Package ‘semTools’

March 1, 2016

Title Useful Tools for Structural Equation Modeling

Version 0.4-11

Description Provides useful tools for structural equation modeling packages.

Depends R(>= 3.0), methods, lavaan(>= 0.5-20), utils, stats, graphics

Suggests MASS, parallel, Amelia, mice, foreign, OpenMx(>= 2.0.0), GPArotation, mnormt, boot

License GPL (>= 2)

LazyData yes

LazyLoad yes

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

Author Sunthud Pornprasertmanit [aut, cre],
Patrick Miller [aut],
Alexander Schoemann [aut],
Yves Rosseel [aut],
Corbin Quick [ctb],
Mauricio Garnier-Villarreal [ctb],
James Selig [ctb],
Aaron Boulton [ctb],
Kristopher Preacher [ctb],
Donna Coffman [ctb],
Mijke Rhemtulla [ctb],
Alexander Robitzsch [ctb],
Craig Enders [ctb],
Ruber Arslan [ctb],
Bell Clinton [ctb],
Pavel Panko [ctb],
Edgar Merkle [ctb],
Terry Jorgensen [ctb],
Steven Chesnut [ctb],
Jarrett Byrnes [ctb],
Jason Rights [ctb]

Maintainer Sunthud Pornprasertmanit <psunthud@gmail.com>

Date/Publication 2016-03-01 10:43:20

NeedsCompilation no

Repository CRAN

R topics documented:

`auxiliary` *Analyzing data with full-information maximum likelihood with auxiliary variables*

Description

Analyzing data with full-information maximum likelihood with auxiliary variables. The techniques used to account for auxiliary variables are both extra-dependent-variables and saturated-correlates approaches (Enders, 2008). The extra-dependent-variables approach is used for all single variables in the model (such as covariates or single-indicator dependent varaible) For variables that are belong to a multiple-indicator factor, the saturated-correlates approach is used. Note that all covariates are treated as endogenous varaibles in this model (fixed.x = FALSE) so multivariate normality is assumed for the covariates. CAUTION: (1) this function will automatically change the missing data handling method to full-information maximum likelihood and (2) this function is still not applicable for categorical variables (because the maximum likelihood method is not available in lavaan for estimating models with categorical variables currently).

Usage

```
auxiliary(model, aux, fun, ...)
cfa.auxiliary(model, aux, ...)
sem.auxiliary(model, aux, ...)
growth.auxiliary(model, aux, ...)
lavaan.auxiliary(model, aux, ...)
```

Arguments

`model`	The `lavaan` object, the parameter table, or lavaan script. If the `lavaan` object is provided, the `lavaan` object must be evaluated with mean structure.
`aux`	The list of auxiliary variable
`fun`	The character of the function name used in running lavaan model (`"cfa"`, `"sem"`, `"growth"`, `"lavaan"`).
`...`	The additional arguments in the `lavaan` function.

Value

The `lavaanStar` object which contains the original `lavaan` object and the additional values of the null model, which need to be adjusted to account for auxiliary variables.

Author(s)

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

References

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

See Also

`lavaanStar`

Examples

```
# Example of confirmatory factor analysis

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

dat <- data.frame(HolzingerSwineford1939, z=rnorm(nrow(HolzingerSwineford1939), 0, 1))

fit <- cfa(HS.model, data=dat, meanstructure=TRUE)
fitaux <- auxiliary(HS.model, aux="z", data=dat, fun="cfa") # Use lavaan script
```

```
fitaux <- cfa.auxiliary(fit, aux="z", data=dat) # Use lavaan output

# Example of multiple groups confirmatory factor analysis

fitgroup <- cfa(HS.model, data=dat, group="school", meanstructure=TRUE)
fitgroupaux <- cfa.auxiliary(fitgroup, aux="z", data=dat, group="school")

## Not run:
# Example of path analysis

mod <- ' x5 ~ x4
x4 ~ x3
x3 ~ x1 + x2'

fitpath <- sem(mod, data=dat, fixed.x=FALSE, meanstructure=TRUE) # fixed.x must be FALSE
fitpathaux <- sem.auxiliary(fitpath, aux="z", data=dat)

# Example of full structural equation modeling

dat2 <- data.frame(PoliticalDemocracy, z=rnorm(nrow(PoliticalDemocracy), 0, 1))
model <- '
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + b*y3 + c*y4
     dem65 =~ y5 + a*y6 + b*y7 + c*y8

    dem60 ~ ind60
    dem65 ~ ind60 + dem60

    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
'
fitsem <- sem(model, data=dat2, meanstructure=TRUE)
fitsemaux <- sem.auxiliary(fitsem, aux="z", data=dat2, meanstructure=TRUE)

# Example of covariate at the factor level

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

fitcov <- cfa(HS.model.cov, data=dat, fixed.x=FALSE, meanstructure=TRUE)
fitcovaux <- cfa.auxiliary(fitcov, aux="z", data=dat)

# Example of  Endogenous variable with single indicator
HS.model.cov2 <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              x7 ~ visual + textual'
```

```
fitcov2 <- sem(HS.model.cov2, data=dat, fixed.x=FALSE, meanstructure=TRUE)
fitcov2aux <- sem.auxiliary(fitcov2, aux="z", data=dat)

# Multiple auxiliary variables
HS.model2 <- ' visual  =~ x1 + x2 + x3
              speed   =~ x7 + x8 + x9'
fit <- cfa(HS.model2, data=HolzingerSwineford1939)
fitaux <- cfa.auxiliary(HS.model2, data=HolzingerSwineford1939, aux=c("x4", "x5"))

## End(Not run)
```

`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.

Objects from the Class

Objects can be created via the `bsBootMiss` function.

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-square values from Bootstrap data sets fitted by the target model

`origChi`: The chi-square value from the original data set

`df`: The degree of freedom of the model

`bootP`: The p-value comparing the original chi-square with the bootstrap distribution

methods

show `signature(object = "BootMiss")`: The `show` function is used to display the results of the Bollen-Stine bootstrap.

summary `signature(object = "BootMiss")`: The summary function prints the same information from the `show` method, but also provides information about the time elapsed, as well as the expected (theoretical) and observed (bootstrap) mean and variance of the chi-squared distribution.

hist `signature(x = "BootMiss", ..., alpha = .05, nd = 2, printLegend = TRUE, legendArgs = list(x = "top`
The `hist` function provides a histogram for the bootstrap distribution of chi-squared, including observed and critical values from the specified `alpha` level. The user can also specify additional graphical parameters to `hist` via `...`, as well as pass a list of arguments to an optional `legend` via `legendArgs`. If the user wants more control over customization, `hist` returns a list of `length == 2`, containing the arguments for the call to `hist` and the arguments to the call for `legend`, respectively.

Author(s)

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

See Also

`bsBootMiss`

Examples

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

---

boreal — *The Boreal Vegetation Dataset*

Description

A part of the Boreal Vegetation dataset from Zuur et al. (2009)

Usage

```
data(boreal)
```

Format

A data frame with 533 observations of 7 variables.

point Point ID

x X-coordinate

y Y-coordinate

nTot species richness

NDVI The normalized difference vegetation index from satellite data

T61 climate

Wet Wetness

References

Zuur, A., Ieno, E. N., Walker, N., Saveliev, A. A., & Smith, G. M. (2009). *Mixed effects models and extensions in ecology with R*. New York: Springer.

Examples

```
head(boreal)
```

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 `bsBootMis`.

The second is designed for users of other software packages (e.g., LISREL, EQS, Amos, or Mplus). Users can import their data, chi-squared 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.

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-implied mean vector 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.

transDataOnly: Logical. If TRUE, the result will provide the transformed data only.

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.

bootSamplesOnly: Logical. If TRUE, the result will provide bootstrap data sets 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.

writeArgs: Optional list. If writeBootData = TRUE or writeBootData = TRUE, user can pass arguments to the 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 function.

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>)

References

Bollen, K. A., \& Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods \& Research, 21*, 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*, 741-763. doi:10.1080/00273170903333590

See Also

BootMiss

Examples

```
## Not run:
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.
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))

## End(Not run)
```

`ci.reliability` *Confidence Interval for a Reliability Coefficient*

Description

A function to calculate the confidence interval for a reliability coefficient: coefficient alpha or coefficient omega.

Usage

```
ci.reliability(data = NULL, S = NULL, N = NULL, aux = NULL,
  type = NULL, inttype = NULL, B = 1000, conf.level = 0.95)
```

Arguments

`data`	The dataset that the reliability coefficient is obtained from. Real data set is required for categorical omega. Also, real data set is required for bootstrap confidence intervals or asymptotic distribution free confidence interval
`S`	Symmetric covariance matrix. Correlation matrix can be specified here but not recommended because, in the function, Confirmatory Factor Analysis (CFA) is analyzed based on covariance matrix.
`N`	The total sample size. Sample size is needed only that `S` is specified.

`aux`	The names of auxiliary variables. Auxiliary variables will not be used as a composite but they will use to handle missing observations. Note that full information maximum likelihood is used if auxiliary variables are specified. See auxiliary for further details.
`type`	The type of reliability coefficient to be calculated: `"alpha"` or `1` for coefficient alpha analyzed by the formula proposed by Cronbach (1951), `"alpha-cfa"` or `2` for coefficient alpha analyzed by CFA with tau-equivalence (method of estimator depending on confidence interval method but none of them is unweighted least square so technically the result is not equal to the formula from Cronbach), `"omega"` for coefficient omega, `"hierarchical"` for hierarchical omega, `"categorical"` for categorical omega. The default is to use hierarchical omega for continuous items and categorical omega for categorical items.
`inttype`	There are 13 options for the methods. See `details` below. The default is to not provide any interval estimates. Based on our simulation studies (Kelley and Pornprasertmanit, in press), bias corrected and accelerated bootstrap, `"bca"`, is recommended for categorical omega. Any bootstrap approaches (e.g., `"bca"` or `"perc"`) are recommended for hierarchical omega, coefficient omega, and coefficient alpha.
`B`	the number of bootstrap replications
`conf.level`	the confidence level (i.e., 1-Type I error rate)

Details

When the coefficient alpha is used, the measurement model is assumed to be true-score equivalent (or tau equivalent) model such that factor loadings are equal across items. When the coefficient omega, hierarchical omega, and categorical omega are used, the measurement model is assumed to be congeneric model (similar to one-factor confirmatory factor analysis model). Coefficient omega assumes that a model fits data perfectly so the variance of the composite scores is calculated from model-implied covariance matrix. However, hierarchical omega allows a model to not fit data perfectly (Kelley and Pornprasertmanit, in press). Categorical omega is a method to calculate coefficient omega for categorical items (Green and Yang, 2009). That is, categorical omega is estimated by the parameter estimates from CFA for categorical items. If coefficient omega or hierarchical omega is used, CFA for continuous items is used, which is not appropriate for categorical items.

If researchers wish to make the measurement model with all parallel items (equal factor loadings and equal error variances), users can specify it by setting `inttype = "parallel"` and `type = "alpha"` or `type = "alpha-cfa"`. See McDonald (1999) for the assumptions of each of these models.

The list below shows all methods to find the confidence interval of reliability.

1. `"none"` or `0` to not find any confidence interval
2. `"parallel"` or `11` to assume that the items are parallel and analyze confidence interval based on wald confidence interval (see van Zyl, Neudecker, & Nel, 2000, Equation 22; also referred as the asymptotic method of Koning & Franses, 2003).
3. `"feldt"` or `12` is based on that $\frac{1-\alpha}{1-al\hat{p}ha}$ is distributed as F distribution with the degree of freedoms of $N-1$ and $(N-1) \times (p-1)$ (Feldt, 1965).
4. `"siotani"` or `13` is the same as the `"feldt"` method but using the degree of freedoms of N and $N \times (p-1)$ (Siotani, Hayakawa, & Fujikoshi, 1985; van Zyl et al., 2000, Equations 7 and 8; also referred as the exact method of Koning & Franses, 2003).

5. `"fisher"` or `21` for the Fisher's z transformation on the correlation coefficient approach, $z = 0.5 \times \log \frac{1+\alpha}{1-\alpha}$, directly on the coefficient alpha and find confidence interval of transformed scale (Fisher, 1950). The variance of the z is $\frac{1}{N-3}$ where N is the total sample size.
6. `"bonett"` or `22` for the Fisher's z transformation on the intraclass correlation approach with the variance of $\frac{2p}{(N-2)(p-1)}$ (Bonett, 2002, Equation 6).
7. `"hakstian"` or `23` uses the cube root transformation and assumes normal distribution on the cube root transformation (Hakstian & Whalen, 1976). The variance of the transformed reliability is based on the degrees of freedom in the `"feldt"` method.
8. `"hakstianbarchard"` or `24` uses a correction of the violation of compound symmetry of covariance matrix by adjusting the degrees of freedom in the `"hakstian"`. This correction is used for the inference in type 12 sampling (both persons and items are sampled from the population of persons and items) See Hakstian and Barchard (2000) for further details.
9. `"icc"` or `25` for the Fisher's z transformation on the intraclass correlation approach, $z = \log 1 - \alpha$. The variance of the z is $\frac{2p}{N(p-1)}$ where p is the number of items (Fisher, 1991, p. 221; van Zyl et al., 2000, p. 277).
10. `"ml"` or `31` to analyze the confidence interval based on normal-theory approach (or multivariate delta method). See van Zyl, Neudecker, & Nel (2000, Equation 21) for the confidence interval of coefficient alpha (also be referred as Iacobucci & Duhachek's, 2003, method). See Raykov (2002) for details for coefficient omega. If users use `analytic.type="cfa"`, the `sem` package will be used to obtain parameter estimates and standard errors used for the formula proposed by Raykov (2002)
11. `"mll"` or `32` to analyze the confidence interval based on normal-theory approach as above. However, the point estimate and standard error were used to build confidence interval using logistic transformation as the note below.
12. `"mlr"` or `33` to analyze the confidence interval based on normal-theory approach (or multivariate delta method). However, the estimation method is to use robust standard error (Satorra and Bentler, 2000).
13. `"mlrl"` or `34` to analyze he confidence interval based on normal-theory approach using robust standard error and logistic transformation (see below).
14. `"adf"` or `35` for asymtotic distribution-free method (see Maydeu-Olivares, Coffman, & Hartman, 2007 for further details for coefficient omega; we use phantom variable approach, Cheung, 2009, and `"WLS"` estimator for coefficient omega, Browne, 1984, in the `lavaan` package, Rosseel, 2012).
15. `"adfl"` or `36` to use asymptotic distribution-free method to derive standard error and parameter estimate. Then, logistic transformation is used to build confidence interval (see below).
16. `"ll"` or `37` for profile likelihood-based confidence interval of both reliability coefficients (Cheung, 2009) analyzed by the `OpenMx` package (Boker et al., 2011)
17. `"bsi"` or `41` for standard bootstrap confidence interval which find the standard deviation across the bootstrap estimates, multiply the standard deviation by critical value, and add and subtract from the reliability estimate.
18. `"bsil"` or `42` to use standard bootstrap confidence interval. However, logistic transformation is used to build confidence interval.
19. `"perc"` or `43` for percentile bootstrap confidence interval.
20. `"bca"` or `44` for bias-corrected and accelerated bootstrap confidence interval.

The logistic transformation (Browne, 1982) is applicable for `"ml"`, `"mlr"`, `"adf"`, and `"bsi"` as `"mll"`, `"mlrl"`, `"adfl"`, and `"bsil"`. The logistic transformation does not assume that the sampling distribution of reliability is symmetric. It acknowledges the fact that reliability ranges from 0 and 1. Logistic transformation is applied to the reliability estimates. Confidence interval is established for the transformed value. The lower and upper bounds of the transformed value is translated back to the reliability estimates. See Browne (1982) or Kelley and Pornprasertmanit (in press) for further details.

Note that not all confidence interval methods are available for all types of reliability and all types of input. For example, bootstrap confidence intervals are not available for covariance matrix input. Parallel confidence intervals are not available for hierarchical omega. We provided appropriate error messages for all impossible combinations.

Value

`est`	The estimated reliability coefficient
`se`	The standard error of the reliability coefficient. If the bootstrap methods are used, this value represents the standard deviation across bootstrap estimates.
`ci.lower`	The lower bound of the computed confidence interval
`ci.upper`	The upper bound of the computed confidence interval
`conf.Level`	The confidence level (i.e., 1 - Type I error rate)
`type`	The type of estimated reliability coefficient (alpha or omega)
`inttype`	The method used to find confidence interval

Author(s)

Sunthud Pornprasertmanit (<`psunthud@gmail.com`>). The previous version was written by Keke Lai (University of California, Merced; <`klai25@ucmerced.edu`>), Leann J. Terry, and Ken Kelley (University of Notre Dame; <`kkelley@nd.edu`>);

References

Boker, S., M., N., Maes, H., Wilde, M., Spiegel, M., Brick, T., et al. (2011). OpenMx: An open source extended structural equation modeling framework. *Psychometrika, 76,* 306-317.

Bonett, D. G. (2002). Sample size requirements for testing and estimating coefficient alpha. *Journal of Educational and Behavioral Statistics, 27,* 335-340.

Browne, M. W. (1982). Covariance structures. In D. M. Hawkins (Ed.), *Topics in applied multivariate analysis* (pp. 72-141). Cambridge, UK: Cambridge University Press.

Browne, M. W. (1984). Asymptotic distribution free methods in the analysis of covariance structures. *British Journal of Mathematical and Statistical Psychology, 24,* 445-455.

Cheung, M. W.-L. (2009). Constructing approximate confidence intervals for parameters with structural constructing approximate confidence intervals for parameters with structural equation models. *Structural Equation Modeling, 16,* 267-294.

Feldt, L.S. (1965). The approximate sampling distribution of Kuder-Richardson reliability coefficient twenty. *Psychometrika, 30*, 357-370.

Fisher, R. A. (1950). *Statistical methods for research workers.* Edinburgh, UK: Oliver & Boyd.

Fisher, R. A. (1991). Statistical methods for research workers. In J.H. Bennett (Ed.), *Statistical methods, experimental design, and scientific inference.* Oxford: Oxford University Press.

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

Hakstian, A. R., & Whalen, T. E. (1976). A k-sample significance test for independent alpha coefficients. *Psychometrika, 41,* 219-231.

Iacobucci, D., & Duhachek, A. (2003). Advancing alpha: measuring reliability with confidence. *Journal of Consumer Psychology, 13,* 478-487.

Kelley, K. & Pornprasertmanit, P. (in press). Confidence intervals for population reliability coefficients: Evaluation of methods, recommendations, and software for homogeneous composite measures. *Psychological Methods.*

Koning, A. J., & Franses, P. H. (2003). *Confidence intervals for Cronbach`s coefficient alpha values* (ERIM Report Series Ref. No. ERS-2003-041-MKT). Rotterdam, The Netherlands: Erasmus Research Institute of Management.

Maydeu-Olivares, A., Coffman, D. L., & Hartmann, W. M. (2007). Asymptotically distribution-free (ADF) interval estimation of coefficient alpha. *Psychological Methods, 12,* 157-176.

McDonald, R. P. (1999). *Test theory: A unified approach*. Mahwah, New Jersey: Lawrence Erlbaum Associates, Publishers.

Raykov, T. (2002). Analytic estimation of standard error and confidence interval for scale reliability. *Multivariate Behavioral Research, 37,* 89-103.

Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software, 48,* 1-36.

Satorra, A. & Bentler, P. M. (2001). A scaled difference chi-square test statistic for moment structure analysis. *Psychometrika, 66,* 507-514.

Siotani, M., Hayakawa, T., & Fujikoshi, Y. (1985). *Modem multivariate statistical analysis: A graduate course and handbook*. Columbus, Ohio: American Sciences Press.

van Zyl, J. M., Neudecker, H., & Nel, D. G. (2000) On the distribution of the maximum likelihood estimator of Cronbach's alpha. *Psychometrika, 65* (3), 271-280.

Yuan, K. & Bentler, P. M. (2002) On robustness of the normal-theory based asymptotic distributions of three reliability coefficient estimates. *Psychometrika, 67* (2), 251-259.

Examples

```
# Use this function for the attitude dataset (ignoring the overall rating variable)
ci.reliability(data=attitude[,-1], type = "omega", inttype = "mlrl", B = 100)

## Forming a hypothetical population covariance matrix
Pop.Cov.Mat <- matrix(.3, 9, 9)
diag(Pop.Cov.Mat) <- 1
ci.reliability(S=Pop.Cov.Mat, N=50, type="alpha", inttype = "bonett")
```

clipboard_saveFile *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, ...)
```

Arguments

`object`	The `lavaan` or `FitDiff` object
`what`	The attributes of the `lavaan` object to be copied in the clipboard. `"summary"` is to copy the screen provided from the `summary` function. `"mifit"` is to copy the result from the `miPowerFit` function. Other attributes listed in the `inspect` method in the lavaan-class could also be used, such as `"coef"`, `"se"`, `"fit"`, `"samp"`, and so on. For the The `FitDiff` object, this argument is not active yet.
`file`	A file name used for saving the result
`tableFormat`	If `TRUE`, save the result in the table format using tabs for seperation. Otherwise, save the result as the output screen printed in the R console.
`...`	Additional argument listed in the `miPowerFit` function (for `lavaan` object only).

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>`)

Examples

```
## Not run:
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", meanstructure=TRUE)
```

```
# Copy the summary of the lavaan object
clipboard(fit)

# Copy the modification indices and the model fit from the miPowerFit function
clipboard(fit, "mifit")

# 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 modification indices and the model fit from the miPowerFit function
saveFile(fit, "out.txt", "mifit")

# 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")

## End(Not run)
```

---

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 that compare fit indices across multiple lavaan outputs. The results can be exported to a clipboard or a file later.

Usage

```
compareFit(..., nested = TRUE)
```

Arguments

`...`	`lavaan` outputs or lists of `lavaan` outputs
`nested`	Logical whether the specified models are nested

Value

A FitDiff object that saves model fit comparisons across multiple models. If the output is not assigned as an object, the output is printed in two parts: 1) nested model comparison (if models are nested) and 2) fit indices summaries. In the fit indices summaries, daggers are tagged to the model with the best fit for each fit index.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

FitDiff, clipboard

Examples

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

fit1 <- cfa(m1, data=HolzingerSwineford1939)

m2 <- ' f1  =~ x1 + x2 + x3 + x4
        f2 =~ x5 + x6 + x7 + x8 + x9 '
fit2 <- cfa(m2, data=HolzingerSwineford1939)
compareFit(fit1, fit2, nested=FALSE)

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

out <- measurementInvariance(HW.model, data=HolzingerSwineford1939, group="school", quiet=TRUE)
compareFit(out)
```

---

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

```
data(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 was generated by the 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

```
data(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 was generated by the 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

```
data(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 was generated using the `lavaan` package.

Examples

```
head(datCat)
```

EFA-class *Class For Rotated Results from EFA*

Description

This class contains the results of rotated exploratory factor analysis

Objects from the Class

Objects can be created via the `orthRotate` or `oblqRotate` function.

Slots

`loading`: Rotated standardized factor loading matrix

`rotate`: Rotation matrix

`gradRotate`: The gradient of the objective function at the rotated loadings

`convergence`: Convergence status

`phi`: Factor correlation. Will be an identity matrix if orthogonal rotation is used.

`se`: Standard errors of the rotated standardized factor loading matrix

`method`: Method of rotation

`call`: The command used to generate this object

methods

- `summary` The `summary` function shows the detailed results of the rotated solution. This function has two arguments: `suppress` and `sort`. The `suppress` argument is used to not show the standardized loading values that less than the specified value. The default is 0.1. The `sort` is used to sort the factor loadings by the sizes of factor loadings in each factor. The default is `TRUE`.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`efaUnrotate`; `orthRotate`; `oblqRotate`

Examples

```
library(lavaan)
unrotated <- efaUnrotate(HolzingerSwineford1939, nf=3, varList=paste0("x", 1:9), estimator="mlr")
summary(unrotated, std=TRUE)
inspect(unrotated, "std")

# Rotated by Quartimin
rotated <- oblqRotate(unrotated, method="quartimin")
summary(rotated)
```

efaUnrotate *Analyze Unrotated Exploratory Factor Analysis Model*

Description

This function will analyze unrotated exploratory factor analysis model. The unrotated solution can be rotated by the `orthRotate` and `oblqRotate` functions.

Usage

```
efaUnrotate(data, nf, varList=NULL, start=TRUE, aux=NULL, ...)
```

Arguments

`data`	A target data frame.
`nf`	The desired number of factors
`varList`	Target observed variables. If not specified, all variables in the target data frame will be used.
`start`	Use starting values in the analysis from the `factanal` function. If `FALSE`, the starting values from the `lavaan` package will be used.
`aux`	The list of auxiliary variables. These variables will be included in the model by the saturated-correlates approach to account for missing information.
`...`	Other arguments in the `cfa` function in the `lavaan` package, such as `ordered`, `se`, or `estimator`

Details

This function will generate a lavaan script for unrotated exploratory factor analysis model such that 1) all factor loadings are estimated, 2) factor variances are fixed to 1, 3) factor covariances are fixed to 0, and 4) the dot products of any pairs of columns in the factor loading matrix are fixed to zero (Johnson and Wichern, 2002). The reason for creating this function in addition to the `factanal` function is that users can enjoy some advanced features from the `lavaan` package such as scaled chi-square, diagonal weighted least square for ordinal indicators, or full-information maximum likelihood.

Value

A `lavaan` output of unrotated exploratory factor analysis solution.

Author(s)

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

Examples

```
unrotated <- efaUnrotate(HolzingerSwineford1939, nf=3, varList=paste0("x", 1:9), estimator="mlr")
summary(unrotated, std=TRUE)
inspect(unrotated, "std")

dat <- data.frame(HolzingerSwineford1939, z=rnorm(nrow(HolzingerSwineford1939), 0, 1))
unrotated2 <- efaUnrotate(dat, nf=2, varList=paste0("x", 1:9), aux="z")
```

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

Description

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

Usage

```
data(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 was 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=.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).

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,* 130-149.

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=.05,
group=1)
```

Arguments

`rmsea0A`	The H0 baseline RMSEA.
`rmsea0B`	The H0 alternative RMSEA (trivial misfit).
`rmsea1A`	The H1 baseline RMSEA.
`rmsea1B`	The H1 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*, 19-35.

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=0.80, alpha=.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,* 130-149.

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=0.80, alpha=.05, group=1)
```

Arguments

`rmsea0A`	The H0 baseline RMSEA.
`rmsea0B`	The H0 alternative RMSEA (trivial misfit).
`rmsea1A`	The H1 baseline RMSEA.
`rmsea1B`	The H1 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*, 19-35.

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

Objects from the Class

Objects can be created via the `compareFit` function.

Slots

`name`**:** The name of each model

`nested`**:** Model fit comparisons between adjacent nested models that are ordered based on their degrees of freedom

`ordernested`**:** The order of nested models regarding to their degrees of freedom

`fit`**:** Fit measures of all models specified in the `name` slot

methods

- `summary` The summary function is used to provide the nested model comparison results and the summary of the fit indices across models. This function has one argument: `fit.measures`. If `"default"` is specified, chi-square values, degree of freedom, *p* value, CFI, TLI, RMSEA, SRMR, AIC, and BIC are provided. If `"all"` is specified, all information given in the `fitMeasures` function is provided. Users may specify a vector of the name of fit indices that they wish.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

compareFit; clipboard

Examples

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

out <- measurementInvariance(HW.model, data=HolzingerSwineford1939, group="school", quiet=TRUE)
modelDiff <- compareFit(out)
summary(modelDiff)
summary(modelDiff, fit.measures="all")
summary(modelDiff, fit.measures=c("aic", "bic"))

## Not run:
# Save results to a file
saveFile(modelDiff, file="modelDiff.txt")

# Copy to a clipboard
clipboard(modelDiff)

## End(Not run)
```

---

fitMeasuresMx — *Find fit measures from an MxModel result*

Description

Find fit measures from MxModel result. The saturate and null models are analyzed in the function and fit measures are calculated based on the comparison with the null and saturate models. The function is adjusted from the `fitMeasures` function in the lavaan package.

Usage

```
fitMeasuresMx(object, fit.measures="all")
```

Arguments

`object`	The target `MxModel` object
`fit.measures`	Target fit measures

Value

A vector of fit measures

Author(s)

The original function is the `fitMeasures` function written by Yves Rosseel in the `lavaan` package. The function is adjusted for an `MxModel` object by Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`nullMx`, `saturateMx`, `standardizeMx`

Examples

```
## Not run:
library(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
latents <- c("G")
factorModel <- mxModel("One Factor",
    type="RAM",
    manifestVars=manifests,
    latentVars=latents,
    mxPath(from=latents, to=manifests),
    mxPath(from=manifests, arrows=2),
    mxPath(from=latents, arrows=2, free=FALSE, values=1.0),
    mxData(observed=cov(demoOneFactor), type="cov", numObs=500)
)
factorFit <- mxRun(factorModel)
round(fitMeasuresMx(factorFit), 3)

# Compare with lavaan
library(lavaan)
script <- "f1 =~ x1 + x2 + x3 + x4 + x5"
fitMeasures(cfa(script, sample.cov = cov(demoOneFactor), sample.nobs = 500, std.lv = TRUE))

## End(Not run)
```

---

fmi — *Fraction of Missing Information.*

Description

This function takes a list of imputed data sets and estimates the Fraction of Missing Information of the Variances and Means for each variable.

Usage

```
fmi(dat.imp, method="saturated", varnames=NULL, group=NULL, exclude=NULL,
digits=3)
```

Arguments

`dat.imp`	List of imputed data sets, the function only accept a list of data frames.
`method`	Specified the model used to estimated the variances and means. Can be one of the following: `"saturated"` (`"sat"`) or `"null"`, the default is `"saturated"`. See Details for more information.
`varnames`	A vector of variables names. This argument allow the user to get the fmi of a subset of variables. The function by default will estimate the fmi for all the variables.
`group`	A variable name defining the groups. This will give the fmi for each group.
`exclude`	A vector of variables names. These variables will be excluded from the analysis.
`digits`	Number of decimals to print in the results.

Details

The function estimates a variance/covariance model for each data set using lavaan. If method = `"saturated"` the function will estimate all the variances and covariances, if method = `"null"` the function will only estimate the variances. The saturated model gives more reliable estimates. With big data sets using the saturated model could take a lot of time. In the case of having problems with big data sets it is helpful to select a subset of variables with `varnames` and/or use the `"null"` model. The function does not accept character variables.

Value

fmi returns a list with the Fraction of Missing Information of the Variances and Means for each variable in the data set.

`Variances`	The estimated variance for each variable, and the respective standard error. Two estimates Fraction of Missing Information of the Variances. The first estimate of fmi (fmi.1) is asymptotic fmi and the second estimate of fmi (fmi.2) is corrected for small numbers of imputations
`Means`	The estimated mean for each variable, and the respective standard error. Two estimates Fraction of Missing Information of the Means. The first estimate of fmi (fmi.1) is asymptotic fmi and the second estimate of fmi (fmi.2) is corrected for small numbers of imputations

Author(s)

Mauricio Garnier Villarreal (University of Kansas; `<mgv@ku.edu>`)

References

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

Savalei, V. & Rhemtulla, M. (2012) On Obtaining Estimates of the Fraction of Missing Information From Full Information Maximum Likelihood, *Structural Equation Modeling: A Multidisciplinary Journal, 19:3*, 477-494.

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.

Examples

```
library(Amelia)
library(lavaan)

modsim <- '
f1 =~ 0.7*y1+0.7*y2+0.7*y3
f2 =~ 0.7*y4+0.7*y5+0.7*y6
f3 =~ 0.7*y7+0.7*y8+0.7*y9'

datsim <- simulateData(modsim,model.type="cfa", meanstructure=TRUE,
                       std.lv=TRUE, sample.nobs=c(200,200))
randomMiss2 <- rbinom(prod(dim(datsim)), 1, 0.1)
randomMiss2 <- matrix(as.logical(randomMiss2), nrow=nrow(datsim))
randomMiss2[,10] <- FALSE
datsim[randomMiss2] <- NA
datsimMI <- amelia(datsim,m=3,idvars="group")

out1 <- fmi(datsimMI$imputations, exclude="group")
out1

out2 <- fmi(datsimMI$imputations, exclude="group", method="null")
out2

out3 <- fmi(datsimMI$imputations, varnames=c("y1","y2","y3","y4"))
out3

out4 <- fmi(datsimMI$imputations, group="group")
out4
```

`impliedFactorStat` *Calculate the model-implied factor means and covariance matrix.*

Description

Calculate reliability values of factors by coefficient omega

Usage

```
impliedFactorStat(object)
impliedFactorMean(object)
impliedFactorCov(object)
```

Arguments

`object`	The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions.

Details

The `impliedFactorMean` function is used to calculated model-implied factor means:

$$\mu = (\boldsymbol{I} - \boldsymbol{B})^{-1} \alpha,$$

where μ is the model-implied factor mean, $\boldsymbol{I}$ is an identity matrix, $\boldsymbol{B}$ is an regression coefficient matrix, and α is a vector of factor intercepts.

The `impliedFactorCov` function is used to calculated model-implied covariance matrix:

$$\Phi = (\boldsymbol{I} - \boldsymbol{B})^{-1} \Psi (\boldsymbol{I} - \boldsymbol{B})^{-1\prime},$$

where Φ is the model-implied factor covariance matrix, Ψ is the residual factor covariance matrix.

The `impliedFactorStat` function is used to provide both model-implied means (if the mean structure is estimated) and covariance matrix.

Value

Model-implied factor means or model-implied factor covariance matrix, or both

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

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

fit <- cfa(HS.model, data=HolzingerSwineford1939, group="school")
impliedFactorStat(fit)
```

`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,* 275-300.

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

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

See Also

- `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 <- indProd(attitude[,-1], var1=1:2, var2=3:4, var3=5:6, match=FALSE, meanC=FALSE,
residualC=TRUE, doubleMC=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)
```

---

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*, 179-182.

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 (g2) 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 is g2. The 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

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

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.* Retrived from MathWorld–A Wolfram Web Resource http://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)
```

---

lavaanStar-class *Class For Representing A (Fitted) Latent Variable Model with Additional Elements*

Description

This is the `lavaan` class that contains additional information about the fit values from the null model. Some functions are adjusted according to the change.

Objects from the Class

Objects can be created via the `auxiliary` function or `runMI`.

Slots

`call`: The function call as returned by `match.called()`.

`timing`: The elapsed time (user+system) for various parts of the program as a list, including the total time.

`Options`: Named list of options that were provided by the user, or filled-in automatically.

`ParTable`: Named list describing the model parameters. Can be coerced to a data.frame. In the documentation, this is called the 'parameter table'.

`Data`: Object of internal class `"Data"`: information about the data.

`SampleStats`: Object of internal class `"SampleStats"`: sample statistics

`Model`: Object of internal class `"Model"`: the internal (matrix) representation of the model

`Fit`: Object of internal class `"Fit"`: the results of fitting the model

`nullfit`: The fit-indices information from the null model

`imputed`: The list of information from running multiple imputation. The first element is the convergence rate of the target and null models. The second element is the fraction missing information. The first estimate of FMI (FMI.1) is asymptotic FMI and the second estimate of FMI (FMI.2) is corrected for small numbers of imputation. The third element is the fit values of the target model by the specified chi-squared methods. The fourth element is the fit values of the null model by the specified chi-square methods.

`auxNames`: The list of auxiliary variables in the analysis.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

see lavaan

See Also

auxiliary; runMI

Examples

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

dat <- data.frame(HolzingerSwineford1939, z=rnorm(nrow(HolzingerSwineford1939), 0, 1))

fit <- cfa(HS.model, data=dat)
fitaux <- auxiliary(HS.model, aux="z", data=dat, fun="cfa")
```

---

lisrel2lavaan *Latent variable modeling in lavaan using LISREL syntax*

Description

This function can be used to estimate a structural equation model in lavaan using LISREL syntax. Data are automatically imported from the LISREL syntax file, or, if data files names are provided within LISREL syntax, from the same directory as the syntax itself, as per standard LISREL data importation.

Usage

```
lisrel2lavaan(filename = NULL, analyze = TRUE, silent = FALSE, ...)
```

Arguments

filename	Filename of the LISREL syntax file. If the `filename` arguement is not specified, the user will be prompted with a file browser with which LISREL syntax file can be selected (recommended).
analyze	Logical. If `analyze==TRUE` (default), data will be automatically imported and analyzed; lavaan summary output displayed and fit object will be returned silently. If `analyze==FALSE`, data will not be imported or analyzed; instead, a lavaan parameter table containing the model specifications will be returned.
silent	Logical. If false (default) the data will be analyzed and output displayed. If true, a fit object will be returned and summary output will not be displayed.
...	Additional arguments to be passed to lavaan.

Value

Output summary is printed to screen and lavaan fit object is returned.

Note

lisrel2lavaan is still in development, and not all LISREL commands are currently functional. A number of known limitations are outlined below. If an error is encountered that is not listed, please contact <corbinq@ku.edu>.

1. data importation lisrel2lavaan currently supports .csv, .dat, and most other delimited data formats. However, formats that are specific to LISREL or PRELIS (e.g., the .PSF file format) cannot be imported. lisrel2lavaan supports raw data, covariance matrices, and correlation matrices (accompanied by a variance vector). Symmetric matrices can either contain lower triangle or full matrix. For MACS structure models, either raw data or summary statistics (that include a mean vector) are supported.
2. variable labels Certain variable labels that are permitted in LISREL cannot be supported in lisrel2lavaan. duplicate labels Most importantly, no two variables of any kind (including phantom variables) should be given the same label when using lisrel2lavaan. If multiple variables are given the same label, lavaan will estimate an incorrect model.

   numeric character labels All variable labels are recommended to include non-numeric characters. In addition, the first character in each variable label is recommended to be non-numeric.

   labels not specified If variable labels are not provided by the user, names will be generated reflecting variable assignment (e.g. 'eta1', 'ksi1'); manifest variables will be in lower case and latent variables in upper case.
3. OU paragraph Not all commands in the OU paragraph are presently supported in lisrel2lavaan. The ME command can be used to specify estimation method; however, not all estimations available in LISREL are currently supported by lavaan. If the specified ME is unsupported, lisrel2lavaan will revert to default estimation. The AD, EP, IT, ND and NP keywords will be ignored. Requests for text files containing starting values (e.g., OU BE) will also be ignored.
4. starting values Certain functionalities related to starting values in LISREL are not yet operational in lisrel2lavaan. Note that due to differences in estimation, starting values are not as important in lavaan model estimation as in LISREL. text file output Requests for text files containing starting values for individual matrices in the in the OU command (e.g., OU BE) are not currently supported. These requests will be ignored.

   MA paragraph Specification of matrix starting values using the MA command is permitted by providing starting values within syntax directly. However, lisrel2lavaan has sometimes encountered problems with importation when files are specified following the MA paragraph.

Author(s)

Corbin Quick (University of Michigan; <corbinq@umich.edu>)

Examples

```
## Not run:
## calling lisrel2lavaan without specifying the filename argument will
## open a file browser window with which LISREL syntax can be selected.
```

```
## any additional arguments to be passed to lavaan for data analysis can
## be specified normally.

lisrel2lavaan(se="standard")
## lavaan output summary printed to screen
## lavaan fit object returned silently

## manual file specification

lisrel2lavaan(filename="myFile.LS8", se="standard")
## lavaan output summary printed to screen
## lavaan fit object returned silently

## 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)
```

`longInvariance` *Measurement Invariance Tests Within Person*

Description

Testing measurement invariance across timepoints (longitudinal) or any context involving the use of the same scale in one case (e.g., a dyad case with husband and wife answering the same scale). The measurement invariance uses a typical sequence of model comparison tests. This function currently works with only one scale.

Usage

```
longInvariance(model, varList, auto = "all", constrainAuto = FALSE,
fixed.x = TRUE, std.lv = FALSE, group=NULL, group.equal="",
group.partial="", warn=TRUE, debug=FALSE, strict = FALSE, quiet = FALSE,
fit.measures = "default", method = "satorra.bentler.2001", ...)
```

Arguments

`model`	lavaan syntax or parameter table
`varList`	A list containing indicator names of factors used in the invariance testing, such as the list that the first element is the vector of indicator names in the first timepoint and the second element is the vector of indicator names in the second timepoint. The order of indicator names should be the same (but measured in different times or different units).
`auto`	The order of autocorrelation on the measurement errors on the similar items across factor (e.g., Item 1 in Time 1 and Time 2). If 0 is specified, the autocorrelation will be not imposed. If 1 is specified, the autocorrelation will imposed for the adjacent factor listed in `varList`. The maximum number can be specified is the number of factors specified minus 1. If `"all"` is specified, the maximum number of order will be used.
`constrainAuto`	If `TRUE`, the function will equate the auto-*covariance* to be equal within the same item across factors. For example, the covariance of item 1 in time 1 and time 2 is equal to the covariance of item 1 in time 2 and time 3.
`fixed.x`	See `lavaan`.
`std.lv`	See `lavaan`.
`group`	See `lavaan`.
`group.equal`	See `lavaan`.
`group.partial`	See `lavaan`.
`warn`	See `lavaan`.
`debug`	See `lavaan`.
`strict`	If `TRUE`, the sequence requires 'strict' invariance. See details for more information.

quiet — If TRUE, a summary is printed out containing an overview of the different models that are fitted, together with some model comparison tests.

fit.measures — Fit measures used to calculate the differences between nested models.

method — The method used to calculate likelihood ratio test. See lavTestLRT for available options

... — Additional arguments in the lavaan function.

Details

If strict = FALSE, the following four models are tested in order:

1. Model 1: configural invariance. The same factor structure is imposed on all units.
2. Model 2: weak invariance. The factor loadings are constrained to be equal across units.
3. Model 3: strong invariance. The factor loadings and intercepts are constrained to be equal across units.
4. Model 4: The factor loadings, intercepts and means are constrained to be equal across units.

Each time a more restricted model is fitted, a chi-square difference test is reported, comparing the current model with the previous one, and comparing the current model to the baseline model (Model 1). In addition, the difference in cfi is also reported (delta.cfi).

If strict = TRUE, the following five models are tested in order:

1. Model 1: configural invariance. The same factor structure is imposed on all units.
2. Model 2: weak invariance. The factor loadings are constrained to be equal across units.
3. Model 3: strong invariance. The factor loadings and intercepts are constrained to be equal across units.
4. Model 4: strict invariance. The factor loadings, intercepts and residual variances are constrained to be equal across units.
5. Model 5: The factor loadings, intercepts, residual variances and means are constrained to be equal across units.

Note that if the chi-square test statistic is scaled (eg. a Satorra-Bentler or Yuan-Bentler test statistic), a special version of the chi-square difference test is used as described in http://www.statmodel.com/chidiff.shtml

Value

Invisibly, all model fits in the sequence are returned as a list.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>); Yves Rosseel (Ghent University; <Yves.Rosseel@UGent.be>)

References

Vandenberg, R. J., and Lance, C. E. (2000). A review and synthesis of the measurement invariance literature: Suggestions, practices, and recommendations for organizational research. *Organizational Research Methods, 3,* 4-70.

See Also

measurementinvariance For the measurement invariance test between groups

Examples

```
model <- ' f1t1 =~ y1t1 + y2t1 + y3t1
              f1t2 =~ y1t2 + y2t2 + y3t2
  f1t3 =~ y1t3 + y2t3 + y3t3'

# Create list of variables
var1 <- c("y1t1", "y2t1", "y3t1")
var2 <- c("y1t2", "y2t2", "y3t2")
var3 <- c("y1t3", "y2t3", "y3t3")
constrainedVar <- list(var1, var2, var3)

# Invariance of the same factor across timepoints
longInvariance(model, auto=1, constrainAuto=TRUE, varList=constrainedVar, data=exLong)

# Invariance of the same factor across timepoints and groups
longInvariance(model, auto=1, constrainAuto=TRUE, varList=constrainedVar, data=exLong, group="sex",
group.equal=c("loadings", "intercepts"))
```

---

mardiaKurtosis — *Finding Mardia's multivariate kurtosis*

Description

Finding Mardia's multivariate kurtosis of multiple variables

Usage

```
mardiaKurtosis(dat)
```

Arguments

dat	The target matrix or data frame with multiple variables

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*, 519-530.

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[,paste("x", 1:9, sep="")])
```

---

mardiaSkew — *Finding Mardia's multivariate skewness*

Description

Finding Mardia's multivariate skewness of multiple variables

Usage

```
mardiaSkew(dat)
```

Arguments

`dat`	The target matrix or data frame with multiple variables

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 chi-square 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*, 519-530.

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[,paste("x", 1:9, sep="")])
```

`maximalRelia` *Calculate maximal reliability*

Description

Calculate maximal reliability of a scale

Usage

```
maximalRelia(object)
```

Arguments

`object`	The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions.

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*, 245-249.

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: 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")
```

`measurementInvariance` *Measurement Invariance Tests*

Description

Testing measurement invariance across groups using a typical sequence of model comparison tests.

Usage

```
measurementInvariance(..., std.lv = FALSE, strict = FALSE, quiet = FALSE,
fit.measures = "default", method = "satorra.bentler.2001")
```

Arguments

`...`	The same arguments as for any lavaan model. See `cfa` for more information.
`std.lv`	If `TRUE`, the fixed-factor method of scale identification is used. If `FALSE`, the first variable for each factor is used as marker variable.
`strict`	If `TRUE`, the sequence requires 'strict' invariance. See details for more information.
`quiet`	If `TRUE`, a summary is printed out containing an overview of the different models that are fitted, together with some model comparison tests.
`fit.measures`	Fit measures used to calculate the differences between nested models.
`method`	The method used to calculate likelihood ratio test. See `lavTestLRT` for available options

Details

If `strict = FALSE`, the following four models are tested in order:

1. Model 1: configural invariance. The same factor structure is imposed on all groups.
2. Model 2: weak invariance. The factor loadings are constrained to be equal across groups.
3. Model 3: strong invariance. The factor loadings and intercepts are constrained to be equal across groups.
4. Model 4: The factor loadings, intercepts and means are constrained to be equal across groups.

Each time a more restricted model is fitted, a chi-square difference test is reported, comparing the current model with the previous one, and comparing the current model to the baseline model (Model 1). In addition, the difference in cfi is also reported (delta.cfi).

If `strict = TRUE`, the following five models are tested in order:

1. Model 1: configural invariance. The same factor structure is imposed on all groups.
2. Model 2: weak invariance. The factor loadings are constrained to be equal across groups.
3. Model 3: strong invariance. The factor loadings and intercepts are constrained to be equal across groups.
4. Model 4: strict invariance. The factor loadings, intercepts and residual variances are constrained to be equal across groups.
5. Model 5: The factor loadings, intercepts, residual variances and means are constrained to be equal across groups.

Note that if the chi-square test statistic is scaled (eg. a Satorra-Bentler or Yuan-Bentler test statistic), a special version of the chi-square difference test is used as described in http://www.statmodel.com/chidiff.shtml

Value

Invisibly, all model fits in the sequence are returned as a list.

Author(s)

Yves Rosseel (Ghent University; <Yves.Rosseel@UGent.be>); Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Vandenberg, R. J., and Lance, C. E. (2000). A review and synthesis of the measurement invariance literature: Suggestions, practices, and recommendations for organizational research. *Organizational Research Methods, 3,* 4-70.

See Also

`longInvariance` for the measurement invariance test within person; `partialInvariance` for the automated function for finding partial invariance models

Examples

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

measurementInvariance(HW.model, data=HolzingerSwineford1939, group="school")
```

---

measurementInvarianceCat *Measurement Invariance Tests for Categorical Items*

Description

Testing measurement invariance across groups using a typical sequence of model comparison tests.

Usage

```
measurementInvarianceCat(..., std.lv = FALSE, strict = FALSE, quiet = FALSE,
fit.measures = "default", method = "satorra.bentler.2001")
```

Arguments

`...`	The same arguments as for any lavaan model. See `cfa` for more information.
`std.lv`	If `TRUE`, the fixed-factor method of scale identification is used. If `FALSE`, the first variable for each factor is used as marker variable.
`strict`	If `TRUE`, the sequence requires 'strict' invariance. See details for more information.
`quiet`	If `TRUE`, a summary is printed out containing an overview of the different models that are fitted, together with some model comparison tests.
`fit.measures`	Fit measures used to calculate the differences between nested models.
`method`	The method used to calculate likelihood ratio test. See `lavTestLRT` for available options

Details

Theta parameterization is used to represent SEM for categorical items. That is, residual variances are modeled instead of the total variance of underlying normal variate for each item. Five models can be tested based on different constraints across groups.

1. Model 1: configural invariance. The same factor structure is imposed on all groups.
2. Model 2: weak invariance. The factor loadings are constrained to be equal across groups.
3. Model 3: strong invariance. The factor loadings and thresholds are constrained to be equal across groups.
4. Model 4: strict invariance. The factor loadings, thresholds and residual variances are constrained to be equal across groups. For categorical variables, all residual variances are fixed as 1.

5. Model 5: The factor loadings, threshoulds, residual variances and means are constrained to be equal across groups.

However, if all items have two items (dichotomous), scalar invariance and weak invariance cannot be separated because thresholds need to be equal across groups for scale identification. Users can specify `strict` option to include the strict invariance model for the invariance testing. See the further details of scale identification and different parameterization in Millsap and Yun-Tein (2004).

Value

Invisibly, all model fits in the sequence are returned as a list.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>) Yves Rosseel (Ghent University; <Yves.Rosseel@UGent.be>)

References

Millsap, R. E., & Yun-Tein, J. (2004). Assessing factorial invariance in ordered-categorical measures. *Multivariate Behavioral Research, 39*, 479-515.

See Also

`measurementInvariance` for measurement invariance for continuous variables; `longInvariance` For the measurement invariance test within person with continuous variables; `partialInvariance` for the automated function for finding partial invariance models

Examples

```
## Not run:
model <- ' f1 =~ u1 + u2 + u3 + u4'

measurementInvarianceCat(model, data = datCat, group = "g", parameterization="theta",
    estimator="wlsmv", ordered = c("u1", "u2", "u3", "u4"))

## End(Not run)
```

`miPowerFit` *Modification indices and their power approach for model fit evaluation*

Description

The model fit evaluation approach using modification indices and expected parameter changes.

Usage

```
miPowerFit(lavaanObj, stdLoad=0.4, cor=0.1, stdBeta=0.1, intcept=0.2, stdDelta=NULL,
delta=NULL, cilevel = 0.90)
```

Arguments

`lavaanObj` The lavaan model object used to evaluate model fit

`stdLoad` The amount of standardized factor loading that one would like to be detected (rejected). The default value is 0.4, which is suggested by Saris and colleagues (2009, p. 571).

`cor` The amount of factor or error correlations that one would like to be detected (rejected). The default value is 0.1, which is suggested by Saris and colleagues (2009, p. 571).

`stdBeta` The amount of standardized regression coefficients that one would like to be detected (rejected). The default value is 0.1, which is suggested by Saris and colleagues (2009, p. 571).

`intcept` The amount of standardized intercept (similar to Cohen's *d* that one would like to be detected (rejected). The default value is 0.2, which is equivalent to a low effect size proposed by Cohen (1988, 1992).

`stdDelta` The vector of the standardized parameters that one would like to be detected (rejected). If this argument is specified, the value here will overwrite the other arguments above. The order of the vector must be the same as the row order from modification indices from the `lavaan` object. If a single value is specified, the value will be applied to all parameters.

`delta` The vector of the unstandardized parameters that one would like to be detected (rejected). If this argument is specified, the value here will overwrite the other arguments above. The order of the vector must be the same as the row order from modification indices from the `lavaan` object. If a single value is specified, the value will be applied to all parameters.

`cilevel` The confidence level of the confidence interval of expected parameter changes. The confidence intervals are used in the equivalence testing.

Details

In the lavaan object, one can inspect the modification indices and expected parameter changes. Those values can be used to evaluate model fit by two methods.

First, Saris, Satorra, and van der Veld (2009, pp. 570-573) used the power to detect modification indices and expected parameter changes to evaluate model fit. First, one should evaluate whether the modification index of each parameter is significant. Second, one should evaluate whether the power to detect a target expected parameter change is high enough. If the modification index is not significant and the power is high, there is no misspecification. If the modification index is significant and the power is low, the fixed parameter is misspecified. If the modification index is significant and the power is high, the expected parameter change is investigated. If the expected parameter change is large (greater than the the target expected parameter change), the parameter is misspecified. If the expected parameter change is low (lower than the target expected parameter change), the parameter is not misspecificied. If the modification index is not significant and the power is low, the decision is inconclusive.

Second, the confidence intervals of the expected parameter changes are formed. These confidence intervals are compared with the range of trivial misspecification, which could be (-`delta`, `delta`) or (0, `delta`) for nonnegative parameters. If the confidence intervals are outside of the range of trivial misspecification, the fixed parameters are severely misspecified. If the confidence intervals are inside the range of trivial misspecification, the fixed parameters are trivially misspecified. If confidence intervals are overlapped the range of trivial misspecification, the decision is inconclusive.

Value

A data frame with these variables:

1. lhs The left-hand side variable (with respect to the lavaan operator)
2. op The lavaan syntax operator: "~~" represents covariance, "=~" represents factor loading, "~" represents regression, and "~1" represents intercept.
3. rhs The right-hand side variable (with respect to the lavaan operator)
4. group The group of the parameter
5. mi The modification index of the fixed parameter
6. epc The expected parameter change if the parameter is freely estimated
7. target.epc The target expected parameter change that represents the minimum size of misspecification that one would like to be detected by the test with a high power
8. std.epc The standardized expected parameter change if the parameter is freely estimated
9. std.target.epc The standardized target expected parameter change
10. significant.mi Represents whether the modification index value is significant
11. high.power Represents whether the power is enough to detect the target expected parameter change
12. decision.pow The decision whether the parameter is misspecified or not based on Saris et al's method: `"M"` represents the parameter is misspecified, `"NM"` represents the parameter is not misspecified, `"EPC:M"` represents the parameter is misspecified decided by checking the expected parameter change value, `"EPC:NM"` represents the parameter is not misspecified decided by checking the expected parameter change value, and `"I"` represents the decision is inconclusive.
13. se.epc The standard errors of the expected parameter changes.
14. lower.epc The lower bound of the confidence interval of expected parameter changes.
15. upper.epc The upper bound of the confidence interval of expected parameter changes.
16. lower.std.epc The lower bound of the confidence interval of standardized expected parameter changes.
17. upper.std.epc The upper bound of the confidence interval of standardized expected parameter changes.
18. decision.ci The decision whether the parameter is misspecified or not based on the confidence interval method: `"M"` represents the parameter is misspecified, `"NM"` represents the parameter is not misspecified, and `"I"` represents the decision is inconclusive.

The row numbers matches with the results obtained from the `inspect(object, "mi")` function.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Hillsdale, NJ: Erlbaum.

Cohen, J. (1992). A power primer. *Psychological Bulletin, 112*, 155-159.

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

See Also

`moreFitIndices` For the additional fit indices information

Examples

```
library(lavaan)

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

fit <- cfa(HS.model, data=HolzingerSwineford1939, group="sex", meanstructure=TRUE)
miPowerFit(fit)

model <- '
  # latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + b*y3 + c*y4
     dem65 =~ y5 + a*y6 + b*y7 + c*y8

  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60

  # residual correlations
    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
'
fit2 <- sem(model, data=PoliticalDemocracy, meanstructure=TRUE)
miPowerFit(fit2, stdLoad=0.3, cor=0.2, stdBeta=0.2, intcept=0.5)
```

`monteCarloMed` *Monte Carlo Confidence Intervals to Test Complex Indirect Effects*

Description

This function takes an expression for an indirect effect, the parameters and standard errors associated with the expression and returns a confidence interval based on a Monte Carlo test of mediation (MacKinnon, Lockwood, & Williams, 2004).

Usage

```
monteCarloMed(expression, ..., ACM=NULL, object=NULL, rep=20000, CI=95, plot=FALSE,
  outputValues=FALSE)
```

Arguments

`expression`	A character scalar representing the computation of an indirect effect. Different parameters in the expression should have different alphanumeric values. Expressions can use either addition (+) or multiplication (*) operators.
`...`	Parameter estimates for all parameters named in `expression`. The order of parameters should follow from `expression` (the first parameter named in `expression` should be the first parameter listed in ...). Alternatively ... can be a vector of parameter estimates.
`ACM`	A matrix representing the asymptotic covariance matrix of the parameters described in `expression`. This matrix should be a symetric matrix with dimensions equal to the number of parameters names in `expression`. Information on finding the ACOV is popular SEM software is described below.)
`object`	A lavaan model object fitted after running the running the `cfa`, `sem`, `growth`, or `lavaan` functions. The model must have parameters labelled with the same labels used in `expression`. When using this option do not specify values for ... or `ACM`
`rep`	The number of replications to compute. Many thousand are reccomended.
`CI`	Width of the confidence interval computed.
`plot`	Should the function output a plot of simulated values of the indirect effect?
`outputValues`	Should the function output all simulated values of the indirect effect?

Details

This function implements the Monte Carlo test of mediation first described in MacKinnon, Lockwood, & Williams (2004) and extends it to complex cases where the indirect effect is more than a function of two parameters. The function takes an expression for the indirect effect, randomly simulated values of the indirect effect based on the values of the parameters (and the associated standard errors) comprising the indirect effect, and outputs a confidence interval of the indirect effect based on the simulated values. For further information on the Monte Carlo test of mediation see MacKinnon, Lockwood, & Williams (2004), Preacher & Selig (in press), and Selig & Preacher (2008). For a Monte Carlo test of mediation with a random effects model see Selig & Preacher (2010).

The asymptotic covariance matrix can be easily found in many popular SEM software applications.

- LISRELIncluding 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
- MplusInclude the command TECH3; in the OUTPUT section. The ACM will be printed in the output.
- lavaan Use the command `vcov` on the fitted lavaan object to print the ACM to the screen

Value

A list with two elements. The first element is the point estimate for the indirect effect. The second element is a matrix with values for the upper and lower limits of the confidence interval generated from the Monte Carlo test of mediation. If `outputValues=TRUE`, output will be a list with a list with the point estimate and values for the upper and lower limits of the confidence interval as the first element and a vector of simulated values of the indirect effect as the second element.

Author(s)

Corbin Quick (University of Michigan; <corbinq@umich.edu>) Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>) James P. Selig (University of New Mexico; <selig@unm.edu>)

References

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 http://quantpsy.org/.

Preacher, K. J., & Selig, J. P. (2012). Advantages of Monte Carlo confidence intervals for indirect effects. *Communication Methods and Measures, 6*, 77-98.

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 http://quantpsy.org/.

Examples

```
#Simple two path mediation
#Write expression of indirect effect
med <- 'a*b'
#Paramter values from analyses
aparam <- 1
bparam<-2
#Asymptotic covariance matrix from analyses
AC <- matrix(c(.01,.00002,
               .00002,.02), nrow=2, byrow=TRUE)
#Compute CI, include a plot
monteCarloMed(med, coef1=aparam, coef2=bparam, outputValues=FALSE, plot=TRUE, ACM=AC)

#Use a vector of parameter estimates as input
aparam<-c(1,2)
monteCarloMed(med, coef1=aparam, outputValues=FALSE, plot=TRUE, ACM=AC)



#Complex mediation with two paths for the indirect effect
#Write expression of indirect effect
med <- 'a1*b1 + a1*b2'
#Paramter values and standard errors from analyses
aparam <- 1
b1param<-2
b2param<-1
```

```
#Asymptotic covariance matrix from analyses
AC <- matrix(c(1,.00002, .00003,
                     .00002,1, .00002,
.00003, .00002, 1), nrow=3, byrow=TRUE)
#Compute CI do not include a plot
monteCarloMed(med, coef1=aparam, coef2=b1param, coef3=b2param, ACM=AC)
```

`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*.

Details

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

$$gammaHat = \frac{p}{p + 2 \times \frac{\chi^2_k - df_k}{N-1}},$$

where p is the number of variables in the model, χ^2_k is the chi-square 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. This formula assumes equal number of indicators across groups.

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

$$adjGammaHat = \left(1 - \frac{K \times p \times (p+1)}{2 \times df_k}\right) \times (1 - gammaHat),$$

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

Corrected Akaike Information Criterion (aic.smallN; Burnham & Anderson, 2003) is the corrected version of aic for small sample size:

$$aic.smallN = f + \frac{2k(k+1)}{N-k-1},$$

where f is the minimized discrepancy function, which is the product of the log likelihood and -2, and k is the number of parameters in the target model.

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}).

$$bic.priorN = f + k \log{(1 + N/N_{prior})},$$

Stochastic information criterion (sic; Preacher, 2006) is similar to aic or bic. This index will account for model complexity in the model's function form, in addition to the number of free parameters. This index will be provided only when the chi-squared value is not scaled. The sic can be computed by

$$sic = \frac{1}{2}\left(f - \log \det I(\hat{\theta})\right),$$

where $I(\hat{\theta})$ is the information matrix of the parameters.

Hannan-Quinn Information Criterion (hqc; Hannan & Quinn, 1979) is used for model selection similar to aic or bic.

$$hqc = f + 2k \log{(\log N)},$$

Note that if Satorra-Bentler or Yuan-Bentler's method is used, the fit indices using the scaled chi-square values are also provided.

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

Value

1. gammaHat Gamma Hat
2. adjGammaHat Adjusted Gamma Hat
3. baseline.rmsea RMSEA of the Baseline (Null) Model
4. aic.smallN Corrected (for small sample size) Akaike Information Criterion
5. bic.priorN Bayesian Information Criterion with specifying the prior sample size
6. sic Stochastic Information Criterion
7. hqc Hannan-Quinn Information Criterion
8. gammaHat.scaled Gamma Hat using Scaled Chi-square
9. adjGammaHat.scaled Adjusted Gamma Hat using Scaled Chi-square
10. baseline.rmsea.scaled RMSEA of the Baseline (Null) Model using Scaled Chi-square

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>) Terrence Jorgensen (University of Amsterdam; <TJorgensen314@gmail.com>) Aaron Boulton (University of North Carolina, Chapel Hill; <aboulton@email.unc.edu>) Ruben Arslan (Humboldt-University of Berlin, <rubenarslan@gmail.com>) Yves Rosseel (Ghent University; <Yves.Rosseel@UGent.be>)

References

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*, 305-319.

Kuha, J. (2004). AIC and BIC: Comparisons of assumptions and performance. *Sociological Methods Research, 33*, 188-229.

Preacher, K. J. (2006). Quantifying parsimony in structural equation modeling. *Multivariate Behavioral Research, 43*, 227-259.

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.* New York: Guilford.

See Also

- `miPowerFit` For the modification indices and their power approach for model fit evaluation
- `nullRMSEA` For RMSEA of the null 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
Sigma	A positive-definite symmetric matrix specifying the covariance matrix of the variables
skewness	A vector of skewness of the variables
kurtosis	A vector of excessive kurtosis of the variables
empirical	If TRUE, mu and Sigma specify the empirical not population mean and covariance matrix

Value

A data matrix

Author(s)

The original function is the simulateData function written by Yves Rosseel in the lavaan package. The function is adjusted for a convenient usage by Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

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

Examples

```
mvrnonnorm(100, c(1, 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 models are nested or equivalent based on Bentler and Satorra's (2010) procedure.

Usage

```
net(..., crit = .0001)
```

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 chi-squared 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 is based on a limited-information estimator (analyzing 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 does not apply to models that estimate thresholds for categorical data, so an error message will be issued if such a model is provided.

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*, 111-123. doi:10.1037/a0019625

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

Objects from the Class

Objects can be created via the `net` function.

Slots

`test`: Logical matrix of results of nesting and equivalence testing across models

`df`: The degrees of freedom of tested models

methods

- `summary` The summary function is used to provide the results in narrative.

Author(s)

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

See Also

`net`

Examples

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

`nullMx` *Analyzing data using a null model*

Description

Analyzing data using a null model by full-information maximum likelihood. In the null model, all means and covariances are free if items are continuous. All covariances are fixed to 0. For ordinal variables, their means are fixed as 0 and their variances are fixed as 1 where their thresholds are estimated. In multiple-group model, all means are variances are separately estimated.

Usage

```
nullMx(data, groupLab = NULL)
```

Arguments

`data`	The target data frame
`groupLab`	The name of grouping variable

Value

The `MxModel` object which contains the analysis result of the null model.

Author(s)

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

See Also

`saturateMx`, `fitMeasuresMx`, `standardizeMx`

Examples

```
## Not run:
library(OpenMx)
data(demoOneFactor)
nullModel <- nullMx(demoOneFactor)

## End(Not run)
```

`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`, calculate the null model from the scaled test.
`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 `http://davidakenny.net/cm/fit.htm`.

Value

A value of RMSEA of the null model. This value is hidden. Users may be assigned the output of this function to any object for further usage.

Author(s)

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

References

Kenny, D. A., Kaniskan, B., & McCoach, D. B. (2011). *The performance of RMSEA in models with small degrees of freedom.* Unpublished paper, University of Connecticut.

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(nPerPar, facPlc, nAlloc=100, syntax, dataset, names='default',
leaveout=0, ...)
```

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 (there should NOT be parcels of size "1").
`facPlc`	A list of vectors, each corresponding to a factor, specifying the variables in 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.
`nAlloc`	The number of random allocations of items to parcels to generate.
`syntax`	lavaan syntax. If substituted with a file name, parcelAllocation will print output data sets to a specified folder rather than analyzing using lavaan (note for Windows users: file path must be specified using forward slashes).
`dataset`	Data set. Can be file path or R object (matrix or dataframe). If the data has missing values multiple imputation before parceling is recommended.
`names`	(Optional) A character vector containing the names of parceled variables.

`leaveout` A vector of variables to be left out of randomized parceling. Either variable names or column numbers are allowed.

`...` Additional arguments to be passed to lavaan

Details

This function implements the random item to parcel allocation procedure described in Sterba (2011) and Sterba and MccCallum (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), and repeats this process for a user specified number of random allocations. Results from all fitted models are summarized and output. For further details on the benefits of the random allocation of itesm to parcels see Sterba (2011) and Sterba and MccCallum (2010).

Value

`Estimates` A data frame containing results related to parameter estimates with columns corresponding to parameter names, average parameter estimates across allocations, the standard deviation of parameter estimates across allocations, the minimum parameter estimate across allocations, the maximum parameter estimate across allocations, the range of parameter estimates across allocations, and the proportions of allocations in which the parameter estimate is significant.

`SE` A data frame containing results related to standard errors with columns corresponding to parameter names, average standard errors across allocations, the standard deviation of standard errors across allocations, the minimum standard error across allocations, the maximum standard error across allocations, and the range of standard errors across allocations.

`Fit` A data frame containing results related to model fit with columns corresponding to fit index names, the average of each index across allocations, the standard deviation of each fit index across allocations, the minimum of each fit index across allocations, the maximum of each fit index across allocations, and the range of each fit index across allocations.

Author(s)

Corbin Quick (University of Michigan; `<corbinq@umich.edu>`) Alexander M. Schoemann (East Carolina University; `<schoemanna@ecu.edu>`)

References

Sterba, S.K. (2011). Implications of parcel-allocation variability for comparing fit of item-solutions and parcel-solutions. *Structural Equation Modeling, 18,* 554-577.

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,* 322-358.

Examples

```
#Fit 3 factor CFA to simulated data.
#Each factor has 9 indicators that are randomly parceled into 3 parcels
#Lavaan syntax for the model to be fit to parceled data
```

```
library(lavaan)

syntax <- 'La =~ V1 + V2 + V3
           Lb =~ V4 + V5 + V6
'
#Parcel and fit data 20 times. The actual parcel number should be higher than 20 times.
name1 <- colnames(simParcel)[1:9]
name2 <- colnames(simParcel)[10:18]
parcelAllocation(list(c(3,3,3),c(3,3,3)), list(name1, name2), nAlloc=20, syntax=syntax,
dataset=simParcel)
```

---

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). The result from `measurementInvariance` or `measurementInvarianceCat` could be used in this argument directly.
`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 `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 `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 `wald` 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 (the default is 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. Chi-square 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: Guilford.

See Also

`measurementInvariance` for measurement invariance for continuous variables; `measurementInvarianceCat` for measurement invariance for categorical variables; `wald` for multivariate Wald test

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")

## Not run:
partialInvariance(models, "metric", free = "x5") # "x5" is free across groups in advance
partialInvariance(models, "metric", fix = "x4") # "x4" is fixed across groups in advance

# Use the result from the measurementInvariance function
HW.model <- ' visual =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed =~ x7 + x8 + x9 '

models2 <- measurementInvariance(HW.model, data=HolzingerSwineford1939, group="school")
partialInvariance(models2, "scalar")

# Conduct weak invariance testing manually by using fixed-factor
# method of scale identification for dichotomous variables

f <- rnorm(1000, 0, 1)
```

```
u1 <- 0.9*f + rnorm(1000, 1, sqrt(0.19))
u2 <- 0.8*f + rnorm(1000, 1, sqrt(0.36))
u3 <- 0.6*f + rnorm(1000, 1, sqrt(0.64))
u4 <- 0.7*f + rnorm(1000, 1, sqrt(0.51))
u1 <- as.numeric(cut(u1, breaks = c(-Inf, 0, Inf)))
u2 <- as.numeric(cut(u2, breaks = c(-Inf, 0.5, Inf)))
u3 <- as.numeric(cut(u3, breaks = c(-Inf, 0, Inf)))
u4 <- as.numeric(cut(u4, breaks = c(-Inf, -0.5, Inf)))
g <- rep(c(1, 2), 500)
dat2 <- data.frame(u1, u2, u3, u4, g)

configural2 <- "
f1 =~ NA*u1 + u2 + u3 + u4
u1 | c(t11, t11)*t1
u2 | c(t21, t21)*t1
u3 | c(t31, t31)*t1
u4 | c(t41, t41)*t1
f1 ~~ c(1, 1)*f1
f1 ~ c(0, NA)*1
u1 ~~ c(1, 1)*u1
u2 ~~ c(1, NA)*u2
u3 ~~ c(1, NA)*u3
u4 ~~ c(1, NA)*u4
"

outConfigural2 <- cfa(configural2, data = dat2, group = "g", parameterization="theta",
estimator="wlsmv", ordered = c("u1", "u2", "u3", "u4"))

weak2 <- "
f1 =~ NA*u1 + c(f11, f11)*u1 + c(f21, f21)*u2 + c(f31, f31)*u3 + c(f41, f41)*u4
u1 | c(t11, t11)*t1
u2 | c(t21, t21)*t1
u3 | c(t31, t31)*t1
u4 | c(t41, t41)*t1
f1 ~~ c(1, NA)*f1
f1 ~ c(0, NA)*1
u1 ~~ c(1, 1)*u1
u2 ~~ c(1, NA)*u2
u3 ~~ c(1, NA)*u3
u4 ~~ c(1, NA)*u4
"

outWeak2 <- cfa(weak2, data = dat2, group = "g", parameterization="theta", estimator="wlsmv",
ordered = c("u1", "u2", "u3", "u4"))
modelsCat <- list(configural = outConfigural2, metric = outWeak2)

partialInvarianceCat(modelsCat, type = "metric")

partialInvarianceCat(modelsCat, type = "metric", free = "u2")
partialInvarianceCat(modelsCat, type = "metric", fix = "u3")

# Use the result from the measurementInvarianceCat function
```

```
model <- ' f1 =~ u1 + u2 + u3 + u4
           f2 =~ u5 + u6 + u7 + u8'

modelsCat2 <- measurementInvarianceCat(model, data = datCat, group = "g",
parameterization="theta", estimator="wlsmv", strict = TRUE)

partialInvarianceCat(modelsCat2, type = "scalar")

## End(Not run)
```

`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 is a modified version of `parcelAllocation` which can be used with only one SEM in isolation. The `PAVranking` function repeatedly generates a specified number of random item-to-parcel allocations, and then fits two models to each allocation. 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 `seed`).

Usage

```
PAVranking(nPerPar, facPlc, nAlloc=100, parceloutput = 0,
   syntaxA, syntaxB, dataset, names = NULL,
   leaveout=0, seed=NA, ...)
```

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.
`nAlloc`	The number of random allocations of items to parcels to generate.
`syntaxA`	lavaan syntax for Model A. Note that, for likelihood ratio test (LRT) results to be interpreted, Model A should be nested within Model B (though the function will still provide results when Models A and B are nonnested).
`syntaxB`	lavaan syntax for Model B. Note that, for likelihood ratio test (LRT) results to be appropriate, Model A should be nested within Model B (though the function will still provide results when Models A and B are nonnested).

`dataset`	Item-level dataset
`parceloutput`	folder where parceled data sets will be outputted (note for Windows users: file path must specified using forward slashes).
`seed`	(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.
`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.
`...`	Additional arguments to be passed to `lavaan`

Details

This is a modified version of `parcelAllocation` which was, in turn, based on the 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 uses a modified version of parcelAllocation to generate a given number (`nAlloc`) of item-to-parcel allocations. Then, `PAVranking` provides the following new developments: specifying more than one SEM and producing results for Model A and Model B separately that summarize parcel allocation variability in estimates, standard errors, and fit indices. `PAVranking` also newly produces results summarizing parcel allocation variability in model selection index values and model ranking between Models A and B. Additionally, `PAVranking` newly allows for nonconverged solutions and outputs the proportion of allocations that converged as well as the proportion of proper solutions (results are summarized for 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).

NOTE: This function requires the `lavaan` package. Missing data code needs to be `NA`. If function returns `"Error in plot.new() : figure margins too large,"` user may need to increase size of the plot window and rerun.

Value

`Estimates_A, Estimates_B`
: A table containing results related to parameter estimates (in table Estimates_A for Model A and in table Estimates_B for Model B) with columns corresponding to parameter name, average parameter estimate across allocations, standard deviation of parameter estimate across allocations, the maximum parameter estimate across allocations, the minimum parameter estimate across allocations, the range of parameter estimates across allocations, and the percent of allocations in which the parameter estimate is significant.

`SE_A, SE_B`
: A table containing results related to standard errors (in table SE_A for Model A and in table SE_B for Model B) with columns corresponding to parameter

name, average standard error across allocations, the standard deviation of standard errors across allocations, the maximum standard error across allocations, the minimum standard error across allocations, and the range of standard errors across allocations.

`Fit_A, Fit_B` A table containing results related to model fit (in table Fit_A for Model A and in table Fit_B for Model B) with columns corresponding to fit index name, the average of the fit index across allocations, the standard deviation of the fit index across allocations, the maximum of the fit index across allocations, the minimum of the fit index across allocations, the range of the fit index across allocations, and the percent of allocations where the chi-square test of absolute fit was significant.

`LRT Summary, Model A vs. Model B`
A table with columns corresponding to: average likelihood ratio test (LRT) statistic for comparing Model A vs. Model B (null hypothesis is no difference in fit between Models A and B in the population), degrees of freedom (i.e. difference in the number of free parameters between Models A and B), as well as the standard deviation, maximum, and minimum of LRT statistics across allocations, and the percent of allocations where the LRT was significant (indicating preference for the more complex Model B).

`LRT Summary, Model A vs. Model B`
A table with columns corresponding to: average likelihood ratio test (LRT) statistic for comparing Model A vs. Model B (null hypothesis is no difference in fit between Models A and B in the population), degrees of freedom (i.e. difference in the number of free parameters between Models A and B), as well as the standard deviation, maximum, and minimum of LRT statistics across allocations, and the percent of allocations where the LRT was significant (indicating preference for the more complex Model B).

`Fit index differences`
A table containing percentage of allocations where Model A is preferred over Model B according to BIC, AIC, RMSEA, CFI, TLI and SRMR and where Model B is preferred over Model A according to the same indices. Also includes the average amount by which the given model is preferred (calculated only using allocations where it was preferred).

`Fit index difference histograms`
Histograms are automatically outputted showing the distribution of the differences (Model A - Model B) for each fit index and for the p-value of the likelihood ratio difference test.

`Percent of Allocations with | BIC Diff | > 10`
A table containing the percentage of allocations with (BIC for Model A) - (BIC for Model B) < -10, indicating "very strong evidence" to prefer Model A over Model B and the percentage of allocations with (BIC for Model A) - (BIC for Model B) > 10, indicating "very strong evidence" to prefer Model B over Model A (Raftery, 1995).

`Converged and proper`
A table containing the proportion of allocations that converged for Model A, Model B, and both models, and the proportion of allocations with converged and proper solutions for Model A, Model B, and both models.

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 on which this function is based.

References

Raftery, A. E. (1995). Bayesian model selection in social research. *Sociological Methodology, 25*, 111-163.

Sterba, S. K. (2011). Implications of parcel-allocation variability for comparing fit of item-solutions and parcel-solutions. *Structural Equation Modeling: A Multidisciplinary Journal, 18*(4), 554-577.

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.

"Sterba, S. K., & Rights, J. D. (2016). Effects of parceling on model selection: Parcel-allocation variability in model ranking. *Psychological Methods*. http://dx.doi.org/10.1037/met0000067

See Also

parcelAllocation

Examples

```
## Not run:
## Lavaan syntax for Model A: a 2 Uncorrelated
## factor CFA model to be fit to parceled data

parmodelA <- '
   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 ~~ 0*f2
'

## Lavaan syntax for Model B: a 2 Correlated
## factor CFA model to be fit to parceled data

parmodelB <- '
```

```
    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
PAVranking(nPerPar=list(c(3,3,3),c(3,3,3)),
  facPlc=list(f1name,f2name), nAlloc=100,
  parceloutput=0, syntaxA=parmodelA,
  syntaxB=parmodelB, dataset = simParcel,
  names=list("p1f1","p2f1","p3f1","p1f2","p2f2","p3f2"),
  leaveout=0)

## End(Not run)
```

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

Description

The function `permuteMeasEq` accepts 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) 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 model is 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 parameters (see `param`). This function is for testing measurement equivalence only across groups, not occasions. 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.

Modification indices for equality constraints on parameters specified in param are calculated from the constrained model (con) using the function lavTestScore, which can also be used to request expected parameter changes if the user has a need for them.

Usage

```
permuteMeasEq(nPermute, con, uncon = NULL, null = NULL,
              param = NULL, AFIs = NULL, moreAFIs = NULL,
              maxSparse = 10, maxNonconv = 10, showProgress = TRUE)
```

Arguments

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

con The constrained lavaan object, in which the parameters specified in param are constrained to equality across all groups. In the case of testing *configural* invariance, con is the configural model (implicitly, the unconstrained model is the saturated model, so use the default uncon = NULL).

uncon Optional. The unconstrained lavaan object, in which the parameters specified in param are freely estimated in all groups. Only in the case of testing *configural* invariance should this argument be 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 A character vector indicating which parameters are constrained across groups in con and are unconstrained in uncon. Parameter names must match those returned by names(coef(uncon)), but omitting any group-specific suffixes (e.g., "f1~1" rather than "f1~1.g2") or user-specified labels. Alternatively, to test an entire set of a certain type of parameter, param may take any one of the following values: "loadings", "intercepts", "thresholds", or "residuals".

AFIs A character vector indicating which alternative fit indices (or chi-squared itself) are to be used to test the multiparameter omnibus null hypothesis of no group differences in any parameters specified in param. Any fit measures returned by 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 of no group differences in any parameters specified in param.

maxSparse 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 randomly permuted group assignment can yield a sample for which the model does not converge on a solution. If such a sample occurs, group assignment is randomly permuted again, 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.

Details

The multiparameter omnibus null hypothesis of measurement equivalence/invariance is that there are no group differences in any measurement parameters. This can be tested using the `anova` method on nested `lavaan` objects, as seen in the output of `measurementInvariance`, or by inspecting the change in alternative fit indices (AFIs) such as the CFI. See Cheung & Rensvold (2002) or Meade, Johnson, & Braddy (2008) for details.

If the multiparameter omnibus null hypothesis is rejected, partial invariance can still be established by freeing parameters that differ across groups, while maintaining equality constraints for at least two other 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.

Value

The permuteMeasEq object representing the results of testing measurement equivalence (the multiparameter omnibus test) and DIF (modification indices).

Author(s)

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

References

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:10.1037/0021-9010.93.3.568

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

`TukeyHSD`, `measurementInvariance`, `measurementInvarianceCat`

Examples

```
## Not run:

########################
## Traditional Method ##
########################

## 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
'
miout <- measurementInvariance(mod.config, data = HS, std.lv = TRUE,
                               group = "ageGroup")

(fit.config <- miout[["fit.configural"]])
(fit.metric <- miout[["fit.loadings"]])
(fit.scalar <- miout[["fit.intercepts"]])

########################
## Permutation Method ##
########################

## fit indices of interest for multiparameter omnibus test
myAFIs <- c("chisq","cfi","rmsea","srmr","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(123)
out.config <- permuteMeasEq(nPermute = 20, con = fit.config)
out.config

## test metric equivalence
set.seed(456)
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(789)
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)


## End(Not run)
```

`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).

Objects from the Class

Objects can be created via the `permuteMeasEq` function.

Slots

`PT`: A `data.frame` returned by a call to `parTable` on the constrained model

`ANOVA`: A vector indicating the results of the observed chi-squared (difference) test, based on the theoretical chi-squared distribution

`AFI.obs`: A vector of observed changes in user-selected alternative fit indices (AFIs)

`AFI.dist`: The permutation distribution(s) of AFI(s). A `data.frame` with `nrow == n.Permutations` and one column for each AFI.

`AFI.pval`: A vector of *p* values (one for each AFI 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 of no group differences were true

MI.obs: A vector of observed Lagrange Multipliers (modification indices) associated with the equality constraints on the user-specified parameters. This is a subset of the output returned by a call to lavTestScore on the constrained model.

MI.dist: The permutation distribution of the maximum modification index (among those seen in slot MI.obs) at each permutation of group assignment

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: A 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. A 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

Methods

show `signature(object = "permuteMeasEq"):` The show function is used to summarize the results of the multiparameter omnibus test of measurement equivalence, using the user-specified AFIs. The parametric chi-squared (difference) test is also displayed.

summary `signature(object = "permuteMeasEq", alpha = .05, nd = 3):` The summary function prints the same information from the show method, but also provides a table summarizing 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 the parametric chi-squared distribution with *df* = 1. 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 uncon = NULL to test configural invariance, in which case the invisibly returned object is object.

hist `signature(x = "permuteMeasEq", ..., AFI, alpha = .05, nd = 3, printLegend = TRUE, legendArgs = lis`
The hist function provides a histogram for the permutation distribution of the specified AFI, including observed and critical values from the specified alpha level. The user can also specify additional graphical parameters to hist via ..., as well as pass a list of arguments to an optional legend via legendArgs. If AFI = "chisq", then the probability density and critical value from the theoretical chi-squared distribution are also included in the plot. If the user wants more control over customization, hist returns a list of length == 2, containing the arguments for the call to hist and the arguments to the call for legend, respectively.

Author(s)

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

See Also

permuteMeasEq

Examples

```
# See the example from the permuteMeasEq function
```

`plotProbe` *Plot the graphs for probing latent interaction*

Description

This function will plot the line graphs representing the simple effect of the independent variable given the values of the moderator.

Usage

```
plotProbe(object, xlim, xlab="Indepedent Variable", ylab="Dependent Variable", ...)
```

Arguments

`object`	The result of probing latent interaction obtained from `probe2WayMC`, `probe2WayRC`, `probe3WayMC`, or `probe3WayRC` function.
`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
`...`	Any addition argument for the `plot` function

Value

None. This function will plot the simple main effect only.

Author(s)

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

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, std.lv=FALSE)
result2wayMC <- probe2WayMC(fitMC2way, c("f1", "f2", "f12"), "f3", "f2", 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, meanstructure=TRUE, std.lv=FALSE)
result3wayMC <- probe3WayMC(fitMC3way, c("f1", "f2", "f3", "f12", "f13", "f23", "f123"),
"f4", c("f1", "f2"), c(-1, 0, 1), 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 non-central chi-square 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 groupand ε is the population RMSEA. Next, the noncentral chi-square distribution with a specified degree of freedom and noncentrality parameter is plotted. Thus, the x-axis represent the sample chi-square value. The sample chi-square 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 chi-square value obtained from the noncentral chi-square 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*, 305-319.

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods, 1,* 130-149.

Steiger, J. H. (1998). A note on multiple sample extensions of the RMSEA fit index. *Structural Equation Modeling, 5*, 411-419.

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(rmsea=c(.05, .08), n=200, df=20, ptile=0.95, rmseaScale = TRUE)
plotRMSEAdist(rmsea=c(.05, .01), n=200, df=20, ptile=0.05, 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=.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: http://quantpsy.org/rmsea/rmseaplot.htm.

Value

1. plot 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,* 19-35.

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods, 1,* 130-149.

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,* 23-41.

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,* 411-419.

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.

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(.025, .075, 23, 100, 500, 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=.05, group=1, ...)
```

Arguments

rmsea0A	The H0 baseline RMSEA.
rmsea0B	The H0 alternative RMSEA (trivial misfit).
rmsea1A	The H1 baseline RMSEA.
rmsea1B	The H1 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*, 19-35.

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)
```

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)
```

Arguments

`fit`	The lavaan model object used to evaluate model fit
`nameX`	The vector of the factor names used as the predictors. The first-order factor will be listed first. The last name must be the name representing the interaction term.
`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 on each moderator variable value will be probed.
`valProbe`	The values of the moderator that will be used to probe the effect of the other independent factor.

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 varaible (X) and the moderator (Z) be

$$Y = b_0 + b_1X + b_2Z + b_3XZ + r,$$

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.

For probing 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

$$b_{0|X=0,Z} = b_0 + b_2Z.$$

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

$$Var\left(b_{0|X=0,Z}\right) = Var\left(b_0\right) + 2ZCov\left(b_0, b_2\right) + Z^2 Var\left(b_2\right)$$

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

$$Var\left(b_{X|Z}\right) = Var\left(b_1\right) + 2ZCov\left(b_1, b_3\right) + Z^2 Var\left(b_3\right)$$

Wald statistic is used for test statistic.

Value

A list with two elements:

1. SimpleIntercept The intercepts given each value of the moderator. This element will be shown only if the factor intercept is estimated (e.g., not fixed as 0).
2. SimpleSlope The 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 simple slope. The fourth column is the Wald (z) statistic. The fifth column is the *p*-value testing whether the simple intercepts or slopes are different from 0.

Author(s)

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

References

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: 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*, 275-300.

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*, 437-448.

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

```
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, std.lv=FALSE)
summary(fitMC2way)

result2wayMC <- probe2WayMC(fitMC2way, c("f1", "f2", "f12"), "f3", "f2", c(-1, 0, 1))
result2wayMC
```

`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 (Pornprasertmanit, Schoemann, Geldhof, & Little, submitted)

Usage

```
probe2WayRC(fit, nameX, nameY, modVar, valProbe)
```

Arguments

`fit`	The lavaan model object used to evaluate model fit
`nameX`	The vector of the factor names used as the predictors. The first-order factor will be listed first. The last name must be the name representing the interaction term.
`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 on each moderator variable value will be probed.
`valProbe`	The values of the moderator that will be used to probe the effect of the other independent factor.

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 Pornprasertmanit, Schoemann, Geldhof, and Little (submitted) 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 `probe2WayMC` for further details.

Value

A list with two elements:

1. SimpleIntercept The intercepts given each value of the moderator. This element will be shown only if the factor intercept is estimated (e.g., not fixed as 0).
2. SimpleSlope The 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 simple slope. The fourth column is the Wald (z) statistic. The fifth column is the p-value testing whether the simple intercepts or slopes are different from 0.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Lance, C. E. (1988). Residual centering, exploratory and confirmatory moderator analysis, and decomposition of effects in path models containing interactions. *Applied Psychological Measurement, 12*, 163-175.

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*, 497-519.

Marsh, H. W., Wen, Z., & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*, 275-300.

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.
- `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

```
library(lavaan)

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
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
"
```

```
fitRC2way <- sem(model1, data=dat2wayRC, meanstructure=TRUE, std.lv=FALSE)
summary(fitRC2way)

result2wayRC <- probe2WayRC(fitRC2way, c("f1", "f2", "f12"), "f3", "f2", c(-1, 0, 1))
result2wayRC
```

---

probe3WayMC *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

```
probe3WayMC(fit, nameX, nameY, modVar, valProbe1, valProbe2)
```

Arguments

fit	The lavaan model object used to evaluate model fit
nameX	The vector of the factor names used as the predictors. The three first-order factors will be listed first. Then the second-order factors will be listeed. The last element of the name will represent the three-way interaction. Note that the fourth element must be the interaction between the first and the second variables. The fifth element must be the interaction between the first and the third variables. The sixth element must be the interaction between the second and the third variables.
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.

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 varaible (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.

For probing 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 varaible 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

$$Var\left(b_{0|X=0,Z,W}\right) = Var\left(b_0\right) + Z^2Var\left(b_2\right) + W^2Var\left(b_3\right) + Z^2W^2Var\left(b_6\right) + 2ZCov\left(b_0, b_2\right) + 2WCov\left(b_0, b_3\right) + 2ZWC$$

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

$$Var\left(b_{X|Z,W}\right) = Var\left(b_1\right) + Z^2Var\left(b_4\right) + W^2Var\left(b_5\right) + Z^2W^2Var\left(b_7\right) + 2ZCov\left(b_1, b_4\right) + 2WCov\left(b_1, b_5\right) + 2ZWCov\left(\right.$$

Wald statistic is used for test statistic.

Value

A list with two elements:

1. SimpleIntercept The intercepts given each value of the moderator. This element will be shown only if the factor intercept is estimated (e.g., not fixed as 0).
2. SimpleSlope The slopes given each value of the moderator.

In each element, the first column represents the values of the first moderator specified in the `valProbe1` argument. The second column represents the 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. The sixth column is the p-value testing whether the simple intercepts or slopes are different from 0.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

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*, 275-300.

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

```
library(lavaan)

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, meanstructure=TRUE, std.lv=FALSE)
summary(fitMC3way)

result3wayMC <- probe3WayMC(fitMC3way, c("f1", "f2", "f3", "f12", "f13", "f23", "f123"),
"f4", c("f1", "f2"), c(-1, 0, 1), c(-1, 0, 1))
result3wayMC
```

`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 (Pornprasertmanit, Schoemann, Geldhof, & Little, submitted)

Usage

```
probe3WayRC(fit, nameX, nameY, modVar, valProbe1, valProbe2)
```

Arguments

`fit`	The lavaan model object used to evaluate model fit
`nameX`	The vector of the factor names used as the predictors. The three first-order factors will be listed first. Then the second-order factors will be listeed. The last element of the name will represent the three-way interaction. Note that the fourth element must be the interaction between the first and the second variables. The fifth element must be the interaction between the first and the third variables. The sixth element must be the interaction between the second and the third variables.

`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.

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, Pornprasertmanit, Schoemann, & Little, in press). 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 Pornprasertmanit, Schoemann, Geldhof, and Little (submitted) 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 intercepts given each value of the moderator. This element will be shown only if the factor intercept is estimated (e.g., not fixed as 0).
2. SimpleSlope The slopes given each value of the moderator.

In each element, the first column represents the values of the first moderator specified in the `valProbe1` argument. The second column represents the 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. The sixth column is the p-value testing whether the simple intercepts or slopes are different from 0.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

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

Lance, C. E. (1988). Residual centering, exploratory and confirmatory moderator analysis, and decomposition of effects in path models containing interactions. *Applied Psychological Measurement, 12*, 163-175.

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*, 497-519.

Marsh, H. W., Wen, Z., & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9*, 275-300.

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

```
library(lavaan)

dat3wayRC <- orthogonalize(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
"

fitRC3way <- sem(model3, data=dat3wayRC, meanstructure=TRUE, std.lv=FALSE)
summary(fitRC3way)

result3wayRC <- probe3WayRC(fitRC3way, c("f1", "f2", "f3", "f12", "f13", "f23", "f123"),
"f4", c("f1", "f2"), c(-1, 0, 1), c(-1, 0, 1))
result3wayRC
```

---

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. Be acknowledging the 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 fails. 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, incorporate the argument order=2 into the quark function in order to reorder the imputation method procedure. Otherwise, the order is defaulted to 1. Example to rerun quark after imputation failure, quark.list <- quark(data=yourdataframe,id=vectorofIDs,order=2).

`silent` If `FALSE`, the details of the `quark` process are printed.

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). The PCA code is copied and modified from the `FactoMineR` package. The function to print correlation matrix is copied from the `psych` package.

References

Howard, W. J., Little, T. D., & Rhemtulla, M. (in press). Using principal component analysis (PCA) to obtain auxiliary variables for missing data estimation in large data sets. *Multivariate Behavioral Research*.

See Also

`combinequark`

Examples

```
set.seed(123321)
library(lavaan)

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)
```

`reliability` *Calculate reliability values of factors*

Description

Calculate reliability values of factors by coefficient omega

Usage

```
reliability(object)
```

Arguments

`object` The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions.

Details

The coefficient alpha (Cronbach, 1951) can be calculated by

$$\alpha = \frac{k}{k-1}\left[1 - \frac{\sum^{k}_{i=1}\sigma_{ii}}{\sum^{k}_{i=1}\sigma_{ii} + 2\sum_{i<j}\sigma_{ij}}\right],$$

where k is the number of items in a factor, σ_{ii} is the item *i* observed variances, σ_{ij} is the observed covariance of items *i* and *j*.

The coefficient omega (Raykov, 2001) can be calculated by

$$\omega_1 = \frac{\left(\sum^{k}_{i=1}\lambda_i\right)^2 Var\left(\psi\right)}{\left(\sum^{k}_{i=1}\lambda_i\right)^2 Var\left(\psi\right) + \sum^{k}_{i=1}\theta_{ii} + 2\sum_{i<j}\theta_{ij}},$$

where λ_i is the factor loading of item *i*, ψ is the factor variance, θ_{ii} is the variance of measurement errors of item *i*, and θ_{ij} is the covariance of measurement errors from item *i* and *j*.

The second coefficient omega (Bentler, 1972, 2009) can be calculated by

$$\omega_2 = 1 - \frac{\mathbf{1}'\Theta\mathbf{1}}{\mathbf{1}'\hat{\Sigma}\mathbf{1}},$$

where Θ is the measurement error covariance matrix, $\hat{\Sigma}$ is the model-implied covariance matrix, and $\mathbf{1}$ is the k-dimensional vector of 1. The first and the second coefficients omega will have different values if there are dual loadings (or the existence of method factors). The first coefficient omega can be viewed as the reliability controlling for the other factors. The second coefficient omega can be viewed as the unconditional reliability.

The third coefficient omega (McDonald, 1999), which is sometimes referred to hierarchical omega, can be calculated by

$$\omega_3 = \frac{\left(\sum^{k}_{i=1}\lambda_i\right)^2 Var\left(\psi\right)}{\mathbf{1}'\Sigma\mathbf{1}},$$

where Σ is the observed covariance matrix. If the model fits the data well, the third coefficient omega will be similar to the other two. Note that if there is a directional effect in the model, all coefficients omega will use the total factor variances, which is calculated by the `impliedFactorCov` function.

In conclusion, ω_1, ω_2, and ω_3 are different in the denominator. The denominator of the first formula assumes that a model is congeneric factor model where measurement errors are not correlated. The second formula is accounted for correlated measurement errors. However, these two formulas assume that the model-implied covariance matrix explains item relationships perfectly. The residuals are subject to sampling error. The third formula use observed covariance matrix instead of model-implied covariance matrix to calculate the observed total variance. This formula is the most conservative method in calculating coefficient omega.

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.

Regarding to categorical items, coefficient alpha and AVE are calculated based on polychoric correlations. The coefficient alpha from this function may be not the same as the standard alpha calculation for categorical items. Researchers may check the `alpha` function in the `psych` package for the standard coefficient alpha calculation.

Item thresholds are not accounted for. Coefficient omega for categorical items, however, is calculated by accounting for both item covariances and item thresholds using Green and Yang's (2009, formula 21) approach. Three types of coefficient omega indicate different methods to calculate item total variances. The original formula from Green and Yang is equivalent to ω_3 in this function.

Value

Reliability values (coefficient alpha, coefficients omega, average variance extracted) of each factor in each group

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>); Yves Rosseel (Ghent University; <Yves.Rosseel@UGent.be>)

References

Bentler, P. M. (1972). A lower-bound method for the dimension-free measurement of internal consistency. *Social Science Research, 1*, 343-357.

Bentler, P. M. (2009). Alpha, dimension-free, and model-based internal consistency reliability. *Psychometrika, 74*, 137-143.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*, 297-334.

Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement errors. *Journal of Marketing Research, 18*, 39-50.

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

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

Raykov, T. (2001). Estimation of congeneric scale reliability using covariance structure analysis with nonlinear constraints *British Journal of Mathematical and Statistical Psychology, 54*, 315-323.

See Also

`reliabilityL2` for reliability value of a desired second-order factor, `maximalRelia` for the maximal reliability of weighted composite

Examples

```
library(lavaan)

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

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

`reliabilityL2` *Calculate the reliability values of a second-order factor*

Description

Calculate the reliability values (coefficient omega) of a second-order factor

Usage

```
reliabilityL2(object, secondFactor)
```

Arguments

`object`	The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions that has a second-order factor
`secondFactor`	The name of the second-order factor

Details

The first formula of the coefficient omega (in the `reliability`) will be mainly used in the calculation. The model-implied covariance matrix of a second-order factor model can be separated into three sources: the second-order factor, the uniqueness of the first-order factor, and the measurement error of indicators:

$$\hat{\Sigma} = \Lambda \boldsymbol{B} \Phi_2 \boldsymbol{B}' \Lambda' + \Lambda \Psi_u \Lambda' + \Theta,$$

where $\hat{\Sigma}$ is the model-implied covariance matrix, Λ is the first-order factor loading, $\boldsymbol{B}$ is the second-order factor loading, Φ_2 is the covariance matrix of the second-order factors, Ψ_u is the covariance matrix of the unique scores from first-order factors, and Θ is the covariance matrix of the measurement errors from indicators. Thus, the proportion of the second-order factor explaining the total score, or the coefficient omega at Level 1, can be calculated:

$$\omega_{L1} = \frac{\mathbf{1}'\Lambda \boldsymbol{B}\Phi_2\boldsymbol{B}'\Lambda'\mathbf{1}}{\mathbf{1}'\Lambda \boldsymbol{B}\Phi_2\boldsymbol{B}'\Lambda'\mathbf{1} + \mathbf{1}'\Lambda\Psi_u\Lambda'\mathbf{1} + \mathbf{1}'\Theta\mathbf{1}},$$

where $\mathbf{1}$ is the k-dimensional vector of 1 and k is the number of observed variables. When model-implied covariance matrix among first-order factors (Φ_1) can be calculated:

$$\Phi_1 = \boldsymbol{B}\Phi_2\boldsymbol{B}' + \Psi_u,$$

Thus, the proportion of the second-order factor explaining the varaince at first-order factor level, or the coefficient omega at Level 2, can be calculated:

$$\omega_{L2} = \frac{\mathbf{1}'_{\boldsymbol{F}}\boldsymbol{B}\Phi_2\boldsymbol{B}'\mathbf{1}_{\boldsymbol{F}}}{\mathbf{1}'_{\boldsymbol{F}}\boldsymbol{B}\Phi_2\boldsymbol{B}'\mathbf{1}_{\boldsymbol{F}} + \mathbf{1}'_{\boldsymbol{F}}\Psi_u\mathbf{1}_{\boldsymbol{F}}},$$

where $\mathbf{1}_{\boldsymbol{F}}$ is the F-dimensional vector of 1 and F is the number of first-order factors.

The partial coefficient omega at Level 1, or the proportion of observed variance explained by the second-order factor after partialling the uniqueness from the first-order factor, can be calculated:

$$\omega_{L1} = \frac{\mathbf{1}'\Lambda \boldsymbol{B}\Phi_2\boldsymbol{B}'\Lambda'\mathbf{1}}{\mathbf{1}'\Lambda \boldsymbol{B}\Phi_2\boldsymbol{B}'\Lambda'\mathbf{1} + \mathbf{1}'\Theta\mathbf{1}},$$

Note that if the second-order factor has a direct factor loading on some observed variables, the observed variables will be counted as first-order factors.

Value

Reliability values at Levels 1 and 2 of the second-order factor, as well as the partial reliability value at Level 1

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`reliability` for the reliability of the first-order factors.

Examples

```
library(lavaan)

HS.model3 <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9
  higher =~ visual + textual + speed'

fit6 <- cfa(HS.model3, data=HolzingerSwineford1939)
reliability(fit6) # Should provide a warning for the endogenous variable
reliabilityL2(fit6, "higher")
```

`residualCovariate` *Residual centered all target indicators by covariates*

Description

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

Usage

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

Arguments

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

Value

The data that the target variables replaced by the residuals

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Geldhof, G. J., Pornprasertmanit, S., Schoemann, A. M., & Little, T. D. (2013). Orthogonalizing through residual centering: Applications and caveats. *Educational and Psychological Measurement, 73*, 27-46.

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)
```

`rotate` *Implement orthogonal or oblique rotation*

Description

These functions will implement orthogonal or oblique rotation on standardized factor loadings from a lavaan output.

Usage

```
orthRotate(object, method="varimax", ...)
oblqRotate(object, method="quartimin", ...)
funRotate(object, fun, ...)
```

Arguments

`object`	A lavaan output
`method`	The method of rotations, such as `"varimax"`, `"quartimax"`, `"geomin"`, `"oblimin"`, or any gradient projection algorithms listed in the GPA function in the `GPArotation` package.
`fun`	The name of the function that users wish to rotate the standardized solution. The functions must take the first argument as the standardized loading matrix and return the `GPArotation` object. Check this page for available functions: `rotations`.
`...`	Additional arguments for the `GPForth` function (for `orthRotate`), the `GPFoblq` function (for `oblqRotate`), or the function that users provide in the `fun` argument.

Details

These functions will rotate the unrotated standardized factor loadings by orthogonal rotation using the `GPForth` function or oblique rotation using the `GPFoblq` function the `GPArotation` package. The resulting rotation matrix will be used to calculate standard errors of the rotated standardized factor loading by delta method by numerically computing the Jacobian matrix by the `lavJacobianD` function in the `lavaan` package.

Value

An `linkS4class{EFA}` object that saves the rotated EFA solution.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

```
library(lavaan)

unrotated <- efaUnrotate(HolzingerSwineford1939, nf=3, varList=paste0("x", 1:9), estimator="mlr")

# Orthogonal varimax
out.varimax <- orthRotate(unrotated, method="varimax")
summary(out.varimax, sort=FALSE, suppress=0.3)

# Orthogonal Quartimin
orthRotate(unrotated, method="quartimin")

# Oblique Quartimin
oblqRotate(unrotated, method="quartimin")

# Geomin
oblqRotate(unrotated, method="geomin")

## Not run:
# Target rotation
library(GPArotation)
target <- matrix(0, 9, 3)
target[1:3, 1] <- NA
target[4:6, 2] <- NA
target[7:9, 3] <- NA
colnames(target) <- c("factor1", "factor2", "factor3")
# This function works with GPArotation version 2012.3-1
funRotate(unrotated, fun="targetQ", Target=target)

## End(Not run)
```

---

runMI *Multiply impute and analyze data using lavaan*

Description

This function takes data with missing observations, multiple imputes the data, runs a SEM using lavaan and combines the results using Rubin's rules. Note that parameter estimates and standard errors are pooled by the Rubin's (1987) rule. The chi-square statistics and the related fit indices are pooled by the method described in `"chi"` argument. SRMR is calculated based on the average model-implied means and covariance matrices across imputations.

Usage

```
runMI(model, data, m, miArgs=list(), chi="all", miPackage="Amelia",
seed=12345, fun, nullModel = NULL, ...)
cfa.mi(model, data, m, miArgs=list(), miPackage="Amelia", chi="all",
seed=12345, nullModel = NULL, ...)
sem.mi(model, data, m, miArgs=list(), miPackage="Amelia", chi="all",
```

```
seed=12345, nullModel = NULL, ...)
growth.mi(model, data, m, miArgs=list(), miPackage="Amelia", chi="all",
seed=12345, nullModel = NULL, ...)
lavaan.mi(model, data, m, miArgs=list(), miPackage="Amelia", chi="all",
seed=12345, nullModel = NULL, ...)
```

Arguments

model	lavaan syntax for the model to be analyzed.
data	Data frame with missing observations or a list of data frames where each data frame is one imputed data set (for imputed data generated outside of the function). If a list of data frames is supplied, then other options can be left at the default.
m	Number of imputations wanted.
miArgs	Addition arguments for the multiple-imputation function. The arguments should be put in a list (see example below).
miPackage	Package to be used for imputation. Currently these functions only support "Amelia" or "mice" for imputation.
chi	The method to combine the chi-square. Can be one of the following: "mr" for the method proposed for Meng & Rubin (1992), "mplus" for the method used in Mplus (Asparouhov & Muthen, 2010), "lmrr" for the method proposed by Li, Meng, Raghunathan, & Rubin (1991), "all" to show the three methods in the output, and "none" to not pool any chi-square values. The default is "all".
seed	Random number seed to be used in imputations.
nullModel	lavaan syntax for the null model. If not specified, the default null model from lavaan is used.
fun	The character of the function name used in running lavaan model ("cfa", "sem", "growth", "lavaan").
...	Other arguments to be passed to the specified lavaan function ("cfa", "sem", "growth", "lavaan").

Value

The lavaanStar object which contains the original lavaan object (where the appropriate parameter estimates, appropriate standard errors, and chi-squares are filled), the additional fit-index values of the null model, which need to be adjusted to multiple datasets, and the information from pooling multiple results.

Author(s)

Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>) Patrick Miller (University of Notre Dame; <pmille13@nd.edu>) Sunthud Pornprasertmanit (<psunthud@gmail.com>) Mijke Rhemtulla (University of Amsterdam; <M.T.Rhemtulla@uva.nl>) Alexander Robitzsch (Federal Institute for Education Research, Innovation, and Development of the Austrian School System, Salzburg, Austria; <a.robitzsch@bifie.at>) Craig Enders (Arizona State University; <Craig.Enders@asu.edu>) Mauricio Garnier Villarreal (University of Kansas; <mgv@ku.edu>) Yves Rosseel (Ghent University; <Yves.Rosseel@UGent.be>)

References

Asparouhov T. & Muthen B. (2010).*Chi-Square Statistics with Multiple Imputation*. Technical Report. www.statmodel.com.

Li, K.H., Meng, X.-L., Raghunathan, T.E. and Rubin, D.B. (1991). Significance Levels From Repeated p-values with Multiply-Imputed Data. *Statistica Sinica, 1*, 65-92.

Meng, X.L. & Rubin, D.B. (1992). Performing likelihood ratio tests with multiply-imputed data sets. *Biometrika, 79*, 103 - 111.

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

Examples

```
library(lavaan)

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

HSMiss <- HolzingerSwineford1939[,paste("x", 1:9, sep="")]
randomMiss <- rbinom(prod(dim(HSMiss)), 1, 0.1)
randomMiss <- matrix(as.logical(randomMiss), nrow=nrow(HSMiss))
HSMiss[randomMiss] <- NA

out <- cfa.mi(HS.model, data=HSMiss, m = 3, chi="all")
summary(out)
inspect(out, "fit")
inspect(out, "impute")

## Not run:
##Multiple group example
HSMiss2 <- cbind(HSMiss, school = HolzingerSwineford1939[,"school"])
out2 <- cfa.mi(HS.model, data=HSMiss2, m = 3, miArgs=list(noms="school"), chi="MR", group="school")
summary(out2)
inspect(out2, "fit")
inspect(out2, "impute")

##Example using previously imputed data with runMI
library(Amelia)

modsim <- '
f1 =~ 0.7*y1+0.7*y2+0.7*y3
f2 =~ 0.7*y4+0.7*y5+0.7*y6
f3 =~ 0.7*y7+0.7*y8+0.7*y9'

mod <- '
f1 =~ y1+y2+y3
f2 =~ y4+y5+y6
f3 =~ y7+y8+y9'

datsim <- simulateData(modsim,model.type="cfa", meanstructure=TRUE,
std.lv=TRUE, sample.nobs=c(200,200))
randomMiss2 <- rbinom(prod(dim(datsim)), 1, 0.1)
```

```
randomMiss2 <- matrix(as.logical(randomMiss2), nrow=nrow(datsim))
datsim[randomMiss2] <- NA
datsimMI <- amelia(datsim,m=3, noms="group")

out3 <- runMI(mod, data=datsimMI$imputations, chi="LMRR", group="group", fun="cfa")
summary(out3)
inspect(out3, "fit")
inspect(out3, "impute")

# Categorical variables
popModel <- "
f1 =~ 0.6*y1 + 0.6*y2 + 0.6*y3 + 0.6*y4
y1 ~*~ 1*y1
y2 ~*~ 1*y2
y3 ~*~ 1*y3
y4 ~*~ 1*y4
f1 ~~ 1*f1
y1 | 0.5*t1
y2 | 0.25*t1
y3 | 0*t1
y4 | -0.5*t1
"
analyzeModel <- "
f1 =~ y1 + y2 + y3 + y4
y1 ~*~ 1*y1
y2 ~*~ 1*y2
y3 ~*~ 1*y3
y4 ~*~ 1*y4
"
dat <- simulateData(popModel, sample.nobs  = 200L)
miss.pat <- matrix(as.logical(rbinom(prod(dim(dat)), 1, 0.2)), nrow(dat), ncol(dat))
dat[miss.pat] <- NA
out5 <- cfa.mi(analyzeModel, data=dat, ordered=paste0("y", 1:4), m = 3,
miArgs=list(ords = c("y1", "y2", "y3", "y4")))
summary(out5)
inspect(out5, "fit")
inspect(out5, "impute")


## End(Not run)
```

saturateMx — *Analyzing data using a saturate model*

Description

Analyzing data using a saturate model by full-information maximum likelihood. In the saturate model, all means and covariances are free if items are continuous. For ordinal variables, their means are fixed as 0 and their variances are fixed as 1–their covariances and thresholds are estimated. In multiple-group model, all means are variances are separately estimated.

Usage

```
saturateMx(data, groupLab = NULL)
```

Arguments

data	The target data frame
groupLab	The name of grouping variable

Value

The MxModel object which contains the analysis result of the saturate model.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

nullMx, fitMeasuresMx, standardizeMx

Examples

```
## Not run:
library(OpenMx)
data(demoOneFactor)
satModel <- saturateMx(demoOneFactor)

## End(Not run)
```

---

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

```
data(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 was generated using the `simsem` package.

Examples

```
head(simParcel)
```

`singleParamTest` *Single Parameter Test Divided from Nested Model Comparison*

Description

In comparing two nested models, chi-square 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 lavTestLRT for available options

Details

This function first identify the differences between these two models. The model with more free parameters is referred to as parent model and the model with less free parameters is referred to as nested model. Three tests are implemented here:

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.
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, two parameters are equally 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. Chi-square and p-value are provided for all methods. Note that the chi-square is all based on 1 degree of freedom. 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 model comparison from the measurementInvariance function
HW.model <- ' visual =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed =~ x7 + x8 + x9 '

models <- measurementInvariance(HW.model, data=HolzingerSwineford1939, group="school")
singleParamTest(models[[1]], models[[2]])

# Note that the comparison between weak (Model 2) and scalar invariance (Model 3) cannot be done
# by this function # because the weak invariance model fixes factor means as 0 in Group 2 but
# the strong invariance model frees the factor means in Group 2. Users may try to compare
# strong (Model 3) and means invariance models by this function.
```

---

skew *Finding skewness*

Description

Finding skewness (g1) 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 is g1. The parameter skewness γ_2 formula is

$$\gamma_2 = \frac{\mu_3}{\mu_2^{3/2}},$$

where μ_i denotes the i order central moment.

The excessive kurtosis formula for sample statistic g_2 is

$$g_2 = \frac{k_3}{k_2^2},$$

where k_i are the i order k-statistic.

The standard error of the skewness is

$$Var(\hat{g}_2) = \frac{6}{N}$$

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.* Retrived from MathWorld–A Wolfram Web Resource http://mathworld.wolfram.com/Skewness.html

See Also

- `kurtosis` Find the univariate excessive kurtosis 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

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

`spatialCorrect` *Calculate reliability values of factors*

Description

Correcting sample sizes and standard errors in the presence of spatial autocorrelation in Structural Equation Models with with spatial structure in the autocorrelation of endogenous variables.

Usage

```
spatialCorrect(obj, xvar, yvar, alpha=0.05)
```

Arguments

`obj`	The lavaan model object provided after running the `cfa`, `sem`, `growth`, or `lavaan` functions.
`xvar`	X-coordinate.
`yvar`	Y-coordinate.
`alpha`	The alpha level used to decide whether the effective sample size is used. If Moran's I is not significant, the original sample size is used for test statistic. Otherwise, the calculated effective sample size is used.

Details

This function implements the standard error correction for all endogenous variables, using Moran's I and an approximation of an effective sample size for large sample sizes. This function does not support the data with missing observations.

Value

- *Morans_I* Moran's I statistics and effective sample sizes for all endogeneous variables.
- *parameters* Adjusted standard errors and test statistics for each parameter estimate for each endogeneous variables.

Author(s)

Jarrett Byrnes (University of Massachusetts, Boston; `<Jarrett.Byrnes@umb.edu>`)

Examples

```
library(lavaan)
borModel <- '
  NDVI ~ nTot + T61 + Wet
  nTot ~ T61
'
#note meanstructure = TRUE to obtain intercepts
borFit <- sem(borModel, data = boreal, meanstructure = TRUE)
spatialCorrect(borFit, boreal$x, boreal$y)
```

`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

dataL	List of output data sets. ONLY IF dataset is an R object and path is default. Otherwise, output will saved to hard drive with the same formatting 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

Determines power for model parameters using the Satorra & Sarris (1985) method

Usage

```
SSpower(popModel, n, powerModel, fun = "cfa", nparam = 1, alpha = .05, ...)
```

Arguments

`popModel`	lavaan syntax for the population model. This model should specify population values for all paramters in the model.
`n`	Sample size used in power calculation
`powerModel`	lavaan syntax for the model to be analyzed. This syntax should have the parameter(s) of interest fixed to 0 (or some other number).
`fun`	The character of the function name used in running lavaan model (`"cfa"`, `"sem"`, `"growth"`, `"lavaan"`).
`nparam`	The number of parameters one is constrained in `powerModel`.
`alpha`	The Type I error rate used to assess power
`...`	Other arguments to be passed to the specified lavaan function (`"cfa"`, `"sem"`, `"growth"`, `"lavaan"`).

Author(s)

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

References

Satorra, A., & Saris, W. E. (1985). Power of the likelihood ratio test in covariance structure analysis. *Psychometrika, 50*, 83-90.

Examples

```
library(lavaan)

#Specify population values. Note every paramter 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 model to be analyzed. Note parameter of interest f1~~f2 is fixed to 0.
modelA <- '
         f1 =~ V1 + V2 + V3 + V4
         f2 =~ V5 + V6 + V7 + V8

         f1 ~~ 0*f2


         '


SSpower(modelP, 150, modelA, std.lv=TRUE)

##Get power for a range of values

Ns <- seq(100, 500, 40)
powVals <- rep(NA, length(Ns))
for(i in 1:length(Ns)){
powVals[i] <- SSpower(modelP, Ns[i], modelA)
}
plot(Ns, powVals, type = 'l')
```

`standardizeMx` *Find standardized estimates for OpenMx output*

Description

Find standardized estimates for OpenMx output. This function is applicable for the `MxRAMObjective` only.

Usage

```
standardizeMx(object, free = TRUE)
```

Arguments

`object`	Target OpenMx output using `MxRAMObjective`
`free`	If `TRUE`, the function will show only standardized values of free parameters. If `FALSE`, the function will show the results for fixed and free parameters.

Value

A vector of standardized estimates

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`saturateMx`, `nullMx`, `fitMeasuresMx`

Examples

```
## Not run:
library(OpenMx)
data(myFADataRaw)
myFADataRaw <- myFADataRaw[,c("x1","x2","x3","x4","x5","x6")]
oneFactorModel <- mxModel("Common Factor Model Path Specification",
type="RAM",
mxData(
observed=myFADataRaw,
type="raw"
),
manifestVars=c("x1","x2","x3","x4","x5","x6"),
latentVars="F1",
mxPath(from=c("x1","x2","x3","x4","x5","x6"),
arrows=2,
free=TRUE,
values=c(1,1,1,1,1,1),
labels=c("e1","e2","e3","e4","e5","e6")
),
# residual variances
# -------------------------------------
mxPath(from="F1",
arrows=2,
free=TRUE,
```

```
values=1,
labels ="varF1"
),
# latent variance
# ------------------------------------
mxPath(from="F1",
to=c("x1","x2","x3","x4","x5","x6"),
arrows=1,
free=c(FALSE,TRUE,TRUE,TRUE,TRUE,TRUE),
values=c(1,1,1,1,1,1),
labels =c("l1","l2","l3","l4","l5","l6")
),
# factor loadings
# ------------------------------------
mxPath(from="one",
to=c("x1","x2","x3","x4","x5","x6","F1"),
arrows=1,
free=c(TRUE,TRUE,TRUE,TRUE,TRUE,TRUE,FALSE),
values=c(1,1,1,1,1,1,0),
labels =c("meanx1","meanx2","meanx3","meanx4","meanx5","meanx6",NA)
)
# means
# ------------------------------------
) # close model
# Create an MxModel object
# -----------------------------------------------------------------------------
oneFactorFit <- mxRun(oneFactorModel)
standardizeMx(oneFactorFit)

# Compare with lavaan
library(lavaan)
script <- "f1 =~ x1 + x2 + x3 + x4 + x5 + x6"
fit <- cfa(script, data=myFADataRaw, meanstructure=TRUE)
standardizedSolution(fit)

## End(Not run)
```

---

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, degrees of freedom 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)
```

`wald` *Calculate multivariate Wald statistics*

Description

Calculate multivariate Wald statistics based on linear combinations of model parameters

Usage

```
wald(object, syntax)
```

Arguments

`object`	An output from `lavaan`
`syntax`	Syntax that each line represents one linear constraint. A plus or minus sign is used to separate between each coefficient. An asterisk is used to separate between coefficients and parameters. The coefficient can have a forward slash to represent a division. The parameter names must be matched with the names of lavaan parameters investigated by running the `coef` function on a lavaan output. Lines can be separated by semi-colon. A pound sign is allowed for comments. Note that the defined parameters (created by ":=") do not work with this function.

Details

The formula for multivariate Wald test is

$$\chi^2 = \left(C\hat{b}\right)' \left[C\hat{V}C'\right]^{-1} \left(C\hat{b}\right),$$

where C is the contrast matrix, $\hat{b}$ is the estimated fixed effect, $\hat{V}$ is the asymptotic covariance matrix among fixed effects.

Value

Chi-square value with *p* value.

Author(s)

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

Examples

```
# Test the difference in factor loadings
library(lavaan)
HS.model <- ' visual  =~ x1 + con1*x2 + con1*x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + con2*x8 + con2*x9 '

fit <- cfa(HS.model, data=HolzingerSwineford1939)
wald(fit, "con2 - con1")

# Simultaneously test the difference in the influences
# of x1 and x2 on intercept and slope
model.syntax <- '
    i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
    s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4
    i ~ x1 + x2
    s ~ x1 + x2
    t1 ~ c1
    t2 ~ c2
    t3 ~ c3
    t4 ~ c4
'

fit2 <- growth(model.syntax, data=Demo.growth)
wald.syntax <- '
i~x1 - i~x2
1/2*s~x1 - 1/2*s~x2
'
wald(fit2, wald.syntax)

# Mplus example of MODEL TEST
model3 <- ' f1  =~ x1 + p2*x2 + p3*x3 + p4*x4 + p5*x5 + p6*x6
p4 == 2*p2'

fit3 <- cfa(model3, data=HolzingerSwineford1939)
wald(fit3, "p3; p6 - 0.5*p5")
```

Index