Package 'lavaan'

September 24, 2016

Title Latent Variable Analysis

Version 0.5-22

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

Depends R(>= 3.1.0)

Imports methods, stats4, stats, utils, graphics, MASS, mnormt, pbivnorm, quadprog

License GPL (>= 2)

LazyData yes

URL http://lavaan.org

NeedsCompilation no

Author Yves Rosseel [aut, cre],
Daniel Oberski [ctb],
Jarrett Byrnes [ctb],
Leonard Vanbrabant [ctb],
Victoria Savalei [ctb],
Ed Merkle [ctb],
Michael Hallquist [ctb],
Mijke Rhemtulla [ctb],
Myrsini Katsikatsou [ctb],
Mariska Barendse [ctb]

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

Repository CRAN

Date/Publication 2016-09-24 18:02:02

R topics documented:

bootstrapLavaan *Bootstrapping a Lavaan Model*

Description

Bootstrap the LRT, or any other statistic (or vector of statistics) you can extract from a fitted lavaan object.

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, iseed = NULL, h0.rmsea = NULL, ...)

bootstrapLRT(h0 = NULL, h1 = NULL, R = 1000L, type="bollen.stine",
             verbose = FALSE, return.LRT = FALSE, double.bootstrap = "no",
             double.bootstrap.R = 500L, double.bootstrap.alpha = 0.05,
             warn = -1L, parallel = c("no", "multicore", "snow"),
             ncpus = 1L, cl = NULL, iseed = 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 "yuan", the data is first transformed by combining data and theory (model), such that the resampling space is closer to the population space. If "parametric", the parametric bootstrap approach is used; currently, this is only valid for continuous data following a multivariate normal distribution. See references for more details.
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.

iseed An integer to set the seed. Or NULL if no reproducible seeds are needed. To make this work, make sure the first RNGkind() element is `"L'Ecuyer-CMRG"`. You can check this by typing `RNGkind()` in the console. You can set it by typing `RNGkind("L'Ecuyer-CMRG")`, before the boostrap functions are called.

h0.rmsea Only used if `type="yuan"`. Allows one to do the Yuan bootstrap under the hypothesis that the population RMSEA equals a specified value.

double.bootstrap
If `"standard"` the genuine double bootstrap is used to compute an additional set of plug-in p-values for each boostrap sample. If `"FDB"`, the fast double bootstrap is used to compute second level LRT-values for each bootstrap sample. If `"no"`, no double bootstrap is used. The default is set to `"FDB"`.

double.bootstrap.R
Integer. The number of bootstrap draws to be use for the double bootstrap.

double.bootstrap.alpha
The significance level to compute the adjusted alpha based on the plugin p-values.

Details

The FUN function can return either a scalar or a numeric vector. This function can be an existing function (for example `coef`) or can be a custom defined function. For example:

```
myFUN <- function(x) {
    # require(lavaan)
    modelImpliedCov <- fitted(x)$cov
    vech(modelImpliedCov)
}
```

If `parallel="snow"`, it is imperative that the `require(lavaan)` is included in the custom function.

Author(s)

Yves Rosseel and Leonard Vanbrabant. Ed Merkle contributed Yuan's bootstrap. Improvements to Yuan's bootstrap were contributed by Hao Wu and Chuchu Cheng.

References

Bollen, K. and Stine, R. (1992) Bootstrapping Goodness of Fit Measures in Structural Equation Models. Sociological Methods and Research, 21, 205–229.

Yuan, K.-H., Hayashi, K., & Yanagihara, H. (2007). A class of population covariance matrices in the bootstrap approach to covariance structure analysis. Multivariate Behavioral Research, 42, 261–281.

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, se="none")

# 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, data = NULL,
    meanstructure = "default",
    conditional.x = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE,
    parameterization = "default", std.ov = FALSE,
    missing = "default", ordered = NULL,
    sample.cov = NULL, sample.cov.rescale = "default",
    sample.mean = NULL, sample.nobs = NULL,
    ridge = 1e-05, group = NULL,
    group.label = NULL, group.equal = "", group.partial = "",
    group.w.free = FALSE, cluster = NULL, constraints = '',
    estimator = "default", likelihood = "default", link = "default",
    information = "default", se = "default", test = "default",
    bootstrap = 1000L, mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), WLS.V = NULL, NACOV = NULL,
    zero.add = "default", zero.keep.margins = "default",
    zero.cell.warn = TRUE, start = "default",
    check = c("start", "post"),
    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 table (eg. the output of the lavaanify() function) is also accepted.
data	An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
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.
conditional.x	If TRUE, we set up the model conditional on the exogenous 'x' covariates; the model-implied sample statistics only include the non-x variables. If FALSE, the exogenous 'x' variables are modeled jointly with the other variables, and the model-implied statistics refect both sets of variables. If "default", the value is set depending on the estimator, and whether or not the model involves categorical endogenous variables.
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 (residual) 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.
parameterization	Currently only used if data is categorical. If "delta", the delta parameterization is used. If "theta", the theta parameterization is used.
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.
ordered	Character vector. Only used if the data is in a data.frame. Treat these variables as ordered (ordinal) variables, if they are endogenous in the model. Importantly, all other variables will be treated as numeric (unless they are declared as ordered in the original data.frame.)
sample.cov	Numeric matrix. A sample variance-covariance matrix. The rownames and/or colnames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group. Note that if maximum likelihood estimation is used and likelihood="normal", the user provided covariance matrix is internally rescaled by multiplying it with a factor

(N-1)/N, to ensure that the covariance matrix has been divided by N. This can be turned off by setting the `sample.cov.rescale` argument to `FALSE`.

`sample.cov.rescale`
: If `TRUE`, the sample covariance matrix provided by the user is internally rescaled by multiplying it with a factor (N-1)/N. If `"default"`, the value is set depending on the estimator and the likelihood option: it is set to `TRUE` if maximum likelihood estimation is used and `likelihood="normal"`, and `FALSE` otherwise.

`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.

`ridge`
: Numeric. Small constant used for ridging. Only used if the sample covariance matrix is non positive definite.

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

`group.label`
: A character vector. The user can specify which group (or factor) levels need to be selected from the grouping variable, and in which order. If `NULL` (the default), all grouping levels are selected, in the order as they appear in the data.

`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"`, `"thresholds"`, `"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).

`group.w.free`
: Logical. If `TRUE`, the group frequencies are considered to be free parameters in the model. In this case, a Poisson model is fitted to estimate the group frequencies. If `FALSE` (the default), the group frequencies are fixed to their observed values.

`cluster`
: Not used yet.

`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), `"ULS"` for unweighted least squares and `"DWLS"` for diagonally weighted least squares. These are the main options that affect the estimation. For convenience, the `"ML"` option can be extended as `"MLM"`, `"MLMV"`, `"MLMVS"`, `"MLF"`, and `"MLR"`. The estimation will still be plain `"ML"`, but now with robust standard errors and a robust (scaled) test statistic. For `"MLM"`, `"MLMV"`, `"MLMVS"`, classic robust standard errors are used (`se="robust.sem"`); for `"MLF"`, standard errors are based on first-order derivatives (`se="first.order"`); for `"MLR"`, 'Huber-White' robust standard errors are used (`se="robust.huber.white"`). In addition, `"MLM"` will compute a Satorra-Bentler scaled (mean adjusted) test

statistic (test="satorra.bentler") , "MLMVS" will compute a mean and variance adjusted test statistic (Satterthwaite style) (test="mean.var.adjusted"), "MLMV" will compute a mean and variance adjusted test statistic (scaled and shifted) (test="scaled.shifted"), and "MLR" will compute a test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Analogously, the estimators "WLSM" and "WLSMV" imply the "DWLS" estimator (not the "WLS" estimator) with robust standard errors and a mean or mean and variance adjusted test statistic. Estimators "ULSM" and "ULSMV" imply the "ULS" estimator with robust standard errors and a mean or mean and variance adjusted test statistic.

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="lavaan" or mimic="Mplus", normal likelihood is used; otherwise, wishart likelihood is used.

link — Currently only used if estimator is MML. If "logit", a logit link is used for binary and ordered observed variables. If "probit", a probit link is used. If "default", it is currently set to "probit" (but this may change).

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.sem", conventional robust standard errors are computed. If "robust.huber.white", standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If "robust", either "robust.sem" or "robust.huber.white" 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 "mean.var.adjusted" or "Satterthwaite", a mean and variance adjusted test statistic is compute. If "scaled.shifted", an alternative mean and variance adjusted test statistic is computed (as in Mplus version 6 or higher). 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 "lavaan", which is very close 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".

WLS.V A user provided weight matrix to be used by estimator "WLS"; if the estimator is "DWLS", only the diagonal of this matrix will be used. For a multiple group analysis, a list with a weight matrix for each group. The elements of the weight matrix should be in the following order (if all data is continuous): first the means (if a meanstructure is involved), then the lower triangular elements of the covariance matrix including the diagonal, ordered column by column. In the categorical case: first the thresholds (including the means for continuous variables), then the slopes (if any), the variances of continuous variables (if any), and finally the lower triangular elements of the correlation/covariance matrix excluding the diagonal, ordered column by column.

NACOV A user provided matrix containing the elements of (N times) the asymptotic variance-covariance matrix of the sample statistics. For a multiple group analysis, a list with an asymptotic variance-covariance matrix for each group. See the WLS.V argument for information about the order of the elements.

zero.add A numeric vector containing two values. These values affect the calculation of polychoric correlations when some frequencies in the bivariate table are zero. The first value only applies for 2x2 tables. The second value for larger tables. This value is added to the zero frequency in the bivariate table. If "default", the value is set depending on the "mimic" option. By default, lavaan uses zero.add = c(0.5. 0.0).

zero.keep.margins Logical. This argument only affects the computation of polychoric correlations for 2x2 tables with an empty cell, and where a value is added to the empty cell. If TRUE, the other values of the frequency table are adjusted so that all margins are unaffected. If "default", the value is set depending on the "mimic". The default is TRUE.

zero.cell.warn Logical. Only used if some observed endogenous variables are categorical. If TRUE, give a warning if one or more cells of a bivariate frequency table are empty.

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.

`check` Character vector. If `check` includes `"start"`, the starting values are checked for possibly inconsistent values (for example values implying correlations larger than one); if `check` includes `"post"`, a check is performed after (post) fitting, to check if the solution is admissable.

`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`, `auto.th = TRUE`, `auto.delta = TRUE`, and `auto.cov.y = TRUE`.

Value

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

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. URL http://www.jstatsoft.org/v48/i02/.

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)
```

estfun *Extract Empirical Estimating Functions*

Description

A function for extracting the empirical estimating functions of a fitted lavaan model. This is the derivative of the objective function with respect to the parameter vector, evaluated at the observed (case-wise) data. In other words, this function returns the case-wise scores, evaluated at the fitted model parameters.

Usage

```
estfun.lavaan(object, scaling = FALSE, ignore.constraints = FALSE,
              remove.duplicated = TRUE, remove.empty.cases = TRUE)
lavScores(object, scaling = FALSE, ignore.constraints = FALSE,
          remove.duplicated = TRUE, remove.empty.cases = TRUE)
```

Arguments

`object`	An object of class `lavaan`.
`scaling`	If `TRUE`, the scores are scaled to reflect the specific objective function used by lavaan. If `FALSE` (the default), the objective function is the loglikelihood function assuming multivariate normality.
`ignore.constraints`	Logical. If `TRUE`, the scores do not reflect the (equality or inequality) constraints. If `FALSE`, the scores are computed by taking the unconstrained scores, and adding the term `t(R) lambda`, where `lambda` are the (case-wise) Lagrange Multipliers, and `R` is the Jacobian of the constraint function. Only in the latter case will the sum of the columns be (almost) equal to zero.
`remove.duplicated`	If `TRUE`, and all the equality constraints have a simple form (eg. a == b), the unconstrained scores are post-multiplied with a transformation matrix in order to remove the duplicated parameters.
`remove.empty.cases`	If `TRUE`, empty cases with only missing values will be removed from the output.

Value

A n x k matrix corresponding to n observations and k parameters.

Author(s)

Ed Merkle; the `remove.duplicated`, `ignore.constraints` and `remove.empty.cases` arguments were added by Yves Rosseel

FacialBurns *Dataset for illustrating the InformativeTesting function.*

Description

A dataset from the Dutch burn center (http://www.adbc.nl). The data were used to examine psychosocial functioning in patients with facial burn wounds. Psychosocial functioning was measured by Anxiety and depression symptoms (HADS), and self-esteem (Rosenberg's self-esteem scale).

Usage

```
data(FacialBurns)
```

Format

A data frame of 77 observations of 6 variables.

`Selfesteem` Rosenberg's self-esteem scale

`HADS` Anxiety and depression scale

`Age` Age measured in years, control variable

`TBSA` Total Burned Surface Area

`RUM` Rumination, control variable

`Sex` Gender, grouping variable

Examples

```
head(FacialBurns)
```

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", baseline.model = NULL)
fitmeasures(object, fit.measures = "all", baseline.model = NULL)
```

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.
baseline.model	If not NULL, an object of class lavaan, representing a user-specified baseline model. If a baseline model is provided, all fit indices relying on a baseline model (eg. CFI or TLI) will use the test statistics from this user-specified baseline model, instead of the default baseline model.

Value

A 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"))
```

---

fsr *Factor Score Regression*

Description

Fit a SEM model using factor score regression.

Usage

```
fsr(model = NULL, data = NULL, cmd = "sem",
    fsr.method = "Croon", fs.method = "Bartlett", ...)
```

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 table (eg. the output of the lavaanify() function) is also accepted.
data	An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
cmd	Charcater. Which command is used to run the sem models. The possible choices are "sem" or "lavaan", determining how we deal with default options.

`fsr.method`	Character. Factor score regression method. Possible options are `naive`, `Skrondal-Laake`, and `Croon`.
`fs.method`	Character. Factor score estimation method. Possible options are `Bartlett` and `regression`.
`...`	Further arguments that we pass to the `"cfa"`, `"sem"` or `"lavaan"` functions.

Details

The `fsr` function implements a two-step procedure to estimate the parameters of the structural (regression) part of a SEM model. In a first step, factor scores are computed for each latent variable. In a second step, the latent variables are replaced by the factor scores, and a path analysis is used to estimate all remaining model parameters. Special techniques are used in order to ensure (approximately) unbiased estimation of point estimates and standard errors.

Value

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

References

Devlieger, I., Mayer, A., & Rosseel, Y. (2015). Hypothesis Testing Using Factor Score Regression:
A Comparison of Four Methods. Educational and Psychological Measurement. http://epm.sagepub.com/content/early/2015/0

See Also

`lavaan`, `sem`, `lavPredict`

Examples

```
## The industrialization and Political Democracy Example
## Bollen (1989), page 332, simplified
model <- '
  # latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + y2 + y3 + y4

  # regressions
    dem60 ~ ind60
'

fit <- fsr(model, data = PoliticalDemocracy, fsr.method = "Skrondal-Laake")
summary(fit)
```

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; comma's and semi-colons are ignored.
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, data = NULL,
    conditional.x = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE,
```

```
    parameterization = "default", std.ov = FALSE,
    missing = "default", ordered = NULL,
    sample.cov = NULL, sample.cov.rescale = "default",
    sample.mean = NULL, sample.nobs = NULL,
    ridge = 1e-05, group = NULL,
    group.label = NULL, group.equal = "", group.partial = "",
    group.w.free = FALSE, cluster = NULL, constraints = '',
    estimator = "default", likelihood = "default", link = "default",
    information = "default", se = "default", test = "default",
    bootstrap = 1000L, mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), WLS.V = NULL, NACOV = NULL,
    zero.add = "default", zero.keep.margins = "default",
    zero.cell.warn = TRUE, start = "default",
    check = c("start", "post"),
    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 table (eg. the output of the lavaanify() function) is also accepted.
data	An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
conditional.x	If TRUE, we set up the model conditional on the exogenous 'x' covariates; the model-implied sample statistics only include the non-x variables. If FALSE, the exogenous 'x' variables are modeled jointly with the other variables, and the model-implied statistics refect both sets of variables. If "default", the value is set depending on the estimator, and whether or not the model involves categorical endogenous variables.
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 (residual) 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.
parameterization	Currently only used if data is categorical. If "delta", the delta parameterization is used. If "theta", the theta parameterization is used.
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.

ordered — Character vector. Only used if the data is in a data.frame. Treat these variables as ordered (ordinal) variables, if they are endogenous in the model. Importantly, all other variables will be treated as numeric (unless they are declared as ordered in the original data.frame.)

sample.cov — Numeric matrix. A sample variance-covariance matrix. The rownames and/or colnames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group. Note that if maximum likelihood estimation is used and likelihood="normal", the user provided covariance matrix is internally rescaled by multiplying it with a factor (N-1)/N, to ensure that the covariance matrix has been divided by N. This can be turned off by setting the sample.cov.rescale argument to FALSE.

sample.cov.rescale — If TRUE, the sample covariance matrix provided by the user is internally rescaled by multiplying it with a factor (N-1)/N. If "default", the value is set depending on the estimator and the likelihood option: it is set to TRUE if maximum likelihood estimation is used and likelihood="normal", and FALSE otherwise.

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.

ridge — Numeric. Small constant used for ridging. Only used if the sample covariance matrix is non positive definite.

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

group.label — A character vector. The user can specify which group (or factor) levels need to be selected from the grouping variable, and in which order. If NULL (the default), all grouping levels are selected, in the order as they appear in the data.

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","thresholds", "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).

group.w.free — Logical. If TRUE, the group frequencies are considered to be free parameters in the model. In this case, a Poisson model is fitted to estimate the group frequencies. If FALSE (the default), the group frequencies are fixed to their observed values.

cluster — Not used yet.

`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), `"ULS"` for unweighted least squares and `"DWLS"` for diagonally weighted least squares. These are the main options that affect the estimation. For convenience, the `"ML"` option can be extended as `"MLM"`, `"MLMV"`, `"MLMVS"`, `"MLF"`, and `"MLR"`. The estimation will still be plain `"ML"`, but now with robust standard errors and a robust (scaled) test statistic. For `"MLM"`, `"MLMV"`, `"MLMVS"`, classic robust standard errors are used (`se="robust.sem"`); for `"MLF"`, standard errors are based on first-order derivatives (`se="first.order"`); for `"MLR"`, 'Huber-White' robust standard errors are used (`se="robust.huber.white"`). In addition, `"MLM"` will compute a Satorra-Bentler scaled (mean adjusted) test statistic (`test="satorra.bentler"`) , `"MLMVS"` will compute a mean and variance adjusted test statistic (Satterthwaite style) (`test="mean.var.adjusted"`), `"MLMV"` will compute a mean and variance adjusted test statistic (scaled and shifted) (`test="scaled.shifted"`), and `"MLR"` will compute a test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Analogously, the estimators `"WLSM"` and `"WLSMV"` imply the `"DWLS"` estimator (not the `"WLS"` estimator) with robust standard errors and a mean or mean and variance adjusted test statistic. Estimators `"ULSM"` and `"ULSMV"` imply the `"ULS"` estimator with robust standard errors and a mean or mean and variance adjusted test statistic.

`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="lavaan"` or `mimic="Mplus"`, normal likelihood is used; otherwise, wishart likelihood is used.

`link` Currently only used if estimator is MML. If `"logit"`, a logit link is used for binary and ordered observed variables. If `"probit"`, a probit link is used. If `"default"`, it is currently set to `"probit"` (but this may change).

`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.sem"`, conventional robust standard errors are computed. If `"robust.huber.white"`, standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If `"robust"`, either `"robust.sem"` or `"robust.huber.white"` 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 "mean.var.adjusted" or "Satterthwaite", a mean and variance adjusted test statistic is compute. If "scaled.shifted", an alternative mean and variance adjusted test statistic is computed (as in Mplus version 6 or higher). 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 to "lavaan", which is very close to"Mplus".

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

WLS.V — A user provided weight matrix to be used by estimator "WLS"; if the estimator is "DWLS", only the diagonal of this matrix will be used. For a multiple group analysis, a list with a weight matrix for each group. The elements of the weight matrix should be in the following order (if all data is continuous): first the means (if a meanstructure is involved), then the lower triangular elements of the covariance matrix including the diagonal, ordered column by column. In the categorical case: first the thresholds (including the means for continuous variables), then the slopes (if any), the variances of continuous variables (if any), and finally the lower triangular elements of the correlation/covariance matrix excluding the diagonal, ordered column by column.

NACOV — A user provided matrix containing the elements of (N times) the asymptotic variance-covariance matrix of the sample statistics. For a multiple group analysis, a list with an asymptotic variance-covariance matrix for each group. See the WLS.V argument for information about the order of the elements.

zero.add — A numeric vector containing two values. These values affect the calculation of polychoric correlations when some frequencies in the bivariate table are zero. The first value only applies for 2x2 tables. The second value for larger tables. This value is added to the zero frequency in the bivariate table. If "default", the value is set depending on the "mimic" option. By default, lavaan uses zero.add = c(0.5. 0.0).

zero.keep.margins — Logical. This argument only affects the computation of polychoric correlations for 2x2 tables with an empty cell, and where a value is added to the empty cell. If TRUE, the other values of the frequency table are adjusted so that all margins are unaffected. If "default", the value is set depending on the "mimic". The default is TRUE.

zero.cell.warn — Logical. Only used if some observed endogenous variables are categorical. If TRUE, give a warning if one or more cells of a bivariate frequency table are empty.

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".

check — Character vector. If check includes "start", the starting values are checked for possibly inconsistent values (for example values implying correlations larger than one); if check includes "post", a check is performed after (post) fitting, to check if the solution is admissable.

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, auto.th = TRUE, auto.delta = TRUE, and auto.cov.y = TRUE.

Value

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

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. URL http://www.jstatsoft.org/v48/i02/.

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

This dataset was originally retrieved from http://web.missouri.edu/~kolenikovs/stata/hs-cfa.dta (link no longer active) 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 order/inequality Constrained Hypotheses in SEM*

Description

Testing order/inequality constrained Hypotheses in SEM

Usage

```
InformativeTesting(model = NULL, data, constraints = NULL,
                   R = 1000L, type = "bollen.stine",
                   return.LRT = TRUE,
                   double.bootstrap = "standard",
                   double.bootstrap.R = 249L,
                   double.bootstrap.alpha = 0.05,
                   parallel = c("no", "multicore", "snow"),
                   ncpus = 1L, cl = NULL, verbose = 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. The default value is set to 1000.
type	If "parametric", the parametric bootstrap is used. If "bollen.stine", the semi-nonparametric Bollen-Stine bootstrap is used. The default is set to "bollen.stine".
return.LRT	Logical; if TRUE, the function returns bootstrapped LRT-values.
double.bootstrap	If "standard" (default) the genuine double bootstrap is used to compute an additional set of plug-in p-values for each bootstrap sample. If "no", no double bootstrap is used. If "FDB", the fast double bootstrap is used to compute second level LRT-values for each bootstrap sample. Note that the "FDB" is experimental and should not be used by inexperienced users.
double.bootstrap.R	Integer; number of double bootstrap draws. The default value is set to 249.
double.bootstrap.alpha	The significance level to compute the adjusted alpha based on the plugin p-values. Only used if double.bootstrap = "standard". The default value is set to 0.05.
parallel	The type of parallel operation to be used (if any). If missing, the default is set "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.
...	Other named arguments from the lavaan package which are passed to the function. For example "group" in a multiple group model.

Details

The following hypothesis tests are available:

- Type A: Test H0: all restriktions with equalities ("=") active against HA: at least one inequality restriktion (">") strictly true.
- Type B: Test H0: all restriktions with inequalities (">") (including some equalities ("=")) active against HA: at least one restriktion false (some equality restriktions may be maintained).

Value

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

Author(s)

Leonard Vanbrabant <lgf.vanbrabant@gmail.com>

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**, 180-190.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York.

Examples

```
## Not run:
#########################
### real data example ###
#########################
# Multiple group path model for facial burns example.

# model syntax with starting values.
  burns.model <- 'Selfesteem ~ Age + c(m1, f1)*TBSA + HADS +
                             start(-.10, -.20)*TBSA
                 HADS ~ Age + c(m2, f2)*TBSA + RUM +
                        start(.10, .20)*TBSA '


# constraints syntax
 burns.constraints <- 'f2 > 0  ; m1 < 0
                       m2 > 0  ; f1 < 0
                       f2 > m2 ; f1 < m1'

# we only generate 2 bootstrap samples in this example; in practice
# you may wish to use a much higher number.
# the double bootstrap was switched off; in practice you probably
# want to set it to "standard".
example1 <- InformativeTesting(model = burns.model, data = FacialBurns,
                               R = 2, constraints = burns.constraints,
                               double.bootstrap = "no", group = "Sex")

example1

#########################
### artificial example ###
#########################
# Simple ANOVA model with 3 groups (N = 20 per group)
set.seed(1234)
Y <- cbind(c(rnorm(20,0,1), rnorm(20,0.5,1), rnorm(20,1,1)))
grp <- c(rep("1", 20), rep("2", 20), rep("3", 20))
Data <- data.frame(Y, grp)
```

```
#create model matrix
fit.lm <- lm(Y ~ grp, data = Data)
mfit <- fit.lm$model
mm <- model.matrix(mfit)

Y <- model.response(mfit)
X <- data.frame(mm[,2:3])
names(X) <- c("d1", "d2")
Data.new <- data.frame(Y, X)

# model
model <- 'Y ~ 1 + a1*d1 + a2*d2'

# fit without constraints
fit <- sem(model, data = Data.new)

# constraints syntax: mu1 < mu2 < mu3
constraints <- ' a1 > 0
                 a1 < a2 '

# we only generate 10 bootstrap samples in this example; in practice
# you may wish to use a much higher number, say > 1000. The double
# bootstrap is not necessary in case of an univariate ANOVA model.
example2 <- InformativeTesting(model = model, data = Data.new,
                               start = parTable(fit),
                               R = 10L, double.bootstrap = "no",
                               constraints = constraints)
example2

## End(Not run)
```

`InformativeTesting methods`

Methods for output InformativeTesting()

Description

The print function shows the results of hypothesis tests Type A and Type B. The plot function plots the distributions of bootstrapped LRT values and plug-in p-values.

Usage

```
## S3 method for class 'InformativeTesting'
print(x, digits = max(3, getOption("digits") - 3), ...)

## S3 method for class 'InformativeTesting'
plot(x, ..., type = c("lr","ppv"),
   main = "main", xlab = "xlabel", ylab = "Frequency", freq = TRUE,
```

```
breaks = 15, cex.main = 1, cex.lab = 1, cex.axis = 1,
col = "grey", border = par("fg"), vline = TRUE,
vline.col = c("red", "blue"), lty = c(1,2), lwd = 1,
legend = TRUE, bty = "o", cex.legend = 1, loc.legend = "topright")
```

Arguments

x	object of class "InformativeTesting".
digits	the number of significant digits to use when printing.
...	Currently not used.
type	If "lr", a distribution of the first-level bootstrapped LR values is plotted. If "ppv" a distribution of the bootstrapped 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.
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.
breaks	see hist
cex.main	The magnification to be used for main titles relative to the current setting of cex.
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex.
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
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").
vline	Logical; if TRUE a vertical line is drawn at the observed LRT value. If double.bootstrap = "FDB" a vertical line is drawn at the 1-p* quantile of the second-level LRT values, where p* is the first-level bootstrapped p-value
vline.col	Color(s) for the vline.LRT.
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.
bty	A character string which determined the type of box which is drawn about plots. If bty is one of "o" (the default), "l", "7", "c", "u", or "]" the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box.

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.

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".

Author(s)

Leonard Vanbrabant <lgf.vanbrabant@gmail.com>

Examples

```
## Not run:
#########################
### real data example ###
#########################
# Multiple group path model for facial burns example.

# model syntax with starting values.
  burns.model <- 'Selfesteem ~ Age + c(m1, f1)*TBSA + HADS +
                              start(-.10, -.20)*TBSA
                 HADS ~ Age + c(m2, f2)*TBSA + RUM +
                        start(.10, .20)*TBSA '


# constraints syntax
 burns.constraints <- 'f2 > 0  ; m1 < 0
                       m2 > 0  ; f1 < 0
                       f2 > m2 ; f1 < m1'

# we only generate 2 bootstrap samples in this example; in practice
# you may wish to use a much higher number.
# the double bootstrap was switched off; in practice you probably
# want to set it to "standard".
example1 <- InformativeTesting(model = burns.model, data = FacialBurns,
                               R = 2, constraints = burns.constraints,
                               double.bootstrap = "no", group = "Sex")
example1
plot(example1)

#########################
### artificial example ###
#########################
# Simple ANOVA model with 3 groups (N = 20 per group)
set.seed(1234)
Y <- cbind(c(rnorm(20,0,1), rnorm(20,0.5,1), rnorm(20,1,1)))
grp <- c(rep("1", 20), rep("2", 20), rep("3", 20))
Data <- data.frame(Y, grp)

#create model matrix
fit.lm <- lm(Y ~ grp, data = Data)
```

```
mfit <- fit.lm$model
mm <- model.matrix(mfit)

Y <- model.response(mfit)
X <- data.frame(mm[,2:3])
names(X) <- c("d1", "d2")
Data.new <- data.frame(Y, X)

# model
model <- 'Y ~ 1 + a1*d1 + a2*d2'

# fit without constraints
fit <- sem(model, data = Data.new)

# constraints syntax: mu1 < mu2 < mu3
constraints <- ' a1 > 0
                 a1 < a2 '

# we only generate 10 bootstrap samples in this example; in practice
# you may wish to use a much higher number, say > 1000. The double
# bootstrap is not necessary in case of an univariate ANOVA model.
example2 <- InformativeTesting(model = model, data = Data.new,
                               start = parTable(fit),
                               R = 10L, double.bootstrap = "no",
                               constraints = constraints)
example2
# plot(example2)

## End(Not run)
```

`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, data = NULL,
    model.type = "sem", meanstructure = "default",
    int.ov.free = FALSE, int.lv.free = FALSE,
    conditional.x = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE,
    parameterization = "default", auto.fix.first = FALSE,
    auto.fix.single = FALSE, auto.var = FALSE, auto.cov.lv.x = FALSE,
    auto.cov.y = FALSE, auto.th = FALSE, auto.delta = FALSE,
    std.ov = FALSE, missing = "default", ordered = NULL,
    sample.cov = NULL, sample.cov.rescale = "default",
    sample.mean = NULL, sample.nobs = NULL, ridge = 1e-05,
    group = NULL, group.label = NULL, group.equal = "", group.partial = "",
    group.w.free = FALSE, cluster = NULL,
    constraints = "", estimator = "default",
    likelihood = "default", link = "default", information = "default",
    se = "default", test = "default", bootstrap = 1000L, mimic = "default",
    representation = "default", do.fit = TRUE, control = list(),
    WLS.V = NULL, NACOV = NULL,
    zero.add = "default", zero.keep.margins = "default",
    zero.cell.warn = TRUE, start = "default",
    slotOptions = NULL, slotParTable = NULL,
    slotSampleStats = NULL, slotData = NULL, slotModel = NULL,
    slotCache = NULL, check = c("start", "post"),
    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 table (eg. the output of the lavaanify() function) is also accepted.
data	An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
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.
conditional.x	If TRUE, we set up the model conditional on the exogenous 'x' covariates; the model-implied sample statistics only include the non-x variables. If FALSE, the exogenous 'x' variables are modeled jointly with the other variables, and the model-implied statistics refect both sets of variables. If "default", the value is set depending on the estimator, and whether or not the model involves categorical endogenous variables.
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 (residual) 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.
parameterization	Currently only used if data is categorical. If "delta", the delta parameterization is used. If "theta", the theta parameterization is used.
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.

`auto.th` If `TRUE`, thresholds for limited dependent variables are included in the model and set free.

`auto.delta` If `TRUE`, response scaling parameters for limited dependent variables are included in the model and set free.

`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.

`ordered` Character vector. Only used if the data is in a data.frame. Treat these variables as ordered (ordinal) variables, if they are endogenous in the model. Importantly, all other variables will be treated as numeric (unless they are declared as ordered in the original data.frame.)

`sample.cov` Numeric matrix. A sample variance-covariance matrix. The rownames and/or colnames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group. Note that if maximum likelihood estimation is used and `likelihood="normal"`, the user provided covariance matrix is internally rescaled by multiplying it with a factor (N-1)/N, to ensure that the covariance matrix has been divided by N. This can be turned off by setting the `sample.cov.rescale` argument to `FALSE`.

`sample.cov.rescale` If `TRUE`, the sample covariance matrix provided by the user is internally rescaled by multiplying it with a factor (N-1)/N. If `"default"`, the value is set depending on the estimator and the likelihood option: it is set to `TRUE` if maximum likelihood estimation is used and `likelihood="normal"`, and `FALSE` otherwise.

`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.

`ridge` Numeric. Small constant used for ridging. Only used if the sample covariance matrix is non positive definite.

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

`group.label` A character vector. The user can specify which group (or factor) levels need to be selected from the grouping variable, and in which order. If missing, all grouping levels are selected, in the order as they appear in the data.

`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"`,`"thresholds"`, `"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).

group.w.free Logical. If TRUE, the group frequencies are considered to be free parameters in the model. In this case, a Poisson model is fitted to estimate the group frequencies. If FALSE (the default), the group frequencies are fixed to their observed values.

cluster Not used yet.

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), "ULS" for unweighted least squares and "DWLS" for diagonally weighted least squares. These are the main options that affect the estimation. For convenience, the "ML" option can be extended as "MLM", "MLMV", "MLMVS", "MLF", and "MLR". The estimation will still be plain "ML", but now with robust standard errors and a robust (scaled) test statistic. For "MLM", "MLMV", "MLMVS", classic robust standard errors are used (se="robust.sem"); for "MLF", standard errors are based on first-order derivatives (se="first.order"); for "MLR", 'Huber-White' robust standard errors are used (se="robust.huber.white"). In addition, "MLM" will compute a Satorra-Bentler scaled (mean adjusted) test statistic (test="satorra.bentler") , "MLMVS" will compute a mean and variance adjusted test statistic (Satterthwaite style) (test="mean.var.adjusted"), "MLMV" will compute a mean and variance adjusted test statistic (scaled and shifted) (test="scaled.shifted"), and "MLR" will compute a test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Analogously, the estimators "WLSM" and "WLSMV" imply the "DWLS" estimator (not the "WLS" estimator) with robust standard errors and a mean or mean and variance adjusted test statistic. Estimators "ULSM" and "ULSMV" imply the "ULS" estimator with robust standard errors and a mean or mean and variance adjusted test statistic.

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="lavaan" or mimic="Mplus", normal likelihood is used; otherwise, wishart likelihood is used.

link Currently only used if estimator is MML. If "logit", a logit link is used for binary and ordered observed variables. If "probit", a probit link is used. If "default", it is currently set to "probit" (but this may change).

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.sem", conventional robust standard errors are computed. If "robust.huber.white", standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If "robust", either "robust.sem" or "robust.huber.white" 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 "mean.var.adjusted" or "Satterthwaite", a mean and variance adjusted test statistic is compute. If "scaled.shifted", an alternative mean and variance adjusted test statistic is computed (as in Mplus version 6 or higher). 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 to "lavaan", which is very close 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".

WLS.V — A user provided weight matrix to be used by estimator "WLS"; if the estimator is "DWLS", only the diagonal of this matrix will be used. For a multiple group analysis, a list with a weight matrix for each group. The elements of the weight matrix should be in the following order (if all data is continuous): first the means (if a meanstructure is involved), then the lower triangular elements of the covariance matrix including the diagonal, ordered column by column. In the categorical case: first the thresholds (including the means for continuous variables), then the slopes (if any), the variances of continuous variables (if any), and finally the lower triangular elements of the correlation/covariance matrix excluding the diagonal, ordered column by column.

NACOV — A user provided matrix containing the elements of (N times) the asymptotic variance-covariance matrix of the sample statistics. For a multiple group analysis, a list with an asymptotic variance-covariance matrix for each group. See the WLS.V argument for information about the order of the elements.

`zero.add` A numeric vector containing two values. These values affect the calculation of polychoric correlations when some frequencies in the bivariate table are zero. The first value only applies for 2x2 tables. The second value for larger tables. This value is added to the zero frequency in the bivariate table. If `"default"`, the value is set depending on the `"mimic"` option. By default, lavaan uses `zero.add = c(0.5, 0.0)`.

`zero.keep.margins`
Logical. This argument only affects the computation of polychoric correlations for 2x2 tables with an empty cell, and where a value is added to the empty cell. If `TRUE`, the other values of the frequency table are adjusted so that all margins are unaffected. If `"default"`, the value is set depending on the `"mimic"`. The default is `TRUE`.

`zero.cell.warn` Logical. Only used if some observed endogenous variables are categorical. If `TRUE`, give a warning if one or more cells of a bivariate frequency table are empty.

`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.

`slotParTable` ParTable slot from a fitted lavaan object. If provided, no new ParTable slot will be created by this call.

`slotSampleStats`
SampleStats slot from a fitted lavaan object. If provided, no new SampleStats 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.

`slotCache` Cache slot from a fitted lavaan object. If provided, no new Cache slot will be created by this call.

`check` Character vector. If `check` includes `"start"`, the starting values are checked for possibly inconsistent values (for example values implying correlations larger than one); if `check` includes `"post"`, a check is performed after (post) fitting, to check if the solution is admissable.

`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.

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. URL http://www.jstatsoft.org/v48/i02/.

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.call().

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'.

pta: Named list containing parameter table attributes.

`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

`Cache`: List using objects that we try to compute only once, and reuse many times.

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

`boot`: List. Results and information about the bootstrap.

`optim`: List. Information about the optimization.

`implied`: List. Model implied statistics.

`vcov`: List. Information about the variance matrix (vcov) of the model parameters.

`test`: List. Different test statistics.

`external`: List. Empty slot to be used by add-on packages.

Methods

coef `signature(object = "lavaan", type = "free")`: Returns the estimates of the parameters in the model as a named numeric vector. If `type="free"`, only the free parameters are returned. If `type="user"`, all parameters listed in the parameter table are returned, including constrained and fixed parameters.

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 `type="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 divided by the square root of an asymptotic variance estimate of the corresponding sample statistic (the variance estimate depends on the choice for the `se` argument). If `type="standardized"`, the residuals are divided by the square root of the difference between an asymptotic variance estimate of the corresponding sample statistic and an asymptotic variance estimate of the corresponding model-implied statistic. In the latter case, the residuals have a metric similar to z-values. On the other hand, they may often result in `NA` values; for these cases, it may be better to use the normalized residuals. For more information about the normalized and standardized residuals, see the Mplus reference below.

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. For complete data only.

anova `signature(object = "lavaan")`: returns model comparison statistics. This method is just a wrapper around the function `lavTestLRT`. If only a single argument (a fitted model) is provided, this model is compared to the unrestricted model. If two or more arguments (fitted models) are provided, the models are compared in a sequential order. Test statistics are based on the likelihood ratio test. For more details and further options, see the `lavTestLRT` page.

update `signature(object = "lavaan", model, ..., 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 method is now a shortcut for the `lavInspect()` function. See lavInspect for more details.

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

summary `signature(object = "lavaan", header = TRUE, fit.measures=FALSE, estimates = TRUE, ci = FALSE`
Print a nice summary of the model estimates. If `header = TRUE`, the header section (including fit measures) is printed. If `fit.measures = TRUE`, additional fit measures are added to the header section. If `estimates = TRUE`, print the parameter estimates section. If `ci = TRUE`, add confidence intervals to the parameter estimates section. If `fmi = TRUE`, add the fmi (fraction of missing information) column, if it is available. If `standardized=TRUE`, the standardized solution is also printed. If `rsquare=TRUE`, the R-Square values for the dependent variables in the model are printed. If `std.nox = TRUE`, the `std.all` column contains the the `std.nox` column from the parameterEstimates() output. If `modindices=TRUE`, modification indices are printed for all fixed parameters. The argument `nd` determines the number of digits after the decimal point to be printed (currently only in the parameter estimates section.) Nothing is returned (use `inspect` or another extractor function to extract information from a fitted model).

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. URL http://www.jstatsoft.org/v48/i02/.

Standardized Residuals in Mplus. Document retrieved from URL http://www.statmodel.com/download/StandardizedResidual

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")
```

---

lavaan-deprecated *Deprecated Functions in the lavaan package*

Description

These functions have been renamed, or have been removed. They are still included in this version for convenience, but they may be eventually removed.

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 has been renamed lav_matrix_vech.

The vech.reverse function has been renamed lav_matrix_vech_reverse.

The lower2full function has been renamed lav_matrix_lower2full.

The duplicationMatrix function has been renamed lav_matrix_duplication.

The commutationMatrix function has been renamed lav_matrix_commutation.

The sqrtSymmetricMatrix function has been renamed lav_matrix_symmetric_sqrt.

---

lavaanList *Fit List of Latent Variable Models*

Description

Fit the same latent variable model, for a (potentially large) number of datasets.

Usage

```
lavaanList(model = NULL, dataList = NULL, dataFunction = NULL,
    dataFunction.args = list(), ndat = length(dataList), cmd = "lavaan",
    ..., store.slots = c("partable"), FUN = NULL, show.progress = FALSE,
    parallel = c("no", "multicore", "snow"), ncpus = max(1L,
        parallel::detectCores() - 1L), cl = NULL, iseed = NULL)

semList(model = NULL, dataList = NULL, dataFunction = NULL,
    dataFunction.args = list(), ndat = length(dataList),
    ..., store.slots = c("partable"), FUN = NULL, show.progress = FALSE,
    parallel = c("no", "multicore", "snow"), ncpus = max(1L,
        parallel::detectCores() - 1L), cl = NULL, iseed = NULL)

cfaList(model = NULL, dataList = NULL, dataFunction = NULL,
    dataFunction.args = list(), ndat = length(dataList),
    ..., store.slots = c("partable"), FUN = NULL, show.progress = FALSE,
    parallel = c("no", "multicore", "snow"), ncpus = max(1L,
        parallel::detectCores() - 1L), cl = NULL, iseed = NULL)
```

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 table (eg. the output of the lavaanify() function) is also accepted.
dataList	List. Each element contains a full data frame containing the observed variables used in the model.
dataFunction	Function. A function that generated a full data frame containing the observed variables used in the model. It can also be a matrix, if the columns are named.

dataFunction.args
: List. Optional list of arguments that are passed to the dataFunction function.

ndat
: Integer. The number of datasets that should be generated using the dataFunction function.

cmd
: Character. Which command is used to run the sem models. The possible choices are "sem", "cfa" or "lavaan", determining how we deal with default options.

...
: Other named arguments for lavaan function.

store.slots
: Character vector. Which slots (from a lavaan object) should be stored for each dataset? The possible choices are "timing", "partable", "data", "samplestats", "vcov", "test", "optim" or "implied". Finally, "all" selects all slots.

FUN
: Function. A function which when applied to the lavaan object returns the information of interest.

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

ncpus
: Integer. The 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 lavaanList call.

iseed
: An integer to set the seed. Or NULL if no reproducible seeds are needed. To make this work, make sure the first RNGkind() element is "L'Ecuyer-CMRG". You can check this by typing RNGkind() in the console. You can set it by typing RNGkind("L'Ecuyer-CMRG"), before the lavaanList functions are called.

show.progress
: If TRUE, show information for each dataset.

Value

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

See Also

lavaanList

Examples

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

# a data generating function
generateData <- function() simulateData(HS.model, sample.nobs = 100)

fit <- semList(HS.model, dataFunction = generateData, ndat = 10,
               store.slots = "partable")

# show parameter estimates, per dataset
coef(fit)
```

lavaanList-class *Class For Representing A List of (Fitted) Latent Variable Models*

Description

The lavaanList class represents a collection of (fitted) latent variable models, for a (potentially large) number of datasets. It contains information about the model (which is always the same), and for every dataset a set of (user-specified) slots from a regular lavaan object.

Objects from the Class

Objects can be created via the cfaList, semList, or lavaanList functions.

Slots

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

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'.

pta: Named list containing parameter table attributes.

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

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

timingList: List. Timing slot per dataset.

ParTableList: List. ParTable slot per dataset.

DataList: List. Data slot per dataset.

SampleStatsList: List. SampleStats slot per dataset.

CacheList: List. Cache slot per dataset.

vcovList: List. vcov slot per dataset.

testList: List. test slot per dataset.

optimList: List. optim slot per dataset.

impliedList: List. implied slot per dataset.

funList: List. fun slot per dataset.

external: List. Empty slot to be used by add-on packages.

Methods

coef signature(object = "lavaanList", type = "free"): Returns the estimates of the parameters in the model as the columns in a matrix; each column corresponds to a different datasets. If type="free", only the free parameters are returned. If type="user", all parameters listed in the parameter table are returned, including constrained and fixed parameters.

summary `signature(object = "lavaanList", header = TRUE, estimates = TRUE, nd = 3L)`: Print a summary of the collection of fitted models. If `header = TRUE`, the header section is printed. If `estimates = TRUE`, print the parameter estimates section. The argument `nd` determines the number of digits after the decimal point to be printed (currently only in the parameter estimates section.) Nothing is returned (use `parameterEstimates` or another extractor function to extract information from this object).

See Also

`cfaList`, `semList`, `lavaanList`

`lavCor` *Polychoric, polyserial and Pearson correlations*

Description

Fit an unrestricted model to compute polychoric, polyserial and/or Pearson correlations.

Usage

```
lavCor(object, ordered = NULL, group = NULL, missing = "listwise",
       ov.names.x = NULL, se = "none", estimator = "two.step", ...,
       output = "cor")
```

Arguments

`object`	Either a `data.frame`, or an object of class `lavaan`. If the input is a `data.frame`, and some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
`ordered`	Character vector. Only used if `object` is a `data.frame`. Treat these variables as ordered (ordinal) variables. Importantly, all other variables will be treated as numeric (unless they are declared as ordered in the original data frame.)
`group`	Only used if `object` is a `data.frame`. Specify a grouping variable.
`missing`	If `"listwise"`, cases with missing values are removed listwise from the data frame. If `"direct"` or `"ml"` or `"fiml"` and the estimator is maximum likelihood, an EM algorithm is used to estimate the unrestricted covariance matrix (and mean vector). If `"pairwise"`, pairwise deletion is used. If `"default"`, the value is set depending on the estimator and the mimic option.
`ov.names.x`	Only used if `object` is a `data.frame`. Specify variables that need to be treated as exogenous. Only used if at least one variable is declared as ordered.
`se`	Only used if `output` (see below) contains standard errors. See the `lavaan` function for possible options.

estimator — If "none" or "two.step" or "two.stage", only starting values are computed for the correlations (and thresholds), without any further estimation. If all variables are continuous, the starting values are the sample covariances (converted to correlations if output = "cor"). If at least one variable is ordered, the thresholds are computed using univariate information only. The polychoric and/or polyserial correlations are computed in a second stage, keeping the values of the thresholds constant. If an estimator (other than "two.step" or "two.stage") is specified (for example estimator = "PML"), these starting values are further updated by fitting the unrestricted model using the chosen estimator. See the lavaan function for alternative estimators.

... — Optional parameters that are passed to the lavaan function.

output — If "cor", the function returns the correlation matrix only. If "cov", the function returns the covariance matrix (this only makes a difference if at least one variable is numeric). If "th" or "thresholds", only the thresholds are returned. If "sampstat", the output equals the result of inspect(fit, "sampstat") where fit is the unrestricted model. If "est" or "pe" or "parameterEstimates", the output equals the result of parameterEstimates(fit). Finally, if output is "fit" or "lavaan", the function returns an object of class lavaan.

Details

This function is a wrapper around the lavaan function, but where the model is defined as the unrestricted model. The following free parameters are included: all covariances/correlations among the variables, variances for continuous variables, means for continuous variables, thresholds for ordered variables, and if exogenous variables are included (ov.names.x is not empty) while some variables are ordered, also the regression slopes enter the model.

Value

By default, if output = "cor" or output = "cov", a symmetric matrix (of class "lavaan.matrix.symmetric", which only affects the way the matrix is printed). If output = "th", a named vector of thresholds. If output = "fit" or output = "lavaan", an object of class lavaan.

References

Olsson, U. (1979). Maximum likelihood estimation of the polychoric correlation coefficient. Psychometrika, 44(4), 443-460.

Olsson, U., Drasgow, F., & Dorans, N. J. (1982). The polyserial correlation coefficient. Psychometrika, 47(3), 337-347.

See Also

lavaan

Examples

```
# Holzinger and Swineford (1939) example
HS9 <- HolzingerSwineford1939[,c("x1","x2","x3","x4","x5",
                                 "x6","x7","x8","x9")]
```

```
# Pearson correlations
lavCor(HS9)

# ordinal version, with three categories
HS9ord <- as.data.frame( lapply(HS9, cut, 3, labels=FALSE) )

# polychoric correlations, two-stage estimation
lavCor(HS9ord, ordered=names(HS9ord))

# thresholds only
lavCor(HS9ord, ordered=names(HS9ord), output = "th")

# polychoric correlations, with standard errors
lavCor(HS9ord, ordered=names(HS9ord), se = "standard", output="est")

# polychoric correlations, full output
fit.un <- lavCor(HS9ord, ordered=names(HS9ord), se = "standard", output="fit")
summary(fit.un)
```

`lavExport` *lavaan Export*

Description

Export a fitted lavaan object to an external program.

Usage

```
lavExport(object, target = "lavaan", prefix = "sem", dir.name = "lavExport",
          export = TRUE)
```

Arguments

`object`	An object of class `lavaan`.
`target`	The target program. Current options are `"lavaan"` and `"Mplus"`.
`prefix`	The prefix used to create the input files; the name of the input file has the pattern 'prefix dot target dot in'; the name of the data file has the pattern 'prefix dot target dot raw'.
`dir.name`	The directory name (including a full path) where the input files will be written.
`export`	If `TRUE`, the files are written to the output directory (`dir.name`). If `FALSE`, only the syntax is generated as a character string.

Details

This function was mainly created to quickly generate an Mplus syntax file to compare the results between Mplus and lavaan. The target `"lavaan"` can be useful to create a full model syntax as needed for the `lavaan()` function. More targets (perhaps for `LISREL` or `EQS`) will be added in future releases.

Value

If `export = TRUE`, a directory (called `lavExport` by default) will be created, typically containing a data file, and an input file so that the same analysis can be run using an external program. If `export = FALSE`, a character string containing the model syntax only for the target program.

See Also

`lavaanify`, `mplus2lavaan`

Examples

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

fit <- cfa(HS.model, data=HolzingerSwineford1939)
out <- lavExport(fit, target = "Mplus", export=FALSE)
cat(out)
```

---

lavInspect *Inspect or extract information from a fitted lavaan object*

Description

The `lavInspect()` and `lavTech()` functions can be used to inspect/extract information that is stored inside (or can be computed from) a fitted lavaan object. Note: the (older) S4 `inspect()` method is now a shortcut for `lavInspect()` with default arguments.

Usage

```
lavInspect(lavobject, what = "free", add.labels = TRUE, add.class = TRUE,
           list.by.group = TRUE,
           drop.list.single.group = TRUE)

lavTech(lavobject, what = "free", add.labels = FALSE, add.class = FALSE,
        list.by.group = FALSE,
        drop.list.single.group = FALSE)
```

Arguments

`lavobject`	An object of class `lavaan`.
`what`	Character. What needs to be inspected/extracted? See Details for a full list. Note: the `what` argument is not case-sensitive (everything is converted to lower case.)
`add.labels`	If `TRUE`, variable names are added to the vectors and/or matrices.

`add.class` If `TRUE`, vectors are given the 'lavaan.vector' class; matrices are given the 'lavaan.matrix' class, and symmetric matrices are given the 'lavaan.matrix.symmetric' class. This only affects the way they are printed on the screen.

`list.by.group` Logical. Only used when the output are model matrices. If `TRUE`, the model matrices are nested within groups. If `FALSE`, a flattened list is returned containing all model matrices, with repeated names for multiple groups.

`drop.list.single.group` If `FALSE`, the results are returned as a list, where each element corresponds to a group (even if there is only a single group.) If `TRUE`, the list will be unlisted if there is only a single group.

Details

The `lavInspect()` and `lavTech()` functions only differ in the way they return the results. The `lavInspect()` function will prettify the output by default, while the `lavTech()` will not attempt to prettify the output by default. The (older) `inspect()` function is a simplified version of `lavInspect()` with only the first two arguments.

Below is a list of possible values for the `what` argument, organized in several sections:

Model matrices:

`"free"`: A list of model matrices. The non-zero integers represent the free parameters. The numbers themselves correspond to the position of the free parameter in the parameter vector. This determines the order of the model parameters in the output of for example `coef()` and `vcov()`.

`"partable"`: A list of model matrices. The non-zero integers represent both the fixed parameters (for example, factor loadings fixed at 1.0), and the free parameters if we ignore any equality constraints. They correspond with all entries (fixed or free) in the parameter table. See `parTable`.

`"se"`: A list of model matrices. The non-zero numbers represent the standard errors for the free parameters in the model. If two parameters are constrained to be equal, they will have the same standard error for both parameters. Aliases: `"std.err"` and `"standard.errors"`.

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

`"est"`: A list of model matrices. The values represent the estimated model parameters. Aliases: `"estimates"`, `"coef"`, `"coefficients"` and `"x"`.

`"dx.free"`: A list of model matrices. The values represent the gradient (first derivative) values of the model parameters. If two parameters are constrained to be equal, they will have the same gradient value.

`"dx.all"`: A list of model matrices. The values represent the first derivative with respect to all possible matrix elements. Currently, this is only available when the estimator is `"ML"` or `"GLS"`.

`"std"`: A list of model matrices. The values represent the (completely) standardized model parameters (the variances of both the observed and the latent variables are set to unity). Aliases: `"std.all"`, `"standardized"`.

`"std.lv"`: A list of model matrices. The values represent the standardized model parameters (only the variances of the latent variables are set to unity.)

"std.nox": A list of model matrices. The values represent the (completely) standardized model parameters (the variances of both the observed and the latent variables are set to unity; however, the variances of any observed exogenous variables are not set to unity; hence no-x.)

Information about the data (including missing patterns):

"data": A matrix containing the observed variables that have been used to fit the model. No column/row names are provided. Column names correspond to the output of lavNames(object), while the rows correspond to the output of lavInspect(object, "case.idx".

"group": A character string. The group variable in the data.frame (if any).

"ngroups": Integer. The number of groups.

"group.label": A character vector. The group labels.

"nobs": Integer vector. The number of observations in each group that were used in the analysis.

"norig": Integer vector. The original number of observations in each group.

"ntotal": Integer. The total number of observations that were used in the analysis. If there is just a single group, this is the same as the "nobs" option; if there are multiple groups, this is the sum of the "nobs" numbers for each group.

"case.idx": The case/observation numbers that were used in the analysis. In the case of multiple groups: a list of numbers.

"empty.idx": The case/observation numbers of those cases/observations that contained missing values only (at least for the observed variables that were included in the model). In the case of multiple groups: a list of numbers.

"patterns": A binary matrix. The rows of the matrix are the missing data patterns where 1 and 0 denote non-missing and missing values for the corresponding observed variables respectively (or TRUE and FALSE if lavTech() is used.) If the data is complete (no missing values), there will be only a single pattern. In the case of multiple groups: a list of pattern matrices.

"coverage": A symmetric matrix where each element contains the proportion of observed datapoints for the corresponding pair of observed variables. In the case of multiple groups: a list of coverage matrices.

Observed sample statistics:

"sampstat": Observed sample statistics. Aliases: "samp", "sample", "samplestatistics". In the presence of missing values, the sample covariance matrix is computed using use="pairwise", while listwise deletion is used for the means (and thresholds, if any).

"sampstat.h1": If all variables are continuous, and missing = "ml" (or "fiml"), the EM algorithm is used to compute an estimate of the sample covariance matrix and mean vector under the unrestricted (H1) model. Aliases: "h1", "missing.h1".

"wls.obs": The observed sample statistics (covariance elements, intercepts/thresholds, etc.) in a single vector.

"wls.v": The weight vector as used in weighted least squares estimation.

"gamma": N times the asymptotic variance matrix of the sample statistics. Alias: "sampstat.nacov".

Model features:

"meanstructure": Logical. TRUE if a meanstructure was included in the model.

`"categorical"`: Logical. `TRUE` if categorical endogenous variables were part of the model.

`"fixed.x"`: Logical. `TRUE` if the exogenous x-covariates are treated as fixed.

`"parameterization"`: Character. Either `"delta"` or `"theta"`.

Model-implied sample statistics:

`"cov.lv"`: The model-implied variance-covariance matrix of the latent variables. Alias: `"veta"` [for V(eta)].

`"cor.lv"`: The model-implied correlation matrix of the latent variables.

`"mean.lv"`: The model-implied mean vector of the latent variables. Alias: `"eeta"` [for E(eta)].

`"cov.ov"`: The model-implied variance-covariance matrix of the observed variables. Aliases: `"sigma"`, `"sigma.hat"`.

`"cor.ov"`: The model-implied correlation matrix of the observed variables.

`"mean.ov"`: The model-implied mean vector of the observed variables. Aliases: `"mu"`, `"mu.hat"`.

`"cov.all"`: The model-implied variance-covariance matrix of both the observed and latent variables.

`"cor.all"`: The model-implied correlation matrix of both the observed and latent variables.

`"th"`: The model-implied thresholds. Alias: `"thresholds"`.

`"wls.est"`: The model-implied sample statistics (covariance elements, intercepts/thresholds, etc.) in a single vector.

`"vy"`: The model-implied unconditional variances of the observed variables.

`"rsquare"`: The R-square value for all endogenous variables. Aliases: `"r-square"`, `"r2"`.

Optimizer information:

`"converged"`: Logical. `TRUE` if the optimizer has converged; `FALSE` otherwise.

`"iteratons"`: Integer. The number of iterations used by the optimizer.

`"optim"`: List. All available information regarding the optimization results.

Gradient, Hessian, observed, expected and first.order information matrices:

`"gradient"`: Numeric vector containing the first derivatives of the discrepancy function with respect to the (free) model parameters.

`"hessian"`: Matrix containing the second derivatives of the discrepancy function with respect to the (free) model parameters.

`"information"`: Matrix containing either the observed or the expected information matrix (depending on the information option of the fitted model.

`"information.expected"`: Matrix containing the expected information matrix for the free model parameters.

`"information.observed"`: Matrix containing the observed information matrix for the free model parameters.

`"information.first.order"`: Matrix containing the first.order information matrix for the free model parameters. This is the outer product of the gradient elements (the first derivative of the discrepancy function with respect to the (free) model parameters). Alias: `"first.order"`.

"augmented.information": Matrix containing either the observed or the expected augmented (or bordered) information matrix (depending on the information option of the fitted model. Only relevant if constraints have been used in the model.

"augmented.information.expected": Matrix containing the expected augmented (or bordered) information matrix. Only relevant if constraints have been used in the model.

"augmented.information.observed": Matrix containing the observed augmented (or bordered) information matrix. Only relevant if constraints have been used in the model.

"augmented.information.first.order": Matrix containing the first.order augmented (or bordered) information matrix. Only relevant if constraints have been used in the model.

"inverted.information": Matrix containing either the observed or the expected inverted information matrix (depending on the information option of the fitted model.

"inverted.information.expected": Matrix containing the inverted expected information matrix for the free model parameters.

"inverted.information.observed": Matrix containing the inverted observed information matrix for the free model parameters.

"inverted.information.first.order": Matrix containing the inverted first.order information matrix for the free model parameters.

Variance covariance matrix of the model parameters:

"vcov": Matrix containing the variance covariance matrix of the estimated model parameters.

"vcov.std.all": Matrix containing the variance covariance matrix of the standardized estimated model parameters. Standardization is done with respect to both observed and latent variables.

"vcov.std.lv": Matrix containing the variance covariance matrix of the standardized estimated model parameters. Standardization is done with respect to the latent variables only.

"vcov.std.nox": Matrix containing the variance covariance matrix of the standardized estimated model parameters. Standardization is done with respect to both observed and latent variables, but ignoring any exogenous observed covariates.

Miscellaneous:

"UGamma": Matrix containing the product of 'U' and 'Gamma' matrices as used by the Satorra-Bentler correction. The trace of this matrix, divided by the degrees of freedom, gives the scaling factor.

"list": The parameter table. The same output as given by `parTable()`.

"fit": The fit measures. Aliases: "fitmeasures", "fit.measures", "fit.indices". The same output as given by `fitMeasures()`.

"mi": The modification indices. Alias: "modindices", "modification.indices". The same output as given by `modindices()`.

"options": List. The option list.

"call": List. The call as returned by match.call, coerced to a list.

"timing": List. The timing (in milliseconds) of various lavaan subprocedures.

"test": List. All available information regarding the (goodness-of-fit) test statistic(s).

"post.check": Post-fitting check if the solution is admissible. A warning is raised if negative variances are found, or if either `lavInspect(fit, "cov.lv")` or `lavInspect(fit, "theta")` return a non-positive definite matrix.

See Also

lavaan

Examples

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

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

# extract information
lavInspect(fit, "sampstat")
lavTech(fit, "sampstat")
```

---

lavMatrixRepresentation

lavaan matrix representation

Description

Extend the parameter table with a matrix representation.

Usage

```
lavMatrixRepresentation(partable, representation = "LISREL",
                        add.attributes = FALSE, as.data.frame. = TRUE)
```

Arguments

partable	A lavaan parameter table (as extracted by the parTable function, or generated by the lavPartable function).
representation	Character. The matrix representation style. Currently, only the all-y version of the LISREL representation is supported.
add.attributes	Logical. If TRUE, additional information about the model matrix representation is added as attributes.
as.data.frame.	Logical. If TRUE, the extended parameter table is returned as a data.frame.

Value

A list or a data.frame containing the original parameter table, plus three columns: a "mat" column containing matrix names, and a "row" and "col" column for the row and column indices of the model parameters in the model matrices.

See Also

lavParTable, parTable

Examples

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

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

# extract partable
partable <- parTable(fit)

# add matrix representation (and show only a few columns)
lavMatrixRepresentation(partable)[,c("lhs","op","rhs","mat","row","col")]
```

---

lavNames *lavaan Names*

Description

Extract variables names from a fitted lavaan object.

Usage

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

Arguments

object	An object of class lavaan.
type	Character. The type of variables whose names should be extracted. See details for a complete list.
group	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 order of the variable names, as returned by lavNames determines the order in which the variables are listed in the parameter table, and therefore also in the summary output.

The following variable types are available:

- "ov": observed variables
- "ov.x": (pure) exogenous observed variables (no mediators)
- "ov.nox": non-exogenous observed variables
- "ov.model": modelled observed variables (joint vs conditional)
- "ov.y": (pure) endogenous variables (dependent only) (no mediators)
- "ov.num": numeric observed variables

- "ov.ord": ordinal observed variables
- "ov.ind": observed indicators of latent variables
- "ov.orphan": lonely observed variables (only intercepts/variancesappear in the model syntax)
- "ov.interaction": interaction terms (defined by the colon operator)
- "th": threshold names ordinal variables only
- "th.mean": threshold names ordinal + numeric variables (if any)
- "lv": latent variables
- "lv.regular": latent variables (defined by =~ only)
- "lv.formative": latent variables (defined by <~ only)
- "lv.x": (pure) exogenous variables
- "lv.y": (pure) endogenous variables
- "lv.nox": non-exogenous latent variables
- "lv.nonnormal": latent variables with non-normal indicators
- "lv.interaction": interaction terms at the latent level
- "eqs.y": variables that appear as dependent variables in a regression formula (but not indicators of latent variables)
- "eqs.x": variables that appear as independent variables in a regression formula

See Also

lavaanify, parTable

Examples

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

fit <- cfa(HS.model, data=HolzingerSwineford1939)
lavNames(fit, "ov")
```

---

lavPredict *Predict the values of latent variables (and their indicators).*

Description

The lavPredict() function can be used to compute (or 'predict') estimated values for latent variables, and given these values, the model-implied values for the indicators of these latent variables.

Usage

```
lavPredict(object, type = "lv", newdata = NULL, method = "EBM",
           se.fit = FALSE, label = TRUE, fsm = FALSE,
           optim.method = "nlminb")
```

Arguments

object	An object of class lavaan.
type	A character string. If "lv", estimated values for the latent variables in the model are computed. If "ov", model predicted values for the indicators of the latent variables in the model are computed.
newdata	An optional data.frame, containing the same variables as the data.frame used when fitting the model in object.
method	A character string. In the linear case (when the indicators are continuous), the possible options are "regression" or "Bartlett". In the categorical case, the only option (for now) is "EBM" for the Empirical Bayes Modal approach.
se.fit	Not used yet.
label	Logical. If TRUE, the columns are labeled.
fsm	Logical. If TRUE, return the factor score matrix as an attribute. Only for numeric data.
optim.method	Character string. Only used in the categorical case. If "nlminb" (the default), the "nlminb()" function is used for the optimization. If "BFGS", the "optim()" function is used with the BFGS method.

Details

The predict() function calls the lavPredict() function with its default options.

If there are no latent variables in the model, type = "ov" will simply return the values of the observed variables. Note that this function can not be used to 'predict' values of dependent variables, given the values of independent values (in the regression sense). In other words, the structural component is completely ignored (for now).

See Also

lavaan

Examples

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

fit <- cfa(HS.model, data=HolzingerSwineford1939)
head(lavPredict(fit))
head(lavPredict(fit, type = "ov"))
```

lavTables *lavaan frequency tables*

Description

Frequency tables for categorical variables and related statistics.

Usage

```
lavTables(object, dimension = 2L, type = "cells", categorical = NULL,
    group = NULL, statistic = "default", G2.min = 3, X2.min = 3,
    p.value = FALSE, output = "data.frame", patternAsString = TRUE)
```

Arguments

object	Either a data.frame, or an object of class lavaan.
dimension	Integer. If 0L, display all response patterns. If 1L, display one-dimensional (one-way) tables; if 2L, display two-dimensional (two-way or pairwise) tables. For the latter, we can change the information per row: if type = "cells", each row is a cell in a pairwise table; if type = "table", each row is a table.
type	If "cells", display information for each cell in the (one-way or two-way) table. If "table", display information per table. If "pattern", display response patterns (implying "dimension = 0L").
categorical	Only used if object is a data.frame. Specify variables that need to be treated as categorical.
group	Only used if object is a data.frame. Specify a grouping variable.
statistic	Either a character string, or a vector of character strings requesting one or more statistics for each cell, pattern or table. Always available are X2 and G2 for the Pearson and LRT based goodness-of-fit statistics. A distinction is made between the unrestricted and restricted model. The statistics based on the former have an extension *.un, as in X2.un and G2.un. If object is a data.frame, the unrestricted versions of the statistics are the only ones available. For one-way tables, additional statistics are the thresholds (th.un and th). For two-way tables and type = "table", the following statistics are available: X2, G2, cor (polychoric correlation), RMSEA and the corresponding unrestricted versions (X2.un etc). Additional statistics are G2.average, G2.nlarge and G2.plarge statistics based on the cell values G2: G2.average is the average of the G2 values in each cell of the two-way table; G2.nlarge is the number of cells with a G2 value larger than G2.min, and G2.plarge is the proportion of cells with a G2 value larger than G2.min. A similar set of statistics based on X2 is also available. If "default", the selection of statistics (if any) depends on the dim and type arguments, and if the object is a data.frame or a fitted lavaan object.
G2.min	Numeric. All cells with a G2 statistic larger than this number are considered 'large', as reflected in the (optional) "G2.plarge" and "G2.nlarge" columns.

`X2.min` Numeric. All cells with a X2 statistic larger than this number are considered 'large', as reflected in the (optional) `"X2.plarge"` and `"X2.nlarge"` columns.

`p.value` Logical. If `"TRUE"`, p-values are computed for requested statistics (eg G2 or X2) if possible.

`output` If `"data.frame"`, the output is presented as a data.frame where each row is either a cell, a table, or a response pattern, depending on the `"type"` argument. If `"table"`, the output is presented as a table (or matrix) or a list of tables. Only a single statistic can be shown in this case, and if the `statistic` is empty, the observed frequencies are shown.

`patternAsString` Logical. Only used for response patterns (dimension = 0L). If `"TRUE"`, response patterns are displayed as a compact string. If `"FALSE"`, as many columns as observed variables are displayed.

Value

If `output = "data.frame"`, the output is presented as a data.frame where each row is either a cell, a table, or a response pattern, depending on the `"type"` argument. If `output = "table"` (only for two-way tables), a list of tables (if `type = "cells"`) where each list element corresponds to a pairwise table, or if `type = "table"`, a single table (per group). In both cases, the table entries are determined by the (single) `statistic` argument.

References

Joreskog, K.G. & Moustaki, I. (2001). Factor analysis of ordinal variables: A comparison of three approaches. Multivariate Behavioral Research, 36, 347-387.

See Also

varTable.

Examples

```
HS9 <- HolzingerSwineford1939[,c("x1","x2","x3","x4","x5",
                                 "x6","x7","x8","x9")]
HSbinary <- as.data.frame( lapply(HS9, cut, 2, labels=FALSE) )

# using the data only
lavTables(HSbinary, dim = 0L, categorical = names(HSbinary))
lavTables(HSbinary, dim = 1L, categorical = names(HSbinary), stat=c("th.un"))
lavTables(HSbinary, dim = 2L, categorical = names(HSbinary), type = "table")

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

fit <- cfa(HS.model, data=HSbinary, ordered=names(HSbinary))
```

```
lavTables(fit, 1L)
lavTables(fit, 2L, type="cells")
lavTables(fit, 2L, type="table", stat=c("cor.un", "G2", "cor"))
lavTables(fit, 2L, type="table", output="table", stat="X2")
```

`lavTablesFitCp` *Pairwise maximum likelihood fit statistics*

Description

Three measures of fit for the pairwise maximum likelihood estimation method that are based on likelihood ratios (LR) are defined: C_F, C_M, and C_P. Subscript F signifies a comparison of model-implied proportions of full response patterns with observed sample proportions, subscript M signifies a comparison of model-implied proportions of full response patterns with the proportions implied by the assumption of multivariate normality, and subscript P signifies a comparison of model-implied proportions of pairs of item responses with the observed proportions of pairs of item responses.

Usage

```
lavTablesFitCf(object)
lavTablesFitCp(object, alpha = 0.05)
lavTablesFitCm(object)
```

Arguments

`object`	An object of class `lavaan`.
`alpha`	The nominal level of signifiance of global fit.

Details

C_F: The C_F statistic compares the log-likelihood of the model-implied proportions (π_r) with the observed proportions (p_r) of the full multivariate responses patterns:

$$C_F = 2N \sum_r p_r \ln[p_r/\hat{\pi}_r],$$

which asymptotically has a chi-square distribution with

$$df_F = m^k - n - 1,$$

where k denotes the number of items with discrete response scales, m denotes the number of response options, and n denotes the number of parameters to be estimated. Notice that C_F results may be biased because of large numbers of empty cells in the multivariate contingency table.

C_M: The C_M statistic is based on the C_F statistic, and compares the proportions implied by the model of interest (Model 1) with proportions implied by the assumption of an underlying multivariate normal distribution (Model 0):

$$C_M = C_{F1} - C_{F0},$$

where C_{F0} is C_F for Model 0 and C_{F1} is C_F for Model 1. Statistic C_M has a chi-square distribution with degrees of freedom

$$df_M = k(k-1)/2 + k(m-1) - n_1,$$

where k denotes the number of items with discrete response scales, m denotes the number of response options, and $k(k-1)/2$ denotes the number of polychoric correlations, $k(m-1)$ denotes the number of thresholds, and n_1 is the number of parameters of the model of interest. Notice that C_M results may be biased because of large numbers of empty cells in the multivariate contingency table. However, bias may cancels out as both Model 1 and Model 0 contain the same pattern of empty responses.

C_P**:** With the C_P statistic we only consider pairs of responses, and compare observed sample proportions (p) with model-implied proportions of pairs of responses(π). For items i and j we obtain a pairwise likelihood ratio test statistic $C_{P_{ij}}$

$$C_{P_{ij}} = 2N \sum_{c_i=1}^{m} \sum_{c_j=1}^{m} p_{c_i,c_j} \ln[p_{c_i,c_j}/\hat{\pi}_{c_i,c_j}],$$

where m denotes the number of response options and N denotes sample size. The C_P statistic has an asymptotic chi-square distribution with degrees of freedom equal to the information $(m^2 - 1)$ minus the number of parameters (2(m-1) thresholds and 1 correlation),

$$df_P = m^2 - 2(m-1) - 2.$$

As k denotes the number of items, there are $k(k-1)/2$ possible pairs of items. The C_P statistic should therefore be applied with a Bonferroni adjusted level of significance α^*, with

$$\alpha^* = \alpha/(k(k-1)/2)),$$

to keep the family-wise error rate at α. The hypothesis of overall goodness-of-fit is tested at α and rejected as soon as C_P is significant at α^* for at least one pair of items. Notice that with dichotomous items, $m = 2$, and $df_P = 0$, so that hypothesis can not be tested.

References

Barendse, M. T., Ligtvoet, R., Timmerman, M. E., & Oort, F. J. (2016). Structural Equation Modeling of Discrete data: Model Fit after Pairwise Maximum Likelihood.*Frontiers in psychology, 7*, 1-8.

Joreskog, K. G., & Moustaki, I. (2001). Factor analysis of ordinal variables: A comparison of three approaches. *Multivariate Behavioral Research, 36*, 347-387.

See Also

lavTables, lavaan

Examples

```
# Data
HS9 <- HolzingerSwineford1939[,c("x1","x2","x3","x4","x5",
```

```
                                    "x6","x7","x8","x9")]
HSbinary <- as.data.frame( lapply(HS9, cut, 2, labels=FALSE) )

# Single group example with one latent factor
HS.model <- ' trait =~ x1 + x2 + x3 + x4 '
fit <- cfa(HS.model, data=HSbinary[,1:4], ordered=names(HSbinary),
           estimator="PML")
lavTablesFitCm(fit)
lavTablesFitCp(fit)
lavTablesFitCf(fit)
```

`lavTestLRT` *LRT test*

Description

LRT test for comparing (nested) lavaan models.

Usage

```
lavTestLRT(object, ..., method = "default", A.method = "exact",
           H1 = TRUE, type = "Chisq", model.names = NULL)
anova(object, ...)
```

Arguments

`object`	An object of class `lavaan`.
`...`	additional objects of class `lavaan`.
`method`	Character string. The possible options are `"satorra.bentler.2001"`, `"satorra.bentler.2010"` and `"satorra.2000"`. See details.
`H1`	Not used yet
`A.method`	Character string. The possible options are `"exact"` and `"delta"`. This is only used when method = `"satorra.2000"`. It determines how the Jacobian of the constraint function (the matrix A) will be computed.
`type`	Character. If `"Chisq"`, the test statistic for each model is the (scaled or unscaled) model fit test statistic. If `"Cf"`, the test statistic for each model is computed by the `lavTablesFitCf` function.
`model.names`	Character vector. If provided, use these model names in the first column of the anova table.

Details

The `anova` function for lavaan objects simply calls the `lavTestLRT` function, which has a few additional arguments.

If `type = "Chisq"` and the test statistics are scaled, a special scaled difference test statistic is computed. If method is `"satorra.bentler.2001"`, a simple approximation is used described in

Satorra \& Bentler (2001). In some settings, this can lead to a negative test statistic. To ensure a positive test statistic, we can use the method proposed by Satorra \& Bentler (2010). Alternatively, when method is `"satorra.2000"`, the original formulas of Satorra (2000) are used. Note that for the Satorra (2000) method, the models must be nested in the parameter sense, while for the other methods, they only need to be nested in the covariance matrix sense.

Value

An object of class anova. When given a single argument, it simply returns the test statistic of this model. When given a sequence of objects, this function tests the models against one another in the order specified.

References

Satorra, A. (2000). Scaled and adjusted restricted tests in multi-sample analysis of moment structures. In Heijmans, R.D.H., Pollock, D.S.G. & Satorra, A. (eds.), Innovations in multivariate statistical analysis. A Festschrift for Heinz Neudecker (pp.233-247). London: Kluwer Academic Publishers.

Satorra, A., & Bentler, P. M. (2001). A scaled difference chi-square test statistic for moment structure analysis. Psychometrika, 66(4), 507-514.

Satorra, A., & Bentler, P. M. (2010). Ensuring postiveness of the scaled difference chi-square test statistic. Psychometrika, 75(2), 243-248.

Examples

```
HS.model <- '
    visual  =~ x1 + b1*x2 + x3
    textual =~ x4 + b2*x5 + x6
    speed   =~ x7 + b3*x8 + x9
'
fit1 <- cfa(HS.model, data = HolzingerSwineford1939)
fit0 <- cfa(HS.model, data = HolzingerSwineford1939,
            orthogonal = TRUE)
lavTestLRT(fit1, fit0)
```

---

lavTestScore *Score test*

Description

Score test (or Lagrange Multiplier test) for releasing one or more fixed or constrained parameters in model.

Usage

```
lavTestScore(object, add = NULL, release = NULL,
             univariate = TRUE, cumulative = FALSE, epc = FALSE,
             verbose = FALSE, warn = TRUE)
```

Arguments

object	An object of class lavaan.
add	Either a character string (typically between single quotes) or a parameter table containing additional (currently fixed-to-zero) parameters for which the score test must be computed.
release	Vector of Integers. The indices of the constraints that should be released. The indices correspond to the order of the equality constraints as they appear in the parameter table.
univariate	Logical. If TRUE, compute the univariate score statistics, one for each constraints.
cumulative	Logical. If TRUE, order the univariate score statistics from large to small, and compute a series of multivariate score statistics, each time adding an additional constraint.
epc	Logical. If TRUE, and we are releasing existing constraints, compute the expected parameter changes for the existing (free) parameters, for each released constraint.
verbose	Logical. Not used for now.
warn	Logical. If TRUE, print out warnings if they occur.

Details

This function can be used to compute both multivariate and univariate score tests. There are two modes: 1) releasing fixed-to-zero parameters (using the add argument), and 2) releasing existing equality constraints (using the release argument). The two modes can not be used simultaneously.

When adding new parameters, they should not already be part of the model (i.e. not listed in the parameter table). If you want to test for a parameter that was explicitly fixed to a constant (say to zero), it is better to label the parameter, and use an explicit equality constraint.

Value

A list containing at least three elements: the Score test statistic (stat), the degrees of freedom (df), and a p-value under the chi-square distribution (p.value). If univariate tests were requested, an additional element (TS.univariate) containing a numeric vector of univariate score statistics. If cumulative tests were requested, an additional element (TS.order) showing the order of the univariate test statistics, and an element (TS.cumulative) containing a numeric vector of cumulative multivariate score statistics.

References

Bentler, P. M., & Chou, C. P. (1993). Some new covariance structure model improvement statistics. Sage Focus Editions, 154, 235-255.

Examples

```
HS.model <- '
    visual  =~ x1 + b1*x2 + x3
    textual =~ x4 + b2*x5 + x6
```

```
    speed   =~ x7 + b3*x8 + x9

    b1 == b2
    b2 == b3
'
fit <- cfa(HS.model, data=HolzingerSwineford1939)

# test 1: release both two equality constraints
lavTestScore(fit, cumulative = TRUE)

# test 2: the score test for adding two (currently fixed
# to zero) cross-loadings
newpar = '
    visual =~ x9
    textual =~ x3
'
lavTestScore(fit, add = newpar)
```

`lavTestWald` *Wald test*

Description

Wald test for testing a linear hypothesis about the parameters of fitted lavaan object.

Usage

```
lavTestWald(object, constraints = NULL, verbose = FALSE)
```

Arguments

`object`	An object of class `lavaan`.
`constraints`	A character string (typically between single quotes) containing one or more equality constraints. See examples for more details.
`verbose`	Logical. If `TRUE`, print out the restriction matrix and the estimated restricted values.

Details

The constraints are specified using the `"=="` operator. Both the left-hand side and the right-hand side of the equality can contain a linear combination of model parameters, or a constant (like zero). The model parameters must be specified by their user-specified labels. Names of defined parameters (using the `":="` operator) can be included too.

Value

A list containing three elements: the Wald test statistic (stat), the degrees of freedom (df), and a p-value under the chi-square distribution (p.value).

Examples

```
HS.model <- '
    visual  =~ x1 + b1*x2 + x3
    textual =~ x4 + b2*x5 + x6
    speed   =~ x7 + b3*x8 + x9
'

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

# test 1: test about a single parameter
# this is the 'chi-square' version of the
# z-test from the summary() output
lavTestWald(fit, constraints = "b1 == 0")

# test 2: several constraints
con = '
   2*b1 == b3
   b2 - b3 == 0
'
lavTestWald(fit, constraints = con)
```

---

lav_constraints *Utility Functions: Constraints*

Description

Utility functions for equality and inequality constraints.

Usage

```
lav_constraints_parse(partable = NULL, constraints = NULL, theta = NULL,
                      debug = FALSE)
```

Arguments

partable	A lavaan parameter table.
constraints	A character string containing the constraints.
theta	A numeric vector. Optional vector with values for the model parameters in the parameter table.
debug	Logical. If TRUE, show debugging information.

Details

This is a collection of lower-level constraint related functions that are used in the lavaan code. They are made public per request of package developers. Below is a brief description of what they do:

The lav_constraints_parse function parses the constraints specification (provided as a string, see examples), and generates a list with useful information about the constraints.

Examples

```
myModel <- 'x1 ~ a*x2 + b*x3 + c*x4'
myParTable <- lavaanify(myModel, as.data.frame. = FALSE)
con <- ' a == 2*b
         b - c == 5 '
conInfo <- lav_constraints_parse(myParTable, constraints = con)
```

---

lav_func *Utility Functions: Gradient and Jacobian*

Description

Utility functions for computing the gradient of a scalar-valued function or the Jacobian of a vector-valued function by numerical approximation.

Usage

```
lav_func_gradient_complex(func, x, h = .Machine$double.eps, ...,
                          check.scalar = TRUE, fallback.simple = TRUE)
lav_func_jacobian_complex(func, x, h = .Machine$double.eps, ...,
                          fallback.simple = TRUE)

lav_func_gradient_simple(func, x, h = sqrt(.Machine$double.eps), ...,
                         check.scalar = TRUE)
lav_func_jacobian_simple(func, x, h = sqrt(.Machine$double.eps), ...)
```

Arguments

func	A real-valued function returning a numeric scalar or a numeric vector.
x	A numeric vector: the point(s) at which the gradient/Jacobian of the function should be computed.
h	Numeric value representing a small change in 'x' when computing the gradient/Jacobian.
...	Additional arguments to be passed to the function 'func'.
check.scalar	Logical. If TRUE, check if the function is scalar-valued.
fallback.simple	Logical. If TRUE, and the function evaluation fails, we call the corresponding simple (non-complex) method instead.

Details

The complex versions use complex numbers to gain more precision, while retaining the simplicity (and speed) of the simple forward method (see references). These functions were added to lavaan (around 2012) when the complex functionality was not part of the numDeriv package. They were used internally, and made public in 0.5-17 per request of other package developers.

References

Squire, W. and Trapp, G. (1998). Using Complex Variables to Estimate Derivatives of Real Functions. SIAM Review, 40(1), 110-112.

Examples

```
# very accurate complex method
lav_func_gradient_complex(func = exp, x = 1) - exp(1)

# less accurate forward method
lav_func_gradient_simple(func = exp, x = 1) - exp(1)

# very accurate complex method
diag(lav_func_jacobian_complex(func = exp, x = c(1,2,3))) - exp(c(1,2,3))

# less accurate forward method
diag(lav_func_jacobian_simple(func = exp, x = c(1,2,3))) - exp(c(1,2,3))
```

---

lav_matrix *Utility Functions: Matrices and Vectors*

Description

Utility functions for Matrix and Vector operations.

Usage

```
# matrix to vector
lav_matrix_vec(A)
lav_matrix_vecr(A)
lav_matrix_vech(S, diagonal = TRUE)
lav_matrix_vechr(S, diagonal = TRUE)

# matrix/vector indices
lav_matrix_vech_idx(n = 1L, diagonal = TRUE)
lav_matrix_vech_row_idx(n = 1L, diagonal = TRUE)
lav_matrix_vech_col_idx(n = 1L, diagonal = TRUE)
lav_matrix_vechr_idx(n = 1L, diagonal = TRUE)
lav_matrix_vechru_idx(n = 1L, diagonal = TRUE)
lav_matrix_diag_idx(n = 1L)
lav_matrix_diagh_idx(n = 1L)
lav_matrix_antidiag_idx(n = 1L)

# vector to matrix
lav_matrix_vech_reverse(x, diagonal = TRUE)
lav_matrix_vechru_reverse(x, diagonal = TRUE)
lav_matrix_upper2full(x, diagonal = TRUE)
lav_matrix_vechr_reverse(x, diagonal = TRUE)
```

```
lav_matrix_vechu_reverse(x, diagonal = TRUE)
lav_matrix_lower2full(x, diagonal = TRUE)

# the duplication matrix
lav_matrix_duplication(n = 1L)
lav_matrix_duplication_pre(A = matrix(0,0,0))
lav_matrix_duplication_post(A = matrix(0,0,0))
lav_matrix_duplication_pre_post(A = matrix(0,0,0))
lav_matrix_duplication_ginv(n = 1L)
lav_matrix_duplication_ginv_pre(A = matrix(0,0,0))
lav_matrix_duplication_ginv_post(A = matrix(0,0,0))
lav_matrix_duplication_ginv_pre_post(A = matrix(0,0,0))

# the commutation matrix
lav_matrix_commutation(m = 1L, n = 1L)
lav_matrix_commutation_pre(A = matrix(0,0,0))
lav_matrix_commutation_mn_pre(A, m = 1L, n = 1L)

# other matrix operations
lav_matrix_symmetric_sqrt(S = matrix(0,0,0))
lav_matrix_orthogonal_complement(A = matrix(0,0,0))
lav_matrix_bdiag(...)
lav_matrix_trace(..., check = TRUE)
```

Arguments

A	A general matrix.
S	A symmetric matrix.
diagonal	Logical. If TRUE, include the diagonal.
n	Integer. When it is the only argument, the dimension of a square matrix. If m is also provided, the number of column of the matrix.
m	Integer. The number of rows of a matrix.
x	Numeric. A vector.
...	One or more matrices, or a list of matrices.
check	Logical. If check = TRUE, we check if the (final) matrix is square.

Details

These are a collection of lower-level matrix/vector related functions that are used throughout the lavaan code. They are made public per request of package developers. Below is a brief description of what they do:

The lav_matrix_vec function implements the vec operator (for 'vectorization') and transforms a matrix into a vector by stacking the columns of the matrix one underneath the other.

The lav_matrix_vecr function is similar to the lav_matrix_vec function but transforms a matrix into a vector by stacking the rows of the matrix one underneath the other.

The lav_matrix_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. If diagonal = FALSE, the diagonal elements are also eliminated.

The lav_matrix_vechr function is similar to the lav_matrix_vech function but transforms a matrix into a vector by stacking the rows of the matrix one underneath the other, eliminating all supradiagonal elements.

The lav_matrix_vech_idx function returns the vector indices of the lower triangular elements of a symmetric matrix of size n, column by column.

The lav_matrix_vech_row_idx function returns the row indices of the lower triangular elements of a symmetric matrix of size n.

The lav_matrix_vech_col_idx function returns the column indices of the lower triangular elements of a symmetric matrix of size n.

The lav_matrix_vechr_idx function returns the vector indices of the lower triangular elements of a symmetric matrix of size n, row by row.

The lav_matrix_vechu_idx function returns the vector indices of the upper triangular elements of a symmetric matrix of size n, column by column.

The lav_matrix_vechru_idx function returns the vector indices of the upper triangular elements of a symmetric matrix of size n, row by row.

The lav_matrix_diag_idx function returns the vector indices of the diagonal elements of a symmetric matrix of size n.

The lav_matrix_diagh_idx function returns the vector indices of the lower part of a symmetric matrix of size n.

The lav_matrix_antidiag_idx function returns the vector indices of the anti diagonal elements a symmetric matrix of size n.

The lav_matrix_vech_reverse function (alias: lav_matrix_vechru_reverse and lav_matrix_upper2full) creates a symmetric matrix, given only upper triangular elements, row by row. If diagonal = FALSE, an diagonal with zero elements is added.

The lav_matrix_vechr_reverse (alias: lav_matrix_vechu_reverse and lav_matrix_lower2full) creates a symmetric matrix, given only the lower triangular elements, row by row. If diagonal = FALSE, an diagonal with zero elements is added.

The lav_matrix_duplication function generates the duplication matrix for a symmetric matrix of size n. This matrix duplicates the elements in vech(S) to create vec(S) (where S is symmetric). This matrix is very sparse, and should probably never be explicitly created. Use one of the functions below.

The lav_matrix_duplication_pre function computes the product of the transpose of the duplication matrix and a matrix A. The A matrix should have n*n rows, where n is an integer. The duplication matrix is not explicitly created.

The lav_matrix_duplication_post function computes the product of a matrix A with the duplication matrix. The A matrix should have n*n columns, where n is an integer. The duplication matrix is not explicitly created.

The lav_matrix_duplication_pre_post function first pre-multiplies a matrix A with the transpose of the duplication matrix, and then post multiplies the result again with the duplication matrix.

A must be square matrix with n*n rows and columns, where n is an integer. The duplication matrix is not explicitly created. multiplies a matrix A with the

The `lav_matrix_duplication_ginv` function computes the generalized inverse of the duplication matrix. The matrix removes the duplicated elements in vec(S) to create vech(S). This matrix is very sparse, and should probably never be explicitly created. Use one of the functions below.

The `lav_matrix_duplication_ginv_pre` function computes the product of the generalized inverse of the duplication matrix and a matrix A with n*n rows, where n is an integer. The generalized inverse of the duplication matrix is not explicitly created.

The `lav_matrix_duplication_ginv_post` function computes the product of a matrix A (with n*n columns, where n is an integer) and the transpose of the generalized inverse of the duplication matrix. The generalized inverse of the duplication matrix is not explicitly created.

The `lav_matrix_duplication_ginv_pre_post` function first pre-multiplies a matrix A with the transpose of the generalized inverse of the duplication matrix, and the post multiplies the result again with the transpose of the generalized inverse matrix. The matrix A must be square with n*n rows and columns, where n is an integer. The generalized inverse of the duplication matrix is not explicitly created.

The `lav_matrix_commutation` function computes the commutation matrix which is a permutation matrix which transforms vec(A) (with m rows and n columns) into vec(t(A)).

The `lav_matrix_commutation_pre` function computes the product of the commutation matrix with a matrix A, without explicitly creating the commutation matrix. The matrix A must have n*n rows, where n is an integer.

The `lav_matrix_commutation_mn_pre` function computes the product of the commutation matrix with a matrix A, without explicitly creating the commutation matrix. The matrix A must have m*n rows, where m and n are integers.

The `lav_matrix_symmetric_sqrt` function computes the square root of a positive definite symmetric matrix (using an eigen decomposition). If some of the eigenvalues are negative, they are silently fixed to zero.

The `lav_matrix_orthogonal_complement` function computes an orthogonal complement of the matrix A, using a qr decomposition.

The `lav_matrix_bdiag` function constructs a block diagonal matrix from its arguments.

The `lav_matrix_trace` function computes the trace (the sum of the diagonal elements) of a single (square) matrix, or if multiple matrices are provided (either as a list, or as multiple arguments), we first compute their product (which must result in a square matrix), and then we compute the trace; if `check = TRUE`, we check if the (final) matrix is square.

References

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

Examples

```
# upper elements of a 3 by 3 symmetric matrix (row by row)
x <- c(30, 16, 5, 10, 3, 1)
# construct full symmetric matrix
S <- lav_matrix_upper2full(x)
```

```
# compute the normal theory `Gamma' matrix given a covariance
# matrix (S), using the formula: Gamma = 2 * D^{+} (S %x% S) t(D^{+})
Gamma.NT <- 2 * lav_matrix_duplication_ginv_pre_post(S %x% S)
Gamma.NT
```

---

lav_model *lavaan model functions*

Description

Utility functions related to internal model representation (lavmodel)

Usage

```
# set/get free parameters
lav_model_set_parameters(lavmodel, x = NULL, estimator = "ML")
lav_model_get_parameters(lavmodel, GLIST = NULL, type = "free",
                         extra = TRUE)

# compute model-implied statistics
lav_model_implied(lavmodel)

# compute standard errors
lav_model_vcov_se(lavmodel, lavpartable, VCOV = NULL, BOOT = NULL)
```

Arguments

lavmodel	An internal representation of a lavaan model.
x	Numeric.A vector containing the values of all the free model parameters.
estimator	Character string.The estimator we should assume has been used.
GLIST	List. A list of model matrices, similar to the output of `lavInspect(object, "est")`.
type	Character string. If "free", only return the free model parameters. If "user", return all the parameters (free and fixed) as they appear in the user-specified parameter table.
extra	Logical. If TRUE, also include values for rows in the parameter table where the operator is one of ":=", "==", "<" or ">".
lavpartable	A parameter table.
VCOV	Numeric matrix containing an estimate of the variance covariance matrix of the free model parameters.
BOOT	Numeric matrix containing the bootstrap based parameter estimates (in the columns) for each bootstrap sample (in the rows).

Examples

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

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

est <- lav_model_get_parameters(lavmodel)
est
```

---

lav_partable *lavaan partable functions*

Description

Utility functions related to the parameter table (partable)

Usage

```
# extract information from a parameter table
lav_partable_df(partable, group = NULL)
lav_partable_ndat(partable, group = NULL)
lav_partable_npar(partable)
lav_partable_attributes(partable, pta = NULL)

# generate parameter labels
lav_partable_labels(partable, blocks = "group", group.equal = "",
                    group.partial = "", type = "user")

# generate parameter table for specific models
lav_partable_independence(lavobject = NULL, lavdata = NULL, lavoptions = NULL,
    lavsamplestats = NULL, sample.cov = NULL, sample.mean = NULL,
    sample.th = NULL, sample.th.idx = NULL)

lav_partable_unrestricted(lavobject = NULL, lavdata = NULL, lavoptions = NULL,
    lavsamplestats = NULL, sample.cov = NULL, sample.mean = NULL,
    sample.slopes = NULL, sample.th = NULL, sample.th.idx = NULL)

lav_partable_from_lm(object, est = FALSE, label = FALSE, as.data.frame. = FALSE)

# complete a parameter table only containing a few columns (lhs,op,rhs)
lav_partable_complete(partable = NULL, start = TRUE)

# merge two parameter tables
lav_partable_merge(pt1 = NULL, pt2 = NULL, remove.duplicated = FALSE,
    fromLast = FALSE, warn = TRUE)
```

Arguments

partable	A parameter table. see lavParTable for more information.
group	Integer. If non-null, only consider this group.
blocks	Character vector. Which columns in the parameter table should be taken to distinguish between different blocks/groups of parameters (and hence be given different labels)?
group.equal	The same options can be used here as in the fitting functions. Parameters that are constrained to be equal across groups will be given the same label.
group.partial	A vector of character strings containing the labels of the parameters which should be free in all groups.
type	Character string. Can be either 'user' or 'free' to select all entries or only the free parameters from the parameter table respectively.
lavobject	An object of class 'lavaan'. If this argument is provided, it should be the only argument. All the values for the other arguments are extracted from this object.
lavdata	An object of class 'lavData'. The Data slot from a lavaan object.
lavoptions	A names list. The Options lsot from a lavaan object.
lavsamplestats	An object of class 'lavSampleStats'. The SampleStats slot from a lavaan object.
sample.cov	Optional list of numeric matrices. Each list element contains a sample variance-covariance matrix for this group. If provided, these values will be used as starting values.
sample.mean	Optional list of numeric vectors. Each list element contains a sample mean vector for this group. If provided, these values will be used as starting values.
sample.slopes	Optional list of numeric matrices. Each list element contains the sample slopes for this group (only used when conditional.x = TRUE). If provided, these values will be used as starting values.
sample.th	Optional list of numeric vectors. Each list element contains a vector of sample thresholds for this group. If provided (and also sample.th.idx is provided), these values will be used as starting values.
sample.th.idx	Optional list of integers. Each list contains the threshold indices for this group.
est	Logical. If TRUE, include the fitted estimates in the parameter table.
label	Logical. If TRUE, include parameter labels in the parameter table.
as.data.frame.	Logical. If TRUE, return the parameter table as a data.frame.
object	An object of class lm.
start	Logical. If TRUE, include a start column, based on the simple method for generating starting values.
pta	A list containing parameter attributes.
pt1	A parameter table.
pt2	A parameter table.
remove.duplicated	Logical. If TRUE, remove duplicated elements when merging two parameter tables.

fromLast	Logical. If TRUE, duplicated elements are considered from the bottom of the merged parameter table.
warn	Logical. If codeTRUE, a warning is produced when duplicated elements are found, when merging two parameter tables.

Examples

```
# generate syntax for an independence model
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- cfa(HS.model, data=HolzingerSwineford1939)
lav <- lav_partable_independence(fit)
as.data.frame(lav, stringsAsFactors = FALSE)


# how many free parameters?
lav_partable_npar(lav)

# how many sample statistics?
lav_partable_ndat(lav)

# how many degrees of freedom?
lav_partable_df(lav)
```

---

model.syntax *The Lavaan Model Syntax*

Description

The lavaan model syntax describes a latent variable model. The function lavaanify turns it into a table that represents the full model as specified by the user. We refer to this table as the parameter table.

Usage

```
lavaanify(model = NULL, meanstructure = FALSE, int.ov.free = FALSE,
    int.lv.free = FALSE, orthogonal = FALSE, std.lv = FALSE,
    conditional.x = FALSE, fixed.x = TRUE, parameterization = "delta",
    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, auto.th = FALSE,
    auto.delta = FALSE, varTable = NULL, ngroups = 1L, group.equal = NULL,
    group.partial = NULL, group.w.free = FALSE,
    debug = FALSE, warn = TRUE, as.data.frame. = TRUE)

lavParTable(model = NULL, meanstructure = FALSE, int.ov.free = FALSE,
```

```
    int.lv.free = FALSE, orthogonal = FALSE, std.lv = FALSE,
    conditional.x = FALSE, fixed.x = TRUE, parameterization = "delta",
    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, auto.th = FALSE,
    auto.delta = FALSE, varTable = NULL, ngroups = 1L, group.equal = NULL,
    group.partial = NULL, group.w.free = FALSE,
    debug = FALSE, warn = TRUE, as.data.frame. = TRUE)

lavParseModelString(model.syntax = '', as.data.frame.=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 details for more information. Alternatively, a parameter table (e.g., the output of lavParseModelString is also accepted.
model.syntax	The model syntax specifying the model. Must be a literal string.
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.
conditional.x	If TRUE, we set up the model conditional on the exogenous 'x' covariates; the model-implied sample statistics only include the non-x variables. If FALSE, the exogenous 'x' variables are modeled jointly with the other variables, and the model-implied statistics refect both sets of variables.
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.
parameterization	Currently only used if data is categorical. If "delta", the delta parameterization is used. If "theta", the theta parameterization is used.
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.
auto.th	If TRUE, thresholds for limited dependent variables are included in the model and set free.
auto.delta	If TRUE, response scaling parameters for limited dependent variables are included in the model and set free.
varTable	The variable table containing information about the observed variables in the model.
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).
group.w.free	Logical. If TRUE, the group frequencies are considered to be free parameters in the model. In this case, a Poisson model is fitted to estimate the group frequencies. If FALSE (the default), the group frequencies are fixed to their observed values.
warn	If TRUE, some (possibly harmless) warnings are printed out.
as.data.frame.	If TRUE, return the list of model parameters as a data.frame.
debug	If TRUE, debugging information is printed out.

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 <- '
  # 1. 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

  # 2. regressions
```

```
  f1 ~ f3 + f4
  f2 ~ f4
  y1 + y2 ~ x1 + x2 + x3

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

# 4. intercepts
  f1 ~ 1; y5 ~ 1

# 5. thresholds
  y11 | t1 + t2 + t3
  y12 | t1
  y13 | t1 + t2

# 6. scaling factors
  y11 ~*~ y11
  y12 ~*~ y12
  y13 ~*~ y13

# 7. formative factors
  f5 <~ z1 + z2 + z3 + z4
'
```

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 seven 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.

5. Thresholds: The `"|"` operator can be used to define the thresholds of categorical endogenous variables (on the left hand side of the operator). By convention, the thresholds (on the right hand sided, separated by the `"+"` operator, are named `"t1"`, `"t2"`, etcetera.
6. Scaling factors: The `"~*~"` operator defines a scale factor. The variable name on the left hand side must be the same as the variable name on the right hand side. Scale factors are used in the Delta parameterization, in a multiple group analysis when factor indicators are categorical.
7. Formative factors: The `"<~"` operator can be used to define a formative factor (on the right hand side of the operator), in a similar why as a reflexive factor is defined (using the `"=~"` operator). This is just syntax sugar to define a phantom latent variable (equivalent to using `"f =~ 0"`). And in addition, the (residual) variance of the formative factor is fixed to zero.

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). The only exception are scaling factors, where only a single element is allowed on the left hand side.

In the right-hand side of these formula-like expressions, each element can be modified (using the `"*"` operator) by either a numeric constant, an expression resulting in a numeric constant, an expression resulting in a character vector, 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. All `"*"` expressions are referred to as *modifiers*. They are 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 a modifier resulting in a numerical constaint. 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 special function `start()`, containing a numeric constant. 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 simply by pre-multiplying the corresponding term (on the right hand side of the operator only) by a character string (starting with a letter). For example, in the formula `f1 =~ x1 + x2 + mylabel*x3`, the parameter corresponding with the factor loading of `x3` will be named `"mylabel"`. An alternative way to specify the label is as follows: `f1 =~ x1 + x2 + label("mylabel")*x3`, where the label is the argument of special function `label()`; this can be useful if the label contains a space, or an operator (like "~").

To constrain a parameter to be equal to another target parameter, there are two ways. If you have specified your own labels, you can use the fact that *equal labels imply equal parameter values*. If you rely on automatic parameter labels, you can use 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 ~ int2*1
    y2b ~ int2*1
    y2c ~ int2*1
'
```

Multiple groups

In a multiple group analysis, modifiers that contain a single constant must be replaced by a vector, having the same length as the number of groups. The only exception are numerical constants (for fixing values): if you provide only a single number, the same number will be used for all groups.

However, it is safer (and cleaner) to specify the same number of elements as the number of 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 + 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. Note that character vectors can contain unquoted strings.

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 + x3.loading*x3
```

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. URL http://www.jstatsoft.org/v48/i02/.

---

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,
                    sort. = FALSE, minimum.value = 0,
                    maximum.number = nrow(LIST), free.remove = TRUE,
                    na.remove = TRUE, op = NULL)
modindices(object, standardized = TRUE, power = FALSE,
                    delta = 0.1, alpha = 0.05, high.power = 0.75,
                    sort. = FALSE, minimum.value = 0,
                    maximum.number = nrow(LIST), free.remove = TRUE,
                    na.remove = TRUE, op = NULL)
```

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.
sort.	Logical. If TRUE, sort the output using the values of the modification index values. Higher values appear first.
minimum.value	Numeric. Filter output and only show rows with a modification index value equal or higher than this minimum value.
maximum.number	Integer. Filter output and only show the first maximum number rows. Most useful when combined with the sort. option.
free.remove	Logical. If TRUE, filter output by removing all rows corresponding to free (unconstrained) parameters in the original model.
na.remove	Logical. If TRUE, filter output by removing all rows with NA values for the modification indices.
op	Character string. Filter the output by selectin only those rows with operator op.

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)
```

`mplus2lavaan` *mplus to lavaan converter*

Description

Read in an Mplus input file, convert it to lavaan syntax, and fit the model.

Usage

```
mplus2lavaan(inpfile)
```

Arguments

`inpfile`	The filename (including a full path) of the Mplus input file. The data (as referred to in the Mplus input file) should be in the same directory as the Mplus input file.

Value

A list with two elements: `mplus.inp` contains the input data, a title, the variable names, and the converted (lavaan) model syntax; `lav.out` contains the fitted lavaan object.

Author(s)

Michael Hallquist

See Also

`lavExport`.

Examples

```
## Not run:
out <- mplus2lavaan("ex5.1.inp")
summary(out$lav.out)

## End(Not run)
```

parameterEstimates *Parameter Estimates*

Description

Parameter estimates of a latent variable model.

Usage

```
parameterEstimates(object, se = TRUE, zstat = TRUE, pvalue = TRUE, ci = TRUE,
                   level = 0.95, boot.ci.type = "perc", standardized = FALSE,
                   fmi = FALSE, remove.system.eq = TRUE, remove.eq = TRUE,
                   remove.ineq = TRUE, remove.def = FALSE, rsquare = FALSE,
                   add.attributes = FALSE)
```

Arguments

object	An object of class lavaan.
se	Logical. If TRUE, include column containing the standard errors. If FALSE, this implies zstat and pvalue and ci are also FALSE.
zstat	Logical. If TRUE, an extra column is added containing the so-called z-statistic, which is simply the value of the estimate divided by its standard error.
pvalue	Logical. If TRUE, an extra column is added containing the pvalues corresponding to the z-statistic, evaluated under a standard normal distribution.
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	Logical. If TRUE, standardized estimates are added to the output
fmi	Logical. If TRUE, an extra column is added containing the fraction of missing information for each estimated parameter. Only available if estimator="ML", missing="(fi)ml", and se="standard". See references for more information.
remove.eq	Logical. If TRUE, filter the output by removing all rows containing user-specified equality constraints, if any.
remove.system.eq	Logical. If TRUE, filter the output by removing all rows containing system-generated equality constraints, if any.
remove.ineq	Logical. If TRUE, filter the output by removing all rows containing inequality constraints, if any.
remove.def	Logical. If TRUE, filter the ouitput by removing all rows containing parameter definitions, if any.

rsquare	Logical. If TRUE, add additional rows containing the rsquare values (in the est column) of all endogenous variables in the model. Both the lhs and rhs column contain the name of the endogenous variable, while the codeop column contains r2, to indicate that the values in the est column are rsquare values.
add.attributes	Logical. If TRUE, add a class attribute (class lavaan.parameterEstimates) and other attributes to be used by the print function for this class (print.lavaan.parameterEstimates). This is used by the summary() function, to prettify the output.

Value

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

References

Savalei, V. & Rhemtulla, M. (2012). On obtaining estimates of the fraction of missing information from FIML. Structural Equation Modeling: A Multidisciplinary Journal, 19(3), 477-494.

Examples

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

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

---

parTable *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)
parTable(fit)
```

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 (link no longer valid; 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, data = NULL,
    meanstructure = "default",
    conditional.x = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE,
    parameterization = "default", std.ov = FALSE,
    missing = "default", ordered = NULL,
    sample.cov = NULL, sample.cov.rescale = "default",
    sample.mean = NULL, sample.nobs = NULL,
    ridge = 1e-05, group = NULL,
    group.label = NULL, group.equal = "", group.partial = "",
    group.w.free = FALSE, cluster = NULL, constraints = '',
    estimator = "default", likelihood = "default", link = "default",
    information = "default", se = "default", test = "default",
    bootstrap = 1000L, mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), WLS.V = NULL, NACOV = NULL,
    zero.add = "default", zero.keep.margins = "default",
    zero.cell.warn = TRUE, start = "default",
    check = c("start", "post"),
    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 table (eg. the output of the `lavaanify()` function) is also accepted.

`data` An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.

`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.

`conditional.x` If `TRUE`, we set up the model conditional on the exogenous 'x' covariates; the model-implied sample statistics only include the non-x variables. If `FALSE`, the exogenous 'x' variables are modeled jointly with the other variables, and the model-implied statistics refect both sets of variables. If `"default"`, the value is set depending on the estimator, and whether or not the model involves categorical endogenous variables.

`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 (residual) 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.

`parameterization`
Currently only used if data is categorical. If `"delta"`, the delta parameterization is used. If `"theta"`, the theta parameterization is used.

`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.

`ordered` Character vector. Only used if the data is in a data.frame. Treat these variables as ordered (ordinal) variables, if they are endogenous in the model. Importantly, all other variables will be treated as numeric (unless they are declared as ordered in the original data.frame.)

`sample.cov` Numeric matrix. A sample variance-covariance matrix. The rownames and/or colnames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group. Note that if maximum likelihood estimation is used and `likelihood="normal"`, the user provided covariance matrix is internally rescaled by multiplying it with a factor (N-1)/N, to ensure that the covariance matrix has been divided by N. This can be turned off by setting the `sample.cov.rescale` argument to `FALSE`.

`sample.cov.rescale`
If `TRUE`, the sample covariance matrix provided by the user is internally rescaled by multiplying it with a factor (N-1)/N. If `"default"`, the value is set depending on the estimator and the likelihood option: it is set to `TRUE` if maximum likelihood estimation is used and `likelihood="normal"`, and `FALSE` otherwise.

`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.

`ridge` Numeric. Small constant used for ridging. Only used if the sample covariance matrix is non positive definite.

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

`group.label` A character vector. The user can specify which group (or factor) levels need to be selected from the grouping variable, and in which order. If `NULL` (the default), all grouping levels are selected, in the order as they appear in the data.

`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"`, `"thresholds"`, `"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).

`group.w.free` Logical. If `TRUE`, the group frequencies are considered to be free parameters in the model. In this case, a Poisson model is fitted to estimate the group frequencies. If `FALSE` (the default), the group frequencies are fixed to their observed values.

`cluster` Not used yet.

`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), `"ULS"` for unweighted least squares and `"DWLS"` for diagonally weighted least squares. These are the main options that affect the estimation. For convenience, the `"ML"` option can be extended as `"MLM"`, `"MLMV"`, `"MLMVS"`, `"MLF"`, and `"MLR"`. The estimation will still be plain `"ML"`, but now with robust standard errors and a robust (scaled) test statistic. For `"MLM"`, `"MLMV"`, `"MLMVS"`, classic robust standard errors are used (`se="robust.sem"`); for `"MLF"`, standard errors are based on first-order derivatives (`se="first.order"`); for `"MLR"`, 'Huber-White' robust standard errors are used (`se="robust.huber.white"`). In addition, `"MLM"` will compute a Satorra-Bentler scaled (mean adjusted) test statistic (`test="satorra.bentler"`) , `"MLMVS"` will compute a mean and variance adjusted test statistic (Satterthwaite style) (`test="mean.var.adjusted"`), `"MLMV"` will compute a mean and variance adjusted test statistic (scaled and shifted) (`test="scaled.shifted"`), and `"MLR"` will compute a test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Analogously, the estimators `"WLSM"` and `"WLSMV"` imply the `"DWLS"` estimator (not the `"WLS"` estimator) with robust standard errors and a mean or mean and variance adjusted test statistic. Estimators `"ULSM"` and `"ULSMV"` imply the `"ULS"`

estimator with robust standard errors and a mean or mean and variance adjusted test statistic.

`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="lavaan"` or `mimic="Mplus"`, normal likelihood is used; otherwise, wishart likelihood is used.

`link` Currently only used if estimator is MML. If `"logit"`, a logit link is used for binary and ordered observed variables. If `"probit"`, a probit link is used. If `"default"`, it is currently set to `"probit"` (but this may change).

`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.sem"`, conventional robust standard errors are computed. If `"robust.huber.white"`, standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If `"robust"`, either `"robust.sem"` or `"robust.huber.white"` 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 `"mean.var.adjusted"` or `"Satterthwaite"`, a mean and variance adjusted test statistic is compute. If `"scaled.shifted"`, an alternative mean and variance adjusted test statistic is computed (as in Mplus version 6 or higher). 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 to `"lavaan"`, which is very close 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".

WLS.V — A user provided weight matrix to be used by estimator "WLS"; if the estimator is "DWLS", only the diagonal of this matrix will be used. For a multiple group analysis, a list with a weight matrix for each group. The elements of the weight matrix should be in the following order (if all data is continuous): first the means (if a meanstructure is involved), then the lower triangular elements of the covariance matrix including the diagonal, ordered column by column. In the categorical case: first the thresholds (including the means for continuous variables), then the slopes (if any), the variances of continuous variables (if any), and finally the lower triangular elements of the correlation/covariance matrix excluding the diagonal, ordered column by column.

NACOV — A user provided matrix containing the elements of (N times) the asymptotic variance-covariance matrix of the sample statistics. For a multiple group analysis, a list with an asymptotic variance-covariance matrix for each group. See the WLS.V argument for information about the order of the elements.

zero.add — A numeric vector containing two values. These values affect the calculation of polychoric correlations when some frequencies in the bivariate table are zero. The first value only applies for 2x2 tables. The second value for larger tables. This value is added to the zero frequency in the bivariate table. If "default", the value is set depending on the "mimic" option. By default, lavaan uses zero.add = c(0.5. 0.0).

zero.keep.margins — Logical. This argument only affects the computation of polychoric correlations for 2x2 tables with an empty cell, and where a value is added to the empty cell. If TRUE, the other values of the frequency table are adjusted so that all margins are unaffected. If "default", the value is set depending on the "mimic". The default is TRUE.

zero.cell.warn — Logical. Only used if some observed endogenous variables are categorical. If TRUE, give a warning if one or more cells of a bivariate frequency table are empty.

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.

check — Character vector. If check includes "start", the starting values are checked for possibly inconsistent values (for example values implying correlations larger than one); if check includes "post", a check is performed after (post) fitting, to check if the solution is admissable.

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, auto.th = TRUE, auto.delta = TRUE, and auto.cov.y = TRUE.

Value

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

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. URL http://www.jstatsoft.org/v48/i02/.

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 = FALSE,
    int.ov.free = TRUE, int.lv.free = FALSE, conditional.x = FALSE,
    fixed.x = FALSE,
    orthogonal = FALSE, std.lv = TRUE, auto.fix.first = FALSE,
    auto.fix.single = FALSE, auto.var = TRUE, auto.cov.lv.x = TRUE,
    auto.cov.y = TRUE, ..., sample.nobs = 500L, ov.var = NULL,
    group.label = paste("G", 1:ngroups, sep = ""), skewness = NULL,
    kurtosis = NULL, seed = NULL, empirical = FALSE,
    return.type = "data.frame", return.fit = FALSE,
    debug = FALSE, standardized = 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 table (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.
conditional.x	If TRUE, we set up the model conditional on the exogenous 'x' covariates; the model-implied sample statistics only include the non-x variables. If FALSE, the exogenous 'x' variables are modeled jointly with the other variables, and the model-implied statistics refect both sets of variables. If "default", the value is set depending on the estimator, and whether or not the model involves categorical endogenous variables.
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.

`ov.var` The user-specified variances of the observed variables.

`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).

`return.fit` If `TRUE`, return the fitted model that has been used to generate the data as an attribute (called `"fit"`); this may be useful for inspection.

`debug` If `TRUE`, debugging information is displayed.

`standardized` If `TRUE`, the residual variances of the observed variables are set in such a way such that the model implied variances are unity. This allows regression coefficients and factor loadings (involving observed variables) to be specified in a standardized metric.

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
                    '

# generate data
set.seed(1234)
myData <- simulateData(population.model, sample.nobs=100L)

# 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", se = TRUE,
                     zstat = TRUE, pvalue = TRUE, remove.eq = TRUE,
                     remove.ineq = TRUE, remove.def = FALSE)
```

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.
se	Logical. If TRUE, standard errors for the standardized parameters will be computed, together with a z-statistic and a p-value.
zstat	Logical. If TRUE, an extra column is added containing the so-called z-statistic, which is simply the value of the estimate divided by its standard error.
pvalue	Logical. If TRUE, an extra column is added containing the pvalues corresponding to the z-statistic, evaluated under a standard normal distribution.
remove.eq	Logical. If TRUE, filter the output by removing all rows containing equality constraints, if any.
remove.ineq	Logical. If TRUE, filter the output by removing all rows containing inequality constraints, if any.
remove.def	Logical. If TRUE, filter the ouitput by removing all rows containing parameter definitions, if any.

Value

A data.frame containing 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)
```

varTable *Variable Table*

Description

Summary information about the variables included in either a data.frame, or a fitted lavaan object.

Usage

```
varTable(object, ov.names = names(object), ov.names.x = NULL,
    ordered = NULL, factor = NULL, as.data.frame. = TRUE)
```

Arguments

object	Either a data.frame, or an object of class lavaan.
ov.names	Only used if object is a data.frame. A character vector containing the variables that need to be summarized.
ov.names.x	Only used if object is a data.frame. A character vector containing additional variables that need to be summarized.
ordered	Character vector. Which variables should be treated as ordered factors
factor	Character vector. Which variables should be treated as (unordered) factors?
as.data.frame.	If TRUE, return the list as a data.frame.

Value

A list or data.frame containing summary information about variables in a data.frame. If object is a fitted lavaan object, it displays the summary information about the observed variables that are included in the model. The summary information includes variable type (numeric, ordered, . . .), the number of non-missing values, the mean and variance for numeric variables, the number of levels of ordered variables, and the labels for ordered variables.

Examples

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

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

Index