.0), lavaan(>= 0.6-21), methods

Imports graphics, pbivnorm, stats, utils

Suggests blavaan, emmeans, lavaan.mi, MASS, mice, mnormt, parallel, restriktor, testthat

License GPL (>= 2)

LazyData yes

LazyLoad yes

URL `https://github.com/simsem/semTools/wiki`

BugReports `https://github.com/simsem/semTools/issues`

Date 2026-02-14

RoxygenNote 7.3.3

NeedsCompilation no

Author Terrence D. Jorgensen [aut, cre] (ORCID: <`https://orcid.org/0000-0001-5111-6773`>),
Sunthud Pornprasertmanit [aut] (ORCID: <`https://orcid.org/0009-0006-6887-9930`>),
Alexander M. Schoemann [aut] (ORCID: <`https://orcid.org/0000-0002-8479-8798`>),
Yves Rosseel [aut] (ORCID: <`https://orcid.org/0000-0002-4129-4477`>),
Patrick Miller [ctb],
Corbin Quick [ctb],

Mauricio Garnier-Villarreal [ctb] (ORCID:
<https://orcid.org/0000-0002-2951-6647>),
James Selig [ctb],
Aaron Boulton [ctb],
Kristopher Preacher [ctb],
Donna Coffman [ctb],
Mijke Rhemtulla [ctb] (ORCID: <https://orcid.org/0000-0003-2572-2424>),
Alexander Robitzsch [ctb] (ORCID:
<https://orcid.org/0000-0002-8226-3132>),
Craig Enders [ctb],
Ruben Arslan [ctb] (ORCID: <https://orcid.org/0000-0002-6670-5658>),
Bell Clinton [ctb],
Pavel Panko [ctb],
Edgar Merkle [ctb] (ORCID: <https://orcid.org/0000-0001-7158-0653>),
Steven Chesnut [ctb],
Jarrett Byrnes [ctb],
Jason D. Rights [ctb],
Ylenio Longo [ctb],
Maxwell Mansolf [ctb] (ORCID: <https://orcid.org/0000-0001-6861-8657>),
Mattan S. Ben-Shachar [ctb] (ORCID:
<https://orcid.org/0000-0002-4287-4801>),
Mikko Rönkkö [ctb] (ORCID: <https://orcid.org/0000-0001-7988-7609>),
Andrew R. Johnson [ctb] (ORCID:
<https://orcid.org/0000-0001-7000-8065>),
Leonard Vanbrabant [ctb]

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

Repository CRAN

Date/Publication 2026-02-14 07:40:02 UTC

Contents

`auxiliary` *Implement Saturated Correlates with FIML*

Description

Automatically add auxiliary variables to a lavaan model when using full information maximum likelihood (FIML) to handle missing data

Usage

```
auxiliary(model, data, aux, fun, ..., envir = getNamespace("lavaan"),
  return.syntax = FALSE)

lavaan.auxiliary(model, data, aux, ..., envir = getNamespace("lavaan"))

cfa.auxiliary(model, data, aux, ..., envir = getNamespace("lavaan"))

sem.auxiliary(model, data, aux, ..., envir = getNamespace("lavaan"))

growth.auxiliary(model, data, aux, ..., envir = getNamespace("lavaan"))
```

Arguments

`model`	The analysis model can be specified with 1 of 2 objects: 1. lavaan `lavaan::model.syntax()` specifying a hypothesized model *without* mention of auxiliary variables in `aux` 2. a parameter table, as returned by `lavaan::parTable()`, specifying the target model *without* auxiliary variables. This option requires these columns (and silently ignores all others): `c("lhs","op","rhs","user","group","free","label","plabe`
`data`	`data.frame` that includes auxiliary variables as well as any observed variables in the `model`
`aux`	`character`. Names of auxiliary variables to add to `model`
`fun`	`character`. Name of a specific lavaan function used to fit `model` to `data` (i.e., `"lavaan"`, `"cfa"`, `"sem"`, or `"growth"`). Only required for `auxiliary`.
`...`	Additional arguments to pass to `fun=`.
`envir`	Passed to `do.call()`.
`return.syntax`	`logical` indicating whether to return a `character` string of `lavaan::model.syntax()` that can be added to a target `model=` that is also a `character` string. This can be advantageous, for example, to use add saturated correlates to a **blavaan** model.

Details

These functions are wrappers around the corresponding lavaan functions. You can use them the same way you use `lavaan::lavaan()`, but you *must* pass your full `data.frame` to the `data` argument. Because the saturated-correlates approaches (Enders, 2008) treats exogenous variables as random, `fixed.x` must be set to `FALSE`. Because FIML requires continuous data (although nonnormality corrections can still be requested), no variables in the model nor auxiliary variables specified in `aux` can be declared as `ordered`.

Value

a fitted lavaan::lavaan object. Additional information is stored as a `list` in the `@external` slot:

- `baseline.model`. a fitted lavaan::lavaan object. Results of fitting an appropriate independence model for the calculation of incremental fit indices (e.g., CFI, TLI) in which the auxiliary variables remain saturated, so only the target variables are constrained to be orthogonal. See Examples for how to send this baseline model to `lavaan::fitMeasures()`.
- `aux`. The character vector of auxiliary variable names.
- `baseline.syntax`. A character vector generated within the `auxiliary` function, specifying the `baseline.model` syntax.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Enders, C. K. (2008). A note on the use of missing auxiliary variables in full information maximum likelihood-based structural equation models. *Structural Equation Modeling, 15*(3), 434–448. doi:10.1080/10705510802154307

Examples

```
dat1 <- lavaan::HolzingerSwineford1939
set.seed(12345)
dat1$z <- rnorm(nrow(dat1))
dat1$x5 <- ifelse(dat1$z < quantile(dat1$z, .3), NA, dat1$x5)
dat1$x9 <- ifelse(dat1$z > quantile(dat1$z, .8), NA, dat1$x9)

targetModel <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"

## works just like cfa(), but with an extra "aux" argument
fitaux1 <- cfa.auxiliary(targetModel, data = dat1, aux = "z",
                         missing = "fiml", estimator = "mlr")

## with multiple auxiliary variables and multiple groups
fitaux2 <- cfa.auxiliary(targetModel, data = dat1, aux = c("z","ageyr","grade"),
```

```
                              group = "school", group.equal = "loadings")

## calculate correct incremental fit indices (e.g., CFI, TLI)
fitMeasures(fitaux2, fit.measures = c("cfi","tli"))
## NOTE: lavaan will use the internally stored baseline model, which
##       is the independence model plus saturated auxiliary parameters
lavInspect(fitaux2@external$baseline.model, "free")
```

`AVE` *Calculate average variance extracted*

Description

Calculate average variance extracted (AVE) per factor from `lavaan` object

Usage

```
AVE(object, obs.var = TRUE, omit.imps = c("no.conv", "no.se"),
  omit.factors = character(0), dropSingle = TRUE, return.df = TRUE)
```

Arguments

`object`	A lavaan::lavaan or lavaan.mi::lavaan.mi object, expected to contain only exogenous common factors (i.e., a CFA model). Cross-loadings are not allowed and will result in `NA` for any factor with indicator(s) that cross-load.
`obs.var`	`logical` indicating whether to compute AVE using observed variances in the denominator. Setting `FALSE` triggers using model-implied variances in the denominator.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However, gross model misspecification could also cause NPD solutions, users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.
`omit.factors`	`character` vector naming any common factors modeled in `object` whose indicators' AVE is not of interest.
`dropSingle`	`logical` indicating whether to exclude factors defined by a single indicator from the returned results. If `TRUE` (default), single indicators will still be included in the `total` column when `return.total = TRUE`.

`return.df`	`logical` indicating whether to return reliability coefficients in a `data.frame` (one row per group/level), which is possible when every model block includes the same factors (after excluding those in `omit.factors` and applying `dropSingle`).

Details

The average variance extracted (AVE) can be calculated by

$$AVE = \frac{\mathbf{1}'\text{diag}\left(\Lambda\Psi\Lambda'\right)\mathbf{1}}{\mathbf{1}'\text{diag}\left(\hat{\Sigma}\right)\mathbf{1}},$$

Note that this formula is modified from Fornell & Larcker (1981) in the case that factor variances are not 1. The proposed formula from Fornell & Larcker (1981) assumes that the factor variances are 1. Note that AVE will not be provided for factors consisting of items with dual loadings. AVE is the property of items but not the property of factors. AVE is calculated with polychoric correlations when ordinal indicators are used.

Value

`numeric` vector of average variance extracted from indicators per factor. For models with multiple "blocks" (any combination of groups and levels), vectors may be returned as columns in a `data.frame` with additional columns indicating the group/level (see `return.df=` argument description for caveat).

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement errors. *Journal of Marketing Research, 18*(1), 39–50. doi:10.2307/3151312

See Also

`compRelSEM()` for composite reliability estimates

Examples

```
data(HolzingerSwineford1939)
HS9 <- HolzingerSwineford1939[ , c("x7","x8","x9")]
HSbinary <- as.data.frame( lapply(HS9, cut, 2, labels=FALSE) )
names(HSbinary) <- c("y7","y8","y9")
HS <- cbind(HolzingerSwineford1939, HSbinary)

HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ y7 + y8 + y9 '
```

```
fit <- cfa(HS.model, data = HS, ordered = c("y7","y8","y9"), std.lv = TRUE)

## works for factors with exclusively continuous OR categorical indicators
AVE(fit) # uses observed (or unconstrained polychoric/polyserial) by default
AVE(fit, obs.var = FALSE)


## works for multigroup models and for multilevel models (and both)
data(Demo.twolevel)
## assign clusters to arbitrary groups
Demo.twolevel$g <- ifelse(Demo.twolevel$cluster %% 2L, "type1", "type2")
model2 <- ' group: type1
  level: within
    fac =~ y1 + L2*y2 + L3*y3
  level: between
    fac =~ y1 + L2*y2 + L3*y3

group: type2
  level: within
    fac =~ y1 + L2*y2 + L3*y3
  level: between
    fac =~ y1 + L2*y2 + L3*y3
'
fit2 <- sem(model2, data = Demo.twolevel, cluster = "cluster", group = "g")
AVE(fit2)
```

`BootMiss-class` *Class For the Results of Bollen-Stine Bootstrap with Incomplete Data*

Description

This class contains the results of Bollen-Stine bootstrap with missing data.

Usage

```
## S4 method for signature 'BootMiss'
show(object)

## S4 method for signature 'BootMiss'
summary(object)

## S4 method for signature 'BootMiss'
hist(x, ..., alpha = 0.05, nd = 2,
  printLegend = TRUE, legendArgs = list(x = "topleft"))
```

Arguments

`object, x`	object of class `BootMiss`

`...`	Additional arguments to pass to `graphics::hist()`
`alpha`	alpha level used to draw confidence limits
`nd`	number of digits to display
`printLegend`	`logical`. If `TRUE` (default), a legend will be printed with the histogram
`legendArgs`	`list` of arguments passed to the `graphics::legend()` function. The default argument is a list placing the legend at the top-left of the figure.

Value

The `hist` method returns a list of `length == 2`, containing the arguments for the call to `hist` and the arguments to the call for `legend`, respectively.

Slots

`time` A list containing 2 `difftime` objects (`transform` and `fit`), indicating the time elapsed for data transformation and for fitting the model to bootstrap data sets, respectively.

`transData` Transformed data

`bootDist` The vector of chi^2 values from bootstrap data sets fitted by the target model

`origChi` The chi^2 value from the original data set

`df` The degree of freedom of the model

`bootP` The p value comparing the original chi^2 with the bootstrap distribution

Objects from the Class

Objects can be created via the `bsBootMiss()` function.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

See Also

`bsBootMiss()`

Examples

```
# See the example from the bsBootMiss function
```

`bsBootMiss` *Bollen-Stine Bootstrap with the Existence of Missing Data*

Description

Implement the Bollen and Stine's (1992) Bootstrap when missing observations exist. The implemented method is proposed by Savalei and Yuan (2009). This can be used in two ways. The first and easiest option is to fit the model to incomplete data in `lavaan` using the FIML estimator, then pass that `lavaan` object to `bsBootMiss`.

Usage

```
bsBootMiss(x, transformation = 2, nBoot = 500, model, rawData, Sigma, Mu,
  group, ChiSquared, EMcov, writeTransData = FALSE, transDataOnly = FALSE,
  writeBootData = FALSE, bootSamplesOnly = FALSE, writeArgs, seed = NULL,
  suppressWarn = TRUE, showProgress = TRUE, ...)
```

Arguments

`x`	A target `lavaan` object used in the Bollen-Stine bootstrap
`transformation`	The transformation methods in Savalei and Yuan (2009). There are three methods in the article, but only the first two are currently implemented here. Use `transformation = 1` when there are few missing data patterns, each of which has a large size, such as in a planned-missing-data design. Use `transformation = 2` when there are more missing data patterns. The currently unavailable `transformation = 3` would be used when several missing data patterns have n = 1.
`nBoot`	The number of bootstrap samples.
`model`	Optional. The target model if `x` is not provided.
`rawData`	Optional. The target raw data set if `x` is not provided.
`Sigma`	Optional. The model-implied covariance matrix if `x` is not provided.
`Mu`	Optional. The model-implied mean vector if `x` is not provided.
`group`	Optional character string specifying the name of the grouping variable in `rawData` if `x` is not provided.
`ChiSquared`	Optional. The model's χ^2 test statistic if `x` is not provided.
`EMcov`	Optional, if `x` is not provided. The EM (or Two-Stage ML) estimated covariance matrix used to speed up Transformation 2 algorithm.
`writeTransData`	Logical. If `TRUE`, the transformed data set is written to a text file, `transDataOnly` is set to `TRUE`, and the transformed data is returned invisibly.
`transDataOnly`	Logical. If `TRUE`, the result will provide the transformed data only.
`writeBootData`	Logical. If `TRUE`, the stacked bootstrap data sets are written to a text file, `bootSamplesOnly` is set to `TRUE`, and the list of bootstrap data sets are returned invisibly.
`bootSamplesOnly`	Logical. If `TRUE`, the result will provide bootstrap data sets only.

`writeArgs`	Optional `list`. If `writeBootData` = `TRUE` or `writeBootData` = `TRUE`, user can pass arguments to the `utils::write.table()` function as a list. Some default values are provided: `file` = "bootstrappedSamples.dat", `row.names` = `FALSE`, and `na` = "-999", but the user can override all of these by providing other values for those arguments in the `writeArgs` list.
`seed`	The seed number used in randomly drawing bootstrap samples.
`suppressWarn`	Logical. If `TRUE`, warnings from `lavaan` function will be suppressed when fitting the model to each bootstrap sample.
`showProgress`	Logical. Indicating whether to display a progress bar while fitting models to bootstrap samples.
`...`	The additional arguments in the `lavaan::lavaan()` function. See also `lavaan::lavOptions()`

Details

The second is designed for users of other software packages (e.g., LISREL, EQS, Amos, or Mplus). Users can import their data, χ^2 value, and model-implied moments from another package, and they have the option of saving (or writing to a file) either the transformed data or bootstrapped samples of that data, which can be analyzed in other programs. In order to analyze the bootstrapped samples and return a p value, users of other programs must still specify their model using lavaan syntax.

Value

As a default, this function returns a BootMiss object containing the results of the bootstrap samples. Use `show`, `summary`, or `hist` to examine the results. Optionally, the transformed data set is returned if `transDataOnly` = `TRUE`. Optionally, the bootstrap data sets are returned if `bootSamplesOnly` = `TRUE`.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

Syntax for transformations borrowed from supplementary materials in Savalei & Yuan (2009)

References

Bollen, K. A., & Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods & Research, 21*(2), 205–229. doi:10.1177/0049124192021002004

Savalei, V., & Yuan, K.-H. (2009). On the model-based bootstrap with missing data: Obtaining a p-value for a test of exact fit. *Multivariate Behavioral Research, 44*(6), 741–763. doi:10.1080/00273170903333590

See Also

BootMiss

Examples

```
dat1 <- HolzingerSwineford1939
dat1$x5 <- ifelse(dat1$x1 <= quantile(dat1$x1, .3), NA, dat1$x5)
dat1$x9 <- ifelse(is.na(dat1$x5), NA, dat1$x9)

targetModel <- "
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
"
targetFit <- sem(targetModel, dat1, meanstructure = TRUE, std.lv = TRUE,
                 missing = "fiml", group = "school")
summary(targetFit, fit = TRUE, standardized = TRUE)


## The number of bootstrap samples should be much higher than this example
temp <- bsBootMiss(targetFit, transformation = 1, nBoot = 10, seed = 31415)

temp
summary(temp)
hist(temp)
hist(temp, printLegend = FALSE) # suppress the legend
## user can specify alpha level (default: alpha = 0.05), and the number of
## digits to display (default: nd = 2).  Pass other arguments to hist(...),
## or a list of arguments to legend() via "legendArgs"
hist(temp, alpha = .01, nd = 3, xlab = "something else", breaks = 25,
     legendArgs = list("bottomleft", box.lty = 2))
```

`chisqSmallN` *Small-*N *correction for* $chi\hat{}2$ *test statistic*

Description

Calculate small-*N* corrections for chi^2 model-fit test statistic to adjust for small sample size (relative to model size).

Usage

```
chisqSmallN(fit0, fit1 = NULL, smallN.method = if (is.null(fit1))
  c("swain", "yuan.2015") else "yuan.2005", ..., omit.imps = c("no.conv",
  "no.se"))
```

Arguments

`fit0`, `fit1`	lavaan::lavaan or lavaan.mi::lavaan.mi object(s)

`smallN.method`	`character` indicating the small-*N* correction method to use. Multiple may be chosen (all of which assume normality), as described in Shi et al. (2018): `c("swain","yuan.2015","yuan.2005","bartlett")`. Users may also simply select `"all"`.
`...`	Additional arguments to the `lavaan::lavTestLRT()` or `lavaan.mi::lavTestLRT.mi()` functions. Ignored when `is.null(fit1)`.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Ignored unless `fit0` (and optionally `fit1`) is a lavaan.mi::lavaan.mi object. See `lavaan.mi::lavTestLRT.mi()` for a description of options and defaults.

Details

Four finite-sample adjustments to the chi-squared statistic are currently available, all of which are described in Shi et al. (2018). These all assume normally distributed data, and may not work well with severely nonnormal data. Deng et al. (2018, section 4) review proposed small-*N* adjustments that do not assume normality, which rarely show promise, so they are not implemented here. This function currently will apply small-*N* adjustments to scaled test statistics with a warning that they do not perform well (Deng et al., 2018).

Value

A `list` of `numeric` vectors: one for the originally requested statistic(s), along with one per requested `smallN.method`. All include the the (un)adjusted test statistic, its *df*, and the *p* value for the test under the null hypothesis that the model fits perfectly (or that the 2 models have equivalent fit). The adjusted chi-squared statistic(s) also include(s) the scaling factor for the small-*N* adjustment.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Deng, L., Yang, M., & Marcoulides, K. M. (2018). Structural equation modeling with many variables: A systematic review of issues and developments. *Frontiers in Psychology, 9*, 580. doi:10.3389/fpsyg.2018.00580

Shi, D., Lee, T., & Terry, R. A. (2018). Revisiting the model size effect in structural equation modeling. *Structural Equation Modeling, 25*(1), 21–40. doi:10.1080/10705511.2017.1369088

Examples

```
HS.model <- '
    visual  =~ x1 + b1*x2 + x3
    textual =~ x4 + b2*x5 + x6
    speed   =~ x7 + b3*x8 + x9
'
fit1 <- cfa(HS.model, data = HolzingerSwineford1939[101:150,])
## test a single model (implicitly compared to a saturated model)
chisqSmallN(fit1)
```

```
## fit a more constrained model
fit0 <- cfa(HS.model, data = HolzingerSwineford1939[101:150,],
            orthogonal = TRUE)
## compare 2 models
chisqSmallN(fit1, fit0)
```

`clipboard` *Copy or save the result of* `lavaan` *or* `FitDiff` *objects into a clipboard or a file*

Description

Copy or save the result of `lavaan` or FitDiff object into a clipboard or a file. From the clipboard, users may paste the result into the Microsoft Excel or spreadsheet application to create a table of the output.

Usage

```
clipboard(object, what = "summary", ...)

saveFile(object, file, what = "summary", tableFormat = FALSE,
  fit.measures = "default", writeArgs = list(), ...)
```

Arguments

`object`	An object of class lavaan::lavaan or FitDiff.
`what`	The attributes of the `lavaan` object to be copied in the clipboard. `"summary"` is to copy the screen provided from the `summary` function. `"epceqfit"` is to copy the result from the `epcEquivFit()` function. Other attributes listed in the `inspect` method in the lavaan::lavaan could also be used, such as `"coef"`, `"se"`, `"fit"`, `"samp"`, and so on. Ignored for FitDiff-class objects.
`...`	Additional arguments when passing a `lavaan` object to the `summary` or `epcEquivFit()` function.
`file`	A file name used for saving the result.
`tableFormat`	If `TRUE`, save the result in the table format using tabs for separation. Otherwise, save the result as the output screen printed in the R console.
`fit.measures`	`character` vector specifying names of fit measures returned by `lavaan::fitMeasures()` to be copied/saved. Only relevant if `object` is class FitDiff.
`writeArgs`	`list` of additional arguments to be passed to `utils::write.table()`

Value

The resulting output will be saved into a clipboard or a file. If using the `clipboard` function, users may paste it in the other applications.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

Examples

```
library(lavaan)
HW.model <- ' visual  =~ x1 + c1*x2 + x3
              textual =~ x4 + c1*x5 + x6
              speed   =~ x7 +    x8 + x9 '

fit <- cfa(HW.model, data = HolzingerSwineford1939, group = "school")

if(interactive()){
# Copy the summary of the lavaan object
clipboard(fit)

# pass additional arguments to summary() method for class?lavaan
clipboard(fit, rsquare = TRUE, standardized = TRUE, fit.measures = TRUE)

# Copy the EPC equivalence testing results from the epcEquivFit() function
clipboard(fit, "epceqfit")

# Copy the parameter estimates
clipboard(fit, "coef")

# Copy the standard errors
clipboard(fit, "se")

# Copy the sample statistics
clipboard(fit, "samp")

# Copy the fit measures
clipboard(fit, "fit")

# Save the summary of the lavaan object
saveFile(fit, "out.txt")

# Save the EPC equivalence testing results from the epcEquivFit() function
saveFile(fit, "out.txt", "epceqfit")

# Save the parameter estimates
saveFile(fit, "out.txt", "coef")

# Save the standard errors
saveFile(fit, "out.txt", "se")

# Save the sample statistics
saveFile(fit, "out.txt", "samp")

# Save the fit measures
saveFile(fit, "out.txt", "fit")
```

```
}
```

`combinequark` *Combine the results from the quark function*

Description

This function builds upon the `quark()` function to provide a final dataset comprised of the original dataset provided to `quark()` and enough principal components to be able to account for a certain level of variance in the data.

Usage

```
combinequark(quark, percent)
```

Arguments

`quark`	Provide the `quark()` object that was returned. It should be a list of objects. Make sure to include it in its entirety.
`percent`	Provide a percentage of variance that you would like to have explained. That many components (columns) will be extracted and kept with the output dataset. Enter this variable as a number WITHOUT a percentage sign.

Value

The output of this function is the original dataset used in quark combined with enough principal component scores to be able to account for the amount of variance that was requested.

Author(s)

Steven R. Chesnut (University of Southern Mississippi <`Steven.Chesnut@usm.edu`>)

See Also

`quark()`

Examples

```
set.seed(123321)
dat <- HolzingerSwineford1939[,7:15]
misspat <- matrix(runif(nrow(dat) * 9) < 0.3, nrow(dat))
dat[misspat] <- NA
dat <- cbind(HolzingerSwineford1939[,1:3], dat)

quark.list <- quark(data = dat, id = c(1, 2))

final.data <- combinequark(quark = quark.list, percent = 80)
```

`compareFit` *Build an object summarizing fit indices across multiple models*

Description

This function will create the template to compare fit indices across multiple fitted lavaan objects. The results can be exported to a clipboard or a file later.

Usage

```
compareFit(..., nested = TRUE, argsLRT = list(), indices = TRUE,
  moreIndices = FALSE, baseline.model = NULL, nPrior = 1)
```

Arguments

`...`	fitted `lavaan` models or list(s) of `lavaan` objects. lavaan.mi::lavaan.mi objects are also accepted, but all models must belong to the same class.
`nested`	`logical` indicating whether the models in `...` are nested. See `net()` for an empirical test of nesting.
`argsLRT`	`list` of arguments to pass to `lavaan::lavTestLRT()`, as well as to `lavaan.mi::lavTestLRT.mi()` and `lavaan::fitMeasures()` when comparing lavaan.mi::lavaan.mi models.
`indices`	`logical` indicating whether to return fit indices from the `lavaan::fitMeasures()` function. Selecting particular indices is controlled in the `summary` method; see FitDiff.
`moreIndices`	`logical` indicating whether to return fit indices from the `moreFitIndices()` function. Selecting particular indices is controlled in the `summary` method; see FitDiff.
`baseline.model`	optional fitted lavaan::lavaan model passed to `lavaan::fitMeasures()` to calculate incremental fit indices.
`nPrior`	passed to `moreFitIndices()`, if relevant

Value

A FitDiff object that saves model fit comparisons across multiple models. If the models are not nested, only fit indices for each model are returned. If the models are nested, the differences in fit indices are additionally returned, as well as test statistics comparing each sequential pair of models (ordered by their degrees of freedom).

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

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

See Also

FitDiff, `clipboard()`

Examples

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

## non-nested models
fit1 <- cfa(HS.model, data = HolzingerSwineford1939)

m2 <- ' f1 =~ x1 + x2 + x3 + x4
        f2 =~ x5 + x6 + x7 + x8 + x9 '
fit2 <- cfa(m2, data = HolzingerSwineford1939)

(out1 <- compareFit(fit1, fit2, nested = FALSE))
summary(out1)


## nested model comparisons: measurement equivalence/invariance
fit.config <- cfa(HS.model, data = HolzingerSwineford1939, group = "school")
fit.metric <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
                  group.equal = "loadings")
fit.scalar <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
                  group.equal = c("loadings","intercepts"))
fit.strict <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
                  group.equal = c("loadings","intercepts","residuals"))

measEqOut <- compareFit(fit.config, fit.metric, fit.scalar, fit.strict,
                        moreIndices = TRUE) # include moreFitIndices()
summary(measEqOut)
summary(measEqOut, fit.measures = "all")
summary(measEqOut, fit.measures = c("aic", "bic", "sic", "ibic"))
```

`compRelSEM` *Composite Reliability using SEM*

Description

Calculate composite reliability from estimated factor-model parameters

Usage

```
compRelSEM(object, W = NULL, return.total = FALSE, obs.var = TRUE,
  tau.eq = FALSE, ord.scale = TRUE, shared = character(0),
  config = character(0), add.IRR = FALSE, higher = character(0),
  true = list(), dropSingle = TRUE, omit.factors = character(0),
  omit.indicators = character(0), omit.imps = c("no.conv", "no.se"),
  simplify = FALSE, return.df = simplify)
```

Arguments

`object`	A lavaan::lavaan or lavaan.mi::lavaan.mi object, expected to contain only exogenous common factors (i.e., a CFA model).
`W`	Composite weights applied to observed variables prior to summing. By default (`NULL`), unit-weights are applied to all indicators per factor (as well as all modeled indicators when `return.total=TRUE`), which is equivalent to specifying equal weights of *any* value to each indicator. Weights can be a `character` string specifying any number of composites using `lavaan::model.syntax()`, in the form `COMPOSITE <~ weight*indicator` (any indicator without a numeric `weight` is given a unit weight = 1). See **Details** and **Examples** about complicated CFAs (e.g., multilevel, higher-order, or bifactor).
`return.total`	For multidimensional CFAs, this `logical` value indicates whether to return a final index for the reliability of a composite of all modeled indicators (labeled `.TOTAL.`). This is redundant whenever there is already a common factor indicated by all items (e.g., the general factor in a bifactor model). This argument is ignored when using the `W=` argument to specify composites (optionally with weights). Setting a negative value (e.g., `-1`) returns **only** the `.TOTAL.` composite reliability (i.e., excluding coefficients per factor).
`obs.var`	`logical` indicating whether to compute reliability using observed (co)variances to compute the denominator. Setting `FALSE` triggers using model-implied (co)variances to compute the denominator.
`tau.eq`	`logical` indicating whether to assume (essential) tau-equivalence by calculating coefficient α (on observed or model-implied (co)variances, depending on `obs.var=`). Triggers error if requested in combination with unequal weights in `W=`. Setting `FALSE` (default) yields an "ω"-type coefficient. Optionally, a `character` vector of composite names can specify calculating coefficient α for a subset of all composites.
`ord.scale`	`logical` relevant only for composites of discrete items. Setting `TRUE` (default) applies Green and Yang's (2009, formula 21) method to calculate reliability of the actual composite (i.e., on the actual ordinal response scale). Setting `FALSE` yields coefficients that are only interpretable on the continuous latent-response scale, which can be interpreted as the upper bound of reliability if items were more approximately continuous. Ignored for factors with continuous indicators. Reliability cannot currently be calculated for composites of both discrete and continuous indicators.
`shared`	`character` vector of **composite names**, to be interpreted as representing (perhaps multidimensional) shared construct(s). Lai's (2021) coefficient ω^{B} or α^{B} is calculated to quantify reliability relative to error associated with both indicators (measurement error) and subjects (sampling error), like a generalizability coefficient. For purely *scale* reliability (relative to item/measurement error alone, i.e., Lai's ω^{2L}), omit the composite(s) from the `shared=` argument.
`config`	Deprecated `character` vector.
`add.IRR`	`logical` indicating whether to calculate an additional reliability coefficient for any composite listed in `shared=`. Given that subjects can be considered as raters of their cluster's shared construct, an interrater reliability (IRR) coefficient can quantify reliability relative to rater/sampling error alone.

`higher`	Deprecated, supplanted by using the `true=` argument.
`true`	Optional `list` of `character` vectors, with list-element names corresponding to composite names. Each composite can have a `character` vector with names of any common factor(s) that should be considered the source(s) of "true-score variance" in that composite. For any composite with a specification in `true=`, the default is to consider all common factors to contribute true-score variance to any items in the composite. Specifying a composite in `true=` is only necessary to deviate from this default, for example, to specify the "general" factor in a bifactor model, in order to obtain "hierarchical omega" (ω_H). A shortcut for this is available when `W=NULL`, by specifying a single `character` string (one of `"omegaH"`, `"omega.h"`, or `"omega_h"`) instead of a `list`.
`dropSingle`	When `W=NULL`, this `logical` indicates whether to exclude single-indicator factors from the list of default composites. Even when `TRUE` (default), single indicators are still included in the `.TOTAL.` composite when `return.total = TRUE`.
`omit.factors`	Deprecated, supplanted by using the `true=` argument.
`omit.indicators`	Deprecated, supplanted by using the `W=` argument.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results (using lavaan.mi::lavaan.mi). Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However, gross model misspecification could also cause NPD solutions. Users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.
`simplify`	`logical` indicating whether to return reliability coefficients in a `numeric` vector (for single-group model) or `data.frame` (one row per group, or per level in some cases). Specifying a negative number (`simplify = -1L`) additionally removes the informative headers printed to facilitate interpretation.
`return.df`	Deprecated `logical` argument, replaced by `simplify=`.

Details

Several coefficients for factor-analysis reliability have been termed "omega", which Cho (2021) argues is a misleading misnomer and argues for using ρ to represent them all, differentiated by descriptive subscripts. In our package, we strive to provide unlabeled coefficients, leaving it to the user to decide on a label in their report. But we do use the symbols α and ω in the formulas below in order to distinguish coefficients that do (not) assume essential tau-equivalence.

Bentler (1968) first introduced factor-analysis reliability for a unidimensional factor model with congeneric indicators, labeling the coefficients α. McDonald (1999) later referred to this *and other reliability coefficients*, first as θ (in 1970), then as ω, which is a source of confusion when reporting coefficients (Cho, 2021). Coefficients based on factor models were later generalized to account for multidimenisionality (possibly with cross-loadings) and correlated errors. The general ω formula implemented in this function is:

$$\omega = \frac{\boldsymbol{w}^{\prime} \Lambda \Phi \Lambda^{\prime} \boldsymbol{w}}{\boldsymbol{w}^{\prime} \hat{\Sigma} \boldsymbol{w}},$$

where $\hat{\Sigma}$ can be the model-implied covariance matrix from either the saturated model (i.e., the "observed" covariance matrix, used by default) or from the hypothesized CFA model, controlled by the `obs.var=` argument. All elements of matrices in the numerator and denominator are effectively summed by the multiplication of the outer terms $\boldsymbol{w}$, a k-dimensional vector of composite weights typically consisting of $\mathbf{1}$s, unless otherwise specified with the `W=` argument), and k is the number of variables in the composite. Reliability of subscale composites (or simply for separate factors in a joint CFA) can be calculated by setting omitted-indicator weights to 0. For unidimensional constructs with simple structure, the equation above is often simplified to a scalar representation (e.g., McDonald, 1999, Eq. 6.20b):

$$\omega = \frac{\left(\sum^{k}_{i = 1} \lambda_i \right)^{2} Var\left(\psi \right)}{\left(\sum^{k}_{i = 1} \lambda_i \right)^{2} Var\left(\psi \right) + \sum^{k}_{i = 1} \theta_{ii}},$$

Note that all coefficients are calculated from *total* factor variances: `lavInspect(object, "cov.lv")`, which assumes the fitted `object=` is a CFA, not a full SEM with latent regression slopes. If there is a Beta matrix, it should only contain higher-order factor loadings (see details below).

When the fitted CFA imposes constraints consistent with (essential) tau-equivalence, ω is equivalent to coefficient α (Cronbach, 1951):

$$\alpha = \frac{k}{k - 1}\left[1 - \frac{\textrm{tr} \left(\hat{\Sigma} \right)}{\mathbf{1}^{\prime} \hat{\Sigma} \mathbf{1}} \right],$$

where $\textrm{tr} \left(. \right)$ is the trace operation (i.e., the sum of diagonal elements). Setting `tau.eq=TRUE` triggers the application of this formula (rather than ω above) to the model-implied or observed covariance matrix (again controlled by the `obs.var=` argument).

Higher-Order Factors:

For higher-order constructs with latent indicators, only ω is available because α was not derived from CFA parameters (although it can be expressed in a particular restricted CFA specification).

The reliability of a composite that represents a higher-order construct requires partitioning the model-implied factor covariance matrix Φ in order to isolate the common-factor variance associated only with the higher-order factor. Using a second-order factor model, the model-implied covariance matrix of observed indicators $\hat{\Sigma}$ can be partitioned into 3 sources:

1. the second-order common-factor (co)variance: $\Lambda \boldsymbol{B} \Phi_2 \boldsymbol{B}^{\prime} \Lambda^{\prime}$
2. the residual variance of the first-order common factors (i.e., not accounted for by the second-order factor): $\Lambda \Psi_{u} \Lambda^{\prime}$
3. the measurement error of observed indicators: Θ

where Λ contains first-order factor loadings, $\boldsymbol{B}$ contains second-order factor loadings, Φ_2 is the model-implied covariance matrix of the second-order factor(s), and Ψ_u is the covariance matrix of first-order factor disturbances. In practice, we can use the full $\boldsymbol{B}$ matrix and full model-implied Φ matrix (i.e., including all latent factors) because the zeros in $\boldsymbol{B}$ will cancel out unwanted components of Φ. Thus, we can calculate the proportion of variance of a composite score that is attributable to the second-order factor:

$$\omega = \frac{\boldsymbol{w}' \Lambda \boldsymbol{B} \Phi \boldsymbol{B}' \Lambda' \boldsymbol{w}}{\boldsymbol{w}' \hat{\Sigma} \boldsymbol{w}},$$

where $\boldsymbol{w}$, $\hat{\Sigma}$, and k are defined as above. **Note** that if a higher-order factor also has observed indicators, it is necessary to model the observed indicators as single-indicator lower-order constructs, so that all of the higher-order factor indicators are latent (with loadings in the Beta matrix, not Lambda); otherwise, higher-order factor variance in the observed indicator is not captured in the numerator.

Bifactor or Multitrait–Multimethod (MTMM) Models:

These multidimensional models partition sources of common variance that are due to the factor of interest (e.g., a trait) as well as non-target factors (e.g., "method factors", such as item wording or type of respondent). The latter can be considered as systematic (i.e., non-random) sources of error, to be excluded from the numerator of a reliability coefficient, yielding so-called "hierarchical omega" (ω_{H}). On the other hand, non-target variance that can be expected in repeated measurement meets the classical test theory definition of reliability. Including method factors in the numerator yields so-called "omega total" (ω_{T}), which is the default approach in `compRelSEM()` because it is consistent with the classical test theory definition of reliability. However, users can obtain ω_{H} for a composite by using the `true=` argument to specify any factor(s) to be treated as representing true scores. The same approach can be taken to obtain the proportion of a (sub)scale composite's variance due to method factors (by listing those in `true=`), if that is of interest.

Categorical Indicators:

When all indicators (per composite) are ordinal, a CFA can be fitted that includes a threshold model (sometimes called Item Factor Analysis: IFA), which assumes a normally distributed latent response underlies each observed ordinal response. Despite making this assumption, a composite of ordinal items can only be calculated by assigning numerical values to the ordinal categories, so that the pseudo-numerical variables can be summed into a composite variable that is more approximately continuous than its items.

Applying the formulas above to IFA parameters provides the *hypothetical* reliability of a composite of latent responses: a composite which cannot be calculated in practice. Nonetheless, this hypothetical reliability can be interpreted as an estimate of what reliability *could* be if a more approximately continuous response scale were used (e.g., with sufficiently many response categories that the standardized solutions are equivalent between a fitted IFA and a fitted CFA that treats the ordinal responses as numeric; Chalmers, 2018). This can be requested by setting `ord.scale=FALSE`, in which case $\hat{\Sigma}$ in the formulas above is a *polychoric* correlation matrix. When `ord.scale=FALSE` and `tau.eq=TRUE`, this results in what Zumbo et al. (2007) termed "ordinal α" (see criticisms by Chalmers, 2018, and and a rejoinder by Zumbo & Kroc, 2019).

Alternatively, Green and Yang (2009, Eq. 21) derived a method to calculate model-based reliability (ω) from IFA parameters (i.e., incorporating the latent-response assumption) but that applies to the actual (i.e., ordinal) observed response scale (the default: `ord.scale=TRUE`). Lu et al. (2020) showed how to incorporate unequal weights into Green and Yang's (2009) formula, so `W=` can be used to estimate the (maximal) reliability of a weighted composite of ordinal variables. However, combining `ord.scale=TRUE` with `tau.eq=TRUE` is not available. For α to be interpretable on the observed ordinal scale, users must choose whether to (a) release the latent-response assumption, by fitting a CFA without a threshold model, or (b) fit an IFA model with constraints consistent with the assumption of (essential) tau-equivalence (i.e., equal factor loadings).

No method analogous to Green and Yang (2009, Eq. 21) has yet been proposed to calculate reliability with a mixture of categorical and continuous indicators, so any such composite is skipped with a warning.

Multilevel Measurement Models:

How to define reliability coefficients for scales employed in nested designs is an ongoing topic of methodological development, with some ongoing controversies about best practice when the target of measurement is the "cluster" or between-level (i.e., Level 2 in a 2-level design). Geldhof et al. (2014) proposed applying the standard formulas above to each level's CFA parameters and/or (model-implied) covariance matrix, whereas Lai (2021) proposed different formulas that account for all sources of variance in composites of observed variables.

There is no controversy about how to define a within-level reliability, coefficient, which can be interpreted as the reliability of a composite calculated by first centering each indicator around its cluster mean, then calculating the composite from the cluster-mean-centered items. Equivalently (i.e., the same formula), this can be interpreted as the *hypothetical* reliability of a composite of the items' latent Level-1 components. This coefficient can be requested with lavaan::model.syntax (to pass to the `W=` argument) that specifies a composite in a Level-1 "block", which not have the same name as any composite in the Level-2 block. If users do not use `W=` (i.e., calculate a reliability index per modeled common factor), then this can be accomplished by using unique factor names across levels.

This contrasts with reliability indices for between-level composites: The reliability of a *hypothetical* composite of items' latent between-level components (using formulas proposed by Geldhof et al., 2014) is **not** equivalent to the coefficient for a composite of items' observed cluster means, using generalizations of formulas proposed by Lai (2021):

$$\omega^{\mathrm{B}} = \frac{\boldsymbol{w}' \Lambda^{\mathrm{B}} \Phi^{\mathrm{B}} \Lambda^{\mathrm{B}\prime} \boldsymbol{w}}{\boldsymbol{w}' \hat{\Sigma}^{\mathrm{B}} \boldsymbol{w} + \frac{1}{\tilde{n}_{\mathrm{clus}}} \left(\boldsymbol{w}' \hat{\Sigma}^{\mathrm{W}} \boldsymbol{w} \right)},$$

$$\alpha^{\mathrm{B}} = \frac{2k}{k-1} \left[\frac{\sum_{i=2}^{k} \sum_{j=1}^{i-1} \hat{\sigma}_{ij}^{\mathrm{B}}}{\mathbf{1}' \hat{\Sigma}^{\mathrm{B}} \mathbf{1} + \frac{1}{\tilde{n}_{\mathrm{clus}}} \left(\mathbf{1}' \hat{\Sigma}^{\mathrm{W}} \mathbf{1} \right)} \right],$$

where $\tilde{n}_{\mathrm{clus}}$ is the harmonic-mean cluster size, and superscripts B and W indicate between- and within-level parameters. Obtaining these estimates of composite reliability requires fitting a 2-level CFA that provides the same factor structure and factor names in the models at both levels (following the advice of Jak et al., 2021), as well as the same composite name in both levels/blocks of syntax passed to `W=` (if used). Furthermore, the between-level composite name must be passed to the `shared=` argument; otherwise, the same factor/composite name across levels will yield Lai's (2021) coefficient for a configural construct (see **Examples**):

$$\omega^{\mathrm{2L}} = \frac{\boldsymbol{w}' \left(\Lambda^{\mathrm{W}} \Phi^{\mathrm{W}} \Lambda^{\mathrm{W}\prime} + \Lambda^{\mathrm{B}} \Phi^{\mathrm{B}} \Lambda^{\mathrm{B}\prime} \right) \boldsymbol{w}}{\boldsymbol{w}' \hat{\Sigma}^{\mathrm{B}} \boldsymbol{w} + \boldsymbol{w}' \hat{\Sigma}^{\mathrm{W}} \boldsymbol{w}},$$

$$\alpha^{2\mathrm{L}} = \frac{2k}{k-1}\left[\frac{\sum_{i=2}^{k}\sum_{j=1}^{i-1}\left(\hat{\sigma}_{ij}^{\mathrm{W}} + \hat{\sigma}_{ij}^{\mathrm{B}}\right)}{\mathbf{1}'\hat{\Sigma}^{\mathrm{B}}\mathbf{1} + \mathbf{1}'\hat{\Sigma}^{\mathrm{W}}\mathbf{1}}\right],$$

This can be interpreted as the scale-reliability coefficient ignoring the nested design, as both the common-factor variance of the Level-1 factor *and* of its Level-2 cluster means are treated as true-score variance.

Note that Lai's (2021) between-level reliability coefficients for a `shared` construct quantify generalizability across both indicators and raters (i.e., subjects rating their cluster's construct). Lüdtke et al. (2011) refer to these as measurement error and sampling error, respectively. From this perspective (and following from generalizability theory), an IRR coefficient can also be calculated:

$$\mathrm{IRR} = \frac{\boldsymbol{w}'\left(\hat{\Sigma}^{\mathrm{B}}\right)\boldsymbol{w}}{\boldsymbol{w}'\hat{\Sigma}^{\mathrm{B}}\boldsymbol{w} + \boldsymbol{w}'\hat{\Sigma}^{\mathrm{W}}\boldsymbol{w}},$$

which quantifies generalizability across rater/sampling-error only, and can be returned for any `shared=` construct's composite by setting `add.IRR=TRUE`.

Value

By default (`simplify=FALSE`) a `list` of `numeric` vectors (1 per composite) is returned. In multigroup CFA, the vector contains a reliability index for each group in which the composite can be computed. Each composite's vector has a `attr(..., "header")` with information to facilitate interpretation of that index:

- A list of variables in the composite, which determines the composite's total variance (denominator of reliability)
- Whether that total variance (denominator) is determined from the restricted model (i.e., CFA parameters) or unrestricted model (i.e., a freely estimated covariance matrix)
- Whether the variables in the composite are (a transformation of) observed variables, or whether they are *latent* (components of) variables. The latter (e.g., latent responses assumed to underlie observed ordinal indicators, or latent level-specific components of variables in a multilevel CFA) cannot be used to calculated an observed composite variable, so the resulting coefficient should be cautiously interpreted as a "hypothetical reliability" (Chalmers, 2018; Lai, 2021).
- The latent variables that contribute common-factor variance to the composite, which determine the composite's "true-score" variance (numerator of reliability)
- Which reliability formula was used: model-based reliability (so-called "omega") or coefficient alpha (a model-free lower-bound estimate of true reliability, equivalent to a model-based reliability that assumes tau-equivalence)

This header will be printed immediately above each composite's reliability coefficient. When multiple reliability coefficients are returned, **and** each vector in the list has the same length, then setting `simplify=TRUE` will collect the list of *single* coefficients into a vector, or the list of *multiple* coefficients into a `data.frame`, and their headers will be concatenated to be printed above the coefficients. Setting `simplify = -1L` (or any negative number) will omit the informative headers.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

Uses hidden functions to implement Green & Yang's (2009) reliability for categorical indicators, written by Sunthud Pornprasertmanit (`<psunthud@gmail.com>`) for the deprecated `reliability()` function.

References

Bentler, P. M. (1968). Alpha-maximized factor analysis (alphamax): Its relation to alpha and canonical factor analysis. *Psychometrika, 33*(3), 335–345. doi:10.1007/BF02289328

Chalmers, R. P. (2018). On misconceptions and the limited usefulness of ordinal alpha. *Educational and Psychological Measurement, 78*(6), 1056–1071. doi:10.1177/0013164417727036

Cho, E. (2021) Neither Cronbach's alpha nor McDonald's omega: A commentary on Sijtsma and Pfadt. *Psychometrika, 86*(4), 877–886. doi:10.1007/s11336021098011

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*(3), 297–334. doi:10.1007/BF02310555

Geldhof, G. J., Preacher, K. J., & Zyphur, M. J. (2014). Reliability estimation in a multilevel confirmatory factor analysis framework. *Psychological Methods, 19*(1), 72–91. doi:10.1037/a0032138

Green, S. B., & Yang, Y. (2009). Reliability of summed item scores using structural equation modeling: An alternative to coefficient alpha. *Psychometrika, 74*(1), 155–167. doi:10.1007/s11336008-90993

Jak, S., Jorgensen, T. D., & Rosseel, Y. (2021). Evaluating cluster-level factor models with `lavaan` and M*plus*. *Psych, 3*(2), 134–152. doi:10.3390/psych3020012

Lai, M. H. C. (2021). Composite reliability of multilevel data: It's about observed scores and construct meanings. *Psychological Methods, 26*(1), 90–102. doi:10.1037/met0000287

Lu, Z., Hong, M., & Kim, S. (2020). Formulas of multilevel reliabilities for tests with ordered categorical responses. In M. Wiberg, D. Molenaar, J. González, U.Böckenholt, & J.-S. Kim (Eds.), *Quantitative psychology: The 85th annual meeting of the Psychometric Society, Virtual* (pp. 103–112). Springer. doi:10.1007/9783030747725_10

Lüdtke, O., Marsh, H. W., Robitzsch, A., & Trautwein, U. (2011). A 2 × 2 taxonomy of multilevel latent contextual models: Accuracy–bias trade-offs in full and partial error correction models. *Psychological Methods, 16*(4), 444–467. doi:10.1037/a0024376

McDonald, R. P. (1999). *Test theory: A unified treatment*. Mahwah, NJ: Erlbaum.

Zumbo, B. D., Gadermann, A. M., & Zeisser, C. (2007). Ordinal versions of coefficients alpha and theta for Likert rating scales. *Journal of Modern Applied Statistical Methods, 6*(1), 21–29. doi:10.22237/jmasm/1177992180

Zumbo, B. D., & Kroc, E. (2019). A measurement is a choice and Stevens' scales of measurement do not help make it: A response to Chalmers. *Educational and Psychological Measurement, 79*(6), 1184–1197. doi:10.1177/0013164419844305

See Also

`maximalRelia()` for the maximal reliability of weighted composite

Examples

```
data(HolzingerSwineford1939)
HS9 <- HolzingerSwineford1939[ , c("x7","x8","x9")]
HSbinary <- as.data.frame( lapply(HS9, cut, 2, labels=FALSE) )
names(HSbinary) <- c("y7","y8","y9")
HS <- cbind(HolzingerSwineford1939, HSbinary)

HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ y7 + y8 + y9 '

fit  <- cfa(HS.model, data = HS, ordered = c("y7","y8","y9"), std.lv = TRUE)
fitg <- cfa(HS.model, data = HS, ordered = c("y7","y8","y9"), std.lv = TRUE,
            group = "school")

## works for factors with exclusively continuous OR categorical indicators
compRelSEM(fit)
compRelSEM(fitg)

## reliability for composite of ALL indicators only available when they are
## all continuous or all categorical.  The example below calculates a
## composite of continuous items from 2 factors (visual and textual)
## using the custom-weights syntax (note the "<~" operator)
w.tot <- '
  visual  <~ x1 + x2 + x3
  textual <~                x4 + x5 + x6
  total   <~ x1 + x2 + x3 + x4 + x5 + x6
'
compRelSEM(fit, W = w.tot)


## ----------------------
## Higher-order construct
## ----------------------

## Reliability of a composite that represents a higher-order factor
mod.hi <- ' visual  =~ x1 + x2 + x3
            textual =~ x4 + x5 + x6
            speed   =~ x7 + x8 + x9
            general =~ visual + textual + speed '

fit.hi <- cfa(mod.hi, data = HolzingerSwineford1939)
## "general" is the factor representing "true scores", but it has no
## observed indicators.  Must use custom-weights syntax:
compRelSEM(fit.hi, W = 'g <~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9')


## ----------------------
## Hierarchical omega
## and omega Total
## ----------------------
```

```
mod.bi <- ' visual  =~ x1 + x2 + x3
            textual =~ x4 + x5 + x6
            speed   =~ x7 + x8 + x9
            general =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 '
fit.bi <- cfa(mod.bi, data = HolzingerSwineford1939,
              orthogonal = TRUE, std.lv = TRUE)
compRelSEM(fit.bi, return.total = -1) # omega_Total
compRelSEM(fit.bi, return.total = -1, # omega_Hierarchical
           true = list(.TOTAL. = "general"))


## ----------------------
## Multilevel Constructs
## ----------------------

## Same factor structure with metric invariance across levels (Jak et al., 2021)
model2 <- '
  level: 1
    f1 =~ y1 + L2*y2 + L3*y3
    f2 =~ y4 + L5*y5 + L6*y6
  level: 2
    f1 =~ y1 + L2*y2 + L3*y3
    f2 =~ y4 + L5*y5 + L6*y6
'
fit2 <- sem(model2, data = Demo.twolevel, cluster = "cluster")

## Lai's (2021, Eq. 13) omega index for a configural (Level-1) construct,
## treating common-factor variance at both levels as "true" variance
compRelSEM(fit2)

## Lai's (2021, Eq. 17) omega index for a shared (Level-2) construct
## (also its interrater reliability coefficient)
compRelSEM(fit2, shared = c("f1","f2"), add.IRR = TRUE)

## Geldhof et al.'s (2014) level-specific indices imply a different
## composite (hypothetically) calculated per level.  Thus, use
## unique composite names per level.

W2.Geldhof <- ' level: 1
  F1w <~ y1 + y2 + y3
  F2w <~ y4 + y5 + y6
level: 2
  F1b <~ y1 + y2 + y3
  F2b <~ y4 + y5 + y6
'
compRelSEM(fit2, W = W2.Geldhof)
```

`dat2way` *Simulated Dataset to Demonstrate Two-way Latent Interaction*

Description

A simulated data set with 2 independent factors and 1 dependent factor where each factor has three indicators

Usage

```
dat2way
```

Format

A `data.frame` with 500 observations of 9 variables.

x1	The first indicator of the first independent factor
x2	The second indicator of the first independent factor
x3	The third indicator of the first independent factor
x4	The first indicator of the second independent factor
x5	The second indicator of the second independent factor
x6	The third indicator of the second independent factor
x7	The first indicator of the dependent factor
x8	The second indicator of the dependent factor
x9	The third indicator of the dependent factor

Source

Data were generated by the `MASS::mvrnorm()` function in the `MASS` package.

Examples

```
head(dat2way)
```

`dat3way` *Simulated Dataset to Demonstrate Three-way Latent Interaction*

Description

A simulated data set with 3 independent factors and 1 dependent factor where each factor has three indicators

Usage

```
dat3way
```

Format

A `data.frame` with 500 observations of 12 variables.

- **x1** The first indicator of the first independent factor
- **x2** The second indicator of the first independent factor
- **x3** The third indicator of the first independent factor
- **x4** The first indicator of the second independent factor
- **x5** The second indicator of the second independent factor
- **x6** The third indicator of the second independent factor
- **x7** The first indicator of the third independent factor
- **x8** The second indicator of the third independent factor
- **x9** The third indicator of the third independent factor
- **x10** The first indicator of the dependent factor
- **x11** The second indicator of the dependent factor
- **x12** The third indicator of the dependent factor

Source

Data were generated by the `MASS::mvrnorm()` function in the `MASS` package.

Examples

```
head(dat3way)
```

`datCat` *Simulated Data set to Demonstrate Categorical Measurement Invariance*

Description

A simulated data set with 2 factors with 4 indicators each separated into two groups

Usage

```
datCat
```

Format

A `data.frame` with 200 observations of 9 variables.

- **g** Sex of respondents
- **u1** Indicator 1
- **u2** Indicator 2
- **u3** Indicator 3

u4 Indicator 4

u5 Indicator 5

u6 Indicator 6

u7 Indicator 7

u8 Indicator 8

Source

Data were generated using the `lavaan` package.

Examples

```
head(datCat)
```

`discriminantValidity` *Calculate discriminant validity statistics*

Description

Calculate discriminant validity statistics based on a fitted lavaan object

Usage

```
discriminantValidity(object, cutoff = 0.9, merge = FALSE, level = 0.95,
  boot.ci.type = "perc")
```

Arguments

`object`	The lavaan::lavaan model object returned by the `lavaan::cfa()` function.
`cutoff`	A cutoff to be used in the constrained models in likelihood ratio tests.
`merge`	Whether the constrained models should be constructed by merging two factors as one. Implies `cutoff` = 1.
`level`	The confidence level required.
`boot.ci.type`	If bootstrapping was used, the type of interval required. The value should be one of `"norm"`, `"basic"`, `"perc"`, or `"bca.simple"`. For the first three options, see the help page of the `boot.ci` function in the boot package. The `"bca.simple"` option produces intervals using the adjusted bootstrap percentile (BCa) method, but with no correction for acceleration (only for bias). Note that the p-value is still computed assuming that the z-statistic follows a standard normal distribution.

Details

Evaluated on the measurement scale level, discriminant validity is commonly evaluated by checking if each pair of latent correlations is sufficiently below one (in absolute value) that the latent variables can be thought of representing two distinct constructs.

`discriminantValidity` function calculates two sets of statistics that are commonly used in discriminant validity evaluation. The first set are factor correlation estimates and their confidence intervals. The second set is a series of nested model tests, where the baseline model is compared against a set of constrained models that are constructed by constraining each factor correlation to the specified cutoff one at a time.

The function assume that the `object` is set of confirmatory factor analysis results where the latent variables are scaled by fixing their variances to 1s. If the model is not a CFA model, the function will calculate the statistics for the correlations among exogenous latent variables, but for the *residual* variances with endogenous variables. If the latent variables are scaled in some other way (e.g. fixing the first loadings), the function issues a warning and re-estimates the model by fixing latent variances to 1 (and estimating all loadings) so that factor covariances are already estimated as correlations.

The likelihood ratio tests are done by comparing the original baseline model against more constrained alternatives. By default, these alternatives are constructed by fixing each correlation at a time to a cutoff value. The typical purpose of this test is to demonstrate that the estimated factor correlation is well below the cutoff and a significant chi^2 statistic thus indicates support for discriminant validity. In some cases, the original correlation estimate may already be greater than the cutoff, making it redundant to fit a "restricted" model. When this happens, the likelihood ratio test will be replaced by comparing the baseline model against itself. For correlations that are estimated to be negative, a negation of the cutoff is used in the constrained model.

Another alternative is to do a nested model comparison against a model where two factors are merged as one by setting the `merge` argument to `TRUE`. In this comparison, the constrained model is constructed by removing one of the correlated factors from the model and assigning its indicators to the factor that remains in the model.

Value

A `data.frame` of latent variable correlation estimates, their confidence intervals, and a likelihood ratio tests against constrained models. with the following attributes:

baseline The baseline model after possible rescaling.

constrained A `list` of the fitted constrained models used in the likelihood ratio test.

Author(s)

Mikko Rönkkö (University of Jyväskylä; <`mikko.ronkko@jyu.fi`>):

References

Rönkkö, M., & Cho, E. (2022). An updated guideline for assessing discriminant validity. *Organizational Research Methods*, 25(1), 6–14. doi:10.1177/1094428120968614

Examples

```
library(lavaan)

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

fit <- cfa(HS.model, data = HolzingerSwineford1939)
discriminantValidity(fit)
discriminantValidity(fit, merge = TRUE)
```

`efa.ekc` *Empirical Kaiser criterion*

Description

Identify the number of factors to extract based on the Empirical Kaiser Criterion (EKC). The analysis can be run on a `data.frame` or data `matrix` (`data`), or on a correlation or covariance matrix (`sample.cov`) and the sample size (`sample.nobs`). A `data.frame` is returned with two columns: the eigenvalues from your data or covariance matrix and the reference eigenvalues. The number of factors suggested by the Empirical Kaiser Criterion (i.e. the sample eigenvalues greater than the reference eigenvalues), and the number of factors suggested by the original Kaiser Criterion (i.e. sample eigenvalues > 1) is printed above the output.

Usage

```
efa.ekc(data = NULL, sample.cov = NULL, sample.nobs = NULL,
  missing = "default", ordered = NULL, plot = TRUE)
```

Arguments

`data`	A `data.frame` or data `matrix` containing columns of variables to be factor-analyzed.
`sample.cov`	A covariance or correlation matrix can be used, instead of `data`, to estimate the eigenvalues.
`sample.nobs`	Number of observations (i.e. sample size) if `is.null(data)` and `sample.cov=` is used.
`missing`	If `"listwise"`, incomplete cases are removed listwise from the `data.frame`. If `"direct"` or `"ml"` or `"fiml"` and the `estimator=` is maximum likelihood, an EM algorithm is used to estimate an unrestricted covariance matrix (and mean vector). If `"pairwise"`, pairwise deletion is used. If ‘"default"‘, the value is set depending on the estimator and the mimic option (see `lavaan::lavCor()` for details).
`ordered`	`character` vector. Only used if object is a `data.frame`. Treat these variables as `ordered=` (ordinal) variables. Importantly, all other variables will be treated as numeric (unless `is.ordered == TRUE` in `data`). (see also lavCor)

`plot`	logical. Whether to print a scree plot comparing the sample eigenvalues with the reference eigenvalues.

Value

A `data.frame` showing the sample and reference eigenvalues.

The number of factors suggested by the Empirical Kaiser Criterion (i.e. the sample eigenvalues greater than the reference eigenvalues) is returned as an attribute (see **Examples**).

The number of factors suggested by the original Kaiser Criterion (i.e. sample eigenvalues > 1) is also printed as a header to the `data.frame`

Author(s)

Ylenio Longo (University of Nottingham; <`yleniolongo@gmail.com`>)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Braeken, J., & van Assen, M. A. L. M. (2017). An empirical Kaiser criterion. *Psychological Methods, 22*(3), 450–466. doi:10.1037/met0000074

Examples

```
## Simulate data with 3 factors
model <- '
  f1 =~ .3*x1 + .5*x2 + .4*x3
  f2 =~ .3*x4 + .5*x5 + .4*x6
  f3 =~ .3*x7 + .5*x8 + .4*x9
'
dat <- simulateData(model, seed = 123)
## save summary statistics
myCovMat <- cov(dat)
myCorMat <- cor(dat)
N <- nrow(dat)

## Run the EKC function
(out <- efa.ekc(dat))

## To extract the recommended number of factors using the EKC:
attr(out, "nfactors")

## If you do not have raw data, you can use summary statistics
(x1 <- efa.ekc(sample.cov = myCovMat, sample.nobs = N, plot = FALSE))
(x2 <- efa.ekc(sample.cov = myCorMat, sample.nobs = N, plot = FALSE))
```

`epcEquivCheck` *EPC Equivalence Feasibility Check for Standardized Parameters*

Description

Performs an EPC-based feasibility check to assess whether a set of standardized population parameters defines a valid population covariance matrix and whether trivially misspecified parameters remain within a user-defined smallest effect size of interest (SESOI). Feasibility is evaluated by constructing implied population models under targeted parameter perturbations and examining EPC behavior using `epcEquivFit`.

Usage

```
epcEquivCheck(lavaanObj, minRelEffect = 0.75, stdLoad = 0.4, cor = 0.1,
  corLatent = NULL, corResidual = NULL, stdBeta = 0.1)
```

Arguments

`lavaanObj`	A fitted `lavaan` object representing the target model.
`minRelEffect`	A scalar in (0, 1) specifying the minimum relative magnitude of the standardized perturbation to be evaluated. The default value of 0.75 indicates that perturbations equal to 75\ the SESOI are treated as trivial. If EPCs exceed the SESOI under such perturbations, EPC equivalence testing is not recommended.
`stdLoad`	Standardized factor loading used to define the SESOI for loading misspecifications.
`cor`	Standardized correlation used as a default SESOI for covariance misspecifications. This value is used for both latent and residual covariances unless overridden by `corLatent` or `corResidual`.
`corLatent`	Standardized latent factor correlation used to define the SESOI for latent covariance misspecifications. If `NULL`, defaults to `cor`.
`corResidual`	Standardized residual correlation used to define the SESOI for indicator residual covariance misspecifications. If `NULL`, defaults to `cor`.
`stdBeta`	Standardized regression coefficient used to define the SESOI for structural misspecifications.

Details

This function focuses on standardized parameters and supports recursive SEMs with continuous indicators only.

The procedure first checks whether the standardized parameters imply a positive definite population covariance matrix. It then evaluates EPC behavior under both positive and negative trivial misspecifications by repeatedly constructing implied population covariance matrices with perturbed parameters (`minRelEffect` × SESOI), refitting the model, and re-evaluating EPCs.

Models with categorical indicators, formative indicators, or multiple-group structures are not supported.

Value

An object of class `"epcEquivCheckStd"` containing:

- `feasible`: Logical indicator of whether a valid standardized population model exists.
- `any_M`: Logical indicator of whether any EPC exceeded the SESOI under the evaluated misspecifications.
- `recommendation`: Character string summarizing feasibility (e.g., `"RECOMMENDED"`, `"NOT RECOMMENDED"`).
- `M_table`: Data frame summarizing EPCs exceeding the SESOI, if any.
- `testeffect`: Data frame reporting the smallest tested standardized perturbations in each direction.

See Also

`epcEquivFit`

Examples

```
library(lavaan)

one.model <- ' onefactor  =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 '
fit <- cfa(one.model, data = HolzingerSwineford1939)

epcEquivCheck(fit)
```

`epcEquivFit` *EPC Equivalence Fit Evaluation Using Modification Indices*

Description

Evaluates model fit from an equivalence-testing perspective by aggregating local EPC-based diagnostics into a global, fit-style assessment. The procedure combines modification indices (MI), expected parameter changes (EPC), statistical power, and confidence intervals relative to a smallest effect size of interest (SESOI).

Usage

```
epcEquivFit(lavaanObj, stdLoad = 0.4, cor = 0.1, corLatent = NULL,
  corResidual = NULL, stdBeta = 0.1, stdIntcept = 0.2, stdSesoi = NULL,
  sesoi = NULL, cilevel = 0.9, ...)

## S3 method for class 'epcequivfit.data.frame'
summary(object, ..., top = 5, ssv = FALSE)
```

Arguments

`lavaanObj`	A fitted `lavaan` object used to evaluate model fit.
`stdLoad`	Standardized factor loading defining the SESOI for loading misspecifications. Default is 0.4.
`cor`	Default standardized correlation defining the SESOI for covariance misspecifications. Used for both latent and residual covariances unless overridden.
`corLatent`	Standardized latent factor correlation defining the SESOI for latent covariance misspecifications. If `NULL`, defaults to `cor`.
`corResidual`	Standardized residual correlation defining the SESOI for residual covariance misspecifications. If `NULL`, defaults to `cor`.
`stdBeta`	Standardized regression coefficient defining the SESOI for structural misspecifications. Default is 0.1.
`stdIntcept`	Standardized intercept (Cohen's *d*) defining the SESOI for intercept misspecifications. Default is 0.2.
`stdSesoi`	Optional vector of standardized SESOI values. If provided, overrides operator-specific SESOI definitions.
`sesoi`	Optional vector of unstandardized SESOI values. If provided, overrides `stdSesoi` and all operator-specific SESOI arguments.
`cilevel`	Confidence level for EPC confidence intervals used in CI-based equivalence testing.
`...`	Additional arguments passed to `modificationIndices`.
`object`	An object returned by `epcEquivFit`.
`top`	Number of top-ranked EPCs to display.
`ssv`	Logical; whether to include power-based diagnostics.

Details

Two complementary local decision rules are implemented:

Method 1 (Power-based; Saris, Satorra, & van der Veld, 2009). Modification indices, statistical power, and EPC magnitude are jointly evaluated (the J-rule) to classify fixed parameters as misspecified, not misspecified, or inconclusive.

Method 2 (CI-based equivalence testing). Confidence intervals of EPCs are compared against a trivial misspecification region defined by the SESOI to determine whether fixed parameters are substantially misspecified, trivially misspecified, underpowered, or inconclusive.

The resulting local classifications are returned in a single data frame and can be summarized to yield a global equivalence-style fit evaluation.

This function provides a local-to-global equivalence-based alternative to traditional exact-fit evaluation. It is designed to assess whether fixed parameters are substantively misspecified relative to a SESOI, rather than whether a model fits exactly.

Models with categorical indicators or unsupported constraints may not be fully supported.

Value

A data frame with one row per fixed parameter, containing:

1. Parameter identifiers: `lhs`, `op`, `rhs`, and `group`.
2. Modification index (`mi`) and expected parameter change estimates (`epc`).
3. Unstandardized and standardized smallest effect size of interest values (`sesoi`, `std.sesoi`).
4. Power-based decision (`decision.pow`) and related diagnostics, including whether the modification index is statistically significant (`significant.mi`) and whether the misfit at the SESOI has power greater than 0.80 (`high.power`). Decision labels are: M = Substantially Misspecified, I = Inconclusive, NM = Trivially Misspecified, EPC:M = Substantially Misspecified based on EPC information, EPC:NM = Trivially Misspecified based on EPC information.
5. EPC-related statistics, including the standard error of the EPC (`se.epc`), confidence interval bounds for the EPC (`lower.epc`, `upper.epc`), and confidence interval bounds for the standardized EPC (`lower.std.epc`, `upper.std.epc`).
6. Confidence-interval–based equivalence decision (`decision.ci`), with labels: M = Substantially Misspecified (EPC exceeds the SESOI), I = Inconclusive, NM = Trivially Misspecified, U = Underpowered (CI too wide to evaluate equivalence relative to the SESOI).

References

Saris, W. E., Satorra, A., & van der Veld, W. M. (2009). Testing structural equation models or detection of misspecifications? *Structural Equation Modeling, 16*(4), 561–582.

See Also

`epcEquivCheck`

Examples

```
library(lavaan)

one.model <- ' onefactor  =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 '
fit <- cfa(one.model, data = HolzingerSwineford1939)
out <- epcEquivFit(fit)
out
summary(out)
```

`exLong` *Simulated Data set to Demonstrate Longitudinal Measurement Invariance*

Description

A simulated data set with 1 factors with 3 indicators in three timepoints

Usage

```
exLong
```

Format

A `data.frame` with 200 observations of 10 variables.

sex Sex of respondents

y1t1 Indicator 1 in Time 1

y2t1 Indicator 2 in Time 1

y3t1 Indicator 3 in Time 1

y1t2 Indicator 1 in Time 2

y2t2 Indicator 2 in Time 2

y3t2 Indicator 3 in Time 2

y1t3 Indicator 1 in Time 3

y2t3 Indicator 2 in Time 3

y3t3 Indicator 3 in Time 3

Source

Data were generated using the `simsem` package.

Examples

```
head(exLong)
```

`findRMSEApower` *Find the statistical power based on population RMSEA*

Description

Find the proportion of the samples from the sampling distribution of RMSEA in the alternative hypothesis rejected by the cutoff dervied from the sampling distribution of RMSEA in the null hypothesis. This function can be applied for both test of close fit and test of not-close fit (MacCallum, Browne, & Suguwara, 1996)

Usage

```
findRMSEApower(rmsea0, rmseaA, df, n, alpha = 0.05, group = 1)
```

Arguments

`rmsea0`	Null RMSEA
`rmseaA`	Alternative RMSEA
`df`	Model degrees of freedom
`n`	Sample size of a dataset
`alpha`	Alpha level used in power calculations
`group`	The number of group that is used to calculate RMSEA.

Details

This function find the proportion of sampling distribution derived from the alternative RMSEA that is in the critical region derived from the sampling distribution of the null RMSEA. If `rmseaA` is greater than `rmsea0`, the test of close fit is used and the critical region is in the right hand side of the null sampling distribution. On the other hand, if `rmseaA` is less than `rmsea0`, the test of not-close fit is used and the critical region is in the left hand side of the null sampling distribution (MacCallum, Browne, & Suguwara, 1996).

There is also a Shiny app called "power4SEM" that provides a graphical user interface for this functionality (Jak et al., in press). It can be accessed at `https://sjak.shinyapps.io/power4SEM/`.

Author(s)

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

References

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods, 1*(2), 130–149. doi:10.1037/1082989X.1.2.130

Jak, S., Jorgensen, T. D., Verdam, M. G., Oort, F. J., & Elffers, L. (2021). Analytical power calculations for structural equation modeling: A tutorial and Shiny app. *Behavior Research Methods, 53*, 1385–1406. doi:10.3758/s13428020014790

See Also

- `plotRMSEApower()` to plot the statistical power based on population RMSEA given the sample size
- `plotRMSEAdist()` to visualize the RMSEA distributions
- `findRMSEAsamplesize()` to find the minium sample size for a given statistical power based on population RMSEA

Examples

```
findRMSEApower(rmsea0 = .05, rmseaA = .08, df = 20, n = 200)
```

`findRMSEApowernested` *Find power given a sample size in nested model comparison*

Description

Find the sample size that the power in rejection the samples from the alternative pair of RMSEA is just over the specified power.

Usage

```
findRMSEApowernested(rmsea0A = NULL, rmsea0B = NULL, rmsea1A,
  rmsea1B = NULL, dfA, dfB, n, alpha = 0.05, group = 1)
```

Arguments

`rmsea0A`	The H_0 baseline RMSEA
`rmsea0B`	The H_0 alternative RMSEA (trivial misfit)
`rmsea1A`	The H_1 baseline RMSEA
`rmsea1B`	The H_1 alternative RMSEA (target misfit to be rejected)
`dfA`	degree of freedom of the more-restricted model
`dfB`	degree of freedom of the less-restricted model
`n`	Sample size
`alpha`	The alpha level
`group`	The number of group in calculating RMSEA

Author(s)

Bell Clinton

Pavel Panko (Texas Tech University; `<pavel.panko@ttu.edu>`)

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

References

MacCallum, R. C., Browne, M. W., & Cai, L. (2006). Testing differences between nested covariance structure models: Power analysis and null hypotheses. *Psychological Methods, 11*(1), 19–35. doi:10.1037/1082989X.11.1.19

See Also

- `plotRMSEApowernested()` to plot the statistical power for nested model comparison based on population RMSEA given the sample size
- `findRMSEAsamplesizenested()` to find the minium sample size for a given statistical power in nested model comparison based on population RMSEA

Examples

```
findRMSEApowernested(rmsea0A = 0.06, rmsea0B = 0.05, rmsea1A = 0.08,
                     rmsea1B = 0.05, dfA = 22, dfB = 20, n = 200,
                     alpha = 0.05, group = 1)
```

`findRMSEAsamplesize` *Find the minimum sample size for a given statistical power based on population RMSEA*

Description

Find the minimum sample size for a specified statistical power based on population RMSEA. This function can be applied for both test of close fit and test of not-close fit (MacCallum, Browne, & Suguwara, 1996)

Usage

```
findRMSEAsamplesize(rmsea0, rmseaA, df, power = 0.8, alpha = 0.05,
  group = 1)
```

Arguments

`rmsea0`	Null RMSEA
`rmseaA`	Alternative RMSEA
`df`	Model degrees of freedom
`power`	Desired statistical power to reject misspecified model (test of close fit) or retain good model (test of not-close fit)
`alpha`	Alpha level used in power calculations
`group`	The number of group that is used to calculate RMSEA.

Details

This function find the minimum sample size for a specified power based on an iterative routine. The sample size keep increasing until the calculated power from `findRMSEApower()` function is just over the specified power. If `group` is greater than 1, the resulting sample size is the sample size per group.

Author(s)

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

References

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods, 1*(2), 130–149. doi:10.1037/1082989X.1.2.130

Jak, S., Jorgensen, T. D., Verdam, M. G., Oort, F. J., & Elffers, L. (2021). Analytical power calculations for structural equation modeling: A tutorial and Shiny app. *Behavior Research Methods, 53*, 1385–1406. doi:10.3758/s13428020014790

See Also

- `plotRMSEApower()` to plot the statistical power based on population RMSEA given the sample size
- `plotRMSEAdist()` to visualize the RMSEA distributions
- `findRMSEApower()` to find the statistical power based on population RMSEA given a sample size

Examples

```
findRMSEAsamplesize(rmsea0 = .05, rmseaA = .08, df = 20, power = 0.80)
```

`findRMSEAsamplesizenested`

Find sample size given a power in nested model comparison

Description

Find the sample size that the power in rejection the samples from the alternative pair of RMSEA is just over the specified power.

Usage

```
findRMSEAsamplesizenested(rmsea0A = NULL, rmsea0B = NULL, rmsea1A,
  rmsea1B = NULL, dfA, dfB, power = 0.8, alpha = 0.05, group = 1)
```

Arguments

`rmsea0A`	The H_0 baseline RMSEA
`rmsea0B`	The H_0 alternative RMSEA (trivial misfit)
`rmsea1A`	The H_1 baseline RMSEA
`rmsea1B`	The H_1 alternative RMSEA (target misfit to be rejected)
`dfA`	degree of freedom of the more-restricted model.
`dfB`	degree of freedom of the less-restricted model.
`power`	The desired statistical power.
`alpha`	The alpha level.
`group`	The number of group in calculating RMSEA.

Author(s)

Bell Clinton

Pavel Panko (Texas Tech University; <pavel.panko@ttu.edu>)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

MacCallum, R. C., Browne, M. W., & Cai, L. (2006). Testing differences between nested covariance structure models: Power analysis and null hypotheses. *Psychological Methods, 11*(1), 19–35. doi:10.1037/1082989X.11.1.19

See Also

- `plotRMSEApowernested()` to plot the statistical power for nested model comparison based on population RMSEA given the sample size
- `findRMSEApowernested()` to find the power for a given sample size in nested model comparison based on population RMSEA

Examples

```
findRMSEAsamplesizenested(rmsea0A = 0, rmsea0B = 0, rmsea1A = 0.06,
                          rmsea1B = 0.05, dfA = 22, dfB = 20, power = 0.80,
                          alpha = .05, group = 1)
```

`FitDiff-class` *Class For Representing A Template of Model Fit Comparisons*

Description

This class contains model fit measures and model fit comparisons among multiple models

Usage

```
## S4 method for signature 'FitDiff'
show(object)

## S4 method for signature 'FitDiff'
summary(object, fit.measures = "default", nd = 3,
  tag = "†")
```

Arguments

`object`	object of class `FitDiff`
`fit.measures`	`character` vector naming fit indices the user can request from `lavaan::fitMeasures()`. If `"default"`, the fit measures will be `c("chisq", "df", "pvalue", "cfi", "tli", "rmsea", "srmr", "aic", "bic")`. If `"all"`, all available fit measures will be returned.
`nd`	number of digits printed
`tag`	single `character` used to flag the model preferred by each fit index. To omit tags, set to `NULL` or `NA`.

Slots

`name` `character`. The name of each model

`model.class` `character`. One class to which each model belongs

`nested` `data.frame`. Model fit comparisons between adjacently nested models that are ordered by their degrees of freedom (*df*)

`fit` `data.frame`. Fit measures of all models specified in the `name` slot, ordered by their *df*

`fit.diff` `data.frame`. Sequential differences in fit measures in the `fit` slot

Objects from the Class

Objects can be created via the `compareFit()` function.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

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

See Also

`compareFit()`; `clipboard()`

Examples

```
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '
fit.config <- cfa(HS.model, data = HolzingerSwineford1939, group = "school")
## invariance constraints
fit.metric <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
                  group.equal = "loadings")
fit.scalar <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
                  group.equal = c("loadings","intercepts"))
fit.strict <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
                  group.equal = c("loadings","intercepts","residuals"))
measEqOut <- compareFit(fit.config, fit.metric, fit.scalar, fit.strict)
summary(measEqOut)
summary(measEqOut, fit.measures = "all")
```

```
summary(measEqOut, fit.measures = c("aic", "bic"))

if(interactive()){
## Save results to a file
saveFile(measEqOut, file = "measEq.txt")

## Copy to a clipboard
clipboard(measEqOut)
}
```

`fmi` *Fraction of Missing Information.*

Description

This function estimates the Fraction of Missing Information (FMI) for summary statistics of each variable, using either an incomplete data set or a list of imputed data sets.

Usage

```
fmi(data, method = "saturated", group = NULL, ords = NULL,
  varnames = NULL, exclude = NULL, return.fit = FALSE)
```

Arguments

`data`	Either a single `data.frame` with incomplete observations, or a `list` of imputed data sets.
`method`	character. If `"saturated"` or `"sat"` (default), the model used to estimate FMI is a freely estimated covariance matrix and mean vector for numeric variables, and/or polychoric correlations and thresholds for ordered categorical variables, for each group (if applicable). If `"null"`, only means and variances are estimated for numeric variables, and/or thresholds for ordered categorical variables (i.e., covariances and/or polychoric/polyserial correlations are constrained to zero). See **Details** for more information.
`group`	`character`. The optional name of a grouping variable, to request FMI in each group.
`ords`	Optional `character` vector naming ordered-categorical variables, if they are not already stored as class `ordered` in `data`.
`varnames`	Optional `character` vector of variable names, to calculate FMI for a subset of variables in `data`. By default, all numeric and `ordered=` variables will be included, unless `data=` is a single incomplete `data.frame`, in which case only numeric variables can be used with FIML estimation. Other variable types will be removed.
`exclude`	Optional `character` vector naming variables to exclude from the analysis.
`return.fit`	logical. If `TRUE`, the fitted lavaan::lavaan or lavaan.mi::lavaan.mi model is returned, so FMI can be found from `summary(..., fmi=TRUE)`.

Details

The function estimates a saturated model with `lavaan::lavaan()` for a single incomplete data set using FIML, or with `lavaan.mi::lavaan.mi()` for a list of imputed data sets. If method = `"saturated"`, FMI will be estiamted for all summary statistics, which could take a lot of time with big data sets. If method = `"null"`, FMI will only be estimated for univariate statistics (e.g., means, variances, thresholds). The saturated model gives more reliable estimates, so it could also help to request a subset of variables from a large data set.

Value

`fmi()` returns a list with at least 2 of the following:

`Covariances`	A list of symmetric matrices: (1) the estimated/pooled covariance matrix, or a list of group-specific matrices (if applicable) and (2) a matrix of FMI, or a list of group-specific matrices (if applicable). Only available if `method = "saturated"`. When `method="cor"`, this element is replaced by `Correlations`.
`Variances`	The estimated/pooled variance for each numeric variable. Only available if `method = "null"` (otherwise, it is on the diagonal of Covariances).
`Means`	The estimated/pooled mean for each numeric variable.
`Thresholds`	The estimated/pooled threshold(s) for each ordered-categorical variable.

Author(s)

Mauricio Garnier Villarreal (Vrije Universiteit Amsterdam; `<m.garniervillarreal@vu.nl>`)

Terrence Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys*. New York, NY: Wiley.

Savalei, V. & Rhemtulla, M. (2012). On obtaining estimates of the fraction of missing information from full information maximum likelihood. *Structural Equation Modeling, 19*(3), 477–494. doi:10.1080/10705511.2012.687669

Wagner, J. (2010). The fraction of missing information as a tool for monitoring the quality of survey data. *Public Opinion Quarterly, 74*(2), 223–243. doi:10.1093/poq/nfq007

Examples

```
HSMiss <- HolzingerSwineford1939[ , c(paste("x", 1:9, sep = ""),
                                      "ageyr","agemo","school")]
set.seed(12345)
HSMiss$x5 <- ifelse(HSMiss$x5 <= quantile(HSMiss$x5, .3), NA, HSMiss$x5)
age <- HSMiss$ageyr + HSMiss$agemo/12
HSMiss$x9 <- ifelse(age <= quantile(age, .3), NA, HSMiss$x9)

## calculate FMI (using FIML, provide partially observed data set)
(out1 <- fmi(HSMiss, exclude = "school"))
(out2 <- fmi(HSMiss, exclude = "school", method = "null"))
(out3 <- fmi(HSMiss, varnames = c("x5","x6","x7","x8","x9")))
```

```
(out4 <- fmi(HSMiss, method = "cor", group = "school")) # correlations by group

## significance tests in lavaan(.mi) object
out5 <- fmi(HSMiss, method = "cor", return.fit = TRUE)
summary(out5) # factor loading == SD, covariance = correlation

if(requireNamespace("lavaan.mi")){
  ## ordered-categorical data
  data(binHS5imps, package = "lavaan.mi")

  ## calculate FMI, using list of imputed data sets
  fmi(binHS5imps, group = "school")
}
```

`goricaSEM` *Wrapper for* `goric.lavaan()` *from the* `restriktor` *package*

Description

The `goricaSEM()` function is an interface to `restriktor::goric.lavaan()`, allowing users to perform generalized order-restricted information criterion approximation (GORICA) analysis specifically for structural equation models fitted using the **lavaan** package.

Usage

```
goricaSEM(object, ..., hypotheses = NULL, comparison = NULL,
  type = "gorica", standardized = FALSE, debug = FALSE)
```

Arguments

`object`	A lavaan::lavaan object.
`...`	Additional arguments passed to `restriktor::goric.lavaan()`.
`hypotheses`	A named `list` of hypotheses to test. See **Details** for information on how to specify hypotheses.
`comparison`	A `character` string specifying the type of comparison. Options are `"unconstrained"`, `"complement"`, or `"none"`. Default behavior depends on the number of hypotheses.
`type`	A `character` string indicating the type of analysis, either `"gorica"` (default) or `"goricac"`.
`standardized`	`logical` indicating whether standardized estimates are used in the analysis. Defaults to `FALSE`.
`debug`	`logical` indicating whether to print debugging information. Defaults to `FALSE`.

Details

This function is designed as a wrapper for the `restriktor::goric.lavaan()` function. It calculates GORICA values and weights, which can be used to compare models or hypotheses under inequality constraints.

The `hypotheses=` argument allows users to specify constraints in text-based syntax or matrix notation. For text-based syntax, constraints are specified as a string (e.g., `"a1 > a2"`). For matrix notation, a named list with `$constraints`, `$rhs`, and `$neq` elements can be provided.

The `comparison=` argument determines whether the specified hypothesis is compared against its `"complement"`, the `"unconstrained"` model, or neither (`"none"`).

Value

A `list` containing the results of the `goric.lavaan` function, including:

- The log-likelihood.
- Penalty term.
- GORIC(A) values and weights.
- Relative GORIC(A) weights.

Author(s)

Leonard Vanbrabant and Rebecca Kuiper

References

Kuiper, R. M., Hoijtink, H., & Silvapulle, M. J. (2011). An Akaike-type information criterion for model selection under inequality constraints. *Biometrika, 98*(2), 495–501. doi:10.1093/biomet/asr002

Vanbrabant, L., Van Loey, N., & Kuiper, R. M. (2020). Evaluating a theory-based hypothesis against its complement using an AIC-type information criterion with an application to facial burn injury. *Psychological Methods, 25*(2), 129–142. doi:10.1037/met0000238

See Also

`restriktor::goric.lavaan()`

Examples

```
## Example: Perform GORICA analysis on a lavaan model
library(lavaan)
library(restriktor)

## Define the SEM model
model <- '
  ind60 =~ x1 + x2 + x3
  dem60 =~ y1 + a1*y2 + b1*y3 + c1*y4
  dem65 =~ y5 + a2*y6 + b2*y7 + c2*y8
  dem60 ~ ind60
```

```
  dem65 ~ ind60 + dem60
  y1 ~~ y5
  y2 ~~ y4 + y6
  y3 ~~ y7
  y4 ~~ y8
  y6 ~~ y8
'

## Fit the model
data(PoliticalDemocracy)
fit <- sem(model, data = PoliticalDemocracy)

## Define hypotheses
myHypothesis <- 'a1 > a2, b1 > b2, c1 > c2'

## Perform GORICA analysis
result <- goricaSEM(fit, hypotheses = list(H1 = myHypothesis),
                    standardized = FALSE, comparison = "complement",
                    type = "gorica")

## Print result
print(result)
```

`htmt` *Assessing Discriminant Validity using Heterotrait–Monotrait Ratio*

Description

This function assesses discriminant validity through the heterotrait-monotrait ratio (HTMT) of the correlations (Henseler, Ringlet & Sarstedt, 2015). Specifically, it assesses the arithmetic (Henseler et al.,) or geometric (Roemer et al., 2021) mean correlation among indicators across constructs (i.e. heterotrait–heteromethod correlations) relative to the geometric-mean correlation among indicators within the same construct (i.e. monotrait–heteromethod correlations). The resulting HTMT(2) values are interpreted as estimates of inter-construct correlations. Absolute values of the correlations are recommended to calculate the HTMT matrix, and are required to calculate HTMT2. Correlations are estimated using the `lavaan::lavCor()` function.

Usage

```
htmt(model, data = NULL, sample.cov = NULL, missing = "listwise",
  ordered = NULL, absolute = TRUE, htmt2 = TRUE)
```

Arguments

`model`	lavaan `lavaan::model.syntax()` of a confirmatory factor analysis model where at least two factors are required for indicators measuring the same construct.
`data`	A `data.frame` or data `matrix`

`sample.cov`	A covariance or correlation matrix can be used, instead of `data=`, to estimate the HTMT values.
`missing`	If `"listwise"`, cases with missing values are removed listwise from the data frame. If `"direct"` or `"ml"` or `"fiml"` and the estimator is maximum likelihood, an EM algorithm is used to estimate the unrestricted covariance matrix (and mean vector). If `"pairwise"`, pairwise deletion is used. If `"default"`, the value is set depending on the estimator and the mimic option (see details in `lavaan::lavCor()`).
`ordered`	Character vector. Only used if object is a `data.frame`. Treat these variables as ordered (ordinal) variables. Importantly, all other variables will be treated as numeric (unless `is.ordered` in `data=`). See also `lavaan::lavCor()`.
`absolute`	`logical` indicating whether HTMT values should be estimated based on absolute correlations (default is `TRUE`). This is recommended for HTMT but required for HTMT2 (so silently ignored).
`htmt2`	`logical` indicating whether to use the geometric mean (default, appropriate for congeneric indicators) or arithmetic mean (which assumes tau-equivalence).

Value

A matrix showing HTMT(2) values (i.e., discriminant validity) between each pair of factors.

Author(s)

Ylenio Longo (University of Nottingham; `<yleniolongo@gmail.com>`)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Henseler, J., Ringle, C. M., & Sarstedt, M. (2015). A new criterion for assessing discriminant validity in variance-based structural equation modeling. *Journal of the Academy of Marketing Science, 43*(1), 115–135. doi:10.1007/s1174701404038

Roemer, E., Schuberth, F., & Henseler, J. (2021). HTMT2—An improved criterion for assessing discriminant validity in structural equation modeling. *Industrial Management & Data Systems, 121*(21), 2637–2650. doi:10.1108/IMDS0220210082

Voorhees, C. M., Brady, M. K., Calantone, R., & Ramirez, E. (2016). Discriminant validity testing in marketing: An analysis, causes for concern, and proposed remedies. *Journal of the Academy of Marketing Science, 44*(1), 119–134. doi:10.1007/s1174701504554

Examples

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

dat <- HolzingerSwineford1939[, paste0("x", 1:9)]
htmt(HS.model, dat)

## save covariance matrix
```

```
HS.cov <- cov(HolzingerSwineford1939[, paste0("x", 1:9)])
## HTMT using arithmetic mean
htmt(HS.model, sample.cov = HS.cov, htmt2 = FALSE)
```

`imposeStart` *Specify starting values from a lavaan output*

Description

This function will save the parameter estimates of a lavaan output and impose those parameter estimates as starting values for another analysis model. The free parameters with the same names or the same labels across two models will be imposed the new starting values. This function may help to increase the chance of convergence in a complex model (e.g., multitrait-multimethod model or complex longitudinal invariance model).

Usage

```
imposeStart(out, expr, silent = TRUE)
```

Arguments

`out`	The `lavaan` output that users wish to use the parameter estimates as staring values for an analysis model
`expr`	The original code that users use to run a lavaan model
`silent`	Logical to print the parameter table with new starting values

Value

A fitted lavaan model

Author(s)

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

Examples

```
## The following example show that the longitudinal weak invariance model
## using effect coding was not convergent with three time points but convergent
## with two time points. Thus, the parameter estimates from the model with
## two time points are used as starting values of the three time points.
## The model with new starting values is convergent properly.

weak2time <- '
# Loadings
f1t1 =~ LOAD1*y1t1 + LOAD2*y2t1 + LOAD3*y3t1
    f1t2 =~ LOAD1*y1t2 + LOAD2*y2t2 + LOAD3*y3t2
```

```
# Factor Variances
f1t1 ~~ f1t1
f1t2 ~~ f1t2

# Factor Covariances
f1t1 ~~ f1t2

# Error Variances
y1t1 ~~ y1t1
y2t1 ~~ y2t1
y3t1 ~~ y3t1
y1t2 ~~ y1t2
y2t2 ~~ y2t2
y3t2 ~~ y3t2

# Error Covariances
y1t1 ~~ y1t2
y2t1 ~~ y2t2
y3t1 ~~ y3t2

# Factor Means
f1t1 ~ NA*1
f1t2 ~ NA*1

# Measurement Intercepts
y1t1 ~ INT1*1
y2t1 ~ INT2*1
y3t1 ~ INT3*1
y1t2 ~ INT4*1
y2t2 ~ INT5*1
y3t2 ~ INT6*1

# Constraints for Effect-coding Identification
LOAD1 == 3 - LOAD2 - LOAD3
INT1 == 0 - INT2 - INT3
INT4 == 0 - INT5 - INT6
'
model2time <- lavaan(weak2time, data = exLong)

weak3time <- '
# Loadings
f1t1 =~ LOAD1*y1t1 + LOAD2*y2t1 + LOAD3*y3t1
    f1t2 =~ LOAD1*y1t2 + LOAD2*y2t2 + LOAD3*y3t2
    f1t3 =~ LOAD1*y1t3 + LOAD2*y2t3 + LOAD3*y3t3

# Factor Variances
f1t1 ~~ f1t1
f1t2 ~~ f1t2
f1t3 ~~ f1t3

# Factor Covariances
f1t1 ~~ f1t2 + f1t3
f1t2 ~~ f1t3
```

```
# Error Variances
y1t1 ~~ y1t1
y2t1 ~~ y2t1
y3t1 ~~ y3t1
y1t2 ~~ y1t2
y2t2 ~~ y2t2
y3t2 ~~ y3t2
y1t3 ~~ y1t3
y2t3 ~~ y2t3
y3t3 ~~ y3t3

# Error Covariances
y1t1 ~~ y1t2
y2t1 ~~ y2t2
y3t1 ~~ y3t2
y1t1 ~~ y1t3
y2t1 ~~ y2t3
y3t1 ~~ y3t3
y1t2 ~~ y1t3
y2t2 ~~ y2t3
y3t2 ~~ y3t3

# Factor Means
f1t1 ~ NA*1
f1t2 ~ NA*1
f1t3 ~ NA*1

# Measurement Intercepts
y1t1 ~ INT1*1
y2t1 ~ INT2*1
y3t1 ~ INT3*1
y1t2 ~ INT4*1
y2t2 ~ INT5*1
y3t2 ~ INT6*1
y1t3 ~ INT7*1
y2t3 ~ INT8*1
y3t3 ~ INT9*1

# Constraints for Effect-coding Identification
LOAD1 == 3 - LOAD2 - LOAD3
INT1 == 0 - INT2 - INT3
INT4 == 0 - INT5 - INT6
INT7 == 0 - INT8 - INT9
'
### The following command does not provide convergent result
# model3time <- lavaan(weak3time, data = exLong)

### Use starting values from the model with two time points
model3time <- imposeStart(model2time, lavaan(weak3time, data = exLong))
summary(model3time)
```

`indProd` *Make products of indicators using no centering, mean centering, double-mean centering, or residual centering*

Description

The `indProd` function will make products of indicators using no centering, mean centering, double-mean centering, or residual centering. The `orthogonalize` function is the shortcut of the `indProd` function to make the residual-centered indicators products.

Usage

```
indProd(data, var1, var2, var3 = NULL, match = TRUE, meanC = TRUE,
  residualC = FALSE, doubleMC = TRUE, namesProd = NULL)

orthogonalize(data, var1, var2, var3 = NULL, match = TRUE,
  namesProd = NULL)
```

Arguments

`data`	The desired data to be transformed.
`var1`	Names or indices of the variables loaded on the first factor
`var2`	Names or indices of the variables loaded on the second factor
`var3`	Names or indices of the variables loaded on the third factor (for three-way interaction)
`match`	Specify `TRUE` to use match-paired approach (Marsh, Wen, & Hau, 2004). If `FALSE`, the resulting products are all possible products.
`meanC`	Specify `TRUE` for mean centering the main effect indicator before making the products
`residualC`	Specify `TRUE` for residual centering the products by the main effect indicators (Little, Bovaird, & Widaman, 2006).
`doubleMC`	Specify `TRUE` for centering the resulting products (Lin et. al., 2010)
`namesProd`	The names of resulting products

Value

The original data attached with the products.

Author(s)

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

References

Marsh, H. W., Wen, Z. & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*(3), 275–300. doi:10.1037/1082989X.9.3.275

Lin, G. C., Wen, Z., Marsh, H. W., & Lin, H. S. (2010). Structural equation models of latent interactions: Clarification of orthogonalizing and double-mean-centering strategies. *Structural Equation Modeling, 17*(3), 374–391. doi:10.1080/10705511.2010.488999

Little, T. D., Bovaird, J. A., & Widaman, K. F. (2006). On the merits of orthogonalizing powered and product terms: Implications for modeling interactions among latent variables. *Structural Equation Modeling, 13*(4), 497–519. doi:10.1207/s15328007sem1304_1

See Also

- `probe2WayMC()` For probing the two-way latent interaction when the results are obtained from mean-centering, or double-mean centering.
- `probe3WayMC()` For probing the three-way latent interaction when the results are obtained from mean-centering, or double-mean centering.
- `probe2WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `probe3WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `plotProbe()` Plot the simple intercepts and slopes of the latent interaction.

Examples

```
## Mean centering / two-way interaction / match-paired
dat <- indProd(attitude[ , -1], var1 = 1:3, var2 = 4:6)

## Residual centering / two-way interaction / match-paired
dat2 <- indProd(attitude[ , -1], var1 = 1:3, var2 = 4:6, match = FALSE,
                meanC = FALSE, residualC = TRUE, doubleMC = FALSE)

## Double-mean centering / two-way interaction / match-paired
dat3 <- indProd(attitude[ , -1], var1 = 1:3, var2 = 4:6, match = FALSE,
                meanC = TRUE, residualC = FALSE, doubleMC = TRUE)

## Mean centering / three-way interaction / match-paired
dat4 <- indProd(attitude[ , -1], var1 = 1:2, var2 = 3:4, var3 = 5:6)

## Residual centering / three-way interaction / match-paired
dat5 <- orthogonalize(attitude[ , -1], var1 = 1:2, var2 = 3:4, var3 = 5:6,
                      match = FALSE)

## Double-mean centering / three-way interaction / match-paired
dat6 <- indProd(attitude[ , -1], var1 = 1:2, var2 = 3:4, var3 = 5:6,
                match = FALSE, meanC = TRUE, residualC = TRUE,
                doubleMC = TRUE)
```

```
## To add product-indicators to multiple-imputed data sets

if (requireNamespace("lavaan.mi")) {
  data(HS20imps, package = "lavaan.mi")

  ## apply indProd() to the list of data.frames
  imps2 <- lapply(HS20imps, indProd,
                  var1 = c("x1","x2","x3"), var2 = c("x4","x5","x6"))
  ## verify:
  lapply(imps2, head)
}
```

`kd` *Generate data via the Kaiser-Dickman (1962) algorithm.*

Description

Given a covariance matrix and sample size, generate raw data that correspond to the covariance matrix. Data can be generated to match the covariance matrix exactly, or to be a sample from the population covariance matrix.

Usage

```
kd(covmat, n, type = c("exact", "sample"))
```

Arguments

`covmat`	a symmetric, positive definite covariance matrix
`n`	the sample size for the data that will be generated
`type`	type of data generation. `exact` generates data that exactly correspond to `covmat`. `sample` treats `covmat` as a poulation covariance matrix, generating a sample of size `n`.

Details

By default, R's `cov()` function divides by `n`-1. The data generated by this algorithm result in a covariance matrix that matches `covmat`, but you must divide by `n` instead of `n`-1.

Value

`kd` returns a data matrix of dimension `n` by `nrow(covmat)`.

Author(s)

Ed Merkle (University of Missouri; `<merklee@missouri.edu>`)

References

Kaiser, H. F. and Dickman, K. (1962). Sample and population score matrices and sample correlation matrices from an arbitrary population correlation matrix. *Psychometrika, 27*(2), 179–182. doi:10.1007/BF02289635

Examples

```
#### First Example

## Get data
dat <- HolzingerSwineford1939[ , 7:15]
hs.n <- nrow(dat)

## Covariance matrix divided by n
hscov <- ((hs.n-1)/hs.n) * cov(dat)

## Generate new, raw data corresponding to hscov
newdat <- kd(hscov, hs.n)

## Difference between new covariance matrix and hscov is minimal
newcov <- (hs.n-1)/hs.n * cov(newdat)
summary(as.numeric(hscov - newcov))

## Generate sample data, treating hscov as population matrix
newdat2 <- kd(hscov, hs.n, type = "sample")

#### Another example

## Define a covariance matrix
covmat <- matrix(0, 3, 3)
diag(covmat) <- 1.5
covmat[2:3,1] <- c(1.3, 1.7)
covmat[3,2] <- 2.1
covmat <- covmat + t(covmat)

## Generate data of size 300 that have this covariance matrix
rawdat <- kd(covmat, 300)

## Covariances are exact if we compute sample covariance matrix by
## dividing by n (vs by n - 1)
summary(as.numeric((299/300)*cov(rawdat) - covmat))

## Generate data of size 300 where covmat is the population covariance matrix
rawdat2 <- kd(covmat, 300)
```

`kurtosis` *Finding excessive kurtosis*

Description

Finding excessive kurtosis (g_2) of an object

Usage

```
kurtosis(object, population = FALSE)
```

Arguments

`object`	A vector used to find a excessive kurtosis
`population`	`TRUE` to compute the parameter formula. `FALSE` to compute the sample statistic formula.

Details

The excessive kurtosis computed by default is g_2, the fourth standardized moment of the empirical distribution of `object`. The population parameter excessive kurtosis γ_2 formula is

$$\gamma_2 = \frac{\mu_4}{\mu_2^2} - 3,$$

where μ_i denotes the i order central moment.

The excessive kurtosis formula for sample statistic g_2 is

$$g_2 = \frac{k_4}{k_2^2} - 3,$$

where k_i are the i order *k*-statistic.

The standard error of the excessive kurtosis is

$$Var(\hat{g}_2) = \frac{24}{N}$$

where N is the sample size.

Value

A value of an excessive kurtosis with a test statistic if the population is specified as `FALSE`

Author(s)

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

References

Weisstein, Eric W. (n.d.). *Kurtosis.* Retrieved from *MathWorld*–A Wolfram Web Resource: `https://mathworld.wolfram.com/Kurtosis.html`

See Also

- `skew()` Find the univariate skewness of a variable
- `mardiaSkew()` Find the Mardia's multivariate skewness of a set of variables
- `mardiaKurtosis()` Find the Mardia's multivariate kurtosis of a set of variables

Examples

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

`lavaan2emmeans` `emmeans` *Support Functions for* `lavaan` *Models*

Description

Provide emmeans support for lavaan objects

Usage

```
recover_data.lavaan(object, lavaan.DV, data = NULL, ...)

emm_basis.lavaan(object, trms, xlev, grid, lavaan.DV, ...)
```

Arguments

`object`	An object of class `lavaan::lavaan()`. See **Details**.
`lavaan.DV`	`character` string naming the variable(s) for which expected marginal means / trends should be produced. A vector of names indicates a multivariate outcome, treated by default as repeated measures.
`data`	An optional `data.frame` without missing values, to be passed when `missing="FIML"` estimation was useed, thus avoiding a reference-grid with missing values.
`...`	Further arguments passed to `emmeans::recover_data.lm` or `emmeans::emm_basis.lm`
`trms`, `xlev`, `grid`	See `emmeans::emm_basis`

Details

Supported DVs: `lavaan.DV` must be an *endogenous variable*, by appearing on the left-hand side of either a regression operator (`"~"`) or an intercept operator (`"~1"`), or both.

`lavaan.DV` can also be a vector of endogenous variable, in which case they will be treated by `emmeans` as a multivariate outcome (often, this indicates repeated measures) represented by an additional factor named `rep.meas` by default. The `mult.name=` argument can be used to overwrite this default name.

Unsupported Models: This functionality does not support the following models:

- Multi-level models are not supported.
- Models not fit to a `data.frame` (i.e., models fit to a covariance matrix).

Dealing with Fixed Parameters: Fixed parameters (set with `lavaan`'s modifiers) are treated as-is: their values are set by the users, and they have a *SE* of 0 (as such, they do not co-vary with any other parameter).

Dealing with Multigroup Models: If a multigroup model is supplied, a factor is added to the reference grid, the name matching the `group` argument supplied when fitting the model. *Note that you must set* `nesting = NULL`.

Dealing with Missing Data: Limited testing suggests that these functions do work when the model was fit to incomplete data.

Dealing with Factors: By default `emmeans` recognizes binary variables (0,1) as a "factor" with two levels (and not a continuous variable). With some clever contrast defenitions it should be possible to get the desired emmeans / contasts. See example below.

Author(s)

Mattan S. Ben-Shachar (Ben-Gurion University of the Negev; `<matanshm@post.bgu.ac.il>`)

Examples

```
## Not run:

  library(lavaan)
  library(emmeans)

  #### Moderation Analysis ####

  mean_sd <- function(x) mean(x) + c(-sd(x), 0, sd(x))

  model <- '
  # regressions
  Sepal.Length ~ b1 * Sepal.Width + b2 * Petal.Length + b3 * Sepal.Width:Petal.Length


  # define mean parameter label for centered math for use in simple slopes
  Sepal.Width ~ Sepal.Width.mean * 1

  # define variance parameter label for centered math for use in simple slopes
  Sepal.Width ~~ Sepal.Width.var * Sepal.Width

  # simple slopes for condition effect
  SD.below := b2 + b3 * (Sepal.Width.mean - sqrt(Sepal.Width.var))
  mean     := b2 + b3 * (Sepal.Width.mean)
  SD.above := b2 + b3 * (Sepal.Width.mean + sqrt(Sepal.Width.var))
  '

  semFit <- sem(model = model,
                data = iris)
```

```
## Compare simple slopes
# From `emtrends`
test(
  emtrends(semFit, ~ Sepal.Width, "Petal.Length",
           lavaan.DV = "Sepal.Length",
           cov.red = mean_sd)
)

# From lavaan
parameterEstimates(semFit, output = "pretty")[13:15, ]
# Identical slopes.
# SEs differ due to lavaan estimating uncertainty of the mean / SD
# of Sepal.Width, whereas emmeans uses the mean+-SD as is (fixed).


#### Latent DV ####

model <- '
LAT1 =~ Sepal.Length + Sepal.Width

LAT1 ~ b1 * Petal.Width + 1 * Petal.Length

Petal.Length ~ Petal.Length.mean * 1

V1 := 1 * Petal.Length.mean + 1 * b1
V2 := 1 * Petal.Length.mean + 2 * b1
'

semFit <- sem(model = model,
              data = iris, std.lv = TRUE)

## Compare emmeans
# From emmeans
test(
  emmeans(semFit, ~ Petal.Width,
          lavaan.DV = "LAT1",
          at = list(Petal.Width = 1:2))
)

# From lavaan
parameterEstimates(semFit, output = "pretty")[15:16, ]
# Identical means.
# SEs differ due to lavaan estimating uncertainty of the mean
# of Petal.Length, whereas emmeans uses the mean as is.

#### Multi-Variate DV ####

model <- '
ind60 =~ x1 + x2 + x3

# metric invariance
dem60 =~ y1 + a*y2 + b*y3 + c*y4
```

```
dem65 =~ y5 + a*y6 + b*y7 + c*y8

# scalar invariance
y1 + y5 ~ d*1
y2 + y6 ~ e*1
y3 + y7 ~ f*1
y4 + y8 ~ g*1

# regressions (slopes differ: interaction with time)
dem60 ~ b1*ind60
dem65 ~ b2*ind60 + NA*1 + Mean.Diff*1

# residual correlations
y1 ~~ y5
y2 ~~ y4 + y6
y3 ~~ y7
y4 ~~ y8
y6 ~~ y8

# conditional mean differences (besides mean(ind60) == 0)
 low := (-1*b2 + Mean.Diff) - (-1*b1) # 1 SD below M
high := (b2 + Mean.Diff) - b1         # 1 SD above M
'

semFit <- sem(model, data = PoliticalDemocracy)


## Compare contrasts
# From emmeans
emmeans(semFit, pairwise ~ rep.meas|ind60,
        lavaan.DV = c("dem60","dem65"),
        at = list(ind60 = c(-1,1)))[[2]]

# From lavaan
parameterEstimates(semFit, output = "pretty")[49:50, ]


#### Multi Group ####

model <- 'x1 ~ c(int1, int2)*1 + c(b1, b2)*ageyr
diff_11 := (int2 + b2*11) - (int1 + b1*11)
diff_13 := (int2 + b2*13) - (int1 + b1*13)
diff_15 := (int2 + b2*15) - (int1 + b1*15)
'
semFit <- sem(model, group = "school", data = HolzingerSwineford1939)


## Compare contrasts
# From emmeans (note `nesting = NULL`)
emmeans(semFit, pairwise ~ school | ageyr, lavaan.DV = "x1",
        at = list(ageyr = c(11, 13, 15)), nesting = NULL)[[2]]

# From lavaan
```

```
  parameterEstimates(semFit, output = "pretty")

  #### Dealing with factors ####

  warpbreaks <- cbind(warpbreaks,
                      model.matrix(~ wool + tension, data = warpbreaks))

  model <- "
  # Split for convenience
  breaks ~ 1
  breaks ~ woolB
  breaks ~ tensionM + tensionH
  breaks ~ woolB:tensionM + woolB:tensionH
  "

  semFit <- sem(model, warpbreaks)

  ## Compare contrasts
  # From lm -> emmeans
  lmFit <- lm(breaks ~ wool * tension, data = warpbreaks)
  lmEM <- emmeans(lmFit, ~ tension + wool)
  contrast(lmEM, method = data.frame(L_all = c(-1, .05, 0.5),
                                     M_H   = c(0, 1, -1)), by = "wool")

  # From lavaan -> emmeans
  lavEM <- emmeans(semFit, ~ tensionM + tensionH + woolB,
                   lavaan.DV = "breaks")
  contrast(lavEM,
           method = list(
             "L_all|A" = c(c(-1, .05, 0.5, 0), rep(0, 4)),
             "M_H  |A" = c(c(0, 1, -1, 0),     rep(0, 4)),
             "L_all|A" = c(rep(0, 4),          c(-1, .05, 0.5, 0)),
             "M_H  |A" = c(rep(0, 4),          c(0, 1, -1, 0))
           ))

## End(Not run)
```

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

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

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

Examples

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

`lrv2ord` *Calculate Population Moments for Ordinal Data Treated as Numeric*

Description

This function calculates ordinal-scale moments implied by LRV-scale moments

Usage

```
lrv2ord(Sigma, Mu, thresholds, cWts)
```

Arguments

`Sigma`	Population covariance `matrix()`, with variable names saved in the `dimnames()` attribute.
`Mu`	Optional `numeric` vector of population means. If missing, all means will be set to zero.
`thresholds`	Either a single `numeric` vector of population thresholds used to discretize each normally distributed variable, or a named `list` of each discretized variable's vector of thresholds. The discretized variables may be a subset of all variables in `Sigma` if the remaining variables are intended to be observed rather than latent normally distributed variables.
`cWts`	Optional (default when missing is to use 0 for the lowest category, followed by successive integers for each higher category). Either a single `numeric` vector of category weights (if they are identical across all variables) or a named `list` of each discretized variable's vector of category weights.

Details

Binary and ordinal data are frequently accommodated in SEM by incorporating a threshold model that links each observed categorical response variable to a corresponding latent response variable that is typically assumed to be normally distributed (Kamata & Bauer, 2008; Wirth & Edwards, 2007). This function can be useful for real-data analysis or for designing Monte Carlo simulations, as described by Jorgensen and Johnson (2022).

Value

A `list` including the LRV-scale population moments (means, covariance matrix, correlation matrix, and thresholds), the category weights, a `data.frame` of implied univariate moments (means, *SD*s, skewness, and excess kurtosis (i.e., in excess of 3, which is the kurtosis of the normal distribution) for discretized data treated as `numeric`, and the implied covariance and correlation matrix of discretized data treated as `numeric`.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

Andrew R. Johnson (Curtin University; `<andrew.johnson@curtin.edu.au>`)

References

Jorgensen, T. D., & Johnson, A. R. (2022). How to derive expected values of structural equation model parameters when treating discrete data as continuous. *Structural Equation Modeling, 29*(4), 639–650. doi:10.1080/10705511.2021.1988609

Kamata, A., & Bauer, D. J. (2008). A note on the relation between factor analytic and item response theory models. *Structural Equation Modeling, 15*(1), 136–153. doi:10.1080/10705510701758406

Wirth, R. J., & Edwards, M. C. (2007). Item factor analysis: Current approaches and future directions. *Psychological Methods, 12*(1), 58–79. doi:10.1037/1082989X.12.1.58

Examples

```
## SCENARIO 1: DIRECTLY SPECIFY POPULATION PARAMETERS

## specify population model in LISREL matrices
Nu <- rep(0, 4)
Alpha <- c(1, -0.5)
Lambda <- matrix(c(1, 1, 0, 0, 0, 0, 1, 1), nrow = 4, ncol = 2,
                 dimnames = list(paste0("y", 1:4), paste0("eta", 1:2)))
Psi <- diag(c(1, .75))
Theta <- diag(4)
Beta <- matrix(c(0, .5, 0, 0), nrow = 2, ncol = 2)

## calculate model-implied population means and covariance matrix
## of latent response variables (LRVs)
IB <- solve(diag(2) - Beta) # to save time and space
Mu_LRV <- Nu + Lambda %*% IB %*% Alpha
Sigma_LRV <- Lambda %*% IB %*% Psi %*% t(IB) %*% t(Lambda) + Theta
```

```
## Specify (unstandardized) thresholds to discretize normally distributed data
## generated from Mu_LRV and Sigma_LRV, based on marginal probabilities
PiList <- list(y1 = c(.25, .5, .25),
               y2 = c(.17, .33, .33, .17),
               y3 = c(.1, .2, .4, .2, .1),
               ## make final variable highly asymmetric
               y4 = c(.33, .25, .17, .12, .08, .05))
sapply(PiList, sum) # all sum to 100%
CumProbs <- sapply(PiList, cumsum)
## unstandardized thresholds
TauList <- mapply(qnorm, p = lapply(CumProbs, function(x) x[-length(x)]),
                  m = Mu_LRV, sd = sqrt(diag(Sigma_LRV)))
for (i in 1:4) names(TauList[[i]]) <- paste0(names(TauList)[i], "|t",
                                             1:length(TauList[[i]]))

## assign numeric weights to each category (optional, see default)
NumCodes <- list(y1 = c(-0.5, 0, 0.5), y2 = 0:3, y3 = 1:5, y4 = 1:6)


## Calculate Population Moments for Numerically Coded Ordinal Variables
lrv2ord(Sigma = Sigma_LRV, Mu = Mu_LRV, thresholds = TauList, cWts = NumCodes)


## SCENARIO 2: USE ESTIMATED PARAMETERS AS POPULATION

data(datCat) # already stored as c("ordered","factor")
fit <- cfa(' f =~ 1*u1 + 1*u2 + 1*u3 + 1*u4 ', data = datCat)
lrv2ord(Sigma = fit, thresholds = fit) # use same fit for both
## or use estimated thresholds with specified parameters, but note that
## lrv2ord() will only extract standardized thresholds
dimnames(Sigma_LRV) <- list(paste0("u", 1:4), paste0("u", 1:4))
lrv2ord(Sigma = cov2cor(Sigma_LRV), thresholds = fit)
```

`mardiaKurtosis` *Finding Mardia's multivariate kurtosis*

Description

Finding Mardia's multivariate kurtosis of multiple variables

Usage

```
mardiaKurtosis(dat, use = "everything")
```

Arguments

`dat`	The target matrix or data frame with multiple variables
`use`	Missing data handling method from the `stats::cov()` function.

Details

The Mardia's multivariate kurtosis formula (Mardia, 1970) is

$$b_{2,d} = \frac{1}{n} \sum_{i=1}^{n} \left[\left(\boldsymbol{X}_i - \bar{\boldsymbol{X}} \right)' \boldsymbol{S}^{-1} \left(\boldsymbol{X}_i - \bar{\boldsymbol{X}} \right) \right]^2,$$

where d is the number of variables, X is the target dataset with multiple variables, n is the sample size, $\boldsymbol{S}$ is the sample covariance matrix of the target dataset, and $\bar{\boldsymbol{X}}$ is the mean vectors of the target dataset binded in n rows. When the population multivariate kurtosis is normal, the $b_{2,d}$ is asymptotically distributed as normal distribution with the mean of $d(d+2)$ and variance of $8d(d+2)/n$.

Value

A value of a Mardia's multivariate kurtosis with a test statistic

Author(s)

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

References

Mardia, K. V. (1970). Measures of multivariate skewness and kurtosis with applications. *Biometrika, 57*(3), 519–530. doi:10.2307/2334770

See Also

- `skew()` Find the univariate skewness of a variable
- `kurtosis()` Find the univariate excessive kurtosis of a variable
- `mardiaSkew()` Find the Mardia's multivariate skewness of a set of variables

Examples

```
library(lavaan)
mardiaKurtosis(HolzingerSwineford1939[ , paste0("x", 1:9)])
```

`mardiaSkew` *Finding Mardia's multivariate skewness*

Description

Finding Mardia's multivariate skewness of multiple variables

Usage

```
mardiaSkew(dat, use = "everything")
```

Arguments

`dat`	The target matrix or data frame with multiple variables
`use`	Missing data handling method from the `stats::cov()` function.

Details

The Mardia's multivariate skewness formula (Mardia, 1970) is

$$b_{1,d} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\left(\boldsymbol{X}_i - \bar{\boldsymbol{X}} \right)' \boldsymbol{S}^{-1} \left(\boldsymbol{X}_j - \bar{\boldsymbol{X}} \right) \right]^3,$$

where d is the number of variables, X is the target dataset with multiple variables, n is the sample size, $\boldsymbol{S}$ is the sample covariance matrix of the target dataset, and $\bar{\boldsymbol{X}}$ is the mean vectors of the target dataset binded in n rows. When the population multivariate skewness is normal, the $\frac{n}{6} b_{1,d}$ is asymptotically distributed as χ^2 distribution with $d(d+1)(d+2)/6$ degrees of freedom.

Value

A value of a Mardia's multivariate skewness with a test statistic

Author(s)

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

References

Mardia, K. V. (1970). Measures of multivariate skewness and kurtosis with applications. *Biometrika, 57*(3), 519–530. doi:10.2307/2334770

See Also

- `skew()` Find the univariate skewness of a variable
- `kurtosis()` Find the univariate excessive kurtosis of a variable
- `mardiaKurtosis()` Find the Mardia's multivariate kurtosis of a set of variables

Examples

```
library(lavaan)
mardiaSkew(HolzingerSwineford1939[ , paste0("x", 1:9)])
```

`maximalRelia` *Calculate maximal reliability*

Description

Calculate maximal reliability of a scale

Usage

```
maximalRelia(object, omit.imps = c("no.conv", "no.se"))
```

Arguments

`object`	A lavaan::lavaan or lavaan.mi::lavaan.mi object, expected to contain only exogenous common factors (i.e., a CFA model).
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However, gross model misspecification could also cause NPD solutions, users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.

Details

Given that a composite score (W) is a weighted sum of item scores:

$$W = \boldsymbol{w}'\boldsymbol{x},$$

where $\boldsymbol{x}$ is a $k \times 1$ vector of the scores of each item, $\boldsymbol{w}$ is a $k \times 1$ weight vector of each item, and k represents the number of items. Then, maximal reliability is obtained by finding $\boldsymbol{w}$ such that reliability attains its maximum (Li, 1997; Raykov, 2012). Note that the reliability can be obtained by

$$\rho = \frac{\boldsymbol{w}'\boldsymbol{S}_T\boldsymbol{w}}{\boldsymbol{w}'\boldsymbol{S}_X\boldsymbol{w}}$$

where $\boldsymbol{S}_T$ is the covariance matrix explained by true scores and $\boldsymbol{S}_X$ is the observed covariance matrix. Numerical method is used to find $\boldsymbol{w}$ in this function.

For continuous items, $\boldsymbol{S}_T$ can be calculated by

$$\boldsymbol{S}_T = \Lambda\Psi\Lambda',$$

where Λ is the factor loading matrix and Ψ is the covariance matrix among factors. $\boldsymbol{S}_X$ is directly obtained by covariance among items.

For categorical items, Green and Yang's (2009) method is used for calculating $\boldsymbol{S}_T$ and $\boldsymbol{S}_X$. The element i and j of $\boldsymbol{S}_T$ can be calculated by

$$[\boldsymbol{S}_T]_{ij} = \sum_{c_i=1}^{C_i-1} \sum_{c_j-1}^{C_j-1} \Phi_2\left(\tau_{x_{c_i}}, \tau_{x_{c_j}}, [\Lambda\Psi\Lambda']_{ij}\right) - \sum_{c_i=1}^{C_i-1} \Phi_1(\tau_{x_{c_i}}) \sum_{c_j-1}^{C_j-1} \Phi_1(\tau_{x_{c_j}}),$$

where C_i and C_j represents the number of thresholds in Items i and j, $\tau_{x_{c_i}}$ represents the threshold c_i of Item i, $\tau_{x_{c_j}}$ represents the threshold c_i of Item j, $\Phi_1(\tau_{x_{c_i}})$ is the cumulative probability of $\tau_{x_{c_i}}$ given a univariate standard normal cumulative distribution and $\Phi_2\left(\tau_{x_{c_i}}, \tau_{x_{c_j}}, \rho\right)$ is the joint cumulative probability of $\tau_{x_{c_i}}$ and $\tau_{x_{c_j}}$ given a bivariate standard normal cumulative distribution with a correlation of ρ

Each element of $\boldsymbol{S}_X$ can be calculated by

$$[\boldsymbol{S}_T]_{ij} = \sum_{c_i=1}^{C_i-1} \sum_{c_j-1}^{C_j-1} \Phi_2\left(\tau_{V_{c_i}}, \tau_{V_{c_j}}, \rho^*_{ij}\right) - \sum_{c_i=1}^{C_i-1} \Phi_1(\tau_{V_{c_i}}) \sum_{c_j-1}^{C_j-1} \Phi_1(\tau_{V_{c_j}}),$$

where ρ^*_{ij} is a polychoric correlation between Items i and j.

Value

Maximal reliability values of each group. The maximal-reliability weights are also provided. Users may extracted the weighted by the `attr` function (see example below).

Author(s)

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

References

Li, H. (1997). A unifying expression for the maximal reliability of a linear composite. *Psychometrika, 62*(2), 245–249. doi:10.1007/BF02295278

Raykov, T. (2012). Scale construction and development using structural equation modeling. In R. H. Hoyle (Ed.), *Handbook of structural equation modeling* (pp. 472–494). New York, NY: Guilford.

See Also

`reliability()` for reliability of an unweighted composite score

Examples

```
total <- 'f =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 '
fit <- cfa(total, data = HolzingerSwineford1939)
maximalRelia(fit)

# Extract the weight
mr <- maximalRelia(fit)
attr(mr, "weight")
```

`measEq.syntax` *Syntax for measurement equivalence*

Description

Automatically generates `lavaan` model syntax to specify a confirmatory factor analysis (CFA) model with equality constraints imposed on user-specified measurement (or structural) parameters. Optionally returns the fitted model (if data are provided) representing some chosen level of measurement equivalence/invariance across groups and/or repeated measures.

Usage

```
measEq.syntax(configural.model, ..., ID.fac = "std.lv",
  ID.cat = "Wu.Estabrook.2016", ID.thr = c(1L, 2L), group = NULL,
  group.equal = "", group.partial = "", longFacNames = list(),
  longIndNames = list(), long.equal = "", long.partial = "",
  auto = "all", warn = TRUE, debug = FALSE, return.fit = FALSE)
```

Arguments

`configural.model`

A model with no measurement-invariance constraints (i.e., representing only configural invariance), unless required for model identification. `configural.model` can be either:

- `lavaan::model.syntax()` or a `lavaan::parTable()` specifying the configural model. Using this option, the user can also provide either raw `data` or summary statistics via `sample.cov` and (optionally) `sample.mean`. See argument descriptions in `lavaan::lavaan()`. In order to include thresholds in the generated syntax, either users must provide raw `data`, or the `configural.model` syntax must specify all thresholds (see first example). If raw `data` are not provided, the number of blocks (groups, levels, or combination) must be indicated using an arbitrary `sample.nobs` argument (e.g., 3 groups could be specified using `sample.nobs=rep(1, 3)`).
- a fitted lavaan::lavaan model (e.g., as returned by `lavaan::cfa()`) estimating the configural model

	Note that the specified or fitted model must not contain any latent structural parameters (i.e., it must be a CFA model), unless they are higher-order constructs with latent indicators (i.e., a second-order CFA).
`...`	Additional arguments (e.g., `data`, `ordered`, or `parameterization`) passed to the `lavaan::lavaan()` function. See also `lavaan::lavOptions()`.
`ID.fac`	`character`. The method for identifying common-factor variances and (if `meanstructure = TRUE`) means. Three methods are available, which go by different names in the literature: • Standardize the common factor (mean = 0, *SD* = 1) by specifying any of: `"std.lv"`, `"unit.variance"`, `"UV"`, `"fixed.factor"`, `"fixed-factor"` • Choose a reference indicator by specifying any of: `"auto.fix.first"`, `"unit.loading"`, `"UL"`, `"marker"`, `"ref"`, `"ref.indicator"`, `"reference.indicator"`, `"reference-indicator"`, `"marker.variable"`, `"marker-variable"` • Apply effects-code constraints to loadings and intercepts by specifying any of: `"FX"`, `"EC"`, `"effects"`, `"effects.coding"`, `"effects-coding"`, `"effects.code"`, `"effects-code"` See Kloessner & Klopp (2019) for details about all three methods.
`ID.cat`	`character`. The method for identifying (residual) variances and intercepts of latent item-responses underlying any `ordered` indicators. Four methods are available: • To follow Wu & Estabrook's (2016) guidelines (default), specify any of: `"Wu.Estabrook.2016"`, `"Wu.2016"`, `"Wu.Estabrook"`, `"Wu"`, `"Wu2016"`. For consistency, specify `ID.fac = "std.lv"`. • To use the default settings of M*plus* and `lavaan`, specify any of: `"default"`, `"Mplus"`, `"Muthen"`. Details provided in Millsap & Tein (2004). • To use the constraints recommended by Millsap & Tein (2004; see also Liu et al., 2017, for the longitudinal case) specify any of: `"millsap"`, `"millsap.2004"`, `"millsap.tein.2004"`. For consistency, specify `ID.fac = "marker"` and `parameterization = "theta"`. • To use the default settings of LISREL, specify `"LISREL"` or `"Joreskog"`. Details provided in Millsap & Tein (2004). For consistency, specify `parameterization = "theta"`. See **Details** and **References** for more information.
`ID.thr`	`integer`. Only relevant when `ID.cat = "Millsap.Tein.2004"`. Used to indicate which thresholds should be constrained for identification. The first integer indicates the threshold used for all indicators, the second integer indicates the additional threshold constrained for a reference indicator (ignored if binary).
`group`	optional `character` indicating the name of a grouping variable. See `lavaan::cfa()`.
`group.equal`	optional `character` vector indicating type(s) of parameter to equate across groups. Ignored if `is.null(group)`. See `lavaan::lavOptions()`.
`group.partial`	optional `character` vector or a parameter table indicating exceptions to `group.equal` (see `lavaan::lavOptions()`). Any variables not appearing in the `configural.model` will be ignored, and any parameter constraints needed for identification (e.g., two thresholds per indicator when `ID.cat = "Millsap"`) will be removed.

`longFacNames`	optional named `list` of `character` vectors, each indicating multiple factors in the model that are actually the same construct measured repeatedly. See **Details** and **Examples**.
`longIndNames`	optional named `list` of `character` vectors, each indicating multiple indicators in the model that are actually the same indicator measured repeatedly. See **Details** and **Examples**.
`long.equal`	optional `character` vector indicating type(s) of parameter to equate across repeated measures. Ignored if no factors are indicated as repeatedly measured in `longFacNames`.
`long.partial`	optional `character` vector or a parameter table indicating exceptions to `long.equal`. Any longitudinal variable names not appearing in `names(longFacNames)` or `names(longIndNames)` will be ignored, and any parameter constraints needed for identification will be removed.
`auto`	Used to automatically included autocorrelated measurement errors among repeatedly measured indicators in `longIndNames`. Specify a single `integer` to set the maximum order (e.g., `auto = 1L` indicates that an indicator's unique factors should only be correlated between adjacently measured occasions). `auto = TRUE` or `"all"` will specify residual covariances among all possible lags per repeatedly measured indicator in `longIndNames`.
`warn, debug`	`logical`. Passed to `lavaan::lavaan()` and `lavaan::lavParseModelString()`. See `lavaan::lavOptions()`.
`return.fit`	`logical` indicating whether the generated syntax should be fitted to the provided `data` (or summary statistics, if provided via `sample.cov`). If `configural.model` is a fitted lavaan model, the generated syntax will be fitted using the `update` method (see lavaan::lavaan), and ... will be passed to `lavaan::lavaan()`. If neither data nor a fitted lavaan model were provided, this must be `FALSE`. If `TRUE`, the generated `measEq.syntax` object will be included in the `lavaan` object's `@external` slot, accessible by `fit@external$measEq.syntax`.

Details

This function is a pedagogical and analytical tool to generate model syntax representing some level of measurement equivalence/invariance across any combination of multiple groups and/or repeated measures. Support is provided for confirmatory factor analysis (CFA) models with simple or complex structure (i.e., cross-loadings and correlated residuals are allowed). For any complexities that exceed the limits of automation, this function is intended to still be useful by providing a means to generate syntax that users can easily edit to accommodate their unique situations.

Limited support is provided for bifactor models and higher-order constructs. Because bifactor models have cross-loadings by definition, the option `ID.fac = "effects.code"` is unavailable. `ID.fac = "UV"` is recommended for bifactor models, but `ID.fac = "UL"` is available on the condition that each factor has a unique first indicator in the `configural.model`. In order to maintain generality, higher-order factors may include a mix of manifest and latent indicators, but they must therefore require `ID.fac = "UL"` to avoid complications with differentiating lower-order vs. higher-order (or mixed-level) factors. The keyword `"loadings"` in `group.equal` or `long.equal` constrains factor loadings of all manifest indicators (including loadings on higher-order factors that also have latent indicators), whereas the keyword `"regressions"` constrains factor loadings of latent indicators. Users can edit the model syntax manually to adjust constraints as necessary, or clever use of the `group.partial` or `long.partial` arguments could make it possible for users to still automated their model syntax. The keyword `"intercepts"` constrains the intercepts of all manifest indicators, and the keyword `"means"` constrains intercepts and means of all latent common factors, regardless of whether they are latent indicators of higher-order factors. To test equivalence of lower-order and higher-order intercepts/means in separate steps, the user can either manually edit their generated syntax or conscientiously exploit the `group.partial` or `long.partial` arguments as necessary.

`ID.fac`**:** If the `configural.model` fixes any (e.g., the first) factor loadings, the generated syntax object will retain those fixed values. This allows the user to retain additional constraints that might be necessary (e.g., if there are only 1 or 2 indicators). Some methods must be used in conjunction with other settings:

- `ID.cat = "Millsap"` requires `ID.fac = "UL"` and `parameterization = "theta"`.
- `ID.cat = "LISREL"` requires `parameterization = "theta"`.
- `ID.fac = "effects.code"` is unavailable when there are any cross-loadings.

`ID.cat`**:** Wu & Estabrook (2016) recommended constraining thresholds to equality first, and doing so should allow releasing any identification constraints no longer needed. For each `ordered` indicator, constraining one threshold to equality will allow the item's intercepts to be estimated in all but the first group or repeated measure. Constraining a second threshold (if applicable) will allow the item's (residual) variance to be estimated in all but the first group or repeated measure. For binary data, there is no independent test of threshold, intercept, or residual-variance equality. Equivalence of thresholds must also be assumed for three-category indicators. These guidelines provide the least restrictive assumptions and tests, and are therefore the default.

The default setting in M*plus* is similar to Wu & Estabrook (2016), except that intercepts are always constrained to zero (so they are assumed to be invariant without testing them). Millsap & Tein (2004) recommended `parameterization = "theta"` and identified an item's residual variance in all but the first group (or occasion; Liu et al., 2017) by constraining its intercept to zero and one of its thresholds to equality. A second threshold for the reference indicator (so `ID.fac = "UL"`) is used to identify the common-factor means in all but the first group/occasion. The LISREL software fixes the first threshold to zero and (if applicable) the second threshold to 1, and assumes any remaining thresholds to be equal across groups / repeated measures; thus, the intercepts are always identified, and residual variances (`parameterization = "theta"`) are identified except for binary data, when they are all fixed to one.

Repeated Measures: If each repeatedly measured factor is measured by the same indicators (specified in the same order in the `configural.model`) on each occasion, without any cross-loadings, the user can let `longIndNames` be automatically generated. Generic names for the repeatedly measured indicators are created using the name of the repeatedly measured factors (i.e., `names(longFacNames)`) and the number of indicators. So the repeatedly measured first indicator (`"ind"`) of a longitudinal construct called "factor" would be generated as `"._factor_ind.1"`.

The same types of parameter can be specified for `long.equal` as for `group.equal` (see `lavaan::lavOptions()` for a list), except for `"residual.covariances"` or `"lv.covariances"`. Instead, users can constrain *auto*covariances using keywords `"resid.autocov"` or `"lv.autocov"`. Note that `group.equal = "lv.covariances"` or `group.equal = "residual.covariances"` will constrain any autocovariances across groups, along with any other covariances the user specified in the `configural.model`. Note also that autocovariances cannot be specified as exceptions in `long.partial`, so anything more complex than the `auto` argument automatically provides should instead be manually specified in the `configural.model`.

When users set `orthogonal=TRUE` in the `configural.model` (e.g., in bifactor models of repeatedly measured constructs), autocovariances of each repeatedly measured factor will still be freely estimated in the generated syntax.

Missing Data: If users wish to utilize the `auxiliary()` function to automatically include auxiliary variables in conjunction with `missing = "FIML"`, they should first generate the hypothesized-model syntax, then submit that syntax as the model to `auxiliary()`. If users utilized `lavaan.mi::lavaan.mi()` to fit their `configural.model` to multiply imputed data, that model can also be passed to the `configural.model` argument, and if `return.fit = TRUE`, the generated model will be fitted to the multiple imputations.

Value

By default, an object of class measEq.syntax. If `return.fit = TRUE`, a fitted `lavaan::lavaan()` model, with the `measEq.syntax` object stored in the `@external` slot, accessible by `fit@external$measEq.syntax`.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Kloessner, S., & Klopp, E. (2019). Explaining constraint interaction: How to interpret estimated model parameters under alternative scaling methods. *Structural Equation Modeling, 26*(1), 143–155. doi:10.1080/10705511.2018.1517356

Liu, Y., Millsap, R. E., West, S. G., Tein, J.-Y., Tanaka, R., & Grimm, K. J. (2017). Testing measurement invariance in longitudinal data with ordered-categorical measures. *Psychological Methods, 22*(3), 486–506. doi:10.1037/met0000075

Millsap, R. E., & Tein, J.-Y. (2004). Assessing factorial invariance in ordered-categorical measures. *Multivariate Behavioral Research, 39*(3), 479–515. doi:10.1207/S15327906MBR3903_4

Wu, H., & Estabrook, R. (2016). Identification of confirmatory factor analysis models of different levels of invariance for ordered categorical outcomes. *Psychometrika, 81*(4), 1014–1045. doi:10.1007/s1133601695060

See Also

`compareFit()`

Examples

```
mod.cat <- ' FU1 =~ u1 + u2 + u3 + u4
             FU2 =~ u5 + u6 + u7 + u8 '
## the 2 factors are actually the same factor (FU) measured twice
longFacNames <- list(FU = c("FU1","FU2"))

## CONFIGURAL model: no constraints across groups or repeated measures
syntax.config <- measEq.syntax(configural.model = mod.cat,
                               # NOTE: data provides info about numbers of
                               #       groups and thresholds
                               data = datCat,
```

```
                               ordered = paste0("u", 1:8),
                               parameterization = "theta",
                               ID.fac = "std.lv", ID.cat = "Wu.Estabrook.2016",
                               group = "g", longFacNames = longFacNames)
## print lavaan syntax to the Console
cat(as.character(syntax.config))
## print a summary of model features
summary(syntax.config)

## THRESHOLD invariance:
## only necessary to specify thresholds if you have no data
mod.th <- '
  u1 | t1 + t2 + t3 + t4
  u2 | t1 + t2 + t3 + t4
  u3 | t1 + t2 + t3 + t4
  u4 | t1 + t2 + t3 + t4
  u5 | t1 + t2 + t3 + t4
  u6 | t1 + t2 + t3 + t4
  u7 | t1 + t2 + t3 + t4
  u8 | t1 + t2 + t3 + t4
'
syntax.thresh <- measEq.syntax(configural.model = c(mod.cat, mod.th),
                               # NOTE: data not provided, so syntax must
                               #       include thresholds, and number of
                               #       groups == 2 is indicated by:
                               sample.nobs = c(1, 1),
                               parameterization = "theta",
                               ID.fac = "std.lv", ID.cat = "Wu.Estabrook.2016",
                               group = "g", group.equal = "thresholds",
                               longFacNames = longFacNames,
                               long.equal = "thresholds")
## notice that constraining 4 thresholds allows intercepts and residual
## variances to be freely estimated in all but the first group & occasion
cat(as.character(syntax.thresh))
## print a summary of model features
summary(syntax.thresh)


## Fit a model to the data either in a subsequent step (recommended):
mod.config <- as.character(syntax.config)
fit.config <- cfa(mod.config, data = datCat, group = "g",
                  ordered = paste0("u", 1:8), parameterization = "theta")
## or in a single step (not generally recommended):
fit.thresh <- measEq.syntax(configural.model = mod.cat, data = datCat,
                            ordered = paste0("u", 1:8),
                            parameterization = "theta",
                            ID.fac = "std.lv", ID.cat = "Wu.Estabrook.2016",
                            group = "g", group.equal = "thresholds",
                            longFacNames = longFacNames,
                            long.equal = "thresholds", return.fit = TRUE)
## compare their fit to test threshold invariance
anova(fit.config, fit.thresh)
```

```
## ---------------------------------------------------------
## RECOMMENDED PRACTICE: fit one invariance model at a time
## ---------------------------------------------------------

## - A downside of setting return.fit=TRUE is that if the model has trouble
##   converging, you don't have the opportunity to investigate the syntax,
##   or even to know whether an error resulted from the syntax-generator or
##   from lavaan itself.
## - A downside of automatically fitting an entire set of invariance models
##   (like the old measurementInvariance() function did) is that you might
##   end up testing models that shouldn't even be fitted because less
##   restrictive models already fail (e.g., don't test full scalar
##   invariance if metric invariance fails! Establish partial metric
##   invariance first, then test equivalent of intercepts ONLY among the
##   indicators that have invariate loadings.)

## The recommended sequence is to (1) generate and save each syntax object,
## (2) print it to the screen to verify you are fitting the model you expect
## to (and potentially learn which identification constraints should be
## released when equality constraints are imposed), and (3) fit that model
## to the data, as you would if you had written the syntax yourself.

## Continuing from the examples above, after establishing invariance of
## thresholds, we proceed to test equivalence of loadings and intercepts
##   (metric and scalar invariance, respectively)
## simultaneously across groups and repeated measures.



## metric invariance
syntax.metric <- measEq.syntax(configural.model = mod.cat, data = datCat,
                               ordered = paste0("u", 1:8),
                               parameterization = "theta",
                               ID.fac = "std.lv", ID.cat = "Wu.Estabrook.2016",
                               group = "g", longFacNames = longFacNames,
                               group.equal = c("thresholds","loadings"),
                               long.equal  = c("thresholds","loadings"))
summary(syntax.metric)                    # summarize model features
mod.metric <- as.character(syntax.metric) # save as text
cat(mod.metric)                           # print/view lavaan syntax
## fit model to data
fit.metric <- cfa(mod.metric, data = datCat, group = "g",
                  ordered = paste0("u", 1:8), parameterization = "theta")
## test equivalence of loadings, given equivalence of thresholds
anova(fit.thresh, fit.metric)

## scalar invariance
syntax.scalar <- measEq.syntax(configural.model = mod.cat, data = datCat,
                               ordered = paste0("u", 1:8),
                               parameterization = "theta",
                               ID.fac = "std.lv", ID.cat = "Wu.Estabrook.2016",
                               group = "g", longFacNames = longFacNames,
```

```
                               group.equal = c("thresholds","loadings",
                                               "intercepts"),
                               long.equal  = c("thresholds","loadings",
                                               "intercepts"))
summary(syntax.scalar)                    # summarize model features
mod.scalar <- as.character(syntax.scalar) # save as text
cat(mod.scalar)                           # print/view lavaan syntax
## fit model to data
fit.scalar <- cfa(mod.scalar, data = datCat, group = "g",
                  ordered = paste0("u", 1:8), parameterization = "theta")
## test equivalence of intercepts, given equal thresholds & loadings
anova(fit.metric, fit.scalar)


## For a single table with all results, you can pass the models to
## summarize to the compareFit() function
Comparisons <- compareFit(fit.config, fit.thresh, fit.metric, fit.scalar)
summary(Comparisons)


## --------------------------------------------------------
## NOT RECOMMENDED: fit several invariance models at once
## --------------------------------------------------------
test.seq <- c("thresholds","loadings","intercepts","means","residuals")
meq.list <- list()
for (i in 0:length(test.seq)) {
  if (i == 0L) {
    meq.label <- "configural"
    group.equal <- ""
    long.equal <- ""
  } else {
    meq.label <- test.seq[i]
    group.equal <- test.seq[1:i]
    long.equal <- test.seq[1:i]
  }
  meq.list[[meq.label]] <- measEq.syntax(configural.model = mod.cat,
                                         data = datCat,
                                         ordered = paste0("u", 1:8),
                                         parameterization = "theta",
                                         ID.fac = "std.lv",
                                         ID.cat = "Wu.Estabrook.2016",
                                         group = "g",
                                         group.equal = group.equal,
                                         longFacNames = longFacNames,
                                         long.equal = long.equal,
                                         return.fit = TRUE)
}

evalMeasEq <- compareFit(meq.list)
summary(evalMeasEq)


## -----------------
```

```
## Binary indicators
## -----------------

## borrow example data from Mplus user guide
myData <- read.table("https://www.statmodel.com/usersguide/chap5/ex5.16.dat")
names(myData) <- c("u1","u2","u3","u4","u5","u6","x1","x2","x3","g")
bin.mod <- '
  FU1 =~ u1 + u2 + u3
  FU2 =~ u4 + u5 + u6
'
## Must SIMULTANEOUSLY constrain thresholds, loadings, and intercepts
test.seq <- list(strong = c("thresholds","loadings","intercepts"),
                 means = "means",
                 strict = "residuals")
meq.list <- list()
for (i in 0:length(test.seq)) {
  if (i == 0L) {
    meq.label <- "configural"
    group.equal <- ""
    long.equal <- ""
  } else {
    meq.label <- names(test.seq)[i]
    group.equal <- unlist(test.seq[1:i])
    # long.equal <- unlist(test.seq[1:i])
  }
  meq.list[[meq.label]] <- measEq.syntax(configural.model = bin.mod,
                                         data = myData,
                                         ordered = paste0("u", 1:6),
                                         parameterization = "theta",
                                         ID.fac = "std.lv",
                                         ID.cat = "Wu.Estabrook.2016",
                                         group = "g",
                                         group.equal = group.equal,
                                         #longFacNames = longFacNames,
                                         #long.equal = long.equal,
                                         return.fit = TRUE)
}

evalMeasEq <- compareFit(meq.list)
summary(evalMeasEq)



## ---------------------
## Multilevel Invariance
## ---------------------

## To test invariance across levels in a MLSEM, specify syntax as though
## you are fitting to 2 groups instead of 2 levels.

mlsem <- ' f1 =~ y1 + y2 + y3
           f2 =~ y4 + y5 + y6 '
## metric invariance
```

```
syntax.metric <- measEq.syntax(configural.model = mlsem, meanstructure = TRUE,
                               ID.fac = "std.lv", sample.nobs = c(1, 1),
                               group = "cluster", group.equal = "loadings")
## by definition, Level-1 means must be zero, so fix them
syntax.metric <- update(syntax.metric,
                        change.syntax = paste0("y", 1:6, " ~ c(0, NA)*1"))
## save as a character string
mod.metric <- as.character(syntax.metric, groups.as.blocks = TRUE)
## convert from multigroup to multilevel
mod.metric <- gsub(pattern = "group:", replacement = "level:",
                   x = mod.metric, fixed = TRUE)
## fit model to data
fit.metric <- lavaan(mod.metric, data = Demo.twolevel, cluster = "cluster")
summary(fit.metric)
```

`measEq.syntax-class` *Class for Representing a Measurement-Equivalence Model*

Description

This class of object stores information used to automatically generate lavaan model syntax to represent user-specified levels of measurement equivalence/invariance across groups and/or repeated measures. See `measEq.syntax()` for details.

Usage

```
## S4 method for signature 'measEq.syntax'
as.character(x, package = "lavaan",
  params = NULL, single = TRUE, groups.as.blocks = FALSE)

## S4 method for signature 'measEq.syntax'
show(object)

## S4 method for signature 'measEq.syntax'
summary(object, verbose = TRUE)

## S4 method for signature 'measEq.syntax'
update(object, ..., evaluate = TRUE,
  change.syntax = NULL)
```

Arguments

`x, object`	an object of class `measEq.syntax`
`package`	`character` indicating the package for which the model syntax should be generated. Currently, only `"lavaan"` and `"mplus"` are supported.

`params`	`character` vector indicating which type(s) of parameter to print syntax for. Must match a type that can be passed to `group.equal` or `long.equal`, but `"residual.covariances"` and `"lv.covariances"` will be silently ignored. Instead, requesting `"residuals"` or `"lv.variances"` will return covariances along with variances. By default (`NULL`), all types are printed.
`single`	`logical` indicating whether to concatenate lavaan `lavaan::model.syntax()` into a single `character` string. Setting `FALSE` will return a vector of strings, which may be convenient (or even necessary to prevent an error) in models with long variable names, many variables, or many groups.
`groups.as.blocks`	`logical` indicating whether to write lavaan `lavaan::model.syntax()` using vectors of labels and values for multiple groups (the default: `FALSE`), or whether to write a separate "block" of syntax per group. The block structure could allow users to apply the generated multigroup syntax (after some editing) to test invariance across levels in a multilevel SEM (see final example on `measEq.syntax()` help page).
`verbose`	`logical` indicating whether to print a summary to the screen (default). If `FALSE`, only a pattern matrix is returned.
`...`	Additional arguments to the `call`, or arguments with changed values.
`evaluate`	If `TRUE`, evaluate the new `call`; otherwise, return the new `call`.
`change.syntax`	`lavaan::model.syntax()` specifying labels or fixed/free values of parameters in `object`. These provide some flexibility to customize existing parameters without having to copy/paste the output of `as.character(object)` into an R script. For example, `group.partial` will free a parameter across all groups, but `update` allows users to free the parameter in just one group while maintaining equality constraints among other groups.

Value

`summary`	`signature(object = "measEq.syntax", verbose = TRUE)`: A `character` matrix indicating the pattern of `numeric`, `ordered`, or latent indicators loading on common factors. By default (`verbose = TRUE`), `summary` also prints descriptive details about the model, including the numbers of indicators and factors, and which parameters are constrained to equality.
`show`	`signature(object = "measEq.syntax")`: Prints a message about how to use the `object` for model fitting. Invisibly returns the `object`.
`update`	`signature(object = "measEq.syntax", ..., evaluate = TRUE, change.syntax = NULL)`: Creates a new `object` with updated arguments in `...`, or updated parameter labels or fixed/free specifications in `object`.
`as.character`	`signature(x = "measEq.syntax", package = "lavaan")`: Converts the `measEq.syntax` object to model syntax that can be copy/pasted or written to a syntax file to be edited before analysis, or simply passed to `lavaan::lavaan()` to fit the model to data. Generated M*plus* syntax could also be utilized using the **MplusAuthomation** package.

Slots

- `package` `character` indicating the software package used to represent the model. Currently, only `"lavaan"` is available, which uses the LISREL representation (see `lavaan::lavOptions()`). In the future, `"OpenMx"` may become available, using RAM representation.
- `model.type` `character`. Currently, only "cfa" is available. Future versions may allow for MIMIC / RFA models, where invariance can be tested across levels of exogenous variables explicitly included as predictors of indicators, controlling for their effects on (or correlation with) the common factors.
- `call` The function call as returned by `match.call()`, with some arguments updated if necessary for logical consistency.
- `meanstructure` `logical` indicating whether a mean structure is included in the model.
- `numeric` `character` vector naming `numeric` manifest indicators.
- `ordered` `character` vector naming `ordered` indicators.
- `parameterization` `character`. See `lavaan::lavOptions()`.
- `specify` `list` of parameter matrices, similar in form to the output of `lavInspect(fit, "free")`. These matrices are `logical`, indicating whether each parameter should be specified in the model syntax.
- `values` `list` of parameter matrices, similar in form to the output of `lavInspect(fit, "free")`. These matrices are `numeric`, indicating whether each parameter should be freely estimated (indicated by `NA`) or fixed to a particular value.
- `labels` `list` of parameter matrices, similar in form to the output of `lavInspect(fit, "free")`. These matrices contain `character` labels used to constrain parameters to equality.
- `constraints` `character` vector containing additional equality constraints used to identify the model when `ID.fac = "fx"`.
- `ngroups` `integer` indicating the number of groups.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

Examples

```
## See ?measEq.syntax help page for examples using lavaan
```

`monteCarloCI` *Monte Carlo Confidence Intervals to Test Functions of Parameter Estimates*

Description

Robust confidence intervals for functions of parameter estimates, based on empirical sampling distributions of estimated model parameters.

Usage

```
monteCarloCI(object = NULL, expr, coefs, ACM, nRep = 20000,
  standardized = FALSE, fast = TRUE, level = 0.95, na.rm = TRUE,
  append.samples = FALSE, plot = FALSE,
  ask = getOption("device.ask.default"), ...)
```

Arguments

`object`	A object of class lavaan::lavaan in which functions of parameters have already been defined using the `:=` operator in `lavaan`'s `lavaan::model.syntax()`. When `NULL`, users must specify `expr`, `coefs`, and `ACM`.
`expr`	Optional `character` vector specifying functions of model parameters (e.g., an indirect effect). Ideally, the vector should have names, which is necessary if any user-defined parameters refer to other user-defined parameters defined earlier in the vector (order matters!). All parameters appearing in the vector must be provided in `coefs`, or defined (as functions of `coefs`) earlier in `expr`. If `length(expr) > 1L`, `nRep` samples will be drawn simultaneously from a single multivariate distribution; thus, `ACM` must include all parameters in `coefs`.
`coefs`	`numeric` vector of parameter estimates used in `expr`. Ignored when `object` is used.
`ACM`	Symmetric `matrix` representing the asymptotic sampling covariance matrix (ACOV) of the parameter estimates in `coefs`. Ignored when `object` is used. Information on how to obtain the ACOV in popular SEM software is described in **Details**.
`nRep`	`integer`. The number of samples to draw, to obtain an empirical sampling distribution of model parameters. Many thousand are recommended to minimize Monte Carlo error of the estimated CIs.
`standardized`	`logical` indicating whether to obtain CIs for the fully standardized (`"std.all"`) estimates, using their asymptotic sampling covariance matrix.
`fast`	`logical` indicating whether to use a fast algorithm that assumes all functions of parameters (in `object` or `expr`) use standard operations. Set to `FALSE` if using (e.g.) `c()` to concatenate parameters in the definition, which would have unintended consequences when vectorizing functions in `expr` across sampled parameters.
`level`	`numeric` confidence level, between 0–1
`na.rm`	`logical` passed to `stats::quantile()`
`append.samples`	`logical` indicating whether to return the simulated empirical sampling distribution of parameters (in `coefs`) and functions (in `expr`) in a `list` with the results. This could be useful to calculate more precise highest-density intervals (see examples).
`plot`	`logical` indicating whether to plot the empirical sampling distribution of each function in `expr`
`ask`	whether to prompt user before printing each plot
`...`	arguments passed to `graphics::hist()` when `plot = TRUE`.

Details

This function implements the Monte Carlo method of obtaining an empirical sampling distribution of estimated model parameters, as described by MacKinnon et al. (2004) for testing indirect effects in mediation models. This is essentially a parametric bootstrap method, which (re)samples parameters (rather than raw data) from a multivariate-normal distribution with mean vector equal to estimates in `coef()` and covariance matrix equal to the asymptotic covariance matrix `vcov()` of estimated parameters.

The easiest way to use the function is to fit a SEM to data with `lavaan::lavaan()`, using the `:=` operator in the `lavaan::model.syntax()` to specify user-defined parameters. All information is then available in the resulting lavaan::lavaan object. Alternatively (especially when using external SEM software to fit the model), the expression(s) can be explicitly passed to the function, along with the vector of estimated model parameters and their associated asymptotic sampling covariance matrix (ACOV). For further information on the Monte Carlo method, see MacKinnon et al. (2004) and Preacher & Selig (2012).

The asymptotic covariance matrix can be obtained easily from many popular SEM software packages.

- LISREL: Including the EC option on the OU line will print the ACM to a seperate file. The file contains the lower triangular elements of the ACM in free format and scientific notation.
- M*plus*: Include the command TECH3; in the OUTPUT section. The ACM will be printed in the output.
- `lavaan`: Use the `vcov()` method on the fitted lavaan::lavaan object to return the ACM.

Value

A `lavaan.data.frame` (to use lavaan's `print` method) with point estimates and confidence limits of each requested function of parameters in `expr` is returned. If `append.samples = TRUE`, output will be a `list` with the same `$Results` along with a second `data.frame` with the `$Samples` (in rows) of each parameter (in columns), and an additional column for each requested function of those parameters.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

MacKinnon, D. P., Lockwood, C. M., & Williams, J. (2004). Confidence limits for the indirect effect: Distribution of the product and resampling methods. *Multivariate Behavioral Research, 39*(1) 99–128. doi:10.1207/s15327906mbr3901_4

Preacher, K. J., & Selig, J. P. (2010, July). Monte Carlo method for assessing multilevel mediation: An interactive tool for creating confidence intervals for indirect effects in 1-1-1 multilevel models. Computer software available from `https://quantpsy.org/`.

Preacher, K. J., & Selig, J. P. (2012). Advantages of Monte Carlo confidence intervals for indirect effects. *Communication Methods and Measures, 6*(2), 77–98. doi:10.1080/19312458.2012.679848

Selig, J. P., & Preacher, K. J. (2008, June). Monte Carlo method for assessing mediation: An interactive tool for creating confidence intervals for indirect effects. Computer software available from `https://quantpsy.org/`.

Examples

```
## From the mediation tutorial:
## https://lavaan.ugent.be/tutorial/mediation.html

set.seed(1234)
X <- rnorm(100)
M <- 0.5*X + rnorm(100)
Y <- 0.7*M + rnorm(100)
dat <- data.frame(X = X, Y = Y, M = M)

mod <- ' # direct effect
  Y ~ c*X
  # mediator
  M ~ a*X
  Y ~ b*M
  # indirect effect (a*b)
  ind := a*b
  # total effect
  total := ind + c
'
fit <- sem(mod, data = dat)
summary(fit, ci = TRUE) # print delta-method CIs

## Automatically extract information from lavaan object
set.seed(1234)
monteCarloCI(fit) # CIs more robust than delta method in smaller samples

## delta method for standardized solution
standardizedSolution(fit)
## compare to Monte Carlo CIs:
set.seed(1234)
monteCarloCI(fit, standardized = TRUE)


## save samples to calculate more precise intervals:
set.seed(1234)
foo <- monteCarloCI(fit, append.samples = TRUE)
# library(HDInterval) # not a dependency; must be installed
# hdi(foo$Samples)

## Parameters can also be obtained from an external analysis
myParams <- c("a","b","c")
(coefs <- coef(fit)[myParams]) # names must match those in the "expression"
## Asymptotic covariance matrix from an external analysis
(AsyCovMat <- vcov(fit)[myParams, myParams])
## Compute CI, include a plot
set.seed(1234)
monteCarloCI(expr = c(ind = 'a*b', total = 'ind + c',
                      ## other arbitrary functions are also possible
                      meaningless = 'sqrt(a)^b / log(abs(c))'),
             coefs = coefs, ACM = AsyCovMat,
             plot = TRUE, ask = TRUE) # print a plot for each
```

`moreFitIndices` *Calculate more fit indices*

Description

Calculate more fit indices that are not already provided in lavaan.

Usage

```
moreFitIndices(object, fit.measures = "all", nPrior = 1)
```

Arguments

`object`	The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions.
`fit.measures`	Additional fit measures to be calculated. All additional fit measures are calculated by default
`nPrior`	The sample size on which prior is based. This argument is used to compute `bic.priorN`.

Details

See `nullRMSEA()` for the further details of the computation of RMSEA of the null model.

Gamma-Hat (`gammaHat`; West, Taylor, & Wu, 2012) is a global goodness-of-fit index which can be computed (assuming equal number of indicators across groups) by

$$\hat{\Gamma} = \frac{p}{p + 2 \times \frac{\chi^2_k - df_k}{N}},$$

where p is the number of variables in the model, χ^2_k is the χ^2 test statistic value of the target model, df_k is the degree of freedom when fitting the target model, and N is the sample size (or sample size minus the number of groups if `mimic` is set to `"EQS"`).

Adjusted Gamma-Hat (`adjGammaHat`; West, Taylor, & Wu, 2012) is a global fit index which can be computed by

$$\hat{\Gamma}_{\text{adj}} = \left(1 - \frac{K \times p \times (p+1)}{2 \times df_k}\right) \times \left(1 - \hat{\Gamma}\right),$$

where K is the number of groups (please refer to Dudgeon, 2004, for the multiple-group adjustment for `adjGammaHat`).

Note that if Satorra–Bentler's or Yuan–Bentler's method is used, the fit indices using the scaled χ^2 values are also provided.

The remaining indices are information criteria calculated using the `object`'s $-2\times$ log-likelihood, abbreviated $-2LL$.

Corrected Akaike Information Criterion (`aic.smallN`; Burnham & Anderson, 2003) is a corrected version of AIC for small sample size, often abbreviated AICc:

$$\text{AIC}_{\text{small}-N} = AIC + \frac{2q(q+1)}{N-q-1},$$

where AIC is the original AIC: $-2LL + 2q$ (where q = the number of estimated parameters in the target model). Note that AICc is a small-sample correction derived for univariate regression models, so it is probably *not* appropriate for comparing SEMs.

Corrected Bayesian Information Criterion (`bic.priorN`; Kuha, 2004) is similar to BIC but explicitly specifying the sample size on which the prior is based (N_{prior}) using the `nPrior` argument.

$$\text{BIC}_{\text{prior}-N} = -2LL + q\log{(1 + \frac{N}{N_{prior}})}.$$

Bollen et al. (2012, 2014) discussed additional BICs that incorporate more terms from a Taylor series expansion, which the standard BIC drops. The "Scaled Unit-Information Prior" BIC is calculated depending on whether the product of the vector of estimated model parameters ($\hat{\theta}$) and the observed information matrix (FIM) exceeds the number of estimated model parameters (Case 1) or not (Case 2), which is checked internally:

$$\text{SPBIC}_{\text{Case 1}} = -2LL + q(1 - \frac{q}{\hat{\theta}^{\prime}\text{FIM}\hat{\theta}}), \text{ or}$$

$$\text{SPBIC}_{\text{Case 2}} = -2LL + \hat{\theta}^{\prime}\text{FIM}\hat{\theta},$$

Note that this implementation of SPBIC is calculated on the assumption that priors for all estimated parameters are centered at zero, which is inappropriate for most SEMs (e.g., variances should not have priors centered at the lowest possible value; Bollen, 2014, p. 6).

Bollen et al. (2014, eq. 14) credit the HBIC to Haughton (1988):

$$\text{HBIC} = -2LL + q\log{\frac{N}{2\pi}}.$$

Bollen et al. (2012, p. 305) proposed the information matrix (I)-based BIC by adding another term:

$$\text{IBIC} = -2LL + q\log{\frac{N}{2\pi}} + \log{\det{\text{FIM}}},$$

or equivalently, using the inverse information (the asymptotic sampling covariance matrix of estimated parameters: ACOV):

$$\text{IBIC} = -2LL - q\log{2\pi} - \log{\det{\text{ACOV}}}.$$

Stochastic information criterion (SIC; see Preacher, 2006, for details) is similar to IBIC but does not include the $q\log{2\pi}$ term that is also in HBIC. SIC and IBIC both account for model complexity in a model's functional form, not merely the number of free parameters. The SIC can be computed as:

$$\text{SIC} = -2LL + q\log{N} + \log{\det{\text{FIM}}} = -2LL - \log{\det{\text{ACOV}}}.$$

Hannan–Quinn Information Criterion (HQC; Hannan & Quinn, 1979) is used for model selection, similar to AIC or BIC.

$$\text{HQC} = -2LL + 2q \log\left(\log N\right),$$

Bozdogan Information Complexity (ICOMP) Criteria (Howe et al., 2011), instead of penalizing the number of free parameters directly, ICOMP penalizes the covariance complexity of the model.

$$\text{ICOMP} = -2LL + s \times log(\frac{\bar{\lambda_a}}{\bar{\lambda_g}})$$

Value

A `numeric` `lavaan.vector` including any of the following requested via `fit.measures=`

1. `gammaHat`: Gamma-Hat
2. `adjGammaHat`: Adjusted Gamma-Hat
3. `baseline.rmsea`: RMSEA of the default baseline (i.e., independence) model
4. `gammaHat.scaled`: Gamma-Hat using scaled χ^2
5. `adjGammaHat.scaled`: Adjusted Gamma-Hat using scaled χ^2
6. `baseline.rmsea.scaled`: RMSEA of the default baseline (i.e., independence) model using scaled χ^2
7. `aic.smallN`: Corrected (for small sample size) AIC
8. `bic.priorN`: BIC with specified prior sample size
9. `spbic`: Scaled Unit-Information Prior BIC (SPBIC)
10. `hbic`: Haughton's BIC (HBIC)
11. `ibic`: Information-matrix-based BIC (IBIC)
12. `sic`: Stochastic Information Criterion (SIC)
13. `hqc`: Hannan-Quinn Information Criterion (HQC)
14. `icomp`: Bozdogan Information Complexity (ICOMP) Criteria

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

Aaron Boulton (University of Delaware)

Ruben Arslan (Humboldt-University of Berlin, <`rubenarslan@gmail.com`>)

Yves Rosseel (Ghent University; <`Yves.Rosseel@UGent.be`>)

Mauricio Garnier-Villarreal (Vrije Universiteit Amsterdam; <`mgv@pm.me`>)

A great deal of feedback was provided by Kris Preacher regarding Bollen et al.'s (2012, 2014) extensions of BIC.

References

Bollen, K. A., Ray, S., Zavisca, J., & Harden, J. J. (2012). A comparison of Bayes factor approximation methods including two new methods. *Sociological Methods & Research, 41*(2), 294–324. doi:10.1177/0049124112452393

Bollen, K. A., Harden, J. J., Ray, S., & Zavisca, J. (2014). BIC and alternative Bayesian information criteria in the selection of structural equation models. *Structural Equation Modeling, 21*(1), 1–19. doi:10.1080/10705511.2014.856691

Burnham, K., & Anderson, D. (2003). *Model selection and multimodel inference: A practical–theoretic approach*. New York, NY: Springer–Verlag.

Dudgeon, P. (2004). A note on extending Steiger's (1998) multiple sample RMSEA adjustment to other noncentrality parameter-based statistic. *Structural Equation Modeling, 11*(3), 305–319. doi:10.1207/s15328007sem1103_1

Howe, E. D., Bozdogan, H., & Katragadda, S. (2011). Structural equation modeling (SEM) of categorical and mixed-data using the novel Gifi transformations and information complexity (ICOMP) criterion. *Istanbul University Journal of the School of Business Administration, 40*(1), 86–123.

Kuha, J. (2004). AIC and BIC: Comparisons of assumptions and performance. *Sociological Methods Research, 33*(2), 188–229. doi:10.1177/0049124103262065

Preacher, K. J. (2006). Quantifying parsimony in structural equation modeling. *Multivariate Behavioral Research, 43*(3), 227–259. doi:10.1207/s15327906mbr4103_1

West, S. G., Taylor, A. B., & Wu, W. (2012). Model fit and model selection in structural equation modeling. In R. H. Hoyle (Ed.), *Handbook of structural equation modeling* (pp. 209–231). New York, NY: Guilford.

See Also

- `epcEquivFit()` For the equivalence testing based on expected parameter changes for model fit evaluation
- `nullRMSEA()` For RMSEA of the default independence model

Examples

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

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

fit2 <- cfa(HS.model, data = HolzingerSwineford1939, estimator = "mlr")
moreFitIndices(fit2)
```

`mvrnonnorm` *Generate Non-normal Data using Vale and Maurelli (1983) method*

Description

Generate Non-normal Data using Vale and Maurelli (1983) method. The function is designed to be as similar as the popular `mvrnorm` function in the `MASS` package. The codes are copied from `mvrnorm` function in the `MASS` package for argument checking and `lavaan` package for data generation using Vale and Maurelli (1983) method.

Usage

```
mvrnonnorm(n, mu, Sigma, skewness = NULL, kurtosis = NULL,
  empirical = FALSE)
```

Arguments

`n`	Sample size
`mu`	A mean vector. If elements are named, those will be used as variable names in the returned data matrix.
`Sigma`	A positive-definite symmetric matrix specifying the covariance matrix of the variables. If rows or columns are named (and `mu` is unnamed), those will be used as variable names in the returned data matrix.
`skewness`	A vector of skewness of the variables
`kurtosis`	A vector of excessive kurtosis of the variables
`empirical`	deprecated, ignored.

Value

A data matrix

Author(s)

The original function is the `lavaan::simulateData()` function written by Yves Rosseel in the `lavaan` package. The function is adjusted for a convenient usage by Sunthud Pornprasertmanit (`<psunthud@gmail.com>`). Terrence D. Jorgensen added the feature to retain variable names from `mu` or `Sigma`.

References

Vale, C. D. & Maurelli, V. A. (1983). Simulating multivariate nonormal distributions. *Psychometrika, 48*(3), 465–471. doi:10.1007/BF02293687

Examples

```
set.seed(123)
mvrnonnorm(20, c(1, 2), matrix(c(10, 2, 2, 5), 2, 2),
skewness = c(5, 2), kurtosis = c(3, 3))
## again, with variable names specified in mu
set.seed(123)
mvrnonnorm(20, c(a = 1, b = 2), matrix(c(10, 2, 2, 5), 2, 2),
skewness = c(5, 2), kurtosis = c(3, 3))
```

`net` *Nesting and Equivalence Testing*

Description

This test examines whether pairs of SEMs are nested or equivalent.

Usage

```
net(..., crit = 1e-04)
```

Arguments

`...`	The `lavaan` objects used for test of nesting and equivalence
`crit`	The upper-bound criterion for testing the equivalence of models. Models are considered nested (or equivalent) if the difference between their χ^2 fit statistics is less than this criterion.

Details

The concept of nesting/equivalence should be the same regardless of estimation method. However, the particular method of testing nesting/equivalence (as described in Bentler & Satorra, 2010) employed by the `net` function analyzes summary statistics (model-implied means and covariance matrices, not raw data). In the case of robust methods like MLR, the raw data is only utilized for the robust adjustment to SE and chi-sq, and the net function only checks the unadjusted chi-sq for the purposes of testing nesting/equivalence. This method also applies to models for categorical data, following the procedure described by Asparouhov & Muthen (2019).

Value

The Net object representing the outputs for nesting and equivalent testing, including a logical matrix of test results and a vector of degrees of freedom for each model.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Bentler, P. M., & Satorra, A. (2010). Testing model nesting and equivalence. *Psychological Methods, 15*(2), 111–123. doi:10.1037/a0019625

Asparouhov, T., & Muthen, B. (2019). Nesting and equivalence testing for structural equation models. *Structural Equation Modeling, 26*(2), 302–309. doi:10.1080/10705511.2018.1513795

Examples

```
m1 <- ' visual  =~ x1 + x2 + x3
       textual =~ x4 + x5 + x6
       speed   =~ x7 + x8 + x9 '


m2 <- ' f1  =~ x1 + x2 + x3 + x4
       f2 =~ x5 + x6 + x7 + x8 + x9 '

m3 <- ' visual  =~ x1 + x2 + x3
       textual =~ eq*x4 + eq*x5 + eq*x6
       speed   =~ x7 + x8 + x9 '

fit1 <- cfa(m1, data = HolzingerSwineford1939)
fit1a <- cfa(m1, data = HolzingerSwineford1939, std.lv = TRUE) # Equivalent to fit1
fit2 <- cfa(m2, data = HolzingerSwineford1939) # Not equivalent to or nested in fit1
fit3 <- cfa(m3, data = HolzingerSwineford1939) # Nested in fit1 and fit1a


tests <- net(fit1, fit1a, fit2, fit3)
tests
summary(tests)
```

`Net-class` *Class For the Result of Nesting and Equivalence Testing*

Description

This class contains the results of nesting and equivalence testing among multiple models

Usage

```
## S4 method for signature 'Net'
show(object)

## S4 method for signature 'Net'
summary(object)
```

Arguments

`object`	An object of class `Net`.

Value

`show`	`signature(object = "Net")`: prints the logical matrix of test results. `NA` indicates a model did not converge.
`summary`	`signature(object = "Net")`: prints a narrative description of results. The original `object` is invisibly returned.

Slots

`test` Logical `matrix` indicating nesting/equivalence among models

`df` The degrees of freedom of tested models

Objects from the Class

Objects can be created via the `net()` function.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

See Also

`net()`

Examples

```
# See the example in the net function.
```

`nullRMSEA` *Calculate the RMSEA of the null model*

Description

Calculate the RMSEA of the null (baseline) model

Usage

```
nullRMSEA(object, scaled = FALSE, silent = FALSE)
```

Arguments

`object`	The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions.
`scaled`	If `TRUE`, the scaled (or robust, if available) RMSEA is returned. Ignored if a robust test statistic was not requested.
`silent`	If `TRUE`, do not print anything on the screen.

Details

RMSEA of the null model is calculated similar to the formula provided in the `lavaan` package. The standard formula of RMSEA is

$$RMSEA = \sqrt{\frac{\chi^2}{N \times df} - \frac{1}{N}} \times \sqrt{G}$$

where χ^2 is the chi-square test statistic value of the target model, N is the total sample size, df is the degree of freedom of the hypothesized model, G is the number of groups. Kenny proposed in his website that

"A reasonable rule of thumb is to examine the RMSEA for the null model and make sure that is no smaller than 0.158. An RMSEA for the model of 0.05 and a TLI of .90, implies that the RMSEA of the null model is 0.158. If the RMSEA for the null model is less than 0.158, an incremental measure of fit may not be that informative."

See also the paper cited in **References**.

Value

A value of RMSEA of the null model (a `numeric` vector) returned invisibly.

Author(s)

Ruben Arslan (Humboldt-University of Berlin, `<rubenarslan@gmail.com>`)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Kenny, D. A., Kaniskan, B., & McCoach, D. B. (2015). The performance of RMSEA in models with small degrees of freedom. *Sociological Methods Research, 44*(3), 486–507. doi:10.1177/0049124114543236

See Also

- `miPowerFit()` For the modification indices and their power approach for model fit evaluation
- `moreFitIndices()` For other fit indices

Examples

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

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

`parcelAllocation` *Random Allocation of Items to Parcels in a Structural Equation Model*

Description

This function generates a given number of randomly generated item-to-parcel allocations, fits a model to each allocation, and provides averaged results over all allocations.

Usage

```
parcelAllocation(model, data, parcel.names, item.syntax, nAlloc = 100,
  fun = "sem", alpha = 0.05, fit.measures = c("chisq", "df", "cfi",
  "tli", "rmsea", "srmr"), ..., show.progress = FALSE, iseed = 12345,
  do.fit = TRUE, return.fit = FALSE, warn = FALSE)
```

Arguments

`model`	`lavaan::lavaan()` model syntax specifying the model fit to (at least some) parceled data. Note that there can be a mixture of items and parcels (even within the same factor), in case certain items should never be parceled. Can be a character string or parameter table. Also see `lavaan::lavaanify()` for more details.
`data`	A `data.frame` containing all observed variables appearing in the `model`, as well as those in the `item.syntax` used to create parcels. If the data have missing values, multiple imputation before parceling is recommended: submit a stacked data set (with a variable for the imputation number, so they can be separateed later) and set `do.fit = FALSE` to return the list of `data.frames` (one per allocation), each of which is a stacked, imputed data set with parcels.
`parcel.names`	`character` vector containing names of all parcels appearing as indicators in `model`.
`item.syntax`	`lavaan::model.syntax()` specifying the model that would be fit to all of the unparceled items, including items that should be randomly allocated to parcels appearing in `model`.
`nAlloc`	The number of random items-to-parcels allocations to generate.
`fun`	`character` string indicating the name of the `lavaan::lavaan()` function used to fit `model` to `data`. Can only take the values `"lavaan"`, `"sem"`, `"cfa"`, or `"growth"`.

`alpha`	Alpha level used as criterion for significance.
`fit.measures`	`character` vector containing names of fit measures to request from each fitted `lavaan::lavaan()` model. See the output of `lavaan::fitMeasures()` for a list of available measures.
`...`	Additional arguments to be passed to `lavaan::lavaanList()`. See also `lavaan::lavOptions()`
`show.progress`	If `TRUE`, show a `utils::txtProgressBar()` indicating how fast the model-fitting iterates over allocations.
`iseed`	(Optional) Random seed used for parceling items. When the same random seed is specified and the program is re-run, the same allocations will be generated. Using the same `iseed` argument will ensure the any model is fit to the same parcel allocations. *Note*: When using **parallel** options, you must first type `RNGkind("L'Ecuyer-CMRG")` into the R Console, so that the seed will be controlled across cores.
`do.fit`	If `TRUE` (default), the `model` is fitted to each parceled data set, and the summary of results is returned (see the Value section below). If `FALSE`, the items are randomly parceled, but the model is not fit; instead, the `list` of `data.frames` is returned (so assign it to an object).
`return.fit`	If `TRUE`, a lavaan::lavaanList object is returned with the `list` of results across allocations
`warn`	Whether to print warnings when fitting `model` to each allocation

Details

This function implements the random item-to-parcel allocation procedure described in Sterba (2011) and Sterba and MacCallum (2010). The function takes a single data set with item-level data, randomly assigns items to parcels, fits a structural equation model to the parceled data using `lavaan::lavaanList()`, and repeats this process for a user-specified number of random allocations. Results from all fitted models are summarized in the output. For further details on the benefits of randomly allocating items to parcels, see Sterba (2011) and Sterba and MacCallum (2010).

Value

`Estimates`	A `data.frame` containing results related to parameter estimates with columns corresponding to their names; average and standard deviation across allocations; minimum, maximum, and range across allocations; and the proportion of allocations in which each parameter estimate was significant.
`SE`	A `data.frame` containing results similar to `Estimates`, but related to the standard errors of parameter estimates.
`Fit`	A `data.frame` containing results related to model fit, with columns corresponding to fit index names; their average and standard deviation across allocations; the minimum, maximum, and range across allocations; and (if the test statistic or RMSEA is included in `fit.measures`) the proportion of allocations in which each test of (exact or close) fit was significant.
`Model`	A lavaan::lavaanList object containing results of the `model` fitted to each parcel allocation. Only returned if `return.fit = TRUE`.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Sterba, S. K. (2011). Implications of parcel-allocation variability for comparing fit of item-solutions and parcel-solutions. *Structural Equation Modeling, 18*(4), 554–577. doi:10.1080/10705511.2011.607073

Sterba, S. K. & MacCallum, R. C. (2010). Variability in parameter estimates and model fit across random allocations of items to parcels. *Multivariate Behavioral Research, 45*(2), 322–358. doi:10.1080/00273171003680302

Sterba, S. K., & Rights, J. D. (2016). Accounting for parcel-allocation variability in practice: Combining sources of uncertainty and choosing the number of allocations. *Multivariate Behavioral Research, 51*(2–3), 296–313. doi:10.1080/00273171.2016.1144502

Sterba, S. K., & Rights, J. D. (2017). Effects of parceling on model selection: Parcel-allocation variability in model ranking. *Psychological Methods, 22*(1), 47–68. doi:10.1037/met0000067

See Also

`PAVranking()` for comparing 2 models, `poolMAlloc()` for choosing the number of allocations

Examples

```
## Fit 2-factor CFA to simulated data. Each factor has 9 indicators.

## Specify the item-level model (if NO parcels were created)
item.syntax <- c(paste0("f1 =~ f1item", 1:9),
                 paste0("f2 =~ f2item", 1:9))
cat(item.syntax, sep = "\n")
## Below, we reduce the size of this same model by
## applying different parceling schemes


## 3-indicator parcels
mod.parcels <- '
f1 =~ par1 + par2 + par3
f2 =~ par4 + par5 + par6
'
## names of parcels
(parcel.names <- paste0("par", 1:6))


## override default random-number generator to use parallel options
RNGkind("L'Ecuyer-CMRG")

parcelAllocation(mod.parcels, data = simParcel, nAlloc = 100,
                 parcel.names = parcel.names, item.syntax = item.syntax,
               # parallel = "multicore",  # parallel available in Mac/Linux
                 std.lv = TRUE)           # any addition lavaan arguments
```

```
## POOL RESULTS by treating parcel allocations as multiple imputations
## Details provided in Sterba & Rights (2016); see ?poolMAlloc.

## save list of data sets instead of fitting model yet
dataList <- parcelAllocation(mod.parcels, data = simParcel, nAlloc = 100,
                             parcel.names = parcel.names,
                             item.syntax = item.syntax,
                             do.fit = FALSE)
## now fit the model to each data set
library(lavaan.mi)
fit.parcels <- cfa.mi(mod.parcels, data = dataList, std.lv = TRUE)
summary(fit.parcels)        # pooled using Rubin's rules
anova(fit.parcels)          # pooled test statistic
help(package = "lavaan.mi") # find more methods for pooling results



## multigroup example
simParcel$group <- 0:1 # arbitrary groups for example
mod.mg <- '
f1 =~ par1 + c(L2, L2)*par2 + par3
f2 =~ par4 + par5 + par6
'
## names of parcels
(parcel.names <- paste0("par", 1:6))

parcelAllocation(mod.mg, data = simParcel, parcel.names, item.syntax,
                 std.lv = TRUE, group = "group", group.equal = "loadings",
                 nAlloc = 20, show.progress = TRUE)



## parcels for first factor, items for second factor
mod.items <- '
f1 =~ par1 + par2 + par3
f2 =~ f2item2 + f2item7 + f2item8
'
## names of parcels
(parcel.names <- paste0("par", 1:3))

parcelAllocation(mod.items, data = simParcel, parcel.names, item.syntax,
                 nAlloc = 20, std.lv = TRUE)



## mixture of 1- and 3-indicator parcels for second factor
mod.mix <- '
f1 =~ par1 + par2 + par3
f2 =~ f2item2 + f2item7 + f2item8 + par4 + par5 + par6
'
## names of parcels
(parcel.names <- paste0("par", 1:6))
```

```
parcelAllocation(mod.mix, data = simParcel, parcel.names, item.syntax,
                 nAlloc = 20, std.lv = TRUE)
```

`partialInvariance` *Partial Measurement Invariance Testing Across Groups*

Description

This test will provide partial invariance testing by (a) freeing a parameter one-by-one from nested model and compare with the original nested model or (b) fixing (or constraining) a parameter one-by-one from the parent model and compare with the original parent model. This function only works with congeneric models. The `partialInvariance` is used for continuous variable. The `partialInvarianceCat` is used for categorical variables.

Usage

```
partialInvariance(fit, type, free = NULL, fix = NULL, refgroup = 1,
  poolvar = TRUE, p.adjust = "none", fbound = 2, return.fit = FALSE,
  method = "satorra.bentler.2001")

partialInvarianceCat(fit, type, free = NULL, fix = NULL, refgroup = 1,
  poolvar = TRUE, p.adjust = "none", return.fit = FALSE,
  method = "satorra.bentler.2001")
```

Arguments

`fit`	A list of models for invariance testing. Each model should be assigned by appropriate names (see details).
`type`	The types of invariance testing: "metric", "scalar", "strict", or "means"
`free`	A vector of variable names that are free across groups in advance. If partial mean invariance is tested, this argument represents a vector of factor names that are free across groups.
`fix`	A vector of variable names that are constrained to be equal across groups in advance. If partial mean invariance is tested, this argument represents a vector of factor names that are fixed across groups.
`refgroup`	The reference group used to make the effect size comparison with the other groups.
`poolvar`	If `TRUE`, the variances are pooled across group for standardization. Otherwise, the variances of the reference group are used for standardization.
`p.adjust`	The method used to adjust p values. See `stats::p.adjust()` for the options for adjusting p values. The default is to not use any corrections.
`fbound`	The z-scores of factor that is used to calculate the effect size of the loading difference proposed by Millsap and Olivera-Aguilar (2012).

`return.fit`	Return the submodels fitted by this function
`method`	The method used to calculate likelihood ratio test. See `lavaan::lavTestLRT()` for available options

Details

There are four types of partial invariance testing:

- Partial weak invariance. The model named `fit.configural` from the list of models is compared with the model named `fit.loadings`. Each loading will be freed or fixed from the metric and configural invariance models respectively. The modified models are compared with the original model. Note that the objects in the list of models must have the names of `"fit.configural"` and `"fit.loadings"`. Users may use "metric", "weak", "loading", or "loadings" in the `type` argument. Note that, for testing invariance on marker variables, other variables will be assigned as marker variables automatically.
- Partial strong invariance. The model named `fit.loadings` from the list of models is compared with the model named either `fit.intercepts` or 'fit.thresholds'. Each intercept will be freed or fixed from the scalar and metric invariance models respectively. The modified models are compared with the original model. Note that the objects in the list of models must have the names of "fit.loadings" and either "fit.intercepts" or "fit.thresholds". Users may use "scalar", "strong", "intercept", "intercepts", "threshold", or "thresholds" in the `type` argument. Note that, for testing invariance on marker variables, other variables will be assigned as marker variables automatically. Note that if all variables are dichotomous, scalar invariance testing is not available.
- Partial strict invariance. The model named either 'fit.intercepts' or 'fit.thresholds' (or 'fit.loadings') from the list of models is compared with the model named 'fit.residuals'. Each residual variance will be freed or fixed from the strict and scalar (or metric) invariance models respectively. The modified models are compared with the original model. Note that the objects in the list of models must have the names of "fit.residuals" and either "fit.intercepts", "fit.thresholds", or "fit.loadings". Users may use "strict", "residual", "residuals", "error", or "errors" in the `type` argument.
- Partial mean invariance. The model named either 'fit.intercepts' or 'fit.thresholds' (or 'fit.residuals' or 'fit.loadings') from the list of models is compared with the model named 'fit.means'. Each factor mean will be freed or fixed from the means and scalar (or strict or metric) invariance models respectively. The modified models are compared with the original model. Note that the objects in the list of models must have the names of "fit.means" and either "fit.residuals", "fit.intercepts", "fit.thresholds", or "fit.loadings". Users may use "means" or "mean" in the `type` argument.

Two types of comparisons are used in this function:

1. `free`: The nested model is used as a template. Then, one parameter indicating the differences between two models is free. The new model is compared with the nested model. This process is repeated for all differences between two models. The likelihood-ratio test and the difference in CFI are provided.
2. `fix`: The parent model is used as a template. Then, one parameter indicating the differences between two models is fixed or constrained to be equal to other parameters. The new model is then compared with the parent model. This process is repeated for all differences between two models. The likelihood-ratio test and the difference in CFI are provided.

3. `wald`: This method is similar to the `fix` method. However, instead of building a new model and compare them with likelihood-ratio test, multivariate wald test is used to compare equality between parameter estimates. See `lavaan::lavTestWald()` for further details. Note that if any rows of the contrast cannot be summed to 0, the Wald test is not provided, such as comparing two means where one of the means is fixed as 0. This test statistic is not as accurate as likelihood-ratio test provided in `fix`. I provide it here in case that likelihood-ratio test fails to converge.

Note that this function does not adjust for the inflated Type I error rate from multiple tests. The degree of freedom of all tests would be the number of groups minus 1.

The details of standardized estimates and the effect size used for each parameters are provided in the vignettes by running `vignette("partialInvariance")`.

Value

A list of results are provided. The list will consists of at least two elements:

1. `estimates`: The results of parameter estimates including pooled estimates (`poolest`), the estimates for each group, standardized estimates for each group (`std`), the difference in standardized values, and the effect size statistic (*q* for factor loading difference and *h* for error variance difference). See the details of this effect size statistic by running `vignette("partialInvariance")`. In the `partialInvariance` function, the additional effect statistics proposed by Millsap and Olivera-Aguilar (2012) are provided. For factor loading, the additional outputs are the observed mean difference (`diff_mean`), the mean difference if factor scores are low (`low_fscore`), and the mean difference if factor scores are high (`high_fscore`). The low factor score is calculated by (a) finding the factor scores that its *z* score equals -`bound` (the default is -2) from all groups and (b) picking the minimum value among the factor scores. The high factor score is calculated by (a) finding the factor scores that its *z* score equals `bound` (default = 2) from all groups and (b) picking the maximum value among the factor scores. For measurement intercepts, the additional outputs are the observed means difference (`diff_mean`) and the proportion of the differences in the intercepts over the observed means differences (`propdiff`). For error variances, the additional outputs are the proportion of the difference in error variances over the difference in observed variances (`propdiff`).
2. `results`: Statistical tests as well as the change in CFI are provided. χ^2 and *p* value are provided for all methods.
3. `models`: The submodels used in the `free` and `fix` methods, as well as the nested and parent models. The nested and parent models will be changed from the original models if `free` or `fit` arguments are specified.

Author(s)

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

References

Millsap, R. E., & Olivera-Aguilar, M. (2012). Investigating measurement invariance using confirmatory factor analysis. In R. H. Hoyle (Ed.), *Handbook of structural equation modeling* (pp. 380–392). New York, NY: Guilford.

Examples

```
## Conduct weak invariance testing manually by using fixed-factor
## method of scale identification

library(lavaan)

conf <- "
f1 =~ NA*x1 + x2 + x3
f2 =~ NA*x4 + x5 + x6
f1 ~~ c(1, 1)*f1
f2 ~~ c(1, 1)*f2
"

weak <- "
f1 =~ NA*x1 + x2 + x3
f2 =~ NA*x4 + x5 + x6
f1 ~~ c(1, NA)*f1
f2 ~~ c(1, NA)*f2
"

configural <- cfa(conf, data = HolzingerSwineford1939, std.lv = TRUE, group="school")
weak <- cfa(weak, data = HolzingerSwineford1939, group="school", group.equal="loadings")
models <- list(fit.configural = configural, fit.loadings = weak)
partialInvariance(models, "metric")
```

`PAVranking` *Parcel-Allocation Variability in Model Ranking*

Description

This function quantifies and assesses the consequences of parcel-allocation variability for model ranking of structural equation models (SEMs) that differ in their structural specification but share the same parcel-level measurement specification (see Sterba & Rights, 2016). This function calls `parcelAllocation()`—which can be used with only one SEM in isolation—to fit two (assumed) nested models to each of a specified number of random item-to-parcel allocations. Output includes summary information about the distribution of model selection results (including plots) and the distribution of results for each model individually, across allocations within-sample. Note that this function can be used when selecting among more than two competing structural models as well (see instructions below involving the seed= argument).

Usage

```
PAVranking(model0, model1, data, parcel.names, item.syntax, nAlloc = 100,
  fun = "sem", alpha = 0.05, bic.crit = 10, fit.measures = c("chisq",
  "df", "cfi", "tli", "rmsea", "srmr", "logl", "aic", "bic", "bic2"), ...,
  show.progress = FALSE, iseed = 12345, warn = FALSE)
```

Arguments

`model0, model1`	`lavaan::lavaan()` model syntax specifying nested models (`model0` within `model1`) to be fitted to the same parceled data. Note that there can be a mixture of items and parcels (even within the same factor), in case certain items should never be parceled. Can be a character string or parameter table. Also see `lavaan::lavaanify()` for more details.
`data`	A `data.frame` containing all observed variables appearing in `model0=` and `model1=`, as well as those in the `item.syntax=` used to create parcels. If the data have missing values, multiple imputation before parceling is recommended: submit a stacked data set (with a variable for the imputation number, so they can be separated later) and set `do.fit=FALSE` to return the list of `data.frames` (one per allocation), each of which is a stacked, multiply imputed data set with parcels created using the same allocation scheme.
`parcel.names`	`character` vector containing names of all parcels appearing as indicators in `model0=` or `model1=`.
`item.syntax`	`lavaan::lavaan()` model syntax specifying the model that would be fit to all of the unparceled items, including items that should be randomly allocated to parcels appearing in `model0=` and `model1=`.
`nAlloc`	The number of random items-to-parcels allocations to generate.
`fun`	`character` string indicating the name of the `lavaan::lavaan()` function used to fit `model0=` and `model1=` to `data=`. Can only take the values `"lavaan"`, `"sem"`, `"cfa"`, or `"growth"`.
`alpha`	Alpha level used as criterion for significance.
`bic.crit`	Criterion for assessing evidence in favor of one model over another. See Rafferty (1995) for guidelines (default is "very strong evidence" in favor of the model with lower BIC).
`fit.measures`	`character` vector containing names of fit measures to request from each fitted lavaan::lavaan model. See the output of `lavaan::fitMeasures()` for a list of available measures.
`...`	Additional arguments to be passed to `lavaan::lavaanList()`. See also `lavaan::lavOptions()`
`show.progress`	If `TRUE`, show a `utils::txtProgressBar()` indicating how fast each model-fitting iterates over allocations.
`iseed`	(Optional) Random seed used for parceling items. When the same random seed is specified and the program is re-run, the same allocations will be generated. The seed argument can be used to assess parcel-allocation variability in model ranking when considering more than two models. For each pair of models under comparison, the program should be rerun using the same random seed. Doing so ensures that multiple model comparisons will employ the same set of parcel datasets. *Note*: When using **parallel** options, you must first type `RNGkind("L'Ecuyer-CMRG")` into the R Console, so that the seed will be controlled across cores.
`warn`	Whether to print warnings when fitting models to each allocation

Details

This is based on a SAS macro `ParcelAlloc` (Sterba & MacCallum, 2010). The `PAVranking()` function produces results discussed in Sterba and Rights (2016) relevant to the assessment of parcel-allocation variability in model selection and model ranking. Specifically, the `PAVranking()` function first calls `parcelAllocation()` to generate a given number (`nAlloc=`) of item-to-parcel allocations, fitting both specified models to each allocation, and providing summaryies of PAV for each model. Additionally, `PAVranking()` provides the following new summaries:

- PAV in model selection index values and model ranking between Models `model0=` and `model1=`.
- The proportion of allocations that converged and the proportion of proper solutions (results are summarized for allocations with both converged and proper allocations only).

For further details on the benefits of the random allocation of items to parcels, see Sterba (2011) and Sterba and MacCallum (2010).

To test whether nested models have equivalent fit, results can be pooled across allocations using the same methods available for pooling results across multiple imputations of missing data (see **Examples**).

Note: This function requires the `lavaan` package. Missing data must be coded as `NA`. If the function returns `"Error in plot.new() : figure margins too large"`, the user may need to increase size of the plot window (e.g., in RStudio) and rerun the function.

Value

A `list` with 3 elements. The first two (`model0.results` and `model1.results`) are results returned by `parcelAllocation()` for `model0` and `model1`, respectively. The third element (`model0.v.model1`) is a `list` of model-comparison results, including the following:

`\verb{LRT_Summary:}`	The average likelihood ratio test across allocations, as well as the *SD*, minimum, maximum, range, and the proportion of allocations for which the test was significant.
`\verb{Fit_Index_Differences:}`	Differences in fit indices, organized by what proportion favored each model and among those, what the average difference was.
`\verb{Favored_by_BIC:}`	The proportion of allocations in which each model met the criterion (`bic.crit`) for a substantial difference in fit.
`\verb{Convergence_Summary:}`	The proportion of allocations in which each model (and both models) converged on a solution.

Histograms are also printed to the current plot-output device.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Raftery, A. E. (1995). Bayesian model selection in social research. *Sociological Methodology, 25*, 111–163. doi:10.2307/271063

Sterba, S. K. (2011). Implications of parcel-allocation variability for comparing fit of item-solutions and parcel-solutions. *Structural Equation Modeling, 18*(4), 554–577.doi:10.1080/10705511.2011.607073

Sterba, S. K., & MacCallum, R. C. (2010). Variability in parameter estimates and model fit across repeated allocations of items to parcels. *Multivariate Behavioral Research, 45*(2), 322–358. doi:10.1080/00273171003680302

Sterba, S. K., & Rights, J. D. (2016). Accounting for parcel-allocation variability in practice: Combining sources of uncertainty and choosing the number of allocations. *Multivariate Behavioral Research, 51*(2–3), 296–313. doi:10.1080/00273171.2016.1144502

Sterba, S. K., & Rights, J. D. (2017). Effects of parceling on model selection: Parcel-allocation variability in model ranking. *Psychological Methods, 22*(1), 47–68. doi:10.1037/met0000067

See Also

`parcelAllocation()` for fitting a single model, `poolMAlloc()` for choosing the number of allocations

Examples

```
## Specify the item-level model (if NO parcels were created)
## This must apply to BOTH competing models

item.syntax <- c(paste0("f1 =~ f1item", 1:9),
                 paste0("f2 =~ f2item", 1:9))
cat(item.syntax, sep = "\n")
## Below, we reduce the size of this same model by
## applying different parceling schemes

## Specify a 2-factor CFA with correlated factors, using 3-indicator parcels
mod1 <- '
f1 =~ par1 + par2 + par3
f2 =~ par4 + par5 + par6
'
## Specify a more restricted model with orthogonal factors
mod0 <- '
f1 =~ par1 + par2 + par3
f2 =~ par4 + par5 + par6
f1 ~~ 0*f2
'
## names of parcels (must apply to BOTH models)
(parcel.names <- paste0("par", 1:6))


## override default random-number generator to use parallel options
RNGkind("L'Ecuyer-CMRG")

PAVranking(model0 = mod0, model1 = mod1, data = simParcel, nAlloc = 100,
```

```
            parcel.names = parcel.names, item.syntax = item.syntax,
            # parallel = "multicore",   # parallel available on Mac/Linux
            std.lv = TRUE)       # any addition lavaan arguments



## POOL RESULTS by treating parcel allocations as multiple imputations.
## Details provided in Sterba & Rights (2016); see ?poolMAlloc.

## save list of data sets instead of fitting model yet
dataList <- parcelAllocation(mod0, # or mod1 (either uses same allocations)
                             data = simParcel, nAlloc = 100,
                             parcel.names = parcel.names,
                             item.syntax = item.syntax,
                             do.fit = FALSE)
## now fit each model to each data set
if(requireNamespace("lavaan.mi")){
  library(lavaan.mi)
  fit0 <- cfa.mi(mod0, data = dataList, std.lv = TRUE)
  fit1 <- cfa.mi(mod1, data = dataList, std.lv = TRUE)
  anova(fit0, fit1)           # Pooled test statistic comparing models.
  help(package = "lavaan.mi") # Find more methods for pooling results.
}
```

`permuteMeasEq` *Permutation Randomization Tests of Measurement Equivalence and Differential Item Functioning (DIF)*

Description

The function `permuteMeasEq` provides tests of hypotheses involving measurement equivalence, in one of two frameworks: multigroup CFA or MIMIC models.

Usage

```
permuteMeasEq(nPermute, modelType = c("mgcfa", "mimic"), con, uncon = NULL,
  null = NULL, param = NULL, freeParam = NULL, covariates = NULL,
  AFIs = NULL, moreAFIs = NULL, maxSparse = 10, maxNonconv = 10,
  showProgress = TRUE, warn = -1, datafun, extra,
  parallelType = c("none", "multicore", "snow"), ncpus = NULL, cl = NULL,
  iseed = 12345)
```

Arguments

`nPermute`	An integer indicating the number of random permutations used to form empirical distributions under the null hypothesis.

`modelType`	A character string indicating type of model employed: multiple-group CFA (`"mgcfa"`) or MIMIC (`"mimic"`).
`con`	The constrained `lavaan` object, in which the parameters specified in `param` are constrained to equality across all groups when `modelType = "mgcfa"`, or which regression paths are fixed to zero when `modelType = "mimic"`. In the case of testing *configural* invariance when `modelType = "mgcfa"`, `con` is the configural model (implicitly, the unconstrained model is the saturated model, so use the defaults `uncon = NULL` and `param = NULL`). When `modelType = "mimic"`, `con` is the MIMIC model in which the covariate predicts the latent construct(s) but no indicators (unless they have already been identified as DIF items).
`uncon`	Optional. The unconstrained `lavaan` object, in which the parameters specified in `param` are freely estimated in all groups. When `modelType = "mgcfa"`, only in the case of testing *configural* invariance should `uncon = NULL`. When `modelType = "mimic"`, any non-`NULL uncon` is silently set to `NULL`.
`null`	Optional. A `lavaan` object, in which an alternative null model is fit (besides the default independence model specified by `lavaan`) for the calculation of incremental fit indices. See Widamin & Thompson (2003) for details. If `NULL`, `lavaan`'s default independence model is used.
`param`	An optional character vector or list of character vectors indicating which parameters the user would test for DIF following a rejection of the omnibus null hypothesis tested using `(more)AFIs`. Note that `param` does not guarantee certain parameters *are* constrained in `con`; that is for the user to specify when fitting the model. If users have any "anchor items" that they would never intend to free across groups (or levels of a covariate), these should be excluded from `param`; exceptions to a type of parameter can be specified in `freeParam`. When `modelType = "mgcfa"`, `param` indicates which parameters of interest are constrained across groups in `con` and are unconstrained in `uncon`. Parameter names must match those returned by `names(coef(con))`, but omitting any group-specific suffixes (e.g., `"f1~1"` rather than `"f1~1.g2"`) or user-specified labels (that is, the parameter names must follow the rules of lavaan's `lavaan::model.syntax()`). Alternatively (or additionally), to test all constraints of a certain type (or multiple types) of parameter in `con`, `param` may take any combination of the following values: `"loadings"`, `"intercepts"`, `"thresholds"`, `"residuals"`, `"residual.covariances"`, `"means"`, `"lv.variances"`, and/or `"lv.covariances"`. When `modelType = "mimic"`, `param` must be a vector of individual parameters or a list of character strings to be passed one-at-a-time to `lavaan::lavTestScore(object = con, add = param[i])`, indicating which (sets of) regression paths fixed to zero in `con` that the user would consider freeing (i.e., exclude anchor items). If `modelType = "mimic"` and `param` is a list of character strings, the multivariate test statistic will be saved for each list element instead of 1-*df* modification indices for each individual parameter, and `names(param)` will name the rows of the `MI.obs` slot (see permuteMeasEq). Set `param = NULL` (default) to avoid collecting modification indices for any follow-up tests.
`freeParam`	An optional character vector, silently ignored when `modelType = "mimic"`. If `param` includes a type of parameter (e.g., `"loadings"`), `freeParam` indicates exceptions (i.e., anchor items) that the user would *not* intend to free across groups and should therefore be ignored when calculating *p* values adjusted for

	the number of follow-up tests. Parameter types that are already unconstrained across groups in the fitted `con` model (i.e., a *partial* invariance model) will automatically be ignored, so they do not need to be specified in `freeParam`. Parameter names must match those returned by `names(coef(con))`, but omitting any group-specific suffixes (e.g., `"f1~1"` rather than `"f1~1.g2"`) or user-specified labels (that is, the parameter names must follow the rules of lavaan `lavaan::model.syntax()`).
`covariates`	An optional character vector, only applicable when `modelType = "mimic"`. The observed data are partitioned into columns indicated by `covariates`, and the rows are permuted simultaneously for the entire set before being merged with the remaining data. Thus, the covariance structure is preserved among the covariates, which is necessary when (e.g.) multiple dummy codes are used to represent a discrete covariate or when covariates interact. If `covariates = NULL` when `modelType = "mimic"`, the value of `covariates` is inferred by searching `param` for predictors (i.e., variables appearing after the "~" operator).
`AFIs`	A character vector indicating which alternative fit indices (or chi-squared itself) are to be used to test the multiparameter omnibus null hypothesis that the constraints specified in `con` hold in the population. Any fit measures returned by `lavaan::fitMeasures()` may be specified (including constants like `"df"`, which would be nonsensical). If both `AFIs` and `moreAFIs` are `NULL`, only `"chisq"` will be returned.
`moreAFIs`	Optional. A character vector indicating which (if any) alternative fit indices returned by `moreFitIndices()` are to be used to test the multiparameter omnibus null hypothesis that the constraints specified in `con` hold in the population.
`maxSparse`	Only applicable when `modelType = "mgcfa"` and at least one indicator is `ordered`. An integer indicating the maximum number of consecutive times that randomly permuted group assignment can yield a sample in which at least one category (of an `ordered` indicator) is unobserved in at least one group, such that the same set of parameters cannot be estimated in each group. If such a sample occurs, group assignment is randomly permuted again, repeatedly until a sample is obtained with all categories observed in all groups. If `maxSparse` is exceeded, `NA` will be returned for that iteration of the permutation distribution.
`maxNonconv`	An integer indicating the maximum number of consecutive times that a random permutation can yield a sample for which the model does not converge on a solution. If such a sample occurs, permutation is attempted repeatedly until a sample is obtained for which the model does converge. If `maxNonconv` is exceeded, `NA` will be returned for that iteration of the permutation distribution, and a warning will be printed when using `show` or `summary`.
`showProgress`	Logical. Indicating whether to display a progress bar while permuting. Silently set to `FALSE` when using parallel options.
`warn`	Sets the handling of warning messages when fitting model(s) to permuted data sets. See `base::options()`.
`datafun`	An optional function that can be applied to the data (extracted from `con`) after each permutation, but before fitting the model(s) to each permutation. The `datafun` function must have an argument named `data` that accepts a `data.frame`, and it must return a `data.frame` containing the same column names. The column order may differ, the values of those columns may differ (so be careful!),

	and any additional columns will be ignored when fitting the model, but an error will result if any column names required by the model syntax do not appear in the transformed data set. Although available for any `modelType`, `datafun` may be useful when using the MIMIC method to test for nonuniform DIF (metric/weak invariance) by using product indicators for a latent factor representing the interaction between a factor and one of the `covariates`, in which case the product indicators would need to be recalculated after each permutation of the `covariates`. To access other R objects used within `permuteMeasEq`, the arguments to `datafun` may also contain any subset of the following: `"con"`, `"uncon"`, `"null"`, `"param"`, `"freeParam"`, `"covariates"`, `"AFIs"`, `"moreAFIs"`, `"maxSparse"`, `"maxNonconv"`, and/or `"iseed"`. The values for those arguments will be the same as the values supplied to `permuteMeasEq`.
`extra`	An optional function that can be applied to any (or all) of the fitted lavaan objects (`con`, `uncon`, and/or `null`). This function will also be applied after fitting the model(s) to each permuted data set. To access the R objects used within `permuteMeasEq`, the arguments to `extra` must be any subset of the following: `"con"`, `"uncon"`, `"null"`, `"param"`, `"freeParam"`, `"covariates"`, `"AFIs"`, `"moreAFIs"`, `"maxSparse"`, `"maxNonconv"`, and/or `"iseed"`. The values for those arguments will be the same as the values supplied to `permuteMeasEq`. The `extra` function must return a named `numeric` vector or a named `list` of scalars (i.e., a `list` of `numeric` vectors of `length == 1`). Any unnamed elements (e.g., `""` or `NULL`) of the returned object will result in an error.
`parallelType`	The type of parallel operation to be used (if any). The default is `"none"`. Forking is not possible on Windows, so if `"multicore"` is requested on a Windows machine, the request will be changed to `"snow"` with a message.
`ncpus`	Integer: number of processes to be used in parallel operation. If `NULL` (the default) and `parallelType %in% c("multicore","snow")`, the default is one less than the maximum number of processors detected by `parallel::detectCores()`. This default is also silently set if the user specifies more than the number of processors detected.
`cl`	An optional **parallel** or **snow** cluster for use when `parallelType = "snow"`. If `NULL`, a `"PSOCK"` cluster on the local machine is created for the duration of the `permuteMeasEq` call. If a valid `parallel::makeCluster()` object is supplied, `parallelType` is silently set to `"snow"`, and `ncpus` is silently set to `length(cl)`.
`iseed`	Integer: Only used to set the states of the RNG when using parallel options, in which case `base::RNGkind()` is set to `"L'Ecuyer-CMRG"` with a message. See `parallel::clusterSetRNGStream()` and Section 6 of `vignette("parallel", "parallel")` for more details. If user supplies an invalid value, `iseed` is silently set to the default (12345). To set the state of the RNG when not using parallel options, call `base::set.seed()` before calling `permuteMeasEq`.

Details

The function `permuteMeasEq` provides tests of hypotheses involving measurement equivalence, in one of two frameworks:

1. 1 For multiple-group CFA models, provide a pair of nested lavaan objects, the less constrained of which (`uncon`) freely estimates a set of measurement parameters (e.g., factor loadings,

intercepts, or thresholds; specified in `param`) in all groups, and the more constrained of which (`con`) constrains those measurement parameters to equality across groups. Group assignment is repeatedly permuted and the models are fit to each permutation, in order to produce an empirical distribution under the null hypothesis of no group differences, both for (a) changes in user-specified fit measures (see `AFIs` and `moreAFIs`) and for (b) the maximum modification index among the user-specified equality constraints. Configural invariance can also be tested by providing that fitted lavaan object to `con` and leaving `uncon = NULL`, in which case `param` must be `NULL` as well.

2. 2 In MIMIC models, one or a set of continuous and/or discrete `covariates` can be permuted, and a constrained model is fit to each permutation in order to provide a distribution of any fit measures (namely, the maximum modification index among fixed parameters in `param`) under the null hypothesis of measurement equivalence across levels of those covariates.

In either framework, modification indices for equality constraints or fixed parameters specified in `param` are calculated from the constrained model (`con`) using the function `lavaan::lavTestScore()`.

For multiple-group CFA models, the multiparameter omnibus null hypothesis of measurement equivalence/invariance is that there are no group differences in any measurement parameters (of a particular type). This can be tested using the `anova` method on nested `lavaan` objects, or by inspecting the change in alternative fit indices (AFIs) such as the CFI. The permutation randomization method employed by `permuteMeasEq` generates an empirical distribution of any `AFIs` under the null hypothesis, so the user is not restricted to using fixed cutoffs proposed by Cheung & Rensvold (2002), Chen (2007), or Meade, Johnson, & Braddy (2008).

If the multiparameter omnibus null hypothesis is rejected, partial invariance can still be established by freeing invalid equality constraints, as long as equality constraints are valid for at least two indicators per factor. Modification indices can be calculated from the constrained model (`con`), but multiple testing leads to inflation of Type I error rates. The permutation randomization method employed by `permuteMeasEq` creates a distribution of the maximum modification index if the null hypothesis is true, which allows the user to control the familywise Type I error rate in a manner similar to Tukey's *q* (studentized range) distribution for the Honestly Significant Difference (HSD) post hoc test.

For MIMIC models, DIF can be tested by comparing modification indices of regression paths to the permutation distribution of the maximum modification index, which controls the familywise Type I error rate. The MIMIC approach could also be applied with multiple-group models, but the grouping variable would not be permuted; rather, the covariates would be permuted separately within each group to preserve between-group differences. So whether parameters are constrained or unconstrained across groups, the MIMIC approach is only for testing null hypotheses about the effects of `covariates` on indicators, controlling for common factors.

In either framework, `lavaan::lavaan()`'s `group.label` argument is used to preserve the order of groups seen in `con` when permuting the data.

Value

The permuteMeasEq object representing the results of testing measurement equivalence (the multiparameter omnibus test) and DIF (modification indices), as well as diagnostics and any `extra` output.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Papers about permutation tests of measurement equivalence:

Jorgensen, T. D., Kite, B. A., Chen, P.-Y., & Short, S. D. (2018). Permutation randomization methods for testing measurement equivalence and detecting differential item functioning in multiple-group confirmatory factor analysis. *Psychological Methods, 23*(4), 708–728. doi:10.1037/met0000152

Kite, B. A., Jorgensen, T. D., & Chen, P.-Y. (2018). Random permutation testing applied to measurement invariance testing with ordered-categorical indicators. *Structural Equation Modeling 25*(4), 573–587. doi:10.1080/10705511.2017.1421467

Jorgensen, T. D. (2017). Applying permutation tests and multivariate modification indices to configurally invariant models that need respecification. *Frontiers in Psychology, 8*(1455). doi:10.3389/fpsyg.2017.01455

Additional reading:

Chen, F. F. (2007). Sensitivity of goodness of fit indexes to lack of measurement invariance. *Structural Equation Modeling, 14*(3), 464–504. doi:10.1080/10705510701301834

Cheung, G. W., & Rensvold, R. B. (2002). Evaluating goodness-of-fit indexes for testing measurement invariance. *Structural Equation Modeling, 9*(2), 233–255. doi:10.1207/S15328007SEM0902_5

Meade, A. W., Johnson, E. C., & Braddy, P. W. (2008). Power and sensitivity of alternative fit indices in tests of measurement invariance. *Journal of Applied Psychology, 93*(3), 568–592. doi:10.1037/00219010.93.3.568

Widamin, K. F., & Thompson, J. S. (2003). On specifying the null model for incremental fit indices in structural equation modeling. *Psychological Methods, 8*(1), 16–37. doi:10.1037/1082-989X.8.1.16

See Also

`stats::TukeyHSD()`, `lavaan::lavTestScore()`

Examples

```
########################
## Multiple-Group CFA ##
########################

## create 3-group data in lavaan example(cfa) data
HS <- lavaan::HolzingerSwineford1939
HS$ageGroup <- ifelse(HS$ageyr < 13, "preteen",
                      ifelse(HS$ageyr > 13, "teen", "thirteen"))

## specify and fit an appropriate null model for incremental fit indices
mod.null <- c(paste0("x", 1:9, " ~ c(T", 1:9, ", T", 1:9, ", T", 1:9, ")*1"),
              paste0("x", 1:9, " ~~ c(L", 1:9, ", L", 1:9, ", L", 1:9, ")*x", 1:9))
```

```
fit.null <- cfa(mod.null, data = HS, group = "ageGroup")

## fit target model with varying levels of measurement equivalence
mod.config <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
'
fit.config <- cfa(mod.config, data = HS, std.lv = TRUE, group = "ageGroup")
fit.metric <- cfa(mod.config, data = HS, std.lv = TRUE, group = "ageGroup",
                  group.equal = "loadings")
fit.scalar <- cfa(mod.config, data = HS, std.lv = TRUE, group = "ageGroup",
                  group.equal = c("loadings","intercepts"))


####################### Permutation Method

## fit indices of interest for multiparameter omnibus test
myAFIs <- c("chisq","cfi","rmsea","mfi","aic")
moreAFIs <- c("gammaHat","adjGammaHat")

## Use only 20 permutations for a demo.  In practice,
## use > 1000 to reduce sampling variability of estimated p values

## test configural invariance
set.seed(12345)
out.config <- permuteMeasEq(nPermute = 20, con = fit.config)
out.config

## test metric equivalence
set.seed(12345) # same permutations
out.metric <- permuteMeasEq(nPermute = 20, uncon = fit.config, con = fit.metric,
                            param = "loadings", AFIs = myAFIs,
                            moreAFIs = moreAFIs, null = fit.null)
summary(out.metric, nd = 4)

## test scalar equivalence
set.seed(12345) # same permutations
out.scalar <- permuteMeasEq(nPermute = 20, uncon = fit.metric, con = fit.scalar,
                            param = "intercepts", AFIs = myAFIs,
                            moreAFIs = moreAFIs, null = fit.null)
summary(out.scalar)

## Not much to see without significant DIF.
## Try using an absurdly high alpha level for illustration.
outsum <- summary(out.scalar, alpha = .50)

## notice that the returned object is the table of DIF tests
outsum

## visualize permutation distribution
hist(out.config, AFI = "chisq")
hist(out.metric, AFI = "chisq", nd = 2, alpha = .01,
```

```
     legendArgs = list(x = "topright"))
hist(out.scalar, AFI = "cfi", printLegend = FALSE)


######################## Extra Output

## function to calculate expected change of Group-2 and -3 latent means if
## each intercept constraint were released
extra <- function(con) {
  output <- list()
  output["x1.vis2"] <- lavTestScore(con, release = 19:20, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[70]
  output["x1.vis3"] <- lavTestScore(con, release = 19:20, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[106]
  output["x2.vis2"] <- lavTestScore(con, release = 21:22, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[70]
  output["x2.vis3"] <- lavTestScore(con, release = 21:22, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[106]
  output["x3.vis2"] <- lavTestScore(con, release = 23:24, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[70]
  output["x3.vis3"] <- lavTestScore(con, release = 23:24, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[106]
  output["x4.txt2"] <- lavTestScore(con, release = 25:26, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[71]
  output["x4.txt3"] <- lavTestScore(con, release = 25:26, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[107]
  output["x5.txt2"] <- lavTestScore(con, release = 27:28, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[71]
  output["x5.txt3"] <- lavTestScore(con, release = 27:28, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[107]
  output["x6.txt2"] <- lavTestScore(con, release = 29:30, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[71]
  output["x6.txt3"] <- lavTestScore(con, release = 29:30, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[107]
  output["x7.spd2"] <- lavTestScore(con, release = 31:32, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[72]
  output["x7.spd3"] <- lavTestScore(con, release = 31:32, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[108]
  output["x8.spd2"] <- lavTestScore(con, release = 33:34, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[72]
  output["x8.spd3"] <- lavTestScore(con, release = 33:34, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[108]
  output["x9.spd2"] <- lavTestScore(con, release = 35:36, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[72]
  output["x9.spd3"] <- lavTestScore(con, release = 35:36, univariate = FALSE,
                                    epc = TRUE, warn = FALSE)$epc$epc[108]
  output
}

## observed EPC
extra(fit.scalar)

## permutation results, including extra output
```

```
set.seed(12345) # same permutations
out.scalar <- permuteMeasEq(nPermute = 20, uncon = fit.metric, con = fit.scalar,
                            param = "intercepts", AFIs = myAFIs,
                            moreAFIs = moreAFIs, null = fit.null, extra = extra)
## summarize extra output
summary(out.scalar, extra = TRUE)


###########
## MIMIC ##
###########

## Specify Restricted Factor Analysis (RFA) model, equivalent to MIMIC, but
## the factor covaries with the covariate instead of being regressed on it.
## The covariate defines a single-indicator construct, and the
## double-mean-centered products of the indicators define a latent
## interaction between the factor and the covariate.
mod.mimic <- '
visual  =~ x1 + x2 + x3
age =~ ageyr
age.by.vis =~ x1.ageyr + x2.ageyr + x3.ageyr

x1 ~~ x1.ageyr
x2 ~~ x2.ageyr
x3 ~~ x3.ageyr
'

HS.orth <- indProd(var1 = paste0("x", 1:3), var2 = "ageyr", match = FALSE,
                   data = HS[ , c("ageyr", paste0("x", 1:3))] )
fit.mimic <- cfa(mod.mimic, data = HS.orth, meanstructure = TRUE)
summary(fit.mimic, stand = TRUE)

## Whereas MIMIC models specify direct effects of the covariate on an indicator,
## DIF can be tested in RFA models by specifying free loadings of an indicator
## on the covariate's construct (uniform DIF, scalar invariance) and the
## interaction construct (nonuniform DIF, metric invariance).
param <- as.list(paste0("age + age.by.vis =~ x", 1:3))
names(param) <- paste0("x", 1:3)
# param <- as.list(paste0("x", 1:3, " ~ age + age.by.vis")) # equivalent

## test both parameters simultaneously for each indicator
do.call(rbind, lapply(param, function(x) lavTestScore(fit.mimic, add = x)$test))
## or test each parameter individually
lavTestScore(fit.mimic, add = as.character(param))


####################### Permutation Method

## function to recalculate interaction terms after permuting the covariate
datafun <- function(data) {
  d <- data[, c(paste0("x", 1:3), "ageyr")]
  indProd(var1 = paste0("x", 1:3), var2 = "ageyr", match = FALSE, data = d)
}
```

```
set.seed(12345)
perm.mimic <- permuteMeasEq(nPermute = 20, modelType = "mimic",
                            con = fit.mimic, param = param,
                            covariates = "ageyr", datafun = datafun)
summary(perm.mimic)
```

`permuteMeasEq-class` *Class for the Results of Permutation Randomization Tests of Measurement Equivalence and DIF*

Description

This class contains the results of tests of Measurement Equivalence and Differential Item Functioning (DIF).

Usage

```
## S4 method for signature 'permuteMeasEq'
show(object)

## S4 method for signature 'permuteMeasEq'
summary(object, alpha = 0.05, nd = 3,
  extra = FALSE)

## S4 method for signature 'permuteMeasEq'
hist(x, ..., AFI, alpha = 0.05, nd = 3,
  printLegend = TRUE, legendArgs = list(x = "topleft"))
```

Arguments

`object, x`	object of class `permuteMeasEq`
`alpha`	alpha level used to draw confidence limits in `hist` and flag significant statistics in `summary` output
`nd`	number of digits to display
`extra`	`logical` indicating whether the `summary` output should return permutation-based p values for each statistic returned by the `extra` function. If `FALSE` (default), `summary` will return permutation-based p values for each modification index.
`...`	Additional arguments to pass to `graphics::hist()`
`AFI`	`character` indicating the fit measure whose permutation distribution should be plotted
`printLegend`	`logical`. If `TRUE` (default), a legend will be printed with the histogram
`legendArgs`	`list` of arguments passed to the `graphics::legend()` function. The default argument is a list placing the legend at the top-left of the figure.

Value

- The `show` method prints a summary of the multiparameter omnibus test results, using the user-specified AFIs. The parametric $(\Delta)\chi^2$ test is also displayed.
- The `summary` method prints the same information from the `show` method, but when `extra = FALSE` (the default) it also provides a table summarizing any requested follow-up tests of DIF using modification indices in slot `MI.obs`. The user can also specify an `alpha` level for flagging modification indices as significant, as well as `nd` (the number of digits displayed). For each modification index, the *p* value is displayed using a central χ^2 distribution with the *df* shown in that column. Additionally, a *p* value is displayed using the permutation distribution of the maximum index, which controls the familywise Type I error rate in a manner similar to Tukey's studentized range test. If any indices are flagged as significant using the `tukey.p.value`, then a message is displayed for each flagged index. The invisibly returned `data.frame` is the displayed table of modification indices, unless `permuteMeasEq()` was called with `param = NULL`, in which case the invisibly returned object is `object`. If `extra = TRUE`, the permutation-based *p* values for each statistic returned by the `extra` function are displayed and returned in a `data.frame` instead of the modification indices requested in the `param` argument.
- The `hist` method returns a list of `length == 2`, containing the arguments for the call to `hist` and the arguments to the call for `legend`, respectively. This list may facilitate creating a customized histogram of `AFI.dist`, `MI.dist`, or `extra.dist`

Slots

`PT` A `data.frame` returned by a call to `lavaan::parTable()` on the constrained model

`modelType` A character indicating the specified `modelType` in the call to `permuteMeasEq`

`ANOVA` A `numeric` vector indicating the results of the observed $(\Delta)\chi^2$ test, based on the central χ^2 distribution

`AFI.obs` A vector of observed (changes in) user-selected fit measures

`AFI.dist` The permutation distribution(s) of user-selected fit measures. A `data.frame` with `n.Permutations` rows and one column for each `AFI.obs`.

`AFI.pval` A vector of *p* values (one for each element in slot `AFI.obs`) calculated using slot `AFI.dist`, indicating the probability of observing a change at least as extreme as `AFI.obs` if the null hypothesis were true

`MI.obs` A `data.frame` of observed Lagrange Multipliers (modification indices) associated with the equality constraints or fixed parameters specified in the `param` argument. This is a subset of the output returned by a call to `lavaan::lavTestScore()` on the constrained model.

`MI.dist` The permutation distribution of the maximum modification index (among those seen in slot `MI.obs$X2`) at each permutation of group assignment or of `covariates`

`extra.obs` If `permuteMeasEq` was called with an `extra` function, the output when applied to the original data is concatenated into this vector

`extra.dist` A `data.frame`, each column of which contains the permutation distribution of the corresponding statistic in slot `extra.obs`

`n.Permutations` An `integer` indicating the number of permutations requested by the user

`n.Converged` An `integer` indicating the number of permuation iterations which yielded a converged solution `n.nonConverged` An `integer` vector of length `n.Permutations` indicating how many times group assignment was randomly permuted (at each iteration) before converging on a solution

`n.Sparse` Only relevant with `ordered` indicators when `modelType == "mgcfa"`. An `integer` vector of length `n.Permutations` indicating how many times group assignment was randomly permuted (at each iteration) before obtaining a sample with all categories observed in all groups.

`oldSeed` An `integer` vector storing the value of `.Random.seed` before running `permuteMeasEq`. Only relevant when using a parallel/multicore option and the original `RNGkind() != "L'Ecuyer-CMRG"`. This enables users to restore their previous `.Random.seed` state, if desired, by running: `.Random.seed[-1] <- permutedResults@oldSeed[-1]`

Objects from the Class

Objects can be created via the `permuteMeasEq()` function.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

See Also

`permuteMeasEq()`

Examples

```
# See the example from the permuteMeasEq function
```

`plausibleValues` *Plausible-Values Imputation of Factor Scores Estimated from a lavaan Model*

Description

Draw plausible values of factor scores estimated from a fitted `lavaan::lavaan()` model, then treat them as multiple imputations of missing data using `lavaan.mi::lavaan.mi()`.

Usage

```
plausibleValues(object, nDraws = 20L, seed = 12345,
  omit.imps = c("no.conv", "no.se"), ...)
```

Arguments

`object`	A fitted model of class lavaan::lavaan, blavaan::blavaan, or lavaan.mi::lavaan.mi
`nDraws`	`integer` specifying the number of draws, analogous to the number of imputed data sets. If `object` is of class lavaan.mi::lavaan.mi, this will be the number of draws taken *per imputation*. If `object` is of class blavaan::blavaan, `nDraws` cannot exceed `blavInspect(object, "niter") * blavInspect(bfitc, "n.chains")` (number of MCMC samples from the posterior). The drawn samples will be evenly spaced (after permutation for `target="stan"`), using `ceiling()` to resolve decimals.
`seed`	`integer` passed to `set.seed()`.
`omit.imps`	`character` vector specifying criteria for omitting imputations when `object` is of class lavaan.mi::lavaan.mi. Can include any of `c("no.conv", "no.se", "no.npd")`.
`...`	Optional arguments to pass to `lavaan::lavPredict()`. `assemble` will be ignored because multiple groups are always assembled into a single `data.frame` per draw. `type` will be ignored because it is set internally to `type="lv"`.

Details

Because latent variables are unobserved, they can be considered as missing data, which can be imputed using Monte Carlo methods. This may be of interest to researchers with sample sizes too small to fit their complex structural models. Fitting a factor model as a first step, `lavaan::lavPredict()` provides factor-score estimates, which can be treated as observed values in a path analysis (Step 2). However, the resulting standard errors and test statistics could not be trusted because the Step-2 analysis would not take into account the uncertainty about the estimated factor scores. Using the asymptotic sampling covariance matrix of the factor scores provided by `lavaan::lavPredict()`, `plausibleValues` draws a set of `nDraws` imputations from the sampling distribution of each factor score, returning a list of data sets that can be treated like multiple imputations of incomplete data. If the data were already imputed to handle missing data, `plausibleValues` also accepts an object of class lavaan.mi::lavaan.mi, and will draw `nDraws` plausible values from each imputation. Step 2 would then take into account uncertainty about both missing values and factor scores. Bayesian methods can also be used to generate factor scores, as available with the **blavaan** package, in which case plausible values are simply saved parameters from the posterior distribution. See Asparouhov and Muthen (2010) for further technical details and references.

Each returned `data.frame` includes a `case.idx` column that indicates the corresponding rows in the data set to which the model was originally fitted (unless the user requests only Level-2 variables). This can be used to merge the plausible values with the original observed data, but users should note that including any new variables in a Step-2 model might not accurately account for their relationship(s) with factor scores because they were not accounted for in the Step-1 model from which factor scores were estimated.

If `object` is a multilevel `lavaan` model, users can request plausible values for latent variables at particular levels of analysis by setting the `lavaan::lavPredict()` argument `level=1` or `level=2`. If the `level` argument is not passed via `...`, then both levels are returned in a single merged data set per draw. For multilevel models, each returned `data.frame` also includes a column indicating to which cluster each row belongs (unless the user requests only Level-2 variables).

Value

A `list` of length `nDraws`, each of which is a `data.frame` containing plausible values, which can be treated as a `list` of imputed data sets to be passed to the `lavaan.mi` package (see **Examples**). If `object` is of class lavaan.mi::lavaan.mi, the `list` will be of length `nDraws*m`, where `m` is the number of imputations.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Asparouhov, T. & Muthen, B. O. (2010). *Plausible values for latent variables using M*plus. Technical Report. Retrieved from www.statmodel.com/download/Plausible.pdf

See Also

`lavaan.mi::lavaan.mi()`, lavaan.mi::lavaan.mi

Examples

```
## example from ?cfa and ?lavPredict help pages
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit1 <- cfa(HS.model, data = HolzingerSwineford1939)
fs1 <- plausibleValues(fit1, nDraws = 3,
                       ## lavPredict() can add only the modeled data
                       append.data = TRUE)
lapply(fs1, head)


## To merge factor scores to original data.frame (not just modeled data)
fs1 <- plausibleValues(fit1, nDraws = 3)
idx <- lavInspect(fit1, "case.idx")      # row index for each case
if (is.list(idx)) idx <- do.call(c, idx) # for multigroup models
data(HolzingerSwineford1939)             # copy data to workspace
HolzingerSwineford1939$case.idx <- idx   # add row index as variable
## loop over draws to merge original data with factor scores
for (i in seq_along(fs1)) {
  fs1[[i]] <- merge(fs1[[i]], HolzingerSwineford1939, by = "case.idx")
}
lapply(fs1, head)


## multiple-group analysis, in 2 steps
step1 <- cfa(HS.model, data = HolzingerSwineford1939, group = "school",
            group.equal = c("loadings","intercepts"))
PV.list <- plausibleValues(step1)
```

```
## subsequent path analysis
path.model <- ' visual ~ c(t1, t2)*textual + c(s1, s2)*speed '
if(requireNamespace("lavaan.mi")){
  library(lavaan.mi)
  step2 <- sem.mi(path.model, data = PV.list, group = "school")
  ## test equivalence of both slopes across groups
  lavTestWald.mi(step2, constraints = 't1 == t2 ; s1 == s2')
}


## multilevel example from ?Demo.twolevel help page
model <- '
  level: 1
    fw =~ y1 + y2 + y3
    fw ~ x1 + x2 + x3
  level: 2
    fb =~ y1 + y2 + y3
    fb ~ w1 + w2
'
msem <- sem(model, data = Demo.twolevel, cluster = "cluster")
mlPVs <- plausibleValues(msem, nDraws = 3) # both levels by default
lapply(mlPVs, head, n = 10)
## only Level 1
mlPV1 <- plausibleValues(msem, nDraws = 3, level = 1)
lapply(mlPV1, head)
## only Level 2
mlPV2 <- plausibleValues(msem, nDraws = 3, level = 2)
lapply(mlPV2, head)



## example with 20 multiple imputations of missing data:
nPVs <- 5
nImps <- 20

if (requireNamespace("lavaan.mi")) {
  data(HS20imps, package = "lavaan.mi")

  ## specify CFA model from lavaan's ?cfa help page
  HS.model <- '
    visual  =~ x1 + x2 + x3
    textual =~ x4 + x5 + x6
    speed   =~ x7 + x8 + x9
  '
  out2 <- cfa.mi(HS.model, data = HS20imps)
  PVs <- plausibleValues(out2, nDraws = nPVs)

  idx <- out2@Data@case.idx # can't use lavInspect() on lavaan.mi
  ## empty list to hold expanded imputations
  impPVs <- list()
  for (m in 1:nImps) {
    HS20imps[[m]]["case.idx"] <- idx
    for (i in 1:nPVs) {
```

```
      impPVs[[ nPVs*(m - 1) + i ]] <- merge(HS20imps[[m]],
                                            PVs[[ nPVs*(m - 1) + i ]],
                                            by = "case.idx")
    }
  }
  lapply(impPVs, head)
}
```

`plotProbe` *Plot a latent interaction*

Description

This function will plot the line graphs representing the simple effect of the independent variable given the values of the moderator. For multigroup models, it will only generate a plot for 1 group, as specified in the function used to obtain the first argument.

Usage

```
plotProbe(object, xlim, xlab = "Indepedent Variable",
  ylab = "Dependent Variable", legend = TRUE, legendArgs = list(), ...)
```

Arguments

`object`	A `list`, typically the result of probing a latent 2-way or 3-way interaction obtained from the `probe2WayMC()`, `probe2WayRC()`, `probe3WayMC()`, or `probe3WayRC()` functions.
`xlim`	The vector of two numbers: the minimum and maximum values of the independent variable
`xlab`	The label of the x-axis
`ylab`	The label of the y-axis
`legend`	`logical`. If `TRUE` (default), a legend is printed.
`legendArgs`	`list` of arguments passed to `legend()` function if `legend=TRUE`.
`...`	Any additional argument for the `plot()` function

Value

None. This function will plot the simple main effect only.

Note

If the `object` does not contain simple intercepts (i.e., if the `object$SimpleIntcept` element is `NULL`), then all simple intercepts are arbitrarily set to zero in order to plot the simple slopes. This may not be consistent with the fitted model, but was (up until version 0.5-7) the default behavior when the y-intercept was fixed to 0. In this case, although the relative steepness of simple slopes can still meaningfully be compared, the relative vertical positions of lines at any point along the *x*-axis should not be interpreted.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Schoemann, A. M., & Jorgensen, T. D. (2021). Testing and interpreting latent variable interactions using the `semTools` package. *Psych, 3*(3), 322–335. doi:10.3390/psych3030024

See Also

- `indProd()` For creating the indicator products with no centering, mean centering, double-mean centering, or residual centering.
- `probe2WayMC()` For probing the two-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe3WayMC()` For probing the three-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe2WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `probe3WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.

Examples

```
library(lavaan)

dat2wayMC <- indProd(dat2way, 1:3, 4:6)

model1 <- "
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f12 =~ x1.x4 + x2.x5 + x3.x6
f3 =~ x7 + x8 + x9
f3 ~ f1 + f2 + f12
f12 ~~ 0*f1
f12 ~~ 0*f2
x1 ~ 0*1
x4 ~ 0*1
x1.x4 ~ 0*1
x7 ~ 0*1
```

```
f1 ~ NA*1
f2 ~ NA*1
f12 ~ NA*1
f3 ~ NA*1
"

fitMC2way <- sem(model1, data = dat2wayMC, meanstructure = TRUE)
result2wayMC <- probe2WayMC(fitMC2way, nameX = c("f1", "f2", "f12"),
                            nameY = "f3", modVar = "f2", valProbe = c(-1, 0, 1))
plotProbe(result2wayMC, xlim = c(-2, 2))


dat3wayMC <- indProd(dat3way, 1:3, 4:6, 7:9)

model3 <- "
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f3 =~ x7 + x8 + x9
f12 =~ x1.x4 + x2.x5 + x3.x6
f13 =~ x1.x7 + x2.x8 + x3.x9
f23 =~ x4.x7 + x5.x8 + x6.x9
f123 =~ x1.x4.x7 + x2.x5.x8 + x3.x6.x9
f4 =~ x10 + x11 + x12
f4 ~ f1 + f2 + f3 + f12 + f13 + f23 + f123
f1 ~~ 0*f12
f1 ~~ 0*f13
f1 ~~ 0*f123
f2 ~~ 0*f12
f2 ~~ 0*f23
f2 ~~ 0*f123
f3 ~~ 0*f13
f3 ~~ 0*f23
f3 ~~ 0*f123
f12 ~~ 0*f123
f13 ~~ 0*f123
f23 ~~ 0*f123
x1 ~ 0*1
x4 ~ 0*1
x7 ~ 0*1
x10 ~ 0*1
x1.x4 ~ 0*1
x1.x7 ~ 0*1
x4.x7 ~ 0*1
x1.x4.x7 ~ 0*1
f1 ~ NA*1
f2 ~ NA*1
f3 ~ NA*1
f12 ~ NA*1
f13 ~ NA*1
f23 ~ NA*1
f123 ~ NA*1
f4 ~ NA*1
"
```

```
fitMC3way <- sem(model3, data = dat3wayMC, std.lv = FALSE,
                 meanstructure = TRUE)
result3wayMC <- probe3WayMC(fitMC3way, nameX = c("f1", "f2", "f3", "f12",
                                                 "f13", "f23", "f123"),
                            nameY = "f4", modVar = c("f1", "f2"),
                            valProbe1 = c(-1, 0, 1), valProbe2 = c(-1, 0, 1))
plotProbe(result3wayMC, xlim = c(-2, 2))
```

`plotRMSEAdist` *Plot the sampling distributions of RMSEA*

Description

Plots the sampling distributions of RMSEA based on the noncentral chi-square distributions

Usage

```
plotRMSEAdist(rmsea, n, df, ptile = NULL, caption = NULL,
  rmseaScale = TRUE, group = 1)
```

Arguments

`rmsea`	The vector of RMSEA values to be plotted
`n`	Sample size of a dataset
`df`	Model degrees of freedom
`ptile`	The percentile rank of the distribution of the first RMSEA that users wish to plot a vertical line in the resulting graph
`caption`	The name vector of each element of `rmsea`
`rmseaScale`	If `TRUE`, the RMSEA scale is used in the x-axis. If `FALSE`, the chi-square scale is used in the x-axis.
`group`	The number of group that is used to calculate RMSEA.

Details

This function creates overlappling plots of the sampling distribution of RMSEA based on noncentral χ^2 distribution (MacCallum, Browne, & Suguwara, 1996). First, the noncentrality parameter (λ) is calculated from RMSEA (Steiger, 1998; Dudgeon, 2004) by

$$\lambda = (N - 1)d\varepsilon^2/K,$$

where N is sample size, d is the model degree of freedom, K is the number of group, and ε is the population RMSEA. Next, the noncentral χ^2 distribution with a specified *df* and noncentrality parameter is plotted. Thus, the x-axis represents the sample χ^2 value. The sample χ^2 value can be transformed to the sample RMSEA scale ($\hat{\varepsilon}$) by

$$\hat{\varepsilon} = \sqrt{K}\sqrt{\frac{\chi^2 - d}{(N-1)d}},$$

where χ^2 is the χ^2 value obtained from the noncentral χ^2 distribution.

Author(s)

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

References

Dudgeon, P. (2004). A note on extending Steiger's (1998) multiple sample RMSEA adjustment to other noncentrality parameter-based statistic. *Structural Equation Modeling, 11*(3), 305–319. doi:10.1207/s15328007sem1103_1

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods, 1*(2), 130–149. doi:10.1037/1082989X.1.2.130

Steiger, J. H. (1998). A note on multiple sample extensions of the RMSEA fit index. *Structural Equation Modeling, 5*(4), 411–419. doi:10.1080/10705519809540115

See Also

- `plotRMSEApower()` to plot the statistical power based on population RMSEA given the sample size
- `findRMSEApower()` to find the statistical power based on population RMSEA given a sample size
- `findRMSEAsamplesize()` to find the minium sample size for a given statistical power based on population RMSEA

Examples

```
plotRMSEAdist(c(.05, .08), n = 200, df = 20, ptile = .95, rmseaScale = TRUE)
plotRMSEAdist(c(.05, .01), n = 200, df = 20, ptile = .05, rmseaScale = FALSE)
```

`plotRMSEApower` *Plot power curves for RMSEA*

Description

Plots power of RMSEA over a range of sample sizes

Usage

```
plotRMSEApower(rmsea0, rmseaA, df, nlow, nhigh, steps = 1, alpha = 0.05,
  group = 1, ...)
```

Arguments

`rmsea0`	Null RMSEA
`rmseaA`	Alternative RMSEA
`df`	Model degrees of freedom
`nlow`	Lower sample size
`nhigh`	Upper sample size
`steps`	Increase in sample size for each iteration. Smaller values of steps will lead to more precise plots. However, smaller step sizes means a longer run time.
`alpha`	Alpha level used in power calculations
`group`	The number of group that is used to calculate RMSEA.
`...`	The additional arguments for the plot function.

Details

This function creates plot of power for RMSEA against a range of sample sizes. The plot places sample size on the horizontal axis and power on the vertical axis. The user should indicate the lower and upper values for sample size and the sample size between each estimate ("step size") We strongly urge the user to read the sources below (see References) before proceeding. A web version of this function is available at: `https://quantpsy.org/rmsea/rmseaplot.htm`. This function is also implemented in the web application "power4SEM": `https://sjak.shinyapps.io/power4SEM/`

Value

Plot of power for RMSEA against a range of sample sizes

Author(s)

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

Kristopher J. Preacher (Vanderbilt University; <`kris.preacher@vanderbilt.edu`>)

Donna L. Coffman (Pennsylvania State University; <`dlc30@psu.edu`>)

References

MacCallum, R. C., Browne, M. W., & Cai, L. (2006). Testing differences between nested covariance structure models: Power analysis and null hypotheses. *Psychological Methods, 11*(1), 19–35. doi:10.1037/1082989X.11.1.19

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods, 1*(2), 130–149. doi:10.1037/1082989X.1.2.130

MacCallum, R. C., Lee, T., & Browne, M. W. (2010). The issue of isopower in power analysis for tests of structural equation models. *Structural Equation Modeling, 17*(1), 23–41. doi:10.1080/10705510903438906

Preacher, K. J., Cai, L., & MacCallum, R. C. (2007). Alternatives to traditional model comparison strategies for covariance structure models. In T. D. Little, J. A. Bovaird, & N. A. Card (Eds.), *Modeling contextual effects in longitudinal studies* (pp. 33–62). Mahwah, NJ: Lawrence Erlbaum Associates.

Steiger, J. H. (1998). A note on multiple sample extensions of the RMSEA fit index. *Structural Equation Modeling, 5*(4), 411–419. doi:10.1080/10705519809540115

Steiger, J. H., & Lind, J. C. (1980, June). *Statistically based tests for the number of factors.* Paper presented at the annual meeting of the Psychometric Society, Iowa City, IA.

Jak, S., Jorgensen, T. D., Verdam, M. G., Oort, F. J., & Elffers, L. (2021). Analytical power calculations for structural equation modeling: A tutorial and Shiny app. *Behavior Research Methods, 53*, 1385–1406. doi:10.3758/s13428020014790

See Also

- `plotRMSEAdist()` to visualize the RMSEA distributions
- `findRMSEApower()` to find the statistical power based on population RMSEA given a sample size
- `findRMSEAsamplesize()` to find the minium sample size for a given statistical power based on population RMSEA

Examples

```
plotRMSEApower(rmsea0 = .025, rmseaA = .075, df = 23,
               nlow = 100, nhigh = 500, steps = 10)
```

`plotRMSEApowernested` *Plot power of nested model RMSEA*

Description

Plot power of nested model RMSEA over a range of possible sample sizes.

Usage

```
plotRMSEApowernested(rmsea0A = NULL, rmsea0B = NULL, rmsea1A,
  rmsea1B = NULL, dfA, dfB, nlow, nhigh, steps = 1, alpha = 0.05,
  group = 1, ...)
```

Arguments

`rmsea0A`	The H_0 baseline RMSEA
`rmsea0B`	The H_0 alternative RMSEA (trivial misfit)
`rmsea1A`	The H_1 baseline RMSEA
`rmsea1B`	The H_1 alternative RMSEA (target misfit to be rejected)
`dfA`	degree of freedom of the more-restricted model
`dfB`	degree of freedom of the less-restricted model
`nlow`	Lower bound of sample size
`nhigh`	Upper bound of sample size
`steps`	Step size
`alpha`	The alpha level
`group`	The number of group in calculating RMSEA
`...`	The additional arguments for the plot function.

Author(s)

Bell Clinton

Pavel Panko (Texas Tech University; `<pavel.panko@ttu.edu>`)

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

References

MacCallum, R. C., Browne, M. W., & Cai, L. (2006). Testing differences between nested covariance structure models: Power analysis and null hypotheses. *Psychological Methods, 11*(1), 19–35. doi:10.1037/1082989X.11.1.19

See Also

- `findRMSEApowernested()` to find the power for a given sample size in nested model comparison based on population RMSEA
- `findRMSEAsamplesizenested()` to find the minium sample size for a given statistical power in nested model comparison based on population RMSEA

Examples

```
plotRMSEApowernested(rmsea0A = 0, rmsea0B = 0, rmsea1A = 0.06,
                     rmsea1B = 0.05, dfA = 22, dfB = 20, nlow = 50,
                     nhigh = 500, steps = 1, alpha = .05, group = 1)
```

`poolMAlloc` *Combine sampling variability with parcel-allocation variability by pooling results across M parcel-allocations*

Description

This function employs an iterative algorithm to pick the number of random item-to-parcel allocations needed to meet user-defined stability criteria for a fitted structural equation model (SEM) (see **Details** below for more information). Pooled point and standard-error estimates from this SEM can be outputted at this final selected number of allocations (however, it is more efficient to save the allocations and treat them as multiple imputations using `lavaan.mi::lavaan.mi()`; see **See Also** for links with examples). Additionally, new indices (see Sterba & Rights, 2016) are outputted for assessing the relative contributions of parcel-allocation variability vs. sampling variability in each estimate. At each iteration, this function generates a given number of random item-to-parcel allocations, fits a SEM to each allocation, pools estimates across allocations from that iteration, and then assesses whether stopping criteria are met. If stopping criteria are not met, the algorithm increments the number of allocations used (generating all new allocations).

Usage

```
poolMAlloc(nPerPar, facPlc, nAllocStart, nAllocAdd = 0,
  parceloutput = NULL, syntax, dataset, stopProp, stopValue,
  selectParam = NULL, indices = "default", double = FALSE,
  checkConv = FALSE, names = "default", leaveout = 0,
  useTotalAlloc = FALSE, ...)
```

Arguments

`nPerPar`	A list in which each element is a vector, corresponding to each factor, indicating sizes of parcels. If variables are left out of parceling, they should not be accounted for here (i.e., there should not be parcels of size "1").
`facPlc`	A list of vectors, each corresponding to a factor, specifying the item indicators of that factor (whether included in parceling or not). Either variable names or column numbers. Variables not listed will not be modeled or included in output datasets.
`nAllocStart`	The number of random allocations of items to parcels to generate in the first iteration of the algorithm.
`nAllocAdd`	The number of allocations to add with each iteration of the algorithm. Note that if only one iteration is desired, `nAllocAdd` can be set to 0 and results will be output for `nAllocStart` allocations only.
`parceloutput`	Optional `character`. Path (folder/directory) where *M* (the final selected number of allocations) parceled data sets will be outputted from the iteration where the algorithm met stopping criteria. Note for Windows users: file path must be specified using forward slashes (`/`), not backslashes (`\\`). See `base::path.expand()` for details. If `NULL` (default), nothing is saved to disk.

`syntax`	lavaan syntax that defines the model.
`dataset`	Item-level dataset
`stopProp`	Value used in defining stopping criteria of the algorithm (δ_a in Sterba & Rights, 2016). This is the minimum proportion of change (in any pooled parameter or pooled standard error estimate listed in `selectParam`) that is allowable from one iteration of the algorithm to the next. That is, change in pooled estimates and pooled standard errors from one iteration to the next must all be less than (`stopProp`) × (value from former iteration). Note that `stopValue` can override this criterion (see below). Also note that values less than .01 are unlikely to lead to more substantively meaningful precision. Also note that if only `stopValue` is a desired criterion, `stopProp` can be set to 0.
`stopValue`	Value used in defining stopping criteria of the algorithm (δ_b in Sterba & Rights, 2016). `stopValue` is a minimum allowable amount of absolute change (in any pooled parameter or pooled standard error estimate listed in `selectParam`) from one iteration of the algorithm to the next. For a given pooled estimate or pooled standard error, `stopValue` is only invoked as a stopping criteria when the minimum change required by `stopProp` is less than `stopValue`. Note that values less than .01 are unlikely to lead to more substantively meaningful precision. Also note that if only `stopProp` is a desired criterion, `stopValue` can be set to 0.
`selectParam`	(Optional) A list of the pooled parameters to be used in defining stopping criteria (i.e., `stopProp` and `stopValue`). These parameters should appear in the order they are listed in the lavaan syntax. By default, all pooled parameters are used. Note that `selectParam` should only contain freely-estimated parameters. In one example from Sterba & Rights (2016) `selectParam` included all free parameters except item intercepts and in another example `selectParam` included only structural parameters.
`indices`	Optional `character` vector indicating the names of available `lavaan::fitMeasures()` to be included in the output. The first and second elements should be a chi-squared test statistic and its associated degrees of freedom, both of which will be added if missing. If `"default"`, the indices will be `c("chisq", "df", "cfi", "tli", "rmsea","srmr")`. If a robust test statistic is requested (see `lavaan::lavOptions()`), `c("chisq","df")` will be replaced by `c("chisq.scaled","df.scaled")`. For the output to include both the naive and robust test statistics, `indices` should include both, but put the scaled test statistics first, as in `indices = c("chisq.scaled", "df.scaled", "chisq", "df")`
`double`	(Optional) If set to `TRUE`, requires stopping criteria (`stopProp` and `stopValue`) to be met for all parameters (in `selectParam`) for two consecutive iterations of the algorithm. By default, this is set to `FALSE`, meaning stopping criteria need only be met at one iteration of the algorithm.
`checkConv`	(Optional) If set to TRUE, function will output pooled estimates and standard errors from 10 iterations post-convergence.
`names`	(Optional) A character vector containing the names of parceled variables.
`leaveout`	(Optional) A vector of variables to be left out of randomized parceling. Either variable names or column numbers are allowed.

`useTotalAlloc`	(Optional) If set to `TRUE`, function will output a separate set of results that uses all allocations created by the algorithm, rather than M allocations (see "Allocations needed for stability" below). This distinction is further discussed in Sterba and Rights (2016).
`...`	Additional arguments to be passed to `lavaan::lavaan()`. See also `lavaan::lavOptions()`

Details

This function implements an algorithm for choosing the number of allocations (M; described in Sterba & Rights, 2016), pools point and standard-error estimates across these M allocations, and produces indices for assessing the relative contributions of parcel-allocation variability vs. sampling variability in each estimate.

To obtain pooled test statistics for model fit or model comparison, the `list` or parcel allocations can be passed to `lavaan.mi::lavaan.mi()` (find **Examples** on the help pages for `parcelAllocation()` and `PAVranking()`).

This function randomly generates a given number (`nAllocStart`) of item-to-parcel allocations, fits a SEM to each allocation, and then increments the number of allocations used (by `nAllocAdd`) until the pooled point and standard-error estimates fulfill stopping criteria (`stopProp` and `stopValue`, defined above). A summary of results from the model that was fit to the M allocations are returned.

Additionally, this function outputs the proportion of allocations with solutions that converged (using a maximum likelihood estimator) as well as the proportion of allocations with solutions that were converged and proper. The converged and proper solutions among the final M allocations are used in computing pooled results.

Additionally, after each iteration of the algorithm, information useful in monitoring the algorithm is outputted. The number of allocations used at that iteration, the proportion of pooled parameter estimates meeting stopping criteria at the previous iteration, the proportion of pooled standard errors meeting stopping criteria at the previous iteration, and the runtime of that iteration are outputted. When stopping criteria are satisfied, the full set of results are outputted.

For further details on the benefits of the random allocation of items to parcels, see Sterba (2011) and Sterba & MacCallum (2010).

Value

`Estimates`	A table containing pooled results across M allocations at the iteration where stopping criteria were met. Columns correspond to individual parameter name, pooled estimate, pooled standard error, p value for a z test of the parameter, normal-theory 95% CI, p value for a t test of the parameter (using df described in Sterba & Rights, 2016), and t-based 95% CI for the parameter.
`Fit`	A table containing results related to model fit from the M allocations at the iteration where stopping criteria were met. Columns correspond to fit index names, the mean of each index across allocations, the SD of each fit index across allocations, the minimum, maximum and range of each fit index across allocations, and the percent of the M allocations where the chi-square test of absolute fit was significant.
`Proportions`	A table containing the proportion of the final M allocations that (a) met the optimizer convergence criteria) and (b) converged to proper solutions. Note that

	pooled estimates, pooled standard errors, and other results are computed using only the converged, proper allocations.
`Stability`	The number of allocations (M) needed for stability, at which point the algorithm's stopping criteria (defined above) were met.
`Uncertainty`	Indices used to quantify uncertainty in estimates due to sample vs. allocation variability. A table containing individual parameter names, an estimate of the proportion of total variance of a pooled parameter estimate that is attributable to parcel-allocation variability (PPAV), and an estimate of the ratio of the between-allocation variance of a pooled parameter estimate to the within-allocation variance (RPAV). See Sterba & Rights (2016) for more detail.
`Time`	The total runtime of the function, in minutes. Note that the total runtime will be greater when the specified model encounters convergence problems for some allocations, as is the case with the `simParcel()` dataset used below.

Author(s)

Jason D. Rights (Vanderbilt University; <`jason.d.rights@vanderbilt.edu`>)

The author would also like to credit Corbin Quick and Alexander Schoemann for providing the original `parcelAllocation()` function (prior to its revision by Terrence D. Jorgensen) on which this function is based.

References

Sterba, S. K. (2011). Implications of parcel-allocation variability for comparing fit of item-solutions and parcel-solutions. *Structural Equation Modeling, 18*(4), 554–577. doi:10.1080/10705511.2011.607073

Sterba, S. K., & MacCallum, R. C. (2010). Variability in parameter estimates and model fit across random allocations of items to parcels. *Multivariate Behavioral Research, 45*(2), 322–358. doi:10.1080/00273171003680302

Sterba, S. K., & Rights, J. D. (2016). Accounting for parcel-allocation variability in practice: Combining sources of uncertainty and choosing the number of allocations. *Multivariate Behavioral Research, 51*(2–3), 296–313. doi:10.1080/00273171.2016.1144502

Sterba, S. K., & Rights, J. D. (2017). Effects of parceling on model selection: Parcel-allocation variability in model ranking. *Psychological Methods, 22*(1), 47–68. doi:10.1037/met0000067

See Also

`lavaan.mi::lavaan.mi()` for treating allocations as multiple imputations to pool results across allocations. See **Examples** on help pages for `parcelAllocation()` (when fitting a single model) and `PAVranking()` (when comparing 2 models).

Examples

```
## lavaan syntax: A 2 Correlated
## factor CFA model to be fit to parceled data

parmodel <- '
   f1 =~ NA*p1f1 + p2f1 + p3f1
```

```
    f2 =~ NA*p1f2 + p2f2 + p3f2
    p1f1 ~ 1
    p2f1 ~ 1
    p3f1 ~ 1
    p1f2 ~ 1
    p2f2 ~ 1
    p3f2 ~ 1
    p1f1 ~~ p1f1
    p2f1 ~~ p2f1
    p3f1 ~~ p3f1
    p1f2 ~~ p1f2
    p2f2 ~~ p2f2
    p3f2 ~~ p3f2
    f1 ~~ 1*f1
    f2 ~~ 1*f2
    f1 ~~ f2
'

## specify items for each factor
f1name <- colnames(simParcel)[1:9]
f2name <- colnames(simParcel)[10:18]

## run function
poolMAlloc(nPerPar = list(c(3,3,3), c(3,3,3)),
           facPlc = list(f1name, f2name), nAllocStart = 10, nAllocAdd = 10,
           syntax = parmodel, dataset = simParcel, stopProp = .03,
           stopValue = .03, selectParam = c(1:6, 13:18, 21),
           names = list("p1f1","p2f1","p3f1","p1f2","p2f2","p3f2"),
           double = FALSE, useTotalAlloc = FALSE)


## See examples on ?parcelAllocation and ?PAVranking for how to obtain
## pooled test statistics and other pooled lavaan output.
## Details provided in Sterba & Rights (2016).
```

`probe2WayMC` *Probing two-way interaction on the no-centered or mean-centered latent interaction*

Description

Probing interaction for simple intercept and simple slope for the no-centered or mean-centered latent two-way interaction

Usage

```
probe2WayMC(fit, nameX, nameY, modVar, valProbe, group = 1L,
  omit.imps = c("no.conv", "no.se"))
```

Arguments

`fit`	A fitted lavaan::lavaan or lavaan.mi::lavaan.mi object with a latent 2-way interaction.
`nameX`	`character` vector of all 3 factor names used as the predictors. The lower-order factors must be listed first, and the final name must be the latent interaction factor.
`nameY`	The name of factor that is used as the dependent variable.
`modVar`	The name of factor that is used as a moderator. The effect of the other independent factor will be probed at each value of the moderator variable listed in `valProbe`.
`valProbe`	The values of the moderator that will be used to probe the effect of the focal predictor.
`group`	In multigroup models, the label of the group for which the results will be returned. Must correspond to one of `lavInspect(fit, "group.label")`, or an integer corresponding to which of those group labels.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Ignored unless `fit` is of class lavaan.mi::lavaan.mi. Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However, gross model misspecification could also cause NPD solutions, users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.

Details

Before using this function, researchers need to make the products of the indicators between the first-order factors using mean centering (Marsh, Wen, & Hau, 2004). Note that the double-mean centering may not be appropriate for probing interaction if researchers are interested in simple intercepts. The mean or double-mean centering can be done by the `indProd()` function. The indicator products can be made for all possible combination or matched-pair approach (Marsh et al., 2004). Next, the hypothesized model with the regression with latent interaction will be used to fit all original indicators and the product terms. See the example for how to fit the product term below. Once the lavaan result is obtained, this function will be used to probe the interaction.

Let that the latent interaction model regressing the dependent variable (Y) on the independent variable (X) and the moderator (Z) be

$$Y = b_0 + b_1 X + b_2 Z + b_3 XZ + r,$$

where b_0 is the estimated intercept or the expected value of Y when both X and Z are 0, b_1 is the effect of X when Z is 0, b_2 is the effect of Z when X is 0, b_3 is the interaction effect between X and Z, and r is the residual term.

To probe a two-way interaction, the simple intercept of the independent variable at each value of the moderator (Aiken & West, 1991; Cohen, Cohen, West, & Aiken, 2003; Preacher, Curran, & Bauer, 2006) can be obtained by

$$b_{0|X=0,Z} = b_0 + b_2 Z.$$

The simple slope of the independent varaible at each value of the moderator can be obtained by

$$b_{X|Z} = b_1 + b_3 Z.$$

The variance of the simple intercept formula is

$$Var\left(b_{0|X=0,Z}\right) = Var\left(b_0\right) + 2Z \times Cov\left(b_0, b_2\right) + Z^2 \times Var\left(b_2\right)$$

, where Var denotes the variance of a parameter estimate and Cov denotes the covariance of two parameter estimates.

The variance of the simple slope formula is

$$Var\left(b_{X|Z}\right) = Var\left(b_1\right) + 2Z \times Cov\left(b_1, b_3\right) + Z^2 \times Var\left(b_3\right)$$

Wald z statistic is used for test statistic (even for objects of class lavaan.mi::lavaan.mi).

Value

A list with two elements:

1. `SimpleIntercept`: The simple intercepts given each value of the moderator.
2. `SimpleSlope`: The simple slopes given each value of the moderator.

In each element, the first column represents the values of the moderator specified in the `valProbe` argument. The second column is the simple intercept or simple slope. The third column is the *SE* of the simple intercept or simple slope. The fourth column is the Wald (z) statistic, and the fifth column is the associated *p* value testing the null hypothesis that each simple intercept or slope is 0.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Tutorial:

Schoemann, A. M., & Jorgensen, T. D. (2021). Testing and interpreting latent variable interactions using the `semTools` package. *Psych, 3*(3), 322–335. doi:10.3390/psych3030024

Background literature:

Aiken, L. S., & West, S. G. (1991). *Multiple regression: Testing and interpreting interactions*. Newbury Park, CA: Sage.

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2003). *Applied multiple regression/correlation analysis for the behavioral sciences* (3rd ed.). New York, NY: Routledge.

Marsh, H. W., Wen, Z., & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*(3), 275–300. doi:10.1037/1082989X.9.3.275

Preacher, K. J., Curran, P. J., & Bauer, D. J. (2006). Computational tools for probing interactions in multiple linear regression, multilevel modeling, and latent curve analysis. *Journal of Educational and Behavioral Statistics, 31*(4), 437–448. doi:10.3102/10769986031004437

See Also

- `indProd()` For creating the indicator products with no centering, mean centering, double-mean centering, or residual centering.
- `probe3WayMC()` For probing the three-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe2WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `probe3WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `plotProbe()` Plot the simple intercepts and slopes of the latent interaction.

Examples

```
dat2wayMC <- indProd(dat2way, 1:3, 4:6) # double mean centered by default

model1 <- "
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f12 =~ x1.x4 + x2.x5 + x3.x6
f3 =~ x7 + x8 + x9
f3 ~ f1 + f2 + f12
f12 ~~ 0*f1 + 0*f2 # not necessary, but implied by double mean centering
"

fitMC2way <- sem(model1, data = dat2wayMC, meanstructure = TRUE)
summary(fitMC2way)

probe2WayMC(fitMC2way, nameX = c("f1", "f2", "f12"), nameY = "f3",
            modVar = "f2", valProbe = c(-1, 0, 1))


## can probe multigroup models, one group at a time
dat2wayMC$g <- 1:2

model2 <- "
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f12 =~ x1.x4 + x2.x5 + x3.x6
f3 =~ x7 + x8 + x9
f3 ~ c(b1.g1, b1.g2)*f1 + c(b2.g1, b2.g2)*f2 + c(b12.g1, b12.g2)*f12
f12 ~~ 0*f1 + 0*f2
"
```

```
fit2 <- sem(model2, data = dat2wayMC, group = "g")
probe2WayMC(fit2, nameX = c("f1", "f2", "f12"), nameY = "f3",
            modVar = "f2", valProbe = c(-1, 0, 1)) # group = 1 by default
probe2WayMC(fit2, nameX = c("f1", "f2", "f12"), nameY = "f3",
            modVar = "f2", valProbe = c(-1, 0, 1), group = 2)
```

`probe2WayRC` *Probing two-way interaction on the residual-centered latent interaction*

Description

Probing interaction for simple intercept and simple slope for the residual-centered latent two-way interaction (Geldhof et al., 2013)

Usage

```
probe2WayRC(fit, nameX, nameY, modVar, valProbe, group = 1L,
  omit.imps = c("no.conv", "no.se"))
```

Arguments

`fit`	A fitted lavaan::lavaan or lavaan.mi::lavaan.mi object with a latent 2-way interaction.
`nameX`	`character` vector of all 3 factor names used as the predictors. The lower-order factors must be listed first, and the final name must be the latent interaction factor.
`nameY`	The name of factor that is used as the dependent variable.
`modVar`	The name of factor that is used as a moderator. The effect of the other independent factor will be probed at each value of the moderator variable listed in `valProbe`.
`valProbe`	The values of the moderator that will be used to probe the effect of the focal predictor.
`group`	In multigroup models, the label of the group for which the results will be returned. Must correspond to one of `lavInspect(fit, "group.label")`, or an integer corresponding to which of those group labels.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Ignored unless `fit` is of class lavaan.mi::lavaan.mi. Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However,

gross model misspecification could also cause NPD solutions, users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.

Details

Before using this function, researchers need to make the products of the indicators between the first-order factors and residualize the products by the original indicators (Lance, 1988; Little, Bovaird, & Widaman, 2006). The process can be automated by the `indProd()` function. Note that the indicator products can be made for all possible combination or matched-pair approach (Marsh et al., 2004). Next, the hypothesized model with the regression with latent interaction will be used to fit all original indicators and the product terms. To use this function the model must be fit with a mean structure. See the example for how to fit the product term below. Once the lavaan result is obtained, this function will be used to probe the interaction.

The probing process on residual-centered latent interaction is based on transforming the residual-centered result into the no-centered result. See Geldhof et al. (2013) for further details. Note that this approach is based on a strong assumption that the first-order latent variables are normally distributed. The probing process is applied after the no-centered result (parameter estimates and their covariance matrix among parameter estimates) has been computed. See the `probe2WayMC()` for further details.

Value

A list with two elements:

1. `SimpleIntercept`: The simple intercepts given each value of the moderator.
2. `SimpleSlope`: The simple slopes given each value of the moderator.

In each element, the first column represents the values of the moderators specified in the `valProbe` argument. The second column is the simple intercept or simple slope. The third column is the standard error of the simple intercept or slope. The fourth column is the Wald (z) statistic, and the fifth column is the associated p value testing the null hypothesis that each simple intercept or slope is 0.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Tutorial:

Schoemann, A. M., & Jorgensen, T. D. (2021). Testing and interpreting latent variable interactions using the `semTools` package. *Psych, 3*(3), 322–335. doi:10.3390/psych3030024

Background literature:

Lance, C. E. (1988). Residual centering, exploratory and confirmatory moderator analysis, and decomposition of effects in path models containing interactions. *Applied Psychological Measurement, 12*(2), 163–175. doi:10.1177/014662168801200205

Little, T. D., Bovaird, J. A., & Widaman, K. F. (2006). On the merits of orthogonalizing powered and product terms: Implications for modeling interactions. *Structural Equation Modeling, 13*(4), 497–519. doi:10.1207/s15328007sem1304_1

Marsh, H. W., Wen, Z., & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*(3), 275–300. doi:10.1037/1082989X.9.3.275

Geldhof, G. J., Pornprasertmanit, S., Schoemann, A. M., & Little, T. D. (2013). Orthogonalizing through residual centering: Extended applications and caveats. *Educational and Psychological Measurement, 73*(1), 27–46. doi:10.1177/0013164412445473

See Also

- `indProd()` For creating the indicator products with no centering, mean centering, double-mean centering, or residual centering.
- `probe2WayMC()` For probing the two-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe3WayMC()` For probing the three-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe3WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `plotProbe()` Plot the simple intercepts and slopes of the latent interaction.

Examples

```
dat2wayRC <- orthogonalize(dat2way, 1:3, 4:6)

model1 <- "
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f12 =~ x1.x4 + x2.x5 + x3.x6
f3 =~ x7 + x8 + x9
f3 ~ f1 + f2 + f12
f12 ~~ 0*f1 + 0*f2
x1 + x4 + x1.x4 + x7 ~ 0*1 # identify latent means
f1 + f2 + f12 + f3 ~ NA*1
"

fitRC2way <- sem(model1, data = dat2wayRC, meanstructure = TRUE)
summary(fitRC2way)

probe2WayRC(fitRC2way, nameX = c("f1", "f2", "f12"), nameY = "f3",
            modVar = "f2", valProbe = c(-1, 0, 1))


## can probe multigroup models, one group at a time
dat2wayRC$g <- 1:2

model2 <- "
f1  =~ x1 + x2 + x3
```

```
f2  =~ x4 + x5 + x6
f12 =~ x1.x4 + x2.x5 + x3.x6
f3  =~ x7 + x8 + x9
f3 ~ c(b1.g1, b1.g2)*f1 + c(b2.g1, b2.g2)*f2 + c(b12.g1, b12.g2)*f12
f12 ~~ 0*f1 + 0*f2
x1 + x4 + x1.x4 + x7 ~ 0*1 # identify latent means
f1 + f2 + f12 ~ NA*1
f3 ~ NA*1 + c(b0.g1, b0.g2)*1
"
fit2 <- sem(model2, data = dat2wayRC, group = "g")
probe2WayRC(fit2, nameX = c("f1", "f2", "f12"), nameY = "f3",
            modVar = "f2", valProbe = c(-1, 0, 1)) # group = 1 by default
probe2WayRC(fit2, nameX = c("f1", "f2", "f12"), nameY = "f3",
            modVar = "f2", valProbe = c(-1, 0, 1), group = 2)
```

`probe3WayMC` *Probing three-way interaction on the no-centered or mean-centered latent interaction*

Description

Probing interaction for simple intercept and simple slope for the no-centered or mean-centered latent two-way interaction

Usage

```
probe3WayMC(fit, nameX, nameY, modVar, valProbe1, valProbe2, group = 1L,
  omit.imps = c("no.conv", "no.se"))
```

Arguments

`fit`	A fitted lavaan::lavaan or lavaan.mi::lavaan.mi object with a latent 2-way interaction.
`nameX`	`character` vector of all 7 factor names used as the predictors. The 3 lower-order factors must be listed first, followed by the 3 second-order factors (specifically, the 4th element must be the interaction between the factors listed first and second, the 5th element must be the interaction between the factors listed first and third, and the 6th element must be the interaction between the factors listed second and third). The final name will be the factor representing the 3-way interaction.
`nameY`	The name of factor that is used as the dependent variable.
`modVar`	The name of two factors that are used as the moderators. The effect of the independent factor will be probed at each combination of the moderator variables' chosen values.
`valProbe1`	The values of the first moderator that will be used to probe the effect of the independent factor.

`valProbe2`	The values of the second moderator that will be used to probe the effect of the independent factor.
`group`	In multigroup models, the label of the group for which the results will be returned. Must correspond to one of `lavInspect(fit, "group.label")`.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Ignored unless `fit` is of class lavaan.mi::lavaan.mi. Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However, gross model misspecification could also cause NPD solutions, users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.

Details

Before using this function, researchers need to make the products of the indicators between the first-order factors using mean centering (Marsh, Wen, & Hau, 2004). Note that the double-mean centering may not be appropriate for probing interaction if researchers are interested in simple intercepts. The mean or double-mean centering can be done by the `indProd()` function. The indicator products can be made for all possible combination or matched-pair approach (Marsh et al., 2004). Next, the hypothesized model with the regression with latent interaction will be used to fit all original indicators and the product terms. See the example for how to fit the product term below. Once the lavaan result is obtained, this function will be used to probe the interaction.

Let that the latent interaction model regressing the dependent variable (Y) on the independent variable (X) and two moderators (Z and W) be

$$Y = b_0 + b_1X + b_2Z + b_3W + b_4XZ + b_5XW + b_6ZW + b_7XZW + r,$$

where b_0 is the estimated intercept or the expected value of Y when X, Z, and W are 0, b_1 is the effect of X when Z and W are 0, b_2 is the effect of Z when X and W is 0, b_3 is the effect of W when X and Z are 0, b_4 is the interaction effect between X and Z when W is 0, b_5 is the interaction effect between X and W when Z is 0, b_6 is the interaction effect between Z and W when X is 0, b_7 is the three-way interaction effect between X, Z, and W, and r is the residual term.

To probe a three-way interaction, the simple intercept of the independent variable at the specific values of the moderators (Aiken & West, 1991) can be obtained by

$$b_{0|X=0,Z,W} = b_0 + b_2Z + b_3W + b_6ZW.$$

The simple slope of the independent variable at the specific values of the moderators can be obtained by

$$b_{X|Z,W} = b_1 + b_3Z + b_4W + b_7ZW.$$

The variance of the simple intercept formula is

$$Var\left(b_{0|X=0,Z,W}\right) = Var\left(b_0\right) + Z^2Var\left(b_2\right) + W^2Var\left(b_3\right) + Z^2W^2Var\left(b_6\right)$$

$$+2ZCov\left(b_0, b_2\right)+2WCov\left(b_0, b_3\right)+2ZWCov\left(b_0, b_6\right)+2ZWCov\left(b_2, b_3\right)+2Z^2WCov\left(b_2, b_6\right)+2ZW^2Cov\left(b_3, b_6\right),$$

where Var denotes the variance of a parameter estimate and Cov denotes the covariance of two parameter estimates.

The variance of the simple slope formula is

$$Var\left(b_{X|Z,W}\right) = Var\left(b_1\right) + Z^2Var\left(b_4\right) + W^2Var\left(b_5\right) + Z^2W^2Var\left(b_7\right)$$

$$+2ZCov\left(b_1, b_4\right)+2WCov\left(b_1, b_5\right)+2ZWCov\left(b_1, b_7\right)+2ZWCov\left(b_4, b_5\right)+2Z^2WCov\left(b_4, b_7\right)+2ZW^2Cov\left(b_5, b_7\right).$$

Wald z statistics are calculated (even for objects of class lavaan.mi::lavaan.mi) to test null hypotheses that simple intercepts or slopes are 0.

Value

A list with two elements:

1. `SimpleIntercept`: The model-implied intercepts given each combination of moderator values.
2. `SimpleSlope`: The model-implied slopes given each combination of moderator values.

In each element, the first column represents values of the first moderator specified in the `valProbe1` argument. The second column represents values of the second moderator specified in the `valProbe2` argument. The third column is the simple intercept or simple slope. The fourth column is the standard error of the simple intercept or simple slope. The fifth column is the Wald (z) statistic, and the sixth column is its associated p value to test the null hypothesis that each simple intercept or simple slope equals 0.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Tutorial:

Schoemann, A. M., & Jorgensen, T. D. (2021). Testing and interpreting latent variable interactions using the `semTools` package. *Psych, 3*(3), 322–335. doi:10.3390/psych3030024

Background literature:

Aiken, L. S., & West, S. G. (1991). *Multiple regression: Testing and interpreting interactions*. Newbury Park, CA: Sage.

Marsh, H. W., Wen, Z., & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*(3), 275–300. doi:10.1037/1082989X.9.3.275

See Also

- `indProd()` For creating the indicator products with no centering, mean centering, double-mean centering, or residual centering.
- `probe2WayMC()` For probing the two-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe2WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `probe3WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `plotProbe()` Plot the simple intercepts and slopes of the latent interaction.

Examples

```
dat3wayMC <- indProd(dat3way, 1:3, 4:6, 7:9)

model3 <- " ## define latent variables
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f3 =~ x7 + x8 + x9
## 2-way interactions
f12 =~ x1.x4 + x2.x5 + x3.x6
f13 =~ x1.x7 + x2.x8 + x3.x9
f23 =~ x4.x7 + x5.x8 + x6.x9
## 3-way interaction
f123 =~ x1.x4.x7 + x2.x5.x8 + x3.x6.x9
## outcome variable
f4 =~ x10 + x11 + x12

## latent regression model
f4 ~ b1*f1 + b2*f2 + b3*f3 + b12*f12 + b13*f13 + b23*f23 + b123*f123

## orthogonal terms among predictors
## (not necessary, but implied by double mean centering)
f1 ~~ 0*f12 + 0*f13 + 0*f123
f2 ~~ 0*f12 + 0*f23 + 0*f123
f3 ~~ 0*f13 + 0*f23 + 0*f123
f12 + f13 + f23 ~~ 0*f123
"

fitMC3way <- sem(model3, data = dat3wayMC, meanstructure = TRUE)
summary(fitMC3way)

probe3WayMC(fitMC3way, nameX = c("f1" ,"f2" ,"f3",
                                 "f12","f13","f23", # this order matters!
                                 "f123"),           # 3-way interaction
            nameY = "f4", modVar = c("f1", "f2"),
            valProbe1 = c(-1, 0, 1), valProbe2 = c(-1, 0, 1))
```

`probe3WayRC` *Probing three-way interaction on the residual-centered latent interaction*

Description

Probing interaction for simple intercept and simple slope for the residual-centered latent three-way interaction (Geldhof et al., 2013)

Usage

```
probe3WayRC(fit, nameX, nameY, modVar, valProbe1, valProbe2, group = 1L,
  omit.imps = c("no.conv", "no.se"))
```

Arguments

`fit`	A fitted lavaan::lavaan or lavaan.mi::lavaan.mi object with a latent 2-way interaction.
`nameX`	`character` vector of all 7 factor names used as the predictors. The 3 lower-order factors must be listed first, followed by the 3 second-order factors (specifically, the 4th element must be the interaction between the factors listed first and second, the 5th element must be the interaction between the factors listed first and third, and the 6th element must be the interaction between the factors listed second and third). The final name will be the factor representing the 3-way interaction.
`nameY`	The name of factor that is used as the dependent variable.
`modVar`	The name of two factors that are used as the moderators. The effect of the independent factor on each combination of the moderator variable values will be probed.
`valProbe1`	The values of the first moderator that will be used to probe the effect of the independent factor.
`valProbe2`	The values of the second moderator that will be used to probe the effect of the independent factor.
`group`	In multigroup models, the label of the group for which the results will be returned. Must correspond to one of `lavInspect(fit, "group.label")`.
`omit.imps`	`character` vector specifying criteria for omitting imputations from pooled results. Ignored unless `fit` is of class lavaan.mi::lavaan.mi. Can include any of `c("no.conv", "no.se", "no.npd")`, the first 2 of which are the default setting, which excludes any imputations that did not converge or for which standard errors could not be computed. The last option (`"no.npd"`) would exclude any imputations which yielded a nonpositive definite covariance matrix for observed or latent variables, which would include any "improper solutions" such as Heywood cases. NPD solutions are not excluded by default because they are likely to occur due to sampling error, especially in small samples. However, gross model misspecification could also cause NPD solutions, users can compare pooled results with and without this setting as a sensitivity analysis to see whether some imputations warrant further investigation.

Details

Before using this function, researchers need to make the products of the indicators between the first-order factors and residualize the products by the original indicators (Lance, 1988; Little, Bovaird, & Widaman, 2006). The process can be automated by the `indProd()` function. Note that the indicator products can be made for all possible combination or matched-pair approach (Marsh et al., 2004). Next, the hypothesized model with the regression with latent interaction will be used to fit all original indicators and the product terms (Geldhof et al., 2013). To use this function the model must be fit with a mean structure. See the example for how to fit the product term below. Once the lavaan result is obtained, this function will be used to probe the interaction.

The probing process on residual-centered latent interaction is based on transforming the residual-centered result into the no-centered result. See Geldhof et al. (2013) for further details. Note that this approach based on a strong assumption that the first-order latent variables are normally distributed. The probing process is applied after the no-centered result (parameter estimates and their covariance matrix among parameter estimates) has been computed. See the `probe3WayMC()` for further details.

Value

A list with two elements:

1. `SimpleIntercept`: The model-implied intercepts given each combination of moderator values.
2. `SimpleSlope`: The model-implied slopes given each combination of moderator values.

In each element, the first column represents values of the first moderator specified in the `valProbe1` argument. The second column represents values of the second moderator specified in the `valProbe2` argument. The third column is the simple intercept or simple slope. The fourth column is the *SE* of the simple intercept or simple slope. The fifth column is the Wald (*z*) statistic, and the sixth column is its associated *p* value to test the null hypothesis that each simple intercept or simple slope equals 0.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Tutorial:

Schoemann, A. M., & Jorgensen, T. D. (2021). Testing and interpreting latent variable interactions using the `semTools` package. *Psych, 3*(3), 322–335. doi:10.3390/psych3030024

Background literature:

Geldhof, G. J., Pornprasertmanit, S., Schoemann, A., & Little, T. D. (2013). Orthogonalizing through residual centering: Extended applications and caveats. *Educational and Psychological Measurement, 73*(1), 27–46. doi:10.1177/0013164412445473

Lance, C. E. (1988). Residual centering, exploratory and confirmatory moderator analysis, and decomposition of effects in path models containing interactions. *Applied Psychological Measurement, 12*(2), 163–175. doi:10.1177/014662168801200205

Little, T. D., Bovaird, J. A., & Widaman, K. F. (2006). On the merits of orthogonalizing powered and product terms: Implications for modeling interactions. *Structural Equation Modeling, 13*(4), 497–519. doi:10.1207/s15328007sem1304_1

Marsh, H. W., Wen, Z., & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*(3), 275–300. doi:10.1037/1082989X.9.3.275

Pornprasertmanit, S., Schoemann, A. M., Geldhof, G. J., & Little, T. D. (submitted). *Probing latent interaction estimated with a residual centering approach.*

See Also

- `indProd()` For creating the indicator products with no centering, mean centering, double-mean centering, or residual centering.
- `probe2WayMC()` For probing the two-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe3WayMC()` For probing the three-way latent interaction when the results are obtained from mean-centering, or double-mean centering
- `probe2WayRC()` For probing the two-way latent interaction when the results are obtained from residual-centering approach.
- `plotProbe()` Plot the simple intercepts and slopes of the latent interaction.

Examples

```
dat3wayRC <- orthogonalize(dat3way, 1:3, 4:6, 7:9)

model3 <- " ## define latent variables
f1 =~ x1 + x2 + x3
f2 =~ x4 + x5 + x6
f3 =~ x7 + x8 + x9
## 2-way interactions
f12 =~ x1.x4 + x2.x5 + x3.x6
f13 =~ x1.x7 + x2.x8 + x3.x9
f23 =~ x4.x7 + x5.x8 + x6.x9
## 3-way interaction
f123 =~ x1.x4.x7 + x2.x5.x8 + x3.x6.x9
## outcome variable
f4 =~ x10 + x11 + x12

## latent regression model
f4 ~ b1*f1 + b2*f2 + b3*f3 + b12*f12 + b13*f13 + b23*f23 + b123*f123

## orthogonal terms among predictors
f1 ~~ 0*f12 + 0*f13 + 0*f123
f2 ~~ 0*f12 + 0*f23 + 0*f123
f3 ~~ 0*f13 + 0*f23 + 0*f123
f12 + f13 + f23 ~~ 0*f123

## identify latent means
x1 + x4 + x7 + x1.x4 + x1.x7 + x4.x7 + x1.x4.x7 + x10 ~ 0*1
```

```
f1 + f2 + f3 + f12 + f13 + f23 + f123 + f4 ~ NA*1
"

fitRC3way <- sem(model3, data = dat3wayRC, meanstructure = TRUE)
summary(fitRC3way)

probe3WayMC(fitRC3way, nameX = c("f1" ,"f2" ,"f3",
                                 "f12","f13","f23", # this order matters!
                                 "f123"),           # 3-way interaction
            nameY = "f4", modVar = c("f1", "f2"),
            valProbe1 = c(-1, 0, 1), valProbe2 = c(-1, 0, 1))
```

`quark` *Quark*

Description

The `quark` function provides researchers with the ability to calculate and include component scores calculated by taking into account the variance in the original dataset and all of the interaction and polynomial effects of the data in the dataset.

Usage

```
quark(data, id, order = 1, silent = FALSE, ...)
```

Arguments

`data`	The data frame is a required component for `quark`. In order for `quark` to process a data frame, it must not contain any factors or text-based variables. All variables must be in numeric format. Identifiers and dates can be left in the data; however, they will need to be identified under the `id` argument.
`id`	Identifiers and dates within the dataset will need to be acknowledged as `quark` cannot process these. By acknowledging the identifiers and dates as a vector of column numbers or variable names, `quark` will remove them from the data temporarily to complete its main processes. Among many potential issues of not acknowledging identifiers and dates are issues involved with imputation, product and polynomial effects, and principal component analysis.
`order`	Order is an optional argument provided by quark that can be used when the imputation procedures in mice fail. Under some circumstances, mice cannot calculate missing values due to issues with extreme missingness. Should an error present itself stating a failure due to not having any columns selected, set the argument `order = 2` in order to reorder the imputation method procedure. Otherwise, use the default `order = 1`.
`silent`	If `FALSE`, the details of the `quark` process are printed.
`...`	additional arguments to pass to `mice::mice()`.

Details

The `quark` function calculates these component scores by first filling in the data via means of multiple imputation methods and then expanding the dataset by aggregating the non-overlapping interaction effects between variables by calculating the mean of the interactions and polynomial effects. The multiple imputation methods include one of iterative sampling and group mean substitution and multiple imputation using a polytomous regression algorithm (mice). During the expansion process, the dataset is expanded to three times its normal size (in width). The first third of the dataset contains all of the original data post imputation, the second third contains the means of the polynomial effects (squares and cubes), and the final third contains the means of the non-overlapping interaction effects. A full principal componenent analysis is conducted and the individual components are retained. The subsequent `combinequark()` function provides researchers the control in determining how many components to extract and retain. The function returns the dataset as submitted (with missing values) and the component scores as requested for a more accurate multiple imputation in subsequent steps.

Value

The output value from using the quark function is a list. It will return a list with 7 components.

`ID Columns`	Is a vector of the identifier columns entered when running quark.
`ID Variables`	Is a subset of the dataset that contains the identifiers as acknowledged when running quark.
`Used Data`	Is a matrix / dataframe of the data provided by user as the basis for quark to process.
`Imputed Data`	Is a matrix / dataframe of the data after the multiple method imputation process.
`Big Matrix`	Is the expanded product and polynomial matrix.
`Principal Components`	Is the entire dataframe of principal components for the dataset. This dataset will have the same number of rows of the big matrix, but will have 1 less column (as is the case with principal component analyses).
`Percent Variance Explained`	Is a vector of the percent variance explained with each column of principal components.

Author(s)

Steven R. Chesnut (University of Southern Mississippi; <`Steven.Chesnut@usm.edu`>)

Danny Squire (Texas Tech University)

Terrence D. Jorgensen (University of Amsterdam)

The PCA code is copied and modified from the `FactoMineR` package.

References

Howard, W. J., Rhemtulla, M., & Little, T. D. (2015). Using Principal Components as Auxiliary Variables in Missing Data Estimation. *Multivariate Behavioral Research, 50*(3), 285–299. doi:10.1080/00273171.2014.999267

See Also

`combinequark()`

Examples

```
set.seed(123321)

dat <- HolzingerSwineford1939[,7:15]
misspat <- matrix(runif(nrow(dat) * 9) < 0.3, nrow(dat))
dat[misspat] <- NA
dat <- cbind(HolzingerSwineford1939[,1:3], dat)

quark.list <- quark(data = dat, id = c(1, 2))

final.data <- combinequark(quark = quark.list, percent = 80)

## Example to rerun quark after imputation failure:
quark.list <- quark(data = dat, id = c(1, 2), order = 2)
```

`residualCovariate` *Residual-center all target indicators by covariates*

Description

This function will regress target variables on the covariate and replace the target variables by the residual of the regression analysis. This procedure is useful to control the covariate from the analysis model (Geldhof, Pornprasertmanit, Schoemann, & Little, 2013).

Usage

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

Arguments

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

Value

The data that the target variables replaced by the residuals

Author(s)

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

References

Geldhof, G. J., Pornprasertmanit, S., Schoemann, A. M., & Little, T. D. (2013). Orthogonalizing through residual centering: Extended applications and caveats. *Educational and Psychological Measurement, 73*(1), 27–46. doi:10.1177/0013164412445473

See Also

`indProd()` For creating the indicator products with no centering, mean centering, double-mean centering, or residual centering.

Examples

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

`semTools` *semTools: Useful Tools for Structural Equation Modeling*

Description

The **semTools** package provides many miscellaneous functions that are useful for statistical analysis involving SEM in R. Many functions extend the funtionality of the **lavaan** package. Some sets of functions in **semTools** correspond to the same theme. We call such a collection of functions a *suite*. Our suites include:

- Model Fit Evaluation: `moreFitIndices()`, `nullRMSEA()`, `singleParamTest()`, `epcEquivFit()`, and `chisqSmallN()`
- Measurement Invariance: `measEq.syntax()`, `partialInvariance()`, `partialInvarianceCat()`, and `permuteMeasEq()`
- Power Analysis: `SSpower()`, `findRMSEApower()`, `plotRMSEApower()`, `plotRMSEAdist()`, `findRMSEAsamplesize()`, `findRMSEApowernested()`, `plotRMSEApowernested()`, and `findRMSEAsamplesizeneste`
- Missing Data Analysis: `auxiliary()`, `twostage()`, `fmi()`, `bsBootMiss()`, `quark()`, and `combinequark()`
- Latent Interactions: `indProd()`, `orthogonalize()`, `probe2WayMC()`, `probe3WayMC()`, `probe2WayRC()`, `probe3WayRC()`, and `plotProbe()`
- Exploratory Factor Analysis (EFA): `efa.ekc()`
- Reliability Estimation: `compRelSEM()` and `maximalRelia()` (see also `AVE()`)
- Parceling: `parcelAllocation()`, `PAVranking()`, and `poolMAlloc()`
- Non-Normality: `skew()`, `kurtosis()`, `mardiaSkew()`, `mardiaKurtosis()`, and `mvrnonnorm()`

All users of R (or SEM) are invited to submit functions or ideas for functions by contacting the maintainer, Terrence Jorgensen (`<TJorgensen314@gmail.com>`). Contributors are encouraged to use `Roxygen` comments to document their contributed code, which is consistent with the rest of **semTools**. Read the vignette from the **roxygen2** package for details: `vignette("rd", package = "roxygen2")`

`simParcel` *Simulated Data set to Demonstrate Random Allocations of Parcels*

Description

A simulated data set with 2 factors with 9 indicators for each factor

Usage

```
simParcel
```

Format

A `data.frame` with 800 observations of 18 variables.

f1item1	Item 1 loading on factor 1
f1item2	Item 2 loading on factor 1
f1item3	Item 3 loading on factor 1
f1item4	Item 4 loading on factor 1
f1item5	Item 5 loading on factor 1
f1item6	Item 6 loading on factor 1
f1item7	Item 7 loading on factor 1
f1item8	Item 8 loading on factor 1
f1item9	Item 9 loading on factor 1
f2item1	Item 1 loading on factor 2
f2item2	Item 2 loading on factor 2
f2item3	Item 3 loading on factor 2
f2item4	Item 4 loading on factor 2
f2item5	Item 5 loading on factor 2
f2item6	Item 6 loading on factor 2
f2item7	Item 7 loading on factor 2
f2item8	Item 8 loading on factor 2
f2item9	Item 9 loading on factor 2

Source

Data were generated using the `simsem` package.

Examples

```
head(simParcel)
```

`singleParamTest` *Single Parameter Test Divided from Nested Model Comparison*

Description

In comparing two nested models, $\Delta\chi^2$ test may indicate that two models are different. However, like other omnibus tests, researchers do not know which fixed parameters or constraints make these two models different. This function will help researchers identify the significant parameter.

Usage

```
singleParamTest(model1, model2, return.fit = FALSE,
  method = "satorra.bentler.2001")
```

Arguments

`model1`	Model 1.
`model2`	Model 2. Note that two models must be nested models. Further, the order of parameters in their parameter tables are the same. That is, nested models with different scale identifications may not be able to test by this function.
`return.fit`	Return the submodels fitted by this function
`method`	The method used to calculate likelihood ratio test. See `lavaan::lavTestLRT()` for available options

Details

This function first identifies the differences between these two models. The model with more free parameters is referred to as parent model and the model with fewer free parameters is referred to as nested model. Two tests are implemented here:

1. `free`: The nested model is used as a template. Then, one parameter indicating the differences between two models is freed. The new model is compared with the nested model. This process is repeated for all differences between two models.
2. `fix`: The parent model is used as a template. Then, one parameter indicating the differences between two models is fixed or constrained to be equal to other parameters. The new model is then compared with the parent model. This process is repeated for all differences between two models.
3. `mi`: No longer available because the test of modification indices is not consistent. For example, if two parameters are equality constrained, the modification index from the first parameter is not equal to the second parameter.

Note that this function does not adjust for the inflated Type I error rate from multiple tests.

Value

If `return.fit = FALSE`, the result tables are provided. χ^2 and *p* value are provided for all methods. Note that the χ^2 is all based on 1 *df*. Expected parameter changes and their standardized forms are also provided.

If `return.fit = TRUE`, a list with two elements are provided. The first element is the tabular result. The second element is the submodels used in the `free` and `fix` methods.

Author(s)

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

Examples

```
library(lavaan)

# Nested model comparison by hand
HS.model1 <- ' visual =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6'
HS.model2 <- ' visual =~ a*x1 + a*x2 + a*x3
              textual =~ b*x4 + b*x5 + b*x6'

m1 <- cfa(HS.model1, data = HolzingerSwineford1939, std.lv = TRUE,
          estimator = "MLR")
m2 <- cfa(HS.model2, data = HolzingerSwineford1939, std.lv = TRUE,
          estimator = "MLR")
anova(m1, m2)
singleParamTest(m1, m2)


## Nested models to test measurement invariance
HW.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

mod.config <- cfa(model = HW.model, data = HolzingerSwineford1939,
                  group = "school")
mod.metric <- cfa(model = HW.model, data = HolzingerSwineford1939,
                  group = "school", group.equal = "loadings")
singleParamTest(mod.config, mod.metric)
```

---

skew *Finding skewness*

Description

Finding skewness (g_1) of an object

Usage

```
skew(object, population = FALSE)
```

Arguments

`object`	A vector used to find a skewness
`population`	`TRUE` to compute the parameter formula. `FALSE` to compute the sample statistic formula.

Details

The skewness computed by default is g_1, the third standardized moment of the empirical distribution of `object`. The population parameter skewness γ_1 formula is

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}},$$

where μ_i denotes the i order central moment.

The skewness formula for sample statistic g_1 is

$$g_1 = \frac{k_3}{k_2^2},$$

where k_i are the i order *k*-statistic.

The standard error of the skewness is

$$Var(\hat{g}_1) = \frac{6}{N}$$

where N is the sample size.

Value

A value of a skewness with a test statistic if the population is specified as `FALSE`

Author(s)

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

References

Weisstein, Eric W. (n.d.). *Skewness*. Retrieved from *MathWorld*–A Wolfram Web Resource: `https://mathworld.wolfram.com/Skewness.html`

See Also

- `kurtosis()` Find the univariate excessive kurtosis of a variable
- `mardiaSkew()` Find Mardia's multivariate skewness of a set of variables
- `mardiaKurtosis()` Find the Mardia's multivariate kurtosis of a set of variables

Examples

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

---

splitSample *Randomly Split a Data Set into Halves*

Description

This function randomly splits a data set into two halves, and saves the resulting data sets to the same folder as the original.

Usage

```
splitSample(dataset, path = "default", div = 2, type = "default",
  name = "splitSample")
```

Arguments

`dataset`	The original data set to be divided. Can be a file path to a *.csv or *.dat file (headers will automatically be detected) or an R object (matrix or dataframe). (Windows users: file path must be specified using FORWARD SLASHES (/) ONLY.)
`path`	File path to folder for output data sets. NOT REQUIRED if dataset is a filename. Specify ONLY if dataset is an R object, or desired output folder is not that of original data set. If path is specified as "object", output data sets will be returned as a list, and not saved to hard drive.
`div`	Number of output data sets. NOT REQUIRED if default, 2 halves.
`type`	Output file format ("dat" or "csv"). NOT REQUIRED unless desired output formatting differs from that of input, or dataset is an R object and csv formatting is desired.
`name`	Output file name. NOT REQUIRED unless desired output name differs from that of input, or input dataset is an R object. (If input is an R object and name is not specified, name will be "splitSample".)

Details

This function randomly orders the rows of a data set, divides the data set into two halves, and saves the halves to the same folder as the original data set, preserving the original formatting. Data set type (*.csv or *.dat) and formatting (headers) are automatically detected, and output data sets will preserve input type and formatting unless specified otherwise. Input can be in the form of a file path (*.dat or *.csv), or an R object (matrix or dataframe). If input is an R object and path is default, output data sets will be returned as a list object.

Value

If `path` = `"object"`, `list` of output data sets. Otherwise, output will saved to hard drive in the same format as input.

Author(s)

Corbin Quick (University of Michigan; <`corbinq@umich.edu`>)

Examples

```
#### Input is .dat file
#splitSample("C:/Users/Default/Desktop/MYDATA.dat")
#### Output saved to "C:/Users/Default/Desktop/" in .dat format
#### Names are "MYDATA_s1.dat" and "MYDATA_s2.dat"

#### Input is R object
## Split C02 dataset from the datasets package
library(datasets)
splitMyData <- splitSample(CO2, path = "object")
summary(splitMyData[[1]])
summary(splitMyData[[2]])
#### Output object splitMyData becomes list of output data sets

#### Input is .dat file in "C:/" folder
#splitSample("C:/testdata.dat", path = "C:/Users/Default/Desktop/", type = "csv")
#### Output saved to "C:/Users/Default/Desktop/" in *.csv format
#### Names are "testdata_s1.csv" and "testdata_s2.csv"

#### Input is R object
#splitSample(myData, path = "C:/Users/Default/Desktop/", name = "splitdata")
#### Output saved to "C:/Users/Default/Desktop/" in *.dat format
#### Names are "splitdata_s1.dat" and "splitdata_s2.dat"
```

`SSpower` *Power for model parameters*

Description

Apply Satorra & Saris (1985) method for chi-squared power analysis.

Usage

```
SSpower(powerModel, n, nparam, popModel, mu, Sigma, fun = "sem",
  alpha = 0.05, ...)
```

Arguments

`powerModel`	lavaan `lavaan::model.syntax()` for the model to be analyzed. This syntax should constrain at least one nonzero parameter to 0 (or another number).
`n`	`integer`. Sample size used in power calculation, or a vector of sample sizes if analyzing a multigroup model. If `length(n)` < `length(Sigma)` when `Sigma` is a list, `n` will be recycled. If `popModel` is used instead of `Sigma`, `n` must specify a sample size for each group, because that is used to infer the number of groups.
`nparam`	`integer`. Number of invalid constraints in `powerModel`.
`popModel`	lavaan `lavaan::model.syntax()` specifying the data-generating model. This syntax should specify values for all nonzero parameters in the model. If `length(n)` > `1`, the same population values will be used for each group, unless different population values are specified per group, either in the lavaan `lavaan::model.syntax()` or by utilizing a list of `Sigma` (and optionally `mu`).
`mu`	`numeric` or `list`. For a single-group model, a vector of population means. For a multigroup model, a list of vectors (one per group). If `mu` and `popModel` are missing, mean structure will be excluded from the analysis.
`Sigma`	`matrix` or `list`. For a single-group model, a population covariance matrix. For a multigroup model, a list of matrices (one per group). If missing, `popModel` will be used to generate a model-implied Sigma.
`fun`	character. Name of `lavaan` function used to fit `powerModel` (i.e., `"cfa"`, `"sem"`, `"growth"`, or `"lavaan"`).
`alpha`	Type I error rate used to set a criterion for rejecting H0.
`...`	additional arguments to pass to `lavaan::lavaan()`. See also `lavaan::lavOptions()`.

Details

Specify all non-zero parameters in a population model, either by using lavaan syntax (`popModel`) or by submitting a population covariance matrix (`Sigma`) and optional mean vector (`mu`) implied by the population model. Then specify an analysis model that places at least one invalid constraint (note the number in the `nparam` argument).

There is also a Shiny app called "power4SEM" that provides a graphical user interface for this functionality (Jak et al., in press). It can be accessed at `https://sjak.shinyapps.io/power4SEM/`.

Author(s)

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

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

References

Satorra, A., & Saris, W. E. (1985). Power of the likelihood ratio test in covariance structure analysis. *Psychometrika, 50*(1), 83–90. doi:10.1007/BF02294150

Jak, S., Jorgensen, T. D., Verdam, M. G., Oort, F. J., & Elffers, L. (2021). Analytical power calculations for structural equation modeling: A tutorial and Shiny app. *Behavior Research Methods, 53*, 1385–1406. doi:10.3758/s13428020014790

Examples

```
## Specify population values. Note every parameter has a fixed value.
modelP <- '
  f1 =~ .7*V1 + .7*V2 + .7*V3 + .7*V4
  f2 =~ .7*V5 + .7*V6 + .7*V7 + .7*V8
  f1 ~~ .3*f2
  f1 ~~ 1*f1
  f2 ~~ 1*f2
  V1 ~~ .51*V1
  V2 ~~ .51*V2
  V3 ~~ .51*V3
  V4 ~~ .51*V4
  V5 ~~ .51*V5
  V6 ~~ .51*V6
  V7 ~~ .51*V7
  V8 ~~ .51*V8
'
## Specify analysis model. Note parameter of interest f1~~f2 is fixed to 0.
modelA <- '
  f1 =~ V1 + V2 + V3 + V4
  f2 =~ V5 + V6 + V7 + V8
  f1 ~~ 0*f2
'
## Calculate power
SSpower(powerModel = modelA, popModel = modelP, n = 150, nparam = 1,
        std.lv = TRUE)

## Get power for a range of sample sizes
Ns <- seq(100, 500, 40)
Power <- rep(NA, length(Ns))
for(i in 1:length(Ns)) {
  Power[i] <- SSpower(powerModel = modelA, popModel = modelP,
                      n = Ns[i], nparam = 1, std.lv = TRUE)
}
plot(x = Ns, y = Power, type = "l", xlab = "Sample Size")


## Optionally specify different values for multiple populations

modelP2 <- '
  f1 =~ .7*V1 + .7*V2 + .7*V3 + .7*V4
  f2 =~ .7*V5 + .7*V6 + .7*V7 + .7*V8
  f1 ~~ c(-.3, .3)*f2                  # DIFFERENT ACROSS GROUPS
  f1 ~~ 1*f1
  f2 ~~ 1*f2
  V1 ~~ .51*V1
  V2 ~~ .51*V2
  V3 ~~ .51*V3
  V4 ~~ .51*V4
  V5 ~~ .51*V5
  V6 ~~ .51*V6
  V7 ~~ .51*V7
```

```
  V8 ~~ .51*V8
'
modelA2 <- '
  f1 =~ V1 + V2 + V3 + V4
  f2 =~ V5 + V6 + V7 + V8
  f1 ~~ c(psi21, psi21)*f2        # EQUALITY CONSTRAINT ACROSS GROUPS
'
## Calculate power
SSpower(powerModel = modelA2, popModel = modelP2, n = c(100, 100), nparam = 1,
        std.lv = TRUE)
## Get power for a range of sample sizes
Ns2 <- cbind(Group1 = seq(10, 100, 10), Group2 = seq(10, 100, 10))
Power2 <- apply(Ns2, MARGIN = 1, FUN = function(nn) {
  SSpower(powerModel = modelA2, popModel = modelP2, n = nn,
          nparam = 1, std.lv = TRUE)
})
plot(x = rowSums(Ns2), y = Power2, type = "l", xlab = "Total Sample Size",
     ylim = 0:1)
abline(h = c(.8, .9), lty = c("dotted","dashed"))
legend("bottomright", c("80% Power","90% Power"), lty = c("dotted","dashed"))
```

---

tukeySEM — *Tukey's WSD post-hoc test of means for unequal variance and sample size*

Description

This function computes Tukey's WSD post hoc test of means when variances and sample sizes are not equal across groups. It can be used as a post hoc test when comparing latent means in multiple group SEM.

Usage

```
tukeySEM(m1, m2, var1, var2, n1, n2, ng)
```

Arguments

`m1`	Mean of group 1.
`m2`	Mean of group 2.
`var1`	Variance of group 1.
`var2`	Variance of group 2.
`n1`	Sample size of group 1.
`n2`	Sample size of group 2.
`ng`	Total number of groups to be compared (i.e., the number of groups compared in the omnibus test).

Details

After conducting an omnibus test of means across three of more groups, researchers often wish to know which sets of means differ at a particular Type I error rate. Tukey's WSD test holds the error rate stable across multiple comparisons of means. This function implements an adaptation of Tukey's WSD test from Maxwell & Delaney (2004), that allows variances and sample sizes to differ across groups.

Value

A vector with three elements:

1. `q`: The *q* statistic
2. `df`: The degrees of freedom for the *q* statistic
3. `p`: A *p* value based on the *q* statistic, *df*, and the total number of groups to be compared

Author(s)

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

References

Maxwell, S. E., & Delaney, H. D. (2004). *Designing experiments and analyzing data: A model comparison perspective* (2nd ed.). Mahwah, NJ: Lawrence Erlbaum Associates.

Examples

```
## For a case where three groups have been compared:
## Group 1: mean = 3.91, var = 0.46, n = 246
## Group 2: mean = 3.96, var = 0.62, n = 465
## Group 3: mean = 2.94, var = 1.07, n = 64

## compare group 1 and group 2
tukeySEM(3.91, 3.96, 0.46, 0.62, 246, 425, 3)

## compare group 1 and group 3
tukeySEM(3.91, 2.94, 0.46, 1.07, 246, 64, 3)

## compare group 2 and group 3
tukeySEM(3.96, 2.94, 0.62, 1.07, 465, 64, 3)
```

`twostage` *Fit a lavaan model using 2-Stage Maximum Likelihood (TSML) estimation for missing data.*

Description

This function automates 2-Stage Maximum Likelihood (TSML) estimation, optionally with auxiliary variables. Step 1 involves fitting a saturated model to the partially observed data set (to variables in the hypothesized model as well as auxiliary variables related to missingness). Step 2 involves fitting the hypothesized model to the model-implied means and covariance matrix (also called the "EM" means and covariance matrix) as if they were complete data. Step 3 involves correcting the Step-2 standard errors (*SE*s) and chi-squared statistic to account for additional uncertainty due to missing data (using information from Step 1; see References section for sources with formulas).

Usage

```
twostage(..., aux, fun, baseline.model = NULL)

lavaan.2stage(..., aux = NULL, baseline.model = NULL)

cfa.2stage(..., aux = NULL, baseline.model = NULL)

sem.2stage(..., aux = NULL, baseline.model = NULL)

growth.2stage(..., aux = NULL, baseline.model = NULL)
```

Arguments

`...`	Arguments passed to the `lavaan::lavaan()` function specified in the `fun` argument. See also `lavaan::lavOptions()`. At a minimum, the user must supply the first two named arguments to `lavaan::lavaan()` (i.e., `model` and `data`).
`aux`	An optional character vector naming auxiliary variable(s) in `data`
`fun`	The character string naming the lavaan function used to fit the Step-2 hypothesized model (`"cfa"`, `"sem"`, `"growth"`, or `"lavaan"`).
`baseline.model`	An optional character string, specifying the lavaan `lavaan::model.syntax()` for a user-specified baseline model. Interested users can use the fitted baseline model to calculate incremental fit indices (e.g., CFI and TLI) using the corrected chi-squared values (see the `anova` method in twostage). If `NULL`, the default "independence model" (i.e., freely estimated means and variances, but all covariances constrained to zero) will be specified internally.

Details

All variables (including auxiliary variables) are treated as endogenous varaibles in the Step-1 saturated model (`fixed.x = FALSE`), so data are assumed continuous, although not necessarily multivariate normal (dummy-coded auxiliary variables may be included in Step 1, but categorical endogenous variables in the Step-2 hypothesized model are not allowed). To avoid assuming multivariate normality, request `se = "robust.huber.white"`. CAUTION: In addition to setting `fixed.x = FALSE` and `conditional.x = FALSE` in `lavaan::lavaan()`, this function will automatically set `meanstructure = TRUE`, `estimator = "ML"`, `missing = "fiml"`, and `test = "standard"`. `lavaan::lavaan()`'s `se` option can only be set to `"standard"` to assume multivariate normality or to `"robust.huber.white"` to relax that assumption.

Value

The twostage object contains 3 fitted lavaan models (saturated, target/hypothesized, and baseline) as well as the names of auxiliary variables. None of the individual models provide the correct model results (except the point estimates in the target model are unbiased). Use the methods in twostage to extract corrected *SE*s and test statistics.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; `<TJorgensen314@gmail.com>`)

References

Savalei, V., & Bentler, P. M. (2009). A two-stage approach to missing data: Theory and application to auxiliary variables. *Structural Equation Modeling, 16*(3), 477–497. doi:10.1080/10705510903008238

Savalei, V., & Falk, C. F. (2014). Robust two-stage approach outperforms robust full information maximum likelihood with incomplete nonnormal data. *Structural Equation Modeling, 21*(2), 280–302. doi:10.1080/10705511.2014.882692

See Also

twostage

Examples

```
## impose missing data for example
HSMiss <- HolzingerSwineford1939[ , c(paste("x", 1:9, sep = ""),
                                      "ageyr","agemo","school")]
set.seed(12345)
HSMiss$x5 <- ifelse(HSMiss$x5 <= quantile(HSMiss$x5, .3), NA, HSMiss$x5)
age <- HSMiss$ageyr + HSMiss$agemo/12
HSMiss$x9 <- ifelse(age <= quantile(age, .3), NA, HSMiss$x9)

## specify CFA model from lavaan's ?cfa help page
HS.model <- '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
'

## use ageyr and agemo as auxiliary variables
out <- cfa.2stage(model = HS.model, data = HSMiss, aux = c("ageyr","agemo"))

## two versions of a corrected chi-squared test results are shown
out
## see Savalei & Bentler (2009) and Savalei & Falk (2014) for details

## the summary additionally provides the parameter estimates with corrected
## standard errors, test statistics, and confidence intervals, along with
## any other options that can be passed to parameterEstimates()
summary(out, standardized = TRUE)
```

```
## use parameter labels to fit a more constrained model
modc <- '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + a*x8 + a*x9
'
outc <- cfa.2stage(model = modc, data = HSMiss, aux = c("ageyr","agemo"))


## use the anova() method to test this constraint
anova(out, outc)
## like for a single model, two corrected statistics are provided
```

`twostage-class` *Class for the Results of 2-Stage Maximum Likelihood (TSML) Estimation for Missing Data*

Description

This class contains the results of 2-Stage Maximum Likelihood (TSML) estimation for missing data. The `summary`, `anova`, `vcov` methods return corrected *SE*s and test statistics. Other methods are simply wrappers around the corresponding lavaan::lavaan methods.

Usage

```
## S4 method for signature 'twostage'
show(object)

## S4 method for signature 'twostage'
summary(object, ...)

## S4 method for signature 'twostage'
anova(object, h1 = NULL, baseline = FALSE)

## S4 method for signature 'twostage'
nobs(object, type = c("ntotal", "ngroups",
  "n.per.group", "norig", "patterns", "coverage"))

## S4 method for signature 'twostage'
coef(object, type = c("free", "user"))

## S4 method for signature 'twostage'
vcov(object, baseline = FALSE)
```

```
## S4 method for signature 'twostage'
fitted.values(object, model = c("target", "saturated",
  "baseline"), type = "moments", labels = TRUE)

## S4 method for signature 'twostage'
fitted(object, model = c("target", "saturated",
  "baseline"), type = "moments", labels = TRUE)

## S4 method for signature 'twostage'
residuals(object, type = c("raw", "cor", "normalized",
  "standardized"))

## S4 method for signature 'twostage'
resid(object, type = c("raw", "cor", "normalized",
  "standardized"))
```

Arguments

`object`	An object of class `twostage`.
`...`	arguments passed to `lavaan::parameterEstimates()`.
`h1`	An object of class `twostage` in which `object` is nested, so that their difference in fit can be tested using `anova` (see **Value** section for details).
`baseline`	`logical` indicating whether to return results for the baseline model, rather than the default target (hypothesized) model.
`type`	The meaning of this argument varies depending on which method it it used for. Find detailed descriptions in the **Value** section under `coef`, `nobs`, and `residuals`.
`model`	`character` naming the slot for which to return the model-implied sample moments (see `fitted.values` description.)
`labels`	`logical` indicating whether the model-implied sample moments should have (row/column) labels.

Value

`show`	`signature(object = "twostage")`: The `show` function is used to display the results of the `anova` method, as well as the header of the (uncorrected) target model results.
`summary`	`signature(object = "twostage", ...)`: The summary function prints the same information from the `show` method, but also provides (and returns) the output of `parameterEstimates(object@target, ...)` with corrected *SE*s, test statistics, and confidence intervals. Additional arguments can be passed to `lavaan::parameterEstimates()`, including `fmi = TRUE` to provide an estimate of the fraction of missing information.
`anova`	`signature(object = "twostage", h1 = NULL, baseline = FALSE)`: The `anova` function returns the residual-based χ^2 test statistic result, as well as the scaled χ^2 test statistic result, for the model in the `target` slot, or for the model in the `baseline` slot if `baseline = TRUE`. The user can also provide

a single additional `twostage` object to the `h1` argument, in which case `anova` returns residual-based and scaled $(\Delta)\chi^2$ test results, under the assumption that the models are nested. The models will be automatically sorted according their degrees of freedom.

`nobs` `signature(object = "twostage", type = c("ntotal", "ngroups", "n.per.group", "norig", "p` The `nobs` function will return the total sample sized used in the analysis by default. Also available are the number of groups or the sample size per group, the original sample size (if any rows were deleted because all variables were missing), the missing data patterns, and the matrix of coverage (diagonal is the proportion of sample observed on each variable, and off-diagonal is the proportion observed for both of each pair of variables).

`coef` `signature(object = "twostage", type = c("free", "user"))`: This is simply a wrapper around the corresponding lavaan::lavaan method, providing point estimates from the `target` slot.

`vcov` `signature(object = "twostage", baseline = FALSE)`: Returns the asymptotic covariance matrix of the estimated parameters (corrected for additional uncertainty due to missing data) for the model in the `target` slot, or for the model in the `baseline` slot if `baseline = TRUE`.

`fitted.values`, `fitted`
`signature(object = "twostage", model = c("target", "saturated", "baseline"))`: This is simply a wrapper around the corresponding lavaan::lavaan method, providing model-implied sample moments from the slot specified in the `model` argument.

`residuals`, `resid`
`signature(object = "twostage", type = c("raw", "cor", "normalized", "standardized"))`: This is simply a wrapper around the corresponding lavaan::lavaan method, providing residuals of the specified `type` from the `target` slot.

Slots

`saturated` A fitted lavaan::lavaan object containing the saturated model results

`target` A fitted lavaan::lavaan object containing the target/hypothesized model results

`baseline` A fitted lavaan::lavaan object containing the baseline/null model results

`auxNames` A character string (potentially of `length == 0`) of any auxiliary variable names, if used

Objects from the Class

Objects can be created via the `twostage()` function.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <`TJorgensen314@gmail.com`>)

See Also

`twostage()`

Examples

```
# See the example from the twostage function
```

Index