Package: lavaan (via r-universe)

July 23, 2026

Title Latent Variable Analysis

Version 0.7-2

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

Depends R(>= 3.4)

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

BugReports https://github.com/yrosseel/lavaan/issues

License GPL (>= 2)

LazyData yes

LazyDataCompression xz

ByteCompile true

URL https://lavaan.ugent.be

NeedsCompilation no

Author Yves Rosseel [aut, cre] (ORCID: <https://orcid.org/0000-0002-4129-4477>), Terrence D. Jorgensen [aut] (ORCID: <https://orcid.org/0000-0001-5111-6773>), Luc De Wilde [aut], 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], Nicholas Rockwood [ctb], Florian Scharf [ctb], Han Du [ctb], Haziq Jamil [ctb] (ORCID: <https://orcid.org/0000-0003-3298-1010>), Franz Classe [ctb]

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

Repository https://cran.r-universe.dev

Date/Publication 2026-07-23 08:42:52 UTC

RemoteUrl https://github.com/cran/lavaan

RemoteRef HEAD

RemoteSha 7b78a73ef9d6436405c25c0f4e800e4f0e7566f1

Contents

`cfa` *Fit Confirmatory Factor Analysis Models*

Description

Fit a Confirmatory Factor Analysis (CFA) model.

Usage

```
cfa(model = NULL, data = NULL, ordered = NULL, aux = NULL, sampling_weights = NULL,
    sample_cov = NULL, sample_mean = NULL, sample_th = NULL,
    sample_nobs = NULL, group = NULL, cluster = NULL,
    constraints = "", wls_v = NULL, nacov = NULL, ov_order = "model",
    ...)
```

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 (e.g., the output of the `lavParTable()` 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.

`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 data.frame.) Since 0.6-4, ordered can also be logical. If TRUE, all observed endogenous variables are treated as ordered (ordinal). If FALSE, all observed endogenous variables are considered to be numeric (again, unless they are declared as ordered in the data.frame.)

`aux` Character vector. Names of auxiliary observed variables, used to make the missing-at-random (MAR) assumption more plausible under missing data (continuous data only). With `missing = "ml"` they are added to the model as saturated correlates (Graham, 2003), affecting the estimates and standard errors but hidden from the default output; with `missing = "two.stage"/"robust.two.stage"` they enter the first-stage EM moments. See the `lavaan` function for more details.

`sampling_weights`
A variable name in the data frame containing sampling weight information. Currently only available for non-clustered data. Depending on the `sampling.weights.normalization` option, these weights may be rescaled (or not) so that their sum equals the number of observations (total or per group).

`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.

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

`sample_th` Vector of sample-based thresholds. For a multiple group analysis, a list with a vector of thresholds for each group.

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

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

`cluster` Character. A (single) variable name in the data frame defining the clusters in a two-level dataset.

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

`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.

`ov_order` Character. If `"model"` (the default), the order of the observed variable names (as reflected for example in the output of `lav_object_vnames()`) is determined by the model syntax. If `"data"`, the order is determined by the data (either the full data.frame or the sample (co)variance matrix). If the `wls_v` and/or `nacov` matrices are provided, this argument is currently set to `"data"`.

`...` Many more options can be specified, using 'name = value'. See `lavOptions` for a complete list.

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.efa` = `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. doi:10.18637/jss.v048.i02

See Also

`lavaan`, `estimator_iv`

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 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)
```

`Demo.twolevel` *Demo dataset for illustrating a multilevel CFA*

Description

A toy dataset containing measures on 6 items (y1-y6), 3 within-level covariates (x1-x3) and 2 between-level covariates (w1-w2). The data is clustered (200 clusters of size 5, 10, 15 and 20), and the cluster variable is "cluster".

Usage

```
data(Demo.twolevel)
```

Format

A data frame of 2500 observations of 12 variables.

`y1` item 1

`y2` item 2

`y3` item 3

`y4` item 4

`y5` item 5

`y6` item 6

`x1` within-level covariate 1

`x2` within-level covariate 2

`x3` within-level covariate 3

`w1` between-level covariate 1

`w2` between-level covariate 2

`cluster` cluster variable

Examples

```
head(Demo.twolevel)

model <- '
    level: 1
        fw =~ y1 + y2 + y3
        fw ~ x1 + x2 + x3
    level: 2
        fb =~ y1 + y2 + y3
        fb ~ w1 + w2
'
fit <- sem(model, data = Demo.twolevel, cluster = "cluster")
summary(fit)
```

efa *Exploratory Factor Analysis*

Description

Fit one or more Exploratory Factor Analysis (EFA) model(s).

Usage

```
efa(data = NULL, nfactors = 1L, sample_cov = NULL, sample_nobs = NULL,
    rotation = "geomin", rotation_args = list(), ov_names = NULL,
    bounds = "pos.var", ..., output = "efa")
```

Arguments

data	A data frame containing the observed variables we need for the EFA. If only a subset of the observed variables is needed, use the ov.names argument.
nfactors	Integer, integer vector, or a list. The desired number of factors to extract. Can be a single number, or a vector of numbers (e.g., nfactors = 1:4); for each different number, a model is fitted. When the model has multiple blocks (because the data is multilevel and/or multigroup), the same number of factors is extracted in every block. To extract a different number of factors per block, supply a *list*: each element of the list is one fitted model and contains a vector with the number of factors per block (or a single number that is recycled to all blocks). Blocks are ordered group-major, level-minor, ie block $= (g-1) \times nlevels + level$. For example, for a twolevel model, nfactors = list(c(1, 2)) extracts one factor at the within level and two factors at the between level.
sample_cov	Numeric matrix. A sample variance-covariance matrix. The rownames and/or colnames must contain the observed variable names. Unlike sem and CFA, the matrix may be a correlation matrix.
sample_nobs	Number of observations if the full data frame is missing and only the sample variance-covariance matrix is given.
rotation	Character or a list with first element a character variable. The character variable is the rotation method to be used. Possible options are varimax, quartimax, orthomax, oblimin, quartimin, geomin, promax, entropy, mccammon, infomax, tandem1, tandem2, oblimax, bentler, simplimax, target.strict, target (alias for pst), pst (=partially specified target), cf, crawford-ferguson, cf-quartimax, cf-varimax, cf-equamax, cf-parsimax, cf-facparsim, biquartimin, bigeomin. The latter two are for bifactor rotation only. The rotation algorithms (except promax and target) are similar to those from the GPArotation package, but have been reimplemented for better control. The promax method is taken from the stats package. The target.strict method is equal to the target method in the GPArotation package. The target method is in fact the pst method where all non-zero elements (in the target matrix) are ignored. Options for the rotation algorithm can be provided in the list after the method. The default options (and their alternatives) are orthogonal = FALSE, row_weights = "default" (or "kaiser",

"cureton.mulaik" or "none"), std_ov = TRUE, algorithm = "gpa" (or "pairwise"), rstarts = 30, gpa_tol = 1e-05, tol = 1e-08, max_iter = 10000L, warn = FALSE, verbose = FALSE, reflect = TRUE, order_lv_by = "index" (or "sumofsquares" or "none"). Other options are specific for a particular rotation criterion: geomin_epsilon = 0.001, orthomax_gamma = 1, promax_kappa = 4, cf_gamma = 0, and oblimin_gamma = 0.

For the target rotation methods (target.strict, target and pst), the target option (and, for pst, the optional target_mask option) can take three forms: 1) a single matrix, which is used for all groups and all efa blocks; 2) a *nameless* list of matrices, one per group; or 3) a *named* list of matrices, where the names are the efa block labels used in the model syntax (e.g., list(a = target.a, b = target.b) if the model contains the efa blocks efa("a") and efa("b")), so that each efa block gets its own target matrix (the same targets are then used in all groups). The latter is only relevant when the model syntax (of sem() or cfa()) defines multiple efa blocks (set-ESEM). Note that a named list must provide a target matrix for *every* efa block (with two or more factors) in the model; omitting a block results in an error, as the rotation criterion would be undefined for that block.

rotation_args — List. Options related to the rotation algorithm. DEPRECATED. The options should now be given in the rotation argument as a list.

ov_names — Character vector. The variables names that are needed for the EFA. Should be a subset of the variables names in the data.frame. By default (if NULL), all the variables in the data are used.

bounds — Per default, bounds = "pos.var" forces all variances of both observed and latent variables to be strictly nonnegative. See the entry in lavOptions for more options.

... — Additional options to be passed to lavaan, using 'name = value'. See lavOptions for a complete list.

output — Character. If "efa" (the default), the output mimics the typical output of an EFA. If "lavaan", a lavaan object is returned. The latter is only possible if nfactors contains a single (integer) number.

Details

The efa function is essentially a wrapper around the lavaan function. It generates the model syntax (for a given number of factors) and then calls lavaan() treating the factors (in each block) as a set that should be rotated. Each block is rotated independently. Categorical data is handled as usual by first computing an appropriate (e.g., tetrachoric or polychoric) correlation matrix, which is then used as input for the EFA. Multiple groups (via the group= argument) and twolevel data (via the cluster= argument) are supported, and these may be combined. By default, the same number of factors is extracted in every block, but a different number of factors per block can be requested by supplying nfactors as a list (see the nfactors argument). The promax rotation method (taken from the stats package) is only provided for convenience. Because promax is a two-step algorithm (first varimax, then oblique rotation to get simple structure), it does not use the gpa or pairwise rotation algorithms, and as a result, no standard errors are provided.

Value

If output = "lavaan", an object of class lavaan. If output = "efa", a list of class efaList for which a print(), summary() and fitMeasures() method are available. Because we added the (standardized) loadings as an extra element, the loadings function (which is not a generic function) from the stats package will also work on efaList objects.

See Also

lav_efalist_summary for a summary method if the output is of class efaList.

Examples

```
## The famous Holzinger and Swineford (1939) example
fit <- efa(data = HolzingerSwineford1939,
           ov.names = paste("x", 1:9, sep = ""),
           nfactors = 1:3,
           rotation = list("geomin", geomin_epsilon = 0.01, rstarts = 1))
summary(fit, nd = 3L, cutoff = 0.2, dot.cutoff = 0.05)
fitMeasures(fit, fit.measures = "all")

# target rotation
target <- matrix(0, 9, 3)
target[1:3, 1] <- 1
target[4:6, 2] <- 1
target[7:9, 3] <- 1
fit2 <- efa(data = HolzingerSwineford1939,
            ov.names = paste("x", 1:9, sep = ""),
            nfactors = 3,
            rotation = list("target", target = target))
summary(fit2)

## Not run:
# twolevel EFA with a different number of factors per level:
# one factor at the within level, two factors at the between level
fit3 <- efa(data = Demo.twolevel, ov.names = paste0("y", 1:6),
            cluster = "cluster", nfactors = list(c(1, 2)))
summary(fit3)

## End(Not run)
```

---

estimator_iv *Instrumental Variable (MIIV-2SLS) Estimation*

Description

Estimate a structural equation model using model-implied instrumental variables and two-stage least squares (MIIV-2SLS), a non-iterative, equation-by-equation alternative to maximum likelihood (Bollen, 1996). This page documents how to invoke the estimator, the options that can be passed via the estimator = list() argument, how to specify instruments manually with the |~ operator, and the lavInspect() fields that are relevant for IV estimation.

Details

Instead of minimizing a single discrepancy function for the whole model, MIIV-2SLS estimates each measurement and structural equation separately by two-stage least squares (2SLS). For each equation, the instruments are other observed variables in the model that are (under the hypothesized model) uncorrelated with the equation's composite error term: the *model-implied instrumental variables* (MIIVs). Because the equations are estimated separately, a misspecification in one equation does not necessarily propagate to the others, the method is non-iterative, and no starting values are required.

The estimator is selected by setting `estimator = "IV"` (the alias `"MIIV"` is also accepted; names are case-insensitive), for example

```
fit <- sem(model, data = Data, estimator = "IV")
```

Estimation proceeds in two stages. In the first stage, the *directed* effects (factor loadings and regression coefficients) are estimated equation-by-equation by 2SLS. In the second stage, the *undirected* effects (residual variances and covariances) are estimated by a weighted least-squares step applied to the residual moments. Standard errors are available for both stages (see the options below).

Both continuous and ordered-categorical data are supported (for ordered data, the polychoric-based PIV estimator of Bollen & Maydeu-Olivares, 2007 is used). Simple equality constraints (e.g. equal factor loadings) and general linear equality constraints (e.g. `a == 2*b`) are honored. Multiple groups are supported, with standard errors (for continuous, two-stage-missing and categorical data), including the mean structure and cross-group equality constraints for measurement invariance testing (configural, metric and scalar, e.g. via `group.equal = c("loadings", "intercepts")`). For scalar invariance the latent-mean (and intercept) estimates are obtained by a joint GLS solve over the mean structure (see `iv_mean_structure`), which matches the maximum likelihood mean estimates given the model-implied covariance matrix. The IV estimator is restricted to single-level models.

Estimator options:

Options specific to the IV estimator are passed as named elements of a list to the `estimator` argument, alongside the estimator name:

```
fit <- sem(model, data = Data,
           estimator = list(estimator = "IV",
                            iv_varcov_method = "2RLS",
                            iv_sargan_adjust = "BH"))
```

The option names use snake_case. For backward compatibility, dot.case names (e.g. `iv.varcov.method`) are also accepted and silently converted.

`iv_method`**:** Character. The instrumental-variable method. Currently only `"2SLS"` (two-stage least squares) is available. Default is `"2SLS"`.

`iv_samplestats`**:** Logical. If `TRUE` (the default), the equations are estimated from the sample moments (covariances and means) rather than from the raw data. This is required for categorical data and when only sample moments are provided as input.

`iv_varcov_method`**:** Character. The second-stage method used to estimate the residual variances and covariances. One of `"ULS"`, `"GLS"`, `"2RLS"`, `"RLS"` or `"NONE"`. Default is `"RLS"` (reweighted least squares). For categorical data, `"ULS"` is used. If `"NONE"`, the variance/covariance parameters are not estimated.

`iv_sargan`: Logical. If `TRUE` (the default), overidentification tests are computed for each overidentified equation. Two tests are reported side by side: the classical Sargan test (Sargan, 1958), which assumes normality (continuous data) and is not valid for polychoric correlations; and a robust, residual-based test that is valid more generally. The robust test is Browne's (1984) residual statistic applied to the single equation, computed with the asymptotic covariance matrix (ACOV) of the sample statistics: the distribution-free (ADF) ACOV of the sample covariances for continuous data (so it is robust to non-normality, and is then equivalent to Hansen's overidentification test), or the polychoric ACOV for categorical data. Both tests share the same degrees of freedom (number of instruments minus number of regressors). For continuous data with missing values the robust test is not available. The results can be retrieved with `lavInspect(fit, "sargan")` (or, equivalently, `lavInspect(fit, "hansen")`).

`iv_sargan_adjust`: Character. A multiple-comparison adjustment applied to the per-equation overidentification p-values, using any method accepted by `p.adjust` (`"holm"`, `"hochberg"`, `"hommel"`, `"bonferroni"`, `"BH"`, `"BY"`, `"fdr"`, or `"none"`). Default is `"none"`. When a method other than `"none"` is requested, extra `sargan.pval.adj` and `browne.pval.adj` columns are added to the table returned by `lavInspect(fit, "sargan")`.

`iv_weak`: Character. How to respond to weak instruments, diagnosed per equation by the first-stage F-statistic (Staiger & Stock, 1997). One of:

- `"warn"`: (the default) report the affected equations and their first-stage F, and suggest supplying stronger instruments manually; the estimates are not changed.
- `"prune"`: greedily drop the weakest excess instruments from weak, overidentified equations (never below just-identified) and report what was dropped.
- `"none"`: skip the weak-instrument check.

`iv_weak_threshold`: Numeric. The first-stage F-statistic below which instruments are deemed weak. Default is `10`.

`iv_vcov_stage1`: Character. The standard-error method for the first-stage (directed) coefficients. One of `"lm.vcov"`, `"lm.vcov.dfres"`, `"gamma"` or `"none"`. Default is `"lm.vcov.dfres"` for continuous data and `"gamma"` for categorical data. The `"gamma"` method requires `iv_samplestats = TRUE`. When the model contains *simple* equality constraints among the directed coefficients (e.g. equal factor loadings), the `"lm.vcov"`/`"lm.vcov.dfres"` methods use a variance-weighted restricted-2SLS covariance for the constrained coefficients (as in the MIIVsem package); set `iv_vcov_stage1 = "gamma"` to obtain the delta-method (moment-Jacobian) standard errors instead.

`iv_vcov_stage2`: Character. The standard-error method for the second-stage (undirected) parameters. One of `"h2"`, `"delta"` or `"none"`. Default is `"h2"`. The `"h2"` method requires `iv_samplestats = TRUE`.

`iv_vcov_gamma_modelbased`: Logical. If `TRUE` (the default), the normal-theory weight matrix (gamma) used for the standard errors is based on the model-implied covariance matrix; if `FALSE`, it is based on the unrestricted (H1) sample covariance matrix.

`iv_mean_structure`: Character. How the free mean-structure parameters (observed intercepts and latent means) are estimated. With `"wls"` (the default) they are obtained by a joint GLS solve that fits the model-implied means to the sample means across all groups, pooling parameters that are constrained equal across groups (e.g. intercepts for scalar invariance); this matches the maximum likelihood mean estimates (given the model-implied covariance matrix). With `"moments"` the intercepts are recomputed per equation and shared intercepts are pooled by a (simpler) nobs-weighted average. The two agree unless intercepts are constrained across groups (scalar invariance).

`iv_vcov_jack_numerical`, `iv_vcov_jaca_numerical`, `iv_vcov_jacb_numerical`: Logical. If `TRUE`, the corresponding Jacobians used in the standard-error computation are obtained numerically rather than from analytic expressions. The default is `FALSE`; these are mainly intended for checking the analytic derivatives.

`iv_mimic_ml`: Logical. If `FALSE` (the default), the IV estimator uses unbiased divisors: the sample covariance matrix is divided by $N - 1$ (`sample.cov.rescale = FALSE`) and the per-equation residual variances (the RSS terms entering the standard errors and the Sargan test) use the $N - P$ divisor. If `TRUE`, the maximum likelihood divisor N is used throughout: the sample covariance is rescaled to the ML divisor N and the residual variances use the N divisor as well. As a result, for just-identified models the IV point estimates and standard errors match the ML solution exactly.

Specifying instruments manually (the `|~` operator):

By default lavaan determines the model-implied instruments for each equation automatically. The `|~` operator overrides this choice for a specific equation: the left-hand side is the dependent variable of the equation (for a latent variable, its scaling indicator), and the right-hand side lists the instruments to use.

```
model <- '
    ind60 =~ x1 + x2 + x3
    dem60 =~ y1 + y2 + y3 + y4
    dem60 ~ ind60
    # use only x2 and x3 as instruments for the
    # dem60 ~ ind60 equation
    y1 |~ x2 + x3
'
fit <- sem(model, data = PoliticalDemocracy, estimator = "IV")
```

This is useful when the automatically selected instruments are weak (see `iv_weak` above) or when subject-matter knowledge suggests a different set of instruments.

External instruments. The instruments listed on the right-hand side of `|~` need not be part of the model. If an instrument is an observed variable that does not otherwise appear in the model (it is not an indicator, a predictor, an outcome, or otherwise mentioned), it is treated as an *external* instrument: it is read from the data so that its covariances with the model variables are available to the 2SLS estimator, but it never enters the model-implied summary statistics. The model degrees of freedom, the fit measures, and the model-implied moments are therefore exactly the same as if the instrument had not been mentioned; only the equations for which it is used are affected. External instruments require raw data (a `data=` argument); they are not available when only sample moments are supplied. For categorical data the external instruments must themselves be ordered (categorical); the augmented polychoric correlations and the asymptotic covariance of the sample statistics are then used for the estimates and the standard errors. Single-group and multiple-group categorical models are supported, but (for now) not equality constraints among the directed coefficients (e.g. measurement invariance) together with external instruments.

```
## z is not part of the model; it is used only as an
## instrument for the (endogenous) regression y ~ x
model <- '
    y ~ x
    y |~ z
```

```
'
fit <- sem(model, data = Data, estimator = "IV")
```

For external instruments the point estimates and the directed-coefficient (loading and regression) standard errors are the usual 2SLS quantities. The second-stage (residual variance and covariance) standard errors also account for the sampling variability of the instrument moments: the directed coefficients depend on the instrument-model covariances, and this uncertainty is propagated into the variance/covariance parameters through the full (augmented) moment Jacobian.

Missing data:

For continuous data with missing values, the IV estimator defaults to a two-stage (FIML) approach: the saturated mean vector and covariance matrix are estimated by the EM algorithm, the model is then estimated from these moments, and the standard errors are corrected for the additional uncertainty stemming from the EM estimates (Savalei & Bentler, 2009). This default is used when the data contain missing values and the user does not set the `missing` argument. Set `missing = "robust.two.stage"` for sandwich-corrected standard errors (Savalei & Falk, 2014), or `missing = "listwise"` to delete incomplete cases instead. Two-stage missing-data estimation is also available for multiple-group models.

Multiple groups and measurement invariance:

Multiple-group models are supported, with point estimates and standard errors for continuous, two-stage-missing and categorical data. Cross-group equality constraints are imposed in the usual way, either through shared parameter labels in the model syntax or through the `group.equal` argument, so the standard measurement-invariance sequence is available: configural (`group.equal = NULL`), metric (`group.equal = "loadings"`) and scalar (`group.equal = c("loadings", "intercepts")`).

A parameter that is constrained equal across groups is estimated by pooling the information from all groups; for a configural model the per-group estimates (and their standard errors) reproduce the separate single-group fits. The mean structure (observed intercepts and latent means) is estimated by a joint GLS solve over all groups (see `iv_mean_structure`), which reproduces the maximum likelihood mean estimates given the model-implied covariance matrix.

Inspecting the instruments and the Sargan test:

Several `lavInspect` fields are specific to IV estimation:

`lavInspect(fit, "iv")`: (aliases `"miiv"`, `"instruments"`) a per-equation table with the dependent variable (`lhs`), the regressors (`rhs`), the observed variables actually used (`lhs_new`, `rhs_new`), the equation type (`type`: `"miiv"`, `"ols"` or `"user"`), and the instruments (`iv`) for that equation.

`lavInspect(fit, "sargan")`: a per-equation table with the overidentification tests: the degrees of freedom (`df`), the classical Sargan statistic (`sargan.stat`) and its p-value (`sargan.pval`), and the robust Browne residual-based statistic (`browne.stat`) and its p-value (`browne.pval`); see `iv_sargan` above. For categorical data the classical Sargan p-value is `NA` (not valid), so use the `browne.*` columns. Adjusted p-values (`sargan.pval.adj` and `browne.pval.adj`) are added when `iv_sargan_adjust` is set. `lavInspect(fit, "hansen")` is an alias.

`lavInspect(fit, "eqs")`: the underlying per-equation information.

References

Bollen, K. A. (1996). An alternative two stage least squares (2SLS) estimator for latent variable equations. *Psychometrika*, 61(1), 109-121.

Bollen, K. A., & Maydeu-Olivares, A. (2007). A polychoric instrumental variable (PIV) estimator for structural equation models with categorical variables. *Psychometrika*, 72(3), 309-326.

Browne, M. W. (1984). Asymptotically distribution-free methods for the analysis of covariance structures. *British Journal of Mathematical and Statistical Psychology*, 37(1), 62-83.

Sargan, J. D. (1958). The estimation of economic relationships using instrumental variables. *Econometrica*, 26(3), 393-415.

Savalei, V., & Bentler, P. M. (2009). A two-stage approach to missing data: Theory and application to auxiliary variables. *Structural Equation Modeling*, 16(3), 477-497.

Savalei, V., & Falk, C. F. (2014). Robust two-stage approach outperforms robust full information maximum likelihood with incomplete nonnormal data. *Structural Equation Modeling*, 21(2), 280-302.

Staiger, D., & Stock, J. H. (1997). Instrumental variables regression with weak instruments. *Econometrica*, 65(3), 557-586.

See Also

lavaan, sem, lavOptions, lavInspect.

Examples

```
## The classic Bollen (1989) Political Democracy example
model <- '
  # measurement model
    ind60 =~ x1 + x2 + x3
    dem60 =~ y1 + y2 + y3 + y4
    dem65 =~ y5 + y6 + y7 + y8
  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60
'

## fit using MIIV-2SLS instead of maximum likelihood
fit <- sem(model, data = PoliticalDemocracy, estimator = "IV")
summary(fit)

## the model-implied instruments used for each equation
lavInspect(fit, "iv")

## the per-equation Sargan overidentification tests
lavInspect(fit, "sargan")

## pass estimator options via estimator = list(...)
fit2 <- sem(model, data = PoliticalDemocracy,
            estimator = list(estimator = "IV",
                             iv_varcov_method = "2RLS",
                             iv_sargan_adjust = "BH",
                             iv_weak = "warn"))
lavInspect(fit2, "sargan")

## specify instruments manually with the |~ operator:
```

```
## use only x2, x3 as instruments for the dem60 ~ ind60 equation
model.iv <- '
    ind60 =~ x1 + x2 + x3
    dem60 =~ y1 + y2 + y3 + y4
    dem60 ~ ind60
    y1 |~ x2 + x3
'
fit3 <- sem(model.iv, data = PoliticalDemocracy, estimator = "IV")
lavInspect(fit3, "iv")

## Not run:
## equality constraints are honored (here: equal loadings across factors)
model.eq <- '
    ind60 =~ x1 + x2 + x3
    dem60 =~ y1 + a*y2 + b*y3 + c*y4
    dem65 =~ y5 + a*y6 + b*y7 + c*y8
    dem60 ~ ind60
    dem65 ~ ind60 + dem60
'
fit4 <- sem(model.eq, data = PoliticalDemocracy, estimator = "IV")
coef(fit4)

## multiple groups: metric measurement invariance (equal loadings)
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '
fit.metric <- sem(HS.model, data = HolzingerSwineford1939, estimator = "IV",
                  group = "school", group.equal = "loadings")
summary(fit.metric)

## End(Not run)
```

`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, h1_model = NULL,
            fm_args = list(standard.test     = "default",
                           scaled.test       = "default",
                           rmsea.ci.level    = 0.90,
                           rmsea.close.h0    = 0.05,
                           rmsea.notclose.h0 = 0.08,
                           robust            = TRUE,
```

```
                        cat.nonpd         = "na"),
            output = "vector", level = NULL, ...)
fitmeasures(object, fit_measures = "all",
            baseline_model = NULL, h1_model = NULL,
            fm_args = list(standard.test     = "default",
                        scaled.test       = "default",
                        rmsea.ci.level    = 0.90,
                        rmsea.close.h0    = 0.05,
                        rmsea.notclose.h0 = 0.08,
                        robust            = TRUE,
                        cat.nonpd         = "na"),
            output = "vector", level = 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. If additional options for certain fit measures are needed, fit_measures should be a list whose fit.measures (or fit_measures) element holds the measures wanted, with the options supplied as further named elements of the list. The standard.test option determines the main test statistic (chi-square value) that will be used to compute all the fit measures that depend on this test statistic. Usually this is "standard". FMG p-value tests can be selected here using names such as "peba4", "pols3", "pall", or "all". The scaled.test option determines which scaling method is to be used for the scaled fit measures (in case multiple scaling methods were requested). The rmsea.ci.level option, default 0.9, determines the level of the confidence interval for the rmsea value. The rmsea.close.h0 option, default 0.05, is the rmsea value that is used under the null hypothesis that rmsea <= rmsea.close.h0. The rmsea.notclose.h0 option, default 0.08, is the rmsea value that is used under the null hypothesis that rmsea >= rmsea.notclose.h0. The gfi.ci.level option, default 0.9, determines the level of the confidence interval for the (new) goodness of fit index gfi. The gfi index (Maydeu-Olivares et al., 2024) is computed from the reweighted least squares (RLS) statistic; the version based on the (standard) likelihood ratio statistic is also available, and the two can be requested by name as gfi_lrt and gfi_rls (the latter being identical to gfi), together with their .ci.lower, .ci.upper and .robust variants. The robust option, default TRUE, can be set to FALSE to avoid computing the so-called robust rmsea/cfi/gfi measures (for example if the computations take too long). The cat.nonpd option, default "na", is only used when data is categorical, and determines what happens to the robust values of RMSEA, CFI and GFI when an input (polychoric/tetrachoric) correlation matrix is not positive-definite (for at least one group): if "na", the robust values are NA; if "refit", the input correlation matrix is smoothed to be positive-definite, and the model parameters are re-estimated using estimator = "catML"; if "smooth", the input correlation matrix is smoothed, but the original (DWLS) parameter estimates are retained (this was the behavior in lavaan 0.6-13 and earlier). The older cat.check.pd = FALSE option is a deprecated alias

for `cat.nonpd = "refit"`.

`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 (e.g., CFI or TLI) will use the test statistics from this user-specified baseline model, instead of the default baseline model.

`h1_model` If not NULL, an object of class `lavaan`, representing a user-specified alternative to the default unrestricted model. If `h1_model` is provided, all fit indices calculated from chi-squared will use the chi-squared *difference* test statistics from `lavTestLRT`, which compare the user-provided `h1_model` to `object`.

`fm_args` List. Additional options for certain fit measures. DEPRECATED. The options should now be specified in the `"fit_measures"` argument!

`output` Character. If `"vector"` (the default), display the output as a named (lavaan-formatted) vector. If `"matrix"`, display the output as a 1-column matrix. If `"text"`, display the output using subsections and verbose descriptions. The latter is used in the summary output, and does not print the chi-square test by default. In addition, `fit_measures` should contain the main ingredient (for example `"rmsea"`) if related fit measures are requested (for example `"rmsea.ci.lower"`). Otherwise, nothing will be printed in that section. See the examples for how to add the chi-square test in the text output.

`level` Only used for multilevel models. If not `NULL`, either a level number (`1` or `2`) or a level name (`"within"` or `"between"`, or the level label used in the model syntax). The returned fit measures then reflect the fit of the model at that level only, while the model at the other level is saturated. This uses the so-called partially saturated model approach (Ryu & West, 2009): any misfit of the partially saturated model can be attributed to the target level. The incremental fit indices (e.g., CFI and TLI) are computed with respect to a level-specific baseline model: the independence model at the target level, again combined with a saturated model at the other level. By default, these partially saturated models are fitted (and stored) along with the original model; see the `fit.by.level` option in `lavOptions`. If they were not stored, they are re-fitted on the fly.

`...` Further arguments passed to or from other methods. Not currently used for `lavaan` objects.

Details

When a scaled (or robust) test statistic is requested (for example, by using `test = "satorra.bentler"`), the function will also return fit indices based on the scaled chi-square statistic, rather than the standard version. These scaled versions of fit measures, such as CFI and RMSEA, are calculated in the same way as their standard counterparts, with the key difference being that the scaled chi-square statistic is used in place of the regular one. In the output of `fitMeasures()`, these appear with the `.scaled` suffix, or in the `Scaled` column of the `summary()` output.

However, this substitution-based approach—used in SEM software for many years—has since been shown to be incorrect. Improved versions of robust fit indices have been proposed, offering better theoretical properties. Although still under development and not yet implemented for all estimation settings, these improved robust fit measures are provided when available. They appear with a `.robust` suffix in the output of `fitMeasures()`, or in the `Scaled` column of the `summary()` output on a row labeled `Robust`. As a general recommendation, these newer robust versions should be

used whenever available, in preference to the older scaled ones. See the references below for more details.

It is also worth noting that, for models involving ordered categorical data, robust fit indices are only computed if the underlying matrix of tetrachoric or polychoric correlations is positive definite. If this condition is not met—which is not uncommon in small samples—the robust measures are reported as NA.

Finally, in some situations (especially when the data contains missing values), computing these robust fit indices may be computationally intensive. To avoid long runtimes, the calculation of robust fit measures can be disabled by setting the robust argument to FALSE in the fit_measures list.

FMG p-values can be selected through the existing standard.test mechanism, for example fitMeasures(fit, "pvalue", fm_args = list(standard.test = "peba4")). No separate FMG-specific fit-measure names are needed.

For models fitted with sam using a local sam.method ("local", "fsr" or "cfsr"), the fit measures are computed for the step-2 structural model only, conditional on the (fixed) measurement model of step 1; loglikelihood-based measures (aic, bic, ...) are not available. If sam.method = "global", the fit measures are based on the joint model; since the parameters were estimated using a two-step procedure, they should be interpreted with caution.

Value

A named numeric vector of fit measures.

References

Ryu, E., & West, S. G. (2009). Level-specific evaluation of model fit in multilevel structural equation modeling. Structural Equation Modeling, 16(4), 583–601. doi:10.1080/10705510903203466

Brosseau-Liard, P. E., Savalei, V., & Li, L. (2012). An investigation of the sample performance of two nonnormality corrections for RMSEA. Multivariate behavioral research, 47(6), 904-930. doi:10.1080/00273171.2012.715252

Brosseau-Liard, P. E., & Savalei, V. (2014). Adjusting incremental fit indices for nonnormality. Multivariate behavioral research, 49(5), 460-470. doi:10.1080/00273171.2014.933697

Savalei, V. (2018). On the computation of the RMSEA and CFI from the mean-and-variance corrected test statistic with nonnormal data in SEM. Multivariate behavioral research, 53(3), 419-429. doi:10.1080/00273171.2018.1455142

Savalei, V. (2021). Improving fit indices in structural equation modeling with categorical data. Multivariate Behavioral Research, 56(3), 390-407. doi:10.1080/00273171.2020.1717922

Savalei, V., Brace, J. C., & Fouladi, R. T. (2023). We need to change how we compute RMSEA for nested model comparisons in structural equation modeling. Psychological Methods. doi:10.1037/met0000537

Zhang, X., & Savalei, V. (2023). New computations for RMSEA and CFI following FIML and TS estimation with missing data. Psychological Methods, 28(2), 263-283. doi:10.1037/met0000445

Foldnes, N., Moss, J., & Gronneberg, S. (2024). Improved goodness of fit procedures for structural equation models. *Structural Equation Modeling: A Multidisciplinary Journal*, 1-13. doi:10.1080/10705511.2024.2372028

Foldnes, N., Gronneberg, S., & Moss, J. (2026). Penalized eigenvalue block averaging: Extension to nested model comparison and Monte Carlo evaluations. *Behavior Research Methods, 58*(4). doi:10.3758/s13428026029684

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"))
fitMeasures(fit, c("chisq", "df", "pvalue", "cfi", "rmsea"),
            output = "matrix")
fitMeasures(fit, c("chisq", "df", "pvalue", "cfi", "rmsea"),
            output = "text")
fitMeasures(fit, "pvalue", fm_args = list(standard.test = "peba4"))
## specify another threshold for RMSEA confidence interval
fitMeasures(fit, list(
  fit.measures = c("cfi", "rmsea"),
  rmsea.ci.level = 0.95))

## fit a more restricted model
fit0 <- cfa(HS.model, data = HolzingerSwineford1939, orthogonal = TRUE)
## Calculate RMSEA_D (Savalei et al., 2023)
## See https://psycnet.apa.org/doi/10.1037/met0000537
fitMeasures(fit0, "rmsea", h1_model = fit)

## level-specific fit measures for a multilevel model (Ryu & West, 2009)
model <- '
    level: 1
        fw =~ y1 + y2 + y3
        fw ~ x1 + x2 + x3
    level: 2
        fb =~ y1 + y2 + y3
        fb ~ w1 + w2
'
fit2l <- sem(model, data = Demo.twolevel, cluster = "cluster")
fitMeasures(fit2l, c("chisq", "df", "pvalue", "cfi", "rmsea",
                     "srmr_within"), level = "within")
fitMeasures(fit2l, c("chisq", "df", "pvalue", "cfi", "rmsea",
                     "srmr_between"), level = "between")
```

`growth` *Fit Growth Curve Models*

Description

Fit a Growth Curve model. Only useful if all the latent variables in the model are growth factors. For more complex models, it may be better to use the lavaan function.

Usage

```
growth(model = NULL, data = NULL, ordered = NULL, aux = NULL, sampling_weights = NULL,
       sample_cov = NULL, sample_mean = NULL, sample_th = NULL,
       sample_nobs = NULL, group = NULL, cluster = NULL,
       constraints = "", wls_v = NULL, nacov = NULL, ov_order = "model",
       ...)
```

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 (e.g., the output of the lavParTable() 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.
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 data.frame.) Since 0.6-4, ordered can also be logical. If TRUE, all observed endogenous variables are treated as ordered (ordinal). If FALSE, all observed endogenous variables are considered to be numeric (again, unless they are declared as ordered in the data.frame.)
aux	Character vector. Names of auxiliary observed variables, used to make the missing-at-random (MAR) assumption more plausible under missing data (continuous data only). With missing = "ml" they are added to the model as saturated correlates (Graham, 2003), affecting the estimates and standard errors but hidden from the default output; with missing = "two.stage"/"robust.two.stage" they enter the first-stage EM moments. See the lavaan function for more details.
sampling_weights	A variable name in the data frame containing sampling weight information. Currently only available for non-clustered data. Depending on the sampling.weights.normalization option, these weights may be rescaled (or not) so that their sum equals the number of observations (total or per group).
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.
sample_mean	A sample mean vector. For a multiple group analysis, a list with a mean vector for each group.
sample_th	Vector of sample-based thresholds. For a multiple group analysis, a list with a vector of thresholds for each group.

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

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

`cluster` Character. A (single) variable name in the data frame defining the clusters in a two-level dataset.

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

`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.

`ov_order` Character. If `"model"` (the default), the order of the observed variable names (as reflected for example in the output of `lav_object_vnames()`) is determined by the model syntax. If `"data"`, the order is determined by the data (either the full data.frame or the sample (co)variance matrix). If the `wls_v` and/or `nacov` matrices are provided, this argument is currently set to `"data"`.

`...` Many more options can be specified, using 'name = value'. See `lavOptions` for a complete list.

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.efa = 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. doi:10.18637/jss.v048.i02

See Also

lavaan, estimator_iv

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)
```

---

lav_constraints *Utility Functions: Constraints*

Description

Utility functions for equality and inequality constraints.

Usage

```
lav_con_parse(partable = NULL, constraints = NULL, theta = NULL,
                      debug = FALSE)
lav_pt_con_ceq(partable, con = NULL, debug = FALSE,
                              txt_only = FALSE)
lav_pt_con_ciq(partable, con = NULL, debug = FALSE,
                              txt_only = FALSE)
lav_pt_con_def(partable, con = NULL, debug = FALSE,
                              txt_only = FALSE, warn = TRUE)
```

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.
con	An optional partable where the operator is one of '==', '>', '<' or ':='
txt_only	Logical. If TRUE, only the body of the function is returned as a character string. If FALSE, a function is returned.
warn	Logical. If FALSE, warnings are suppressed.

Details

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

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

The lav_pt_con_ceq function creates a function which takes the (unconstrained) parameter vector as input, and returns the slack values for each equality constraint. If the equality constraints hold perfectly, this function returns zeroes.

The lav_pt_con_ciq function creates a function which takes the (unconstrained) parameter vector as input, and returns the slack values for each inequality constraint.

The lav_pt_con_def function creates a function which takes the (unconstrained) parameter vector as input, and returns the computed values of the defined parameters.

Examples

```
myModel <- 'x1 ~ a*x2 + b*x3 + c*x4'
myParTable <- lavParTable(myModel, as_data_frame = FALSE)
con <- ' a == 2*b
         b - c == 5 '
conInfo <- lav_con_parse(myParTable, constraints = con)

myModel2 <- 'x1 ~ a*x2 + b*x3 + c*x4
             a == 2*b
```

```
            b - c == 5 '
ceq <- lav_pt_con_ceq(partable = lavParTable(myModel2))
ceq( c(2,3,4) )
```

---

lav_data *lavaan data functions*

Description

Utility functions related to the Data slot

Usage

```
# update data slot with new data (of the same size)
lav_data_update(lavdata = NULL, new_x = NULL, boot_idx = NULL, boot_clus = NULL,
                lavoptions = NULL, ...)
```

Arguments

lavdata	A lavdata object.
new_x	A list of (new) data matrices (per group) of the same size. They replace the data stored in the internal data slot.
boot_idx	A list of integers. If bootstrapping was used to produce the data in newX, use these indices to adapt the remaining slots.
boot_clus	A list of integer matrices (per group), or NULL. For a cluster bootstrap of two-level data, the (relabeled) cluster ids of the resampled rows, used to adapt the multilevel (Lp) slots.
lavoptions	A named list. The Options slot from a lavaan object.
...	To accept old argument names with dots or capitals. No other arguments are accepted.

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)

# extract data slot and options
lavdata <- fit@Data
lavoptions <- lavInspect(fit, "options")

# create bootstrap sample
boot.idx <- sample(x = nobs(fit), size = nobs(fit), replace = TRUE)
newX <- list(lavdata@X[[1]][boot.idx,])
```

```
# generate update lavdata object
newdata <- lav_data_update(lavdata = lavdata, new_x = newX,
                           lavoptions = lavoptions)
```

`lav_efalist_summary` *Summarizing EFA Fits*

Description

S3 summary and print methods for class `efaList`.

Usage

```
lav_efalist_summary(object,
        nd = 3L, cutoff = 0.3, dot_cutoff = 0.1, alpha_level = 0.01,
        lambda = TRUE, theta = TRUE, psi = TRUE, fit_table = TRUE,
        fs_determinacy = FALSE, eigenvalues = TRUE, sumsq_table = TRUE,
        lambda_structure = FALSE, se = FALSE, zstat = FALSE,
        pvalue = FALSE, ...)
lav_efalist_summary_print(x, nd = 3L, cutoff = 0.3, dot_cutoff = 0.1,
               alpha_level = 0.01, ...)
```

Arguments

`object`	An object of class `efaList`, usually, a result of a call to `efa` with (the default) `output = "efa"`.
`x`	An object of class `lav_efalist_summary`, usually, a result of a call to `lav_efalist_summary`.
`nd`	Integer. The number of digits that are printed after the decimal point in the output.
`cutoff`	Numeric. Factor loadings smaller than this value (in absolute value) are not printed (even if they are significantly different from zero). The idea is that only medium to large factor loadings are printed, to better see the overall structure.
`dot_cutoff`	Numeric. Factor loadings larger (in absolute value) than this value, but smaller (in absolute value) than the cutoff value are shown as a dot. They represent small loadings that may still need your attention.
`alpha_level`	Numeric. If the p-value of a factor loading is smaller than this value, a significance star is printed to the right of the factor loading. To switch this off, use `alpha_level = 0`.
`lambda`	Logical. If `TRUE`, include the (standardized) factor loadings in the summary.
`theta`	Logical. If `TRUE`, include the unique variances and the communalities in the table of factor loadings.
`psi`	Logical. If `TRUE`, include the factor correlations in the summary. Ignored if only a single factor is used.
`fit_table`	Logical. If `TRUE`, show fit information for each model.

`fs_determinacy` Logical. If `TRUE`, show the factor score determinacy values per factor (assuming regression factor scores are used) and their squared values.

`eigenvalues` Logical. If `TRUE`, include the eigenvalues of the sample variance-covariance matrix in the summary.

`sumsq_table` Logical. If `TRUE`, include a table including sums of squares of factor loadings (and related measures) in the summary. The sums of squares are computed as the diagonal elements of Lambda times Psi (where Psi is the matrix of factor correlations). If orthogonal rotation was used, Psi is diagonal and the sums of squares are identical to the sums of the squared column elements of the Lambda matrix (i.e., the factor loadings). This is no longer the case when oblique rotation has been used. But in both cases (orthogonal or oblique), the (total) sum of the sums of squares equals the sum of the communalities. In the second row of the table (Proportion of total), the sums of squares are divided by the total. In the third row of the table (Proportion var), the sums of squares are divided by the number of items.

`lambda_structure` Logical. If `TRUE`, show the structure matrix (i.e., the factor loadings multiplied by the factor correlations).

`se` Logical. If `TRUE`, include the standard errors of the standardized lambda, theta and psi elements in the summary.

`zstat` Logical. If `TRUE`, include the Z-statistics of the standardized lambda, theta and psi elements in the summary.

`pvalue` Logical. If `TRUE`, include the P-values of the standardized lambda, theta and psi elements in the summary.

`...` Further arguments passed to or from other methods.

Value

The function `lav_efalist_summary` computes and returns a list of summary statistics for the list of EFA models in `object`.

Examples

```
## The famous Holzinger and Swineford (1939) example
fit <- efa(data = HolzingerSwineford1939,
           ov_names = paste("x", 1:9, sep = ""),
           nfactors = 1:3,
           rotation = list("geomin", geomin_epsilon = 0.01, rstarts = 1))
summary(fit, nd = 3L, cutoff = 0.2, dot_cutoff = 0.05,
        lambda_structure = TRUE, pvalue = TRUE)
```

`lav_export_estimation` *lav_export_estimation*

Description

lavaan provides a range of optimization methods with the optim.method argument (nlminb, BFGS, L-BFGS-B, GN, and nlminb_constr). 'lav_export_estimation' allows exporting objects and functions necessary to pass a lavaan model into any optimizer that takes a combination of (1) starting values, (2) fit-function, (3) gradient-function, and (4) upper and lower bounds. This allows testing new optimization frameworks.

Usage

```
lav_export_estimation(lavaan_model)
```

Arguments

`lavaan_model`	a fitted lavaan model

Value

List with:

- get_coef - When equality constraints are present, lavaan applies internal transformations. get_coef is a function that reconstructs the output of the coef function for the parameters.
- starting_values - starting_values to be used in the optimization
- objective_function - objective function, expecting the current parameter values and the lavaan model
- gradient_function - gradient function, expecting the current parameter values and the lavaan model
- lower - lower bounds for parameters
- upper - upper bound for parameters

Examples

```
library(lavaan)
model <- '
  # latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + y2 + y3 + y4
     dem65 =~ y5 + a*y6 + y7 + y8

  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60
'
```

```
fit <- sem(model,
           data = PoliticalDemocracy,
           do.fit = FALSE)

est <- lav_export_estimation(lavaan_model = fit)

# The starting values are:
est$starting_values
# Note that these do not have labels (and may also differ from coef(fit)
# in case of equality constraints):
coef(fit)
# To get the same parameters, use:
est$get_coef(parameter_values = est$starting_values,
             lavaan_model = fit)

# The objective function can be used to compute the fit at the current estimates:
est$objective_function(parameter_values = est$starting_values,
                       lavaan_model = fit)

# The gradient function can be used to compute the gradients at the current estimates:
est$gradient_function(parameter_values = est$starting_values,
                      lavaan_model = fit)

# Together, these elements provide the means to estimate the parameters with a large
# range of optimizers. For simplicity, here is an example using optim:
est_fit <- optim(par = est$starting_values,
                 fn = est$objective_function,
                 gr = est$gradient_function,
                 lavaan_model = fit,
                 method = "BFGS")
est$get_coef(parameter_values = est_fit$par,
             lavaan_model = fit)

# This is identical to
coef(sem(model,
         data = PoliticalDemocracy))

# Example using ridge regularization for parameter a
fn_ridge <- function(parameter_values, lavaan_model, est, lambda){
  return(est$objective_function(parameter_values = parameter_values,
                           lavaan_model = lavaan_model) + lambda * parameter_values[6]^2)
}
ridge_fit <- optim(par = est$get_coef(est$starting_values,
                                      lavaan_model = fit),
                   fn = fn_ridge,
                   lavaan_model = fit,
                   est = est,
                   lambda = 10)
est$get_coef(parameter_values = ridge_fit$par,
             lavaan_model = fit)
```

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_grad_complex(func, x, h = .Machine$double.eps, ...,
                          fallback_simple = TRUE)
lav_func_jacobian_complex(func, x, h = .Machine$double.eps, ...,
                          fallback_simple = TRUE)

lav_func_grad_simple(func, x, h = sqrt(.Machine$double.eps), ...)
lav_func_jacobian_simple(func, x, h = sqrt(.Machine$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'.
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 yet part of the numDeriv package. They were used internally, and made public in 0.5-17 at the 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_grad_complex(func = exp, x = 1) - exp(1)

# less accurate forward method
lav_func_grad_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_getcov *Utility Functions For Covariance Matrices*

Description

Convenience functions to deal with covariance and correlation matrices.

Usage

```
lav_getcov(x, lower = TRUE, diagonal = TRUE, sds = NULL,
       names = paste("V", 1:nvar, sep=""))
getCov(x, lower = TRUE, diagonal = TRUE, sds = NULL,
       names = paste("V", 1:nvar, sep=""))
lav_char2num(s)
char2num(s)
lav_cor2cov(r, sds, names = NULL, ...)
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 lav_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 assume 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; commas and semi-colons are ignored.
r	A correlation matrix, to be scaled into a covariance matrix.
...	To accept old argument name R (is now r). No other arguments are accepted.

Details

The lav_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 lav_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 the 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 <- lav_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)
```

---

lav_label_code *Provides a formatted label for use in tikz, svg or Rplot*

Description

Creates the code used to show the label in tikz, svg or R. The label is plotted in an xy-plot if show = TRUE.

Usage

```
lav_label_code(label = "", value = "", show =FALSE,
               idx_font_size = 20L, dy = 7L,
               italic = TRUE, auto_subscript = TRUE)
```

Arguments

label	A character string in one of the formats • *name* • *name_index* • *name=value* • *name_index=value* If *value* is specified in parameter label and parameter value is empty, the value in label will be used.
value	A character string specifying a value or empty.
show	A logical indicating that the result should be plotted in an Rplot.
idx_font_size	An integer specifying font size to use for the subscript in svg.
dy	An integer specifying the distance to move the baseline of the subscript in svg.
italic	A logical indicating whether the label's font should be italic. Only used for tikz output and in the displayed Rplot when the result is not an expression.
auto_subscript	Logical, if TRUE and label starts with one or more alphabetic chars followed by one or more digits, an underscore will be inserted between these parts.

Details

If both label and value are empty, the resulting codes are also empty and nothing will be shown.

If label is empty and value is not, processing is done as if label was specified and value was empty.

If label contains the string "1van", the label value is set to "1". This allows distinct names for regression intercepts while still labeling them as "1".

If *name* in label is a Greek character or varepsilon, the function attempts to generate code that shows the Greek symbol. If label contains an *index* part, the function attempts to generate code that displays this value as a subscript. In the svg code, the values idx_font_size and dy are used for the subscript. If a *value* is present, the function attempts to show this value after the label, with an equal sign between the two.

Value

a list with members svg, tikz and r, giving the result.

Examples

```
lav_label_code("x3")
lav_label_code("beta10", 0.65, show = TRUE)
lav_label_code("A_i,j=0.45", show = TRUE)
lav_label_code("Gamma")
lav_label_code(value="1.2345")
```

`lav_long_names` *Deprecated functions with long names*

Description

Functions that are deprecated following the adoption of shorter names in May 2026.

Usage

```
lav_matrix_duplication_ginv_pre_post(A = matrix(0, 0, 0))
lav_matrix_orthogonal_complement(A = matrix(0, 0, 0))
lav_func_gradient_complex(
                          func,
                          x,
                          h = .Machine$double.eps,
                          ...,
                          fallback.simple = TRUE)
lav_matrix_duplication_ginv_post(A = matrix(0, 0, 0))
lav_func_gradient_simple(
                         func,
                         x,
                         h = sqrt(.Machine$double.eps),
                         ...)
lav_matrix_vech_row_idx(n = 1L, diagonal = TRUE)
lav_matrix_vech_col_idx(n = 1L, diagonal = TRUE)
lav_matrix_antidiag_idx(n = 1L)
lav_matrix_duplication_pre_post(A = matrix(0, 0, 0))
lav_matrix_duplication_ginv_pre(A = matrix(0, 0, 0))
lav_matrix_commutation_pre_post(A = matrix(0, 0, 0))
lav_partable_unrestricted(
                          lavobject = NULL,
                          lavdata = NULL,
                          lavpta = NULL,
                          lavoptions = NULL,
                          lavsamplestats = NULL,
                          lavh1 = NULL,
                          sample.cov = NULL,
                          sample.mean = NULL,
                          sample.slopes = NULL,
                          sample.th = NULL,
```

```
                          sample.th.idx = NULL,
                          sample.cov.x = NULL,
                          sample.mean.x = NULL)
lav_partable_independence(
                          lavobject = NULL,
                          lavdata = NULL,
                          lavpta = NULL,
                          lavoptions = NULL,
                          lavsamplestats = NULL,
                          lavh1 = NULL,
                          sample.cov = NULL,
                          sample.mean = NULL,
                          sample.slopes = NULL,
                          sample.th = NULL,
                          sample.th.idx = NULL,
                          sample.cov.x = NULL,
                          sample.mean.x = NULL)
lav_matrix_vechru_idx(n = 1L, diagonal = TRUE)
lav_matrix_upper2full(x, diagonal = TRUE)
lav_matrix_lower2full(x, diagonal = TRUE)
lav_matrix_commutation_mn_pre(
                              A,
                              m = 1L,
                              n = 1L)
lav_samplestats_from_data(
                          lavdata = NULL,
                          lavoptions = NULL,
                          WLS.V = NULL,
                          NACOV = NULL)
lav_matrix_vechru_reverse(x, diagonal = TRUE)
lav_matrix_vechr_idx(n = 1L, diagonal = TRUE)
lav_matrix_vechu_idx(n = 1L, diagonal = TRUE)
lav_matrix_diagh_idx(n = 1L)
lav_partable_attributes(partable, pta = NULL)
lav_matrix_vechr_reverse(x, diagonal = TRUE)
lav_matrix_vechu_reverse(x, diagonal = TRUE)
lav_matrix_vech_idx(n = 1L, diagonal = TRUE)
lav_matrix_diag_idx(n = 1L)
lav_matrix_duplication_post(A = matrix(0, 0, 0))
lav_matrix_duplication_ginv(n = 1L)
lav_matrix_commutation_post(A = matrix(0, 0, 0))
lav_matrix_symmetric_sqrt(S = matrix(0, 0, 0))
lav_matrix_vech_reverse(x, diagonal = TRUE)
lav_matrix_duplication_pre(A = matrix(0, 0, 0))
lav_matrix_commutation_pre(A = matrix(0, 0, 0))
lav_partable_complete(partable = NULL, start = TRUE)
lav_matrix_vechru(S, diagonal = TRUE)
lav_partable_constraints_def(
```

```
                              partable,
                              con = NULL,
                              debug = FALSE,
                              txtOnly = FALSE,
                              warn = TRUE)
    lav_partable_constraints_ceq(
                              partable,
                              con = NULL,
                              debug = FALSE,
                              txtOnly = FALSE)
    lav_partable_constraints_ciq(
                              partable,
                              con = NULL,
                              debug = FALSE,
                              txtOnly = FALSE)
    lav_partable_from_lm(
                         object,
                         est = FALSE,
                         label = FALSE,
                         as.data.frame. = FALSE)
    lav_constraints_parse(
                          partable = NULL,
                          constraints = NULL,
                          theta = NULL,
                          debug = FALSE)
    lav_matrix_vechr(S, diagonal = TRUE)
    lav_matrix_vechu(S, diagonal = TRUE)
    lav_matrix_bdiag(...)
    lav_matrix_trace(..., check = TRUE)
    lav_partable_labels(
                        partable,
                        blocks = c("group", "level"),
                        group.equal = "",
                        group.partial = "",
                        type = "user")
    lav_matrix_vecr(A)
    lav_matrix_vech(S, diagonal = TRUE)
    lav_partable_merge(
                       pt1 = NULL,
                       pt2 = NULL,
                       remove.duplicated = FALSE,
                       fromLast = FALSE,
                       warn = TRUE)
    lav_matrix_vec(A)
    lav_matrix_duplication(n = 1L)
    lav_matrix_commutation(m = 1L, n = 1L)
    lav_matrix_cov(Y, Mu = NULL)
    lav_partable_ndat(partable)
```

```
lav_partable_npar(partable)
lav_partable_add(partable = NULL, add = list())
lav_partable_df(partable)
```

Arguments

...	See argument ... in the corresponding function with shortened name.
A	See argument a in the corresponding function with shortened name.
add	See argument add in the corresponding function with shortened name.
as.data.frame.	See argument as_data_frame in the corresponding function with shortened name.
blocks	See argument blocks in the corresponding function with shortened name.
check	See argument check in the corresponding function with shortened name.
con	See argument con in the corresponding function with shortened name.
constraints	See argument constraints in the corresponding function with shortened name.
debug	See argument debug in the corresponding function with shortened name.
diagonal	See argument diagonal in the corresponding function with shortened name.
est	See argument est in the corresponding function with shortened name.
fallback.simple	See argument fallback_simple in the corresponding function with shortened name.
fromLast	See argument from_last in the corresponding function with shortened name.
func	See argument func in the corresponding function with shortened name.
group.equal	See argument group_equal in the corresponding function with shortened name.
group.partial	See argument group_partial in the corresponding function with shortened name.
h	See argument h in the corresponding function with shortened name.
label	See argument label in the corresponding function with shortened name.
lavdata	See argument lavdata in the corresponding function with shortened name.
lavh1	See argument lavh1 in the corresponding function with shortened name.
lavobject	See argument lavobject in the corresponding function with shortened name.
lavoptions	See argument lavoptions in the corresponding function with shortened name.
lavpta	See argument lavpta in the corresponding function with shortened name.
lavsamplestats	See argument lavsamplestats in the corresponding function with shortened name.
m	See argument m in the corresponding function with shortened name.
Mu	See argument mu in the corresponding function with shortened name.
n	See argument n in the corresponding function with shortened name.
NACOV	See argument nacov in the corresponding function with shortened name.
object	See argument object in the corresponding function with shortened name.

partable	See argument partable in the corresponding function with shortened name.
pt1	See argument pt1 in the corresponding function with shortened name.
pt2	See argument pt2 in the corresponding function with shortened name.
pta	See argument pta in the corresponding function with shortened name.
remove.duplicated	See argument remove_duplicated in the corresponding function with shortened name.
S	See argument s in the corresponding function with shortened name.
sample.cov	See argument sample_cov in the corresponding function with shortened name.
sample.cov.x	See argument sample_cov_x in the corresponding function with shortened name.
sample.mean	See argument sample_mean in the corresponding function with shortened name.
sample.mean.x	See argument sample_mean_x in the corresponding function with shortened name.
sample.slopes	See argument sample_slopes in the corresponding function with shortened name.
sample.th	See argument sample_th in the corresponding function with shortened name.
sample.th.idx	See argument sample_th_idx in the corresponding function with shortened name.
start	See argument start in the corresponding function with shortened name.
theta	See argument theta in the corresponding function with shortened name.
txtOnly	See argument txt_only in the corresponding function with shortened name.
type	See argument type in the corresponding function with shortened name.
warn	See argument warn in the corresponding function with shortened name.
WLS.V	See argument wls_v in the corresponding function with shortened name.
x	See argument x in the corresponding function with shortened name.
Y	See argument y in the corresponding function with shortened name.

Details

Function names are shortened by the following replacements:

_mvnorm _mvn

_cluster _cl

_missing _mi

_matrix _mat

_duplication _dup

_commutation _com

_information _info

_samplestats _samp

_satorra_bentler _sb

_SatorraBentler _sb

_yuan_bentler _yb

_inverted _inv

_estimate _est

_symmetric _sym

_lavaan_step _step

_univariate _uni

_fit_measures _fit

_constraints _con

_scores _sc

_reverse _rev

_vech_sigma _sigma

_vechsigma _sigma

_object_inspect _inspect

_orthogonal _ortho

_gradient _grad

_partable _pt

`lav_matrix` *Utility Functions: Matrices and Vectors*

Description

Utility functions for Matrix and Vector operations.

Usage

```
# matrix to vector
lav_mat_vec(a)
lav_mat_vecr(a)
lav_mat_vech(s, diagonal = TRUE)
lav_mat_vechr(s, diagonal = TRUE)

# matrix/vector indices
lav_mat_vech_idx(n = 1L, diagonal = TRUE)
lav_mat_vech_row_idx(n = 1L, diagonal = TRUE)
lav_mat_vech_col_idx(n = 1L, diagonal = TRUE)
lav_mat_vechr_idx(n = 1L, diagonal = TRUE)
lav_mat_vechru_idx(n = 1L, diagonal = TRUE)
lav_mat_diag_idx(n = 1L)
lav_mat_diagh_idx(n = 1L)
lav_mat_antidiag_idx(n = 1L)
```

```
# vector to matrix
lav_mat_vech_rev(x, diagonal = TRUE)
lav_mat_vechru_rev(x, diagonal = TRUE)
lav_mat_upper2full(x, diagonal = TRUE)
lav_mat_vechr_rev(x, diagonal = TRUE)
lav_mat_vechu_rev(x, diagonal = TRUE)
lav_mat_lower2full(x, diagonal = TRUE)

# the duplication matrix
lav_mat_dup(n = 1L)
lav_mat_dup_pre(a = matrix(0,0,0))
lav_mat_dup_post(a = matrix(0,0,0))
lav_mat_dup_pre_post(a = matrix(0,0,0))
lav_mat_dup_ginv(n = 1L)
lav_mat_dup_ginv_pre(a = matrix(0,0,0))
lav_mat_dup_ginv_post(a = matrix(0,0,0))
lav_mat_dup_ginv_pre_post(a = matrix(0,0,0))

# the commutation matrix
lav_mat_com(m = 1L, n = 1L)
lav_mat_com_pre(a = matrix(0,0,0))
lav_mat_com_post(a = matrix(0,0,0))
lav_mat_com_pre_post(a = matrix(0,0,0))
lav_mat_com_mn_pre(a, m = 1L, n = 1L)

# sample statistics
lav_mat_cov(y, mu = NULL)

# other matrix operations
lav_mat_sym_sqrt(s = matrix(0,0,0))
lav_mat_ortho_complement(a = matrix(0,0,0))
lav_mat_bdiag(...)
lav_mat_trace(..., check = TRUE)
```

Arguments

a	A general matrix.
s	A symmetric matrix.
y	A matrix representing a (numeric) dataset.
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 columns of the matrix.
m	Integer. The number of rows of a matrix.
x	Numeric. A vector.
mu	Numeric. If given, use mu (instead of sample mean) to center, before taking the crossproduct.

...	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 and vector functions that are used throughout the lavaan code. They are made public at the request of package developers. Below is a brief description of what they do:

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

The lav_mat_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_mat_vechr function is similar to the lav_mat_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_mat_vech_idx function returns the vector indices of the lower triangular elements of a symmetric matrix of size n, column by column.

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

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

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

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

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

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

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

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

The lav_mat_vech_rev function (alias: lav_mat_vechru_rev and lav_mat_upper2full) creates a symmetric matrix, given only upper triangular elements, row by row. If diagonal = FALSE, a diagonal with zero elements is added.

The lav_mat_vechr_rev (alias: lav_mat_vechu_rev and lav_mat_lower2full) creates a symmetric matrix, given only the lower triangular elements, row by row. If diagonal = FALSE, a diagonal with zero elements is added.

The lav_mat_dup 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_mat_dup_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_mat_dup_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_mat_dup_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.

The lav_mat_dup_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_mat_dup_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_mat_dup_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_mat_dup_ginv_pre_post function first pre-multiplies a matrix A with the transpose of the generalized inverse of the duplication matrix, and then 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_mat_com function computes the commutation matrix, a permutation matrix that transforms vec(A) (with m rows and n columns) into vec(t(A)).

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

The lav_mat_com_pre_post function first pre-multiplies a matrix A with the commutation matrix, and then post multiplies the result again with the commutation matrix, without explicitly creating the commutation matrix. The matrix A must have n*n rows, where n is an integer.

The lav_mat_com_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_mat_cov function computes the sample covariance matrix of its input matrix, where the elements are divided by N (the number of rows).

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

The lav_mat_bdiag function constructs a block diagonal matrix from its arguments.

The lav_mat_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_mat_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_mat_dup_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)
lav_model_get_parameters(lavmodel, glist = NULL, type = "free",
                         extra = TRUE, ...)

# compute model-implied statistics
lav_model_implied(lavmodel, glist = NULL, delta = TRUE, ...)

# compute standard errors
lav_model_vcov_se(lavmodel, lavpartable, vcov = NULL, boot = NULL,
                  mc = NULL, lavoptions = NULL, ...)
```

Arguments

lavmodel	An internal representation of a lavaan model.
x	Numeric. A vector containing the values of all the free model parameters.
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 ">".
delta	Logical. If TRUE, and a Delta matrix is present in GLIST, use the (diagonal) values of the Delta matrix to rescale the covariance matrix. This is usually needed in the categorical setting to convert covariances to correlations.
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).
mc	Numeric matrix containing the Monte-Carlo based parameter estimates (in the columns) for each Monte-Carlo sample (in the rows).
lavoptions	A named list. The Options slot from a lavaan object.
...	To accept old argument names with dots or capitals. No other arguments are accepted.

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_model_plotinfo *Get model information*

Description

Extracts the information from a model that is needed to produce a plot.

Usage

```
lav_model_plotinfo(model = NULL, infile = NULL, varlv = FALSE)
```

Arguments

model	A character vector specifying the model in lavaan syntax or a list (or data.frame) with at least members lhs, op, rhs, label and fixed or a fitted lavaan object (in which case the ParTable object is extracted and column est is used as value to show). Should be NULL if infile is given.
infile	A character string specifying the file that contains the model syntax.
varlv	A logical indicating that the (residual) variance of a variable should be plotted as a separate latent variable (with a smaller circle than ordinary latent variables). In this case, a covariance between two such variables is plotted as a covariance between their variance latent variables.

Value

A structure 'plotinfo', which is a list with members nodes and edges. These are data.frames containing the data needed to create a diagram.

1. nodes

   id character, identification of the node consisting of blok and naam.

   naam character, name of the node as specified in the model. For intercepts the name is "1vanXXXX", with XXXX the name of the regressed variable.

   tiepe character, type of node: ov (observed variable), lv (latent variable), varlv (variance as latent variable), cv (composite variable), wov (within level variable in multilevel model), bov (between level variable in multilevel model), const (intercept of regression).

   blok integer, level (0 if not a multilevel model).

2. edges

   id integer, autoincrement identification of the edge.

   label character, label for the edge, made from the label specified in the model and the fixed (or estimated) value if present.

   van character, id of the starting node.

   naar character, id of the destination node.

   tiepe character, lavaan operator, with two exceptions: a (residual) variance is coded here as '~~~', and a regression introduced by varlv = TRUE is coded as '~.'.

Examples

```
model <- 'alpha  =~ 1 * x1 + x2 + x3        # latent variable
          beta <~ x4 + x5 + x6              # composite
          gamma =~ 1 * x7 + x8 + x9         # latent variable
          Xi =~ 1 * x10 + x11 + x12 + x13   # latent variable
          # regressions
          Xi ~ v * alpha + t * beta + cc * 1
          alpha ~ tt * beta + ss * gamma + yy * Theta1
          # variances and covariances
```

```
            x2 ~~ cc25 * x5
            x3 ~~ cc36 * x6
            x3 ~~ cc34 * x4
            gamma ~~ 0.55 * gamma
            '
(test <- lav_model_plotinfo(model))
```

`lav_mplus_lavaan` *mplus to lavaan converter*

Description

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

Usage

```
lav_mplus_lavaan(inpfile, run = TRUE)
mplus2lavaan(inpfile, run = TRUE)
```

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.
`run`	Whether to run the specified Mplus input syntax (`TRUE`) or only to parse and convert the syntax (`FALSE`).

Value

A `lavaan` object with the fitted results of the Mplus model. The parsed and converted Mplus syntax is preserved in the `@external` slot of the `lavaan` object in the `$mplus.inp` element. If `run` is `FALSE`, a `list` of converted syntax is returned.

Author(s)

Michael Hallquist

See Also

`lavExport`

Examples

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

## End(Not run)
```

`lav_mplus_syntax_model` *Convert Mplus model syntax to lavaan*

Description

Converts Mplus model syntax into lavaan model syntax.

Usage

```
lav_mplus_syntax_model(syntax)
mplus2lavaan.modelSyntax(syntax)
```

Arguments

`syntax`	A character vector containing Mplus model syntax to be converted to lavaan model syntax. Note that parsing Mplus syntax often requires correct usage of newline characters. If `syntax` is a vector of multiple strings, these will be joined with newlines prior to conversion. Alternatively, `\n` characters can be included inline in `syntax`.

Value

A character string of converted `lavaan` model syntax.

Author(s)

Michael Hallquist

See Also

`lav_mplus_lavaan`

Examples

```
## Not run:
syntax <- '
    f1 BY x1*1 x2 x3;
    x1 WITH x2;
    x3 (1);
    x2 (1);
'
lavSyntax <- lav_mplus_syntax_model(syntax)
cat(lavSyntax)

## End(Not run)
```

lav_partable *lavaan partable functions*

Description

Utility functions related to the parameter table (partable)

Usage

```
# extract information from a parameter table
lav_pt_df(partable)
lav_pt_ndat(partable)
lav_pt_npar(partable)
lav_pt_ngroups(partable)
lav_pt_attributes(partable, pta = NULL)

# generate parameter labels
lav_pt_labels(partable, blocks = c("group", "level"),
                    group_equal = "", group_partial = "", type = "user")

# generate parameter table for specific models
lav_pt_independence(lavobject = NULL, lavdata = NULL,
    lavpta = NULL, lavoptions = NULL, lavsamplestats = NULL,
    lavh1 = NULL,
    sample_cov = NULL, sample_mean = NULL, sample_slopes = NULL,
    sample_th = NULL, sample_th_idx = NULL,
    sample_cov_x = NULL, sample_mean_x = NULL)

lav_pt_unrestricted(lavobject = NULL, lavdata = NULL,
    lavpta = NULL, lavoptions = NULL, lavsamplestats = NULL,
    lavh1 = NULL,
    sample_cov = NULL, sample_mean = NULL, sample_slopes = NULL,
    sample_th = NULL, sample_th_idx = NULL,
    sample_cov_x = NULL, sample_mean_x = NULL)

lav_pt_from_lm(object, est = FALSE, label = FALSE,
    as_data_frame = FALSE)

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

# merge two parameter tables
lav_pt_merge(pt1 = NULL, pt2 = NULL, remove_duplicated = FALSE,
    from_last = FALSE, warn = TRUE)

# add a single parameter to an existing parameter table
lav_pt_add(partable = NULL, add = list())
```

Arguments

partable	A parameter table. See lavParTable for more information.
blocks	Character vector. Which columns in the parameter table should be taken to distinguish between different blocks of parameters (and hence be given different labels)? If "blocks" includes "group", a suffix ".g" and the group number (or group label) is added for the parameters of all but the first group. If "blocks" includes "level", a suffix ".l" and the level number is added for the parameters of all but the first level. If "blocks" includes, say "foo", a suffix ".foo" and the corresponding value of "foo" is added to all parameters.
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. Either 'user' to select all entries, or 'free' to select only the free parameters from the parameter table.
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 named list. The Options slot from a lavaan object.
lavsamplestats	An object of class 'lavSampleStats'. The SampleStats slot from a lavaan object.
lavh1	A named list. The h1 slot from a lavaan object.
lavpta	The pta (parameter table attributes) 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.
sample_cov_x	Optional list of numeric matrices. Each list element contains a sample variance-covariance matrix for the exogenous variables for this group (only used when conditional.x = TRUE). If provided, these values will be used as starting values.
sample_mean_x	Optional list of numeric vectors. Each list element contains a sample mean vector for the exogenous variables for this group (only used when conditional.x = TRUE). If provided, these values will be used as starting values.
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.

from_last Logical. If TRUE, duplicated elements are considered from the bottom of the merged parameter table.

warn Logical. If TRUE, a warning is produced when duplicated elements are found, when merging two parameter tables.

add A named list. A single row of a parameter table as a named list.

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_pt_independence(fit)
as.data.frame(lav, stringsAsFactors = FALSE)

# how many free parameters?
lav_pt_npar(lav)

# how many sample statistics?
lav_pt_ndat(lav)
```

---

lav_plot *Create diagram code for tikz or svg, or produce an R plot*

Description

Creates code to draw a diagram in tikz or svg, plots the diagram, or stores the diagram in a png file.

Usage

```
lav_plot(model = NULL,
        infile = NULL,
        varlv = FALSE,
        placenodes = NULL,
        edgelabelsbelow = NULL,
        group_covar_indicators = FALSE,
        common_opts = list(sloped_labels = TRUE,
                           mlovcolors = c("lightgreen", "lightblue"),
                           italic = TRUE,
                           lightness = 1,
                           auto_subscript = TRUE),
        rplot = list(outfile = "",
                     addgrid = TRUE),
        tikz = list(outfile = "",
                    cex = 1.3,
                    standalone = FALSE),
        svg = list(outfile = "",
                   stroke_width = 2L,
                   font_size = 20L,
                   idx_font_size = 15L,
                   dy = 5L,
                   font_family = "Latin Modern Math, arial, Arial, sans",
                   standalone = FALSE)
        )
```

Arguments

model	A character vector specifying the model in lavaan syntax or a list (or data.frame) with at least members lhs, op, rhs, label and fixed or a fitted lavaan object (in which case the ParTable object is extracted and column est is used as value to show). Should be NULL if infile is given.
infile	A character string specifying the file that contains the model syntax.
varlv	A logical indicating that the (residual) variance of a variable should be plotted as a separate latent variable (with a smaller circle than ordinary latent variables). In this case, a covariance between two such variables is plotted as a covariance between their variance latent variables.
placenodes	optional list with members *nodename* = c(row, column), row and column don't have to be integers.
edgelabelsbelow	optional list with members c(*nodename1* , *nodename2*).
group_covar_indicators	logical, should items with indicators which have an explicit covariance link be placed in the same group, i.e. forced to be on the same side of the diagram?
common_opts	options common to the three types of generated plots.
rplot	options for creating Rplot, see lav_plotinfo_rgraph.

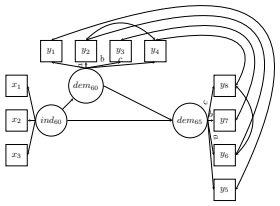
`tikz` options for creating code for tikz plot, see `lav_plotinfo_tikzcode`.

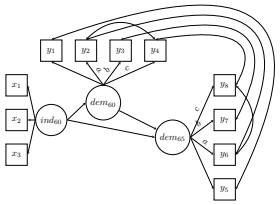
`svg` options for creating code for svg plot, see `lav_plotinfo_svgcode`.

Details

If `rplot` is specified, or if neither `tikz` nor `svg` is specified, an R plot is generated, and stored in a png file if the outfile member of `rplot` is set. If `tikz` is specified, the code for a tikz diagram is stored in the specified `outfile`; the same applies to svg.

The lav_plot command **tries** to create a nice plot from the input model, but the variable names should be kept short, and the nodes in the plot may sometimes need to be rearranged. As an example, the (slightly modified) example of the `sem` function in `lavaan` produces the first plot shown below. Using the placenodes argument produces the second plot.





1

More details on the parameters can be found in the help for the `lav_...` functions.

Value

NULL, invisible

See Also

lav_model_plotinfo, lav_plotinfo_positions, lav_plotinfo_rgraph, lav_plotinfo_tikzcode, lav_plotinfo_svgcode

Examples

```
model <- 'alpha11  =~ 1 * x1 + x2 + x3     # latent variable
          alpha12 <~ x4 + x5 + x6          # composite
          gamma =~ 1 * x7 + x8 + x9        # latent variable
          xi =~ 1 * x10 + x11 + x12 + x13  # latent variable
          x1 ~~ x3
          x2 ~~ epsilon2 * x2
          x12 ~~ epsilon12 * x12
          x4 ~~ epsilon4 * x4
          x7 ~~ x9
          x10 ~~ x11 + x13
          gamma ~~ 0.7 * xi
          # regressions
          xi ~ v * alpha11 + t * alpha12 + 1
```

```
          alpha11 ~ yy * Theta1 + tt1 * 0.12 * alpha12 + ss * gamma
          Theta1 ~~ alpha12
          '
lav_plot(model)
lav_plot(model,
        placenodes=list(Theta1 = c(2, 2.5)),
        tikz = list(outfile=stdout()))
modelml <- '
        level: 1
        fw =~ y_1 + y_2 + y_3 + y_4
        level: 2
        fb =~ y_1 + y_2 + y_3 + y_5
        y_2 ~~ cv24 * y_5
  '
tikzcodeml <- lav_plot(modelml,
              common_opts = list(auto_subscript = FALSE),
              svg = list(outfile=stdout())
              )

## Not run:
# example creating tex file with the above models
zz <- file("testtikz.tex", open="w")
writeLines(c(
  '\documentclass{article}',
  '\usepackage{amsmath, amssymb}',
  '\usepackage{amsfonts}',
  '\usepackage[utf8]{inputenc}',
  '\usepackage[english]{babel}',
  '\usepackage{xcolor}',
  '\usepackage{color}',
  '\usepackage{tikz}',
  '\usetikzlibrary {shapes.geometric}',
  '\begin{document}'),
  zz)
lav_plot(model,
        tikz = list(outfile = "tmp.tex")
        )
tmp <- readLines("tmp.tex")
writeLines(tmp, zz)
writeLines(" ", zz)
lav_plot(modelml,
        common_opts = list(sloped_labels = FALSE,
                           mlovcolors = c("lightgreen", "lightblue")),
        tikz = list(outfile = "tmp.tex", cex = 1.4)
        )
tmp <- readLines("tmp.tex")
writeLines(tmp, zz)
writeLines("\end{document}", zz)
close(zz)
openPDF <- function(f) {
  os <- .Platform$OS.type
  if (os=="windows")
    shell.exec(normalizePath(f))
```

```
  else {
    pdf <- getOption("pdfviewer", default='')
    if (nchar(pdf)==0)
      stop("The 'pdfviewer' option is not set. Use options(pdfviewer=...)")
    system2(pdf, args=c(f))
  }
}
tools::texi2dvi("testtikz.tex", pdf = TRUE, clean = TRUE)
openPDF("testtikz.pdf")

# example creating html file with the above diagrams
zz <- file("demosvg.html", open="w")
writeLines(c(
  '<!DOCTYPE html>',
  '<html>',
  '<body>',
  '<h2>SVG diagrams created by lav_plot R package</h2>'),
  zz)
lav_plot(model,
        svg =  list(outfile = "temp.svg")
        )
tmp <- readLines("tmp.svg")
writeLines(tmp, zz)
writeLines("<br />", zz)
lav_plot(modelml,
        common_opts = list(sloped_labels = FALSE),
        tikz = list(outfile = "tmp.svg")
        )
tmp <- readLines("tmp.svg")
writeLines(tmp, zz)
writeLines(c("</body>", "</html>"), zz)
close(zz)
browseURL("demosvg.html")

## End(Not run)
```

`lav_plotinfo_positions` *Position the nodes in the diagram*

Description

Computes the positions for the nodes and anchors and control points for the edges in the diagram.

Usage

```
lav_plotinfo_positions(plotinfo,
                       placenodes = NULL,
                       edgelabelsbelow = NULL,
                       group_covar_indicators = FALSE,
                       debug = FALSE)
```

Arguments

plotinfo	The plotinfo structure as returned from lav_model_plotinfo.
placenodes	optional list with members *nodename* = c(row, column), row and column don't have to be integers.
edgelabelsbelow	optional list with members c(*nodename1* , *nodename2*).
group_covar_indicators	logical, should items with indicators which have an explicit covariance link be placed in the same group, i.e. forced to be on the same side of the diagram?
debug	logical, print debug information?

Details

This function **tries** to arrange the nodes and the anchor points for the edges in a way that keeps the diagram reasonably compact, while placing the dependent variable of a regression to the right of the variables it depends on. If the result is not what you want, you can inspect the plot with the lav_plotinfo_rgraph or lav_plot functions and then place some nodes at a different location (placenodes) and/or move some edge labels to the other side of the edge (labelsbelow).

For multilevel models – maximum two levels – the nodes in block 2 are grouped at the top of the diagram and those in block 1 at the bottom. The item mlrij gives the position of the separation between the blocks.

Value

A plotinfo structure with modified nodes and edges data.frames, and an integer mlrij giving the position at which a line should be drawn for multilevel models.

The data.frames have new columns defined as follows:

1. nodes

   rij index of the row where the node will be placed, initialized NA.

   kolom index of the column where the node will be placed, initialized NA.

2. edges

   vananker character, anchor point for starting node, initialized NA.

   naaranker character, anchor point for destination node, initialized NA.

   controlpt.kol real, column position of control point if the edge has to be drawn as a quadratic Bezier curve.

   controlpt.row real, row position of control point if the edge has to be drawn as a quadratic Bezier curve.

   labelbelow logical, TRUE if label has to be positioned under the line, initialized FALSE.

See Also

lav_model_plotinfo, lav_plotinfo_rgraph, lav_plot

Examples

```
model <- 'alfa  =~ 1 * x1 + x2 + x3         # latent variable
          beta <~ x4 + x5 + x6              # composite
          gamma =~ 1 * x7 + x8 + x9         # latent variable
          Xi =~ 1 * x10 + x11 + x12 + x13   # latent variable
          # regressions
          Xi ~ v * alfa + t * beta + cc * 1
          alfa ~ tt * beta + ss * gamma + yy * Theta1
          # variances and covariances
          x2 ~~ cc25 * x5
          x3 ~~ cc36 * x6
          x3 ~~ cc34 * x4
          gamma ~~ 0.55 * gamma
          '
test <- lav_model_plotinfo(model)
(test_positioned <- lav_plotinfo_positions(test))
```

---

lav_plotinfo_rgraph *Plots a graph in R*

Description

Creates a graph in R showing a simple diagram of the model.

Usage

```
lav_plotinfo_rgraph(plotinfo,
                    sloped_labels = TRUE,
                    outfile = "",
                    addgrid = TRUE,
                    mlovcolors = c("lightgreen", "lightblue"),
                    lightness = 1,
                    italic = TRUE,
                    auto_subscript = TRUE
                   )
```

Arguments

plotinfo	A plotinfo structure as returned from lav_plotinfo_positions.
sloped_labels	Logical, should labels be sloped above (or below) the edges?
outfile	Character string naming the file in which to store the diagram as a PNG, or NA to show the plot in R.
addgrid	Logical, add a grid with indicated 'positions' to the graph?
mlovcolors	Array of two colors for distinguishing ov nodes with the same name across the levels of a multilevel model.
lightness	A scalar factor to modify the distances between nodes.

`italic` Should labels be in an italic font? **Attention:** switching to an italic font is only possible if the label value is not an expression!

`auto_subscript` Logical, see lav_label_code.

Value

NULL (invisible)

See Also

lav_plotinfo_positions

Examples

```
model <- 'alpha  =~ x1 + x2 + x_3     # latent variable
          beta <~ x4 + x5 + x6        # composite
          gamma =~ x7 + x8 + x9       # latent variable
          Xi =~ x10 + x11 + x12 + x13 # latent variable
          # regressions
          Xi ~ v * alpha + t * beta + c * 1
          alpha ~ yy * Theta1 + tt * beta + ss * gamma
          '
test <- lav_model_plotinfo(model)
test1 <- lav_plotinfo_positions(test)
lav_plotinfo_rgraph(test1, lightness = 1.1)
# better position for constant in regressen Xi, no sloped labels
test2 <- lav_plotinfo_positions(test, placenodes = list(`1vanXi` = c(2, 5)))
lav_plotinfo_rgraph(test2, FALSE, lightness = 1.1)
modelml <- '
        level: 1
        fw =~ 1*y_1 + y_2 + y_3 + y_5
        level: 2
        fb =~ 1*y_1 + y_2 + y_3 + y_4
        y_2 ~~ cv24 * y_4
  '
test <- lav_model_plotinfo(modelml)
test <- lav_plotinfo_positions(test)
lav_plotinfo_rgraph(test, sloped_labels = FALSE, addgrid = FALSE,
                auto_subscript = FALSE)
## Not run:
# example where plot is stored in a PNG file
lav_plotinfo_rgraph(test, sloped_labels = FALSE, addgrid = FALSE,
                auto_subscript = FALSE, outfile="demo_rplot.png")

## End(Not run)
```

`lav_plotinfo_svgcode` *Creates code to show a diagram in svg format*

Description

Creates svg code to show a diagram of the model.

Usage

```
lav_plotinfo_svgcode(plotinfo,
                     outfile = "",
                     sloped_labels = TRUE,
                     standalone = FALSE,
                     stroke_width = 2L,
                     font_size = 20L,
                     idx_font_size = 15L,
                     dy = 5L,
                     mlovcolors = c("lightgreen", "lightblue"),
                     lightness= 1,
                     font_family = "Latin Modern Math, arial, Arial, sans",
                     italic = TRUE,
                     auto_subscript = TRUE
                     )
```

Arguments

`plotinfo`	A plotinfo structure as returned from `lav_plotinfo_positions`.
`outfile`	A connection or a character string, the file to store the code.
`sloped_labels`	logical, sloped labels for the edges.
`standalone`	logical, should code be added to produce an html file with the svg embedded in it? If FALSE (the default), the outfile (if specified) must have the file extension 'svg'; if TRUE, the file extension must be 'htm' or 'html'.
`stroke_width`	Value for stroke-width parameter in svg.
`font_size`	An integer specifying the normal font size to use.
`idx_font_size`	An integer specifying the font size to use for a subscript.
`dy`	An integer specifying the distance by which to move the baseline of the subscript.
`mlovcolors`	Array of two colors for distinguishing ov nodes with the same name across the levels of a multilevel model.
`lightness`	A scalar factor to modify the distances between nodes.
`font_family`	Fonts to be tried for rendering the labels. The first one, Latin Modern Math, is close to the default font used in tikz for mathematical expressions and can be downloaded from CTAN.
`italic`	Should labels be in an italic font? If FALSE, the labels are shown in mathrm font.
`auto_subscript`	Logical, see `lav_label_code`.

Value

NULL (invisible)

See Also

lav_plotinfo_positions

Examples

```
model <- 'alpha  =~ x1 + x2 + x3       # latent variable
          beta <~ x4 + x5 + x6         # composite
          gamma =~ x7 + x8 + x9        # latent variable
          Xi =~ x10 + x11 + x12 + x13  # latent variable
          # regressions
          Xi ~ v * alpha + t * beta + 1
          alpha ~ yy * Theta1 + tt * beta + ss * gamma
          '
test <- lav_model_plotinfo(model)
test <- lav_plotinfo_positions(test)
lav_plotinfo_svgcode(test) # no file given, so output to R console
modelml <- '
        level: 1
        fw =~ 1*y_1 + y_2 + y_3 + y_5
        level: 2
        fb =~ 1*y_1 + y_2 + y_3 + y_4
        y_2 ~~ cv24 * y_4
  '
testml <- lav_model_plotinfo(modelml)
testml <- lav_plotinfo_positions(testml)
# in the line hereunder no file is given, so output to R console
lav_plotinfo_svgcode(testml, sloped_labels = FALSE, standalone = TRUE,
            auto_subscript = FALSE)
## Not run:
# example creating html file with the above diagrams
zz <- file("demosvg.html", open="w")
writeLines(c(
  '<!DOCTYPE html>',
  '<html>',
  '<body>',
  '<h2>SVG diagrams created by lav_plot R package</h2>'),
  zz)
lav_plotinfo_svgcode(test, outfile = "temp.svg")
tmp <- readLines("tmp.svg")
writeLines(tmp, zz)
writeLines("<br />", zz)
lav_plotinfo_svgcode(testml, outfile = "temp.svg", sloped_labels = FALSE,
             standalone = TRUE)
tmp <- readLines("tmp.svg")
writeLines(tmp, zz)
writeLines(c("</body>", "</html>"), zz)
close(zz)
browseURL("demosvg.html")
```

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

`lav_plotinfo_tikzcode` *Creates code to show a diagram in tikz format*

Description

Creates the code to make a diagram in tikz.

Usage

```
lav_plotinfo_tikzcode(plotinfo,
                      outfile = "",
                      cex = 1.3,
                      sloped_labels = TRUE,
                      standalone = FALSE,
                      mlovcolors = c("lightgreen", "lightblue"),
                      lightness = 1,
                      italic = TRUE,
                      auto_subscript = TRUE
                     )
```

Arguments

`plotinfo`	A plotinfo structure as returned from `lav_plotinfo_positions`.
`outfile`	A connection or a character string, the file to store the code.
`cex`	Minimum distance between nodes in cm.
`sloped_labels`	logical, sloped labels for the edges.
`standalone`	logical, should code be added to make the TeX file standalone?
`mlovcolors`	Array of two colors for distinguishing ov nodes with the same name across the levels of a multilevel model.
`lightness`	A scalar factor to modify the distances between nodes.
`italic`	Should labels be in an italic font? If FALSE, the labels are shown in mathrm font.
`auto_subscript`	Logical, see `lav_label_code`.

Value

NULL, invisible

See Also

`lav_plotinfo_positions`

Examples

```
model <- 'alpha  =~ x1 + x2 + x3       # latent variable
          beta <~ x4 + x5 + x6         # composite
          gamma =~ x7 + x8 + x9        # latent variable
          Xi =~ x10 + x11 + x12 + x13 # latent variable
          # regressions
          Xi ~ v * alpha + t * beta + 1
          alpha ~ yy * Theta1 + tt * beta + ss * gamma
          '
test <- lav_model_plotinfo(model)
test <- lav_plotinfo_positions(test)
lav_plotinfo_tikzcode(test) # no file given, so output to R console
modelml <- '
        level: 1
        fw =~ 1*y_1 + y_2 + y_3 + y_5
        level: 2
        fb =~ 1*y_1 + y_2 + y_3 + y_4
        y_2 ~~ cv24 * y_4
  '
testml <- lav_model_plotinfo(modelml)
testml <- lav_plotinfo_positions(testml)
# in the line hereunder no file is given, so output to R console
lav_plotinfo_tikzcode(testml, cex = 1.4, sloped_labels = FALSE,
        standalone = TRUE, auto_subscript = FALSE)
## Not run:
# example creating tex file with the above diagrams
zz <- file("testtikz.tex", open="w")
writeLines(c(
  '\documentclass{article}',
  '\usepackage{amsmath, amssymb}',
  '\usepackage{amsfonts}',
  '\usepackage[utf8]{inputenc}',
  '\usepackage[english]{babel}',
  '\usepackage{xcolor}',
  '\usepackage{color}',
  '\usepackage{tikz}',
  '\usetikzlibrary {shapes.geometric}',
  '\begin{document}'),
  zz)
lav_plotinfo_tikzcode(test, outfile = "temp.tex")
tmp <- readLines("tmp.tex")
writeLines(tmp, zz)
writeLines(" ", zz)
lav_plotinfo_tikzcode(testml, outfile = "temp.tex", cex = 1.4,
              sloped_labels = FALSE, auto_subscript = FALSE)
tmp <- readLines("tmp.tex")
writeLines(tmp, zz)
writeLines("\end{document}", zz)
close(zz)
openPDF <- function(f) {
  os <- .Platform$OS.type
  if (os=="windows")
```

```
    shell.exec(normalizePath(f))
  else {
    pdf <- getOption("pdfviewer", default='')
    if (nchar(pdf)==0)
      stop("The 'pdfviewer' option is not set. Use options(pdfviewer=...)")
    system2(pdf, args=c(f))
  }
}
tools::texi2dvi("testtikz.tex", pdf = TRUE, clean = TRUE)
openPDF("testtikz.pdf")

## End(Not run)
```

`lav_samplestats` *lavaan samplestats functions*

Description

Utility functions related to the sample statistics

Usage

```
# generate samplestats object from full data
lav_samp_from_data(lavdata = NULL, lavoptions = NULL,
                           wls_v = NULL, nacov = NULL)
```

Arguments

`lavdata`	A lavdata object.
`lavoptions`	A named list. The Options slot from a lavaan object.
`wls_v`	A user provided weight matrix.
`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.

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)

# extract data slot and options
lavdata <- fit@Data
lavoptions <- lavInspect(fit, "options")
```

```
# generate sample statistics object
sampleStats <- lav_samp_from_data(lavdata = lavdata,
                                  lavoptions = lavoptions)
```

---

lavaan *Fit a Latent Variable Model*

Description

Fit a latent variable model.

Usage

```
lavaan(model = NULL, data = NULL, ordered = NULL, aux = NULL,
       sampling_weights   = NULL,
       sample_cov = NULL, sample_mean = NULL, sample_th = NULL,
       sample_nobs = NULL,
       group = NULL, cluster = NULL, constraints = "",
       wls_v = NULL, nacov = NULL, ov_order = "model",
       slot_options = NULL, slot_par_table = NULL, slot_sample_stats = NULL,
       slot_data = NULL, slot_model = NULL, slot_cache = NULL,
       sloth1 = 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 (e.g., the output of the lavParTable() 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.
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 data.frame.) Since 0.6-4, ordered can also be logical. If TRUE, all observed endogenous variables are treated as ordered (ordinal). If FALSE, all observed endogenous variables are considered to be numeric (again, unless they are declared as ordered in the data.frame.)
aux	Character vector. Names of auxiliary observed variables. Auxiliary variables are not part of the user-specified model; they are used to make the missing-at-random (MAR) assumption more plausible under missing data. Only available (for now) with continuous data and one of missing = "ml", "two.stage" or "robust.two.stage". With missing = "ml" (full-information ML), the auxiliary variables are added to the model as a block of 'saturated correlates' (Graham, 2003): each auxiliary variable is freely correlated with every other auxiliary variable, with every model observed variable, and has a free mean. The

(enlarged) model is estimated with casewise FIML, so the auxiliary variables affect both the point estimates and the standard errors, while leaving the model degrees of freedom unchanged. The baseline model is adjusted accordingly (so CFI/TLI remain correct), and the auxiliary (nuisance) parameters are hidden from the default output (use `remove_aux = FALSE` in `parameterEstimates()` to show them; they are always part of `coef()`). With `missing = "two.stage"` or `"robust.two.stage"`, the auxiliary variables are instead included in the first-stage EM run that estimates the saturated summary statistics the model is fitted to, so they also affect the model estimates; the two-stage standard errors account for the auxiliary variables via the augmented first-stage asymptotic covariance (Savalei & Bentler, 2009), except when fixed-x covariates are present (in which case they fall back to the model-only covariance). Binary or ordered/categorical auxiliary variables are not supported and are removed (with a warning).

`sampling_weights`
: A variable name in the data frame containing sampling weight information. Currently only available for non-clustered data. Depending on the `sampling_weights.normalization` option, these weights may be rescaled (or not) so that their sum equals the number of observations (total or per group).

`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. If `conditional.x = TRUE`, this is the *residual* variance-covariance matrix (given the exogenous covariates), and the regression intercepts, slopes and covariate moments are provided through the attributes `res.slopes`, `cov.x` and `mean.x` of `sample_cov` (with `sample_mean` holding the residual intercepts). It is recommended to supply these attributes with (dim)names, so that they can be matched against the model's internal variable order; unnamed attributes are assumed to be in that internal order already.

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

`sample_th`
: Vector of sample-based thresholds. For a multiple group analysis, a list with a vector of thresholds for each group.

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

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

`cluster`
: Character. A (single) variable name in the data frame defining the clusters in a two-level dataset.

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

`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.

`ov_order` Character. If `"model"` (the default), the order of the observed variable names (as reflected for example in the output of `lav_object_vnames()`) is determined by the model syntax. If `"data"`, the order is determined by the data (either the full data.frame or the sample (co)variance matrix). If the `wls_v` and/or `nacov` matrices are provided, this argument is currently set to `"data"`.

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

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

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

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

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

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

`sloth1` h1 slot from a fitted lavaan object. If provided, no new h1 slot will be created by this call.

`...` Many more options can be specified, using 'name = value'. See `lavOptions` for a complete list.

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. doi:10.18637/jss.v048.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

version: The lavaan package version used to create this object

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 of objects that are computed only once and reused many times.

Fit: Object of internal class "Fit": the results of fitting the model. No longer used.

boot: List. Results and information about the bootstrap.

optim: List. Information about the optimization.

loglik: List. Information about the loglikelihood of the model (if maximum likelihood was used).

implied: List. Model implied statistics.

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

`test`: List. Different test statistics.

`h1`: List. Information about the unrestricted h1 model (if available).

`baseline`: List. Information about a baseline model (often the independence model) (if available).

`internal`: List. For internal use only.

`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 (= unscaled) difference between the observed and the expected (model-implied) summary statistics. If `type = "cor"`, or `type = "cor.bollen"`, the observed and model implied covariance matrices are first transformed to a correlation matrix (using `cov2cor()`), before the residuals are computed. If `type = "cor.bentler"`, both the observed and model implied covariance matrices are rescaled by dividing the elements by the square roots of the corresponding variances of the observed covariance matrix. If `type="normalized"`, the residuals are divided by the square root of the asymptotic variance of the corresponding summary statistic (the variance estimate depends on the choice for the `se` argument). Unfortunately, these normalized residuals are not entirely correct, and this option is retained only for historical interest. If `type="standardized"`, the residuals are divided by the square root of the asymptotic variance of these residuals. The resulting standardized residuals can be interpreted as z-scores. If `type="standardized.mplus"`, the residuals are divided by the square root of the asymptotic variance of these residuals. However, a simplified formula is used (see the Mplus reference below) which often results in negative estimates for the variances, resulting in many `NA` values for the standardized residuals.

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

vcov `signature(object = "lavaan", type = "free")`: returns the covariance matrix of the estimated parameters. If `type = "free"` (the default), only the free parameters are included. If `type = "user"` (or `"joint"`), the joint covariance matrix of the free and the user-defined (`:=`) parameters is returned; the parameter names are the same as those used by `coef()`.

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, add, ..., 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. The `add` argument is analogous to the one described in the `lavTestScore` page, and can be used to add parameters to the specified model rather than passing an entirely new `model` argument.

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()`.

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

summary `signature(object = "lavaan", header = TRUE, fit_measures = FALSE, residuals = FALSE, estimates = TRUE, ci = FALSE, fmi = FALSE, standardized = FALSE, std = standardized, std_nox = FALSE, remove_system_eq = TRUE, remove_eq = TRUE, remove_ineq = TRUE, remove_def = FALSE, remove_nonfree = FALSE, remove_step1 = TRUE, remove_unused = TRUE, plabel = FALSE, cov_std = TRUE, rsquare = FALSE, baseline_model = NULL, h1_model = NULL, fm_args = list(standard.test = "default", scaled.test = "default", rmsea.ci.level = 0.90, rmsea.h0.closefit = 0.05, rmsea.h0.notclosefit = 0.08, robust = TRUE, cat.nonpd = "na"), modindices = FALSE, srmr_close_h0 = NULL, nd = 3L, cutoff = 0.3, dot_cutoff = 0.1)`: 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. `fit.measures` can also be a list, which allows one to set options related to the fit measures. See `fitMeasures` for more details. If `residuals = TRUE`, a residuals section is added (the largest residuals and the residual summary, printed as by `lavResiduals` with `output = "text"`); the underlying `lavResiduals(object, output = "list")` result is stored as the `residuals` element of the summary object. `residuals` can also be a list of arguments passed on to `lavResiduals` (for example `residuals = list(type = "raw", n.largest = 10)`). If both `fit.measures = TRUE` and `residuals = TRUE` and the residual summary is the SRMR (the default), the SRMR is removed from the fit measures section to avoid printing it twice. 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` or a character vector, the standardized solution is also printed (see `lavParameterEstimates`). Note that *SE*s and tests are still based on unstandardized estimates. Use `standardizedSolution` to obtain *SE*s and test statistics for standardized estimates. The `std.nox` argument is deprecated; the `standardized` argument allows `"std.nox"` solution to be specifically requested. The `standardized` argument may also be a character vector of (observed) variable names (for example `c("x1", "x2")`); only the parameters involving these variables are then standardized, and the result is shown in a `Std.usr` column. This generalizes `"std.nox"`, where the exogenous `x` variables are the ones left unstandardized. If `remove_system_eq = TRUE`, the system-generated equality constraints (using plabels) are not shown. If `remove_eq = TRUE` or `remove_ineq = TRUE`, the user-specified (in)equality constraints are not shown. If `remove_def = TRUE`, the user-specified parameter definitions are not shown. If `remove_nonfree = TRUE`, the nonfree parameters are not shown. If `remove_step1 = TRUE`, the parameters of the measurement part are not shown (only used when using `sam()`.) If `remove_unused = TRUE`, automatically added parameters that are fixed to their default (0 or 1) values are removed. If `rsquare = TRUE`, the R-Square values for the dependent variables in the model are printed. The `fm.args` list allows one to set options related to the fit measures (see `fitMeasures`).

baseline.model and h1.model allow one to specify user-defined baseline and/or h1 models for the fit measures. If the model is an exploratory factor analysis (EFA) model, EFA related information is printed automatically. The cutoff and dot.cutoff arguments control how the factor loadings are displayed: loadings whose absolute value is at or above cutoff are shown as numbers; loadings whose absolute value falls between dot.cutoff and cutoff are replaced by a dot; loadings whose absolute value is below dot.cutoff are left blank. If modindices = TRUE or modindices is a list, 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.) Historically, nothing was returned, but since 0.6-12, a list is returned of class lavaan.summary for which a print function is available.

References

Yves Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36. doi:10.18637/jss.v048.i02

Standardized Residuals in Mplus. Document retrieved from URL https://www.statmodel.com/download/StandardizedResidua

See Also

cfa, sem, fitMeasures, standardizedSolution, lavParameterEstimates, lavInspect, 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)
fitted(fit)
coef(fit)
resid(fit, type = "normalized")
```

---

lavaan-deprecated *Deprecated functions in lavaan package*

Description

Deprecated functions in lavaan.

Details

The functions listed in the first column below are deprecated; calls to them should be replaced by calls to the corresponding function in the second column.

# bootstrapLavaan	lavBootstrap

char2num	lav_char2num
cor2cov	lav_cor2cov
getCov	lav_getcov
inspectSampleCov	lavInspectSampleCov
# lavaanNames	lavNames
mplus2lavaan.modelSyntax	lav_mplus_syntax_model
mplus2lavaan	lav_Mplus_lavaan
parameterestimates	lavParameterEstimates
# simulateData	lavSimulateData

`lavaanList` *Fit List of Latent Variable Models*

Description

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

Usage

```
lavaanList(model = NULL, data_list = NULL, data_function = NULL,
    data_function_args = list(), ndat = length(data_list), cmd = "lavaan",
    ..., store_slots = c("partable"), fun = NULL, show_progress = FALSE,
    store_failed = FALSE, parallel = c("no", "multicore", "snow"),
    ncpus = min(2L, max(1L, parallel::detectCores() - 1L, na.rm = TRUE)),
    cl = NULL, iseed = NULL)

semList(model = NULL, data_list = NULL, data_function = NULL,
    data_function_args = list(), ndat = length(data_list),
    ..., store_slots = c("partable"), fun = NULL, show_progress = FALSE,
    store_failed = FALSE, parallel = c("no", "multicore", "snow"),
    ncpus = min(2L, max(1L, parallel::detectCores() - 1L, na.rm = TRUE)),
    cl = NULL, iseed = NULL)

cfaList(model = NULL, data_list = NULL, data_function = NULL,
    data_function_args = list(), ndat = length(data_list),
    ..., store_slots = c("partable"), fun = NULL, show_progress = FALSE,
    store_failed = FALSE, parallel = c("no", "multicore", "snow"),
    ncpus = min(2L, max(1L, parallel::detectCores() - 1L, na.rm = TRUE)),
    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 (e.g., the output of the `lavParTable()` function) is also accepted.

`data_list` List. Each element contains a full data frame containing the observed variables used in the model.

`data_function` Function. A function that generates a full data frame containing the observed variables used in the model. It can also be a matrix, if the columns are named.

`data_function_args`
List. Optional list of arguments that are passed to the `data_function` function.

`ndat` Integer. The number of datasets that should be generated using the `data_function` function.

`cmd` Character. The command used to run the SEM models. The possible choices are `"sem"`, `"cfa"` or `"lavaan"`, which determine how the default options are handled.

`...` Other named arguments for the `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"`, `"h1"`, `"loglik"`, or `"implied"`. Finally, `"all"` selects all slots.

`fun` Function. A function which when applied to the `lavaan` object returns the information of interest.

`store_failed` Logical. If `TRUE`, write (to tempdir()) the dataset and (if available) the fitted object when the estimation for a particular dataset failed for some reason. This allows post hoc inspection of the problem.

`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: By default, this is 2 (following CRAN policy). You may set this to a higher value (for example, the number of available CPUs) if you wish to use more processes.

`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

class `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() lavSimulateData(HS.model, sample_nobs = 100)

set.seed(1234)
fit <- semList(HS.model, data_function = generateData, ndat = 5,
               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, fitted to 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

`version`: The lavaan package version used to create this object

`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

`meta`: List containing additional flags. For internal use only.

`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.

`h1List`: List. h1 slot per dataset.

`loglikList`: List. loglik slot per dataset.

`baselineList`: List. baseline slot per dataset.

`funList`: List. fun slot per dataset.

`internalList`: List. internal 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 dataset. 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, print = TRUE, nd = 3L, simulate.args = list(est.bias = TRUE, se.bias = TRUE, se.bias.med = TRUE, prop.sig = TRUE, coverage = TRUE, level = 0.95, trim = 0))`: 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. If `print = TRUE`, the summary is printed to the screen. The argument `nd` determines the number of digits after the decimal point to be printed (currently only in the parameter estimates section.) The argument `simulate.args` is only used if the meta slot indicates that the parameter tables are obtained in the context of a simulation. The options switch on/off the columns that are printed, and the `trim` option determines the amount of trimming that is used when taking the average (or standard deviation) across all replications. If `se.bias.med = TRUE` (the default, only relevant when `se.bias = TRUE`), two additional columns are printed: `se.med` (the median, rather than the mean, of the computed standard errors across replications) and `se.bias.med` (the ratio `se.med/se.obs`). The median is a robust alternative to the mean-based `se.ave/se.bias`, which can be dominated by a few outliers for nonlinear defined parameters (e.g. ratios with near-zero denominators in some replications).

See Also

`cfaList`, `semList`, `lavaanList`

lavBootstrap *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

```
lavBootstrap(object, r = 1000L, type = "ordinary", verbose = FALSE,
             fun = "coef", keep_idx = FALSE,
             parallel = c("no", "multicore", "snow"),
          ncpus = min(2L, max(1L, parallel::detectCores() - 2L, na.rm = TRUE)),
             cl = NULL, iseed = NULL, h0_rmsea = NULL, ...)
bootstrapLavaan(object, r = 1000L, type = "ordinary", verbose = FALSE,
             fun = "coef", keep_idx = FALSE,
             parallel = c("no", "multicore", "snow"),
          ncpus = min(2L, max(1L, parallel::detectCores() - 2L, na.rm = TRUE)),
             cl = NULL, iseed = NULL, h0_rmsea = NULL, ...)
```

Arguments

object	An object of class lavaan.
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. Note that both "bollen.stine" and "yuan" require the data to be continuous. They will not work with ordinal data. 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. If the fitted model uses clustered or multilevel data (that is, a cluster= argument was used, either for a two-level model or to obtain cluster-robust standard errors), only the "ordinary" (nonparametric) bootstrap is available, and whole clusters (rather than individual observations) are resampled; see 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.
keep_idx	If TRUE, store the indices of each bootstrap run (i.e., the observations that were used for this bootstrap run) as an attribute.

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. By default, this is 2 (following CRAN policy). You may set this to a higher value (for example, the number of cores as detected by parallel::detectCores() minus one) if you wish to use more processes.
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 lavBootstrap call.
iseed	An integer to set the seed. Or NULL if no reproducible results are needed. This works for both serial (non-parallel) and parallel settings. Internally, RNGkind() is set to "L'Ecuyer-CMRG" if parallel = "multicore". If parallel = "snow" (under windows), parallel::clusterSetRNGStream() is called which automatically switches to "L'Ecuyer-CMRG". When iseed is not NULL, .Random.seed (if it exists) in the global environment is left untouched.
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.

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.

When the fitted model was estimated with clustered or multilevel data (i.e., a cluster= argument was used), the bootstrap automatically switches to a *cluster bootstrap*: instead of resampling individual observations, whole (level-2) clusters are resampled with replacement, keeping all level-1 observations within a selected cluster together. The number of clusters is held fixed (equal to the number of clusters in the original data), while the total number of observations may vary from one bootstrap sample to the next. A cluster that is drawn more than once is treated as several distinct clusters. This resampling scheme preserves the within-cluster dependence and is the appropriate bootstrap both for two-level models and for single-level models with cluster-robust standard errors. It is currently only available for a single cluster (level-2) variable, and the "bca" confidence interval type (in parameterEstimates) is not supported in this case.

When the fitted model contains rotated exploratory factors (EFA or ESEM; that is, the model syntax uses the efa() modifier together with a rotation= method other than "none"), bootstrap standard errors and confidence intervals (requested via se = "bootstrap" when fitting the model) are computed for the *rotated* solution. Each bootstrap sample is refit and rotated, and its factors are then aligned (reordered and reflected) to the original (full-sample) rotated solution before their variability is used. This alignment resolves the rotational indeterminacy (the sign and order of the exploratory factors), which would otherwise inflate the standard errors. Note that the raw bootstrap coefficients returned by lavBootstrap(object,fun = "coef") are *not* aligned to a reference solution in this way.

Value

For `lavBootstrap()`, the bootstrap distribution of the value(s) returned by `fun`, when the object can be simplified to a vector.

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. The handling of iseed was contributed by Shu Fai Cheung.

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.

Field, C. A., & Welsh, A. H. (2007). Bootstrapping clustered data. Journal of the Royal Statistical Society: Series B, 69, 369–390.

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 samples in this example; in practice
# you may wish to use a much higher number
T.boot <- lavBootstrap(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)
```

---

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, sampling_weights = NULL,
       se = "none", test = "none",
       estimator = "two.step", baseline = FALSE, ...,
       cor_smooth = FALSE, cor_smooth_tol = 1e-04, 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.
sampling_weights	Only used if object is a data.frame. Specify a variable containing sampling weights.
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 lavOptions for possible options.
test	Only used if output is "fit" or "lavaan". See lavOptions 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.
baseline	Only used if output is "fit" or "lavaan". If TRUE, a baseline model is also estimated. Note that the test argument should also be set to a value other than "none".
...	Optional parameters that are passed to the lavaan function.
cor_smooth	Logical. Only used if output = "cor". If TRUE, ensure the resulting correlation matrix is positive definite. The following simple method is used: an eigenvalue

decomposition is computed; then, eigenvalues smaller than `cor.smooth.tol` are set to be equal to `cor.smooth.tol`, before the matrix is again reconstructed. Finally, the matrix (which may no longer have unit diagonal elements) is converted to a correlation matrix using `cov2cor`.

`cor_smooth_tol` Numeric. Smallest eigenvalue used when reconstructing the correlation matrix after an eigenvalue decomposition.

`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 `lavInspect(fit, "sampstat")` where fit is the unrestricted model. If `"est"` or `"pe"` or `"parameterEstimates"`, the output equals the result of `lavParameterEstimates(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)
```

`lavEffects` *Indirect and Total Effects*

Description

'lavEffects' computes various 'effects' that are functions of the estimated model parameters of a fitted lavaan object. In this version, the focus is on the classic (LISREL-style) *total*, *indirect* and *direct* effects among the variables that appear in the structural part of the model (that is, the variables that are involved in a regression). Both observed and latent variables are supported.

This avoids the need to pre-specify all indirect (and total) effects manually in the model syntax (using the `:=` operator), which can be tedious when there are many of them.

Usage

```
lavEffects(object, effects = c("total", "indirect"),
    se_def = NULL, level = 0.95, monte_carlo = NULL,
    boot_ci_type = "perc", zstat = TRUE, pvalue = TRUE, ci = TRUE,
    standardized = FALSE, cov_std = TRUE,
    add_class = TRUE, output = "data.frame", ...)
```

Arguments

`object`	An object of class `lavaan`.
`effects`	Character vector. One or more of `"total"`, `"indirect"` and `"direct"`. The default is `c("total", "indirect")`. The order of the elements does not matter; effects are always reported in the order total, indirect, direct.
`se_def`	Character (or `NULL`). The method used to compute standard errors (and confidence intervals) for the effects. If `"monte_carlo"`, the Monte Carlo method is used: parameters are drawn from a multivariate normal distribution with mean the parameter estimates and covariance matrix the estimated covariance matrix of the parameters, and the effects are recomputed for each draw (percentile

confidence intervals). If "delta", the delta method is used (based on a numerical Jacobian of the effects with respect to the free parameters; symmetric normal-theory confidence intervals). If "bootstrap", the bootstrap draws that are stored in the fitted object (when the model was fitted with se = "bootstrap") are used to compute bootstrap standard errors and bootstrap percentile confidence intervals. If "none", no standard errors are computed.

If se_def = NULL (the default), the choice is inherited from the fitted object: if the model was fitted with se = "bootstrap" (and bootstrap draws are available), the bootstrap method is used; else if the model was fitted with se_def = "monte.carlo", the Monte Carlo method is used; otherwise the Monte Carlo method is used as the default. An explicit value always overrides this inheritance.

level — Numeric. The confidence level for the confidence intervals.

monte_carlo — List (or NULL). Settings for the Monte Carlo method (only used when se_def = "monte.carlo"), with elements R (the number of Monte Carlo draws) and seed (the random seed). If NULL (the default), the monte_carlo settings that were used when the model was fitted are inherited (which themselves default to list(R = 20000L, seed = NULL)). The state of the global random number generator is saved and restored, so the seed used here does not affect the user's random number stream.

boot_ci_type — Character. Only used when se_def = "bootstrap". The type of bootstrap confidence interval, as in parameterEstimates: "norm" (normal-theory using the bootstrap bias and standard error), "basic" (basic bootstrap), "perc" (the default; percentile method), "bca.simple" (bias-corrected percentile method, without the acceleration term) or "bca" (the adjusted bootstrap percentile method, with both bias and acceleration correction).

zstat — Logical. If TRUE, an extra column is added with the z-statistic for each effect (the ratio of the estimate to its standard error).

pvalue — Logical. If TRUE, an extra column is added with the (two-sided) p-value, computed using the standard normal distribution.

ci — Logical. If TRUE (the default), two extra columns (ci.lower and ci.upper) are added with the lower and upper bounds of the confidence interval. If FALSE, the confidence interval bounds are not shown.

standardized — Logical, character vector, or vector of (observed) variable names, behaving as in parameterEstimates. If TRUE, standardized effects are added as extra columns (point estimates only): std.lv (only the latent variables are standardized), std.all (both observed and latent variables are standardized) and, if there are exogenous observed variables and fixed.x = TRUE, std.nox (like std.all, but the exogenous observed variables are not standardized). A character vector selects specific types (a subset of "std.lv", "std.all", "std.nox"); a vector of observed variable names standardizes only those observed variables (plus the latent variables), and the result is reported in a std.user column. The standardized effect of j on i equals the unstandardized effect times the ratio of the model-implied standard deviations sd_j/sd_i.

cov_std — Logical. See parameterEstimates. (Has no influence on the standardized regression-type effects reported here.)

add_class	Logical. If TRUE, the result (when output = "data.frame") is given the class c("lavaan.effects", "lavaan.data.frame", "data.frame"), so that a dedicated print method is used.
output	Character. If "data.frame" (the default), the result is a (classed) data.frame with one row per effect. If "list", the result is a list of matrices (one per effect type, and one per block in the multilevel/multigroup case), where element $[i, j]$ contains the effect of variable j on variable i.
...	Only old names of arguments - with dots - are allowed here.

Details

All effects are derived from the reduced-form matrix $(I - B)^{-1}$, where B is the matrix of (direct) regression coefficients among the structural variables (the beta matrix in the LISREL representation that lavaan uses internally). Writing $B[i, j]$ for the direct effect of variable j on variable i, we have:

- the *direct* effect of j on i equals $B[i, j]$;
- the *total* effect of j on i equals $\left((I - B)^{-1} - I\right)[i, j]$;
- the *indirect* effect of j on i equals the total effect minus the direct effect.

Only effects that are structurally non-zero (that is, for which a directed path from j to i exists) are reported. For indirect effects, this means that at least one path of length two or more must exist.

The delta and Monte Carlo methods only require the parameter estimates and the estimated covariance matrix of the parameters. The delta method produces (symmetric) normal-theory confidence intervals; the Monte Carlo method produces percentile-based confidence intervals, which need not be symmetric around the point estimate. The Monte Carlo method typically behaves better than the delta method for effects that are products of parameters (such as indirect effects), in particular in smaller samples. When the model was fitted with se = "bootstrap", the bootstrap draws are reused to obtain bootstrap standard errors and bootstrap percentile confidence intervals (this requires no additional model fitting).

Models fitted with conditional.x = TRUE are also supported: in that case the structural model is $eta = Beta + Gammax + zeta$, where the exogenous covariates x are stored in the gamma matrix. The effects of these covariates on the endogenous variables are then computed as $(I-B)^{-1}Gamma$ (total), $Gamma$ (direct) and $((I-B)^{-1}-I)Gamma$ (indirect), and reported alongside the effects among the beta variables.

Value

If output = "data.frame" (the default), a data.frame (of class lavaan.effects) with the following columns: effect (the type of effect: total, indirect or direct), lhs (the outcome variable), op (always "~"), rhs (the predictor variable), and, depending on the arguments, group/level/block, est, se, z, pvalue, ci.lower, ci.upper and (if standardized is requested) one or more of std.lv, std.all, std.nox and std.user. The effect in a given row is the effect of rhs on lhs.

If output = "list", a (possibly nested) list of matrices, where element $[i, j]$ contains the effect of variable j on variable i.

See Also

parameterEstimates, standardizedSolution.

Examples

```
# a simple mediation model
set.seed(1234)
X <- rnorm(300)
M <- 0.5 * X + rnorm(300)
Y <- 0.4 * M + 0.3 * X + rnorm(300)
Data <- data.frame(X = X, Y = Y, M = M)
model <- ' # direct effect
             Y ~ c*X
           # mediator
             M ~ a*X
             Y ~ b*M
         '
fit <- sem(model, data = Data)

# total and indirect effects, with Monte Carlo standard errors
lavEffects(fit)

# all effects, using the delta method
lavEffects(fit, effects = c("total", "indirect", "direct"), se_def = "delta")

# add standardized (total and indirect) effects
lavEffects(fit, se_def = "delta", standardized = TRUE)
```

---

lavExport *lavaan Export*

Description

Export a fitted lavaan object to an external program.

Usage

```
lavExport(object, target = "lavaan", prefix = "sem", dir_name = "lav_export",
          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.
...	Only to support old argument name 'dir.name'.

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 lav_export 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

lavParTable, lav_mplus_lavaan

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) inspect() function is now simply a shortcut for lavInspect() with default arguments.

Usage

```
lavInspect(object, what = "free", add_labels = TRUE, add_class = TRUE,
           list_by_group = TRUE,
           drop_list_single_group = TRUE, ...)

lavTech(object, what = "free", add_labels = FALSE, add_class = FALSE,
        list_by_group = FALSE,
        drop_list_single_group = FALSE, ...)

inspect(object, what = "free", ...)
```

Arguments

object	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.
...	For inspect() additional arguments can be specified, not used by lavaan, but by other packages. For the functions lavInspect() and lavTech() this argument is used to support old names of arguments.

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 (ignoring any equality constraints). They correspond to 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 share the same standard error. Aliases: "std.err" and "standard.errors".

"se.std": A list of model matrices. The non-zero numbers represent the standard errors for the (completely) standardized free parameters in the model. Alias: "std.se".

"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", and "x".

"est.unrotated": A list of model matrices. The values represent the estimated model parameters before rotation was applied. Only relevant if rotation was used (for example in EFA); otherwise this is identical to "est".

"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.)

"mm.lambda": The estimated factor loading matrix (per group), as extracted from the @GLIST slot. Alias: "lambda".

"mm.delta": The estimated delta (scaling) matrix (per group), as extracted from the @GLIST slot. Only available for categorical data or when a correlation structure is used. (Note: this is the delta model matrix, not the delta *jacobian* that is returned by "delta".)

To extract a single model matrix (from the @GLIST slot), use the "mm.*" options (for example "mm.lambda" for the factor loading matrix). For backward compatibility, the (older) bare names are still accepted as aliases.

Information about the data:

"data": A matrix containing the observed variables that were used to fit the model. The matrix has no row or column names. The columns correspond to the output of lav_object_vnames(object), while the rows correspond to the output of lavInspect(object, "case.idx").

"ordered": A character vector. The ordered variables.

"nobs": Numeric vector. The effective number of observations in each group that were used in the analysis. If sampling weights were provided, this is the (per group) sum of the weights, consistent with the value returned by the nobs() method. Use "norig" to obtain the number of rows in the data.

"norig": Integer vector. The original number of observations (rows in the data) in each group.

"ntotal": Numeric. The total effective 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. If sampling weights were provided, this is the sum of the weights.

"case.idx": Integer vector. 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.

"th.idx": Integer vector. For categorical data, the index of the observed variable that each threshold belongs to. In the case of multiple groups: a list of integer vectors.

"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 in which each element contains the proportion of jointly observed data points for the corresponding pair of observed variables. In the case of multiple groups: a list of coverage matrices.

"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.

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

"cluster": A character vector. The cluster variable(s) in the data.frame (if any).

"nlevels": Integer. The number of levels.

"nclusters": Integer. The number of clusters that were used in the analysis.

"ncluster.size": Integer. The number of different cluster sizes.

"cluster.size": Integer vector. The number of observations within each cluster. For multigroup multilevel models, a list of integer vectors, indicating cluster sizes within each group.

"cluster.id": Integer vector. The cluster IDs identifying the clusters. For multigroup multilevel models, a list of integer vectors, indicating cluster IDs within each group.

"cluster.idx": Integer vector. The cluster index for each observation. The cluster index ranges from 1 to the number of clusters. For multigroup multilevel models, a list of integer vectors, indicating cluster indices within each group.

"cluster.label": Integer vector. The cluster ID for each observation. For multigroup multilevel models, a list of integer vectors, indicating the cluster ID for each observation within each group.

"cluster.sizes": Integer vector. The different cluster sizes that were used in the analysis. For multigroup multilevel models, a list of integer vectors, indicating the different cluster sizes within each group.

"average.cluster.size": Integer. The average cluster size (using the formula `s = (N^2 - sum(cluster.size^2)) / (N*(nclusters - 1L))`). For multigroup multilevel models, a list containing the average cluster size per group.

Observed sample statistics:

"sampstat": Observed sample statistics. Aliases: "obs", "observed", "samp", "sample", "samplestatistics". Since 0.6-3, we always check if an h1 slot is available (the estimates for the unrestricted model); if present, we extract the sample statistics from this slot. This implies that if the variables are continuous and `missing = "ml"` (or "fiml"), we return the covariance matrix (and mean vector) as computed by the EM algorithm under the unrestricted (h1) model. If the h1 slot is not present (for example, because the model was fitted with `h1 = FALSE`), we return the sample statistics from the SampleStats slot. In that case, pairwise deletion is used for the elements of the covariance matrix (or correlation matrix), and listwise deletion for all univariate statistics (means, intercepts, and thresholds).

"sampstat.std": Standardized observed sample statistics. The covariance matrix is rescaled to a correlation matrix. Aliases: "obs.std", "observed.std", "samp.std", "sample.std", "samplestatistics.std".

"sampstat.h1": Deprecated. Do not use any longer.

"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:

"implied": The model-implied summary statistics. Alias: "fitted", "expected", "exp".

"resid": The difference between observed and model-implied summary statistics. Alias: "residuals", "residual", "res".

"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".

"mm.theta": The (residual) variance-covariance matrix of the observed variables (the theta model matrix). Aliases: "theta", "theta.cov".

"theta.cor": The (residual) variance-covariance matrix of the observed variables, expressed in correlation metric.

"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".

"fs_determinacy": The factor determinacies (based on regression factor scores). They represent the (estimated) correlation between the factor scores and the latent variable scores.

"fs.reliability": The factor reliabilities (based on regression factor scores). They are the square of the factor determinacies.

"fs_determinacy_Bartlett": The factor determinacies (based on Bartlett factor scores). They represent the (estimated) correlation between the factor scores and the latent variable scores.

"fs_reliability_bartlett": The factor reliabilities (based on Bartlett factor scores). They are the square of the factor determinacies.

"icc": The intraclass correlation coefficients for clustered data with multilevel models. Computed from the model-implied (H1) within-group and between-group covariance matrices. Only available for models with clustered data and multiple levels.

"ranef": The random effects (empirical Bayes predictions of the cluster-level latent variables). Only available for clustered data (in the long format) with multiple levels.

Diagnostics:

"mdist2.fs": The squared Mahalanobis distances for the (Bartlett) factor scores.

"mdist.fs": The Mahalanobis distances for the (Bartlett) factor scores.

"mdist2.resid": The squared Mahalanobis distances for the (Bartlett-based) casewise residuals.

"mdist.resid": The Mahalanobis distances for the (Bartlett-based) casewise residuals.

If (some of) the data is ordered categorical, the Mahalanobis distances are generalized as in Mansolf and Reise (2017): each distance is replaced by its expected value over the region of the (multivariate normally distributed) latent response vector that is consistent with the observed response pattern, approximated by Monte Carlo integration (`set.seed()` can be used for reproducibility). Observed continuous variables are conditioned on, and missing values are integrated out. Note that in this case "mdist.fs"/"mdist.resid" return the expected distance $E[d]$ (the person-fit indices d_f^* and d_r^* of Mansolf and Reise), while "mdist2.fs"/"mdist2.resid" return the expected squared distance $E[d^2]$, which is not the square of the former.

Optimizer information:

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

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

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

"npar": Integer. Number of free parameters (ignoring constraints).

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.

"gradient.logl": Numeric vector containing the first derivatives of the loglikelihood with respect to the (free) model parameters.

"optim.gradient": Numeric vector containing the first derivatives of the objective function (as seen by the optimizer) 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). This is unit-information, not total-information.

`"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.

`"h1.information"`: Matrix containing either the observed, expected or first.order information matrix (depending on the information option of the fitted model) of the unrestricted h1 model. This is unit-information, not total-information.

`"h1.information.expected"`: Matrix containing the expected information matrix for the unrestricted h1 model.

`"h1.information.observed"`: Matrix containing the observed information matrix for the unrestricted h1 model.

`"h1.information.first.order"`: Matrix containing the first.order information matrix for the unrestricted h1 model. Alias: `"h1.first.order"`.

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.

"vcov.def": Matrix containing the variance covariance matrix of the user-defined (using the := operator) parameters.

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

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

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

"vcov.def.joint": Matrix containing the joint variance covariance matrix of both the estimated model parameters and the defined (using the := operator) parameters.

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

"vcov.def.joint.std.lv": Matrix containing the joint variance covariance matrix of both the standardized model parameters and the user-defined parameters. Standardization is done with respect to the latent variables only.

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

Miscellaneous:

"coef.boot": Matrix containing estimated model parameters for each bootstrap sample. Only relevant when bootstrapping was used.

"monte.carlo": Matrix containing the Monte Carlo draws of the model parameters, as used for the Monte Carlo confidence intervals of defined parameters (Preacher & Selig, 2012). Only relevant when these draws were requested. Aliases: "mc", "mc.coef", "coef.mc".

"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.

"UfromUGamma": Matrix containing the 'U' matrix as used by the Satorra-Bentler correction. Alias: "U".

"delta": The delta matrix (per group): the Jacobian of the model-implied summary statistics (the rows: means, (co)variances, thresholds, ...) with respect to the free model parameters (the columns). This is the matrix used in the delta method. (Note: this is the delta *Jacobian*, not the delta model matrix that is returned by "mm.delta".) In the case of multiple groups: a list of matrices.

"delta.rownames": The row names of the delta matrix (see "delta"): the labels of the model-implied summary statistics. In the case of multiple groups: a list of character vectors.

"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().

"loglik.casewise": Vector containing the casewise loglikelihood contributions. Only available if estimator = "ML".

"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).

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

"baseline.partable": Data.frame. The parameter table of the (internal) baseline model.

"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.

"zero.cell.tables": List. List of bivariate frequency tables where at least one cell is empty.

"iv": Data.frame. One row per (transformed) equation, with the (model-implied) instrumental variables that were used for each equation. Columns are lhs, rhs, lhs.new, rhs.new, type, and instruments. Only available when estimator = "IV". Aliases: "ivs", "miiv", "miivs", "instr", "instruments". In the case of multiple groups: a list of data.frames.

"eqs": List. The (internal) list of equations that was used for the (MI)IV-based estimation, including the instruments, the coefficients, and the Sargan test per equation. Only available when estimator = "IV".

"sargan": Data.frame. The per-equation overidentification tests, for each overidentified equation. Columns are lhs, rhs, df, sargan.stat, sargan.pval, browne.stat, and browne.pval. The sargan.* columns are the classical Sargan test (assumes normality; the p-value is NA for categorical data, where it is not valid); the browne.* columns are a robust residual-based test computed with the distribution-free (ADF) ACOV for continuous data or the polychoric ACOV for categorical data. "hansen" is an alias. Only available when estimator = "IV". In the case of multiple groups: a list of data.frames.

"sam.struc.fit.object": The (internal) step-2 structural model of a model fitted with sam using a local sam.method ("local", "fsr" or "cfsr"): a lavaan object, fitted to the estimated latent (co)variances (and means) of step 1. fitMeasures(), lavResiduals() and modindices() operate on this object when they are called with the sam object itself. "sam.struc" is an alias.

"version": The lavaan version number that was used to construct the fitted lavaan object.

See Also

lavaan, estimator_iv

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")
```

---

lavInspectSampleCov *Observed Variable Covariance Matrix from a Model and Data*

Description

The lavaan model syntax describes a latent variable model. Often, the user wants to see the sample covariance matrix of the variables in their model for diagnostic purposes. However, their data may contain many more columns than the variables used in the model.

Usage

```
lavInspectSampleCov(model, data, ...)
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 without fitting the model, and then uses lavInspect to obtain the sample covariance matrix and the meanstructure.

See also

sem, lavInspect

Author(s)

Jarrett Byrnes

lavListInspect *Inspect or extract information from a lavaanList object*

Description

The lavListInspect() and lavListTech() functions can be used to inspect/extract information that is stored inside (or can be computed from) a lavaanList object.

Usage

```
lavListInspect(object, what = "free", add_labels = TRUE,
               add_class = TRUE, list_by_group = TRUE,
               drop_list_single_group = TRUE, ...)

lavListTech(object, what = "free", add_labels = FALSE,
            add_class = FALSE, list_by_group = FALSE,
            drop_list_single_group = FALSE, ...)
```

Arguments

object	An object of class lavaanList.
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.
...	To support old argument names.

Details

The lavListInspect() and lavListTech() functions only differ in the way they return the results. The lavListInspect() function will prettify the output by default, while the lavListTech() will not attempt to prettify the output by default.

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 (ignoring any equality constraints). They correspond to all entries (fixed or free) in the parameter table. See parTable.

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

Information about the data (including missing patterns):

"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.

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

"cluster": A character vector. The cluster variable(s) in the data.frame (if any).

"nlevels": Integer. The number of levels.

"ordered": A character vector. The ordered variables.

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

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

"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 (in each dataset).

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".

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

"options": List. The option list.

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

See Also

lavaanList

Examples

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

# a data generating function
generateData <- function() lavSimulateData(HS.model, sample_nobs = 100)

set.seed(1234)
fit <- semList(HS.model, dataFunction = generateData, ndat = 5,
               store_slots = "partable")

# extract information
lavListInspect(fit, "free")
lavListTech(fit, "free")
```

---

lavMatrixRepresentation

lavaan matrix representation

Description

Extend the parameter table with a matrix representation.

Usage

```
lavMatrixRepresentation(partable, representation = "LISREL",
                        allow_composites = TRUE,
                        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.
allow_composites	Logical. If TRUE, and the model syntax contains the "<~" operator, handle the lhs as proper composites. If FALSE, use the old (<0.6-20) approach.
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.
...	Used only to support old argument names.

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 variable names from a fitted lavaan object.

Usage

```
lavNames(object, type = "ov", ...)
```

Arguments

object	An object of class lavaan.
type	Character. The type of variables whose names should be extracted. See details for a complete list.
...	Additional selection criteria. For example, "group = 2L" (in a multiple-group analysis) restricts the selection to the variables included in the model for the second group.

Details

The order of the variable names, as returned by `lav_object_vnames`, 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"`: modeled 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"`: isolated observed variables (only their intercepts or variances appear in the model syntax)
- `"ov.interaction"`: interaction terms (defined by the colon operator)
- `"th"`: threshold names (ordinal variables only)
- `"th.mean"`: threshold names (ordinal and numeric variables, if any)
- `"lv"`: latent variables
- `"lv.regular"`: measured latent variables (defined by =~ only)
- `"lv.composite"`: composites (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

lavParTable, 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")
```

lavOptions *lavaan Options*

Description

Show the default options used by the lavaan() function. The options can be changed by passing 'name = value' arguments to the lavaan() function call, where they are added to the '...' argument.

Usage

```
lavOptions(x = NULL, default = NULL, mimic = "lavaan")
```

Arguments

x	Character. A character string holding an option name, or a character string vector holding multiple option names. All option names are converted to lower case.
default	If a single option is specified but not available, this value is returned.
mimic	Not used for now.

Details

This is the full list of options that are accepted by the lavaan() function, organized in several sections:

Model features:

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 reflect 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, all covariances among latent variables are set to zero.

orthogonal.y: If TRUE, all covariances among endogenous latent variables only are set to zero.

orthogonal.x: If TRUE, all covariances among exogenous latent variables only are set to zero.

`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. If there are multiple groups, `std.lv = TRUE` and `"loadings"` is included in the `group.equal` argument, then only the latent variances of the first group will be fixed to 1.0, while the latent variances of other groups are set free.

`effect.coding`: Can be logical or character string. If logical and `TRUE`, this implies `effect.coding = c("loadings", "intercepts")`. If logical and `FALSE`, it is set equal to the empty string. If `"loadings"` is included, equality constraints are used so that the average of the factor loadings (per latent variable) equals 1. Note that this should not be used together with `std.lv = TRUE`. If `"intercepts"` is included, equality constraints are used so that the sum of the intercepts (belonging to the indicators of a single latent variable) equals zero. As a result, the latent mean is freely estimated and usually equals the average of the means of the involved indicators.

`ceq.simple`: Logical. If `TRUE`, and no other general (equality or inequality) constraints are used in the model, simple equality constraints are represented in the parameter table as duplicated free parameters (instead of extra rows with `op = "=="`). The default is `FALSE`.

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

`correlation`: Only used for (single-level) continuous data. If `TRUE`, analyze a correlation matrix (instead of a (co)variance matrix). This implies that the residual observed variances are no longer free parameters. Instead, they are set to values that ensure the model-implied variances are unity. This also affects the standard errors. The only available estimators are GLS and WLS, which produce correct standard errors and a correct test statistic under normal and non-normal conditions respectively. Both `fixed.x = FALSE` and `fixed.x = TRUE` are supported (the latter affects the way the standard errors are computed). For now, this option always assumes `conditional.x = FALSE`. Alternatively, `correlation` may be a character vector of (observed) variable names (for example `correlation = c("x1", "x2", "x3")`). In that case only the listed variables are standardized to unit variance (their residual variances are fixed accordingly), while all remaining observed variables keep their original metric (a 'partial' correlation structure). If the model contains observed product (interaction) terms (for example `x1:y1`) and `correlation = TRUE`, the product terms are automatically excluded from the correlation set: the product of standardized variables does not have unit variance, so their (co)variances remain free parameters. Note that product terms are typically highly non-normal, so the normal-theory (GLS) standard errors for their (co)variances may be too small; consider `se = "robust.sem"`.

`composites.cov`: Character string. Only used if the model contains composites. Controls whether the composite-indicator (co)variances (the elements of the `T` matrix) are fixed to their sample values or estimated as free parameters. If `"fixed"`, the `T` matrix is fixed to the sample (co)variances. If `"free"`, the `T` matrix is estimated as free parameters. If `"default"`, the value is resolved to `"free"` for multilevel models (which makes composites at every level identified) and `"fixed"` otherwise. For a single-level model, freeing `T` is fit-invariant (the degrees of freedom are unchanged).

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

`bad.marker.crit`: Only used if `auto.fix.first = TRUE`. A single number (default 0.1). If larger than zero, lavaan checks – once the unrestricted (h1) sample statistics are available – whether the first indicator of each latent variable is a poor item, in the sense that it correlates weakly with the other indicators of the same factor (more precisely: the absolute value of its corrected

item-total correlation is below bad.marker.crit). If so (and a clearly better indicator is available), a warning is issued and the loading of that better indicator is fixed to 1.0 instead, in order to set the metric of that latent variable. The main purpose is to avoid convergence problems caused by a (very) poor marker item. The check is based on the (pooled) unrestricted (h1) covariance matrix, so it behaves consistently with or without missing data, categorical data, multiple groups, etc. Larger values make the switching more aggressive; a value of 0 disables the check and always keeps the first indicator as the marker.

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 of both observed and latent variables are set free.

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

auto.cov.x: If TRUE, the covariances between exogenous latent variables and observed exogenous covariates are also included in the model and set free (this implies auto.cov.lv.x = TRUE). Ignored if conditional.x = TRUE, as these covariances cannot be represented when the model is conditioned on the exogenous covariates.

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.

auto.efa: If TRUE, the necessary constraints are imposed to make the (unrotated) exploratory factor analysis blocks identifiable: for each block, factor variances are set to 1, factor covariances are constrained to be zero, and factor loadings are constrained to follow an echelon pattern.

Data options:

std.ov: If TRUE, observed variables are standardized before entering the analysis. By default, these are only the non-exogenous observed variables, unless fixed.x = FALSE. Use this option with caution; it can be used to test whether (for example) nonconvergence was due to scaling issues. Note that this is still a covariance-based analysis: no constraints are imposed to ensure the model-implied (co)variance matrix has unit variances, and the standard errors still assume that the input was unstandardized. See also the correlation option.

missing: The default setting is "listwise": all cases with missing values are removed listwise from the data before the analysis starts. This is only valid if the data are missing completely at random (MCAR). Therefore, it may not be the optimal choice, but it can be useful for a first run. If the estimator belongs to the ML family, another option is "ml" (alias: "fiml" or "direct"). This corresponds to the so-called full information maximum likelihood approach (fiml), where we compute the likelihood case by case, using all available data from that case. Note that if the model contains exogenous observed covariates, and fixed.x = TRUE (the default), all cases with any missing values on these covariates will be deleted first. The option "ml.x" (alias: "fiml.x" or "direct.x") is similar to "ml", but does not delete any cases with missing values for the exogenous covariates, even if fixed.x = TRUE. (Note: all lavaan versions < 0.6 used "ml.x" instead of "ml"). Since version 0.7-2, "ml" can also be combined with conditional.x = TRUE (single-level, continuous data): the likelihood of the observed non-x variables is then computed case by case, conditional on the (complete)

exogenous covariates of that case. Cases with missing values on the exogenous covariates are deleted first, and "ml.x" is not available in this setting. Note that the robust (missing-data corrected) versions of RMSEA/CFI are not (yet) available when conditional.x = TRUE. If you wish to use multiple imputation, you need to use an external package (e.g., mice) to generate imputed datasets, which can then be analyzed using the semList function. The semTools package contains several functions to do this automatically. Another option (with continuous data) is to use "two.stage" or "robust.two.stage". In this approach, we first estimate the sample statistics (mean vector, variance-covariance matrix) using an EM algorithm. Then, we use these estimated sample statistics as input for a regular analysis (as if the data were complete). The standard errors and test statistics are adjusted correctly to reflect the two-step procedure. The "robust.two.stage" option produces standard errors and a test statistic that are robust against non-normality. Both options are available for the ML estimator (since version 0.7-2 also in combination with conditional.x = TRUE), and also for the (continuous-data) least-squares estimators "ULS", "GLS", "WLS" and "DLS"; for the latter, lavaan automatically switches to se = "robust.sem" and test = "satorra.bentler", using the two-stage variance-covariance matrix of the EM-based sample statistics. In addition, for these least-squares estimators, requesting missing = "ml" (or "fiml") is interpreted as a generic request to handle the missing values, and is silently treated as missing = "two.stage". If (part of) the data is categorical, and the estimator is from the (W)LS family, the only option (besides listwise deletion) is "pairwise". In this three-step approach, missingness is only an issue in the first two steps. In the first step, we compute thresholds (for categorical variables) and means or intercepts (for continuous variables) using univariate information only. In this step, we simply ignore the missing values just like in mean(x, na.rm = TRUE). In the second step, we compute polychoric/polyserial/pearson correlations using (only) two variables at a time. Here we use pairwise deletion: we only keep those observations for which both values are observed (not missing), and this set may change from pair to pair. By default, in the categorical case we use conditional.x = TRUE. Therefore, any cases with missing values on the exogenous covariates will be deleted listwise from the data first. Finally, if the estimator is "PML", the available options are "pairwise", "available.cases" and "doubly.robust". See the PML tutorial on the lavaan website for more information about these approaches.

sampling.weights.normalization: If "none", the sampling weights (if provided) will not be transformed. If "total", the sampling weights are normalized by dividing by the total sum of the weights, and multiplying again by the total sample size. If "group", the sampling weights are normalized per group: by dividing by the sum of the weights (in each group), and multiplying again by the group size. The default is "group". (For a single group, "group" and "total" are identical; they only differ when there are multiple groups, in which case "group" keeps each group's relative contribution proportional to its sample size, matching the behavior of Mplus.)

sampling.weights.type: Only used when sampling weights are provided and the estimator relies on a weight matrix or sandwich variance: the least-squares family (GLS/WLS/DWLS/ULS/DLS, both continuous and categorical) and PML. Determines how the sampling weights enter the asymptotic variance (the Gamma / NACOV matrix, or the PML first-order information), and hence the (robust) standard errors and the scaled test statistic. For GLS and ULS only the standard errors are affected; for WLS/DWLS/DLS the Gamma also feeds the weight matrix, so the point estimates shift as well. If "design" (the default), the weights are treated as sampling (design) weights and a design-based sandwich is used (the meat is weighted by the sum of the squared weights), matching the behavior of Mplus. If "frequency", the weights are treated as frequency (replication) counts (the meat is weighted by the sum of the weights), so that the

results match a fit on the row-replicated data. Without sampling weights the two are identical.

`samplestats`: Logical. If `FALSE`, no sample statistics will be computed (and no estimation can take place). This can be useful when only a dummy lavaan object is requested, without any computations. The default is `TRUE`.

Data summary options:

`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.

`ridge`: Logical. If `TRUE`, a small constant value is added to the diagonal elements of the covariance (or correlation) matrix before analysis. The value can be set using the `ridge.constant` option.

`ridge.constant`: Numeric. Small constant used for ridging. The default value is 1e-05.

Multiple group options:

`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"`, `"composite.weights"`, `"intercepts"`, `"means"`, `"thresholds"`, `"regressions"`, `"residuals"`, `"residual.covariances"`, `"lv.variances"` or `"lv.covariances"`, specifying the pattern of equality constraints across multiple groups. As a shortcut, the single value `"all"` can be used to constrain all (free) parameters to be equal across groups; it is equivalent to listing all of the values above. When the model is not the same across groups (for example when the `group:` modifier is used to specify a different model per group), only parameters that are present in two or more groups are constrained to be equal; a parameter that occurs in a single group only is always left free. The constraints are symmetric and do not depend on the first group: a parameter that is shared by (say) the second and third group is constrained to be equal across those two groups, even if it does not appear in the first group.

`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.

Estimation options:

`estimator`: The estimator to be used. Can be one of the following: `"ML"` for maximum likelihood, `"GLS"` for (normal theory) generalized least squares, `"WLS"` for weighted least squares (sometimes called ADF estimation), `"ULS"` for unweighted least squares, `"DWLS"` for diagonally weighted least squares, and `"DLS"` for distributionally-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 (information = "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 (test="yuan.bentler.mplus"). 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. Finally, "RBM" requests reduced-bias M-estimation (penalized maximum likelihood; continuous-outcome models for now). By default sandwich standard errors are used (se = "robust.huber.white"). The type of bias reduction is controlled by the rbm.method element of estimator.args (either "implicit" (default) or "explicit"), for example estimator = list(estimator = "RBM", rbm.method = "explicit"). The "explicit" method is much faster for models with many parameters (it requires a single ML fit plus a one-step correction), and is recommended in that case.

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: Not used yet. This is just a placeholder until the MML estimator is back.

information: If "expected", the expected information matrix is used (to compute the standard errors). If "observed", the observed information matrix is used. If "first.order", the information matrix is based on the outer product of the casewise scores. See also the options "h1.information" and "observed.information" for further control. If "default", the value is set depending on the estimator, the missing argument, and the mimic option. If the argument is a vector with two elements, the first element is used for the computation of the standard errors, while the second element is used for the (robust) test statistic.

h1.missing.method: Character. Only used for multilevel models with missing data (and missing = "ml"). If "em" (the default since 0.7-1), the means and (co)variances of the unrestricted (h1) model are estimated using the EM algorithm, integrating over both the cluster random effects and the missing values (this mimics Mplus). If "fiml", the unrestricted model is estimated using quasi-Newton optimization (nlminb) with a cholesky parameterization of the covariance matrices (this was the default in versions < 0.7-1). See the em.h1.args option to control the EM algorithm.

em.h1.args: List. Options related to the EM algorithm that is used to estimate the means and (co)variances of the unrestricted (h1) model. This is only relevant for multilevel models, and for single-level models with missing data (and missing = "ml" or missing = "two.stage"). The default values are em.h1.args = list(max_iter = "default", tol = "default", min_variance = 1e-05, warn = TRUE, non_pd_action = "stop", non_pd_tol = 1e-05, acceleration = "squarem", fused = TRUE). The max_iter element sets the maximum number of EM iterations; the tol element sets the tolerance used to determine convergence. If "default", they are set to 500

and 1e-05 respectively in the single-level setting (where the EM algorithm stops when the largest absolute change in the parameter values is smaller than `tol`), and to 5000 and 1e-04 respectively in the multilevel setting (where the EM algorithm stops when the change in the loglikelihood is smaller than `tol`). The `min_variance` element sets the minimum value a variance can take during the EM iterations (multilevel only): whenever a within-level variance drops below this value, it is fixed to this value (and the corresponding covariances are set to zero). If the `warn` element is `TRUE` (the default), a warning is produced if the EM algorithm reaches `max_iter` iterations without converging. The `non_pd_action` element controls what happens if an EM estimated variance-covariance matrix is (near) singular: if any variance is (near) zero, or if the smallest eigenvalue of the matrix is smaller than the `non_pd_tol` element. This typically happens when a (within or between) variance of an observed variable approaches zero, but the full matrix may also be (near) singular. If `non_pd_action = "stop"` (the default), estimation stops with an error message; if `"warn"`, a warning is produced, and estimation continues; if `"none"`, estimation continues silently. The `acceleration` element controls whether the EM iterations are accelerated. If `"squarem"` (the default), the SQUAREM scheme (Varadhan & Roland, 2008) is used: every cycle takes two EM steps, extrapolates, and takes one stabilizing EM step from the extrapolated point, with a loglikelihood safeguard; this typically reduces the number of EM steps considerably (in particular when some variances approach zero), and converges to the same solution. If `"qn"`, a quasi-Newton scheme (Zhou, Alexander & Lange, 2011; the `"qn"` method of the **turboEM** package) is used instead: a low-rank secant approximation of the EM map is used to take a quasi-Newton step towards the fixed point, again with a loglikelihood safeguard. If `"none"`, the plain (unaccelerated) EM iterations are used. If the `fused` element is `TRUE` (the default), the loglikelihood is computed as a cheap byproduct of the E-step quantities (only used in the two-level setting with missing data, for the plain EM iterations); if `FALSE`, a separate (stand-alone) loglikelihood evaluation is used in every iteration; both give identical results.

`h1.information`**:** If `"structured"` (the default), the unrestricted (h1) information part of the (expected, first.order or observed if h1 is used) information matrix is based on the structured, or model-implied statistics (model-implied covariance matrix, model-implied mean vector, etc.). If `"unstructured"`, the unrestricted (h1) information part is based on sample-based statistics (observed covariance matrix, observed mean vector, etc.) If the argument is a vector with two elements, the first element is used for the computation of the standard errors, while the second element is used for the (robust) test statistic.

`observed.information`**:** If `"hessian"`, the observed information matrix is based on the hessian of the objective function. If `"h1"`, an approximation is used that is based on the observed information matrix of the unrestricted (h1) model. If the argument is a vector with two elements, the first element is used for the computation of the standard errors, while the second element is used for the (robust) test statistic.

`se`**:** If `"standard"`, conventional standard errors are computed based on inverting the (expected, observed or first.order) information matrix. 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`**:** Character vector. See the documentation of the `lavTest` function for a full list. Multiple

names of test statistics can be provided. If "default", the value depends on the values of other arguments. See also the lavTest function to extract (alternative) test statistics from a fitted lavaan object. FMG test names such as "peba4", "pols3", "pall", and "all" can be requested here for supported ML-family models. The convenience value "fmg" resolves to "peba4_rls" for one-model tests.

standard.test: Character. Choose the test statistic that will be used to compute fit measures (like CFI or RMSEA). The default is "standard", but it could also be (for example) "Browne.residual.nt", or an FMG test name such as "peba4" or "fmg".

scaled.test: Character. Choose the test statistic that will be scaled (if a scaled test statistic is requested). The default is "standard", but it could also be (for example) "Browne.residual.nt".

gamma.n.minus.one Logical. If TRUE, we divide the Gamma matrix by N-1 (instead of the default N).

gamma.unbiased Logical. If TRUE, we compute an unbiased version for the Gamma matrix. Only available for single-level complete data and when conditional.x = FALSE and fixed.x = FALSE (for now). Suffixless FMG test names use this option to choose between biased and unbiased Gamma.

se.delta.second.order Logical. If FALSE (the default), the standard errors of defined parameters (introduced with the := operator) and of standardized parameters (as returned by standardizedSolution() or lavInspect(., "vcov.std.*")) are computed using the first-order delta method. If TRUE, the second-order delta method is used, which adds the term $\frac{1}{2}\text{tr}(HVHV)$ (where H is the Hessian of the transformed parameter and V is the parameter covariance matrix) to the first-order variance approximation. This may improve accuracy when the transformation is a strongly non-linear function of the model parameters. Has no effect when standard errors are obtained by bootstrapping or by the Monte Carlo method.

se.def Character. Method for computing standard errors (and confidence intervals) of defined parameters (introduced with the := operator). The default is "default" (or equivalently "delta"), which uses the delta method (respecting se.delta.second.order). Set to "monte.carlo" to use the Monte Carlo method of Preacher and Selig (2012). Under this method, R draws are taken from $MVN(\hat{\theta}, \widehat{Var}(\hat{\theta}))$, the defined parameter is evaluated for each draw, and standard errors are obtained as the standard deviation of these realizations. Confidence intervals (in parameterEstimates() and standardizedSolution()) are based on percentile bounds of the Monte Carlo distribution, and therefore can be asymmetric (just like bootstrap intervals). Has no effect when se = "bootstrap".

monte.carlo List of settings for the Monte Carlo method (used when se.def = "monte.carlo"). Currently recognizes: R (integer; number of Monte Carlo draws; default 20000) and seed (integer or NULL; optional seed for reproducibility).

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

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

Optimization options:

integration.ngh: Integer. The number of Gauss-Hermite quadrature points (per dimension) used for numerical integration. The default is 21. Numerical integration is used when estimator = "MML", and for two-level models with random slopes for *latent* covariates (the rv() modifier; only the latent-covariate slopes are integrated numerically, all other random effects are handled in closed form).

control: A list containing control parameters passed to the external optimizer. By default, lavaan uses "nlminb". See the manpage of nlminb for an overview of the control parameters. If another (external) optimizer is selected, see the manpage for that optimizer to see the possible control parameters.

optim.method: Character. The optimizer that should be used. For unconstrained optimization or models with only linear equality constraints (i.e., the model syntax does not include any "==", ">" or "<" operators), the available options are "nlminb" (the default), "BFGS", "L-BFGS-B". These are all quasi-newton methods. A Gauss-Newton (Fisher scoring) method with Levenberg-Marquardt damping is also available (optim.method = "GN"); it can be used with the "ML", "GLS", "WLS", "DWLS", "ULS", "DLS", "NTRLS" and "catML" estimators, and is the default when estimator = "DLS". For multilevel models, an EM algorithm is also available (optim.method = "em"); this is the default for two-level models with missing = "ml" (as the EM algorithm is more robust in that setting; this also mimics Mplus); see the em.args option to control the EM algorithm. For two-level models with random slopes (the rv() modifier), optim.method = "em" uses a dedicated EM algorithm. Note that for any optim.method = "em" fit, variances that are fixed to zero are replaced by the (small) value em.args$zerovar_offset (as in Mplus, which uses the same 'minimum variance' device); as a result, the reported loglikelihood may differ slightly from an optim.method = "nlminb" fit of the same model (which uses the exact zero values). For constrained optimization, the only available option is "nlminb_constr", which uses an augmented Lagrangian minimization algorithm.

optim.force.converged: Logical. If TRUE, pretend the model has converged, no matter what.

optim.fix.saturated: Logical. Only used for multilevel models. If TRUE (the default), and one or more blocks (levels) of the model are saturated (i.e., all variances/covariances and means/intercepts of the observed variables of that block are free and unconstrained), the free parameters of these blocks are (temporarily) fixed to their h1 (unrestricted model) estimates during optimization, as their estimated values are already known. This often helps convergence. After the optimization, they are treated as free parameters again (e.g., when standard errors are computed).

optim.dx.tol Numeric. Tolerance used for checking if the elements of the (unscaled) gradient are all zero (in absolute value). The default value is 0.001.

gn.args: List. Options related to the Gauss-Newton optimizer (optim.method = "GN"). The default values are gn.args = list(max_iter = 200, tol_x = 1e-05, tol_g = 1e-06). The max_iter element sets the maximum number of Gauss-Newton iterations. Optimization stops when either the (scaled) gradient of the discrepancy function is smaller than tol_g, or the root mean square of the difference between the old and new parameter values is smaller than tol_x.

bounds: Only used if optim.method = "nlminb". If logical: FALSE implies no bounds are imposed on the parameters. If TRUE, this implies bounds = "wide". If character, possible options are "none" (the default), "standard", "wide", "pos.var", "pos.ov.var", and "pos.lv.var". If bounds = "pos.ov.var", the observed variances are forced to be nonnegative. If bounds = "pos.lv.var", the latent variances are forced to be nonnegative. If bounds = "pos.var", both observed and latent variances are forced to be nonnegative. If bounds = "standard", lower and upper bounds are computed for observed and latent variances, and factor loadings. If bounds = "wide", lower and upper bounds are computed for observed and latent variances, and factor loadings; but the range of the bounds is enlarged (allowing again for slightly negative variances).

optim.bounds: List. This can be used instead of the bounds argument to allow more control. Possible elements of the list are lower, upper, lower.factor and upper.factor. All of

these accept a vector. The lower and upper elements indicate for which type of parameters bounds should be computed. Possible choices are "ov.var", "lv.var", "loadings" and "covariances". The lower.factor and upper.factor elements should have the same length as the lower and upper elements respectively. They indicate the factor by which the range of the bounds should be enlarged (for example, 1.1 or 1.2; the default is 1.0). Other elements are min.reliability.marker which sets the lower bound for the reliability of the marker indicator (if any) of each factor (default is 0.1). Finally, the min.var.lv.endo element indicates the lower bound of the variance of any endogenous latent variable (default is 0.0).

em.args: List. Options related to the EM algorithm that is used to estimate the model parameters when optim.method = "em" (currently only available for multilevel models, with complete or missing data). The default values are em.args = list(max_iter = 10000, fx_tol = 1e-08, dx_tol = 1e-04, zerovar_offset = 1e-04, acceleration = "squarem", fused = TRUE). The max_iter element sets the maximum number of EM iterations. The EM algorithm stops when either the change in the loglikelihood is smaller than fx_tol, or the largest absolute change in the parameter values is smaller than dx_tol. The zerovar_offset element is used as the starting value for variances that are fixed to zero, as the EM algorithm cannot handle exact zero variances. The acceleration element controls whether the EM iterations are accelerated by the SQUAREM scheme, and the fused element controls whether the loglikelihood is computed as a byproduct of the E-step (see the em.h1.args option for details on both). If the nlminb_handoff element is TRUE (the default), and the EM iterations end with a non-negligible gradient (larger than dx_tol; this typically happens when the loglikelihood stalls before the gradient has converged), the solution is refined by a quasi-Newton (nlminb) run, warm-started at the EM values; the refined solution is only accepted if nlminb converges to an (at least) equally good solution; otherwise, the EM solution is retained. If, in addition, the early_handoff element is TRUE (the default), the EM phase stops early (using a loosened tolerance, max(fx_tol, 0.001)), as it only needs to provide a reliable warm start for nlminb; this is usually considerably faster, and gives the same solution; if nlminb fails from the early start, a safety net resumes the EM iterations with the strict fx_tol (and tries the hand-off once more), reproducing the non-early behavior. If verbose = TRUE, only the EM (or SQUAREM) cycles are shown; set the verbose element to TRUE to also show the optimizer output of each (inner) M-step.

Parallelization options (currently only used for bootstrapping):

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 set this to the number of available CPUs. By default this is 2 (following CRAN policy); you may set this to a higher value (for example, the number of cores as detected by parallel::detectCores() minus one) if you wish to use more processes.

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 lavBootstrap call.

iseed An integer to set the seed. Or NULL if no reproducible results are needed. This works for both serial (non-parallel) and parallel settings. Internally, RNGkind() is set to "L'Ecuyer-CMRG" if parallel = "multicore". If parallel = "snow" (under windows), parallel::clusterSetRNGStream() is called which automatically switches to "L'Ecuyer-CMRG". When iseed is not NULL, .Random.seed (if it exists) in the global environment is left untouched.

Categorical estimation options:

`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"` option. 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.

`allow.empty.cell`: Logical. If `TRUE`, ignore situations where an ordinal variable has fewer categories than expected, or where a category is empty in a specific group. This argument is currently used by the blavaan package (which uses lavaan to set up the model). The argument is not expected to salvage a lavaan model that results in errors.

Starting values options:

`start`: If it is a character string, the two options are currently `"simple"` and `"default"`. In the first case, all parameter values are set to zero, except the factor loadings and (residual) variances, which are set to 0.7 and 1.0 respectively. When `start` is `"default"`, the factor loadings are estimated using the fabin3 estimator (tsls) per factor. The residual variances of observed variables are set to half the observed variance, and all other (residual) variances are set to 0.05. The remaining parameters (regression coefficients, covariances) are set to zero. If `start` is a numerical vector, it should contain the starting values for all the free parameters (in the same order as the parameter table). If `start` is a fitted object of class `lavaan`, the estimated values of the corresponding parameters will be extracted. If it is a parameter table, for example the output of the `parameterEstimates()` function, the values of the `est` or `start` or `ustart` column (whichever is found first) will be extracted.

`rstarts`: Integer. The number of refits that lavaan should try with random starting values. Random starting values are computed by drawing random numbers from a uniform distribution. Correlations are drawn from the interval [-0.5, +0.5] and then converted to covariances. Lower and upper bounds for (residual) variances are computed just like the standard bounds in bounded estimation. Random starting values are not computed for regression coefficients (which are always zero) and factor loadings of higher-order constructs (which are always unity). From all the runs that converged, the final solution is the one that resulted in the smallest value for the discrepancy function.

Check options:

`check.start`: Logical. If `TRUE`, the starting values are checked for possibly inconsistent values (for example values implying correlations larger than one). If needed, a warning is given.

`check.sigma.pd`: Character. If `"chol"`, a Cholesky decomposition is used to check if the model implied covariance matrix (Sigma) is positive definite. If `"eigen"`, an eigen decomposition is used to check if Sigma is positive definite. The latter was the default until 0.6-20. From 0.6-21 onwards, `"chol"` became the default (as it is much faster).

`check.gradient`: Logical. If `TRUE`, and the model converged, a warning is given if the optimizer reported a (local) solution while not all elements of the (unscaled) gradient (as seen by the optimizer) are (near) zero, as they should be (the tolerance used is 0.001).

`check.post`: Logical. If `TRUE`, and the model converged, a check is performed after (post) fitting, to verify if the solution is admissible. This implies that all variances are non-negative, and all the model-implied covariance matrices are positive (semi-)definite. For the latter test, we tolerate a tiny negative eigenvalue that is smaller than .Machine$double.eps^(3/4), treating it as being zero.

`check.vcov`: Logical. If `TRUE`, and the model converged, we check if the variance-covariance matrix of the free parameters is positive definite. We take into account possible equality and active inequality constraints. If needed, a warning is given.

`check.lv.names`: Logical. If `TRUE`, and latent variables are defined in the model, lavaan will stop with an error message if a latent variable name also occurs in the data (implying it is also an observed variable).

Verbosity options:

`verbose`: If `TRUE`, show what lavaan is doing. During estimation, the function value is printed out during each iteration.

`warn`: If `FALSE`, suppress all lavaan-specific warning messages.

`debug`: If `TRUE`, debugging information is printed out.

Miscellaneous:

`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.

`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"`. See `mimic` for the list of options that are affected by each setting.

`representation`: If `"LISREL"` the classical LISREL matrix representation is used to represent the model (using the all-y variant). No other options are available (for now).

`implied`: Logical. If `TRUE`, compute the model-implied statistics, and store them in the implied slot.

`h1`: Logical. If `TRUE`, compute the unrestricted model and store the unrestricted summary statistics (and perhaps a loglikelihood) in the h1 slot.

`baseline`: Logical. If `TRUE`, compute a baseline model (currently always the independence model, assuming all variables are uncorrelated) and store the results in the baseline slot.

`baseline.conditional.x.free.slopes`: Logical. If `TRUE`, and `conditional.x = TRUE`, the (default) baseline model will allow the slope structure to be unrestricted.

`fit.by.level`: Logical. Only used for multilevel models. If `TRUE`, fit (and store) the partially saturated models that are needed to compute level-specific fit measures: for each level, an auxiliary model is fitted where the model at that level is retained, while the model at the other level is saturated (in addition, level-specific baseline models are fitted as well). See the `level` argument of `fitMeasures`. Set to `FALSE` to skip these additional model fits.

store.vcov Logical. If TRUE, and se= is not set to "none", store the full variance-covariance matrix of the model parameters in the vcov slot of the fitted lavaan object.

parser Character. If "new" (the default), the new parser is used to parse the model syntax. If "old", the original (pre 0.6-18) parser is used.

See Also

lavaan, mimic, estimator_iv

Examples

```
lavOptions()
lavOptions("std.lv")
lavOptions(c("std.lv", "orthogonal"))
```

---

lavParameterEstimates *Parameter Estimates*

Description

Parameter estimates of a latent variable model.

Usage

```
lavParameterEstimates(object,
                      se = TRUE, zstat = TRUE, pvalue = TRUE, ci = TRUE,
                      standardized = FALSE,
                      fmi = FALSE, plabel = FALSE,
                      level = 0.95, boot_ci_type = "perc",
                      cov_std = TRUE, fmi_options = list(),
                      rsquare = FALSE,
                      remove_system_eq = TRUE, remove_eq = TRUE,
                      remove_ineq = TRUE, remove_def = FALSE,
                      remove_nonfree = FALSE, remove_step1 = TRUE,
                      remove_unused = FALSE,
                      remove_aux = TRUE, add_attributes = FALSE,
                      output = "data.frame", header = FALSE, ...)
parameterEstimates(object,
                      se = TRUE, zstat = TRUE, pvalue = TRUE, ci = TRUE,
                      standardized = FALSE,
                      fmi = FALSE, plabel = FALSE,
                      level = 0.95, boot_ci_type = "perc",
                      cov_std = TRUE, fmi_options = list(),
                      rsquare = FALSE,
                      remove_system_eq = TRUE, remove_eq = TRUE,
                      remove_ineq = TRUE, remove_def = FALSE,
                      remove_nonfree = FALSE, remove_step1 = TRUE,
```

```
remove_unused = FALSE,
remove_aux = TRUE, add_attributes = FALSE,
output = "data.frame", header = FALSE, ...)
```

Arguments

object	An object of class lavaan.
se	Logical. If TRUE, include column containing the standard errors. If FALSE, this implies that zstat, 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.
plabel	Logical. If TRUE, show the plabel column of the parameter table in the output.
boot_ci_type	If bootstrapping was used, the type of interval required. The value should be one of "norm", "basic", "perc", or "bca.simple". For the first three options, see the help page of the boot.ci function in the boot package. The "bca.simple" option produces intervals using the adjusted bootstrap percentile (BCa) method, but with no correction for acceleration (only for bias). Note that the p-value is still computed assuming that the z-statistic follows a standard normal distribution.
standardized	Logical or character. If TRUE, standardized estimates are added to the output. Note that *SE*s and tests are still based on unstandardized estimates. Use standardizedSolution to obtain *SE*s and test statistics for standardized estimates. If a character vector is passed with any of c("std.lv","std.all","std.nox"), only the selected standardization methods are added. Note that "std.nox" only differs from "std.all" if fixed.x = TRUE; if fixed.x = FALSE, the exogenous covariates are treated as random variables (and standardized) just like any other variable, and a warning is issued. Alternatively, a character vector of (observed) variable names may be passed (for example standardized = c("x1", "x2")); in that case only the parameters involving these variables are standardized (the other observed variables are left unstandardized), and the result is added in a std.user column. This is a generalization of "std.nox", where the (observed) exogenous x variables are the ones left unstandardized.
cov_std	Logical. If TRUE, the (residual) observed covariances are scaled by the square root of the 'Theta' diagonal elements, and the (residual) latent covariances are scaled by the square root of the 'Psi' diagonal elements. If FALSE, the (residual) observed covariances are scaled by the square root of the diagonal elements of the observed model-implied covariance matrix (Sigma), and the (residual) latent covariances are scaled by the square root of diagonal elements of the model-implied covariance matrix of the latent variables.
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.

fmi_options	List. If non-empty, arguments can be provided to alter the default options when the model is fitted with the complete(d) data; otherwise, the same options are used as the original model.
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 output by removing all rows containing parameter definitions, if any.
remove_nonfree	Logical. If TRUE, filter the output by removing all rows containing fixed (non-free) parameters.
remove_step1	Logical. Only used by sam(). If TRUE, filter the output by removing all rows corresponding to the measurement parameters that are part of the first step.
remove_unused	Logical. If TRUE, filter the output by removing all rows containing automatically added parameters (user == 0) that are nonfree, and with their final (est) values fixed to their default values (typically 1 or 0); currently only used for intercepts and scaling-factors.
remove_aux	Logical. If TRUE (the default), filter the output by removing all rows corresponding to auxiliary (aux=) variables that were added to the model as saturated correlates (only relevant when missing = "ml" and the aux= argument was used).
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 op column contains r2, to indicate that the values in the est column are rsquare values.
add_attributes	Deprecated argument. Please use output= instead.
output	Character. If "data.frame", the parameter table is displayed as a standard (albeit lavaan-formatted) data.frame. If "text" (or alias "pretty"), the parameter table is formatted and displayed with subsections (as used by the summary function).
header	Logical. Only used if output = "text". If TRUE, print a header at the top of the parameter list. This header contains information about the information matrix, whether the saturated (h1) model is structured or unstructured, and which type of standard errors are shown in the output.
...	to support old argument names

Value

A data.frame containing the estimated 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.

If the model was fitted with se = "bootstrap" and boot_ci_type = "bca", the empirical-influence design matrix used to compute the acceleration constant is stored as the "design" attribute of the

returned data.frame (retrieve with `attr(x, "design")`). The parameter estimates in the bootstrap samples themselves are available via `lavInspect(object, "coef.boot")`.

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)
lavParameterEstimates(fit)
lavParameterEstimates(fit, output = "text")
```

---

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

Description

The main purpose of the `lavPredict()` function is to compute (or 'predict') individual scores for the latent variables in the model ('factor scores'). NOTE: the goal of this function is NOT to predict future values of dependent variables as in the regression framework! (For models with only continuous observed variables, the function `lavPredictY()` supports this.)

Usage

```
lavPredict(object, newdata = NULL, type = "lv", method = "EBM",
           transform = FALSE, se = "none", acov = "none",
           label = TRUE, fsm = FALSE, mdist = FALSE, rel = FALSE,
           append_data = FALSE, assemble = FALSE,
           level = 1L, optim_method = "bfgs", eta = NULL,
           parallel = c("auto", "no", "multicore", "snow"),
           ncpus = NULL, cl = NULL,
           drop_list_single_group = TRUE,
           mdist_draws = 2000L,
           ...)
```

Arguments

`object`	An object of class `lavaan`.
`newdata`	An optional data.frame, containing the same variables as the data.frame used when fitting the model in object. For multilevel models, the cluster variable(s) must be included; the cluster structure and (for `missing = "ml"`) the missing-data patterns are rebuilt from newdata, so the predictions are conditional on the newdata only (the cluster labels need not match those of the original data).

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. If "yhat", the estimated values for the observed indicators are computed, given the user-specified values for the latent variables provided by the eta argument. If "fy", densities (or probabilities) are computed for each observed indicator, given the user-specified values for the latent variables provided by the eta argument.

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 two options are "EBM" for the Empirical Bayes Modal approach, and "ML" for the maximum likelihood approach. For higher-order factors (which have no observed indicators of their own), the "Bartlett" scores are computed from the collapsed measurement model (regressing the observed indicators of the lower-order factors on the higher-order factors), as in sam.

transform — Logical. If TRUE, transform the factor scores (per group) so that their mean and variance-covariance matrix matches the model-implied mean and variance-covariance matrix. This may be useful if the individual factor scores will be used in a follow-up (regression) analysis. Note: the standard errors (if requested) are not transformed (yet). The resulting factor scores are often called correlation-preserving factor scores.

se — Character. If "none", no standard errors are computed. If "standard", standard errors are computed (assuming the parameters of the measurement model are known). The standard errors are returned as an attribute. For continuous data, the (naive) standard errors are the same for every observation. For categorical data, where the factor scores are obtained by numerical optimization (method = "EBM" or method = "ML"), the standard errors are based on the curvature of the objective at the optimum (the inverse of its Hessian, a Laplace/observed - information approximation) and therefore differ from one response pattern to the next; the result is an (nobs x nfactor) matrix per group.

acov — Similar to the "se" argument, but optionally returns the full sampling covariance matrix of the factor scores as an attribute. For continuous data this is a single matrix per group; for categorical data it is a list with one (nfactor x nfactor) matrix per observation.

label — Logical. If TRUE, the columns in the output are labeled.

fsm — Logical. If TRUE, return the factor score matrix as an attribute. Only for numeric data.

mdist — Logical. If TRUE, the (squared) Mahalanobis distances of the factor scores (if type = "lv") or the casewise residuals (if type = "resid") are returned as an attribute. If (some of) the data is ordered categorical, the distances are generalized as in Mansolf and Reise (2017): the continuous distance is replaced by its expected value over the region of the (multivariate normally distributed) latent response vector that is consistent with the observed response pattern; this expectation is approximated by Monte Carlo integration (using mdist_draws random draws per distinct response pattern; the result therefore depends on the state of the random number generator, and set.seed() can be used for reproducibility). Observed continuous variables are conditioned on, and missing values are integrated out. In this categorical setting, type = "resid" is allowed, and returns

the expected casewise residuals of the latent responses (a by-product of the same Monte Carlo integration).

`rel` Logical. Only used if `type = "lv"`. If TRUE, the factor reliabilities are returned as an attribute. (The squared values are often called the factor determinacies.)

`append_data` Logical. Only used when `type = "lv"`. If TRUE, the original data (or the data provided in the newdata argument) is appended to the factor scores.

`assemble` Logical. If TRUE, the separate groups are reassembled into a single data.frame with a group column, having the same dimensions as the original (or newdata) dataset.

`level` Integer. Only used in a multilevel SEM. If `level = 1`, only factor scores for latent variable defined at the first (within) level are computed; if `level = 2`, only factor scores for latent variables defined at the second (between) level are computed. For two-level models with `missing = "ml"`, the posterior means of the cluster-level components and of the missing values (given all the observed data; computed as in the E-step of the EM algorithm) are used, so that the factor scores are the exact posterior means given all the observed data (new in 0.7-1).

`optim_method` Character string. Only used in the categorical case. If `"nlminb"` (the default in 0.5), the `"nlminb()"` function is used for the optimization. If `"bfgs"` or `"BFGS"` (the default in 0.6), the `"optim()"` function is used with the BFGS method.

`eta` An optional matrix or list, containing latent variable values for each observation. Used for computations when `type = "ov"`.

`parallel` Character. Only used in the categorical case, where factor scores are computed by a per-observation numerical optimization. The options are `"no"` (serial), `"multicore"` (fork-based, not available on Windows) and `"snow"` (a PSOCK cluster). The default, `"auto"`, uses `"multicore"` automatically (on non-Windows platforms) when a large number of distinct optimizations is required, and runs serially otherwise. Because the computation is deterministic, the results do not depend on the value of this argument.

`ncpus` Integer. The number of processes to use in parallel computation. By default, this is 2 (following CRAN policy). You may set this to a higher value (for example, the number of available cores, minus two) if you wish to use more processes.

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

`drop_list_single_group`
Logical. 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.

`mdist_draws` Integer. Only used if `mdist = TRUE` and (some of) the data is ordered categorical: the number of Monte Carlo draws per distinct response pattern used to approximate the expected Mahalanobis distances.

`...` To support old argument names.

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 variables (in the regression sense). In other words, the structural component is completely ignored (for now).

References

For an overview (and evaluation) of the various factor score methods, see:

Grice, J. W. (2001). Computing and evaluating factor scores. *Psychological Methods, 6*(4), 430-450. doi:10.1037/1082989X.6.4.430

For the (continuous) regression and Bartlett methods, see:

Bartlett, M. S. (1937). The statistical conception of mental factors. *British Journal of Psychology, 28*, 97-104. doi:10.1111/j.20448295.1937.tb00863.x

Bentler, P. M., & Yuan, K.-H. (1997). Optimal conditionally unbiased equivariant factor score estimators. In M. Berkane (Ed.), *Latent variable modeling and applications to causality* (pp. 259-281). New York: Springer-Verlag. doi:10.1007/9781461218425_14

For the (categorical) Empirical Bayes Modal (EBM) and Maximum Likelihood (ML) methods, see:

Skrondal, A., & Rabe-Hesketh, S. (2004). *Generalized latent variable modeling: Multilevel, longitudinal, and structural equation models.* Boca Raton, FL: Chapman & Hall/CRC.

For the Mahalanobis distances with ordered categorical data (`mdist = TRUE`), see:

Mansolf, M., & Reise, S. P. (2017). Case diagnostics for factor analysis of ordered categorical data with applications to person-fit measurement. *Structural Equation Modeling: A Multidisciplinary Journal, 25*(1), 86-100. doi:10.1080/10705511.2017.1367926

For the correlation-preserving factor scores (`transform = TRUE`), see:

ten Berge, J. M. F., Krijnen, W. P., Wansbeek, T., & Shapiro, A. (1999). Some new results on correlation-preserving factor scores prediction methods. *Linear Algebra and its Applications, 289*(1-3), 311-318. doi:10.1016/S00243795(97)100076

See Also

`lavPredictY` to predict y-variables given x-variables.

Examples

```
data(HolzingerSwineford1939)

## 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"))


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

```
## standard errors for the factor scores (se =)
## ------------------------------------------

## the standard errors are returned as the "se" attribute (a list, one
## (nobs x nfactor) matrix per group)

## for continuous indicators, the (naive) standard errors are the same
## for every observation
fscores <- lavPredict(fit, se = "standard")
attr(fscores, "se")[[1]]

## for categorical indicators, the factor scores are obtained by numerical
## optimization, and their standard errors differ from one response pattern
## (observation) to the next

## (a two-factor model is used here, as the numerical optimization and the
##  Monte Carlo integration below both get expensive as the number of
##  ordered indicators grows)
HS.model.ord <- ' visual  =~ x1 + x2 + x3
                  textual =~ x4 + x5 + x6 '
HS.ord <- HolzingerSwineford1939
HS.ord[ , paste0("x", 1:6)] <-
    lapply(HS.ord[ , paste0("x", 1:6)], function(x) ordered(cut(x, 3)))
fit.ord <- cfa(HS.model.ord, data = HS.ord, ordered = paste0("x", 1:6))
fscores <- lavPredict(fit.ord, se = "standard")
head(attr(fscores, "se")[[1]])


## ------------------------------------------------
## casewise Mahalanobis distances (case diagnostics)
## ------------------------------------------------

## continuous data: the squared distances are exact
fscores <- lavPredict(fit, mdist = TRUE)
head(attr(fscores, "mdist")[[1]])

## ordered categorical data: expected (squared) distances of the latent
## responses, obtained by Monte Carlo integration (Mansolf & Reise, 2017);
## set the seed for reproducibility
set.seed(123)
resid.ord <- lavPredict(fit.ord, type = "resid", mdist = TRUE)
## expected casewise residuals of the latent responses:
head(resid.ord)
## expected squared residual-based distances (person fit):
head(attr(resid.ord, "mdist")[[1]])


## ----------------------------------------
## merge factor scores to original data.frame
## ----------------------------------------

idx <- lavInspect(fit, "case.idx")
fscores <- lavPredict(fit)
```

```
## loop over factors
for (fs in colnames(fscores)) {
  HolzingerSwineford1939[idx, fs] <- fscores[ , fs]
}
head(HolzingerSwineford1939)


## multigroup models return a list of factor scores (one per group)
data(HolzingerSwineford1939)
mgfit <- update(fit, group = "school", group.equal = c("loadings","intercepts"))

idx <- lavInspect(mgfit, "case.idx") # list: 1 vector per group
fscores <- lavPredict(mgfit)         # list: 1 matrix per group
## loop over groups and factors
for (g in seq_along(fscores)) {
  for (fs in colnames(fscores[[g]])) {
    HolzingerSwineford1939[ idx[[g]], fs] <- fscores[[g]][ , fs]
  }
}
head(HolzingerSwineford1939)

## ------------------------------------
## Use factor scores in subsequent models
## ------------------------------------

## see Examples in semTools package: ?plausibleValues
```

`lavPredictY` *Predict the values of y-variables given the values of x-variables*

Description

This function can be used to predict the values of (observed) y-variables given the values of (observed) x-variables in a structural equation model.

Usage

```
lavPredictY(object, newdata = NULL,
            ynames = lav_object_vnames(object, "ov.y"),
            xnames = lav_object_vnames(object, "ov.x"),
            method = "conditional.mean",
            label = TRUE, assemble = TRUE,
            force_zero_mean = FALSE,
            lambda = 0,
            ...)
```

Arguments

`object`	An object of class `lavaan`.

newdata — An optional data.frame, containing the same variables as the data.frame that was used when fitting the model in object. This data.frame should also include the y-variables (although their values will be ignored). Note that if no meanstructure was used in the original fit, we will use the saturated sample means of the original fit as substitutes for the model-implied means. Alternatively, refit the model using meanstructure = TRUE. For a multiple-group model, newdata can also be a *list* with one data.frame per group (in the same order as the groups in object). An element may be NULL to skip that group. This is useful when the groups have a different set of variables, so that ynames and xnames can be specified per group (as a list). In that case, the result contains predictions only for the groups for which data were provided (a single matrix if only one group was requested, otherwise a named list).

ynames — The names of the observed variables that should be treated as the y-variables. It is for these variables that the function will predict the (model-based) values for each observation. Can also be a list to allow for a separate set of variable names per group (or block).

xnames — The names of the observed variables that should be treated as the x-variables. Can also be a list to allow for a separate set of variable names per group (or block).

method — A character string. The only available option for now is "conditional.mean". See Details.

label — Logical. If TRUE, the columns of the output are labeled.

assemble — Logical. If TRUE, the predictions for the separate groups in the output are reassembled into a single data.frame with a group column, having the same dimensions as the original (or newdata) dataset.

force_zero_mean — Logical. Only relevant if there is no mean structure. If TRUE, the (model-implied) mean vector is set to the zero vector. If FALSE, the (model-implied) mean vector is set to the (unrestricted) sample mean vector.

lambda — Numeric. A lambda regularization penalty term.

... — To support old argument name ‘force.zero.mean‘

Details

This function can be used for (SEM-based) out-of-sample predictions of outcome (y) variables, given the values of predictor (x) variables. This is in contrast to the lavPredict() function which (historically) only ‘predicts’ the (factor) scores for latent variables, ignoring the structural part of the model.

When method = "conditional.mean", predictions (for y given x) are based on the (joint y and x) model-implied variance-covariance (Sigma) matrix and mean vector (Mu), and the standard expression for the conditional mean of a multivariate normal distribution. Note that if the model is saturated (and hence df = 0), the SEM-based predictions are identical to ordinary least squares predictions.

Lambda is a regularization penalty term to improve prediction accuracy that can be determined using the lavPredictY_cv function.

References

de Rooij, M., Karch, J.D., Fokkema, M., Bakk, Z., Pratiwi, B.C, and Kelderman, H. (2022) SEM-Based Out-of-Sample Predictions, Structural Equation Modeling: A Multidisciplinary Journal. doi:10.1080/10705511.2022.2061494

Molina, M. D., Molina, L., & Zappaterra, M. W. (2024). Aspects of Higher Consciousness: A Psychometric Validation and Analysis of a New Model of Mystical Experience. doi:10.31219/osf.io/cgb6e

See Also

lavPredict to compute scores for latent variables.

lavPredictY_cv to determine an optimal lambda to increase prediction accuracy.

lavResidualsY to compute the residuals for the y-variables (observed minus predicted).

Examples

```
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)

lavPredictY(fit, ynames = c("y5", "y6", "y7", "y8"),
                 xnames = c("x1", "x2", "x3", "y1", "y2", "y3", "y4"))
```

---

lavPredictY_cv *Determine an optimal lambda penalty value through cross-validation*

Description

This function can be used to determine an optimal lambda value for the lavPredictY function, based on cross-validation.

Usage

```
lavPredictY_cv(object, data = NULL,
               xnames = lav_object_vnames(object, "ov.x"),
               ynames = lav_object_vnames(object, "ov.y"),
               n_folds = 10L,
               lambda_seq = seq(0, 1, 0.1),
               ...)
```

Arguments

object	An object of class lavaan.
data	A data.frame, containing the same variables as the data.frame that was used when fitting the model in object.
xnames	The names of the observed variables that should be treated as the x-variables. Can also be a list to allow for a separate set of variable names per group (or block).
ynames	The names of the observed variables that should be treated as the y-variables. It is for these variables that the function will predict the (model-based) values for each observation. Can also be a list to allow for a separate set of variable names per group (or block).
n_folds	Integer. The number of folds to be used during cross-validation.
lambda_seq	An R seq() containing the range of lambda penalty values to be tested during cross-validation.
...	To support old argument names.

Details

This function determines an optimal lambda value for use with lavPredictY, in order to improve prediction accuracy.

References

de Rooij, M., Karch, J.D., Fokkema, M., Bakk, Z., Pratiwi, B.C, and Kelderman, H. (2022) SEM-Based Out-of-Sample Predictions, Structural Equation Modeling: A Multidisciplinary Journal. doi:10.1080/10705511.2022.2061494

Molina, M. D., Molina, L., & Zappaterra, M. W. (2024). Aspects of Higher Consciousness: A Psychometric Validation and Analysis of a New Model of Mystical Experience. doi:10.31219/osf.io/cgb6e

See Also

lavPredictY to predict the values of (observed) y-variables given the values of (observed) x-variables in a structural equation model.

Examples

```
## Not run:
colnames(PoliticalDemocracy) <- c("z1", "z2", "z3", "z4",
                                  "y1", "y2", "y3", "y4",
                                  "x1", "x2", "x3")

model <- '
  # latent variable definitions
  ind60 =~ x1 + x2 + x3
  dem60 =~ z1 + z2 + z3 + z4
  dem65 =~ y1 + y2 + y3 + y4
  # regressions
  dem60 ~ ind60
  dem65 ~ ind60 + dem60
  # residual correlations
  z1 ~~ y1
  z2 ~~ z4 + y2
  z3 ~~ y3
  z4 ~~ y4
  y2 ~~ y4
'
fit <- sem(model, data = PoliticalDemocracy, meanstructure = TRUE)

percent <- 0.5
nobs <- lavInspect(fit, "ntotal")
idx <- sort(sample(x = nobs, size = floor(percent*nobs)))

xnames = c("z1", "z2", "z3", "z4", "x1", "x2", "x3")
ynames = c("y1", "y2", "y3", "y4")

reg.results <- lavPredictY_cv(
    fit,
    PoliticalDemocracy[-idx, ],
    xnames = xnames,
    ynames = ynames,
    n_folds = 10L,
    lambda_seq = seq(from = .6, to = 2.5, by = .1)
)
lam <- reg.results$lambda.min

lavPredictY(fit, newdata = PoliticalDemocracy[idx,],
                 ynames  = ynames,
                 xnames  = xnames,
                 lambda  = lam)

## End(Not run)
```

---

lavResiduals *Residuals*

Description

'lavResiduals' provides model residuals and standardized residuals from a fitted lavaan object, as well as various summaries of these residuals.

The 'residuals()' (and 'resid()') methods are just shortcuts to this function with a limited set of arguments.

Usage

```
lavResiduals(object, type = "cor.bentler", h1 = NULL,
    se = FALSE, zstat = TRUE, summary = TRUE, elementwise = TRUE,
    combine = FALSE, usrmr_ci_level = 0.90, usrmr_close_h0 = 0.05,
    h1_acov = "unstructured",
    add_type = TRUE, add_labels = TRUE, add_class = TRUE,
    drop_list_single_group = TRUE,
    maximum_number = 0L, n_largest = 5L, output = "list", ...)
```

Arguments

`object`	An object of class `lavaan`.
`type`	Character. If `type = "raw"`, this function returns the raw (= unscaled) difference between the observed and the expected (model-implied) summary statistics, as well as the standardized version of these residuals. If `type = "cor"`, or `type = "cor.bollen"`, the observed and model implied covariance matrices are first transformed to a correlation matrix (using `cov2cor()`), before the residuals are computed. If `type = "cor.bentler"`, both the observed and model implied covariance matrices are rescaled by dividing the elements by the square roots of the corresponding variances of the observed covariance matrix.
`h1`	Optional. A user-provided saturated (unrestricted) model supplying the observed summary statistics against which the model-implied statistics are compared. It can be a fitted `lavaan` object (its `h1` slot is used), a `lavh1` list (a list with an `implied` element), or an implied-moments list (a list with a `cov` or `res.cov` element, per block). It must be compatible with `object` (same number of blocks and same dimensions). If `NULL` (the default), the saturated statistics of `object` itself are used.
`se`	Logical. If `TRUE`, show the estimated standard errors for the residuals.
`zstat`	Logical. If `TRUE`, show the standardized residuals, which are the raw residuals divided by the corresponding (estimated) standard errors.
`summary`	Logical. If `TRUE`, show various summaries of the (possibly scaled) residuals. When `type = "raw"`, we compute the RMR. When `type = "cor.bentler"`, we compute the SRMR. When `type = "cor.bollen"`, we compute the CRMR. An unbiased version of these summaries (the URMR, USRMR and UCRMR) is also computed (Maydeu-Olivares, 2017). The two versions are reported with different inferential statistics, and this asymmetry is intentional. The sample RMR/SRMR/CRMR overestimate (are positively biased for) their population values. For these (biased) statistics, we report a standard error and a test of *exact* fit (the null hypothesis that the population

value equals zero). This test relies on the known sampling distribution of the (biased) sample statistic under exact fit (Maydeu-Olivares, 2017, Eqs. 28–30), where the bias is explicitly accounted for. For the (approximately) unbiased URMR/USRMR/UCRMR, we report a standard error, a confidence interval, and a test of *close* fit (the null hypothesis that the population value equals a small nonzero cutoff, e.g. 0.05). The confidence interval and the close-fit test require an (approximately) unbiased and normally distributed estimator, which is precisely why the unbiased version is used here. Conversely, no test of exact fit is reported for the unbiased estimator: because it is truncated at zero, its sampling distribution is degenerate at the exact-fit boundary, so the test of exact fit is (correctly) only based on the biased statistic.

`elementwise` Logical. If `TRUE` (the default), the element-wise residuals are included: the (residualized) summary statistics when `output = "list"`, or the largest-residuals table(s) when `output = "text"`. If `FALSE`, these are omitted and only the summary information is returned/shown (provided `summary = TRUE`). Combined with `summary = FALSE`, this gives full control over which parts are produced.

`combine` Logical. Only relevant when multiple blocks are involved (multiple groups, or multiple levels). If `FALSE` (the default), a separate summary table is shown for each block. If `TRUE`, the block-specific summary tables are replaced by a single overall summary table in which the residual elements are pooled across all blocks (so, e.g., the SRMR is computed from the residuals of all groups or all levels at once). The combined table is returned as the `summary` element at the top level of the result. The pooling correctly accounts for the (block-diagonal) independence of separate groups and for the (joint) sampling covariance across the levels of a multilevel model.

`usrmr_ci_level` Numeric. The confidence level of the confidence interval that is reported (in the summary) for the unbiased URMR/USRMR/UCRMR. The default is 0.90.

`usrmr_close_h0` Numeric. The value of the population URMR/USRMR/UCRMR under the null hypothesis of the test of close fit that is reported (in the summary) for the unbiased estimator. The default is 0.05.

`h1_acov` Character. If `"unstructured"`, the observed summary statistics are used as consistent estimates of the corresponding (unrestricted) population statistics. If `"structured"`, the model-implied summary statistics are used as consistent estimates of the corresponding (unrestricted) population statistics. This affects the way the asymptotic variance matrix of the summary statistics is computed.

`add_type` Logical. If `TRUE`, show the type of residuals in the output.

`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.

`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.

`maximum_number` Integer. Only used if `output = "table"`. If larger than zero, show only the first maximum_number rows of the data.frame. The default (`0L`) shows all rows.

`n_largest` Integer. Only used if `output = "text"`. The number of largest (in absolute value) residuals that are shown, per element group (covariances, variances, intercepts, thresholds, regression slopes) and per block.

`output` Character. By default, `output = "list"`, and the output is a list of elements. If `output = "table"`, the residuals are shown in a single data.frame, one row per residual element, sorted from high (in absolute value) to low. All residual elements are included: the (co)variances (`lhs~~rhs`), the means/intercepts (`lhs~1`), the thresholds (`lhs|t1`) and, when `conditional.x = TRUE`, the regression slopes (`lhs~rhs`). Variances (the diagonal) are only shown for the raw-style metrics (`type = "raw"`, `"normalized"` or `"standardized"`), not for the correlation metrics where they are structurally zero. Structurally fixed moments (e.g. saturated means or just-identified thresholds) are omitted. If `se = TRUE`, a `se` column with the standard errors is added; if `zstat = TRUE`, a `z` column with the z-statistics is added. If `output = "text"` (or, equivalently, `"pretty"`), the output is printed in the same style as the fit measures (e.g.\ the RMSEA). First, a table with the `n_largest` largest residuals (per element group, and per block) is shown, formatted like `parameterEstimates(., output = "text")` but with a 'Residual' column (and, if available, 'Std.Err' and 'Z-value' columns); its header reflects the residual `type` (e.g.\ "Largest Standardized Residuals (Cor.Bentler)" or "Largest Raw Residuals"). Next, the residual summary table(s) are shown. When multiple blocks are involved and `combine = FALSE`, one summary is printed per block; otherwise a single (combined) summary is printed.

`...` To support old argument names.

Value

If `drop_list_single_group = TRUE`, a list of (residualized) summary statistics, including type, standardized residuals, and summaries. If `drop_list_single_group = FALSE`, the list of summary statistics is nested within a list for each group.

References

Bentler, P.M. and Dijkstra, T. (1985). Efficient estimation via linearization in structural models. In Krishnaiah, P.R. (Ed.), *Multivariate analysis - VI*, (pp. 9–42). New York, NY: Elsevier.

Ogasawara, H. (2001). Standard errors of fit indices using residuals in structural equation modeling. *Psychometrika, 66*(3), 421–436. doi:10.1007/BF02294443

Maydeu-Olivares, A. (2017). Assessing the size of model misfit in structural equation models. *Psychometrika, 82*(3), 533–558. doi:10.1007/s11336-016-9552-7

Standardized Residuals in M*plus*. Document retrieved from URL http://www.statmodel.com/download/StandardizedResidua

Examples

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

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

`lavResidualsY` *Compute residuals for the y-variables given the values of the x-variables*

Description

This function can be used to compute the (case-wise) residuals for the (observed) y-variables in a structural equation model, defined as the difference between the observed values and the (model-based) predicted values that are returned by `lavPredictY`.

Usage

```
lavResidualsY(object, newdata = NULL,
              ynames = lav_object_vnames(object, "ov.y"),
              xnames = lav_object_vnames(object, "ov.x"),
              method = "conditional.mean",
              label = TRUE, assemble = TRUE,
              force_zero_mean = FALSE,
              lambda = 0,
              ...)
```

Arguments

`object`	An object of class `lavaan`.
`newdata`	An optional data.frame, containing the same variables as the data.frame that was used when fitting the model in `object`. This data.frame should also include the y-variables (their observed values are used to compute the residuals). Note that if no meanstructure was used in the original fit, we will use the saturated sample means of the original fit as substitutes for the model-implied means. Alternatively, refit the model using `meanstructure = TRUE`. For a multiple-group model, `newdata` can also be a *list* with one data.frame per group (in the same order as the groups in `object`). An element may be `NULL` to skip that group. This is useful when the groups have a different set of variables, so that `ynames` and `xnames` can be specified per group (as a list). In that case, the result contains residuals only for the groups for which data were provided (a single matrix if only one group was requested, otherwise a named list).
`ynames`	The names of the observed variables that should be treated as the y-variables. It is for these variables that the function will compute the (model-based) residuals for each observation. Can also be a list to allow for a separate set of variable names per group (or block).
`xnames`	The names of the observed variables that should be treated as the x-variables. Can also be a list to allow for a separate set of variable names per group (or block).
`method`	A character string. The only available option for now is `"conditional.mean"`. See Details.
`label`	Logical. If TRUE, the columns of the output are labeled.

assemble	Logical. If TRUE, the residuals for the separate groups in the output are re-assembled into a single data.frame with a group column, having the same dimensions as the original (or newdata) dataset.
force_zero_mean	Logical. Only relevant if there is no mean structure. If TRUE, the (model-implied) mean vector is set to the zero vector. If FALSE, the (model-implied) mean vector is set to the (unrestricted) sample mean vector.
lambda	Numeric. A lambda regularization penalty term.
...	To support old argument names.

Details

The residuals are computed as observed - predicted, where the predicted values are obtained by lavPredictY using the same arguments. See the help page of lavPredictY for more details about how the predictions are computed.

These residuals can be used, for example, to diagnose the (SEM-based) out-of-sample prediction of outcome (y) variables, or to check the distributional assumptions (for example, multivariate normality) for the y-variables given the x-variables.

References

de Rooij, M., Karch, J.D., Fokkema, M., Bakk, Z., Pratiwi, B.C, and Kelderman, H. (2022) SEM-Based Out-of-Sample Predictions, Structural Equation Modeling: A Multidisciplinary Journal. doi:10.1080/10705511.2022.2061494

See Also

lavPredictY to compute the (model-based) predicted values for the y-variables given the x-variables.

Examples

```
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)
```

```
lavResidualsY(fit, ynames = c("y5", "y6", "y7", "y8"),
                   xnames = c("x1", "x2", "x3", "y1", "y2", "y3", "y4"))
```

---

lavScores *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	Only used for the ML estimator. 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 (e.g., 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.
...	To accept old argument names with dots. No other arguments are accepted.

Details

Casewise scores are available for the ML, GLS, ULS and WLS estimators, and (since 0.7-1) for the PML (pairwise maximum likelihood) estimator. For PML, the scores are the casewise derivatives of the pairwise log-likelihood (including the weighted univariate part when missing = "available.cases"); missing = "doubly.robust" is not supported (yet). Exogenous covariates (conditional.x = TRUE) are supported for PML only.

Value

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

Author(s)

Ed Merkle for the ML case; the `remove.duplicated`, `ignore.constraints` and `remove.empty.cases` arguments were added by Yves Rosseel; Franz Classe for the WLS case; Yves Rosseel for the PML case.

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)
head(lavScores(fit))
```

---

lavSimulateData *Simulate Data From a Lavaan Model Syntax*

Description

Simulate data starting from a lavaan model syntax.

Usage

```
lavSimulateData(model = NULL, model_type = "sem", meanstructure = FALSE,
    int_ov_free = TRUE, int_lv_free = FALSE,
    marker_int_zero = FALSE, conditional_x = FALSE,
    composites = TRUE, 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 = NULL, skewness = NULL,
    kurtosis = NULL, cluster_idx = NULL, seed = NULL, empirical = FALSE,
    mass = TRUE, ordered_center = TRUE, return_type = "data.frame",
    return_fit = FALSE, debug = FALSE, standardized = FALSE)
simulateData(model = NULL, model_type = "sem", meanstructure = FALSE,
    int_ov_free = TRUE, int_lv_free = FALSE,
    marker_int_zero = FALSE, conditional_x = FALSE,
    composites = TRUE, 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 = NULL, skewness = NULL,
```

```
    kurtosis = NULL, cluster_idx = NULL, seed = NULL, empirical = FALSE,
    mass = FALSE, ordered_center = TRUE, return_type = "data.frame",
    return_fit = FALSE, debug = FALSE, standardized = FALSE)
lav_data_simulate_old(..., ordered_center = 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 (e.g., the output of the lavParTable() 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.
marker_int_zero	Logical. Only relevant if the metric of each latent variable is set by fixing the first factor loading to unity. If TRUE, it implies meanstructure = TRUE and std_lv = FALSE, and it fixes the intercepts of the marker indicators to zero, while freeing the means/intercepts of the latent variables. Only works correctly for single group, single level models.
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 reflect both sets of variables. If "default", the value is set depending on the estimator, and whether or not the model involves categorical endogenous variables.
composites	If TRUE, use the new (0.6-20) approach for handling composites.
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 of both observed and latent variables are 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.

cluster_idx
: Optional. Only used (and only available via lavSimulateData() and simulateData()) for multilevel models. An integer vector (or a list of integer vectors, one per group) giving the cluster membership of each level-1 unit. If supplied, it determines both the number of level-1 units and the number of clusters. If not supplied for a multilevel model, balanced clusters are created based on sample_nobs. Providing this argument (or a multilevel model syntax with level: blocks) routes the request to the multilevel data-generation engine.

seed
: Set random seed.

empirical
: Logical. If TRUE, the implied moments (Mu and Sigma) specify the empirical not population mean and covariance matrix.

mass
: Logical. If TRUE, use the mvrnorm() function from the MASS package (the default before 0.7-1) instead of the internal lav_mvrnorm() function.

ordered_center
: Logical. Only relevant for categorical data, which is generated by the (new) multilevel-aware engine. If TRUE (the default), each ordered variable is centered (its sample mean is subtracted) before it is cut at the model-implied thresholds; this also works when the model-implied mean is nonzero. If FALSE, the variable is cut at the thresholds directly (the behaviour of the older engine).

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 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 0.7 and 1.0 respectively. 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.

There is a single data-simulation engine. Multilevel data (model syntax containing `level:` blocks, or when the `cluster_idx` argument is provided) are generated by an internal multilevel worker; all single-level data (continuous and categorical, including the `ov_var`, `skewness`, `kurtosis`, `standardized` and `mass` options) are generated by the historical single-level worker, so that the continuous, single-level output remains byte-identical to previous versions. For multilevel data, the `ov_var`, `skewness`, `kurtosis`, `standardized` and `mass` arguments are not (yet) supported and are ignored (with a warning).

`lav_data_simulate_old()` is a deprecated wrapper kept for backward compatibility; it forwards to the unified engine (with `ordered_center = FALSE` by default, reproducing the historical threshold cut).

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 <- lavSimulateData(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)
```

---

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", pattern_as_string = 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, the information shown per row can be changed: 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 carry the suffix *.un, as in X2.un and G2.un. If object is a data.frame, only the unrestricted versions of the statistics are 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 on whether 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.

pattern_as_string — 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.

... — To support old argument names.

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), the output is a list of tables (if type = "cells") in which each list element corresponds to a pairwise table, or a single table per group (if type = "table"). 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 significance 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 cancel 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 the hypothesis cannot 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[,1:4]),
           estimator="PML")
lavTablesFitCm(fit)
lavTablesFitCp(fit)
lavTablesFitCf(fit)
```

---

lavTest *Test of exact fit*

Description

Compute a variety of test statistics evaluating the global fit of the model.

Usage

```
lavTest(lavobject, test = "standard", scaled_test = "standard",
        output = "list", drop_list_single = TRUE, ...)
```

Arguments

lavobject	An object of class lavaan.
test	Character vector. Multiple names of test statistics can be provided. If "standard" is included, a conventional chi-square test is computed. If "Browne.residual.adf" is included, Browne's residual-based test statistic using ADF theory is computed. If "Browne.residual.nt" is included, Browne's residual-based test statistic using normal theory is computed. If "Satorra.Bentler" is included, a Satorra-Bentler scaled test statistic is computed. If "Yuan.Bentler" is included, a Yuan-Bentler scaled test statistic is computed. If "Yuan.Bentler.Mplus" is included, a test statistic is computed that is asymptotically equivalent to the Yuan-Bentler scaled test statistic. If "mean.var.adjusted" or "Satterthwaite" is included, a mean and variance adjusted test statistic is computed. If "scaled.shifted" is included, an alternative mean and variance adjusted test statistic is computed (as in Mplus version 6 or higher). If "mean.var.adjusted.corrected" (aliases: "mvc", "hayakawa") or "scaled.shifted.corrected" (alias: "ssc") is included, the corresponding test statistic is computed with the corrected (unbiased) estimator of the trace of the squared UGamma matrix (Hayakawa, 2018),

which remains accurate when the number of observed variables is large relative to the sample size (where the naive estimator is severely biased, and the uncorrected adjusted tests underreject). These corrected tests are only available for single-group, single-level, complete, continuous data and `estimator = "ML"`. Following Hayakawa (2018), they are best combined with `scaled_test = "RLS"`. If `"boot"` or `"bootstrap"` or `"Bollen.Stine"` is included, the Bollen-Stine bootstrap is used to compute the bootstrap probability value of the (regular) test statistic. FMG p-value tests can be requested using names such as `"peba4"`, `"pols3"`, `"pall"`, or `"all"`. The convenience value `"fmg"` resolves to `"peba4_rls"` for one-model tests. Suffixless FMG names use the standard ML chi-square statistic by default; set `scaled_test = "Browne.residual.nt.model"` or `scaled_test = "RLS"` to use the Browne residual normal-theory model-based statistic. Suffixes such as `"_ml"`, `"_rls"`, and `"_ug"` can be used for explicit control, for example `"peba4_ug_rls"` or `"pols3_ml"`.

`scaled_test` Character. Choose the test statistic that will be scaled (if a scaled test statistic is requested). The default is `"standard"`, but it could also be (for example) `"Browne.residual.nt"`.

`output` Character. If `"list"` (the default), return a list with all test statistics. If `"text"`, display the output as text with verbose descriptions (as in the summary output). If any scaled test statistics are included, they are printed first in a two-column format, followed by the remaining test statistics in a one-column format.

`drop_list_single`
Logical. Only used when `output = "list"`. If `TRUE` and the list is of length one (i.e. only a single test statistic), drop the outer list. If `FALSE`, return a nested list with one element per test statistic.

`...` To support old argument names.

Value

If `output = "list"`: a nested list with test statistics, or if only a single test statistic is requested (and `drop_list_single = TRUE`), a list with details for this test statistic. If `output = "text"`: the text is printed, and a nested list of test statistics (including an info attribute) is returned.

References

Hayakawa, K. (2019). Corrected goodness-of-fit test in covariance structure analysis. *Psychological Methods, 24*(3), 371–389. doi:10.1037/met0000180

Himeno, T., & Yamada, T. (2014). Estimations for some functions of covariance matrix in high dimension under non-normality and its applications. *Journal of Multivariate Analysis, 130*, 27–44.

Srivastava, M. S. (2005). Some tests concerning the covariance matrix in high dimensional data. *Journal of the Japan Statistical Society, 35*(2), 251–272.

Examples

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

```
'
fit <- cfa(HS.model, data = HolzingerSwineford1939)
lavTest(fit, test = "browne.residual.adf")
lavTest(fit, test = "peba4", output = "text")
lavTest(fit, test = "pols3", scaled_test = "RLS", output = "text")
# corrected adjusted test (Hayakawa, 2018), based on the RLS statistic
lavTest(fit, test = "mean.var.adjusted.corrected", scaled_test = "RLS",
        output = "text")
```

`lavTestLRT` *LRT test*

Description

LRT test for comparing (nested) lavaan models.

Usage

```
lavTestLRT(object, ..., method = "default", test = "default",
           a_method = "delta", scaled_shifted = 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"`, `"satorra.2000"`, and `"standard"`. See details. Note that the robust options (`"satorra.bentler.2001"`, `"satorra.bentler.2010"` and `"satorra.2000"`) require that the models were fitted with a robust (scaled) test statistic (for example, using `estimator = "MLM"` or `test = "satorra.bentler"`). If none of the models were fitted with a robust test statistic, but a robust `method=` is requested, a warning is issued, the `method=` argument is ignored, and a standard (regular) chi-squared difference test is computed instead.
`test`	Character string specifying which scaled test statistics to use, in case multiple scaled `test=` options were requested when fitting the model(s). See details. FMG nested tests can be requested using names such as `"pall"`, `"all"`, `"peba4"`, or `"pols3"`; the convenience value `"fmg"` resolves to the recommended nested default.
`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. Note that if `a_method = "exact"`, the models must be nested in the parameter sense, while if `a_method = "delta"`, they only need to be nested in the covariance matrix sense.

scaled_shifted Logical. Only used when method = "satorra.2000". If TRUE, we use a scaled and shifted test statistic. If FALSE, the statistic depends on the test that was used to fit the models: when the models were fitted with a mean-and-variance adjusted test (test = "mean.var.adjusted" or "scaled_shifted"), a mean and variance adjusted (Satterthwaite style) difference test is used; when they were fitted with a (Satorra-)Bentler scaled test (test = "satorra.bentler", "yuan.bentler" or "yuan.bentler.mplus"), a scaled (non-shifted) difference test is used. The title printed above the anova table reflects the statistic that is actually used.

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. If "browne.residual.adf" (alias "browne") or "browne.residual.nt", the standard chi-squared difference is calculated from each model's residual-based statistic.

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.

The only test= options that currently have actual consequences are "satorra.bentler", "yuan.bentler", or "yuan.bentler.mplus" because "mean.var.adjusted" and "scaled_shifted" are currently distinguished by the scaled_shifted argument. See lavOptions for details about the test= options implied by robust estimator= options. The "default" is to select the first available scaled statistic, if any. To check which test(s) were computed when fitting your model(s), use lavInspect(fit, "options")$test.

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="satorra.2000", the original formulas of Satorra (2000) are used. The latter is used for model comparison when ... contains additional (nested) models. Even when test statistics are scaled in object or ..., users may request the method="standard" test statistic, without a robust adjustment.

Conversely, a robust method= ("satorra.bentler.2001", "satorra.bentler.2010" or "satorra.2000") can only be applied when the models were fitted with a robust (scaled) test statistic. If none of the models in object or ... were fitted with a robust test statistic, a robust method= cannot be honored: in that case a warning is issued, the method= argument is ignored, and a standard (regular) chi-squared difference test is returned.

FMG nested tests are available when type = "Chisq" and test is an FMG test name. The supported nested FMG methods are "pall", "all", "peba" and "pols" variants. The convenience value "fmg" resolves to "pall_ug_rls" for single-group comparisons and "pall_ug_ml" for multiple-group comparisons. They use the Satorra (2000) UGamma projection; method may be left at "default" or set to "standard" or "satorra.2000". Other nested LRT methods are not used for FMG tests.

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, after reordering the models according to their degrees of freedom.

Note

If there is a lavaan model stored in `object@external$h1.model`, it will be added to ...

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, UK: 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. doi:10.1007/BF02296192

Satorra, A., & Bentler, P. M. (2010). Ensuring positiveness of the scaled difference chi-square test statistic. *Psychometrika, 75*(2), 243-248. doi:10.1007/s113360099135y

Foldnes, N., Moss, J., & Gronneberg, S. (2024). Improved goodness of fit procedures for structural equation models. *Structural Equation Modeling: A Multidisciplinary Journal*, 1-13. doi:10.1080/10705511.2024.2372028

Foldnes, N., Gronneberg, S., & Moss, J. (2026). Penalized eigenvalue block averaging: Extension to nested model comparison and Monte Carlo evaluations. *Behavior Research Methods, 58*(4). doi:10.3758/s13428026029684

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)


## When multiple test statistics are selected when the model is fitted,
## use the type= and test= arguments to select a test for comparison.

## refit models, requesting 6 test statistics (in addition to "standard")
t6.1 <- cfa(HS.model, data = HolzingerSwineford1939,
            test = c("browne.residual.adf","scaled.shifted","mean.var.adjusted",
                     "satorra.bentler", "yuan.bentler", "yuan.bentler.mplus"))
t6.0 <- cfa(HS.model, data = HolzingerSwineford1939, orthogonal = TRUE,
            test = c("browne.residual.adf","scaled.shifted","mean.var.adjusted",
                     "satorra.bentler", "yuan.bentler", "yuan.bentler.mplus"))
```

```
## By default (test="default", type="Chisq"), the first scaled statistic
## requested will be used. Here, that is "scaled.shifted"
lavTestLRT(t6.1, t6.0)
## But even if "satorra.bentler" were requested first, method="satorra.2000"
## provides the scaled-shifted chi-squared difference test:
lavTestLRT(t6.1, t6.0, method = "satorra.2000")
## == lavTestLRT(update(t6.1, test = "scaled.shifted"), update(t6.0, test = "scaled.shifted"))

## The mean- and variance-adjusted (Satterthwaite) statistic implies
## scaled_shifted = FALSE
lavTestLRT(t6.1, t6.0, method = "satorra.2000", scaled_shifted = FALSE)

## Because "satorra.bentler" is not the first scaled test in the list,
## we MUST request it explicitly:
lavTestLRT(t6.1, t6.0, test = "satorra.bentler") # method="satorra.bentler.2001"
## == lavTestLRT(update(t6.1, test = "satorra.bentler"),
##               update(t6.0, test = "satorra.bentler"))
## The "strictly-positive test" is necessary when the above test is < 0:
lavTestLRT(t6.1, t6.0, test = "satorra.bentler", method = "satorra.bentler.2010")

## Likewise, other scaled statistics can be selected:
lavTestLRT(t6.1, t6.0, test = "yuan.bentler")
## == lavTestLRT(update(t6.1, test = "yuan.bentler"),
##               update(t6.0, test = "yuan.bentler"))
lavTestLRT(t6.1, t6.0, test = "yuan.bentler.mplus")
## == lavTestLRT(update(t6.1, test = "yuan.bentler.mplus"),
##               update(t6.0, test = "yuan.bentler.mplus"))

## To request the difference between Browne's (1984) residual-based statistics,
## rather than statistics based on the fitted model's discrepancy function,
## use the type= argument:
lavTestLRT(t6.1, t6.0, type = "browne.residual.adf")

## Despite requesting multiple robust tests, it is still possible to obtain
## the standard chi-squared difference test (i.e., without a robust correction)
lavTestLRT(t6.1, t6.0, method = "standard")
## == lavTestLRT(update(t6.1, test = "standard"), update(t6.0, test = "standard"))

## FMG nested p-values use the Satorra (2000) UGamma projection
lavTestLRT(fit1, fit0, method = "satorra.2000", test = "pall")
```

`lavTestScore` *Score test*

Description

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

Usage

```
lavTestScore(object, add = NULL, release = NULL,
             univariate = TRUE, cumulative = FALSE,
             epc = FALSE, standardized = epc, cov_std = epc,
             verbose = FALSE, warn = TRUE, information = "expected", ...)
```

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 in which the equality constraints appear in the parameter table.
univariate	Logical. If TRUE, compute the univariate score statistics, one for each constraint.
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.
standardized	If TRUE, two extra columns (sepc.lv and sepc.all) in the $epc table will contain standardized values for the EPCs. In the first column (sepc.lv), standardization is based on the variances of the (continuous) latent variables. In the second column (sepc.all), standardization is based on the variances of both (continuous) observed and latent variables. (Residual) covariances are standardized using (residual) variances.
cov_std	Logical. See standardizedSolution.
verbose	Logical. Not used for now.
warn	Logical. If TRUE, print out warnings if they occur.
information	character indicating the type of information matrix to use (check lavInspect for available options). "expected" information is the default, which provides better control of Type I errors.
...	to allow use of old argumentname cov_std

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 cannot 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.

In addition to the standard (normal-theory) score test, robust versions are computed whenever the fitted object provides the ingredients for a sandwich-type covariance matrix: either because robust standard errors were requested (e.g., `se = "robust.sem"` or `se = "robust.huber.white"`), or because a scaled test statistic was requested (e.g., `test = "satorra.bentler"`). Following Satorra (2000), three robust versions are reported: a mean-scaled statistic (`score.scaled`), a mean-and-variance adjusted statistic with fractional degrees of freedom (`score.adjusted`), and the generalized, asymptotically distribution-free statistic (`score.robust`). For a single restriction, the three versions coincide. The univariate (and cumulative) score tests are then accompanied by scaled counterparts in the `X2.scaled` column. Note that for estimators that are not asymptotically efficient (e.g., `DWLS`, `ULS` and `PML`), only the robust versions have the correct asymptotic distribution.

Value

A list containing at least one `data.frame`:

- `$test`: The total score test, with columns for the score test statistic (`X2`), the degrees of freedom (`df`), and a *p* value under the χ^2 distribution (`p.value`). If robust versions could be computed, three additional rows are included: `score.scaled`, `score.adjusted` and `score.robust` (see Details).
- `$uni`: Optional (if `univariate=TRUE`). Each 1-*df* score test, equivalent to modification indices. If robust versions could be computed, the scaled statistics are provided in the `X2.scaled` and `p.value.scaled` columns. If `epc=TRUE` when adding parameters (not when releasing constraints), an unstandardized EPC is provided for each added parameter, as would be returned by `modificationIndices`.
- `$cumulative`: Optional (if `cumulative=TRUE`). Cumulative score tests, with scaled counterparts if robust versions could be computed.
- `$epc`: Optional (if `epc=TRUE`). Parameter estimates, expected parameter changes, and expected parameter values if all the tested constraints were freed.

References

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

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.

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)

# equivalently, "add" can be a parameter table specifying parameters to free,
# but must include some additional information:
PT.add <- data.frame(lhs = c("visual","textual"),
                     op = c("=~","=~"),
                     rhs = c("x9","x3"),
                     user = 10L, # needed to identify new parameters
                     free = 1, # arbitrary numbers > 0
                     start = 0) # null-hypothesized value
PT.add
lavTestScore(fit, add = PT.add) # same result as above
```

`lavTestWald` *Wald test*

Description

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

The test statistic is based on the variance-covariance matrix of the estimated parameters (vcov) of the fitted object. Therefore, if robust standard errors were requested for the fitted object (e.g., se = "robust.sem" or se = "robust.huber.white"), the main test statistic is already the generalized ('robust') Wald test described in Satorra (2000). In that case (or when a scaled test statistic was requested, so that the ingredients for a sandwich-type covariance matrix are available), additional versions of the Wald test statistic are computed as well: the standard (normal-theory) statistic (stat.standard), a mean-scaled statistic (stat.scaled), a mean-and-variance adjusted statistic with fractional degrees of freedom (stat.adjusted), and the generalized statistic (stat.robust); see Satorra (2000).

Value

A list containing at least three elements: the Wald test statistic (stat), the degrees of freedom (df), and a p-value under the chi-square distribution (p.value). If robust versions could be computed, the list also contains the standard, scaled, adjusted and generalized versions of the test statistic, together with their p-values (see Details).

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.

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)
```

mimic *The mimic= option*

Description

The mimic= argument of the lavaan function (and its wrappers cfa, sem and growth) controls a collection of primarily technical default settings, with the aim of making lavaan's output as similar as possible to that of other SEM software. This applies in particular to a few key results, such as the value of the Satorra-Bentler scaled test statistic.

The mimic= option was originally introduced to demonstrate that differences between lavaan and other SEM programs were often due to technical implementation details rather than errors in lavaan. Nevertheless, the scope of the mimic= option is intentionally limited. It is designed to reproduce a small number of important results and does not attempt to replicate every quantity reported by alternative SEM software packages.

This help page documents which options are affected by the mimic= argument.

Details

The mimic= argument does not change the model that is fitted, nor the estimator that is used. Instead, it overwrites the default values of a number of (mostly technical) options that are otherwise estimator-dependent. The user can always override these defaults by setting the corresponding option explicitly in the lavaan call.

The argument is case-insensitive and accepts a number of aliases. After normalization, only four values remain:

"lavaan": the native lavaan defaults (also selected by the value "default"); this is the default.

"Mplus": mimic the Mplus program (also selected by the value "mplus").

"EQS": mimic the EQS program (also selected by the values "eqs", "LISREL" and "lisrel").

"lm": a minimal setting used internally for plain regression (also selected by the value "regression"); not intended for general use.

mimic = "lavaan" (the default)

This is the reference setting. For maximum likelihood (ML) estimation, the following defaults are used:

likelihood = "normal": the sample covariance matrix is rescaled by a factor (N-1)/N, so the loglikelihood and standard errors are based on a division by N (rather than N-1).

conditional.x = FALSE: the exogenous x covariates are regressed out first, and a conditional (on x) model is fitted.

fixed.x = TRUE: the means, variances and covariances of the exogenous x covariates are fixed to their sample values (for the ML, MML and IV estimator groups, unless start = "simple").

zero.keep.margins = TRUE: for categorical data, the (one-way) margins are preserved when adding a small constant to empty cells.

In addition, the various Mplus-specific technical options (information.expected.mplus, gamma.vcov.mplus, gamma.wls.mplus, gls.v11.mplus and rmsea.scaled.mplus) are all set to FALSE.

mimic = "Mplus"

Starting from the lavaan defaults, the following options are changed in an attempt to reproduce the output of Mplus:

information.expected.mplus = TRUE: use the Mplus variant of the expected information matrix. This affects the standard errors (when information = "expected") and the scaled (Satorra-Bentler and Yuan-Bentler) test statistics.

gamma.vcov.mplus = TRUE: use the Mplus way of computing the Γ matrix (the asymptotic covariance matrix of the sample statistics) that enters the robust (sandwich) standard errors.

gamma.wls.mplus = TRUE: use the Mplus way of computing the Γ matrix used as the weight matrix in (D)WLS estimation.

gls.v11.mplus = TRUE: use the Mplus variant of the weight matrix used in GLS estimation (when there is a meanstructure and conditional.x = FALSE).

rmsea.scaled.mplus = TRUE: use the Mplus reference distribution for the *scaled* RMSEA confidence interval and p-values. lavaan refers the *unscaled* test statistic to a chi-square distribution with $tr[U\Gamma]$ degrees of freedom, while Mplus refers the *scaled* statistic to a chi-square distribution with the degrees of freedom of the scaled test. Both match the mean of the null distribution, but on a different scale, so they yield different intervals and p-values. This only affects the Satorra-Bentler and Yuan-Bentler family (MLM, MLR, MLMVS, WLSM, ...); for the shifted family (MLMV, WLSMV, ULSMV) lavaan already uses the Mplus convention.

missing = "ml": for continuous data and the ML or MLR estimator, the default missing-data handling becomes full information maximum likelihood (FIML), rather than listwise deletion.

meanstructure = TRUE: a meanstructure is added by default (unless the estimator is PML).

group.equal: in a multiple-group analysis, if no equality constraints are requested, a default set is imposed: "loadings" and "thresholds" for categorical data; "loadings" for continuous data without a meanstructure; and "loadings" and "intercepts" for continuous data with a meanstructure.

baseline.conditional.x.free.slopes = FALSE: when conditional.x = TRUE, the baseline (independence) model does not free the slope structure.

The likelihood, conditional.x, fixed.x and zero.keep.margins defaults are the same as for mimic = "lavaan".

mimic = "EQS"

Starting from the lavaan defaults, the following options are changed in an attempt to reproduce the output of EQS (the value "LISREL" is treated as an alias for "EQS"):

likelihood = "wishart": for ML estimation, the Wishart likelihood is used. The sample covariance matrix is divided by N-1 (instead of N), which affects the chi-square test statistic, the loglikelihood and the standard errors.

baseline.fixed.x.free.cov = FALSE: when fixed.x = TRUE, the baseline (independence) model does not free the (co)variances of the exogenous covariates.

In addition, for the MLR estimator, the robust test statistic defaults to "yuan.bentler" (instead of the "yuan.bentler.mplus" variant that is used for mimic = "lavaan" and mimic = "Mplus").

Note that the mimic= argument only sets *defaults*: any option that is set explicitly in the lavaan call takes precedence. The mimicking is also necessarily incomplete; small numerical differences with the target program may remain.

See Also

lavOptions, lavaan

Examples

```
# default (mimic = "lavaan")
fit1 <- cfa(" visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 ",
            test = "satorra.bentler",
            missing = "listwise",
            data = HolzingerSwineford1939)
fit1

# mimic the Mplus program
fit2 <- cfa(" visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 ",
            test = "satorra.bentler",
            missing = "listwise",
            data = HolzingerSwineford1939, mimic = "Mplus")
fit2

# mimic the EQS program
fit3 <- cfa(" visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 ",
            test = "satorra.bentler",
            missing = "listwise",
            data = HolzingerSwineford1939, mimic = "EQS")
fit3
```

---

model.syntax *The Lavaan Model Syntax*

Description

The lavaan model syntax describes a latent variable model. The function lavParTable 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, marker_int_zero = FALSE,
    orthogonal = FALSE, orthogonal_y = FALSE,
    orthogonal_x = FALSE, orthogonal_efa = FALSE, std_lv = FALSE,
    correlation = FALSE, composites = TRUE, composites_cov_free = FALSE,
    effect_coding = "", conditional_x = FALSE,
    fixed_x = FALSE, parameterization = "delta", constraints = NULL,
    ceq_simple = FALSE, auto = FALSE, model_type = "sem",
    auto_fix_first = FALSE, marker = NULL,
    auto_fix_single = FALSE, auto_var = FALSE,
    auto_cov_lv_x = FALSE, auto_cov_x = FALSE, auto_cov_y = FALSE,
    auto_th = FALSE, auto_delta = FALSE, auto_efa = FALSE,
    var_table = NULL, ngroups = 1L, nthresholds = NULL,
    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, marker_int_zero = FALSE,
    orthogonal = FALSE, orthogonal_y = FALSE,
    orthogonal_x = FALSE, orthogonal_efa = FALSE, std_lv = FALSE,
    correlation = FALSE, composites = TRUE, composites_cov_free = FALSE,
    effect_coding = "", conditional_x = FALSE,
    fixed_x = FALSE, parameterization = "delta", constraints = NULL,
    ceq_simple = FALSE, auto = FALSE, model_type = "sem",
    auto_fix_first = FALSE, marker = NULL,
    auto_fix_single = FALSE, auto_var = FALSE,
    auto_cov_lv_x = FALSE, auto_cov_x = FALSE, auto_cov_y = FALSE,
    auto_th = FALSE, auto_delta = FALSE, auto_efa = FALSE,
    var_table = NULL, ngroups = 1L, nthresholds = NULL,
    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,
                    parser = "open", 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 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.

marker_int_zero
: Logical. Only relevant if the metric of each latent variable is set by fixing the first factor loading to unity. If TRUE, it implies meanstructure = TRUE and std_lv = FALSE, and it fixes the intercepts of the marker indicators to zero, while freeing the means/intercepts of the latent variables. This only works correctly for single-group, single-level models.

orthogonal
: If TRUE, all covariances among latent variables are set to zero.

orthogonal_y
: If TRUE, all covariances among endogenous latent variables only are set to zero.

orthogonal_x
: If TRUE, all covariances among exogenous latent variables only are set to zero.

orthogonal_efa
: If TRUE, all covariances among latent variables involved in rotation only are set to zero.

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. If there are multiple groups, std_lv = TRUE and "loadings" is included in the group.label argument, then only the latent variances of the first group will be fixed to 1.0, while the latent variances of the other groups are freely estimated.

correlation
: If TRUE, a correlation structure is fitted. For continuous data, this implies that the (residual) variances are no longer parameters of the model.

composites
: Logical. If TRUE, use the new (0.6-20) approach to handle composites.

composites_cov_free
: Logical. Only relevant if the model contains composites. If TRUE, the composite-indicator (co)variances (the elements of the T matrix) are estimated as free parameters. If FALSE (the default), they are fixed to their sample values.

effect_coding
: Can be logical or character string. If logical and TRUE, this implies effect_coding = c("loadings", "intercepts"). If logical and FALSE, it is set equal to the empty string. If "loadings" is included, equality constraints are used so that the average of the factor loadings (per latent variable) equals 1. Note that this should not be used together with std_lv = TRUE. If "intercepts" is included, equality constraints are used so that the sum of the intercepts (belonging to the indicators of a single latent variable) equals zero. As a result, the latent mean will be freely estimated and will usually equal the average of the means of the indicators involved.

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 reflect 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.

ceq_simple — If TRUE, and no other general constraints are used in the model, simple equality constraints are represented in the parameter table as duplicated free parameters (instead of extra rows with op = "==").

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.

marker — Optional named character vector mapping a latent variable (name) to the observed indicator (name) whose loading should be fixed to 1.0 (instead of the first indicator) when auto_fix_first = TRUE. Latent variables that are not named in this vector keep the first indicator as their marker. This is used internally by the bad.marker.crit mechanism.

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 of both observed and latent variables are set free.

auto_cov_lv_x — If TRUE, the covariances of exogenous latent variables are included in the model and set free.

auto_cov_x — If TRUE, the covariances between exogenous latent variables and observed exogenous covariates are also included in the model and set free (this implies auto_cov_lv_x = TRUE). Ignored (with a warning) if conditional_x = TRUE.

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.

auto_efa — If TRUE, the necessary constraints are imposed to make the (unrotated) exploratory factor analysis blocks identifiable: for each block, factor variances are set to 1, factor covariances are constrained to be zero, and factor loadings are constrained to follow an echelon pattern.

var_table — The variable table containing information about the observed variables in the model.

ngroups — The number of (independent) groups.

nthresholds — Either a single integer or a named vector of integers. If nthresholds is a single integer, all endogenous variables are assumed to be ordered with nthresholds indicating the number of thresholds needed in the model. If nthresholds is a named vector, it indicates the number of thresholds for these ordered variables only. This argument should not be used in combination with var_table.

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. When (in the model syntax) a vector of labels is used as a modifier for a certain parameter, this will override the group_equal setting if it applies to this parameter. See also the Multiple groups section below for using modifiers in 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.
as_data_frame	If TRUE, return the list of model parameters as a data.frame.
parser	Character. If "old", use the original/classic parser. If "new", use the new/ldw parser. If "open", use the newest, extensible parser. The default (as of version 0.7-1) is "open".
warn	If TRUE, some (possibly harmless) warnings are printed out.
debug	If TRUE, debugging information is printed out.
...	To accept old argument names with dots. No other arguments are accepted.

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. composites
    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. In 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, and the only term on the right is 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 side, separated by the `"+"` operator) are named `"t1"`, `"t2"`, and so on.

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. Composites: The `"<~"` operator can be used to define a composite (on the right hand side of the operator). A composite is a weighted linear combination of its composite indicators. The

name of the composite variable is on the left of the "<~" operator, while the terms on the right, separated by "+" operators, are the indicators of the composite variable.

There are 4 additional operators, also with left- and right-hand sides, that can be included in model syntax. Three of them are used to specify (in)equality constraints on estimated parameters (==, >, and <), and those are demonstrated in a later section about **(In)equality constraints**. The final additional operator (:=) can be used to define "new" parameters that are functions of one or more other estimated parameters. The := operator is demonstrated in a section about **User-defined parameters**.

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 is 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 constant. 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.

Additionally, the == operator can be used to set a *labeled* parameter equal to a specific numeric value. This will be demonstrated in the section below about **(In)equality constraints**.

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 their 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 "~").

There are two ways to constrain a parameter to be equal to another target parameter. 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 element should be replaced by a vector, having the same length as the number of groups. If you provide a single element, it will be recycled for all the groups. This may be dangerous, in particular when the modifier is a label. In that case,

the (same) label is copied across all groups, and this would imply an equality constraint across groups. Therefore, when using modifiers in a multiple group setting, it is always safer (and cleaner) to specify the same number of elements as the number of groups. Consider this example with 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 both 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
```

Random slopes (two-level models)

In a two-level model (with a `cluster=` argument), the `rv()` modifier can be used in the level-1 (within) part of the model to turn a regression coefficient into a *random slope*: a latent variable at the second (between) level. The (quoted) argument of `rv()` is the name of this latent variable. At the between level, the random slope can then be given a (residual) variance, covariances, regressions on between-level covariates, and an intercept (which is the 'fixed effect' of the slope). For example (cfr. ex9.8 in the Mplus User's Guide):

```
model <- '
  level: 1
    fw =~ y1 + y2 + y3 + y4
    fw ~ rv("s1")*x1 + rv("s2")*x2
  level: 2
    fb =~ y1 + y2 + y3 + y4
    s1 + s2 + fb ~ w
'
```

Here, the within-level regression coefficients of `fw` on the observed covariates `x1` and `x2` vary across clusters; the random slopes `s1` and `s2` are regressed on the between-level covariate `w`.

The covariate that carries a random slope may also be a *latent* (within-level) variable, as in

```
model <- '
  level: 1
    fxw =~ x1 + x2 + x3
    fyw =~ y1 + y2 + y3
    fyw ~ rv("s1")*fxw
```

```
  level: 2
    fxb =~ x1 + x2 + x3
    fyb =~ y1 + y2 + y3
    fyb ~ fxb
    fyb ~~ s1
'
```

Such a slope multiplies two random variables, and the marginal loglikelihood is no longer available in closed form: it is computed by (Gauss-Hermite) quadrature over the latent-covariate slopes only, while all other random effects are still integrated out analytically (Rockwood, 2020). The number of quadrature points (per dimension) can be set via the `integration.ngh` option (default: 21).

The same route is taken when the covariate is a 'split' both-level observed variable (an observed variable that appears in both the level-1 and the level-2 part of the model): the random slope then multiplies its *latent* within-cluster component (latent centering), and the covariate is jointly modeled:

```
model <- '
  level: 1
    y ~ rv("s1")*x
  level: 2
    y ~ x
    y ~~ s1
'
```

This is the latent-covariate contextual model of Rockwood (2020, section 4). Note the contrast with a within-only covariate (a variable that does *not* appear in the level-2 part): there, the slope multiplies the observed values, which are conditioned upon.

The current implementation is limited to: two-level models, a single group, continuous data, `estimator = "ML"` and `fixed.x = TRUE`; the observed covariates that carry a random slope must be within-only (level-1) variables. Missing data can be handled by full-information maximum likelihood (`missing = "ml"`); the exogenous covariates themselves must be complete (cases with missing values on exogenous variables are removed). The (observed-data) loglikelihood is *conditional* on *all* exogenous covariates: the covariates carrying random slopes, any other within-only exogenous covariates (as in Mplus `TYPE = TWOLEVEL RANDOM`), and the between-level exogenous covariates. Between-only *endogenous* observed variables (level-2 outcomes, e.g., indicators of a between-level factor) are supported; they must be pure indicators (not involved in regressions), with a free (or fixed to a nonzero value) residual variance, and without missing values. No chi-square test statistic (or fit indices) are available for models with random slopes; `fitMeasures()` provides the loglikelihood-based measures only (`npar`, `ntotal`, `logl`, `aic`, `bic`, `bic2`), plus (with `estimator = "MLR"`) the H0 scaling correction factor (`scaling.factor.h0`), which can be used for scaled (loglikelihood) difference tests. Standard errors can be `"standard"` (observed information) or `"robust.huber.white"` (sandwich; this is also what `estimator = "MLR"` implies). Both `optim.method = "nlminb"` and `optim.method = "em"` are supported; with complete data, `"nlminb"` is the default, while with `missing = "ml"`, the EM algorithm is the default optimizer. `lavPredict()` provides empirical Bayes (posterior mean) predictions: with `level = 2`, the between-level latent variables including the random slopes (and, with `se = "standard"`, their posterior standard deviations); with `level = 1` (the default), the within-level factor scores. For latent-covariate models, these are EAP (expected a posteriori) estimates: the posterior is a mixture over the quadrature nodes.

For random slopes with latent covariates, some additional restrictions apply (for now): a random-slope label may not be attached to both an observed and a latent covariate, and the optimizer is always `"nlminb"`. The latent-covariate slopes may be regressed on between-level covariates (e.g., `s1 ~ w` in the level-2 part), as in the PISA example of Rockwood (2020).

(In)equality constraints

The `==` operator can be used either to fix a parameter to a specific value, or to set an estimated parameter equal to another parameter. Adapting the example in the **Parameter labels and equality constraints** section, we could have used different labels for the second factor's intercepts:

```
y2a ~ int1*1
y2b ~ int2*1
y2c ~ int3*1
```

Then, we could fix the first intercept to zero by including in the syntax an operation that indicates the parameter's label equals that value:

```
int1 == 0
```

Whereas we could still estimate the other two intercepts under an equality constraint by setting their different labels equal to each other:

```
int2 == int3
```

Optimization can be less efficient when constraining parameters this way (see the documentation linked under **See also** for more information). But the flexibility might be advantageous. For example, the constraints could be specified in a separate character-string object, which can be passed to the `lavaan(..., constraints=)` argument, enabling users to compare results with(out) the constraints.

Inequality constraints work in much the same way, using the `<` or `>` operator to indicate which estimated parameter is hypothesized to be greater or less than either a specific value or another estimated parameter. For example, a variance can be constrained to be nonnegative:

```
y1a ~~ var1a*y1a
## hypothesized constraint:
var1a > 0
```

Or the factor loading of a particular indicator might be expected to exceed other indicators' loadings:

```
f1 =~ L1*y1a + L2*y1b + L3*y1c
## hypothesized constraints:
L1 > L2
L3 < L1
```

User-defined parameters

Functions of parameters can be useful to test particular hypotheses. Following from the `Multiple groups` example, we might be interested in which group's factor loading is larger (i.e., an estimate of differential item functioning (DIF) when the latent scales are linked by anchor items with equal loadings).

```
speed =~ c(L7, L7)*x7 + c(L8, L8)*x8 + c(L9.group1, L9.group2)*x9 '
## user-defined parameter:
DIF_L9 := L9.group1 - L9.group2
```

Note that this hypothesis is easily tested without a user-defined parameter by using the `lavTestWald()` function. However, a user-defined parameter additionally provides an estimate of the parameter being tested.

User-defined parameters are particularly useful for specifying indirect effects in models of mediation. For example:

```
model <- ' # direct effect
             Y ~ c*X
           # mediator
             M ~ a*X
             Y ~ b*M

# user defined parameters:

    # indirect effect (a*b)
             ab := a*b
    # total effect (defined using another user-defined parameter)
             total := ab + c
'
```

References

Rockwood, N. J. (2020). Maximum likelihood estimation of multilevel structural equation models with random slopes for latent covariates. *Psychometrika, 85*(2), 275–300. doi:10.1007/s11336020-097029

Rosseel, Y. (2012). `lavaan`: An R package for structural equation modeling. *Journal of Statistical Software, 48*(2), 1–36. doi:10.18637/jss.v048.i02

`modificationIndices` *Modification Indices*

Description

Given a fitted lavaan object, compute the modification indices (= univariate score tests) for a selected set of fixed-to-zero parameters.

Usage

```
modificationIndices(object, standardized = TRUE, cov_std = TRUE,
                    information = "expected",
                    power = FALSE, delta = 0.1, alpha = 0.05,
                    high_power = 0.75, sort = FALSE, minimum_value = 0,
                    maximum_number = nrow(list_1), free_remove = TRUE,
                    na_remove = TRUE, op = NULL, ...)
modindices(object, standardized = TRUE, cov_std = TRUE, information = "expected",
                   power = FALSE, delta = 0.1, alpha = 0.05, high_power = 0.75,
                    sort = FALSE, minimum_value = 0,
                    maximum_number = nrow(list_1), 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 EPCs. In the first column (sepc.lv), standardization is based on the variances of the (continuous) latent variables. In the second column (sepc.all), standardization is based on the variances of both the (continuous) observed and latent variables. (Residual) covariances are standardized using (residual) variances.
cov_std	Logical. See standardizedSolution.
information	character indicating the type of information matrix to use (check lavInspect for available options). "expected" information is the default, which provides better control of Type I errors.
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 by the modification index values. Higher values appear first.
minimum_value	Numeric. Filter output and only show rows with a modification index value equal to 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 selecting only those rows with operator op.
...	To support old argument names.

Details

Modification indices are just 1-df (or univariate) score tests. The modification index (or score test) for a single parameter reflects (approximately) the improvement in model fit (in terms of the chi-square test statistic) if we were to refit the model with this parameter set free. This function is a convenience function in the sense that it produces a (hopefully sensible) table of currently fixed-to-zero (or fixed to another constant) parameters. For each of these parameters, a modification index is computed, together with an expected parameter change (epc) value. It is important to realize that this function will only consider fixed-to-zero parameters. If you have equality constraints in the model, and you wish to examine what happens if you release all (or some) of these equality constraints, use the lavTestScore function.

Value

A data.frame containing modification indices and EPCs.

Examples

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

fit <- cfa(HS.model, data=HolzingerSwineford1939)
modindices(fit, minimum_value = 10, sort = TRUE)
```

---

open_parser *The open parser configuration*

Description

The lavaan model syntax parser can be extended with new operators, modifiers and groupings. These extensions are only used in the 'open' parser.

Usage

```
lav_parse_options(
    pkgname   = NULL,
    operators = NULL,
    modifiers = NULL,
    groupings  = NULL
  )
```

Arguments

pkgname	The name of the package that extends the elements.
operators	A character vector with new operators. These operators must start and end with a percentage sign and be at least 3 characters long.
modifiers	A data.frame with columns mod, side and expect. The elements in mod must be character strings in snake_case. The elements in side must be l or r, indicating whether the modifier is expected on the left-hand side or on the right-hand side of the operator. The elements in expect must be char, num or raw, specifying the type of value the modifier expects, where raw means the value is not evaluated by the parser but returned as an uninterpreted string.
groupings	A character vector with new groupings; they must be in snake_case.

Value

A list with data.frames containing the currently active parser configuration.

Examples

```
curconfig <- lav_parse_options("testPKG",
    operators = c('%w%', '%q%'),
    modifiers = data.frame(mod = c("linksc", "linksn", "rechtsn"),
                           side = c("l", "l", "r"),
                           expect = c("char", "num", "raw")),
    groupings = c("color", "colour")
)
for (j in seq_along(curconfig)) {
    cat(names(curconfig)[j], ":\n")
    print(curconfig[[j]])
}
model <- '
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + start(0.3)*b*y3 + upper(0.99)*c*y4
     dem65 =~ y5 + a*y6 + b*y7 + rechtsn(2 * pi / 3)*c*y8
     dem60 %w% aa *0.2*y4

    dem60 ~ ind60
    dem65 ~ ind60 + dem60
    linksc("klopt") * dem60 =~ aaa * y8
    linksn(sqrt(23.5)) * dem60 =~ bbb * y9
    efa("efatje") * dem65 =~ xyz * x3

    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
'
print(result <- lavParseModelString(model, TRUE, parser = "open"))
print(attributes(result))
```

```
model2 <- '
  color: green
    level: 1
        fw =~ y1 + y2 + y3
        fw ~ x1 + x2 + x3
    level: 2
        fb =~ y1 + y2 + y3
        fb ~ w1 + w2
  color: blue
    level: 1
        fw =~ y1 + y2 + y3
        fw ~ x1 + x2 + x3
    level: 2
        fb =~ y1 + y2 + y3
        fb ~ w1 + w2
'
print(result2 <- lavParseModelString(model2, TRUE, parser = "open"))
print(attributes(result2))
```

---

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 lavParTable function coerced to a data.frame (with stringsAsFactors = FALSE).

See Also

lavParTable.

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 originally retrieved from `http://web.missouri.edu/~kolenikovs/Stat9370/democindus.txt` (link no longer valid; see discussion on SEMNET 18 Jun 2009). The dataset is part of a larger (public) dataset (ICPSR 2532), see `https://www.icpsr.umich.edu/web/ICPSR/studies/2532`.

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)
```

---

sam *Fit Structural Equation Models using the SAM approach*

Description

Fit a Structural Equation Model (SEM) using the Structural After Measurement (SAM) approach.

Usage

```
sam(model = NULL, data = NULL, aux = NULL, cmd = "sem", se = "twostep",
    mm_list = NULL, mm_args = list(bounds = "wide.zerovar"),
    struc_args = list(estimator = "ML"),
    sam_method = "local", ...,
    local_options = list(M.method = "ML", lambda.correction = TRUE,
                         alpha.correction = 0L, twolevel.method = "h1"),
    global_options = list(),
    bootstrap = list(R = 1000L, type = "ordinary", show.progress = FALSE),
    output = "lavaan",
    bootstrap_args = bootstrap
)
```

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 (e.g., the output of the `lavParTable()` function) is also accepted.
`data`	A data frame containing the observed variables used in the model.
`aux`	Character vector. Names of auxiliary observed variables, forwarded to the underlying measurement and structural model fits. See the `lavaan` function for more details.
`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.

se — Character. The type of standard errors that are used in the final (structural) model. If "twostep" (the default), the standard errors take the estimation uncertainty of the first (measurement) stage into account. If "standard", this uncertainty is ignored, and we treat the measurement information as known. If "none", no standard errors are computed. The experimental "twostep.huber.white" variant computes the two-step correction as a stacked (Murphy-Topel type) sandwich with casewise-score based (first-order) meat – the two-step analogue of se = "robust.huber.white" in sem; it is currently available for continuous complete or missing = "ml" data (single- or multigroup, single level, no equality constraints), and falls back to "twostep.robust" or "twostep" otherwise. If the data are clustered (a cluster= argument is provided, without multilevel model syntax), se = "twostep" is automatically replaced by se = "twostep.robust", and the standard errors (also for se = "local" and se = "bootstrap", which resamples whole clusters) account for the clustering. For two-level models, se = "local" is available for the single-group continuous complete-data setting; it uses the cluster-sandwich covariance of the saturated (h1) estimates, and also enables the corrected structural test statistic. For estimator = "PML", se = "local" is the only available two-step correction (single group, complete data, all-ordinal indicators): it combines the casewise influences of the saturated thresholds/polychoric correlations with the casewise pairwise-likelihood influences of the measurement-block estimates, and also enables the corrected structural test statistic.

mm_list — List. Define the measurement blocks. Each element of the list should be either a single name of a latent variable, or a vector of latent variable names. If omitted, a separate measurement block is used for each latent variable.

mm_args — List. Optional arguments for the fitting function(s) of the measurement block(s) only. See lavOptions for a complete list.

struc_args — List. Optional arguments for the fitting function of the structural part only. See lavOptions for a complete list.

sam_method — Character. Can be set to "local", "global" or "fsr". In the latter case, the results are the same as if Bartlett factor scores were used, without any bias correction.

... — Many more options can be specified, using 'name = value'. See lavOptions for a complete list. These options affect both the measurement blocks and the structural part.

local_options — List. Options specific for local SAM method (these options may change over time). If lambda.correction = TRUE, we ensure that the variance matrix of the latent variables (VETA) is positive definite. The alpha.correction option must be an integer. Acceptable values are in the range 0 to N-1. If zero (the default), no small sample correction is performed, and the bias-correction is the same as with local SAM. When equal to N-1, the bias-correction is eliminated, and the results are the same as naive FSR. Typical values are 0, P+1 (where P is the number of predictors in the structural model), P+5, and (N-1)/2.

global_options — List. Options specific for global SAM method (not used for now).

bootstrap — List. Only used when se = "bootstrap". Typical elements of this list are R: the number of bootstrap samples, and type, which can be set to "ordinary" (the

default) or "parametric". A single number is also accepted (as in sem()) and is interpreted as R, the number of bootstrap draws.

output — Character. If "lavaan", a lavaan object is returned. If "list", a list is returned with all the ingredients from the different stages.

bootstrap_args — List. Deprecated, use bootstrap instead.

Details

The sam function automates the SAM approach by first estimating the measurement part of the model and then the structural part of the model. See the reference for more details.

Note that in the current implementation, all indicators of latent variables must be observed. As a result, second-order factor structures are not yet supported.

If sam.method is "local", "fsr" or "cfsr", the structural part of the model is fitted to the estimated latent (co)variances (and means) from step 1. This step-2 structural model is stored in the @internal slot of the returned object, and can be retrieved by lavInspect(fit, "sam.struc.fit.object"). The functions fitMeasures (and summary(fit, fit.measures = TRUE)), lavResiduals (and summary(fit, residuals = TRUE)), modindices and lavTestLRT (when all compared models are local SAM models) operate on this structural model: the reported quantities describe the fit of the structural part only, conditional on the (fixed) measurement model of step 1. When available, the reported (scaled) test statistic and the robust fit measures are corrected for the estimation of the measurement model in step 1.

If sam.method is "global", the returned object is an ordinary (joint) lavaan object, and fitMeasures returns the usual fit measures of the joint model. Note, however, that the parameters were estimated using a two-step procedure; the behavior of fit measures in this setting has not been studied, and they should be interpreted with caution.

Value

If output = "lavaan", an object of class lavaan, for which several methods are available, including a summary method. If output = "list", a list.

Exogenous covariates and conditional.x

Observed exogenous covariates may enter the structural part of the model. With continuous data, they are simply treated as (pseudo) latent variables (the default conditional.x = FALSE), but conditional.x = TRUE is supported as well (single-group, single-level models): the structural part is then fitted to the latent intercepts, the latent-on-covariate slopes, and the residual latent (co)variances, all conditional on the covariates.

With categorical (ordered) data, conditional.x = TRUE is the lavaan default (when exogenous covariates are present), and this is also the recommended setting: the covariates are conditioned on (as in a probit regression), rather than being absorbed into the joint polychoric correlation matrix – which would (wrongly) treat them as jointly normal, a real concern for binary/dummy covariates. In this setting, the measurement blocks of step 1 are estimated with conditional.x = TRUE as well (each block latent variable is regressed on all exogenous covariates, MIMIC style), so that the measurement parameters live on the same conditional latent-response scale (Var(y*|x) = 1, delta parameterization) as the joint model. An *ordered endogenous* variable in the structural part is supported too (with or without covariates): it enters the structural model on its latent-response scale, while its thresholds are fixed at their sample-based values.

When `conditional.x = TRUE` is combined with a local `sam.method`, the default `se = "twostep"` is automatically replaced by `se = "twostep.robust"`: the classic two-step formula is built on the joint information matrix, which does not describe the local step-2 estimator in this setting (it underestimates the sampling variability, especially for binary covariates). The robust variant is computed as a structural-space sandwich here, and coincides with `se = "local"`.

Not (yet) available in combination with `conditional.x = TRUE`: multiple groups (for the local standard errors and the corrected test statistic), multilevel models, latent interactions, and the Yuan-Chan scaled global test statistic of `sam_method = "global"` (the standard, unscaled test is reported instead, with a warning).

References

Rosseel and Loh (2021). A structural-after-measurement approach to Structural Equation Modeling. Psychological Methods. Advance online publication. https://dx.doi.org/10.1037/met0000503

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.sam <- sam(model, data = PoliticalDemocracy,
               mm.list = list(ind = "ind60", dem = c("dem60", "dem65")))
summary(fit.sam)
```

sem *Fit Structural Equation Models*

Description

Fit a Structural Equation Model (SEM).

Usage

```
sem(model = NULL, data = NULL, ordered = NULL, aux = NULL, sampling_weights = NULL,
    sample_cov = NULL, sample_mean = NULL, sample_th = NULL,
    sample_nobs = NULL, group = NULL, cluster = NULL,
    constraints = "", wls_v = NULL, nacov = NULL, ov_order = "model",
    ...)
```

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 (e.g., the output of the lavParTable() 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.
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 data.frame.) Since 0.6-4, ordered can also be logical. If TRUE, all observed endogenous variables are treated as ordered (ordinal). If FALSE, all observed endogenous variables are considered to be numeric (again, unless they are declared as ordered in the data.frame.)
aux	Character vector. Names of auxiliary observed variables, used to make the missing-at-random (MAR) assumption more plausible under missing data (continuous data only). With missing = "ml" they are added to the model as saturated correlates (Graham, 2003), affecting the estimates and standard errors but hidden from the default output; with missing = "two.stage"/"robust.two.stage" they enter the first-stage EM moments. See the lavaan function for more details.
sampling_weights	A variable name in the data frame containing sampling weight information. Currently only available for non-clustered data. Depending on the sampling.weights.normalization option, these weights may be rescaled (or not) so that their sum equals the number of observations (total or per group).
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.

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

sample_th — Vector of sample-based thresholds. For a multiple group analysis, a list with a vector of thresholds for each group.

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

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

cluster — Character. A (single) variable name in the data frame defining the clusters in a two-level dataset.

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

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.

ov_order — Character. If "model" (the default), the order of the observed variable names (as reflected for example in the output of lav_object_vnames()) is determined by the model syntax. If "data", the order is determined by the data (either the full data.frame or the sample (co)variance matrix). If the wls_v and/or nacov matrices are provided, this argument is currently set to "data".

... — Many more options can be specified, using 'name = value'. See lavOptions for a complete list.

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.efa = 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. doi:10.18637/jss.v048.i02

See Also

lavaan, estimator_iv

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)
```

---

standardizedSolution *Standardized Solution*

Description

Standardized solution of a latent variable model.

Usage

```
standardizedSolution(object, type = "std.all", se = TRUE, zstat = TRUE,
                     pvalue = TRUE, ci = TRUE, level = 0.95,
                     boot_ci_type = "perc", cov_std = TRUE,
                     remove_eq = TRUE, remove_ineq = TRUE, remove_def = FALSE,
                     remove_aux = TRUE,
                     partable = NULL, glist = NULL, est = NULL,
                     output = "data.frame", ...)
```

Arguments

object	An object of class lavaan.
type	If "std.lv", the standardized estimates are based on the variances of the (continuous) latent variables only. If "std.all", the standardized estimates are based on the variances of both (continuous) observed and latent variables. If "std.nox", the standardized estimates are based on the variances of both (continuous) observed and latent variables, but not the variances of exogenous covariates. Note that "std.nox" only differs from "std.all" if fixed.x = TRUE; if fixed.x = FALSE, the exogenous covariates are treated as random variables (and standardized) just like any other variable, and a warning is issued. Alternatively, type may be a vector of (observed) variable names (for example type = c("x1", "x2")); in that case only the parameters involving these variables are standardized (the other observed variables are left unstandardized). This is a generalization of "std.nox", where the (observed) exogenous x variables are the ones left unstandardized.
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.
ci	If TRUE, confidence intervals are added to the output.
level	The confidence level required.
boot_ci_type	Character. Only used if the model was fitted with se = "bootstrap". In that case, bootstrap standard errors and bootstrap confidence intervals are computed for the standardized parameters (by re-standardizing each bootstrap draw), and this argument selects the type of bootstrap confidence interval, exactly as in parameterEstimates: "norm", "basic", "perc" (the default), "bca.simple" or "bca".
cov_std	Logical. If TRUE, the (residual) observed covariances are scaled by the square root of the 'Theta' diagonal elements, and the (residual) latent covariances are scaled by the square root of the 'Psi' diagonal elements. If FALSE, the (residual) observed covariances are scaled by the square root of the diagonal elements of the observed model-implied covariance matrix (Sigma), and the (residual) latent covariances are scaled by the square root of diagonal elements of the model-implied covariance matrix of the latent variables.
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 output by removing all rows containing parameter definitions, if any.
remove_aux	Logical. If TRUE (the default), filter the output by removing all rows corresponding to auxiliary (aux=) variables added to the model as saturated correlates (only relevant when missing = "ml" and the aux= argument was used).

glist	List of model matrices. If provided, they will be used instead of the GLIST inside the object@Model slot. Only works if the est argument is also provided. See Note.
est	Numeric. Parameter values (as in the 'est' column of a parameter table). If provided, they will be used instead of the parameters that can be extracted from object. Only works if the glist argument is also provided. See Note.
partable	A custom list or data.frame in which to store the standardized parameter values. If provided, it will be used instead of the parameter table inside the object@ParTable slot.
output	Character. If "data.frame", the parameter table is displayed as a standard (albeit lavaan-formatted) data.frame. If "text" (or alias "pretty"), the parameter table is prettified and displayed with subsections (as used by the summary function).
...	To support old argument names.

Details

The standardized estimates are functions of the (unstandardized) free parameters and of (a subset of) the model-implied (co)variances used for scaling. The standard errors reported by standardizedSolution are therefore standard errors *for the standardized parameters*, and they will in general differ from the standard errors of the unstandardized parameters reported by parameterEstimates (or in the summary() output). They are also not the same as the standard errors that would be obtained by simply rescaling the unstandardized standard errors.

How the standard errors are computed depends on how the model was originally fitted (in particular on the se= argument of lavaan, cfa, sem, ...):

- By default (se = "standard", "robust.sem", "robust.huber.white", ...), the standard errors are obtained with the *delta method*: the Jacobian of the standardization function is computed (numerically) and combined with the variance-covariance matrix of the (unstandardized) parameter estimates. Any robustness present in that variance-covariance matrix (for example robust or sandwich-type standard errors) is automatically propagated to the standardized solution.
- If the model was fitted with se = "bootstrap" (and the bootstrap draws are available), bootstrap standard errors and bootstrap confidence intervals are reported instead. These are obtained by re-standardizing each bootstrap draw and computing the standard deviation (for the standard error) and the requested interval type (see boot_ci_type) of the resulting standardized values. Note that, as in parameterEstimates, the p-value is still computed by referring the z-statistic (standardized estimate divided by its bootstrap standard error) to a standard normal distribution.

There is no separate argument to choose the *type* of standard error within standardizedSolution: the type always follows the se= setting that was used when the model was fitted. To obtain, say, robust or bootstrap standard errors for the standardized solution, refit the model with the corresponding se= argument.

Value

A data.frame containing standardized model parameters.

If the model was fitted with `se = "bootstrap"` (and the bootstrap draws are available), the standardized estimates in the bootstrap samples are stored as the `"boot_std"` attribute of the returned data.frame (a matrix with one row per bootstrap draw and one column per row of the returned solution). These can be retrieved with `attr(x, "boot_std")`, analogous to `lavInspect(object, "coef.boot")` for the unstandardized solution.

In addition, if `boot_ci_type = "bca"`, the empirical-influence design matrix used to compute the acceleration constant is stored as the `"design"` attribute (retrieve with `attr(x, "design")`).

Note

The `est`, `glist`, and `partable` arguments are not meant for everyday users, but for authors of external R packages that depend on `lavaan`. Only to be used with great caution.

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.
`...`	To support old argument names.

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