Package 'lavaan'

February 14, 2012

Title Latent Variable Analysis

Version 0.4-12

Author Yves Rosseel <Yves.Rosseel@UGent.be> with contributions from many people. See the lavaan website for a list of contributors.

Maintainer Yves Rosseel <Yves.Rosseel@UGent.be>

Description Fit a variety of latent variable models, including confirmatory factor analysis, structural equation modeling and latent growth curve models.

Depends methods, R(>= 2.10.1)

Imports stats4, stats, graphics

Suggests psych, qgraph, quadprog, boot

License GPL (>= 2)

LazyData yes

URL http://lavaan.org

Repository CRAN

Date/Publication 2012-02-02 16:16:05

R topics documented:

`bootstrapLavaan` *Bootstrapping a Lavaan Model*

Description

Bootstrap functions to get bootstrap standard errors and bootstrap test statistics

Usage

```
bootstrapLavaan(object, R = 1000L, type = "ordinary", verbose = FALSE,
                FUN = "coef", warn = -1L, return.boot = FALSE,
                parallel = c("no", "multicore", "snow"),
                ncpus = 1L, cl = NULL, ...)

bootstrapLRT(h0 = NULL, h1 = NULL, R = 1000L, type="bollen.stine",
             verbose=FALSE, return.LRT = FALSE,
             calibrate = FALSE, calibrate.R = 1000L, calibrate.alpha = 0.05,
             warn = -1L, parallel = c("no", "multicore", "snow"),
             ncpus = 1L, cl = NULL)
```

Arguments

`object`	An object of class `lavaan`.
`h0`	An object of class `lavaan`. The restricted model.
`h1`	An object of class `lavaan`. The unrestricted model.
`R`	Integer. The number of bootstrap draws.
`type`	If `"ordinary"` or `"nonparametric"`, the usual (naive) bootstrap method is used. If `"bollen.stine"`, the data is first transformed such that the null hypothesis holds exactly in the resampling space. If `"parametric"`, the parametric bootstrap approach is used. Currently, this is only valid for continuous data following a multivariate normal distribution.

FUN	A function which when applied to the lavaan object returns a vector containing the statistic(s) of interest. The default is FUN="coef", returning the estimated values of the free parameters in the model.
...	Other named arguments for FUN which are passed unchanged each time it is called.
verbose	If TRUE, show information for each bootstrap draw.
warn	Sets the handling of warning messages. See options.
return.boot	Not used for now.
return.LRT	If TRUE, return the LRT values as an attribute to the pvalue.
parallel	The type of parallel operation to be used (if any). If missing, the default is "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	An optional **parallel** or **snow** cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the bootstrapLavaan or bootstrapLRT call.
calibrate	If TRUE, a double (nested) bootstrap is used to compute an additional set of plugin-values for each bootstrap sample.
calibrate.R	Integer. The number of bootstrap draws to be use for the double bootstrap.
calibrate.alpha	The significance level to compute the adjusted alpha based on the plugin p-values.

Examples

```
# fit the Holzinger and Swineford (1939) example
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

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

# get the test statistic for the original sample
T.orig <- fitMeasures(fit, "chisq")

# bootstrap to get bootstrap test statistics
# we only generate 10 bootstrap sample in this example; in practice
# you may wish to use a much higher number
T.boot <- bootstrapLavaan(fit, R=10, type="bollen.stine",
                          FUN=fitMeasures, fit.measures="chisq")

# compute a bootstrap based p-value
pvalue.boot <- length(which(T.boot > T.orig))/length(T.boot)
```

cfa *Fit Confirmatory Factor Analysis Models*

Description

Fit a Confirmatory Factor Analysis (CFA) model.

Usage

```
cfa(model = NULL, meanstructure = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE, data = NULL, std.ov = FALSE,
    missing = "default", sample.cov = NULL, sample.mean = NULL,
    sample.nobs = NULL, group = NULL, group.equal = "",
    group.partial = "", constraints = '', estimator = "default",
    likelihood = "default", information = "default", se = "default",
    test = "default", bootstrap = 1000L,
    mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), start = "default",
    verbose = FALSE, warn = TRUE, debug = FALSE)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter list (eg. the output of the lavaanify() function) is also accepted.
meanstructure	If TRUE, the means of the observed variables enter the model. If "default", the value is set based on the user-specified model, and/or the values of other arguments.
fixed.x	If TRUE, the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If FALSE, they are considered random, and the means, variances and covariances are free parameters. If "default", the value is set depending on the mimic option.
orthogonal	If TRUE, the exogenous latent variables are assumed to be uncorrelated.
std.lv	If TRUE, the metric of each latent variable is determined by fixing their variances to 1.0. If FALSE, the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.
data	An optional data frame containing the observed variables used in the model.
std.ov	If TRUE, all observed variables are standardized before entering the analysis.
missing	If "listwise", cases with missing values are removed listwise from the data frame before analysis. If "direct" or "ml" or "fiml" and the estimator is maximum likelihood, Full Information Maximum Likelihood (FIML) estimation is used using all available data in the data frame. This is only valid if the data are missing completely at random (MCAR) or missing at random (MAR). If "default", the value is set depending on the estimator and the mimic option.

`sample.cov` Numeric matrix. A sample variance-covariance matrix. The rownames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group.

`sample.mean` A sample mean vector. For a multiple group analysis, a list with a mean vector for each group.

`sample.nobs` Number of observations if the full data frame is missing and only sample moments are given. For a multiple group analysis, a list or a vector with the number of observations for each group.

`group` A variable name in the data frame defining the groups in a multiple group analysis.

`group.equal` A vector of character strings. Only used in a multiple group analysis. Can be one or more of the following: `"loadings"`, `"intercepts"`, `"means"`, `"regressions"`, `"residuals"`, `"residual.covariances"`, `"lv.variances"` or `"lv.covariances"`, specifying the pattern of equality constraints across multiple groups.

`group.partial` A vector of character strings containing the labels of the parameters which should be free in all groups (thereby overriding the group.equal argument for some specific parameters).

`constraints` Additional (in)equality constraints not yet included in the model syntax. See `model.syntax` for more information.

`estimator` The estimator to be used. Can be one of the following: `"ML"` for maximum likelihood, `"GLS"` for generalized least squares, `"WLS"` for weighted least squares (sometimes called ADF estimation), `"MLM"` for maximum likelihood estimation with robust standard errors and a Satorra-Bentler scaled test statistic, `"MLF"` for maximum likelihood estimation with standard errors based on first-order derivatives and a conventional test statistic, `"MLR"` for maximum likelihood estimation with robust 'Huber-White' standard errors and a scaled test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Note that the `"MLM"`, `"MLF"` and `"MLR"` choices only affect the standard errors and the test statistic. They also imply `mimic="Mplus"`.

`likelihood` Only relevant for ML estimation. If `"wishart"`, the wishart likelihood approach is used. In this approach, the covariance matrix has been divided by N-1, and both standard errors and test statistics are based on N-1. If `"normal"`, the normal likelihood approach is used. Here, the covariance matrix has been divided by N, and both standard errors and test statistics are based on N. If `"default"`, it depends on the mimic option: if `mimic="Mplus"`, normal likelihood is used; otherwise, wishart likelihood is used.

`information` If `"expected"`, the expected information matrix is used (to compute the standard errors). If `"observed"`, the observed information matrix is used. If `"default"`, the value is set depending on the estimator and the mimic option.

`se` If `"standard"`, conventional standard errors are computed based on inverting the (expected or observed) information matrix. If `"first.order"`, standard errors are computed based on first-order derivatives. If `"robust.mlm"`, conventional robust standard errors are computed. If `"robust.mlr"`, standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If `"robust"`, either `"robust.mlm"` or `"robust.mlr"` is used depending on the estimator, the mimic option, and whether the data are complete or not. If `"boot"`

or "bootstrap", bootstrap standard errors are computed using standard bootstrapping (unless Bollen-Stine bootstrapping is requested for the test statistic; in this case bootstrap standard errors are computed using model-based bootstrapping). If "none", no standard errors are computed.

test — If "standard", a conventional chi-square test is computed. If "Satorra-Bentler", a Satorra-Bentler scaled test statistic is computed. If "Yuan-Bentler", a Yuan-Bentler scaled test statistic is computed. If "boot" or "bootstrap" or "bollen.stine", the Bollen-Stine bootstrap is used to compute the bootstrap probability value of the test statistic. If "default", the value depends on the values of other arguments.

bootstrap — Number of bootstrap draws, if bootstrapping is used.

mimic — If "Mplus", an attempt is made to mimic the Mplus program. If "EQS", an attempt is made to mimic the EQS program. If "default", the value is (currently) set to "Mplus".

representation — If "LISREL" the classical LISREL matrix representation is used to represent the model (using the all-y variant).

do.fit — If FALSE, the model is not fit, and the current starting values of the model parameters are preserved.

control — A list containing control parameters passed to the optimizer. By default, lavaan uses "nlminb". See the manpage of nlminb for an overview of the control parameters. A different optimizer can be chosen by setting the value of optim.method. For unconstrained optimization (the model syntax does not include any "==", ">" or "<" operators), the available options are "nlminb" (the default), "BFGS" and "L-BFGS-B". See the manpage of the optim function for the control parameters of the latter two options. For constrained optimization, the only available option is "nlminb.constr".

start — If it is a character string, the two options are currently "simple" and "Mplus". In the first case, all parameter values are set to zero, except the factor loadings (set to one), the variances of latent variables (set to 0.05), and the residual variances of observed variables (set to half the observed variance). If "Mplus", we use a similar scheme, but the factor loadings are estimated using the fabin3 estimator (tsls) per factor. If start is a fitted object of class lavaan, the estimated values of the corresponding parameters will be extracted. If it is a model list, for example the output of the paramaterEstimates() function, the values of the est or start or ustart column (whichever is found first) will be extracted.

verbose — If TRUE, the function value is printed out during each iteration.

warn — If TRUE, some (possibly harmless) warnings are printed out during the iterations.

debug — If TRUE, debugging information is printed out.

Details

The cfa function is a wrapper for the more general lavaan function, using the following default arguments: int.ov.free = TRUE, int.lv.free = FALSE, auto.fix.first = TRUE (unless std.lv = TRUE), auto.fix.single = TRUE, auto.var = TRUE, auto.cov.lv.x = TRUE, and auto.cov.y = TRUE.

Value

An object of class lavaan, for which several methods are available, including a summary method.

See Also

lavaan

Examples

```
## The famous Holzinger and Swineford (1939) example
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- cfa(HS.model, data=HolzingerSwineford1939)
summary(fit, fit.measures=TRUE)
```

Demo.growth *Demo dataset for a illustrating a linear growth model.*

Description

A toy dataset containing measures on 4 time points (t1,t2, t3 and t4), two predictors (x1 and x2) influencing the random intercept and slope, and a time-varying covariate (c1, c2, c3 and c4).

Usage

```
data(Demo.growth)
```

Format

A data frame of 400 observations of 10 variables.

- `t1` Measured value at time point 1
- `t2` Measured value at time point 2
- `t3` Measured value at time point 3
- `t4` Measured value at time point 4
- `x1` Predictor 1 influencing intercept and slope
- `x2` Predictor 2 influencing intercept and slope
- `c1` Time-varying covariate time point 1
- `c2` Time-varying covariate time point 2
- `c3` Time-varying covariate time point 3
- `c4` Time-varying covariate time point 4

See Also

growth

Examples

```
head(Demo.growth)
```

`fitMeasures` *Fit Measures for a Latent Variable Model*

Description

This function computes a variety of fit measures to assess the global fit of a latent variable model.

Usage

```
fitMeasures(object, fit.measures = "all")
fitmeasures(object, fit.measures = "all")
```

Arguments

`object`	An object of class lavaan.
`fit.measures`	If `"all"`, all fit measures available will be returned. If only a single or a few fit measures are specified by name, only those are computed and returned.

Value

An named numeric vector of fit measures.

Examples

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

fit <- cfa(HS.model, data=HolzingerSwineford1939)
fitMeasures(fit)
fitMeasures(fit, "cfi")
fitMeasures(fit, c("chisq", "df", "pvalue", "cfi", "rmsea"))
```

getCov *Utility Functions For Covariance Matrices*

Description

Convenience functions to deal with covariance and correlation matrices.

Usage

```
getCov(x, lower = TRUE, diagonal = TRUE, sds = NULL,
       names = paste("V", 1:nvar, sep=""))
char2num(s)
cor2cov(R, sds, names = NULL)
```

Arguments

x	The elements of the covariance matrix. Either inside a character string or as a numeric vector. In the former case, the function char2num is used to convert the numbers (inside the character string) to numeric values.
lower	Logical. If TRUE, the numeric values in x are the lower-triangular elements of the (symmetric) covariance matrix only. If FALSE, x contains the upper triangular elements only. Note we always assumed the elements are provided row-wise!
diagonal	Logical. If TRUE, the numeric values in x include the diagonal elements. If FALSE, a unit diagonal is assumed.
sds	A numeric vector containing the standard deviations to be used to scale the elements in x or the correlation matrix R into a covariance matrix.
names	The variable names of the observed variables.
s	Character string containing numeric values.
R	A correlation matrix, to be scaled into a covariance matrix.

Details

The getCov function is typically used to input the lower (or upper) triangular elements of a (symmetric) covariance matrix. In many examples found in handbooks, only those elements are shown. However, lavaan needs a full matrix to proceed.

The cor2cov function is the inverse of the cov2cor function, and scales a correlation matrix into a covariance matrix given the standard deviations of the variables. Optionally, variable names can be given.

Examples

```
# The classic Wheaton et. al. (1977) model
# panel data on he stability of alienation
lower <- '
 11.834,
  6.947,    9.364,
```

```
  6.819,    5.091,   12.532,
  4.783,    5.028,    7.495,    9.986,
 -3.839,   -3.889,   -3.841,   -3.625,   9.610,
-21.899,  -18.831,  -21.748,  -18.775,  35.522,  450.288 '

# convert to a full symmetric covariance matrix with names
wheaton.cov <- getCov(lower, names=c("anomia67","powerless67", "anomia71",
                                     "powerless71","education","sei"))

# the model
wheaton.model <- '
  # measurement model
    ses     =~ education + sei
    alien67 =~ anomia67 + powerless67
    alien71 =~ anomia71 + powerless71

  # equations
    alien71 ~ alien67 + ses
    alien67 ~ ses

  # correlated residuals
    anomia67 ~~ anomia71
    powerless67 ~~ powerless71
'

# fitting the model
fit <- sem(wheaton.model, sample.cov=wheaton.cov, sample.nobs=932)

# showing the results
summary(fit, standardized=TRUE)
```

---

growth *Fit Growth Curve Models*

Description

Fit a Growth Curve model.

Usage

```
growth(model = NULL, fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE, data = NULL, std.ov = FALSE,
    missing = "default", sample.cov = NULL, sample.mean = NULL,
    sample.nobs = NULL, group = NULL, group.equal = "",
    group.partial = "", constraints = '', estimator = "default",
    likelihood = "default", information = "default", se = "default",
    test = "default", bootstrap = 1000L,
    mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), start = "default",
    verbose = FALSE, warn = TRUE, debug = FALSE)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter list (eg. the output of the lavaanify() function) is also accepted.
fixed.x	If TRUE, the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If FALSE, they are considered random, and the means, variances and covariances are free parameters. If "default", the value is set depending on the mimic option.
orthogonal	If TRUE, the exogenous latent variables are assumed to be uncorrelated.
std.lv	If TRUE, the metric of each latent variable is determined by fixing their variances to 1.0. If FALSE, the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.
data	An optional data frame containing the observed variables used in the model.
std.ov	If TRUE, all observed variables are standardized before entering the analysis.
missing	If "listwise", cases with missing values are removed listwise from the data frame before analysis. If "direct" or "ml" or "fiml" and the estimator is maximum likelihood, Full Information Maximum Likelihood (FIML) estimation is used using all available data in the data frame. This is only valid if the data are missing completely at random (MCAR) or missing at random (MAR). If "default", the value is set depending on the estimator and the mimic option.
sample.cov	Numeric matrix. A sample variance-covariance matrix. The rownames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group.
sample.mean	A sample mean vector. For a multiple group analysis, a list with a mean vector for each group.
sample.nobs	Number of observations if the full data frame is missing and only sample moments are given. For a multiple group analysis, a list or a vector with the number of observations for each group.
group	A variable name in the data frame defining the groups in a multiple group analysis.
group.equal	A vector of character strings. Only used in a multiple group analysis. Can be one or more of the following: "loadings", "intercepts", "means", "regressions", "residuals", "residual.covariances", "lv.variances" or "lv.covariances", specifying the pattern of equality constraints across multiple groups.
group.partial	A vector of character strings containing the labels of the parameters which should be free in all groups (thereby overriding the group.equal argument for some specific parameters).
constraints	Additional (in)equality constraints not yet included in the model syntax. See model.syntax for more information.
estimator	The estimator to be used. Can be one of the following: "ML" for maximum likelihood, "GLS" for generalized least squares, "WLS" for weighted least squares (sometimes called ADF estimation), "MLM" for maximum likelihood estimation

with robust standard errors and a Satorra-Bentler scaled test statistic, "MLF" for maximum likelihood estimation with standard errors based on first-order derivatives and a conventional test statistic, "MLR" for maximum likelihood estimation with robust 'Huber-White' standard errors and a scaled test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Note that the "MLM", "MLF" and "MLR" choices only affect the standard errors and the test statistic. They also imply mimic="Mplus".

likelihood — Only relevant for ML estimation. If "wishart", the wishart likelihood approach is used. In this approach, the covariance matrix has been divided by N-1, and both standard errors and test statistics are based on N-1. If "normal", the normal likelihood approach is used. Here, the covariance matrix has been divided by N, and both standard errors and test statistics are based on N. If "default", it depends on the mimic option: if mimic="Mplus", normal likelihood is used; otherwise, wishart likelihood is used.

information — If "expected", the expected information matrix is used (to compute the standard errors). If "observed", the observed information matrix is used. If "default", the value is set depending on the estimator and the mimic option.

se — If "standard", conventional standard errors are computed based on inverting the (expected or observed) information matrix. If "first.order", standard errors are computed based on first-order derivatives. If "robust.mlm", conventional robust standard errors are computed. If "robust.mlr", standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If "robust", either "robust.mlm" or "robust.mlr" is used depending on the estimator, the mimic option, and whether the data are complete or not. If "boot" or "bootstrap", bootstrap standard errors are computed using standard bootstrapping (unless Bollen-Stine bootstrapping is requested for the test statistic; in this case bootstrap standard errors are computed using model-based bootstrapping). If "none", no standard errors are computed.

test — If "standard", a conventional chi-square test is computed. If "Satorra-Bentler", a Satorra-Bentler scaled test statistic is computed. If "Yuan-Bentler", a Yuan-Bentler scaled test statistic is computed. If "boot" or "bootstrap" or "bollen.stine", the Bollen-Stine bootstrap is used to compute the bootstrap probability value of the test statistic. If "default", the value depends on the values of other arguments.

bootstrap — Number of bootstrap draws, if bootstrapping is used.

mimic — If "Mplus", an attempt is made to mimic the Mplus program. If "EQS", an attempt is made to mimic the EQS program. If "default", the value is (currently) set to "Mplus".

representation — If "LISREL" the classical LISREL matrix representation is used to represent the model (using the all-y variant).

start — If it is a character string, the two options are currently "simple" and "Mplus". In the first case, all parameter values are set to zero, except the factor loadings (set to one), the variances of latent variables (set to 0.05), and the residual variances of observed variables (set to half the observed variance). If "Mplus", we use a similar scheme, but the factor loadings are estimated using the fabin3 estimator (tsls) per factor. If start is a fitted object of class lavaan, the estimated

values of the corresponding parameters will be extracted. If it is a model list, for example the output of the paramaterEstimates() function, the values of the est or start or ustart column (whichever is found first) will be extracted.

do.fit — If FALSE, the model is not fit, and the current starting values of the model parameters are preserved.

control — A list containing control parameters passed to the optimizer. By default, lavaan uses "nlminb". See the manpage of nlminb for an overview of the control parameters. A different optimizer can be chosen by setting the value of optim.method. For unconstrained optimization (the model syntax does not include any "==", ">" or "<" operators), the available options are "nlminb" (the default), "BFGS" and "L-BFGS-B". See the manpage of the optim function for the control parameters of the latter two options. For constrained optimization, the only available option is "nlminb.constr".

verbose — If TRUE, the function value is printed out during each iteration.

warn — If TRUE, some (possibly harmless) warnings are printed out during the iterations.

debug — If TRUE, debugging information is printed out.

Details

The growth function is a wrapper for the more general lavaan function, using the following default arguments: meanstructure = TRUE, int.ov.free = FALSE, int.lv.free = TRUE, auto.fix.first = TRUE (unless std.lv = TRUE), auto.fix.single = TRUE, auto.var = TRUE, auto.cov.lv.x = TRUE, and auto.cov.y = TRUE.

Value

An object of class lavaan, for which several methods are available, including a summary method.

See Also

lavaan

Examples

```
## linear growth model with a time-varying covariate
model.syntax <- '
  # intercept and slope with fixed coefficients
    i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
    s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4

  # regressions
    i ~ x1 + x2
    s ~ x1 + x2

  # time-varying covariates
    t1 ~ c1
    t2 ~ c2
    t3 ~ c3
    t4 ~ c4
```

```
,

fit <- growth(model.syntax, data=Demo.growth)
summary(fit)
```

`HolzingerSwineford1939` *Holzinger and Swineford Dataset (9 Variables)*

Description

The classic Holzinger and Swineford (1939) dataset consists of mental ability test scores of seventh- and eighth-grade children from two different schools (Pasteur and Grant-White). In the original dataset (available in the `MBESS` package), there are scores for 26 tests. However, a smaller subset with 9 variables is more widely used in the literature (for example in Joreskog's 1969 paper, which also uses the 145 subjects from the Grant-White school only).

Usage

```
data(HolzingerSwineford1939)
```

Format

A data frame with 301 observations of 15 variables.

`id` Identifier

`sex` Gender

`ageyr` Age, year part

`agemo` Age, month part

`school` School (Pasteur or Grant-White)

`grade` Grade

`x1` Visual perception

`x2` Cubes

`x3` Lozenges

`x4` Paragraph comprehension

`x5` Sentence completion

`x6` Word meaning

`x7` Speeded addition

`x8` Speeded counting of dots

`x9` Speeded discrimination straight and curved capitals

Source

The dataset was retrieved from http://web.missouri.edu/~kolenikovs/stata/hs-cfa.dta and converted to an R dataset.

References

Holzinger, K., and Swineford, F. (1939). A study in factor analysis: The stability of a bifactor solution. Supplementary Educational Monograph, no. 48. Chicago: University of Chicago Press.

Joreskog, K. G. (1969). A general approach to confirmatory maximum likelihood factor analysis. *Psychometrika*, 34, 183-202.

See Also

cfa

Examples

```
head(HolzingerSwineford1939)
```

`InformativeTesting` *Testing Inequality Constrained Hypotheses in SEM*

Description

Testing inequality constrained Hypotheses in SEM

Usage

```
InformativeTesting(model = NULL, data, constraints = NULL, R = 1000L,
                   type = "bollen.stine", return.LRT = TRUE, calibrate = FALSE,
                   calibrate.R = 500L, calibrate.alpha = 0.05,
                   parallel = c("no", "multicore", "snow"), ncpus = 1L,
                   cl = NULL, verbose = FALSE, stoptest = NULL,
                   conclusion = FALSE, ...)
```

Arguments

`model`	Model syntax specifying the model. See `model.syntax` for more information.
`data`	The data frame containing the observed variables being used to fit the model.
`constraints`	The imposed inequality constraints on the model.
`R`	Integer; number of bootstrap draws. Default value is set to 1000.
`type`	If `"parametric"`, the parametric bootstrap is used. If `"bollen.stine"`, the semi-nonparametric Bollen-Stine boostrap is used. The default value is set to `"bollen.stine"`.
`return.LRT`	Logical; if `TRUE`, the function returns bootstrapped LRT-values.
`calibrate`	Logical; if `TRUE`, a double bootstrap is used to compute an additional set of plug-in p-values for each boostrap sample.
`calibrate.R`	Integer; number of double bootstrap draws. Only used if `calibrate=TRUE`. The default value is set to 500.

`calibrate.alpha`
: The significance level to compute the adjusted alpha based on the plugin p-values. Only used if `calibrate=TRUE`

`parallel`
: The type of parallel operation to be used (if any). If missing, the default is "no".

`ncpus`
: Integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.

`cl`
: An optional parallel or snow cluster for use if `parallel = "snow"`. If not supplied, a cluster on the local machine is created for the duration of the `InformativeTesting` call.

`verbose`
: Logical; if `TRUE`, information is shown at each bootstrap draw.

`stoptest`
: The `InformativeTesting()` stops when the plug-in p-value for the null-hypothesis for Type A is larger than the pre-specified value.

`conclusion`
: Logical; if `TRUE`, a conclusion in words is printed.

`...`
: Other named arguments from the lavaan package which are passed to the function. For example `"group"` for a multiple group model.

Value

An object of class InformativeTesting for which a summary and a plot method is available.

Author(s)

Leonard Vanbrabant <l.g.f.vanbrabant@uu.nl>

References

Van de Schoot, R., Hoijtink, H., & Dekovic, M. (2010). Testing inequality constrained hypotheses in SEM models. Structural Equation Modeling, 17, 443-463.

Van de Schoot, R., Strohmeier, D. (2011). Testing informative hypotheses in SEM increases power: An illustration contrasting classical. International Journal of Behavioral Development 35(2), 180-190.

Examples

```
#Multiple regression

model  <- '
          y1 ~ b1*x1 + b2*x2 + b3*x3 +
          start(1.8)*x1 + start(0.017)*x2 + start(-0.31)*x3

          y1~~start(5.7)*y1
          '

constraints  <-'
              b2 < b1
              b3 < b1
              '
```

```
#We only generate 5 bootstrap samples and 2 double bootstraps;
#in practice you may wish to use a much higher number.

example <- InformativeTesting(model = model, data = PoliticalDemocracy,
                              constraints = constraints, R = 5,
                              type = "bollen.stine",
                              calibrate = TRUE, calibrate.R = 2,
                              calibrate.alpha = 0.05)
example
summary(example)
#plot(example)
```

`inspectSampleCov` *Observed Variable Correlation Matrix from a Model and Data*

Description

The lavaan model syntax describes a latent variable model. Often, the user wants to see the covariance matrix generated by their model for diagnostic purposes. However, their data may have far more columns of information than what is contained in their model.

Usage

```
inspectSampleCov(model, data, ...)
```

Arguments

`model`	The model that will be fit by lavaan.
`data`	The data frame being used to fit the model.
`...`	Other arguments to `sem` for how to deal with multiple groups, missing values, etc.

Details

One must supply both a model, coded with proper `model.syntax` and a data frame from which a covariance matrix will be calculated. This function essentially calls `sem`, but doesn't fit the model, then uses `inspect` to get the sample covariance matrix and meanstructure.

See also

`sem`, `inspect`

Author(s)

Jarrett Byrnes

lavaan *Fit a Latent Variable Model*

Description

Fit a latent variable model.

Usage

```
lavaan(model = NULL, model.type = "sem", meanstructure = "default",
    int.ov.free = FALSE, int.lv.free = FALSE, fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE, auto.fix.first = FALSE,
    auto.fix.single = FALSE, auto.var = FALSE, auto.cov.lv.x = FALSE,
    auto.cov.y = FALSE, data = NULL, std.ov = FALSE, missing = "default",
    sample.cov = NULL, sample.mean = NULL, sample.nobs = NULL,
    group = NULL, group.equal = "", group.partial = "",
    constraints = '', estimator = "default", likelihood = "default",
    information = "default", se = "default",  test = "default",
    bootstrap = 1000L, mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), start = "default", slotOptions = NULL,
    slotUser = NULL, slotSample = NULL, slotData = NULL, slotModel = NULL,
    verbose = FALSE, warn = TRUE, debug = FALSE)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter list (eg. the output of the lavaanify() function) is also accepted.
model.type	Set the model type: possible values are "cfa", "sem" or "growth". This may affect how starting values are computed, and may be used to alter the terminology used in the summary output, or the layout of path diagrams that are based on a fitted lavaan object.
meanstructure	If TRUE, the means of the observed variables enter the model. If "default", the value is set based on the user-specified model, and/or the values of other arguments.
int.ov.free	If FALSE, the intercepts of the observed variables are fixed to zero.
int.lv.free	If FALSE, the intercepts of the latent variables are fixed to zero.
fixed.x	If TRUE, the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If FALSE, they are considered random, and the means, variances and covariances are free parameters. If "default", the value is set depending on the mimic option.
orthogonal	If TRUE, the exogenous latent variables are assumed to be uncorrelated.

`std.lv` If `TRUE`, the metric of each latent variable is determined by fixing their variances to 1.0. If `FALSE`, the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.

`auto.fix.first` If `TRUE`, the factor loading of the first indicator is set to 1.0 for every latent variable.

`auto.fix.single` If `TRUE`, the residual variance (if included) of an observed indicator is set to zero if it is the only indicator of a latent variable.

`auto.var` If `TRUE`, the residual variances and the variances of exogenous latent variables are included in the model and set free.

`auto.cov.lv.x` If `TRUE`, the covariances of exogenous latent variables are included in the model and set free.

`auto.cov.y` If `TRUE`, the covariances of dependent variables (both observed and latent) are included in the model and set free.

`data` An optional data frame containing the observed variables used in the model.

`std.ov` If `TRUE`, all observed variables are standardized before entering the analysis.

`missing` If `"listwise"`, cases with missing values are removed listwise from the data frame before analysis. If `"direct"` or `"ml"` or `"fiml"` and the estimator is maximum likelihood, Full Information Maximum Likelihood (FIML) estimation is used using all available data in the data frame. This is only valid if the data are missing completely at random (MCAR) or missing at random (MAR). If `"default"`, the value is set depending on the estimator and the mimic option.

`sample.cov` Numeric matrix. A sample variance-covariance matrix. The rownames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group.

`sample.mean` A sample mean vector. For a multiple group analysis, a list with a mean vector for each group.

`sample.nobs` Number of observations if the full data frame is missing and only sample moments are given. For a multiple group analysis, a list or a vector with the number of observations for each group.

`group` A variable name in the data frame defining the groups in a multiple group analysis.

`group.equal` A vector of character strings. Only used in a multiple group analysis. Can be one or more of the following: `"loadings"`, `"intercepts"`, `"means"`, `"regressions"`, `"residuals"`, `"residual.covariances"`, `"lv.variances"` or `"lv.covariances"`, specifying the pattern of equality constraints across multiple groups.

`group.partial` A vector of character strings containing the labels of the parameters which should be free in all groups (thereby overriding the group.equal argument for some specific parameters).

`constraints` Additional (in)equality constraints not yet included in the model syntax. See `model.syntax` for more information.

`estimator` The estimator to be used. Can be one of the following: `"ML"` for maximum likelihood, `"GLS"` for generalized least squares, `"WLS"` for weighted least squares (sometimes called ADF estimation), `"MLM"` for maximum likelihood estimation

with robust standard errors and a Satorra-Bentler scaled test statistic, "MLF" for maximum likelihood estimation with standard errors based on first-order derivatives and a conventional test statistic, "MLR" for maximum likelihood estimation with robust 'Huber-White' standard errors and a scaled test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Note that the "MLM", "MLF" and "MLR" choices only affect the standard errors and the test statistic. They also imply mimic="Mplus".

likelihood — Only relevant for ML estimation. If "wishart", the wishart likelihood approach is used. In this approach, the covariance matrix has been divided by N-1, and both standard errors and test statistics are based on N-1. If "normal", the normal likelihood approach is used. Here, the covariance matrix has been divided by N, and both standard errors and test statistics are based on N. If "default", it depends on the mimic option: if mimic="Mplus", normal likelihood is used; otherwise, wishart likelihood is used.

information — If "expected", the expected information matrix is used (to compute the standard errors). If "observed", the observed information matrix is used. If "default", the value is set depending on the estimator and the mimic option.

se — If "standard", conventional standard errors are computed based on inverting the (expected or observed) information matrix. If "first.order", standard errors are computed based on first-order derivatives. If "robust.mlm", conventional robust standard errors are computed. If "robust.mlr", standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If "robust", either "robust.mlm" or "robust.mlr" is used depending on the estimator, the mimic option, and whether the data are complete or not. If "boot" or "bootstrap", bootstrap standard errors are computed using standard bootstrapping (unless Bollen-Stine bootstrapping is requested for the test statistic; in this case bootstrap standard errors are computed using model-based bootstrapping). If "none", no standard errors are computed.

test — If "standard", a conventional chi-square test is computed. If "Satorra-Bentler", a Satorra-Bentler scaled test statistic is computed. If "Yuan-Bentler", a Yuan-Bentler scaled test statistic is computed. If "boot" or "bootstrap" or "bollen.stine", the Bollen-Stine bootstrap is used to compute the bootstrap probability value of the test statistic. If "default", the value depends on the values of other arguments.

bootstrap — Number of bootstrap draws, if bootstrapping is used.

mimic — If "Mplus", an attempt is made to mimic the Mplus program. If "EQS", an attempt is made to mimic the EQS program. If "default", the value is (currently) set to "Mplus".

representation — If "LISREL" the classical LISREL matrix representation is used to represent the model (using the all-y variant).

do.fit — If FALSE, the model is not fit, and the current starting values of the model parameters are preserved.

control — A list containing control parameters passed to the optimizer. By default, lavaan uses "nlminb". See the manpage of nlminb for an overview of the control parameters. A different optimizer can be chosen by setting the value of optim.method. For unconstrained optimization (the model syntax does not include any "==",

	">" or "<" operators), the available options are "nlminb" (the default), "BFGS" and "L-BFGS-B". See the manpage of the optim function for the control parameters of the latter two options. For constrained optimization, the only available option is "nlminb.constr".
start	If it is a character string, the two options are currently "simple" and "Mplus". In the first case, all parameter values are set to zero, except the factor loadings (set to one), the variances of latent variables (set to 0.05), and the residual variances of observed variables (set to half the observed variance). If "Mplus", we use a similar scheme, but the factor loadings are estimated using the fabin3 estimator (tsls) per factor. If start is a fitted object of class lavaan, the estimated values of the corresponding parameters will be extracted. If it is a model list, for example the output of the paramaterEstimates() function, the values of the est or start or ustart column (whichever is found first) will be extracted.
slotOptions	Options slot from a fitted lavaan object. If provided, no new Options slot will be created by this call.
slotUser	User slot from a fitted lavaan object. If provided, no new User slot will be created by this call.
slotSample	Sample slot from a fitted lavaan object. If provided, no new Sample slot will be created by this call.
slotData	Data slot from a fitted lavaan object. If provided, no new Data slot will be created by this call.
slotModel	Model slot from a fitted lavaan object. If provided, no new Model slot will be created by this call.
verbose	If TRUE, the function value is printed out during each iteration.
warn	If TRUE, some (possibly harmless) warnings are printed out during the iterations.
debug	If TRUE, debugging information is printed out.

Value

An object of class lavaan, for which several methods are available, including a summary method.

See Also

cfa, sem, growth

Examples

```
# The Holzinger and Swineford (1939) example
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- lavaan(HS.model, data=HolzingerSwineford1939,
              auto.var=TRUE, auto.fix.first=TRUE,
              auto.cov.lv.x=TRUE)
summary(fit, fit.measures=TRUE)
```

lavaan-class *Class For Representing A (Fitted) Latent Variable Model*

Description

The lavaan class represents a (fitted) latent variable model. It contains a description of the model as specified by the user, a summary of the data, an internal matrix representation, and if the model was fitted, the fitting results.

Objects from the Class

Objects can be created via the cfa, sem, growth or lavaan functions.

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.

User: Named list describing the model as specified by the user. Can be coerced to a data.frame.

Data: A list containing the raw data (as a numeric matrix with only column names) per group; the matrix contains only the observed variables mentioned in the model.

Sample: Object of internal class "Sample": 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

Methods

coef signature(object = "lavaan"): Returns the estimates of the free parameters in the model as a named numeric vector

fitted.values signature(object = "lavaan"): Returns the implied moments of the model as a list with two elements (per group): cov for the implied covariance matrix, and mean for the implied mean vector. If only the covariance matrix was analyzed, the implied mean vector will be zero.

fitted signature(object = "lavaan"): an alias for fitted.values.

residuals signature(object = "lavaan", type="raw"): If type="raw", this function returns the raw (=unstandardized) difference between the implied moments and the observed moments as a list of two elements: cov for the residual covariance matrix, and mean for the residual mean vector. If only the covariance matrix was analyzed, the residual mean vector will be zero. If codetype="cor", the observed and model implied covariance matrix is first transformed to a correlation matrix (using cov2cor), before the residuals are computed. If type="normalized", the residuals are normalized. If type="standardized", the residuals are standardized. In the latter case, the residuals have a metric similar to z-values.

resid signature(object = "lavaan"): an alias for residuals

vcov `signature(object = "lavaan")`: returns the covariance matrix of the estimated parameters.

predict `signature(object = "lavaan")`: compute factor scores for all cases that are provided in the data frame.

anova `signature(object = "lavaan")`: returns model comparison statistics. See `anova`. At least two arguments (fitted models) are required.

update `signature(object = "lavaan", model.syntax, ..., evaluate=TRUE)`: update a fitted lavaan object and evaluate it (unless `evaluate=FALSE`). Note that we use the environment that is stored within the lavaan object, which is not necessarily the parent frame.

nobs `signature(object = "lavaan")`: returns the effective number of observations used when fitting the model. In a multiple group analysis, this is the sum of all observations per group.

logLik `signature(object = "lavaan")`: returns the log-likelihood of the fitted model, if maximum likelihood estimation was used. The `AIC` and `BIC` methods automatically work via `logLik()`.

inspect `signature(object = "lavaan", what = "free")`: This is the main 'extractor' function for `lavaan` objects. It allows the user to peek into the internal representation of the model. In addition, the requested information is returned (typically as a list) and can be used for further processing. The following values for `what` are allowed:

`"free"`: A list of model matrices counting the free parameters in the model, typically in the same order as they are specified by the user.

`"start"`: A list of model matrices containing the starting values for all parameters.

`"starting.values"`: An alias for `"start"`.

`"sampstat"`: The sample statistics used for the analysis.

`"se"`: A list of model matrices containing the estimated standard errors for all free parameters.

`"std.err"`: An alias for `"se"`.

`"standard.errors"`: An alias for `"se"`.

`"coef"`: A list of model matrices containing the current values of all parameters.

`"coefficients"`: An alias for `"coef"`.

`"parameters"`: An alias for `"coef"`.

`"parameter.estimates"`: An alias for `"coef"`.

`"parameter.values"`: An alias for `"coef"`.

`"estimates"`: An alias for `"coef"`.

`"est"`: An alias for `"coef"`.

`"x"`: An alias for `"coef"`.

`"std.coef"`: A data.frame containing both the raw and (completely) standardized parameter values.

`"std"`: An alias for `"std.coef"`.

`"standardized"`: An alias for `"std.coef"`.

`"standardizedsolution"`: An alias for `"std.coef"`.

`"standardized.solution"`: An alias for `"std.coef"`.

`"rsquare"`: A named vector with the R-Square value of the dependent observed and latent variables.

`"r-square"`: An alias for `"rsquare"`.

"r2": An alias for "rsquare".

"dx": A list of model matrices containing the derivatives of all parameters evaluated at the current (typically minimum) function value.

"gradient": An alias for "dx".

"derivatives": An alias for "dx".

"mi": A data.frame containing modification indices and expected parameter change (EPC) values, both in unstandardized and standardized metric.

"modindices": An alias for "mi".

"modification": An alias for "mi".

"modificationindices": An alias for "mi".

"modification.indices": An alias for "mi".

"converged": Returns TRUE if the optimization routine has converged; FALSE otherwise.

"list": A dataframe showing the internal representation of a lavaan model. Each row corresponds to a model parameter. The columns contain all the information that lavaan stores about these parameters (for example, if it is free of fixed, the user-specified starting values, etcetera).

show `signature(object = "lavaan")`: Print a short summary of the model fit

summary `signature(object = "lavaan", standardized=FALSE, fit.measures=FALSE, rsquare=FALSE, modindices=FALSE)`: Print a nice summary of the model estimates. If `standardized=TRUE`, the standardized solution is also printed. If `fit.measures=TRUE`, the chi-square statistic is supplemented by several fit measures. If `rsquare=TRUE`, the R-Square values for the dependent variables in the model are printed. If `modindices=TRUE`, modification indices are printed for all fixed parameters. Nothing is returned (use `inspect` or another extractor function to extract information from a fitted model).

See Also

cfa, sem, growth, fitMeasures, standardizedSolution, parameterEstimates, modindices

Examples

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

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

summary(fit, standardized=TRUE, fit.measures=TRUE, rsquare=TRUE)
inspect(fit, "free")
inspect(fit, "start")
inspect(fit, "rsquare")
inspect(fit, "fit")
fitted.values(fit)
coef(fit)
resid(fit, type="normalized")
```

`measurementInvariance` *Measurement Invariance Tests*

Description

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

Usage

```
measurementInvariance(..., strict = FALSE, quiet = FALSE)
```

Arguments

`...`	The same arguments as for any lavaan model. See `cfa` for more information.
`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.

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.

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

cfa

Examples

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

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

---

model.syntax *The Lavaan Model Syntax*

Description

The lavaan model syntax describes a latent variable model. The function `lavaanify` turns it into a list that represents the full model as specified by the user.

Usage

```
lavaanify(model.syntax = NULL, meanstructure = FALSE, int.ov.free = FALSE,
    int.lv.free = FALSE, orthogonal = FALSE, std.lv = FALSE, fixed.x = TRUE,
    constraints = NULL, auto = FALSE, model.type = "sem",
    auto.fix.first = FALSE, auto.fix.single = FALSE,  auto.var = FALSE,
    auto.cov.lv.x = FALSE, auto.cov.y = FALSE, ngroups = 1L, group.equal = NULL,
    group.partial = NULL, debug = FALSE, warn = TRUE, as.data.frame. = TRUE)

lavaanNames(object, type = "ov", group = NULL)
```

Arguments

`model.syntax`	The model syntax specifying the model. See details for more information.
`meanstructure`	If `TRUE`, intercepts/means will be added to the model both for both observed and latent variables.
`int.ov.free`	If `FALSE`, the intercepts of the observed variables are fixed to zero.
`int.lv.free`	If `FALSE`, the intercepts of the latent variables are fixed to zero.

orthogonal	If TRUE, the exogenous latent variables are assumed to be uncorrelated.
std.lv	If TRUE, the metric of each latent variable is determined by fixing their variances to 1.0. If FALSE, the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.
fixed.x	If TRUE, the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If FALSE, they are considered random, and the means, variances and covariances are free parameters.
constraints	Additional (in)equality constraints. See details for more information.
auto	If TRUE, the default values are used for the auto.* arguments, depending on the value of model.type.
model.type	Either "sem" or "growth"; only used if auto=TRUE.
auto.fix.first	If TRUE, the factor loading of the first indicator is set to 1.0 for every latent variable.
auto.fix.single	If TRUE, the residual variance (if included) of an observed indicator is set to zero if it is the only indicator of a latent variable.
auto.var	If TRUE, the residual variances and the variances of exogenous latent variables are included in the model and set free.
auto.cov.lv.x	If TRUE, the covariances of exogenous latent variables are included in the model and set free.
auto.cov.y	If TRUE, the covariances of dependent variables (both observed and latent) are included in the model and set free.
ngroups	The number of (independent) groups.
group.equal	A vector of character strings. Only used in a multiple group analysis. Can be one or more of the following: "loadings", "intercepts", "means", "regressions", "residuals" or "covariances", specifying the pattern of equality constraints across multiple groups.
group.partial	A vector of character strings containing the labels of the parameters which should be free in all groups (thereby overriding the group.equal argument for some specific parameters).
warn	If TRUE, some (possibly harmless) warnings are printed out during the iterations.
as.data.frame.	If TRUE, return the list of model parameters as a data.frame.
debug	If TRUE, debugging information is printed out.
object	Either a list containing the user-specificed model, as returned by lavaanify, or an object of class lavaan.
type	Only used in the function lavaanNames. If type contains "ov", only observed variable names are returned. If type contains "lv", only latent variable names are returned. The "ov.x" and "lv.x" types return exogenous variables only. The "ov.y" and "lv.y" types return dependent variables only (in the regression sense, excluding the indicators of latent variables). The "ov.nox" type returns all observed variables, except the exogenous ones.
group	Only used in the function lavaanNames. If NULL, all groups (if any) are used. If an integer (vector), only names from those groups are extracted. The group numbers are found in the group column of the parameter table.

Details

The model syntax consists of one or more formula-like expressions, each one describing a specific part of the model. The model syntax can be read from a file (using readLines), or can be specified as a literal string enclosed by single quotes as in the example below.

```
myModel <- '
  # latent variable definitions
    f1 =~ y1 + y2 + y3
    f2 =~ y4 + y5 + y6
    f3 =~ y7 + y8 +
          y9 + y10
    f4 =~ y11 + y12 + y13

  ! this is also a comment

  # regressions
    f1 ~ f3 + f4
    f2 ~ f4
    y1 + y2 ~ x1 + x2 + x3

  # (co)variances
    y1 ~~ y1
    y2 ~~ y4 + y5
    f1 ~~ f2

  # intercepts
    f1 ~ 1; y5 ~ 1
'
```

Blank lines and comments can be used in between the formulas, and formulas can be split over multiple lines. Both the sharp (#) and the exclamation (!) characters can be used to start a comment. Multiple formulas can be placed on a single line if they are separated by a semicolon (;).

There can be four types of formula-like expressions in the model syntax:

1. Latent variable definitions: The "=~" operator can be used to define (continuous) latent variables. The name of the latent variable is on the left of the "=~" operator, while the terms on the right, separated by "+" operators, are the indicators of the latent variable.
   The operator "=~" can be read as "is manifested by".
2. Regressions: The "~" operator specifies a regression. The dependent variable is on the left of a "~" operator and the independent variables, separated by "+" operators, are on the right. These regression formulas are similar to the way ordinary linear regression formulas are used in R, but they may include latent variables. Interaction terms are currently not supported.
3. Variance-covariances: The "~~" ('double tilde') operator specifies (residual) variances of an observed or latent variable, or a set of covariances between one variable, and several other variables (either observed or latent). Several variables, separated by "+" operators can appear on the right. This way, several pairwise (co)variances involving the same left-hand variable can be expressed in a single expression. The distinction between variances and residual variances is made automatically.

4. Intercepts: A special case of a regression formula can be used to specify an intercept (or a mean) of either an observed or a latent variable. The variable name is on the left of a "~" operator. On the right is only the number "1" representing the intercept. Including an intercept formula in the model automatically implies `meanstructure = TRUE`. The distinction between intercepts and means is made automatically.

Usually, only a single variable name appears on the left side of an operator. However, if multiple variable names are specified, separated by the "+" operator, the formula is repeated for each element on the left side (as for example in the third regression formula in the example above).

In the right-hand side of these formula-like expressions, each element can be multiplied (using the "*" operator) by either a numeric constant, an expression resulting in a numeric constant, or one of three special functions: `start()`, `label()` and `equal()`. This provides the user with a mechanism to fix parameters, to provide alternative starting values, to label the parameters, and to define equality constraints among model parameters. This is explained in more detail in the following sections.

Fixing parameters

It is often desirable to fix a model parameter that is otherwise (by default) free. Any parameter in a model can be fixed by using the pre-multiplication mechanism. Here are some examples:

- Fixing the regression coefficient of the predictor `x2`:

  ```
  y ~ x1 + 2.4*x2 + x3
  ```

- Specifying an orthogonal (zero) covariance between two latent variables:

  ```
  f1 ~~ 0*f2
  ```

- Specifying an intercept and a linear slope in a growth model:

  ```
  i =~ 1*y11 + 1*y12 + 1*y13 + 1*y14
  s =~ 0*y11 + 1*y12 + 2*y13 + 3*y14
  ```

Instead of a numeric constant, one can use a mathematical function that returns a numeric constant, for example `sqrt(10)`. Multiplying with `NA` will force the corresponding parameter to be free.

Starting values

User-provided starting values can be given by using the same pre-multiplication mechanism. But the numeric constant is now the argument of a special function `start()`. For example:

```
y ~ x1 + start(1.0)*x2 + x3
```

Note that if a starting value is provided, the parameter is not automatically considered to be free.

Parameter labels and equality constraints

Each free parameter in a model is automatically given a name (or label). The name given to a model parameter consists of three parts, coerced to a single character vector. The first part is the name of the variable in the left-hand side of the formula where the parameter was implied. The middle part is based on the special 'operator' used in the formula. This can be either one of "=~", "~" or "~~". The third part is the name of the variable in the right-hand side of the formula where the parameter

was implied, or "1" if it is an intercept. The three parts are pasted together in a single string. For example, the name of the fixed regression coefficient in the regression formula y ~ x1 + 2.4*x2 + x3 is the string "y~x2". The name of the parameter corresponding to the covariance between two latent variables in the formula f1 ~~ f2 is the string "f1~~f2".

Although this automatic labeling of parameters is convenient, the user may specify its own labels for specific parameters by using the label() modifier in a formula. For example, in the formula f1 =~ x1 + x2 + label("mylabel")*x3, the parameter corresponding with the factor loading of x3 will be named "mylabel" instead of the default name "f1=~x3".

To constrain a parameter to be equal to another target parameter, the pre-multiplication mechanism can be used using the special function equal(). The argument of equal() is the (automatic or user-specified) name of the target parameter. For example, in the confirmatory factor analysis example below, the intercepts of the three indicators of each latent variable are constrained to be equal to each other. For the first three, we have used the default names. For the last three, we have provided a custom label for the y2a intercept.

```
model <- '
  # two latent variables with fixed loadings
    f1 =~ 1*y1a + 1*y1b + 1*y1c
    f2 =~ 1*y2a + 1*y2b + 1*y2c

  # intercepts constrained to be equal
  # using the default names
    y1a ~ 1
    y1b ~ equal("y1a~1") * 1
    y1c ~ equal("y1a~1") * 1

  # intercepts constrained to be equal
  # using a custom label
    y2a ~ label("int2") * 1
    y2b ~ equal("int2") * 1
    y2c ~ equal("int2") * 1
'
```

Multiple groups

In a multiple group analysis, you can still use the pre-multiplication techniques, but the single argument is now replaced by a vector of arguments, one for each group. If it is a single value, the same value is used for all groups. For example, if there are two groups:

```
HS.model <- ' visual  =~            x1 +
                                 0.5*x2 +
                            c(0.6, 0.8)*x3

              textual =~                   x4 +
                           start(c(1.2, 0.6))*x5 +
                                        x6

              speed   =~                      x7 +
```

```
                                    x8 +
                label(c("x9.group1",
                        "x9.group2"))*x9 '
```

In this example, the factor loading of the 'x2' indicator is fixed to the value 0.5 for all groups. However, the factor loadings of the 'x3' indicator are fixed to 0.6 and 0.8 for group 1 and group 2 respectively. The same logic is used for all modifiers.

Multiple modifiers

In the model syntax, you can specify a variable more than once on the right hand side of an operator; therefore, several 'modifiers' can be applied simultaneously; for example, if you want to fix the value of a parameter and also label that parameter, you can use something like:

```
f1 =~ x1 + x2 + 4*x3 + label("x3.loading")*x3
```

---

modificationIndices *Modification Indices*

Description

Modification indices of a latent variable model.

Usage

```
modificationIndices(object, standardized = TRUE, power = FALSE,
                    delta = 0.1, alpha = 0.05, high.power = 0.75)
modindices(object, standardized = TRUE, power = FALSE,
                    delta = 0.1, alpha = 0.05, high.power = 0.75)
```

Arguments

object	An object of class lavaan.
standardized	If TRUE, two extra columns (sepc.lv and sepc.all) will contain standardized values for the epc's. In the first column (sepc.lv), standardizization is based on the variances of the (continuous) latent variables. In the second column (sepc.all), standardization is based on both the variances of both (continuous) observed and latent variables. (Residual) covariances are standardized using (residual) variances.
power	If TRUE, the (post-hoc) power is computed for each modification index, using the values of delta and alpha.
delta	The value of the effect size, as used in the post-hoc power computation, currently using the unstandardized metric of the epc column.
alpha	The significance level used for deciding if the modification index is statistically significant or not.
high.power	If the computed power is higher than this cutoff value, the power is considered 'high'. If not, the power is considered 'low'. This affects the values in the 'decision' column in the output.

Value

A data.frame containing modification indices and EPC's.

Examples

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

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

---

parameterEstimates *Parameter Estimates*

Description

Parameter estimates of a latent variable model.

Usage

```
parameterEstimates(object, ci = TRUE, level = 0.95,
                   boot.ci.type = "perc", standardized = FALSE)
```

Arguments

object	An object of class lavaan.
ci	If TRUE, confidence intervals are added to the output
level	The confidence level required.
boot.ci.type	If bootstrapping was used, the type of interval required. The value should be one of "norm", "basic", "perc", or "bca.simple". For the first three options, see the help page of the boot.ci function in the boot package. The "bca.simple" option produces intervals using the adjusted bootstrap percentile (BCa) method, but with no correction for acceleration (only for bias).
standardized	If TRUE, standardized estimates are added to the output

Value

A data.frame containing the estimated parameters, parameters, standard errors, z-values, and (by default) the lower and upper values of the confidence intervals. If requested, extra columns are added with standardized versions of the parameter estimates.

Examples

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

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

---

parameterTable *Parameter Table*

Description

Show the parameter table of a fitted model.

Usage

```
parameterTable(object)
parTable(object)
```

Arguments

object	An object of class lavaan.

Value

A data.frame containing the model parameters. This is simply the output of the lavaanify function coerced to a data.frame (with stringsAsFactors = FALSE).

See Also

lavaanify.

Examples

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

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

plot.InformativeTesting *Plot output InformativeTesting()*

Description

The function plots the distributions of bootstrapped LRT values and plug-in p-values.

Usage

```
  ## S3 method for class 'InformativeTesting'
plot(x, ...,
       type = c("all",
                "LRT.A","LRT.B",
                "plugin.pvalues.A",
                "plugin.pvalues.B"),
       main = "main title(s)", xlab = "xlabel",
       ylab = "ylabel", freq = TRUE, cex.main = 1,
       cex.lab = NULL, nclass = NULL, col = "grey",
       border = par("fg"), axes = TRUE, vline.LRT = FALSE,
       vline.col = "red", lty = NULL, lwd = NULL,
       legend = FALSE, cex.legend = 0.75,
       loc.legend = "topright")
```

Arguments

x	The output of the InformativeTesting() function
type	If "all", all available plots are plotted simultaneously. If LRT.A or LRT.B, a distribution of the bootstrapped LRT values is plotted. If plugin.pvalues.A or plugin.pvalues.B, a distribution of the boostrapped plug-in p-values is plotted.
main	The main title(s) for the plot(s)
xlab	A label for the x axis, default depends on input type.
ylab	A label for the y axis, default is "frequency".
freq	Logical; if TRUE, the histogram graphic is a representation of frequencies, the counts component of the result; if FALSE, probability densities, component density, are plotted (so that the histogram has a total area of one). The default is set to TRUE.
cex.main	The magnification to be used for main titles relative to the current setting of cex. The default is set to 1.
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex.
nclass	Integer; number of classes.
col	A colour to be used to fill the bars. The default of NULL yields unfilled bars.

`border`	Color for rectangle border(s). The default means par("fg").
`axes`	Logical; if `TRUE` the x and y axis are plotted.
`vline.LRT`	Logical; if `TRUE` a vertical line is drawn at the observed LRT value.
`vline.col`	Color for the vline.LRT. Default is set on `"red"`
`lty`	The line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses 'invisible lines' (i.e., does not draw them).
`lwd`	The line width, a positive number, defaulting to 1.
`legend`	Logical; if `TRUE` a legend is added to the plot with the observed LRT value and plug-in p-value.
`cex.legend`	A numerical value giving the amount by which the legend text and symbols should be magnified relative to the default. This starts as 1 when a device is opened, and is reset when the layout is changed
`...`	Further arguments to be passed to the plot function.
`loc.legend`	The location of the legend, specified by a single keyword from the list `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, `"topleft"`, `"top"`, `"topright"`, `"right"` and `"center"`.

`PoliticalDemocracy` *Industrialization And Political Democracy Dataset*

Description

The 'famous' Industrialization and Political Democracy dataset. This dataset is used throughout Bollen's 1989 book (see pages 12, 17, 36 in chapter 2, pages 228 and following in chapter 7, pages 321 and following in chapter 8). The dataset contains various measures of political democracy and industrialization in developing countries.

Usage

```
data(PoliticalDemocracy)
```

Format

A data frame of 75 observations of 11 variables.

`y1` Expert ratings of the freedom of the press in 1960

`y2` The freedom of political opposition in 1960

`y3` The fairness of elections in 1960

`y4` The effectiveness of the elected legislature in 1960

`y5` Expert ratings of the freedom of the press in 1965

`y6` The freedom of political opposition in 1965

`y7` The fairness of elections in 1965

`y8` The effectiveness of the elected legislature in 1965

`x1` The gross national product (GNP) per capita in 1960

`x2` The inanimate energy consumption per capita in 1960

`x3` The percentage of the labor force in industry in 1960

Source

The dataset was retrieved from http://web.missouri.edu/~kolenikovs/Stat9370/democindus.txt (see discussion on SEMNET 18 Jun 2009)

References

Bollen, K. A. (1989). *Structural Equations with Latent Variables.* Wiley Series in Probability and Mathematical Statistics. New York: Wiley.

Bollen, K. A. (1979). Political democracy and the timing of development. *American Sociological Review*, 44, 572-587.

Bollen, K. A. (1980). Issues in the comparative measurement of political democracy. *American Sociological Review*, 45, 370-390.

Examples

```
head(PoliticalDemocracy)
```

---

sem *Fit Structural Equation Models*

Description

Fit a Structural Equation Model (SEM).

Usage

```
sem(model = NULL, meanstructure = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE, data = NULL, std.ov = FALSE,
    missing = "default", sample.cov = NULL, sample.mean = NULL,
    sample.nobs = NULL, group = NULL, group.equal = "",
    group.partial = "", constraints = '', estimator = "default",
    likelihood = "default", information = "default", se = "default",
    test = "default", bootstrap = 1000L,
    mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), start = "default",
    verbose = FALSE, warn = TRUE, debug = FALSE)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter list (eg. the output of the lavaanify() function) is also accepted.
meanstructure	If TRUE, the means of the observed variables enter the model. If "default", the value is set based on the user-specified model, and/or the values of other arguments.
fixed.x	If TRUE, the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If FALSE, they are considered random, and the means, variances and covariances are free parameters. If "default", the value is set depending on the mimic option.
orthogonal	If TRUE, the exogenous latent variables are assumed to be uncorrelated.
std.lv	If TRUE, the metric of each latent variable is determined by fixing their variances to 1.0. If FALSE, the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.
data	An optional data frame containing the observed variables used in the model.
std.ov	If TRUE, all observed variables are standardized before entering the analysis.
missing	If "listwise", cases with missing values are removed listwise from the data frame before analysis. If "direct" or "ml" or "fiml" and the estimator is maximum likelihood, Full Information Maximum Likelihood (FIML) estimation is used using all available data in the data frame. This is only valid if the data are missing completely at random (MCAR) or missing at random (MAR). If "default", the value is set depending on the estimator and the mimic option.
sample.cov	Numeric matrix. A sample variance-covariance matrix. The rownames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group.
sample.mean	A sample mean vector. For a multiple group analysis, a list with a mean vector for each group.
sample.nobs	Number of observations if the full data frame is missing and only sample moments are given. For a multiple group analysis, a list or a vector with the number of observations for each group.
group	A variable name in the data frame defining the groups in a multiple group analysis.
group.equal	A vector of character strings. Only used in a multiple group analysis. Can be one or more of the following: "loadings", "intercepts", "means", "regressions", "residuals", "residual.covariances", "lv.variances" or "lv.covariances", specifying the pattern of equality constraints across multiple groups.
group.partial	A vector of character strings containing the labels of the parameters which should be free in all groups (thereby overriding the group.equal argument for some specific parameters).
constraints	Additional (in)equality constraints not yet included in the model syntax. See model.syntax for more information.

`estimator` The estimator to be used. Can be one of the following: `"ML"` for maximum likelihood, `"GLS"` for generalized least squares, `"WLS"` for weighted least squares (sometimes called ADF estimation), `"MLM"` for maximum likelihood estimation with robust standard errors and a Satorra-Bentler scaled test statistic, `"MLF"` for maximum likelihood estimation with standard errors based on first-order derivatives and a conventional test statistic, `"MLR"` for maximum likelihood estimation with robust 'Huber-White' standard errors and a scaled test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Note that the `"MLM"`, `"MLF"` and `"MLR"` choices only affect the standard errors and the test statistic. They also imply `mimic="Mplus"`.

`likelihood` Only relevant for ML estimation. If `"wishart"`, the wishart likelihood approach is used. In this approach, the covariance matrix has been divided by N-1, and both standard errors and test statistics are based on N-1. If `"normal"`, the normal likelihood approach is used. Here, the covariance matrix has been divided by N, and both standard errors and test statistics are based on N. If `"default"`, it depends on the mimic option: if `mimic="Mplus"`, normal likelihood is used; otherwise, wishart likelihood is used.

`information` If `"expected"`, the expected information matrix is used (to compute the standard errors). If `"observed"`, the observed information matrix is used. If `"default"`, the value is set depending on the estimator and the mimic option.

`se` If `"standard"`, conventional standard errors are computed based on inverting the (expected or observed) information matrix. If `"first.order"`, standard errors are computed based on first-order derivatives. If `"robust.mlm"`, conventional robust standard errors are computed. If `"robust.mlr"`, standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If `"robust"`, either `"robust.mlm"` or `"robust.mlr"` is used depending on the estimator, the mimic option, and whether the data are complete or not. If `"boot"` or `"bootstrap"`, bootstrap standard errors are computed using standard bootstrapping (unless Bollen-Stine bootstrapping is requested for the test statistic; in this case bootstrap standard errors are computed using model-based bootstrapping). If `"none"`, no standard errors are computed.

`test` If `"standard"`, a conventional chi-square test is computed. If `"Satorra-Bentler"`, a Satorra-Bentler scaled test statistic is computed. If `"Yuan-Bentler"`, a Yuan-Bentler scaled test statistic is computed. If `"boot"` or `"bootstrap"` or `"bollen.stine"`, the Bollen-Stine bootstrap is used to compute the bootstrap probability value of the test statistic. If `"default"`, the value depends on the values of other arguments.

`bootstrap` Number of bootstrap draws, if bootstrapping is used.

`mimic` If `"Mplus"`, an attempt is made to mimic the Mplus program. If `"EQS"`, an attempt is made to mimic the EQS program. If `"default"`, the value is (currently) set to `"Mplus"`.

`representation` If `"LISREL"` the classical LISREL matrix representation is used to represent the model (using the all-y variant).

`do.fit` If `FALSE`, the model is not fit, and the current starting values of the model parameters are preserved.

control — A list containing control parameters passed to the optimizer. By default, lavaan uses "nlminb". See the manpage of nlminb for an overview of the control parameters. A different optimizer can be chosen by setting the value of optim.method. For unconstrained optimization (the model syntax does not include any "==", ">" or "<" operators), the available options are "nlminb" (the default), "BFGS" and "L-BFGS-B". See the manpage of the optim function for the control parameters of the latter two options. For constrained optimization, the only available option is "nlminb.constr".

start — If it is a character string, the two options are currently "simple" and "Mplus". In the first case, all parameter values are set to zero, except the factor loadings (set to one), the variances of latent variables (set to 0.05), and the residual variances of observed variables (set to half the observed variance). If "Mplus", we use a similar scheme, but the factor loadings are estimated using the fabin3 estimator (tsls) per factor. If start is a fitted object of class lavaan, the estimated values of the corresponding parameters will be extracted. If it is a model list, for example the output of the paramaterEstimates() function, the values of the est or start or ustart column (whichever is found first) will be extracted.

verbose — If TRUE, the function value is printed out during each iteration.

warn — If TRUE, some (possibly harmless) warnings are printed out during the iterations.

debug — If TRUE, debugging information is printed out.

Details

The sem function is a wrapper for the more general lavaan function, using the following default arguments: int.ov.free = TRUE, int.lv.free = FALSE, auto.fix.first = TRUE (unless std.lv = TRUE), auto.fix.single = TRUE, auto.var = TRUE, auto.cov.lv.x = TRUE, and auto.cov.y = TRUE.

Value

An object of class lavaan, for which several methods are available, including a summary method.

See Also

lavaan

Examples

```
## The industrialization and Political Democracy Example
## Bollen (1989), page 332
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
'

fit <- sem(model, data=PoliticalDemocracy)
summary(fit, fit.measures=TRUE)
```

`simulateData` *Simulate Data From a Lavaan Model Syntax*

Description

Simulate data starting from a lavaan model syntax.

Usage

```
simulateData(model = NULL, model.type = "sem", meanstructure = "default",
             int.ov.free = TRUE, int.lv.free = FALSE, fixed.x = "default",
             orthogonal = FALSE, std.lv = FALSE, auto.fix.first = !std.lv,
             auto.fix.single = TRUE, auto.var = TRUE, auto.cov.lv.x = TRUE,
             auto.cov.y = TRUE, ..., sample.nobs = 500L,
             group.label = paste("G", 1:ngroups, sep = ""),
             skewness = NULL, kurtosis = NULL, seed = NULL,
             empirical = FALSE, return.type = "data.frame")
```

Arguments

`model`	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See `model.syntax` for more information. Alternatively, a parameter list (eg. the output of the `lavaanify()` function) is also accepted.
`model.type`	Set the model type: possible values are `"cfa"`, `"sem"` or `"growth"`. This may affect how starting values are computed, and may be used to alter the terminology used in the summary output, or the layout of path diagrams that are based on a fitted lavaan object.
`meanstructure`	If `TRUE`, the means of the observed variables enter the model. If `"default"`, the value is set based on the user-specified model, and/or the values of other arguments.
`int.ov.free`	If `FALSE`, the intercepts of the observed variables are fixed to zero.
`int.lv.free`	If `FALSE`, the intercepts of the latent variables are fixed to zero.

fixed.x	If TRUE, the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If FALSE, they are considered random, and the means, variances and covariances are free parameters. If "default", the value is set depending on the mimic option.
orthogonal	If TRUE, the exogenous latent variables are assumed to be uncorrelated.
std.lv	If TRUE, the metric of each latent variable is determined by fixing their variances to 1.0. If FALSE, the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.
auto.fix.first	If TRUE, the factor loading of the first indicator is set to 1.0 for every latent variable.
auto.fix.single	If TRUE, the residual variance (if included) of an observed indicator is set to zero if it is the only indicator of a latent variable.
auto.var	If TRUE, the residual variances and the variances of exogenous latent variables are included in the model and set free.
auto.cov.lv.x	If TRUE, the covariances of exogenous latent variables are included in the model and set free.
auto.cov.y	If TRUE, the covariances of dependent variables (both observed and latent) are included in the model and set free.
...	additional arguments passed to the lavaan function.
sample.nobs	Number of observations. If a vector, multiple datasets are created. If return.type = "matrix" or return.type = "cov", a list of length(sample.nobs) is returned, with either the data or covariance matrices, each one based on the number of observations as specified in sample.nobs. If return.type = "data.frame", all datasets are merged and a group variable is added to mimic a multiple group dataset.
group.label	The group labels that should be used if multiple groups are created.
skewness	Numeric vector. The skewness values for the observed variables. Defaults to zero.
kurtosis	Numeric vector. The kurtosis values for the observed variables. Defaults to zero.
seed	Set random seed.
empirical	Logical. If TRUE, the implied moments (Mu and Sigma) specify the empirical not population mean and covariance matrix.
return.type	If "data.frame", a data.frame is returned. If "matrix", a numeric matrix is returned (without any variable names). If "cov", a covariance matrix is returned (without any variable names).

Details

Model parameters can be specified by fixed values in the lavaan model syntax. If no fixed values are specified, the value zero will be assumed, except for factor loadings and variances, which are set to unity by default. By default, multivariate normal data are generated. However, by providing skewness and/or kurtosis values, nonnormal multivariate data can be generated, using the Vale & Maurelli (1983) method.

Value

The generated data. Either as a data.frame (if `return.type="data.frame"`, a numeric matrix (if `return.type="matrix"`, or a covariance matrix (if `return.type="cov"`.

Examples

```
# specify population model
population.model <- ' f1 =~ x1 + 0.8*x2 + 1.2*x3
                      f2 =~ x4 + 0.5*x5 + 1.5*x6
                      f3 =~ x7 + 0.1*x8 + 0.9*x9

                      f3 ~ 0.5*f1 + 0.6*f2

                      f3 ~~ 4*f3
                      x1 ~~ 10*x1
                      x2 ~~ 5*x2
                      x3 ~~ 4*x3
                    '

# generate data
myData <- simulateData(population.model, sample.nobs=1000L)

# population moments
fitted(sem(population.model))

# sample moments
round(cov(myData), 3)
round(colMeans(myData), 3)

# fit model
myModel <- ' f1 =~ x1 + x2 + x3
             f2 =~ x4 + x5 + x6
             f3 =~ x7 + x8 + x9
             f3 ~ f1 + f2 '
fit <- sem(myModel, data=myData)
summary(fit)
```

---

standardizedSolution *Standardized Solution*

Description

Standardized solution of a latent variable model.

Usage

```
standardizedSolution(object, type = "std.all")
```

Arguments

object	An object of class lavaan.
type	If "std.lv", the standardized estimates are on the variances of the (continuous) latent variables only. If "std.all", the standardized estimates are based on both the variances of both (continuous) observed and latent variables. If "std.nox", the standardized estimates are based on both the variances of both (continuous) observed and latent variables, but not the variances of exogenous covariates.

Value

A data.frame containing both unstandardized and standardized model parameters.

Examples

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

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

---

utils-matrix *Utility Functions For Matrices.*

Description

Utility functions to deal with (mostly symmetric) matrices.

Usage

```
vech(S, diagonal = TRUE)
vechr(S, diagonal = TRUE)
vechu(S, diagonal = TRUE)
vechru(S, diagonal = TRUE)
vech.reverse(x, diagonal = TRUE)
vechru.reverse(x, diagonal = TRUE)
vechr.reverse(x, diagonal = TRUE)
vechu.reverse(x, diagonal = TRUE)
lower2full(x, diagonal = TRUE)
upper2full(x, diagonal = TRUE)
duplicationMatrix(n = 1L)
commutationMatrix(m = 1L, n = 1L)
sqrtSymmetricMatrix(S)
```

Arguments

S	A symmetric matrix.
x	A numeric vector containing the lower triangular or upper triangular elements of a symmetric matrix, possibly including the diagonal elements.
diagonal	Logical. If TRUE, the diagonal is included. If FALSE, the diagonal is not included.
n	Integer. Dimension of the symmetric matrix, or column dimension of a non-square matrix.
m	Integer. Row dimension of a matrix.

Details

The vech function implements the vech operator (for 'half vectorization') and transforms a symmetric matrix into a vector by stacking the columns of the matrix one underneath the other, but eliminating all supradiagonal elements.

The vech.reverse function does the reverse: given the output of the vech function, it reconstructs the symmetric matrix.

The lower2full function takes the lower

The duplicationMatrix function creates a duplication matrix D: it duplicates the elements in vech(S) to create vec(S) (where S is symmetric), such that `D %*% vech(S) == vec(S)`.

The commutationMatrix function creates a commutation matrix (K): this mn x mx matrix is a permutation matrix which transforms vec(A) into vec(t(A)), such that `K %*% vec(A) == vec(t(A))`.

The sqrtSymmetricMatrix function computes the square root of a (positive definite) symmetric matrix.

References

Magnus, J. R. and H. Neudecker (1999). Matrix Differential Calculus with Applications in Statistics and Econometrics, Second Edition, John Wiley.

Examples

```
# lower.tri elements (including diagonal) of a symmetric matrix
x <- c(4,1,5,2,3,6)

# reconstruct full symmetric matrix (row-wise!)
S <- lower2full(x)

# extract the same lower.tri elements again in the same order
vechr(S)

# without diagonal elements
vechr(S, diagonal=FALSE)

# duplication matrix
nvar <- ncol(S)
vec <- as.vector
Dup <- duplicationMatrix(nvar)
```

```
Dup %*% vech(S) == vec(S) # should all be true

# commutation matrix
K <- commutationMatrix(nvar, nvar)
K %*% vec(S) == vec(t(S)) # should all be true

# take sqrt root of a symmetric matrix
S.sqrt <- sqrtSymmetricMatrix(S)
S.sqrt %*% S.sqrt
# should be equal to S again (ignoring some rounding-off errors)
```

Index