Package: psychonetrics (via r-universe)

October 15, 2026

Type Package

Title Structural Equation Modeling and Confirmatory Network Analysis

Version 0.17.8

Maintainer Sacha Epskamp <mail@sachaepskamp.com>

Description Multi-group (dynamical) structural equation models in combination with confirmatory network models from cross-sectional, time-series and panel data <doi:10.31234/osf.io/8ha93>. Allows for confirmatory testing and fit as well as exploratory model search.

License GPL-2

Copyright see file COPYRIGHTS. Bundles the LBFGS++ (LBFGSpp) headers by Yixuan Qiu and others (MIT license) in inst/include.

LinkingTo Rcpp (>= 0.11.3), RcppArmadillo, RcppEigen, pbv, roptim

Depends R (>= 4.3.0)

Imports methods, qgraph, numDeriv, dplyr, Matrix (>= 1.6-5), lavaan, corpcor, glasso, mgcv, optimx, nloptr, pbapply, parallel, magrittr, IsingSampler (>= 0.5), tidyr, psych, rlang, MASS

Suggests psychTools, semPlot, graphicalVAR, metaSEM, mvtnorm, ggplot2, tinytest

ByteCompile true

URL https://psychonetrics.org/

BugReports https://github.com/SachaEpskamp/psychonetrics/issues

StagedInstall true

NeedsCompilation yes

RoxygenNote 7.3.3

Author Sacha Epskamp [aut, cre]

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

Date/Publication 2026-08-04 09:20:02 UTC

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

RemoteRef HEAD

RemoteSha 1baf83d2287abac9e243260a29739330eddbef51

Contents

psychonetrics-package . . . 3
aggregate_bootstraps . . . 6
allequal . . . 7
bifactor . . . 8
BlumeCapel . . . 9
bootstrap . . . 13
changedata . . . 14
CIplot . . . 15
compare . . . 17
covML . . . 20
diagnostics . . . 21
dlvm1 . . . 22
duplicationMatrix . . . 32
emergencystart . . . 33
equalityScoreTest . . . 33
esa . . . 35
factorscores . . . 37
find_penalized_lambda . . . 37
fit . . . 40
fixpar . . . 41
fixstart . . . 42
fromlavaan . . . 43
generate . . . 45
getmatrix . . . 46
getVCOV . . . 48
groupequal . . . 49
Ising . . . 50
Jonas . . . 54
latentgrowth . . . 55
logbook . . . 57
loop_psychonetrics . . . 57
lvm . . . 59
meta_lvm . . . 73
meta_var1 . . . 78
meta_varcov . . . 82
MIs . . . 87
ml_lvm . . . 89
ml_tsdlvm1 . . . 97
ml_var1 . . . 98
ml_varcov . . . 104
modelsearch . . . 108
NA2020 . . . 109
panelvar . . . 111
parameters . . . 119
parequal . . . 120
partialprune . . . 121

penalize . . . 123
plot.psychonetrics . . . 126
prune . . . 129
psychonetrics-class . . . 130
psychonetrics_bootstrap-class . . . 132
psychonetrics_log-class . . . 133
psychonetrics_update . . . 133
ri_clpm . . . 134
runmodel . . . 140
setestimator . . . 143
setverbose . . . 146
simplestructure . . . 147
StarWars . . . 147
stepup . . . 148
tolavaan . . . 150
transmod . . . 152
tsdlvm1 . . . 154
unionmodel . . . 160
update_baseline_saturated . . . 161
var1 . . . 162
varcov . . . 168
write_psychonetrics . . . 175

Index **177**

`psychonetrics-package` *Structural Equation Modeling and Confirmatory Network Analysis*

Description

Multi-group (dynamical) structural equation models in combination with confirmatory network models from cross-sectional, time-series and panel data <doi:10.31234/osf.io/8ha93>. Allows for confirmatory testing and fit as well as exploratory model search.

Details

The DESCRIPTION file:

Package:	psychonetrics
Type:	Package
Title:	Structural Equation Modeling and Confirmatory Network Analysis
Version:	0.17.8
Authors@R:	person(given = "Sacha",family = "Epskamp", role = c("aut", "cre"), email = "mail@sachaepskamp.com'
Maintainer:	Sacha Epskamp <mail@sachaepskamp.com>
Description:	Multi-group (dynamical) structural equation models in combination with confirmatory network models
License:	GPL-2
Copyright:	see file COPYRIGHTS. Bundles the LBFGS++ (LBFGSpp) headers by Yixuan Qiu and others (MIT lic
LinkingTo:	Rcpp (>= 0.11.3), RcppArmadillo, RcppEigen, pbv, roptim

Depends:	R (>= 4.3.0)
Imports:	methods, qgraph, numDeriv, dplyr, Matrix (>= 1.6-5), lavaan, corpcor, glasso, mgcv, optimx, nloptr, pba
Suggests:	psychTools, semPlot, graphicalVAR, metaSEM, mvtnorm, ggplot2, tinytest
ByteCompile:	true
URL:	https://psychonetrics.org/
BugReports:	https://github.com/SachaEpskamp/psychonetrics/issues
StagedInstall:	true
NeedsCompilation:	yes
RoxygenNote:	7.3.3
Author:	Sacha Epskamp [aut, cre]

Index of help topics:

addMIs	Model updating functions
aggregate_bootstraps	Aggregate Bootstrapped Models
allequal	Constrain all elements of a matrix equal within groups
bifactor	Bi-factor models
BlumeCapel	Blume-Capel model
bootstrap	Bootstrap a psychonetrics model
changedata	Change the data of a psychonetrics object
checkJacobian	Diagnostic functions
CIplot	Plot Analytic Confidence Intervals
compare	Model comparison
covML	Maximum likelihood covariance estimate
dlvm1	Lag-1 dynamic latent variable model family of psychonetrics models for panel data
duplicationMatrix	Model matrices used in derivatives
emergencystart	Reset starting values to simple defaults
equalityScoreTest	Score (Lagrange Multiplier) test for cross-group equality constraints
esa	Ergodic Subspace Analysis
factorscores	Compute factor scores
find_penalized_lambda	Automated Lambda Selection for Penalized Estimation
fit	Print fit indices
fixpar	Parameters modification
fixstart	Attempt to Fix Starting Values
fromlavaan	Convert a lavaan model to a psychonetrics lvm
generate	Generate data from a fitted psychonetrics object
getmatrix	Extract an estimated matrix
getVCOV	Obtain the asymptotic covariance matrix
groupequal	Group equality constraints
Ising	Ising model
Jonas	Jonas dataset
latentgrowth	Latent growth curve model

logbook	Retrieve the psychonetrics logbook
loop_psychonetrics	Parallel loop for psychonetrics
lvm	Continuous latent variable family of psychonetrics models
meta_lvm	Meta-analytic latent variable model
meta_var1	Meta-analytic VAR(1) model
meta_varcov	Variance-covariance and GGM meta analysis
MIs	Print modification indices
ml_lvm	Multi-level latent variable model family
ml_tsdlvm1	Multi-level Lag-1 dynamic latent variable model family of psychonetrics models for time-series data
ml_var1	Multi-level lag-1 vector auto-regression family (random intercepts)
ml_varcov	Multi-level variance-covariance family of psychonetrics models
modelsearch	Stepwise model search
NA2020	Network Analysis 2020 Self-Report Data
panelvar	Lag-1 panel vector autoregression family of psychonetrics models
parameters	Print parameter estimates
parequal	Set equality constraints across parameters
partialprune	Partial pruning of multi-group models
penalize	Penalized Maximum Likelihood Functions
plot.psychonetrics	Plot the networks of a psychonetrics model
prune	Stepdown model search by pruning non-significant parameters
psychonetrics_bootstrap-class	Class '"psychonetrics_bootstrap"'
psychonetrics_log-class	Class '"psychonetrics_log"'
psychonetrics-class	Class '"psychonetrics"'
psychonetrics-package	Structural Equation Modeling and Confirmatory Network Analysis
ri_clpm	Random intercept cross-lagged panel models and networks
runmodel	Run a psychonetrics model
setestimator	Convenience functions
setverbose	Should messages of computation progress be printed?
simplestructure	Generate factor loadings matrix with simple structure
StarWars	Star Wars dataset
stepup	Stepup model search along modification indices
tolavaan	Convert a psychonetrics lvm to a lavaan model
transmod	Transform between model types
tsdlvm1	Lag-1 dynamic latent variable model family of psychonetrics models for time-series data

unionmodel	Unify models across groups
update_baseline	Set a custom baseline or saturated reference model
var1	Lag-1 vector autoregression family of psychonetrics models
varcov	Variance-covariance family of psychonetrics models
write_psychonetrics	Write comprehensive model output to a text file

This package can be used to perform Structural Equation Modeling and confirmatory network modeling. Current implemented families of models are (1) the variance–covariance matrix (varcov), (2) the latent variable model (lvm), (3) the lag-1 vector autoregression model (var1), (4) the dynamical lag-1 latent variable model for panel data (dlvm1) and for time-series data (tsdlvm1), (5) the Ising model (Ising) and its generalization the Blume-Capel model (BlumeCapel), (6) the multi-level latent variable model for two-level data (ml_lvm) and the multi-level dynamical model for multi-subject time-series data (ml_tsdlvm1), (7) the meta-analytic variance–covariance model (meta_varcov, with wrappers meta_lvm and meta_var1), and (8) the random-intercept cross-lagged panel model (ri_clpm, with the network-parameterized wrapper ri_clpn).

Author(s)

Sacha Epskamp [aut, cre]

Maintainer: Sacha Epskamp <mail@sachaepskamp.com>

References

More information: psychonetrics.org

`aggregate_bootstraps` *Aggregate Bootstrapped Models*

Description

Aggregates bootstrap results into a `psychonetrics_bootstrap` object

Usage

```
aggregate_bootstraps(sample, bootstraps, remove_problematic = TRUE)
```

Arguments

sample	The original `psychonetrics` object (not bootstrapped)
bootstraps	A list of bootstrapped `psychonetrics` objects (i.e., using `bootstrap = TRUE`)
remove_problematic	Remove bootstraps that did not converge (sum of absolute gradient > 1)

Details

After running this function, the helper functions `parameters`, `fit`, and `CIplot` can be used to investigate bootstrap results.

Value

An object of the class `psychonetrics_bootstrap`

Author(s)

Sacha Epskamp

`allequal` *Constrain all elements of a matrix equal within groups*

Description

The `allequal` function constrains all free elements of a model matrix to be equal to one another *within* each group (a single shared parameter per group). In contrast to `groupequal`, which constrains corresponding elements equal *across* groups, `allequal` shares one parameter across the elements of the matrix and does so separately for each group, so the constraint is not imposed across groups. This is convenient for, for example, constraining all thresholds (`tau`) or all Blume-Capel quadratic potentials (`delta`) to a common value, or fitting an equal-edge-weight network (`omega`).

Usage

```
allequal(x, matrix, row, col, group, verbose, log = TRUE,
         runmodel = FALSE, identify = TRUE, ...)
```

Arguments

`x`	A `psychonetrics` model.
`matrix`	String indicating the matrix whose elements should be constrained equal.
`row`	Optional integer or string vector indicating which rows of the matrix to include. Defaults to all rows.
`col`	Optional integer or string vector indicating which columns of the matrix to include. Defaults to all columns.
`group`	Optional vector of group ids to which the constraint is applied. Defaults to all groups (each constrained separately).
`verbose`	Logical, should messages be printed?
`log`	Logical, should the log be updated?
`runmodel`	Logical, should the model be updated?
`identify`	Logical, should the model be identified?
`...`	Arguments sent to `runmodel`

Details

Only free parameters are affected; elements that are fixed (e.g. fixed to zero) are left unchanged. For symmetric matrices all free elements, including a free diagonal, are constrained equal; if this is not desired (for example to keep the diagonal separate from the off-diagonal elements), use the `row` and `col` arguments to restrict the constraint to the relevant elements. Within each group the selected free elements are given the same parameter number and a common starting value (the mean of their current values).

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

`groupequal`, `parequal`, `fixpar`

Examples

```
## Not run:
library("dplyr")

# Simulate a small Blume-Capel data set (requires the development version of
# IsingSampler that supports the 'delta' argument):
library("IsingSampler")
nNode  <- 5
graph  <- 0.4 * (matrix(1, nNode, nNode) - diag(nNode)) *
            (matrix(c(0,1,0,0,1, 1,0,1,0,0, 0,1,0,1,0, 0,0,1,0,1, 1,0,0,1,0), 5, 5))
data <- IsingSampler(1000, graph = graph, thresholds = rep(0, nNode),
                     beta = 1, delta = 1, responses = c(-1, 0, 1))

# Blume-Capel model with all quadratic (delta) potentials constrained equal,
# and all thresholds (tau) constrained equal, within the group:
mod <- BlumeCapel(data, responses = c(-1, 0, 1)) %>%
  allequal("delta") %>%
  allequal("tau") %>%
  runmodel

## End(Not run)
```

`bifactor` *Bi-factor models*

Description

Wrapper to `lvm` to specify a bi-factor model.

Usage

```
bifactor(data, lambda, latents, bifactor = "g", ...)
```

Arguments

data	The data as used by lvm
lambda	The factor loadings matrix *without* the bifactor, as used by by lvm
latents	A vector of names of the latent variables, as used by lvm
bifactor	Name of the bifactor
...	Arguments sent to lvm

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

BlumeCapel *Blume-Capel model*

Description

The Blume-Capel model is the full *Spin* distribution model. In addition to the linear node potentials (tau) and pairwise interactions (omega) of the Ising model, it estimates a quadratic node potential delta (the Blume-Capel single-ion / crystal-field term). The Ising model is the special case obtained by fixing all delta parameters to zero, so the two models are nested. By convention the Blume-Capel model is defined for the ternary c(-1, 0, 1) response coding (a warning is raised for any other response set), but any number of ordered numeric response options is allowed (the same set for every variable, and not necessarily integers).

Usage

```
BlumeCapel(data, omega = "full", tau, delta, beta, beta_model =
                 c("beta", "log_beta"), vars, groups, covs, means,
                 nobs, covtype = c("choose", "ML", "UB"), responses,
                 missing = "listwise", equal = "none",
                 baseline_saturated = TRUE, estimator = "default",
                 optimizer, storedata = FALSE, WLS.W, sampleStats,
                 identify = TRUE, verbose = FALSE, maxNodes = 20,
                 maxStates = 2^maxNodes, min_sum = -Inf,
                 bootstrap = FALSE, boot_sub,
                 boot_resample, penalty_lambda = NA,
                penalty_alpha = 1, penalize_matrices)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied.
omega	The network structure. Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions nNode x nNode with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
tau	Optional vector encoding the threshold/intercept (linear node potential) structure. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.
delta	Optional vector encoding the quadratic node potential (Blume-Capel) structure. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free parameters, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector. When missing, every delta parameter is freely estimated (fixing all of them to zero would instead yield the Ising model).
beta	Optional scalar encoding the inverse temperature. 1 indicate free beta parameters, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such scalars.
beta_model	How should beta be modeled? Set beta_model = "log_beta" to model the log of beta rather than beta directly.
vars	An optional character vector encoding the variables used in the analysis. Must equal names of the dataset in data.
groups	An optional string indicating the name of the group variable in the data.
covs	A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure covtype argument is set correctly to the type of covariances used.
means	A vector of sample means, or a list/matrix containing such vectors for multiple groups.
nobs	The number of observations used in covs and means, or a vector of such numbers of observations for multiple groups.
covtype	If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to "ML" for maximum likelihood estimates (denominator n) and "UB" to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.
responses	A vector of the response options used, encoded identically across all variables. Defaults to (and is conventionally) c(-1, 0, 1); a warning is raised for any other set. Automatically detected from the data when not supplied; required when covs is used. Any number of distinct response options is allowed, and the values need not be integers.

`missing` How should missingness be handled when `data` is used. Only `"listwise"` (listwise deletion) is currently supported for the Blume-Capel model; other options (e.g. `"pairwise"`) are rejected at construction because they produce undefined sufficient statistics.

`equal` A character vector indicating which matrices should be constrained equal across groups.

`baseline_saturated` A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

`estimator` The estimator to be used. Only `"ML"` (maximum likelihood) estimation is currently supported for the Blume-Capel model.

`optimizer` The optimizer to be used. Can be one of `"nlminb"` (the default R `nlminb` function), `"ucminf"` (from the `optimr` package), `"nloptr_TNEWTON"` (preconditioned truncated Newton via `nloptr`), and `"LBFGS++"` (pure C++ L-BFGS-B). Defaults to `"nlminb"`.

`storedata` Logical, should the raw data be stored? Needed for bootstrapping (see `bootstrap`).

`WLS.W` Optional WLS weights matrix. CURRENTLY NOT USED.

`sampleStats` An optional sample statistics object. Mostly used internally.

`identify` Logical, should the model be identified?

`verbose` Logical, should messages be printed?

`maxNodes` The maximum number of nodes allowed in the analysis. Used to set the default of `maxStates` (`2^maxNodes`); for binary data this reproduces the historical node limit. It is not recommended to set this higher.

`maxStates` The maximum number of response patterns the exact ML estimator may enumerate. Exact ML estimation sums over every possible response pattern when computing the partition function, expected values and (expected) Hessian, so the cost grows as `length(responses)^nNode`. The function stops with an error when `length(responses)^nNode` exceeds `maxStates`. The default `2^maxNodes` reproduces the historical binary node limit while accounting for more than two response options. Raise this only if the computation is feasible (note that the number of states—and hence the run time—grows very quickly with both the number of nodes and the number of response options).

`min_sum` The minimum sum score that is artificially possible in the dataset. Defaults to -Inf. Set this only if you know a lower sum score is not possible in the data, for example due to selection bias.

`bootstrap` Should the data be bootstrapped? If `TRUE` the data are resampled and a bootstrap sample is created. These must be aggregated using `aggregate_bootstraps`! Can be `TRUE` or `FALSE`. Can also be `"nonparametric"` (which sets `boot_sub = 1` and `boot_resample = TRUE`) or `"case"` (which sets `boot_sub = 0.75` and `boot_resample = FALSE`).

`boot_sub` Proportion of cases to be subsampled (`round(boot_sub * N)`).

`boot_resample` Logical, should the bootstrap be with replacement (`TRUE`) or without replacement (`FALSE`)

`penalty_lambda` Numeric penalty strength for penalized ML estimation (PML/PFIML). `NA` (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See `find_penalized_lambda` and `penalize`.

`penalty_alpha` Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

`penalize_matrices` Character vector of matrix names to penalize. If missing, defaults are selected based on the model type (the network `omega`).

Details

The Blume-Capel model is built on the *Spin* distribution:

$\Pr(\boldsymbol{Y} = \boldsymbol{y}) = \frac{\exp(-\beta H(\boldsymbol{y};\boldsymbol{\tau},\boldsymbol{\delta},\boldsymbol{\Omega}))}{Z(\boldsymbol{\tau},\boldsymbol{\delta},\boldsymbol{\Omega})}$

With Hamiltonian:

$H\left(\boldsymbol{y};\boldsymbol{\tau},\boldsymbol{\delta},\boldsymbol{\Omega}\right) = -\sum_{i=1}^{m}\tau_i y_i + \sum_{i=1}^{m}\delta_i y_i^2 - \sum_{i=2}^{m}\sum_{j=1}^{i-1}\omega_{ij}y_i y_j.$

and Z representing the partition function or normalizing constant. Equivalently,

$\Pr(\boldsymbol{Y} = \boldsymbol{y}) \propto \exp\left(\sum_i \tau_i y_i - \sum_i \delta_i y_i^2 + \sum_{i<j}\omega_{ij}y_i y_j\right).$

For the ternary states `c(-1, 0, 1)` this is the usual Blume-Capel parameterization: `tau` controls the tendency toward the positive versus the negative state, `delta` controls the tendency toward the zero/middle state versus the active/extreme states (positive `delta` favours the middle category), and `omega` controls pairwise alignment. For more general ordered numeric states the same parameterization defines an ordinal spin model.

The `Ising` model is nested within the Blume-Capel model by fixing all `delta` parameters to zero. Exact ML estimation enumerates every response pattern, so the cost grows as `length(responses)^nNode` (see `maxStates`); the model is therefore limited to relatively small networks.

Value

An object of the class psychonetrics

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

Epskamp, S., Maris, G., Waldorp, L. J., & Borsboom, D. (2018). Network Psychometrics. In: Irwing, P., Hughes, D., & Booth, T. (Eds.), The Wiley Handbook of Psychometric Testing, 2 Volume Set: A Multidisciplinary Reference on Survey, Scale and Test Development. New York: Wiley.

See Also

`Ising`

Examples

```
library("dplyr")

# Simulate a small Blume-Capel data set (requires the development version of
# IsingSampler that supports the 'delta' argument):
## Not run:
library("IsingSampler")
nNode  <- 5
graph  <- 0.4 * (matrix(1, nNode, nNode) - diag(nNode)) *
            (matrix(c(0,1,0,0,1, 1,0,1,0,0, 0,1,0,1,0, 0,0,1,0,1, 1,0,0,1,0), 5, 5))
data <- IsingSampler(1000, graph = graph, thresholds = rep(0, nNode),
                     beta = 1, delta = 1, responses = c(-1, 0, 1))

# Fit the Blume-Capel model:
mod <- BlumeCapel(data, responses = c(-1, 0, 1)) %>% runmodel

# Estimated network:
getmatrix(mod, "omega")

# Estimated quadratic node potentials:
getmatrix(mod, "delta")

## End(Not run)
```

`bootstrap` *Bootstrap a psychonetrics model*

Description

This function will bootstrap the data (once) and return a new unevaluated psychonetrics object. It requres `storedata = TRUE` to be used when forming a model.

Usage

```
bootstrap(x, replacement = TRUE, proportion = 1, verbose = TRUE, storedata = FALSE,
          baseline_saturated = TRUE)
```

Arguments

`x`	A `psychonetrics` model.
`replacement`	Logical, should new samples be drawn with replacement?
`proportion`	Proportion of sample to be drawn. Set to lower than 1 for subsampling.
`verbose`	Logical, should messages be printed?
`storedata`	Logical, should the bootstrapped data also be stored?
`baseline_saturated`	Logical, should the baseline and saturated models be included?

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

`changedata` *Change the data of a psychonetrics object*

Description

This function can be used to change the data in a psychonetrics object.

Usage

```
changedata(x, data, covs, nobs, means, groups, missing = "listwise")
```

Arguments

`x`	A `psychonetrics` model.
`data`	A data frame encoding the data used in the analysis. Can be missing if `covs` and `nobs` are supplied.
`covs`	A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. IMPORTANT NOTE: psychonetrics expects the maximum likelihood (ML) covariance matrix, which is NOT obtained from `cov` directly. Manually rescale the result of `cov` with `(nobs - 1)/nobs` to obtain the ML covariance matrix.
`nobs`	The number of observations used in `covs` and `means`, or a vector of such numbers of observations for multiple groups.
`means`	A vector of sample means, or a list/matrix containing such vectors for multiple groups.
`groups`	An optional string indicating the name of the group variable in `data`.
`missing`	How should missingness be handled in computing the sample covariances and number of observations when `data` is used. Can be `"listwise"` for listwise deletion, or `"pairwise"` for pairwise deletion.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

CIplot *Plot Analytic Confidence Intervals*

Description

Function to plot analytic confidence intervals (CI) of matrix elements estimated in psychonetrics.

Usage

```
CIplot(x, matrices, alpha_ci = 0.05, alpha_color = c(0.05,
                  0.01, 0.001, 1e-04), labels, labels2, labelstart,
                  print = TRUE, major_break = 0.2, minor_break = 0.1,
                  split0, prop0, prop0_cex = 1, prop0_alpha = 0.95,
                  prop0_minAlpha = 0.25, time)
```

Arguments

x	A psychonetrics model.
matrices	Vector of strings indicating the matrices to plot CIs for. For ri_clpm / ri_clpn models these may also be block names such as "omega_zeta_within" or "omega_zeta_between"; see Details.
time	Only for ri_clpm / ri_clpn models: the wave (an index or a column name of the design matrix) whose block should be plotted. When omitted, every wave is plotted in its own panel. See Details.
alpha_ci	The alpha level used for the CIs
alpha_color	A vector of alphas used for coloring the CIs
labels	The labels for the variables associated with the rows of a matrix.
labels2	The labels for the variables associated with the columns of a matrix. Defaults to the value of labels for square matrices.
labelstart	The value to determine if labels are printed to the right or to the left of the CI
print	Logical, should the plots also be printed? Only works when one matrix is used in 'matrices'
major_break	Numeric indicating the step size between major breaks
minor_break	Numeric indicating the step size between minor breaks
split0	Logical only used for results of aggregate_bootstraps. When set to TRUE, the displayed intervals are based on occasions when the parameter was not estimated to be zero, and an extra box is added indicating the number of times a parameter is estimated to be zero. Defaults to TRUE when model selection is used and FALSE otherwise.
prop0	Logical only used for results of aggregate_bootstraps, should boxes indicating the proportion of times parameters were estimated to be zero be added to the plot? Defaults to the value of split0.

prop0_cex	Only used for results of aggregate_bootstraps. Size of the boxes indicating number of times a parameter was set to zero.
prop0_alpha	Only used for results of aggregate_bootstraps. Transparency of the boxes indicating number of times a parameter was set to zero.
prop0_minAlpha	Only used for results of aggregate_bootstraps. Minimal transparency of the *lines* of plotted intervals as the proportion of times an edge was not included goes to 0.

Details

RI-CLPM / RI-CLPN models. A model made with ri_clpm or ri_clpn is built on the lvm framework, so each of its model matrices spans all waves at once (the within-person components of every wave, followed by the random intercepts). Plotting such a matrix would mix the temporal effects, the contemporaneous relations of every wave and the between-person relations into a single unreadable figure. For these models CIplot therefore works on the *blocks*, addressed with the same matrix names that panelvar and panelgvar use: "beta" (temporal), "omega_zeta_within" / "sigma_zeta_within" / ... (contemporaneous) and "omega_zeta_between" / ... (between-person), plus "nu_within" (wave-specific intercepts) and "nu_between" (random-intercept means). The default matrices for these models is the temporal, within-person and between-person block, mirroring the panelvar default.

Use time to plot one wave; without it every wave gets its own plot, titled with the matrix name and the wave. Waves whose parameters were equated by ri_clpm_stationary are collapsed into a single plot, so a stationary structure yields one panel rather than several identical ones. Node labels are the plain variable names of the design matrix rather than the internal latent names. Partial directed correlations are not available here, because the RI-CLPM has no PDC parameters and hence no standard errors for them; use getmatrix(x, "PDC", time =) for the point estimates.

Value

A single ggplot2 object, or a list of ggplot2 objects for each matrix requested.

Author(s)

Sacha Epskamp

Examples

```
### Example from ?ggm ###
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)
```

```
# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit an empty GGM:
mod0 <- ggm(ConsData, vars = vars)

# Run the model:
mod0 <- mod0 %>% runmodel

# Labels:
labels <- c(
  "indifferent to the feelings of others",
  "inquire about others' well-being",
  "comfort others",
  "love children",
  "make people feel at ease")

# Plot the CIs:
CIplot(mod0,  "omega", labels = labels, labelstart = 0.2)

### Example from ?gvar ###
library("dplyr")
library("graphicalVAR")

beta <- matrix(c(
  0,0.5,
  0.5,0
),2,2,byrow=TRUE)
kappa <- diag(2)
simData <- graphicalVARsim(50, beta, kappa)

# Form model:
model <- gvar(simData)

# Evaluate model:
model <- model %>% runmodel

# Plot the CIs:
CIplot(model,  "beta")
```

---

compare *Model comparison*

Description

This function will print a table comparing multiple models on chi-square, AIC and BIC. When all compared models are fitted with a robust estimator, a Satorra-Bentler-family scaled chi-square difference test is added.

Usage

```
compare(..., scaled.test.method = c("auto", "satorra.bentler.2001",
        "satorra.bentler.2010", "satorra.2000"))

## S3 method for class 'psychonetrics_compare'
print(x, ...)
```

Arguments

`...`	Any number of psychonetrics models. Can be named to change the rownames of the output.
`scaled.test.method`	The scaled chi-square difference test to use when all compared models carry a robust scaling factor (see Details). `"auto"` (the default) selects the appropriate test from the models' scaled-test type, matching **lavaan**'s `lavTestLRT`: the exact Satorra (2000) scaled-and-shifted test for the scaled-shifted (`MLMV`, `WLSMV`) and mean-and-variance-adjusted (`MLMVS`) estimators, and the Satorra-Bentler (2001) mean-scaled test for `MLM` / `MLR`. The remaining options force a specific test: `"satorra.bentler.2001"` (a mean-scaled test), `"satorra.bentler.2010"` (a strictly positive mean-scaled variant), or `"satorra.2000"` (an exact, scaled-and-shifted test, appropriate for the scaled-shifted estimators such as `MLMV` and `WLSMV`).
`x`	Output of the `compare` function.

Details

When every compared model is fitted with a robust estimator (`MLM`, `MLMV`, `MLMVS` or `MLR`, or one of the least-squares estimators `DWLS`, `WLS` and `ULS`), two additional columns are added: `Chisq_diff_scaled` and `p_value_scaled`. These contain the Satorra-Bentler-family scaled chi-square difference test for each adjacent (nested) pair of models, sorted by degrees of freedom. The ordinary (unscaled) `Chisq_diff`, `DF_diff` and `p_value` columns are always present and unchanged. When the models are not fitted with a robust estimator (e.g. ordinary `ML` or `FIML`), the scaled columns are omitted entirely.

For an adjacent nested pair with the more constrained model M_0 (degrees of freedom r_0, unscaled chi-square T_0, scaling factor c_0) nested in the less constrained model M_1 (r_1, T_1, c_1), and $m = r_0 - r_1$:

- `"satorra.bentler.2001"` computes $c_d = (r_0 c_0 - r_1 c_1)/m$ and $T_d = (T_0 - T_1)/c_d$ (Satorra & Bentler, 2001).
- `"satorra.bentler.2010"` replaces c_1 by the scaling factor of M_1 evaluated at M_0's estimates, guaranteeing a positive statistic (Satorra & Bentler, 2010).
- `"satorra.2000"` computes an exact, scaled-and-shifted statistic from the asymptotic covariance of the sample statistics (Satorra, 2000; Asparouhov & Muthen, 2010); this is the appropriate difference test for the scaled-shifted estimators `MLMV` and `WLSMV`.

The statistic is set to `NA` (with a warning) when the implied scaling factor is non-positive or a required building block cannot be computed.

Value

A data frame with chi-square values, degrees of freedoms, RMSEAs, AICs, and BICs, and—when all models are fitted with a robust estimator—scaled chi-square difference columns.

Author(s)

Sacha Epskamp

References

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

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

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

Asparouhov, T., & Muthen, B. (2010). *Simple second order chi-square correction*. Mplus technical appendix.

Examples

```
## Not run:
library("dplyr")

# Load data:
data(StarWars)

# Originally a partial measurement invariance model in
# https://psychonetrics.org/files/Epskampetal2017.pdf:
mod1 <- lvm(StarWars, lambda = matrix(1, 10), vars = paste0("Q", 1:10),
            identification = "variance", latents = "Force",
            estimator = "MLM") %>% runmodel

# A model fixing two prequel loadings to zero:
mod2 <- mod1 %>% fixpar("lambda", row = "Q4", col = "Force") %>%
  fixpar("lambda", row = "Q5", col = "Force") %>% runmodel

# Compare with the scaled (Satorra-Bentler 2001) difference test:
compare(full = mod1, restricted = mod2)

## End(Not run)
```

covML *Maximum likelihood covariance estimate*

Description

These functions complement the base R cov function by simplifying obtaining maximum likelihood (ML) covariance estimates (denominator n) instead of unbiased (UB) covariance estimates (denominator n-1). The function covML can be used to obtain ML estimates, the function covUBtoML transforms from UB to ML estimates, and the function covMLtoUB transforms from ML to UB estimates.

Usage

```
covML(x, ...)
covUBtoML(x, n, ...)
covMLtoUB(x, n, ...)
```

Arguments

x	A dataset
n	The sample size
...	Arguments sent to the cov function.

Value

A covariance matrix: the maximum likelihood (ML) or unbiased (UB) variance–covariance matrix of x, depending on the function used.

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

Examples

```
data("StarWars")
Y <- StarWars[,1:10]

# Unbiased estimate:
UB <- cov(Y)

# ML Estimate:
ML <- covML(Y)

# Check:
all(abs(UB - covMLtoUB(ML, nrow(Y))) < sqrt(.Machine$double.eps))
all(abs(ML - covUBtoML(UB, nrow(Y))) < sqrt(.Machine$double.eps))
```

diagnostics *Diagnostic functions*

Description

The 'checkJacobian' function can be used to check if the analytic gradient / Jacobian is aligned with the numerically approximated gradient / Jacobian, and the 'checkFisher' function can be used to check if the analytic Hessian is aligned with the numerically approximated Hessian.

Usage

```
checkJacobian(x, f = "default", jac = "default", transpose = FALSE,
          plot = TRUE, perturbStart = FALSE, method = "Richardson")

checkFisher(x, f = "default", fis = "default", transpose = FALSE,
          plot = TRUE,  perturbStart = FALSE)
```

Arguments

x	A 'psychonetrics' object
f	A custom fit function or the psychonetrics default fit function (default).
jac	A custom Jacobian function or the psychonetrics default Jacobian function (default).
fis	A custom Fisher information function or the psychonetrics default Fisher information function (default).
transpose	Should the numeric Jacobian be transposed?
plot	Should a diagnostic plot be produced?
perturbStart	Should start values be perturbed (only used in development)
method	Numeric derivative method (default: Richardson)

Value

Invisibly returns a list with two elements, `analytic` and `numeric`, containing the analytic and the numerically approximated gradient/Jacobian (`checkJacobian`) or Hessian/Fisher information (`checkFisher`) values being compared. A scatter plot of the two is drawn when `plot = TRUE`.

Author(s)

Sacha Epskamp

dlvm1 *Lag-1 dynamic latent variable model family of psychonetrics models for panel data*

Description

This is the family of models that models a dynamic factor model on panel data. All functions accept both wide-format data (with a design matrix for `vars`) and long-format data (with a character vector for `vars` plus `idvar` and optionally `beepvar`). The format is auto-detected from the type of `vars`. There are four covariance structures that can be modeled in different ways: `within_latent`, `between_latent` for the within-person and between-person latent (contemporaneous) models respectively, and `within_residual`, `between_residual` for the within-person and between-person residual models respectively. The `panellvgvar` wrapper automatically models all latent networks as GGMs. The old name `panel_lvgvar` is deprecated in favor of `panellvgvar`. `dlvm()` is the primary short name of `dlvm1()` (identical function; the historical "1" referred to the lag order). The observed-variable special case of this family (a factor loading matrix fixed to identity and zero residual variances) is the panel VAR model, implemented as its own faster framework in `panelvar` and `panelgvar`. The mean structure is stationary by default (item intercepts `nu` invariant across waves, latent means fixed to zero); `stationary_means = FALSE` makes the intercepts wave-specific, giving the trend-stationary case (see the details below).

Usage

```
dlvm1(data, vars, datatype = c("auto", "wide", "long"),
                  idvar, beepvar,
                  standardize = c("none", "z", "quantile", "z_per_wave"),
                  lambda, within_latent = c("cov", "chol",
                  "prec", "ggm", "cor"), within_residual = c("cov", "chol",
                  "prec", "ggm", "cor"), between_latent = c("cov", "chol",
                  "prec", "ggm", "cor"), between_residual = c("cov", "chol",
                  "prec", "ggm", "cor"), beta = "full", omega_zeta_within =
                  "full", delta_zeta_within = "diag", kappa_zeta_within
                  = "full", sigma_zeta_within = "full",
                  lowertri_zeta_within = "full", omega_epsilon_within =
                  "zero", delta_epsilon_within = "diag",
                  kappa_epsilon_within = "diag", sigma_epsilon_within =
                  "diag", lowertri_epsilon_within = "diag",
                  omega_zeta_between = "full", delta_zeta_between =
                  "diag", kappa_zeta_between = "full",
                  sigma_zeta_between = "full", lowertri_zeta_between =
                  "full", omega_epsilon_between = "zero",
                  delta_epsilon_between = "diag", kappa_epsilon_between
                  = "diag", sigma_epsilon_between = "diag",
                  lowertri_epsilon_between = "diag", nu, mu_eta,
                  identify = TRUE, identification = c("loadings",
                  "variance"), latents, groups, groupvar, covs,
                  cors, means, nobs, corinput, start
```

```
= "version2", covtype = c("choose", "ML", "UB"),
missing = "auto", equal = "none",
baseline_saturated = TRUE, estimator = "ML",
optimizer, storedata = FALSE, verbose = FALSE,
sampleStats, baseline =
c("stationary_random_intercept", "stationary",
"independence", "none"), bootstrap = FALSE, boot_sub,
boot_resample, within, between,
penalty_lambda = NA, penalty_alpha = 1,
penalize_matrices,
rho_zeta_within = "full", SD_zeta_within = "diag",
rho_epsilon_within = "zero", SD_epsilon_within = "diag",
rho_zeta_between = "full", SD_zeta_between = "diag",
rho_epsilon_between = "zero", SD_epsilon_between = "diag",
beta_epsilon = "zero",
temporal_latent = c("raw","PDC"),
temporal_residual = c("raw","PDC"),
PDC = "full", PDC_epsilon = "zero",
stationary_means = TRUE)
```

```
dlvm(...)
```

```
panellvgvar(...)
```

```
panel_lvgvar(...)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied.
vars	Required argument. For wide-format data, this must be a *matrix* with each row indicating a variable and each column indicating a measurement. The matrix must be filled with names of the variables in the dataset corresponding to variable i at wave j. NAs can be used to indicate missing waves. The rownames of this matrix will be used as variable names. For long-format data, this should be a character vector of variable names in the dataset.
datatype	Data format: "auto" (default) auto-detects based on whether vars is a matrix (wide) or character vector (long). "wide" forces wide-format interpretation. "long" forces long-format interpretation with automatic reshape to wide.
idvar	Optional string indicating the subject/cluster ID variable in data. Required when datatype = "long" or when vars is a character vector.
beepvar	Optional string indicating the time point / measurement occasion variable. If missing when datatype = "long", a sequential measurement variable is created per subject.
standardize	Standardization method: "none" (default), "z" for global z-scores per variable across all waves (uses the overall mean and standard deviation for each variable computed across all waves of data), "quantile" for quantile normalization across all waves, "z_per_wave" for independent z-scores per wave column

(note: this imposes stationarity as means become 0 and variances become 1 at each wave).

`lambda` Required argument. A model matrix encoding the factor loading structure. Each row indicates an indicator and each column a latent. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`within_latent` The type of within-person latent contemporaneous model to be used.

`within_residual`
The type of within-person residual model to be used.

`between_latent` The type of between-person latent model to be used.

`between_residual`
The type of between-person residual model to be used.

`beta` A model matrix encoding the temporal relationships (transpose of temporal network). A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Can also be `"full"` for a full temporal network or `"zero"` for an empty temporal network.

`omega_zeta_within`
Only used when `within_latent = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_zeta_within`
Only used when `within_latent = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_zeta_within`
Only used when `within_latent = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_zeta_within`
Only used when `within_latent = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_zeta_within`
Only used when `within_latent = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating

free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_epsilon_within`
: Only used when `within_residual = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_epsilon_within`
: Only used when `within_residual = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_epsilon_within`
: Only used when `within_residual = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_epsilon_within`
: Only used when `within_residual = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_epsilon_within`
: Only used when `within_residual = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_zeta_between`
: Only used when `between_latent = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_zeta_between`
: Only used when `between_latent = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_zeta_between`
: Only used when `between_latent = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating

free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_zeta_between`
: Only used when `between_latent = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_zeta_between`
: Only used when `between_latent = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_epsilon_between`
: Only used when `between_residual = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_epsilon_between`
: Only used when `between_residual = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_epsilon_between`
: Only used when `between_residual = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_epsilon_between`
: Only used when `between_residual = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_epsilon_between`
: Only used when `between_residual = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`nu`
: Optional vector encoding the intercepts of the observed variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher

integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector. When `stationary_means = FALSE` this is instead a `nVar` by `nTime` matrix (variables in the rows, waves in the columns), or an array/list of such matrices for multiple groups; elements at variable–wave combinations that are not in the `vars` design matrix are always fixed to zero.

`mu_eta` Optional vector encoding the means of the latent variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector. The latent means are fixed to zero by `identify()` for identification, with a stationary as well as with a wave-specific mean structure.

`identify` Logical, should the model be automatically identified?

`identification` Type of identification used. `"loadings"` to fix the first factor loadings to 1, and `"variance"` to fix the diagonal of the latent variable model matrix (sigma_zeta, lowertri_zeta, delta_zeta or kappa_zeta) to 1.

`latents` An optional character vector with names of the latent variables.

`groups` Deprecated. Use `groupvar` instead. An optional string indicating the name of the group variable in `data`.

`groupvar` An optional string indicating the name of the group variable in `data`. Replaces the deprecated `groups` argument; if both are supplied, `groupvar` takes precedence with a warning.

`covs` A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure the `covtype` argument is set correctly to the type of covariances used.

`cors` A sample correlation matrix, or a list/array of such matrices for multiple groups. When supplied, the matrix is treated as a covariance matrix with a warning (appropriate for standardized data). Requires `nobs`. Note: `corinput = TRUE` is not supported in `dlvm1`.

`means` A vector of sample means, or a list/matrix containing such vectors for multiple groups. This argument always refers to the sample means; the mean *structure* of the model is set with `stationary_means`.

`nobs` The number of observations used in `covs` and `means`, or a vector of such numbers of observations for multiple groups.

`corinput` Logical. Not supported in `dlvm1()` and will produce an error if set to `TRUE`.

`start` Start value specification. Can be either a string or a psychonetrics model. If it is a string, `"version2"` indicates the latest version of start value computation, `"version1"` indicates start values as they were computed up to version 0.11, and `"simple"` indicate simple starting values. If this is a psychonetrics model the starting values will be based on the output. This can be useful, for example, if you first estimate a model with matrices set to a Cholesky decomposition, then use those values as start values for estimating Gaussian graphical models.

`missing` How should missingness be handled in computing the sample covariances and number of observations when `data` is used. Can be `"auto"` (default) for automatic detection, `"listwise"` for listwise deletion, or `"pairwise"` for pairwise

deletion. When "auto", the function checks for missing data and switches ML to FIML, PML to PFIML, or defaults to listwise for LS estimators.

equal — A character vector indicating which matrices should be constrained equal across groups.

baseline_saturated — A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

estimator — The estimator to be used. Currently implemented are "ML" for maximum likelihood estimation, "FIML" for full-information maximum likelihood estimation, "PML" for penalized maximum likelihood estimation, "PFIML" for penalized full-information maximum likelihood estimation, "ULS" for unweighted least squares estimation, "WLS" for weighted least squares estimation, and "DWLS" for diagonally weighted least squares estimation. When missing = "auto" (default), "ML" is automatically switched to "FIML" and "PML" to "PFIML" if missing data is detected.

optimizer — The optimizer to be used. Can be one of "nlminb" (the default R nlminb function), "ucminf" (from the optimr package), "nloptr_TNEWTON" (preconditioned truncated Newton via nloptr), and "LBFGS++" (pure C++ L-BFGS-B). Defaults to "nlminb".

storedata — Logical, should the raw data be stored? Needed for bootstrapping (see bootstrap).

verbose — Logical, should progress be printed to the console?

sampleStats — An optional sample statistics object. Mostly used internally.

covtype — If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to "ML" for maximum likelihood estimates (denominator n) and "UB" to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.

baseline — What baseline model should be used? "stationary_random_intercept" includes both within- and between person variances constrained equal across time (default), "stationary" only includes within-person variances constrained equal across time, "independence" (default up to version 0.11) includes a variance for every variable at every time point (not constrained equal across time), and "none" includes no baseline model. The baseline model always uses the same mean structure as the model itself (see the stationary_means argument and the details below).

bootstrap — Should the data be bootstrapped? If TRUE the data are resampled and a bootstrap sample is created. These must be aggregated using aggregate_bootstraps! Can be TRUE or FALSE. Can also be "nonparametric" (which sets boot_sub = 1 and boot_resample = TRUE) or "case" (which sets boot_sub = 0.75 and boot_resample = FALSE).

boot_sub — Proportion of cases to be subsampled (round(boot_sub * N)).

boot_resample — Logical, should the bootstrap be with replacement (TRUE) or without replacement (FALSE)

within — Optional alias for within_latent. If both within and within_latent are explicitly specified, within_latent takes precedence and a warning is issued.

`between` Optional alias for `between_latent`. If both `between` and `between_latent` are explicitly specified, `between_latent` takes precedence and a warning is issued.

`penalty_lambda` Numeric penalty strength for penalized ML estimation (PML/PFIML). `NA` (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See `find_penalized_lambda` and `penalize`.

`penalty_alpha` Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

`penalize_matrices` Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.

`rho_zeta_within` Only used when `within_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix; see `omega_zeta_within`).

`SD_zeta_within` Only used when `within_latent = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix; see `delta_zeta_within`).

`rho_epsilon_within` Only used when `within_residual = "cor"`. Correlation matrix specification (`"zero"`, `"full"`, or a pattern matrix).

`SD_epsilon_within` Only used when `within_residual = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix).

`rho_zeta_between` Only used when `between_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).

`SD_zeta_between` Only used when `between_latent = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix).

`rho_epsilon_between` Only used when `between_residual = "cor"`. Correlation matrix specification (`"zero"`, `"full"`, or a pattern matrix).

`SD_epsilon_between` Only used when `between_residual = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix).

`beta_epsilon` Residual (within-person) temporal effects: a model matrix for the lag-1 autoregression of the residuals, `epsilon_t = beta_epsilon epsilon_(t-1) + u_t`. `"zero"` (default) gives white-noise residuals (the model as it was before this argument existed). `"diag"` gives each indicator its own residual autoregression, which models item-specific carry-over (autocorrelated uniquenesses) with a geometrically decaying structure; `"full"` or a pattern matrix are also allowed. When `beta_epsilon` is non-zero, `sigma_epsilon_within` (and its typed variants) parameterizes the *innovation* covariance of the residual process, and the stationary residual covariance is available as the `"sigma_epsilon_within_stationary"` matrix in `getmatrix`. At least three waves are required for a non-zero `beta_epsilon` to be identified, and at least four are recommended (the residual autoregression, the stable between-person residual variance in `sigma_epsilon_between`, and the latent dynamics compete for the same lagged covariance information).

`temporal_latent`
: Parameterization of the latent temporal effects: `"raw"` (default) models `beta` directly; `"PDC"` models the latent partial directed correlations directly, with the `PDC` matrix encoding from = row and to = column (the transposed orientation relative to `beta`). A (pattern-constrained) PDC model is fit-equivalent to the corresponding raw model, but standard errors, confidence intervals, equality constraints and modification indices then apply directly on the PDC scale.

`temporal_residual`
: Parameterization of the residual temporal effects (`beta_epsilon`): `"raw"` (default) or `"PDC"` (models the residual partial directed correlations `PDC_epsilon`, standardized with respect to the residual innovation covariance, directly).

`PDC`
: Only used when `temporal_latent = "PDC"`. Model matrix for the latent partial directed correlations (from = row, to = column): `"full"`, `"zero"`, `"diag"`, or a pattern matrix.

`PDC_epsilon`
: Only used when `temporal_residual = "PDC"`. Model matrix for the residual partial directed correlations (from = row, to = column): `"zero"` (default), `"diag"`, `"full"`, or a pattern matrix.

`stationary_means`
: Logical. `TRUE` (default, and the behavior of all earlier versions) has item intercepts `nu` that are invariant across waves and latent means fixed to zero, so that the implied means are equal across waves. `FALSE` makes the item intercepts wave-specific (the `nu` matrix becomes `nVar` by `nTime`), which saturates the mean structure and adds `nVar * (nTime - 1)` parameters; the latent means stay fixed to zero. See the details below.

`...`
: Arguments sent to `dlvm1`.

Details

Stationary versus wave-specific means. By default the model is fully stationary in the means: the item intercepts `nu` are invariant across waves and the latent means are fixed to zero, so the implied means are equal at every wave. With `stationary_means = FALSE` the item intercepts become wave-specific, $\mu_t = \nu_t + \Lambda\alpha$ with $\alpha = 0$. This is the classical *trend-stationary* case: the level shifts are period effects exogenous to the dynamics, the within-person latent deviation process remains covariance-stationary, and wave 1 is still drawn from that stationary distribution. The implied covariance matrix is therefore *identical* to that of the stationary-means model; only the mean part of the model (and of its Jacobian) changes.

Identification. The latent means (`mu_eta`) are fixed to zero with a stationary as well as with a wave-specific mean structure, and the wave-specific intercepts are free: one per variable–wave combination present in the `vars` design matrix (variable–wave combinations that are absent have no intercept parameter). Wave-specific intercepts and wave-specific latent means are not jointly identified, so one of the two has to carry the level shifts. Placing them in the intercepts saturates the mean structure, which is what keeps the baseline model (an observed-variable `panelvar` model, which cannot have latent means) nested in the fitted model, so that incremental fit indices remain interpretable. The estimated intercepts then equal the observed per-wave means, and the likelihood-ratio test of `stationary_means = TRUE` against `FALSE` is a test of equal means over time with `nVar * (nTime - 1)` degrees of freedom. The flip side is that a saturated mean structure cannot separate period effects on the constructs from mean-level measurement drift (scalar non-invariance), and

cannot be tested for misfit: nu absorbs both. Users who want the more restrictive alternative (invariant intercepts with free latent means at waves 2 and later, nLat(nTime - 1) parameters, testable when nLat < nVar) can specify it manually with the nu and mu_eta arguments plus identify = FALSE, keeping in mind that the baseline model is then no longer nested in the fitted model. In the degenerate observed-variable case (lambda the identity matrix with zero residual variances) the model with stationary_means = FALSE is identical to panelvar with stationary_means = FALSE, with nu playing the role of mu.

No free innovation variances. Only the means can be freed over time, not the innovation (co)variances: a wave-varying innovation covariance has no stationary distribution for wave 1 to be drawn from, so the stationarity machinery on which this family is built does not apply. Wave-specific innovation structures are covered by the RI-CLPM / RI-CLPN framework instead, see ri_clpm.

Baseline model. The baseline model always uses the same mean structure as the fitted model: with stationary_means = FALSE the baseline (a panelvar model with empty networks) gets wave-specific means as well, so that both models have exactly the same (saturated) mean structure. A baseline model must be nested in the fitted model, otherwise it can fit better than the fitted model on the mean side (a free-means baseline against an equal-means target would beat it), which makes incremental fit indices meaningless or negative. A corollary is that incremental fit indices (CFI, TLI, NFI, ...) are **not** directly comparable between a stationary_means = TRUE and a stationary_means = FALSE fit of the same data, because the two use different baselines by design; compare those two models with a likelihood-ratio test (compare) instead. The saturated model already has a saturated mean structure and is unchanged.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

panelvar for the observed-variable (panel VAR) special case of this family.

Examples

```
library("dplyr")

# Smoke data cov matrix, based on LISS data panel https://www.dataarchive.lissdata.nl
smoke <- structure(c(47.2361758611759, 43.5366809116809, 41.0057465682466,
                     43.5366809116809, 57.9789886039886, 47.6992521367521,
                     41.0057465682466,
                     47.6992521367521, 53.0669434731935), dim = c(3L, 3L),
                   dimnames = list(
                       c("smoke2008", "smoke2009", "smoke2010"), c("smoke2008",
                   "smoke2009", "smoke2010")))

# Design matrix:
design <- matrix(rownames(smoke),1,3)
```

```
# Form dlvm1 model with a single latent variable per wave:
mod <- dlvm1(vars = design, lambda = matrix(1),
             covs = smoke, nobs = 352
)


# Run model:
mod <- mod %>% runmodel

# Evaluate fit:
mod %>% fit
```

`duplicationMatrix` *Model matrices used in derivatives*

Description

These matrices are used in the analytic gradients

Usage

```
duplicationMatrix(n, diag = TRUE)

eliminationMatrix(n, diag = TRUE)

diagonalizationMatrix(n)
```

Arguments

`n`	Number of rows and columns in the original matrix
`diag`	Logical indicating if the diagonal should be included (set to FALSE for derivative of vech(x))

Value

A sparse matrix

Author(s)

Sacha Epskamp

Examples

```
# Duplication matrix for 10 variables:
duplicationMatrix(10)

# Elimination matrix for 10 variables:
eliminationMatrix(10)
```

```
# Diagonalization matrix for 10 variables:
diagonalizationMatrix(10)
```

`emergencystart` *Reset starting values to simple defaults*

Description

This function overwrites the starting values to simple defaults. This can help in cases where optimization fails.

Usage

```
emergencystart(x)
```

Arguments

`x`	A `psychonetrics` model.

Value

A `psychonetrics` model.

Author(s)

Sacha Epskamp

`equalityScoreTest` *Score (Lagrange Multiplier) test for cross-group equality constraints*

Description

Computes the score test (also known as the Lagrange Multiplier test) for releasing cross-group equality constraints in a multi-group `psychonetrics` model. Both the per-group univariate score statistics and the joint multivariate score statistic (one statistic per equality-constrained parameter, with df = G - 1) are returned.

The algorithm used by `method = "jacobian"` (the default) is a direct port of the score-test logic in `lavaan::lavTestScore` (Rosseel, 2012) and reproduces **lavaan**'s output to numerical precision. The derivation closely follows the implementation in `R/lav_test_score.R` of the **lavaan** source code (https://github.com/yrosseel/lavaan), reused here with thanks to Yves Rosseel under the GPL ($\geq$ 2) license shared by both packages. The underlying gradient and Fisher-information machinery is provided by **psychonetrics**, so no refitting and no run-time dependence on **lavaan** is required.

With the default `method = "jacobian"` the algorithm reproduces `lavaan::lavTestScore` exactly: the model is augmented with G-1 extra parameters per equality-constrained tuple (one per non-reference group), the gradient and expected information are computed at the constrained MLE, and for each constraint the test statistic

$$T = N \cdot s' \cdot \tilde{J}^{-1} \cdot s$$

is formed, where s is the gradient of the log-likelihood and $\tilde{J}^{-1}$ is the upper-left block of the generalised inverse of the bordered information matrix $[J\ R'^T;\ R\ 0]$, with R the rows of the constraint Jacobian for the constraints that are NOT being tested. This is the same construction used by `lavaan::lavTestScore`.

`method = "schur"` uses a faster Schur-complement construction (the same used by **psychonetrics** for ordinary modification indices). It is much faster for complex models because it inverts only the (smaller) block of the information matrix corresponding to the currently free parameters, but it is not exactly equivalent to `lavaan` in unbalanced multi-group samples (it agrees in balanced samples). Both methods follow the same asymptotic chi-square reference distribution.

Usage

```
equalityScoreTest(x, matrices, top = 10, verbose = TRUE,
                  method = c("jacobian","schur"))
```

Arguments

`x`	A fitted (run) multi-group `psychonetrics` model.
`matrices`	Optional vector of matrix names to restrict the test to. By default all matrices are considered.
`top`	How many rows to print in each table.
`verbose`	Logical, should the formatted tables be printed?
`method`	Construction of the score test. `"jacobian"` (default) reproduces `lavaan::lavTestScore` exactly. `"schur"` uses a Schur-complement construction that is faster for complex models but only matches `lavaan` for balanced multi-group samples.

Details

For two groups ($G = 2$) the joint test reduces to the univariate test (both have `df = 1`). The two tests differ for $G \geq 3$, where the joint test correctly accounts for the covariance between the released parameters via the (Schur-complemented) Fisher information, instead of summing or otherwise aggregating univariate statistics.

Value

Invisibly returns a list with two data frames:

`uni` Univariate score tests, one row per released parameter (i.e. one row per group beyond the reference group), sorted from largest to smallest X^2.

`total` Joint multivariate score tests, one row per equality-constrained parameter (i.e. one row per (matrix, row, column) triple), sorted from largest to smallest X^2. The joint test has `df = G - 1` when the parameter is constrained equal across all G groups.

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>. The `method = "jacobian"` implementation is a port of the score-test machinery in `lavaan::lavTestScore` by Yves Rosseel.

References

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

lavaan source code: https://github.com/yrosseel/lavaan (in particular `R/lav_test_score.R`).

See Also

`MIs`, `partialprune`

Examples

```
library("dplyr")
library("psychTools")
data(bfi)

ExData <- bfi %>% select(E1:E5, gender) %>% na.omit
vars <- names(ExData)[1:5]

mod <- ggm(ExData, vars = vars, groups = "gender") %>%
  runmodel %>%
  groupequal("omega") %>%
  runmodel

equalityScoreTest(mod)
```

---

esa *Ergodic Subspace Analysis*

Description

These functions implement Ergodic Subspace Analysis by von Oertzen, Schmiedek and Voelkle (2020). The functions can be used on the output of a `dlvm1` or `panelvar` model, or manually by supplying a within persons and between persons variance-covariance matrix.

Usage

```
esa(x, cutoff = 0.1,
    between = c("crosssection", "between"))
esa_manual(sigma_wp, sigma_bp, cutoff = 0.1)
## S3 method for class 'esa'
print(x, printref = TRUE, ...)
## S3 method for class 'esa_manual'
```

```
print(x, printref = TRUE, ...)
## S3 method for class 'esa'
plot(x, plot = c("observed", "latent"), ...)
## S3 method for class 'esa_manual'
plot(x,  ...)
```

Arguments

x	Output of a dlvm1 or panelvar model
sigma_wp	Manual within-person variance-covariance matrix
sigma_bp	Manual between-person variance-covariance matrix
cutoff	Cutoff used to determine ergodicity
printref	Logical, should the reference be printed?
plot	Should ergodicity of observed or latent variables be plotted?
between	Should the between-persons variance-covariance matrix be based on expected cross-sectional or between-person relations
...	Not used

Value

For each group a esa_manual object with the following elements:

ergodicity	Ergodicity values of each component
Q_esa	Component loadings
V_bp	Between persons subspace
V_ergodic	Ergodic subspace
V_wp	Within person subspace
cutoff	Cutoff value used

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

von Oertzen, T., Schmiedek, F., and Voelkle, M. C. (2020). Ergodic Subspace Analysis. Journal of Intelligence, 8(1), 3.

factorscores *Compute factor scores*

Description

Currently, only the `lvm` framework with no missing data is supported.

Usage

```
factorscores(data, model, method = c("bartlett", "regression"))
```

Arguments

`data`	Dataset to compute factor scores for
`model`	A psychonetrics model
`method`	The method to use: `"regression"` or `"bartlett"`

Value

A data frame of factor scores (one column per latent variable). For multi-group models a group column is appended.

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

find_penalized_lambda *Automated Lambda Selection for Penalized Estimation*

Description

Selects the optimal penalty strength (lambda) for penalized maximum likelihood (PML) or penalized full-information maximum likelihood (PFIML) models via EBIC-based grid search. A log-spaced grid of lambda values is searched from `lambda_max` (the smallest lambda for which all penalized parameters are exactly zero, derived from KKT conditions) down to `lambda_max * lambda_min_ratio`. Each lambda is optimized using ISTA (Iterative Shrinkage-Thresholding Algorithm) with warm starts from the previous solution, and the model with the best EBIC is selected. After selection, a `beta_min` threshold is applied to set near-zero penalized parameters exactly to zero.

This function is called automatically by `runmodel` when the model contains auto-select penalty parameters (i.e., `penalty_lambda = NA` in the parameter table, which is the default when using `estimator = "PML"` or `estimator = "PFIML"`). It can also be called directly for more control over the search.

Usage

```
find_penalized_lambda(model, criterion = "ebic.5", nLambda = 50,
                      lambda_min_ratio = 1e-4,
                      beta_min = c("numerical", "theoretical"),
                      verbose)
```

Arguments

`model`	A `psychonetrics` model with `estimator = "PML"` or `estimator = "PFIML"`. The model should contain parameters with `penalty_lambda = NA` (auto-select), which is the default when specifying `estimator = "PML"` or `"PFIML"` in model constructors such as `ggm`, `lvm`, etc.
`criterion`	Character string specifying the information criterion used to select the best lambda. One of: `"bic"` BIC (equivalent to EBIC with gamma = 0) `"ebic.25"` EBIC with gamma = 0.25 `"ebic.5"` EBIC with gamma = 0.5 (default; recommended for network models) `"ebic.75"` EBIC with gamma = 0.75 `"ebic1"` EBIC with gamma = 1.0 Higher gamma values penalize model complexity more strongly and tend to produce sparser solutions. Gamma = 0.5 is widely recommended for network models (Foygel & Drton, 2010).
`nLambda`	Integer, the number of lambda values in the search grid. Default is 50. The grid is log-spaced between `lambda_max` and `lambda_max * lambda_min_ratio`.
`lambda_min_ratio`	Numeric, the ratio of the smallest to the largest lambda in the grid. Default is `1e-4`, so that `lambda_min = lambda_max * 1e-4`.
`beta_min`	Threshold for zeroing out small penalized parameter estimates. Parameters with absolute value below `beta_min` are set to zero, both during EBIC evaluation (for parameter counting) and in the final model. Can be: `"numerical"` Uses a fixed threshold of `1e-05` (default). `"theoretical"` Uses `sqrt(log(p) / n)` where `p` is the number of penalized parameters and `n` is the total sample size. **numeric value** Uses the provided value directly as the threshold.
`verbose`	Logical, should progress messages be printed? Defaults to the model's `verbose` setting.

Details

The search proceeds as follows:

1. **Compute lambda_max**: The analytical upper bound is derived from KKT (Karush-Kuhn-Tucker) conditions at the null-penalized model (all penalized parameters fixed at zero). Specifically, `lambda_max = max(|grad_j|) / alpha` where `grad_j` are the gradient elements for penalized parameters at the null solution and `alpha` is the elastic net mixing parameter.

2. **Build lambda grid**: A log-spaced sequence of `nLambda` values from `lambda_max` down to `lambda_max * lambda_min_ratio`.
3. **Grid search with warm starts**: For each lambda (from largest to smallest), the model is optimized using ISTA. The solution from the previous lambda serves as the starting point (warm start), which substantially reduces computation time. The EBIC (Extended Bayesian Information Criterion) is computed using the unpenalized log-likelihood (Convention A), counting only parameters with `|estimate| >= beta_min` as non-zero.
4. **Beta-min thresholding**: Penalized parameters with `|estimate| < beta_min` are set exactly to zero in the final model.

The EBIC is computed as:

$$EBIC = -2 \cdot LL + k \cdot \log(n) + 4 \cdot k \cdot \gamma \cdot \log(p_v)$$

where LL is the unpenalized log-likelihood, k is the effective number of parameters (unpenalized free parameters plus non-zero penalized parameters), n is the sample size, γ is the EBIC tuning parameter, and p_v is the number of observed variables.

The returned model stores metadata about the search in `model@optim$lambda_search`, a list containing: `criterion`, `gamma`, `best_lambda`, `best_criterion`, `beta_min`, `lambda_max`, `nLambda`, and `n_thresholded`.

Value

An object of class `psychonetrics` (psychonetrics-class) with:

- The selected lambda assigned to all auto-select penalty parameters
- Parameters optimized at the best lambda
- Near-zero penalized parameters set to exactly zero (via `beta_min`)
- Search metadata stored in `model@optim$lambda_search`

After running `find_penalized_lambda`, use `refit` to obtain valid standard errors and fit indices via post-selection inference.

Author(s)

Sacha Epskamp

References

Foygel, R., & Drton, M. (2010). Extended Bayesian Information Criteria for Gaussian Graphical Models. In *Advances in Neural Information Processing Systems* (Vol. 23).

See Also

`penalize` for manually setting penalty parameters, `runmodel` for running models (calls `find_penalized_lambda` automatically when auto-select penalties are present), `refit` for post-selection inference after penalized estimation.

Examples

```
# Load bfi data from psych package:
library("psychTools")
library("dplyr")
data(bfi)

ConsData <- bfi %>%
  select(A1:A5) %>%
  na.omit

vars <- names(ConsData)

# Automatic lambda selection (called implicitly by runmodel):
mod <- ggm(ConsData, vars = vars, estimator = "PML")
mod <- mod %>% runmodel  # automatically searches for best lambda

# Check lambda search results:
mod@optim$lambda_search

# Post-selection refit for standard errors:
mod_refit <- mod %>% refit
mod_refit %>% parameters

# Direct call with custom criterion:
mod2 <- ggm(ConsData, vars = vars, estimator = "PML")
mod2 <- find_penalized_lambda(mod2, criterion = "ebic1",
                              nLambda = 100)
```

`fit` *Print fit indices*

Description

This function will print all fit indices of the model. For the least-squares estimators with a mean-and-variance adjusted test (`"DWLS"`/`"WLSMV"`) and for the robust maximum likelihood estimators (`"MLM"`, `"MLMV"`, `"MLMVS"`, `"MLR"`; see `setestimator`), the printed measures additionally include the scaled test statistic (`chisq.scaled`, `chisq.scaling.factor`, `df.scaled`, and `chisq.shift.parameters` for the scaled-and-shifted variant) and, for robust ML, the robust fit indices (`rmsea.robust`, `cfi.robust`, `tli.robust`).

Usage

```
fit(x)
```

Arguments

`x`	A `psychonetrics` model.

Value

Invisibly returns a data frame with fit measure estimates.

Author(s)

Sacha Epskamp

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit an empty GGM:
mod0 <- ggm(ConsData, vars = vars, omega = "zero")

# Run model:
mod0 <- mod0 %>% runmodel

# Inspect fit:
mod0 %>% fit # Pretty bad fit...
```

---

fixpar *Parameters modification*

Description

The `fixpar` function can be used to fix a parameter to some value (Typically zero), and the `freepar` function can be used to free a parameter from being fixed to a value.

Usage

```
fixpar(x, matrix, row, col, value = 0, group, verbose,
        log = TRUE, runmodel = FALSE, ...)

freepar(x, matrix, row, col, start, group, verbose, log =
        TRUE, runmodel = FALSE, startEPC = TRUE, ...)
```

Arguments

x	A psychonetrics model.
matrix	String indicating the matrix of the parameter
row	Integer or string indicating the row of the matrix of the parameter
col	Integer or string indicating the column of the matrix of the parameter
value	Used in fixpar to indicate the value to which a parameters is constrained
start	Used in freepar to indicate the starting value of the parameter
group	Integer indicating the group of the parameter to be constrained
verbose	Logical, should messages be printed?
log	Logical, should the log be updated?
runmodel	Logical, should the model be updated?
startEPC	Logical, should the starting value be set at the expected parameter change?
...	Arguments sent to runmodel

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

---

fixstart *Attempt to Fix Starting Values*

Description

This function attempts to fix starting values by comparing the analytic gradient to a numerically approximated gradient. Parameters with a difference between the analytic and numeric gradient that exceeds 'maxdiff' will be reduced by a factor of 'reduce' in each iteration until the average absolute difference between analytic and numeric gradients is lower than 'tol'. Only off-diagonal elements in omega, sigma, kappa, lowertri or rho matrices or any element in beta matrices are adjusted.

Usage

```
fixstart(x, reduce = 0.5, maxdiff = 0.1, tol = 0.01, maxtry = 25)
```

Arguments

x	A 'psychonetrics' model
reduce	The factor with which problematic parameters are reduced in each iteration.
maxdiff	Maximum difference between analytic and numeric gradient to be considered problematic.
tol	Average absolute difference between analytic and numeric gradient that is considered acceptable.
maxtry	Maximum number of iterations to attempt to fix starting values.

Value

An object of the class psychonetrics (psychonetrics-class) with adjusted starting values.

Author(s)

Sacha Epskamp

---

fromlavaan *Convert a lavaan model to a psychonetrics lvm*

Description

Converts a lavaan model into an equivalent psychonetrics latent variable model (lvm) with latent = "cov" and residual = "cov". The input can be a fitted lavaan object (e.g., from lavaan::cfa or lavaan::sem), or lavaan model syntax together with a data frame. Only the public lavaan API is used; no lavaan source code is reused.

Usage

```
fromlavaan(x, data = NULL, run = FALSE, estimator = "default",
           baseline_saturated = TRUE, verbose = FALSE, ...)
```

Arguments

x	A fitted lavaan object, or a character string with lavaan model syntax. When syntax is supplied, the model is set up internally with lavaan::sem(model = x, data = data, meanstructure = TRUE, fixed.x = FALSE, do.fit = FALSE).
data	A data frame. Required when x is model syntax; ignored when x is a fitted lavaan object.
run	Logical. If TRUE, the resulting psychonetrics model is estimated with runmodel before being returned.
estimator	The estimator passed to lvm. The default ("default") uses "ML", or "FIML" when lavaan was fitted with full-information maximum likelihood for missing data.

`baseline_saturated`
: Logical, passed to `lvm`; should the baseline and saturated models be included? Mostly used internally.

`verbose`
: Logical. If `TRUE`, an experimental-feature note is shown and progress is printed.

`...`
: Further arguments passed to `lvm`.

Details

The converted model reproduces the lavaan specification using the LISREL-style model matrices that lavaan exposes through `lavInspect(fit, "free")` and `lavInspect(fit, "est")`: factor loadings (`lambda`), the latent variance–covariance matrix (`sigma_zeta`, lavaan's psi), the residual variance–covariance matrix (`sigma_epsilon`, lavaan's theta), structural regressions (`beta`), observed intercepts (`nu`) and latent intercepts (`nu_eta`, lavaan's alpha). Observed variables that appear as structural (regression) variables are carried as phantom latent columns; any latent whose name clashes with an observed variable is renamed to `eta_<name>`. Equality constraints are detected via shared parameter labels and simple == rows and mapped onto shared psychonetrics parameter (`par`) indices.

The following are guaranteed to match lavaan (for a standard ML fit, see the caveats below): the parameter estimates, the (expected-information) standard errors, the chi-square test statistic and degrees of freedom, the number of free parameters, and the log-likelihood.

Caveats. (1) If the lavaan model has no meanstructure, psychonetrics adds a saturated mean structure (free `nu`, `nu_eta` fixed to 0): estimates, standard errors, chi-square and df still match, but the log-likelihood differs by a constant; a warning is issued. (2) With `fixed.x = TRUE` and exogenous covariates, psychonetrics treats the covariates as random (endogenous): estimates, standard errors and chi-square match, but df-based fit measures and the log-likelihood do not (lavaan conditions on the covariates); a warning is issued. (3) If lavaan used `likelihood = "wishart"` (the $n - 1$ normalization), psychonetrics uses the normal (n) likelihood, so the log-likelihood differs; a warning is issued. (4) Robust standard errors and robust/scaled test statistics are not transferred; the model is converted as plain ML, with a warning. (5) For FIML, psychonetrics uses expected information for standard errors, whereas lavaan defaults to observed information, so FIML standard errors may differ slightly even though the log-likelihood matches.

Unsupported. The following lavaan features cannot be converted and raise an informative error: two-level / multilevel models (`cluster=`, `level:`), defined parameters (`:=`), inequality constraints (<, >), general (non-simple) equality constraints, formative indicators (<~), `conditional.x = TRUE`, and categorical (ordinal) endogenous variables.

Value

An object of class `psychonetrics` (see `psychonetrics-class`).

Author(s)

Sacha Epskamp

References

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

See Also

tolavaan, lvm

Examples

```
if (requireNamespace("lavaan", quietly = TRUE)){
  library("lavaan")

  # A confirmatory factor analysis fitted in lavaan:
  data(HolzingerSwineford1939)
  HS <- HolzingerSwineford1939
  fit <- cfa(' visual  =~ x1 + x2 + x3
               textual =~ x4 + x5 + x6
               speed   =~ x7 + x8 + x9 ',
             data = HS, meanstructure = TRUE)

  # Convert to a psychonetrics lvm and run it:
  mod <- fromlavaan(fit, run = TRUE)
  fit(mod)

  # The log-likelihood matches lavaan:
  c(lavaan = as.numeric(logLik(fit)), psychonetrics = mod@fitmeasures$logl)

  # Equivalent starting from syntax + data (without fitting in lavaan first):
  mod2 <- fromlavaan(' visual  =~ x1 + x2 + x3
                       textual =~ x4 + x5 + x6
                       speed   =~ x7 + x8 + x9 ',
                     data = HS, run = TRUE)
}
```

---

generate *Generate data from a fitted psychonetrics object*

Description

This function will generate new data from the estimated mean and variance-covariance structure of a psychonetrics model.

Usage

```
generate(x, n = 500)
```

Arguments

x	A psychonetrics model.
n	Number of cases to sample per group.

Value

A data frame with simulated data

Author(s)

Sacha Epskamp

`getmatrix` *Extract an estimated matrix*

Description

This function will extract an estimated matrix, and will either return a single matrix for single group models or a list of such matrices for multiple group models.

Usage

```
getmatrix(x, matrix, group, threshold = FALSE, alpha = 0.01,
          adjust = c("none", "holm", "hochberg", "hommel",
          "bonferroni", "BH", "BY", "fdr"), mode = c("tested",
          "all"), diag = TRUE, time)
```

Arguments

`x`	A `psychonetrics` model.
`matrix`	String indicating the matrix to be extracted. For `ri_clpm` / `ri_clpn` models this may also be a block name such as `"omega_zeta_within"` or `"omega_zeta_between"`; see Details.
`group`	Integer indicating the group for the matrix to be extracted.
`threshold`	Logical. Should the matrix be thresholded (non-significant values set to zero? Can also be a value with an absolute threshold below which parameters are set to zero.)
`alpha`	Significance level to use.
`adjust`	p-value adjustment method to use. See `p.adjust`.
`mode`	Mode for adjusting for multiple comparisons. Should all parameters be considered as the total number of tests or only the tested parameters (parameters of interest)?
`diag`	Set to FALSE to set diagonal elements to zero.
`time`	Only for `ri_clpm` / `ri_clpn` models: the wave whose block should be returned, either an integer or a column name of the design matrix. See Details.

Details

RI-CLPM / RI-CLPN blocks. A model made with `ri_clpm` or `ri_clpn` is built on the `lvm` framework, so its model matrices span all waves at once: the m within-person components of each of the T waves, followed by the m random intercepts. The quantities of interest are the $m \times m$ blocks of those matrices, which can be addressed directly with the same matrix names that `panelvar` and `panelgvar` use, together with the `time` argument:

- `getmatrix(x, "beta", time = t)` – the temporal (cross-lagged) network of the transition from wave t to wave $t + 1$, with rows the outcome and columns the predictor, exactly as in `panelvar`.
- `getmatrix(x, "PDC", time = t)` – the partial directed correlations of that transition (rows = from, columns = to), the form that feeds directly into `qgraph`. The RI-CLPM has no PDC parameters of its own, so these are derived from the temporal block and the innovation precision of the target wave $t + 1$; for a fully stationary model this reproduces the `panelvar` PDC exactly.
- `getmatrix(x, "omega_zeta_within", time = t)` – the contemporaneous network of wave t. Note that at `time = 1` this is the *exogenous* wave-1 block, which the RI-CLPM leaves free, and not an innovation structure. The other parameterizations (`sigma_`, `kappa_`, `lowertri_`, `rho_`, `SD_`, `delta_zeta_within`) work the same way, and the plain name plus `time` (e.g. `"omega_zeta"`) is accepted as a shorthand.
- `getmatrix(x, "omega_zeta_between")` – the between-person (random intercept) network. No `time` is given or accepted, since the random intercepts do not vary over waves.
- `getmatrix(x, "nu", time = t)` (or `"nu_within"`) – the wave-specific intercepts, and `getmatrix(x, "nu_between")` – the random-intercept means that `ri_clpm_stationary(x, "intercepts")` frees.

Blocks are returned with the variable names of the design matrix as row and column names. Omitting `time` returns a named list with one block per wave; waves whose parameters were equated by `ri_clpm_stationary` are collapsed into one entry, so a stationary structure yields a single block rather than several identical copies. Supplying a plain matrix name without `time` (for example `getmatrix(x, "beta")`) returns the full latent-sized matrix as in earlier versions.

Thresholding is applied to the full matrix and the block is then cut out of the result, so a thresholded block is exactly the corresponding submatrix of the thresholded full matrix and the multiple-comparison adjustment is unaffected by which block is requested. `diag = FALSE`, in contrast, applies to the block, so that it removes the autoregressive effects of a temporal block (which do not lie on the diagonal of the full `beta` matrix).

Value

A matrix of parameter estimates, or a list of such matrices for multiple group models.

Author(s)

Sacha Epskamp

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars, omega = "full")

# Run model:
mod <- mod %>% runmodel

# Obtain network:
mod %>% getmatrix("omega")
```

---

getVCOV *Obtain the asymptotic covariance matrix*

Description

This function can be used to obtain the estimated asymptotic covariance matrix from a psychonetrics object.

Usage

```
getVCOV(model, approximate_SEs = FALSE)
```

Arguments

model	A psychonetrics model.
approximate_SEs	Logical, should standard errors be approximated? If true, an approximate matrix inverse of the Fisher information is used to obtain the standard errors.

Value

This function returns a matrix.

Author(s)

Sacha Epskamp

`groupequal` *Group equality constraints*

Description

The `groupequal` function constrains parameters equal across groups, and the `groupfree` function frees equality constraints across groups.

Usage

```
groupequal(x, matrix, row, col, verbose, log = TRUE, runmodel =
                    FALSE, identify = TRUE, ...)

groupfree(x, matrix, row, col, verbose, log = TRUE, runmodel =
                    FALSE, identify = TRUE, ...)
```

Arguments

`x`	A `psychonetrics` model.
`matrix`	String indicating the matrix of the parameter
`row`	Integer or string indicating the row of the matrix of the parameter
`col`	Integer or string indicating the column of the matrix of the parameter
`verbose`	Logical, should messages be printed?
`log`	Logical, should the log be updated?
`runmodel`	Logical, should the model be updated?
`identify`	Logical, should the model be identified?
`...`	Arguments sent to `runmodel`

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

`Ising` *Ising model*

Description

This is the family of Ising models fit to datasets with two or more ordered integer response options. The classical case is a dichotomous dataset (e.g., encoded with -1 and 1 or with 0 and 1), but the same parameterization—one threshold (`tau`) per variable and a pairwise network (`omega`)—is also used for any number of ordered response options (e.g., `c(-1,0,1)` or `seq(-5,5)`; the values need not be integers), encoded identically across all variables. Only the partition function changes: it sums over all `length(responses)^nNode` response patterns rather than only `2^nNode`. Note that the input matters (see also https://arxiv.org/abs/1811.02916) in this model! Models based on a dataset that is encoded with -1 and 1 are not entirely equivalent to models based on datasets encoded with 0 and 1 (non-equivalences occur in multi-group settings with equality constraints).

Usage

```
Ising(data, omega = "full", tau, beta, beta_model =
                 c("beta", "log_beta"), vars, groups, covs, means,
                 nobs, covtype = c("choose", "ML", "UB"), responses,
                 missing = "listwise", equal = "none",
                 baseline_saturated = TRUE, estimator = "default",
                 optimizer, storedata = FALSE, WLS.W, sampleStats,
                 identify = TRUE, verbose = FALSE, maxNodes = 20,
                 maxStates = 2^maxNodes, min_sum = -Inf,
                 bootstrap = FALSE, boot_sub,
                 boot_resample, penalty_lambda = NA,
                penalty_alpha = 1, penalize_matrices)
```

Arguments

`data`	A data frame encoding the data used in the analysis. Can be missing if `covs` and `nobs` are supplied.
`omega`	The network structure. Either `"full"` to estimate every element freely, `"zero"` to set all elements to zero, or a matrix of the dimensions nNode x nNode with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
`tau`	Optional vector encoding the threshold/intercept structure. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.
`beta`	Optional scalar encoding the inverse temperature. 1 indicate free beta parameters, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such scalars.

`beta_model` How should beta be modeled? Set `beta_model = "log_beta"` to model the log of beta rather than beta directly.

`vars` An optional character vector encoding the variables used in the analysis. Must equal names of the dataset in `data`.

`groups` An optional string indicating the name of the group variable in the data.

`covs` A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure `covtype` argument is set correctly to the type of covariances used.

`means` A vector of sample means, or a list/matrix containing such vectors for multiple groups.

`nobs` The number of observations used in `covs` and `means`, or a vector of such numbers of observations for multiple groups.

`covtype` If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to `"ML"` for maximum likelihood estimates (denominator n) and `"UB"` to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.

`responses` A vector of the response options used, encoded identically across all variables (e.g., `c(-1,1)`, `c(0,1)`, `c(-1,0,1)` or `seq(-5,5)`). Automatically detected from the data when not supplied; required when `covs` is used. Any number of distinct response options is allowed, and the values need not be integers.

`missing` How should missingness be handled when `data` is used. Only `"listwise"` (listwise deletion) is currently supported for the Ising model; other options (e.g. `"pairwise"`) are rejected at construction because they produce undefined sufficient statistics.

`equal` A character vector indicating which matrices should be constrained equal across groups.

`baseline_saturated`
A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

`estimator` The estimator to be used. Currently implemented are `"ML"` for (exact) maximum likelihood estimation and `"PML"` for penalized maximum likelihood estimation; no other estimators are supported for the Ising model.

`optimizer` The optimizer to be used. Can be one of `"nlminb"` (the default R `nlminb` function), `"ucminf"` (from the `optimr` package), `"nloptr_TNEWTON"` (preconditioned truncated Newton via `nloptr`), and `"LBFGS++"` (pure C++ L-BFGS-B). Defaults to `"nlminb"`.

`storedata` Logical, should the raw data be stored? Needed for bootstrapping (see `bootstrap`).

`WLS.W` Optional WLS weights matrix. CURRENTLY NOT USED.

`sampleStats` An optional sample statistics object. Mostly used internally.

`identify` Logical, should the model be identified?

`verbose` Logical, should messages be printed?

`maxNodes` The maximum number of nodes allowed in the analysis. Used to set the default of `maxStates` (`2^maxNodes`); for binary data this reproduces the historical node limit. It is not recommended to set this higher.

`maxStates` The maximum number of response patterns the exact ML estimator may enumerate. Exact ML estimation of the Ising model sums over every possible response pattern when computing the partition function, expected values and (expected) Hessian, so the cost grows as `length(responses)^nNode`. The function stops with an error when `length(responses)^nNode` exceeds `maxStates`. The default `2^maxNodes` reproduces the historical binary node limit while accounting for more than two response options. Raise this only if the computation is feasible (note that the number of states—and hence the run time—grows very quickly with both the number of nodes and the number of response options).

`min_sum` The minimum sum score that is artificially possible in the dataset. Defaults to -Inf. Set this only if you know a lower sum score is not possible in the data, for example due to selection bias.

`bootstrap` Should the data be bootstrapped? If `TRUE` the data are resampled and a bootstrap sample is created. These must be aggregated using `aggregate_bootstraps`! Can be `TRUE` or `FALSE`. Can also be `"nonparametric"` (which sets `boot_sub = 1` and `boot_resample = TRUE`) or `"case"` (which sets `boot_sub = 0.75` and `boot_resample = FALSE`).

`boot_sub` Proportion of cases to be subsampled (`round(boot_sub * N)`).

`boot_resample` Logical, should the bootstrap be with replacement (`TRUE`) or without replacement (`FALSE`)

`penalty_lambda` Numeric penalty strength for penalized ML estimation (PML/PFIML). `NA` (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See `find_penalized_lambda` and `penalize`.

`penalty_alpha` Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

`penalize_matrices`
Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.

Details

The Ising Model takes the following form:

$\Pr(\boldsymbol{Y} = \boldsymbol{y}) = \frac{\exp(-\beta H(\boldsymbol{y};\boldsymbol{\tau},\boldsymbol{\Omega}))}{Z(\boldsymbol{\tau},\boldsymbol{\Omega})}$

With Hamiltonian:

$H(\boldsymbol{y};\boldsymbol{\tau},\boldsymbol{\Omega}) = -\sum_{i=1}^{m} \tau_i y_i - \sum_{i=2}^{m} \sum_{j=1}^{i-1} \omega_{ij} y_i y_j.$

And Z representing the partition function or normalizing constant.

The responses y_i need not be dichotomous: they may take any number of ordered values (the same set for every variable, and not necessarily integers), supplied through `responses` or detected automatically from the data. The Hamiltonian above is unchanged; only the partition function Z (and the expected values and Hessian derived from it) sums over all `length(responses)^nNode` response patterns instead of only `2^nNode`. With two response options the model is the usual Ising model and reproduces the same estimates as before. With more than two response options the

model remains a maximum-entropy distribution matching the means (via `tau`) and the pairwise products (via `omega`); it does not model higher-order interactions or the within-variable second moments, and the enumeration cost grows steeply with the number of nodes and response options (see `maxStates`).

Value

An object of the class psychonetrics

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

Epskamp, S., Maris, G., Waldorp, L. J., & Borsboom, D. (2018). Network Psychometrics. In: Irwing, P., Hughes, D., & Booth, T. (Eds.), The Wiley Handbook of Psychometric Testing, 2 Volume Set: A Multidisciplinary Reference on Survey, Scale and Test Development. New York: Wiley.

See Also

`BlumeCapel` for the more general Blume-Capel model, which adds a quadratic node potential (`delta`) and reduces to the Ising model when all `delta` are fixed to zero.

Examples

```
library("dplyr")
data("Jonas")

# Variables to use:
vars <- names(Jonas)[1:10]

# Arranged groups to put unfamiliar group first (beta constrained to 1):
Jonas <- Jonas[order(Jonas$group),]

# Form saturated model:
model1 <- Ising(Jonas, vars = vars, groups = "group")

# Run model:
model1 <- model1 %>% runmodel
# Note: SEs are approximated automatically because there are zeroes
# in the crosstables of the "Knows Jonas" group, which makes the
# Fisher information matrix singular. A warning is shown when
# this fallback occurs.

# Prune-stepup to find a sparse model:
model1b <- model1 %>% prune(alpha = 0.05) %>%  stepup(alpha = 0.05)

# Equal networks:
model2 <- model1 %>% groupequal("omega") %>% runmodel

# Prune-stepup to find a sparse model:
```

```
model2b <- model2 %>% prune(alpha = 0.05) %>% stepup(mi = "mi_equal", alpha = 0.05)

# Equal thresholds:
model3 <- model2 %>% groupequal("tau") %>% runmodel

# Prune-stepup to find a sparse model:
model3b <- model3 %>% prune(alpha = 0.05) %>% stepup(mi = "mi_equal", alpha = 0.05)

# Equal beta:
model4 <- model3 %>% groupequal("beta") %>% runmodel

# Prune-stepup to find a sparse model:
model4b <- model4 %>% prune(alpha = 0.05) %>% stepup(mi = "mi_equal", alpha = 0.05)

# Compare all models:
compare(
  `1. all parameters free (dense)` = model1,
  `2. all parameters free (sparse)` = model1b,
  `3. equal networks (dense)` = model2,
  `4. equal networks (sparse)` = model2b,
  `5. equal networks and thresholds (dense)` = model3,
  `6. equal networks and thresholds (sparse)` = model3b,
  `7. all parameters equal (dense)` = model4,
  `8. all parameters equal (sparse)` = model4b
) %>% arrange(BIC)
```

`Jonas` *Jonas dataset*

Description

Responses of 10 attitude items towards a researcher named Jonas. Participants were shown three photos of Jonas with the text: "This is Jonas, a researcher from Germany who is now becoming a PhD in Psychology". Subsequently, the participants had to answer 10 yes / no questions starting with "I believe that Jonas...", as well as rate their familiarity with Jonas. The sample consists of people familiar with Jonas and not familiar with Jonas, and allows for testing Attitudinal Entropy Framework <doi:10.1080/1047840X.2018.1537246>.

Usage

```
data("Jonas")
```

Format

A data frame with 215 observations on the following 12 variables.

`scientist` ... is a good scientist

`jeans` ... Is a person that wears beautiful jeans

`cares` ... really cares about people like you

`economics` ... would solve our economic problems

`hardworking` ... is hardworking

`honest` ... is honest

`intouch` ... is in touch with ordinary people

`knowledgeable` ... is knowledgeable

`makeupmind` ... can't make up his mind

`getsthingsdone` ... gets things done

`familiar` Answers to the question "How familiar are you with Jonas?" (three responses possible)

`group` The question 'familiar' categorized in two groups ("Knows Jonas" and "Doesn't know Jonas")

Source

Data collected by the package author via social media, among people familiar and unfamiliar with the researcher Jonas.

Examples

```
data(Jonas)
```

`latentgrowth` *Latent growth curve model*

Description

Wrapper to `lvm` to specify a latent growth curve model.

Usage

```
latentgrowth(vars, time = seq_len(ncol(vars)) - 1, covariates =
                  character(0), covariates_as = c("regression",
                  "covariance"), ...)
```

Arguments

`vars`	Different from in other psychonetrics models, this must be a *matrix* with each row indicating a variable and each column indicating a measurement. The matrix must be filled with names of the variables in the dataset corresponding to variable i at wave j. NAs can be used to indicate missing waves. The rownames of this matrix will be used as variable names.
`time`	A vector with the encoding of each measurement (e.g., 0, 1, 2, 3).
`covariates`	A vector with strings indicating names of between-person covariate variables in the data
`covariates_as`	Should covariates be included as regressions or actual covariates?
`...`	Arguments sent to `lvm`

Details

See https://github.com/SachaEpskamp/SEM-code-examples/tree/master/Latent_growth_examples/psychonetrics for examples

Value

An object of the class psychonetrics (psychonetrics-class). See for an example https://github.com/SachaEpskamp/SEM-code-examples/tree/master/Latent_growth_examples/psychonetrics.

Author(s)

Sacha Epskamp

Examples

```
library("dplyr")

# Smoke data cov matrix, based on LISS data panel https://www.dataarchive.lissdata.nl
smoke <- structure(c(47.2361758611759, 43.5366809116809, 41.0057465682466,
                     43.5366809116809, 57.9789886039886, 47.6992521367521,
                     41.0057465682466,
                     47.6992521367521, 53.0669434731935), dim = c(3L, 3L),
                   dimnames = list(
                       c("smoke2008", "smoke2009", "smoke2010"), c("smoke2008",
                   "smoke2009", "smoke2010")))

# Design matrix:
design <- matrix(rownames(smoke),1,3)

# Form model:
mod <- latentgrowth(vars = design,
                covs = smoke, nobs = 352
)

## Not run:
# Run model:
mod <- mod %>% runmodel

# Evaluate fit:
mod %>% fit

# Look at parameters:
mod %>% parameters

## End(Not run)
```

logbook *Retrieve the psychonetrics logbook*

Description

This function can be used to retrieve the logbook of a 'psychonetrics' object.

Usage

```
logbook(x, log = TRUE)
```

Arguments

x	A 'psychonetrics' object.
log	Logical, should the entry that the logbook is accessed be added?

Value

A list of log entries (objects of class `psychonetrics_log`) stored in the model.

Author(s)

Sacha Epskamp

loop_psychonetrics *Parallel loop for psychonetrics*

Description

A convenience wrapper for running repeated psychonetrics analyses in parallel. Replaces the typical boilerplate of `makeCluster` / `clusterExport` / `parLapply` / `stopCluster` with a single function call. Particularly useful for bootstrapping, simulation studies, and cross-validation.

Variables referenced in the expression that exist in the calling environment are automatically exported to parallel workers, so explicit `clusterExport` calls are not needed in most cases.

Usage

```
loop_psychonetrics(expr, reps = 1000, nCores = 1,
                   export = character(0),
                   packages = c("psychonetrics", "dplyr"),
                   verbose = TRUE, envir = parent.frame(),
                   seed = NULL)
```

Arguments

expr	An expression (code block) to evaluate on each iteration. Typically enclosed in curly braces {...}. The expression should return a single object (e.g., a psychonetrics model).
reps	Number of replications (default: 1000).
nCores	Number of CPU cores to use. Set to 1 for sequential execution (default), or higher for parallel execution via makePSOCKcluster.
export	Character vector of additional variable names to export to parallel workers. Usually not needed because variables referenced in expr are auto-detected and exported.
packages	Character vector of packages to load on parallel workers (default: c("psychonetrics", "dplyr")).
verbose	Logical; if TRUE (default), shows a progress bar via pbapply.
envir	The environment in which to look up variables for auto-export and to evaluate the expression in sequential mode. Defaults to the calling environment.
seed	Optional integer seed for reproducible random number generation. In parallel mode this calls clusterSetRNGStream so that repeated runs with the same seed and number of cores give identical results; in sequential mode set.seed is used. The default NULL leaves the random number generator untouched (previous behavior).

Details

When nCores > 1, a PSOCK cluster is created and automatically cleaned up when the function exits (even on error). The specified packages are loaded on each worker, and variables (as well as functions defined in the calling environment) referenced in expr are automatically detected and exported.

The replication index i (running from 1 to reps) is visible inside expr, in both sequential and parallel mode.

For bootstrapping, use this function together with bootstrap = "nonparametric" in the model constructor and aggregate_bootstraps to summarize the results.

Value

A list of length reps, where each element is the result of evaluating expr once.

See Also

aggregate_bootstraps, bootstrap, CIplot

Examples

```
library(dplyr)
data(NA2020)

# Fit the original model:
```

```
model <- ggm(NA2020, estimator = "FIML") %>% runmodel

# Bootstrap with pruning (parallel, 100 reps):
bootstraps <- loop_psychonetrics({
  ggm(NA2020, bootstrap = "nonparametric",
      estimator = "FIML") %>%
    runmodel %>%
    prune(alpha = 0.05)
}, reps = 100, nCores = 2)

# Aggregate and inspect:
boot_agg <- aggregate_bootstraps(
  sample = model %>% prune(alpha = 0.05),
  bootstraps = bootstraps
)

# View results:
parameters(boot_agg)
CIplot(boot_agg, "omega", split0 = TRUE)
```

`lvm` *Continuous latent variable family of psychonetrics models*

Description

This is the family of models that models the data as a structural equation model (SEM), allowing the latent and residual variance-covariance matrices to be further modeled as networks. The `latent` and `residual` arguments can be used to define what latent and residual models are used respectively: `"cov"` (default) models a variance-covariance matrix directly, `"chol"` models a Cholesky decomposition, `"prec"` models a precision matrix, `"ggm"` models a Gaussian graphical model (Epskamp, Rhemtulla and Borsboom, 2017), and `"cor"` models a correlation matrix with separate standard deviations. The wrapper `lnm()` sets `latent = "ggm"` for the latent network model (LNM), the wrapper `rnm()` sets `residual = "ggm"` for the residual network model (RNM), and the wrapper `lrnm()` combines the LNM and RNM.

Usage

```
lvm(data, lambda, latent = c("cov", "chol", "prec",
                  "ggm", "cor"), residual = c("cov", "chol", "prec", "ggm",
                  "cor"),
                  sigma_zeta = "full", kappa_zeta = "full", omega_zeta =
                  "full", lowertri_zeta = "full", delta_zeta = "full",
                  sigma_epsilon = "diag", kappa_epsilon = "diag",
                  omega_epsilon = "zero", lowertri_epsilon = "diag",
                  delta_epsilon = "diag", beta = "zero", nu, nu_eta, tau,
                  identify = TRUE, identification = c("loadings",
                  "variance"), vars, ordered = character(0), latents,
                  groups, groupvar, covs, cors, means, nobs,
```

```
  missing = "auto", equal = "none",
  baseline_saturated = TRUE, estimator = "default",
  likelihood = c("normal", "wishart"),
  fixed_x = character(0),
  optimizer, storedata = FALSE, WLS.W, covtype =
c("choose", "ML", "UB"), corinput, standardize = c("none", "z",
  "quantile"), sampleStats, verbose = FALSE,
  simplelambdastart = FALSE, start = "default",
  bootstrap = FALSE,
  boot_sub, boot_resample, penalty_lambda = NA,
  penalty_alpha = 1, penalize_matrices,
  rho_zeta = "full", SD_zeta = "full",
  rho_epsilon = "zero", SD_epsilon = "diag",
  sampling_weights)

lnm(...)
rnm(...)
lrnm(...)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied.
lambda	A model matrix encoding the factor loading structure. Each row indicates an indicator and each column a latent. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. Can also be set to "full" to freely estimate all factor loadings, including all cross-loadings (this requires the latents to be named through the latents argument). For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
latent	The type of latent model used. See description.
residual	The type of residual model used. See description.
sigma_zeta	Only used when latent = "cov". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
kappa_zeta	Only used when latent = "prec". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
omega_zeta	Only used when latent = "ggm". Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate

element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_zeta` Only used when `latent = "chol"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_zeta` Only used when `latent = "ggm"`. Either `"diag"` or `"zero"`, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_epsilon` Only used when `residual = "cov"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_epsilon` Only used when `residual = "prec"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_epsilon` Only used when `residual = "ggm"`. Either `"full"` to estimate every element freely, `"zero"` to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_epsilon`
Only used when `residual = "chol"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_epsilon` Only used when `residual = "ggm"`. Either `"diag"` or `"zero"`, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`beta` A model matrix encoding the structural relations between latent variables. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and

higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`nu` Optional vector encoding the intercepts of the observed variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.

`nu_eta` Optional vector encoding the intercepts of the latent variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.

`tau` Optional matrix encoding the threshold parameters for ordinal variables. Each column corresponds to an observed variable and each row to a threshold. Use 0 for fixed elements, 1 for free elements, and higher integers for equality constraints. Typically set up automatically when `ordered` is specified.

`identify` Logical, should the model be automatically identified?

`identification` Type of identification used. `"loadings"` to fix the first factor loadings to 1, and `"variance"` to fix the diagonal of the latent variable model matrix (sigma_zeta, lowertri_zeta, delta_zeta or kappa_zeta) to 1.

`vars` An optional character vector encoding the variables used in the analysis. Must equal names of the dataset in `data`.

`ordered` Character vector of variable names to treat as ordinal, or `TRUE` to treat all variables as ordinal. When ordinal variables are specified, polychoric correlations and thresholds are estimated from the data and the model is fit using the Muthen (1984) three-stage estimator. The estimator defaults to `"DWLS"` for ordinal models; `"WLS"` and `"ULS"` are also supported. Identification is automatically set to `"variance"`.

`latents` An optional character vector with names of the latent variables.

`groups` Deprecated. Use `groupvar` instead. An optional string indicating the name of the group variable in `data`.

`groupvar` An optional string indicating the name of the group variable in `data`. Replaces the deprecated `groups` argument; if both are supplied, `groupvar` takes precedence with a warning.

`covs` A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure `covtype` argument is set correctly to the type of covariances used.

`cors` A sample correlation matrix, or a list/array of such matrices for multiple groups. When supplied, the matrix is treated as a covariance matrix with a warning (appropriate for standardized data). Requires `nobs`. Note: `corinput = TRUE` is not supported in `lvm`; use `varcov` with `type = "cor"` for correlation-based models.

`means` A vector of sample means, or a list/matrix containing such vectors for multiple groups.

`nobs` The number of observations used in `covs` and `means`, or a vector of such numbers of observations for multiple groups.

`corinput` Logical. Only supported for ordinal data in `lvm()`; defaults to `FALSE` for continuous data. Setting `corinput = TRUE` is not supported for `lvm()` and will produce an error; use `varcov` with `type = "cor"` instead.

`missing` How should missingness be handled in computing the sample covariances and number of observations when `data` is used. Can be `"auto"` (default) for automatic detection, `"listwise"` for listwise deletion, or `"pairwise"` for pairwise deletion. When `"auto"`, the function checks for missing data and switches ML to FIML, PML to PFIML, or defaults to listwise for LS estimators.

`equal` A character vector indicating which matrices should be constrained equal across groups.

`baseline_saturated`
A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

`estimator` The estimator to be used. Currently implemented are `"ML"` for maximum likelihood estimation, `"FIML"` for full-information maximum likelihood estimation, `"PML"` for penalized maximum likelihood estimation, `"PFIML"` for penalized full-information maximum likelihood estimation, `"ULS"` for unweighted least squares estimation, `"WLS"` for weighted least squares estimation, and `"DWLS"` for diagonally weighted least squares estimation. `"WLSMV"` is accepted as a synonym for `"DWLS"` (both use DWLS estimation with a mean-and-variance adjusted scaled test statistic). When `missing = "auto"` (default), `"ML"` is automatically switched to `"FIML"` and `"PML"` to `"PFIML"` if missing data is detected. Defaults to `"ML"` for continuous data and `"DWLS"` when ordinal variables are specified via `ordered`.

`likelihood` The Gaussian likelihood scaling for maximum-likelihood estimation. `"normal"` (the default) uses the n (biased) sample-covariance denominator; `"wishart"` uses the $n - 1$ (unbiased) sample covariance per group, so the chi-square uses $(n_g - 1)$ multipliers and the standard errors are inflated by $\sqrt{n_g/(n_g - 1)}$, matching `lavaan`'s `likelihood = "wishart"` (Rosseel, 2012). Only available for complete-data maximum likelihood (`estimator = "ML"`); it errors for `FIML`, least-squares and ordinal estimators and for raw time-series input. Default `"normal"` reproduces previous behaviour exactly.

`fixed_x` Character vector of exogenous *latent* variable names whose means and mutual (co)variances are fixed to their sample values and excluded from the free-parameter count and the degrees-of-freedom statistic count, matching `lavaan`'s `sem(..., fixed.x = TRUE)` (Rosseel, 2012). This is the way to enter observed exogenous covariates into a **psychonetrics** SEM: model each observed covariate as a single-indicator latent (loading 1, residual variance 0) and pass that latent's name. Fixing conditions the model on these covariates and relaxes the multivariate-normality assumption on them; regression paths from the fixed-x latents ($x \rightarrow \eta$) remain free. Only supported for complete-data `estimator = "ML"`. The reported log-likelihood is the conditional log-likelihood. Default `character(0)` (behaviour unchanged). The baseline and saturated reference models are conditioned on the exogenous block as well, so the incremental fit indices (CFI/TLI) match `lavaan`. *Important*: unlike `lavaan` (which sets `auto.cov.lv.y = TRUE`), **psychonetrics** does *not* free the residual covariances among endogenous latent variables by default. To reproduce a `lavaan` SEM

with several endogenous factors, free those covariances explicitly through the sigma_zeta adjacency matrix (set the corresponding off-diagonal entries to 1); the fixed-x results then match lavaan to numerical precision (estimates, SEs, chi-square, df and all fit measures).

sampling_weights

Optional single column name in data giving (survey) SAMPLING weights, enabling pseudo-maximum-likelihood estimation. The moments are weighted and the model is fit with robust (MLR) Huber-White sandwich standard errors and a Yuan-Bentler-Mplus scaled test, matching lavaan's sampling.weights= (weights normalized so their total over all groups equals the sample size). Estimates, standard errors, the scaled chi-square and robust fit indices agree with lavaan to numerical precision, single- and multi-group. Requires complete-data continuous input; not supported for ordinal data, missing data (FIML), or likelihood = "wishart". References: Asparouhov (2005); Rosseel (2012).

optimizer — The optimizer to be used. Can be one of "nlminb" (the default R nlminb function), "ucminf" (from the optimr package), "nloptr_TNEWTON" (preconditioned truncated Newton via nloptr), and "LBFGS++" (pure C++ L-BFGS-B). Defaults to "nlminb".

storedata — Logical, should the raw data be stored? Needed for bootstrapping (see bootstrap).

verbose — Logical, should progress be printed to the console?

WLS.W — The weights matrix used in WLS estimation (experimental)

sampleStats — An optional sample statistics object. Mostly used internally.

covtype — If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to "ML" for maximum likelihood estimates (denominator n) and "UB" to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.

standardize — Which standardization method should be used? "none" (default) for no standardization, "z" for z-scores, and "quantile" for a non-parametric transformation to the quantiles of the marginal standard normal distribution.

simplelambdastart

Logical, should simple start values be used for lambda? Setting this to TRUE can avoid some estimation problems.

start — Controls the starting values for the beta (structural regression) matrix. Can be "default" (default) for OLS-based starting values derived from the factor-analysis-implied latent covariance matrix, "simple" for the previous behavior (all free elements set to 0.001), or a fitted psychonetrics object to extract beta estimates as starting values. The OLS-based starts compute regression coefficients between latent variables using the FA-derived covariance matrix, which typically leads to faster convergence for models with structural paths. Guard rails automatically fall back to simple starts if numerical issues are detected (e.g., near-singular matrices, extreme coefficients).

bootstrap — Should the data be bootstrapped? If TRUE the data are resampled and a bootstrap sample is created. These must be aggregated using aggregate_bootstraps! Can be TRUE or FALSE. Can also be "nonparametric" (which sets boot_sub

= 1 and boot_resample = TRUE) or "case" (which sets boot_sub = 0.75 and boot_resample = FALSE).

boot_sub — Proportion of cases to be subsampled (round(boot_sub * N)).

boot_resample — Logical, should the bootstrap be with replacement (TRUE) or without replacement (FALSE)

penalty_lambda — Numeric penalty strength for penalized ML estimation (PML/PFIML). NA (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See find_penalized_lambda and penalize.

penalty_alpha — Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

penalize_matrices — Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.

rho_zeta — Only used when latent = "cor". Either "full" or "zero", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the latent correlations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

SD_zeta — Only used when latent = "cor". Either "full" or "diag", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the diagonal matrix of latent standard deviations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

rho_epsilon — Only used when residual = "cor". Either "full" or "zero", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the residual correlations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

SD_epsilon — Only used when residual = "cor". Either "full" or "diag", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the diagonal matrix of residual standard deviations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

... — Arguments sent to lvm

Details

The model used in this family is:

$\mathrm{var}(\boldsymbol{y}) = \boldsymbol{\Lambda}(\boldsymbol{I} - \boldsymbol{B})^{-1}\boldsymbol{\Sigma}_\zeta(\boldsymbol{I} - \boldsymbol{B})^{-1\top}\boldsymbol{\Lambda}^\top + \boldsymbol{\Sigma}_\varepsilon$

$\mathcal{E}(\boldsymbol{y}) = \boldsymbol{\nu} + \boldsymbol{\Lambda}(\boldsymbol{I} - \boldsymbol{B})^{-1}\boldsymbol{\nu}_e ta$

in which the latent covariance matrix can further be modeled in three ways. With latent = "chol" as Cholesky decomposition:

$\boldsymbol{\Sigma}_\zeta = \boldsymbol{L}_\zeta \boldsymbol{L}_\zeta^\top,$

with `latent = "prec"` as Precision matrix:

$\boldsymbol{\Sigma}_\zeta = \boldsymbol{K}_\zeta^{-1}$,

and finally with `latent = "ggm"` as Gaussian graphical model:

$\boldsymbol{\Sigma}_\zeta = \boldsymbol{\Delta}_\zeta(\boldsymbol{I} - \boldsymbol{\Omega}_\zeta)^(-1)\boldsymbol{\Delta}_\zeta$.

Likewise, the residual covariance matrix can also further be modeled in three ways. With `residual = "chol"` as Cholesky decomposition:

$\boldsymbol{\Sigma}_\varepsilon = \boldsymbol{L}_\varepsilon \boldsymbol{L}_\varepsilon^\top$,

with `residual = "prec"` as Precision matrix:

$\boldsymbol{\Sigma}_\varepsilon = \boldsymbol{K}_\varepsilon^{-1}$,

and finally with `residual = "ggm"` as Gaussian graphical model:

$\boldsymbol{\Sigma}_\varepsilon = \boldsymbol{\Delta}_\varepsilon(\boldsymbol{I} - \boldsymbol{\Omega}_\varepsilon)^(-1)\boldsymbol{\Delta}_\varepsilon$.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

References

Epskamp, S., Rhemtulla, M., & Borsboom, D. (2017). Generalized network psychometrics: Combining network and latent variable models. Psychometrika, 82(4), 904-927.

Muthen, B. (1984). A general structural equation model with dichotomous, ordered categorical, and continuous latent variable indicators. Psychometrika, 49(1), 115-132.

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

Examples

```
library("dplyr")

### Confirmatory Factor Analysis ###

# Example also shown in https://youtu.be/Hdu5z-fwuk8

# Load data:
data(StarWars)

# Originals only:
Lambda <- matrix(1,4)

# Model:
mod0 <- lvm(StarWars, lambda = Lambda, vars = c("Q1","Q5","Q6","Q7"),
            identification = "variance", latents = "Originals")
```

```
# Run model:
mod0 <- mod0 %>% runmodel

# Evaluate fit:
mod0 %>% fit


# Full analysis
# Factor loadings matrix:
Lambda <- matrix(0, 10, 3)
Lambda[1:4,1] <- 1
Lambda[c(1,5:7),2] <- 1
Lambda[c(1,8:10),3] <- 1

# Observed variables:
obsvars <- paste0("Q",1:10)

# Latents:
latents <- c("Prequels","Original","Sequels")

# Make model:
mod1 <- lvm(StarWars, lambda = Lambda, vars = obsvars,
            identification = "variance", latents = latents)

# Run model:
mod1 <- mod1 %>% runmodel

# Look at fit:
mod1

# Look at parameter estimates:
mod1 %>% parameters

# Completely standardized (std.all) solution with standard errors:
mod1 %>% parameters(standardized = TRUE)

# Look at modification indices:
mod1 %>% MIs

# Add and refit:
mod2 <- mod1 %>% freepar("sigma_epsilon","Q10","Q4") %>% runmodel

# Compare:
compare(original = mod1, adjusted = mod2)

# Fit measures:
mod2 %>% fit

### Path diagrams ###
# semPlot is not (yet) supported by default, but can be used as follows:
# Load packages:
if (requireNamespace("semPlot", quietly = TRUE)){
library("semPlot")
```

```
# Estimates:
lambdaEst <- getmatrix(mod2, "lambda")
psiEst <- getmatrix(mod2, "sigma_zeta")
thetaEst <- getmatrix(mod2, "sigma_epsilon")

# LISREL Model: LY = Lambda (lambda-y), TE = Theta (theta-epsilon), PS = Psi
mod <- lisrelModel(LY =  lambdaEst, PS = psiEst, TE = thetaEst)

# Plot with semPlot:
semPaths(mod, "std", "est", as.expression = "nodes")


# We can make this nicer (set whatLabels = "none" to hide labels):
semPaths(mod,

# this argument controls what the color of edges represent. In this case,
# standardized parameters:
    what = "std",

# This argument controls what the edge labels represent. In this case, parameter
# estimates:
    whatLabels = "est",

# This argument draws the node and edge labels as mathematical expressions:
    as.expression = "nodes",

# This will plot residuals as arrows, closer to what we use in class:
    style = "lisrel",

# This makes the residuals larger:
    residScale = 10,

# qgraph colorblind friendly theme:
    theme = "colorblind",

# tree layout options are "tree", "tree2", and "tree3":
    layout = "tree2",

# This makes the latent covariances connect at a cardinal center point:
    cardinal = "lat cov",

# Changes curve into rounded straight lines:
    curvePivot = TRUE,

# Size of manifest variables:
    sizeMan = 4,

# Size of latent variables:
    sizeLat = 10,

# Size of edge labels:
    edge.label.cex = 1,
```

```
# Sets the margins:
    mar = c(9,1,8,1),

# Prevents re-ordering of observed variables:
    reorder = FALSE,

# Width of the plot:
    width = 8,

# Height of plot:
    height = 5,

# Colors according to latents:
    groups = "latents",

# Pastel colors:
    pastel = TRUE,

# Disable borders:
    borders = FALSE
    )

# Use arguments filetype = "pdf" and filename = "semPlotExample1" to store PDF
} # end semPlot block

### Latent Network Modeling ###

# Latent network model:
lnm <- lvm(StarWars, lambda = Lambda, vars = obsvars,
           latents = latents, identification = "variance",
           latent = "ggm")

# Run model:
lnm <- lnm %>% runmodel

# Look at parameters:
lnm %>% parameters

# Remove non-sig latent edge:
lnm <- lnm %>% prune(alpha = 0.05)

# Compare to the original CFA model:
compare(cfa = mod1, lnm = lnm)

# Plot network:
library("qgraph")
qgraph(lnm@modelmatrices[[1]]$omega_zeta, labels = latents,
       theme = "colorblind", vsize = 10)

# A wrapper for the latent network model is the lnm function:
lnm2 <- lnm(StarWars, lambda = Lambda, vars = obsvars,
            latents = latents, identification = "variance")
```

```
lnm2 <- lnm2 %>% runmodel %>% prune(alpha = 0.05)
compare(lnm, lnm2) # Is the same as the model before.

# I could also estimate a "residual network model", which adds partial correlations to
# the residual level:
# This can be done using lvm(..., residual = "ggm") or with rnm(...)
rnm <- rnm(StarWars, lambda = Lambda, vars = obsvars,
           latents = latents, identification = "variance")
# Stepup search:
rnm <- rnm %>% stepup

# It will estimate the same model (with link Q10 - Q4) as above. In the case of only one
# partial correlation, There is no difference between residual covariances (SEM) or
# residual partial correlations (RNM).


# For more information on latent and residual network models, see:
# Epskamp, S., Rhemtulla, M.T., & Borsboom, D. Generalized Network Psychometrics:
# Combining Network and Latent Variable Models
# (2017). Psychometrika. doi:10.1007/s11336-017-9557-x

### Gaussian graphical models ###

# All psychonetrics functions (e.g., lvm, lnm, rnm...) allow input via a covariance
# matrix, with the "covs" and "nobs" arguments.
# The following fits a baseline GGM network with no edges:
S <- (nrow(StarWars) - 1)/ (nrow(StarWars)) * cov(StarWars[,1:10])
ggmmod <- ggm(covs = S, nobs = nrow(StarWars))

# Run model with stepup search and pruning:
ggmmod <- ggmmod%>% prune  %>% modelsearch

# Fit measures:
ggmmod %>% fit

# Plot network:
nodeNames <- c(
"I am a huge Star Wars\nfan! (star what?)",
"I would trust this person\nwith my democracy.",
"I enjoyed the story of\nAnakin's early life.",
"The special effects in\nthis scene are awful (Battle of\nGeonosis).",
"I would trust this person\nwith my life.",
"I found Darth Vader's big\nreveal in 'Empire' one of the greatest
moments in movie history.",
"The special effects in\nthis scene are amazing (Death Star\nExplosion).",
"If possible, I would\ndefinitely buy this\ndroid.",
"The story in the Star\nWars sequels is an improvement to\nthe previous movies.",
"The special effects in\nthis scene are marvellous (Starkiller\nBase Firing)."
)
library("qgraph")
qgraph(as.matrix(ggmmod@modelmatrices[[1]]$omega), nodeNames = nodeNames,
    legend.cex = 0.25,  theme = "colorblind", layout = "spring")
```

```
# We can actually compare this model statistically (note they are not nested) to the
# latent variable model:
compare(original_cfa = mod1, adjusted_cfa = mod2, exploratory_ggm = ggmmod)


### Measurement invariance ###
# Let's say we are interested in seeing if people >= 30 like the original trilogy better
# than people < 30.
# First we can make a grouping variable:
StarWars$agegroup <- ifelse(StarWars$Q12 < 30, "young", "less young")

# Let's look at the distribution:
table(StarWars$agegroup) # Pretty even...

# Observed variables:
obsvars <- paste0("Q",1:10)

# Let's look at the mean scores:
StarWars %>% group_by(agegroup) %>% summarize(across(all_of(obsvars), mean))
# Less young people seem to score higher on prequel questions and lower on other
# questions

# Factor loadings matrix:
Lambda <- matrix(0, 10, 3)
Lambda[1:4,1] <- 1
Lambda[c(1,5:7),2] <- 1
Lambda[c(1,8:10),3] <- 1

# Residual covariances:
Theta <- diag(1, 10)
Theta[4,10] <- Theta[10,4] <- 1

# Latents:
latents <- c("Prequels","Original","Sequels")

# Make model:
mod_configural <- lvm(StarWars, lambda = Lambda, vars = obsvars,
            latents = latents, sigma_epsilon = Theta,
            identification = "variance",
            groups =  "agegroup")

# Run model:
mod_configural <- mod_configural %>% runmodel

# Look at fit:
mod_configural
mod_configural %>% fit

# Looks good, let's try weak invariance:
mod_weak <- mod_configural %>% groupequal("lambda") %>% runmodel

# Compare models:
compare(configural = mod_configural, weak = mod_weak)
```

```
# weak invariance can be accepted, let's try strong:
mod_strong <- mod_weak %>% groupequal("nu") %>% runmodel
# Means are automatically identified

# Compare models:
compare(configural = mod_configural, weak = mod_weak, strong = mod_strong)

# Questionable p-value and AIC difference, but ok BIC difference. This is quite good, but
# let's take a look. I have not yet implemented LM tests for equality constraints, but we
# can look at something called "equality-free" MIs:
mod_strong %>% MIs(matrices = "nu", type = "free")

# Indicates that Q10 would improve fit. We can also look at residuals:
residuals(mod_strong)

# Let's try freeing intercept 10:
mod_strong_partial <- mod_strong %>% groupfree("nu",10) %>% runmodel

# Compare all models:
compare(configural = mod_configural,
        weak = mod_weak,
        strong = mod_strong,
        strong_partial = mod_strong_partial)

# This seems worth it and lead to an acceptable model! It seems that older people find
# the latest special effects more marvellous!
mod_strong_partial %>% getmatrix("nu")

# Now let's investigate strict invariance:
mod_strict <- mod_strong_partial %>% groupequal("sigma_epsilon") %>% runmodel

# Compare all models:
compare(configural = mod_configural,
        weak = mod_weak,
        strong_partial = mod_strong_partial,
        strict = mod_strict)
# Strict invariance can be accepted!

#  Now we can test for homogeneity!
# Are the latent variances equal?
mod_eqvar <- mod_strict %>% groupequal("sigma_zeta") %>% runmodel

# Compare:
compare(strict = mod_strict, eqvar = mod_eqvar)

# This is acceptable. What about the means? (alpha = nu_eta)
mod_eqmeans <- mod_eqvar %>% groupequal("nu_eta") %>% runmodel

# Compare:
compare(eqvar = mod_eqvar, eqmeans = mod_eqmeans)

# Rejected! We could look at MIs again:
```

```
mod_eqmeans %>% MIs(matrices = "nu_eta", type = "free")

# Indicates the strongest effect for prequels. Let's see what happens:
eqmeans2 <- mod_eqvar %>%
  groupequal("nu_eta",row = c("Original","Sequels")) %>% runmodel

# Compare:
compare(eqvar = mod_eqvar, eqmeans = eqmeans2)
# Questionable, what about the sequels as well?

eqmeans3 <- mod_eqvar %>% groupequal("nu_eta", row = "Original") %>% runmodel

# Compare:
compare(eqvar = mod_eqvar, eqmeans = eqmeans3)

# Still questionable.. Let's look at the mean differences:
mod_eqvar %>% getmatrix("nu_eta")

# Looks like people over 30 like the prequels better and the other two trilogies less!
```

`meta_lvm` *Meta-analytic latent variable model*

Description

Single-stage meta-analytic latent variable model (LVM) for covariance or correlation matrices. Combines a latent variable model (CFA/SEM) for the mean structure with a random-effects model for between-study heterogeneity. Based on the MAGNA framework (Epskamp, Isvoranu, & Cheung, 2022).

Usage

```
meta_lvm(covs, nobs, data, cors, studyvar, groups, groupvar,
                   corinput, Vmats, Vmethod = c("individual", "pooled",
                   "metaSEM_individual", "metaSEM_weighted"), Vestimation
                   = c("averaged", "per_study"),
                   lambda, beta = "zero",
                   latent = c("cov", "chol", "prec", "ggm", "cor"),
                   sigma_zeta = "full", kappa_zeta = "full",
                   omega_zeta = "full", lowertri_zeta = "full",
                   delta_zeta = "full",
                   rho_zeta = "full", SD_zeta = "full",
                   residual = c("cov", "chol", "prec", "ggm", "cor"),
                   sigma_epsilon = "diag", kappa_epsilon = "diag",
                   omega_epsilon = "zero", lowertri_epsilon = "diag",
                   delta_epsilon = "diag",
                   rho_epsilon = "zero", SD_epsilon = "diag",
                   identify = TRUE,
```

```
identification = c("loadings", "variance"),
randomEffects = c("chol", "cov", "prec", "ggm", "cor"),
sigma_randomEffects = "full",
kappa_randomEffects = "full",
omega_randomEffects = "full",
lowertri_randomEffects = "full",
delta_randomEffects = "full",
rho_randomEffects = "full",
SD_randomEffects = "full",
vars, latents,
baseline_saturated = TRUE, optimizer,
estimator = c("FIML", "ML"),
sampleStats, verbose = FALSE,
bootstrap = FALSE, boot_sub, boot_resample)
```

Arguments

covs	A list of covariance or correlation matrices. Must contain rows and columns with NAs for variables not included in a study.
nobs	A vector with the number of observations per study.
data	Optional data frame. When supplied together with studyvar, covariance matrices and sample sizes are computed internally per study. Cannot be used together with covs, cors, or nobs.
cors	A list of correlation matrices. When supplied, the matrices are treated as covariance matrices with a warning (appropriate for standardized data). Requires nobs.
studyvar	A string indicating the column name in data that identifies the study. Required when data is supplied.
groups	Deprecated. Use groupvar instead. Multi-group support is not yet included for meta-analytic models.
groupvar	Not yet supported for meta-analytic models. Supplying this argument will produce an error.
corinput	Logical. Defaults to FALSE. Controls whether the input is treated as correlation matrices.
Vmats	Optional list with 'V' matrices (sampling error variance approximations).
Vmethod	Which method should be used to approximate the sampling error variance? The "metaSEM_individual" and "metaSEM_weighted" options compute the sampling-error covariances via metaSEM::asyCov and therefore require the Suggests package metaSEM to be installed.
Vestimation	How should the sampling error estimates be evaluated?
lambda	A matrix encoding the factor loading structure. Can be a pattern matrix with TRUE/FALSE or 0/1 entries, or a matrix with starting values and higher integer codes for equality constraints.
beta	Structural (regression) matrix among latent variables. Defaults to "zero".

`latent` Parameterization of the latent covariance structure. One of `"cov"`, `"chol"`, `"prec"`, `"ggm"`, or `"cor"`.

`sigma_zeta` Latent covariance matrix specification (used when `latent = "cov"`).

`kappa_zeta` Latent precision matrix specification (used when `latent = "prec"`).

`omega_zeta` Latent partial correlation matrix specification (used when `latent = "ggm"`).

`lowertri_zeta` Latent Cholesky factor specification (used when `latent = "chol"`).

`delta_zeta` Latent scaling matrix specification (used when `latent = "ggm"`).

`residual` Parameterization of the residual covariance structure. One of `"cov"`, `"chol"`, `"prec"`, `"ggm"`, or `"cor"`.

`sigma_epsilon` Residual covariance matrix specification (used when `residual = "cov"`). Defaults to `"diag"`. Diagonal elements are always fixed (derived from the correlation constraint).

`kappa_epsilon` Residual precision matrix specification (used when `residual = "prec"`).

`omega_epsilon` Residual partial correlation matrix specification (used when `residual = "ggm"`).

`lowertri_epsilon` Residual Cholesky factor specification (used when `residual = "chol"`).

`delta_epsilon` Residual scaling matrix specification (used when `residual = "ggm"`).

`identify` Logical, should the model be identified? Defaults to `TRUE`.

`identification` How to identify the model. `"loadings"` or `"variance"`.

`randomEffects` Parameterization of the random effects covariance structure.

`sigma_randomEffects` Random effects covariance matrix specification (used when `randomEffects = "cov"`).

`kappa_randomEffects` Random effects precision matrix specification (used when `randomEffects = "prec"`).

`omega_randomEffects` Random effects partial correlation matrix specification (used when `randomEffects = "ggm"`).

`lowertri_randomEffects` Random effects Cholesky factor specification (used when `randomEffects = "chol"`).

`delta_randomEffects` Random effects scaling matrix specification (used when `randomEffects = "ggm"`).

`rho_randomEffects` Random effects correlation matrix specification (used when `randomEffects = "cor"`).

`SD_randomEffects` Random effects standard deviation matrix specification (used when `randomEffects = "cor"`).

`vars` Character vector of observed variable names. If missing, names are taken from the correlation matrices.

`latents`	Character vector of latent variable names.
`baseline_saturated`	Logical indicating if baseline and saturated models should be included.
`optimizer`	The optimizer to be used. Defaults to `"nlminb"`.
`estimator`	The estimator to be used. `"ML"` or `"FIML"` (default).
`sampleStats`	Optional sample statistics object.
`verbose`	Logical, should progress be printed?
`bootstrap`	Should the data be bootstrapped?
`boot_sub`	Proportion of cases to subsample for bootstrap.
`boot_resample`	Logical, should bootstrap be with replacement?
`rho_zeta`	Only used when `latent = "cor"`. Latent correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).
`SD_zeta`	Only used when `latent = "cor"`. Diagonal matrix of latent standard deviations (`"full"`, `"diag"`, or a pattern matrix).
`rho_epsilon`	Only used when `residual = "cor"`. Residual correlation matrix specification (`"zero"`, `"full"`, or a pattern matrix).
`SD_epsilon`	Only used when `residual = "cor"`. Diagonal matrix of residual standard deviations (`"diag"`, `"full"`, or a pattern matrix).

Details

This function implements a single-stage meta-analytic latent variable model. The model specifies that the mean of the vectorized correlation matrices follows an LVM-implied correlation structure:

$$\mu = \text{vechs}(\Lambda(I - B)^{-1}\Sigma_\zeta(I - B)^{-\top}\Lambda' + \Sigma_\varepsilon)$$

where diagonal elements of Σ_ε are derived from the constraint that μ represents correlations (diagonal of implied covariance = 1).

Between-study heterogeneity is modeled as:

$$\Sigma = \Sigma^{(\text{ran})} + V$$

where V is the known sampling error covariance and $\Sigma^{(\text{ran})}$ captures true between-study variation.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

Epskamp, S., Isvoranu, A. M., & Cheung, M. W. L. (2022). Meta-Analytic Gaussian Network Aggregation. *Psychometrika*, 87(1), 12-46.

Jak, S., and Cheung, M. W. L. (2019). Meta-analytic structural equation modeling with moderating effects on SEM parameters. *Psychological Methods*.

See Also

meta_varcov, lvm

Examples

```
## Not run:
library("dplyr")

# Generate simulated data
set.seed(42)
nStudies <- 10
nvar <- 6
nlat <- 2

# True factor loading matrix
lambda_true <- matrix(0, nvar, nlat)
lambda_true[1:3, 1] <- c(0.7, 0.8, 0.6)
lambda_true[4:6, 2] <- c(0.7, 0.8, 0.6)

# Generate correlation matrices per study
cors <- list()
nobs <- sample(100:400, nStudies)
for (i in 1:nStudies) {
  sigma <- lambda_true %*% t(lambda_true) + diag(1 - rowSums(lambda_true^2))
  dat <- MASS::mvrnorm(nobs[i], rep(0, nvar), sigma)
  cors[[i]] <- cor(dat)
  colnames(cors[[i]]) <- rownames(cors[[i]]) <- paste0("V", 1:nvar)
}

# Fit meta-analytic CFA
mod <- meta_lvm(covs = cors, nobs = nobs, lambda = lambda_true != 0)
mod <- mod %>% runmodel

# Inspect results
print(mod)
parameters(mod)
getmatrix(mod, "lambda")

## End(Not run)
```

meta_var1 *Meta-analytic VAR(1) model*

Description

Single-stage meta-analytic vector autoregressive model (VAR(1)) for time-series data collected across multiple studies. Pools temporal (lag-1) and contemporaneous network structures across studies using a random-effects framework. The `meta_gvar` wrapper sets `contemporaneous = "ggm"` for graphical VAR estimation, and `meta_var()` is the primary short name of `meta_var1()` (identical function).

Usage

```
meta_var1(data, covs, nobs,
          vars, studyvar, idvar, dayvar, beepvar,
          contemporaneous = c("cov", "chol", "prec", "ggm", "cor"),
          beta = "full",
          omega_zeta = "full", delta_zeta = "full",
          kappa_zeta = "full", sigma_zeta = "full",
          lowertri_zeta = "full",
          rho_zeta = "full", SD_zeta = "full",
          randomEffects = c("chol", "cov", "prec", "ggm", "cor"),
          sigma_randomEffects = "full",
          kappa_randomEffects = "full",
          omega_randomEffects = "full",
          lowertri_randomEffects = "full",
          delta_randomEffects = "full",
          rho_randomEffects = "full",
          SD_randomEffects = "full",
          Vmats,
          Vmethod = c("individual", "pooled"),
          Vestimation = c("averaged", "per_study"),
          baseline_saturated = TRUE, optimizer,
          estimator = c("FIML", "ML"),
          sampleStats, verbose = FALSE,
          bootstrap = FALSE, boot_sub, boot_resample)

meta_var(...)

meta_gvar(...)
```

Arguments

`data`	A data frame containing time-series data for multiple studies. Use together with `studyvar` to identify studies.
`covs`	A list of pre-computed Toeplitz covariance matrices (one per study). Each matrix should have dimension `2p` x `2p` where `p` is the number of variables, with

the first p rows/columns being lagged variables and the second p being current variables. Alternative to data.

nobs — A vector with the number of observations per study. Required when covs is supplied.

vars — Character vector of observed variable names. If missing, inferred from data.

studyvar — A string indicating the column name in data that identifies the study. Required when data is supplied. If not supplied but idvar is, idvar is used as studyvar with a warning.

idvar — Optional string indicating the subject ID column within each study. When both studyvar and idvar are supplied, data is first split by studyvar, then idvar is used within each study to collate time series (as in var1).

dayvar — Optional string indicating the day variable (passed to tsData per study).

beepvar — Optional string indicating the beep variable within day (passed to tsData per study).

contemporaneous — Parameterization of the contemporaneous (residual innovation) covariance structure. One of "cov", "chol", "prec", "ggm", or "cor".

beta — Temporal lag-1 regression matrix specification. Defaults to "full" (all elements free, including diagonal autoregressive effects).

omega_zeta — Contemporaneous partial correlation matrix specification (used when contemporaneous = "ggm").

delta_zeta — Contemporaneous scaling matrix specification (used when contemporaneous = "ggm").

kappa_zeta — Contemporaneous precision matrix specification (used when contemporaneous = "prec").

sigma_zeta — Contemporaneous covariance matrix specification (used when contemporaneous = "cov").

lowertri_zeta — Contemporaneous Cholesky factor specification (used when contemporaneous = "chol").

randomEffects — Parameterization of the random effects covariance structure.

sigma_randomEffects — Random effects covariance matrix specification (used when randomEffects = "cov").

kappa_randomEffects — Random effects precision matrix specification (used when randomEffects = "prec").

omega_randomEffects — Random effects partial correlation matrix specification (used when randomEffects = "ggm").

lowertri_randomEffects — Random effects Cholesky factor specification (used when randomEffects = "chol").

delta_randomEffects — Random effects scaling matrix specification (used when randomEffects = "ggm").

`rho_randomEffects`
: Random effects correlation matrix specification (used when `randomEffects = "cor"`).

`SD_randomEffects`
: Random effects standard deviation matrix specification (used when `randomEffects = "cor"`).

`Vmats`
: Optional list with 'V' matrices (sampling error variance approximations).

`Vmethod`
: Which method should be used to approximate the sampling error variance? `"individual"` or `"pooled"`.

`Vestimation`
: How should the sampling error estimates be evaluated? `"averaged"` or `"per_study"`.

`baseline_saturated`
: Logical indicating if baseline and saturated models should be included.

`optimizer`
: The optimizer to be used. Defaults to `"nlminb"`.

`estimator`
: The estimator to be used. `"ML"` or `"FIML"` (default).

`sampleStats`
: Optional sample statistics object.

`verbose`
: Logical, should progress be printed?

`bootstrap`
: Should the data be bootstrapped?

`boot_sub`
: Proportion of cases to subsample for bootstrap.

`boot_resample`
: Logical, should bootstrap be with replacement?

`rho_zeta`
: Only used when `contemporaneous = "cor"`. Contemporaneous (innovation) correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).

`SD_zeta`
: Only used when `contemporaneous = "cor"`. Diagonal matrix of contemporaneous (innovation) standard deviations (`"full"`, `"diag"`, or a pattern matrix).

`...`
: Arguments sent to `meta_var1`.

Details

This function implements a single-stage meta-analytic VAR(1) model. For `p` observed variables, each study's time-series data is transformed into a Toeplitz covariance structure from which two blocks are extracted:

- **Sigma_0**: the `p` x `p` stationary covariance (symmetric, `p(p+1)/2` unique elements)
- **Sigma_1**: the `p` x `p` lag-1 cross-covariance (non-symmetric, `p^2` elements)

The VAR(1) structural model implies:

$$\text{vec}(\Sigma_0) = (I - \beta \otimes \beta)^{-1}\text{vec}(\Sigma_\zeta)$$

$$\Sigma_1 = \beta\Sigma_0$$

The meta-level mean is $\mu = [\text{vech}(\Sigma_0), \text{vec}(\Sigma_1)]$ and heterogeneity is modeled as $\Sigma = \Sigma^{(\text{ran})} + V$.

Note: the exogenous (lagged) covariance block from the full Toeplitz matrix is not modeled, as it is a nuisance parameter at the meta-analytic level.

Sample size convention. When raw `data` is supplied, the per-study sample size used in the sampling-error approximation (the 'V' matrices) is the number of rows of the lag-augmented data

set, including rows in which the lagged block is missing (e.g., the first measurement of each day or subject). The covariances themselves are computed using pairwise-complete observations. For studies consisting of many short segments (e.g., few beeps per day), this convention slightly overstates the effective number of (current, lagged) pairs for the lagged-covariance elements, and hence slightly understates their sampling variance. When this is a concern, the 'V' matrices can be supplied manually via `Vmats`.

Model size limits. The number of meta-level outcomes is `p(p+1)/2 + p^2`, which grows quickly with the number of variables: the default full random-effects structure has on the order of `p^4` parameters, and internal matrices grow with the fourth power of the number of outcomes. Models with more than 11 variables (215 outcomes) exceed internal limits and are refused with an informative error; models close to that limit are very slow and memory-hungry, and require many studies to be estimable (a warning is given when the number of studies is smaller than the number of outcomes). For large numbers of variables, or when the "studies" are in fact subjects of a single intensive longitudinal study, consider the multi-level VAR model `ml_var1` / `ml_gvar1` instead.

Fit indices. The saturated meta-analytic model (fully free Σ_0, Σ_1 and random effects) has zero degrees of freedom, in which case chi-square based fit indices are not informative and are left blank.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

See Also

`var1`, `meta_varcov`, `meta_lvm`

Examples

```
## Not run:
library("dplyr")
set.seed(42)

# Simulate stationary VAR(1) time-series data for several subjects/studies
# directly in base R (no extra packages needed):
nNode  <- 3
nTime  <- 50
nStudy <- 20

# Fixed lag-1 (temporal) coefficient matrix:
beta <- matrix(c( 0.4,  0.1,  0.0,
                  0.0,  0.4,  0.1,
                  0.1,  0.0,  0.4), nNode, nNode, byrow = TRUE)

# Innovation (contemporaneous) covariance:
Sigma <- diag(nNode)
```

```
varNames <- paste0("V", 1:nNode)

simVAR1 <- function(id, nTime, beta, Sigma){
  p <- nrow(beta)
  L <- chol(Sigma)
  Y <- matrix(0, nTime, p)
  for (t in 2:nTime){
    Y[t, ] <- beta %*% Y[t - 1, ] + as.vector(t(L) %*% rnorm(p))
  }
  out <- as.data.frame(Y)
  names(out) <- paste0("V", 1:p)
  out$study <- id
  out
}

Data <- do.call(rbind, lapply(seq_len(nStudy), simVAR1,
                              nTime = nTime, beta = beta, Sigma = Sigma))

# Fit meta-analytic VAR(1):
mod <- meta_var1(Data,
                 vars = varNames,
                 studyvar = "study",
                 estimator = "ML")
mod <- mod %>% runmodel

# Inspect results:
print(mod)
getmatrix(mod, "beta")

# Fit meta-analytic GVAR (with GGM contemporaneous):
mod_gvar <- meta_gvar(Data,
                      vars = varNames,
                      studyvar = "study",
                      estimator = "ML")
mod_gvar <- mod_gvar %>% runmodel
getmatrix(mod_gvar, "omega_zeta")

## End(Not run)
```

`meta_varcov` *Variance-covariance and GGM meta analysis*

Description

Meta analysis of correlation matrices to fit a homogenous correlation matrix or Gaussian graphical model. Based on meta-analytic SEM (Jak and Cheung, 2019).

Usage

```
meta_varcov(cors, nobs, data, covs, studyvar, groups, groupvar,
```

```
corinput, Vmats, Vmethod = c("individual", "pooled",
"OS_individual", "OS_pooled",
"metaSEM_individual", "metaSEM_weighted"), Vestimation
= c("averaged", "per_study"), type = c("cor", "ggm"),
sigma_y = "full", kappa_y = "full", omega_y = "full",
lowertri_y = "full", delta_y = "full", rho_y = "full",
SD_y = "full", randomEffects = c("chol", "cov",
"prec", "ggm", "cor"), sigma_randomEffects = "full",
kappa_randomEffects = "full", omega_randomEffects =
"full", lowertri_randomEffects = "full",
delta_randomEffects = "full", rho_randomEffects =
"full", SD_randomEffects = "full", vars,
baseline_saturated = TRUE, optimizer, estimator =
c("FIML", "ML"), sampleStats, verbose = FALSE,
bootstrap = FALSE, boot_sub, boot_resample)
```

```
meta_ggm(...)
```

Arguments

cors	A list of correlation matrices. Must contain rows and columns with NAs for variables not included in a study.
nobs	A vector with the number of observations per study.
data	Optional data frame. When supplied together with studyvar, correlation matrices and sample sizes are computed internally per study. Cannot be used together with cors, covs, or nobs.
covs	A list of covariance matrices. Alternative to cors; when supplied, corinput defaults to FALSE.
studyvar	A string indicating the column name in data that identifies the study. Required when data is supplied.
groups	Deprecated. Use groupvar instead. Multi-group support is not yet included for meta-analytic models.
groupvar	Not yet supported for meta-analytic models. Supplying this argument will produce an error.
corinput	Logical. Defaults to TRUE when cors is used, FALSE when covs is used. Controls whether the input is treated as correlation matrices.
Vmats	Optional list with 'V' matrices (sampling error variance approximations). May not be combined with an explicitly supplied Vmethod (which would otherwise silently be ignored).
Vmethod	Which method should be used to approximate the sampling error variance? One of "individual" (the default; the sampling-error covariance matrix is approximated separately for each study), "pooled" (a single pooled approximation is used for all studies), "OS_individual" or "OS_pooled" (the asymptotic sampling covariance matrix of the sample correlations following Olkin and Siotani, 1976, evaluated per study or at the sample-size-weighted pooled correlation matrix; for covariance input the corresponding Wishart expression is used),

"metaSEM_individual", or "metaSEM_weighted". The two metaSEM_* options compute the sampling-error covariances via metaSEM::asyCov and therefore require the Suggests package metaSEM to be installed (per study or using sample-size weighting, respectively). See Details for the difference between the default methods and the OS_* methods.

Vestimation — How should the sampling error estimates be evaluated? Either "averaged" (the default; a single sampling-error covariance matrix is averaged over the studies) or "per_study" (a separate sampling-error covariance matrix is retained for each study).

type — What to model? Currently only "cor" and "ggm" are supported.

sigma_y — Reserved covariance-matrix specification. Not used for the currently supported values of type ("cor" and "ggm"); retained for internal use and forward compatibility.

kappa_y — Reserved precision-matrix specification. Not used for the currently supported values of type ("cor" and "ggm"); retained for internal use and forward compatibility.

omega_y — Only used when type = "ggm". Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

lowertri_y — Reserved Cholesky-factor specification. Not used for the currently supported values of type ("cor" and "ggm"); retained for internal use and forward compatibility.

delta_y — Only used when type = "ggm". Either "diag" or "zero" (not recommended), or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

rho_y — Only used when type = "cor". Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

SD_y — Only used when type = "cor". Either "diag" or "zero", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

randomEffects — What to model for the random effects?

sigma_randomEffects — Only used when type = "cov". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups,

this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_randomEffects`
: Only used when `randomEffects = "prec"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_randomEffects`
: Only used when `randomEffects = "ggm"`. Either `"full"` to estimate every element freely, `"zero"` to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_randomEffects`
: Only used when `randomEffects = "chol"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_randomEffects`
: Only used when `randomEffects = "ggm"`. Either `"diag"` or `"zero"`, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`rho_randomEffects`
: Only used when `randomEffects = "cor"`. Either `"full"` to estimate every element freely, `"zero"` to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`SD_randomEffects`
: Only used when `randomEffects = "cor"`. Either `"diag"` or `"zero"`, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`vars`
: Variables to be included.

`baseline_saturated`
: A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

optimizer	The optimizer to be used. Can be one of "nlminb" (the default R nlminb function), "ucminf" (from the optimr package), "nloptr_TNEWTON" (preconditioned truncated Newton via nloptr), and "LBFGS++" (pure C++ L-BFGS-B). Defaults to "nlminb".
estimator	The estimator to be used. Currently implemented are "ML" for maximum likelihood estimation or "FIML" for full-information maximum likelihood estimation.
sampleStats	An optional sample statistics object. Mostly used internally.
verbose	Logical, should progress be printed to the console?
bootstrap	Should the data be bootstrapped? If TRUE the data are resampled and a bootstrap sample is created. These must be aggregated using aggregate_bootstraps! Can be TRUE or FALSE. Can also be "nonparametric" (which sets boot_sub = 1 and boot_resample = TRUE) or "case" (which sets boot_sub = 0.75 and boot_resample = FALSE).
boot_sub	Proportion of cases to be subsampled (round(boot_sub * N)).
boot_resample	Logical, should the bootstrap be with replacement (TRUE) or without replacement (FALSE)
...	Arguments sent to meta_varcov

Details

The default sampling-error approximations Vmethod = "individual" and "pooled" invert the Fisher information of a saturated correlation model (with unit diagonal constraint). This is the asymptotic covariance matrix of the maximum-likelihood estimates of a constrained correlation model, which is smaller than the sampling covariance of the raw sample correlations that enter the meta-level data (for a single correlation: (1 - rho^2)^2 / ((1 + rho^2) n) instead of (1 - rho^2)^2 / n). The alternative options Vmethod = "OS_individual" and "OS_pooled" instead use the asymptotic sampling covariance matrix of the sample correlations themselves (Olkin and Siotani, 1976), which matches metaSEM::asyCov and may be preferred when the absolute size of the random-effects (heterogeneity) variances is of interest: with the default methods part of the sampling variability can be absorbed into the random-effects variances, inflating them somewhat. The pooled correlation estimates themselves are typically nearly identical between these choices. The default is kept at "individual" for backward compatibility with published results.

Note that the saturated meta-analytic model (a fully free pooled correlation structure with fully free random effects) has zero degrees of freedom, in which case chi-square based fit indices (e.g., RMSEA, CFI) are not informative and are left blank.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

Jak, S., and Cheung, M. W. L. (2019). Meta-analytic structural equation modeling with moderating effects on SEM parameters. Psychological methods.

Olkin, I., and Siotani, M. (1976). Asymptotic distribution of functions of a correlation matrix. In S. Ikeda (Ed.), Essays in probability and statistics (pp. 235-251). Shinko Tsusho.

Examples

```
# Simulate a list of correlation matrices for several studies:
set.seed(123)
nStudy <- 8
nNode  <- 4
trueRho <- diag(nNode)
trueRho[lower.tri(trueRho)] <- trueRho[upper.tri(trueRho)] <- 0.3
nobs <- sample(100:300, nStudy)
cors <- lapply(seq_len(nStudy), function(i){
  X <- MASS::mvrnorm(nobs[i], rep(0, nNode), trueRho)
  R <- cor(X)
  colnames(R) <- rownames(R) <- paste0("V", 1:nNode)
  R
})

# Meta-analytic GGM using the list of correlation matrices and sample sizes:
mod <- meta_varcov(cors = cors, nobs = nobs, type = "ggm")
mod <- runmodel(mod)

# Pooled network:
getmatrix(mod, "omega_y")
```

MIs *Print modification indices*

Description

This function prints a list of modification indices (MIs)

Usage

```
MIs(x, all = FALSE, matrices, type = c("normal", "equal", "free"), top = 10,
    verbose = TRUE, nonZero = FALSE)
```

Arguments

x	A psychonetrics model.
all	Logical, should all MIs be printed or only the highest?
matrices	Optional vector of matrices to include in the output.

type	String (or vector of strings) indicating which kind of modification index should be printed. "normal" (the default) is the typical modification index for freeing a single parameter. "free" is the modification index for freeing the parameter in all groups (for multi-group models this is accompanied by a joint multivariate score test table). "equal" is the modification index for freeing the parameter constrained equal across all groups.
top	Number of MIs to include in output if all = FALSE
verbose	Logical, should messages be printed?
nonZero	Logical, should only MIs be printed of non-zero parameters? Useful to explore violations of group equality.

Value

Invisibly returns a relevant subset of the data frame containing all information on the parameters, or a list of such data frames if multiple types of MIs are requested.

Author(s)

Sacha Epskamp

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars, omega = "zero")

# Run model:
mod <- mod %>% runmodel

# Modification indices:
mod %>% MIs
```

ml_lvm *Multi-level latent variable model family*

Description

This family is the two-level random intercept variant of the lvm model family. It is mostly a special case of the dlvm1 family, with the addition of structural effects rather than temporal effects in the beta matrix.

Usage

```
ml_lnm(...)
ml_rnm(...)
ml_lrnm(...)
ml_lvm(data, lambda, clusters, within_latent = c("cov",
                    "chol", "prec", "ggm", "cor"), within_residual = c("cov",
                    "chol", "prec", "ggm", "cor"), between_latent = c("cov",
                    "chol", "prec", "ggm", "cor"), between_residual = c("cov",
                    "chol", "prec", "ggm", "cor"), beta_within = "zero",
                    beta_between = "zero", omega_zeta_within = "full",
                    delta_zeta_within = "full", kappa_zeta_within =
                    "full", sigma_zeta_within = "full",
                    lowertri_zeta_within = "full", omega_epsilon_within =
                    "zero", delta_epsilon_within = "diag",
                    kappa_epsilon_within = "diag", sigma_epsilon_within =
                    "diag", lowertri_epsilon_within = "diag",
                    omega_zeta_between = "full", delta_zeta_between =
                    "full", kappa_zeta_between = "full",
                    sigma_zeta_between = "full", lowertri_zeta_between =
                    "full", omega_epsilon_between = "zero",
                    delta_epsilon_between = "diag", kappa_epsilon_between
                    = "diag", sigma_epsilon_between = "diag",
                    lowertri_epsilon_between = "diag", nu, nu_eta,
                    identify = TRUE, identification = c("loadings",
                    "variance"), vars, latents, groups, equal = "none",
                    baseline_saturated = TRUE, estimator = c("default",
                    "FIML", "ML"),
                    optimizer, storedata = FALSE, verbose =
                    FALSE, standardize = c("none", "z", "quantile"),
                    sampleStats, bootstrap = FALSE, boot_sub,
                    boot_resample,
                    rho_zeta_within = "full", SD_zeta_within = "full",
                    rho_epsilon_within = "zero", SD_epsilon_within = "diag",
                    rho_zeta_between = "full", SD_zeta_between = "full",
                    rho_epsilon_between = "zero", SD_epsilon_between = "diag")
```

Arguments

data	A data frame encoding the data used in the analysis. Must be a raw dataset.
lambda	A model matrix encoding the factor loading structure. Each row indicates an indicator and each column a latent. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Could also be the result of simplestructure.
clusters	A string indicating the variable in the dataset that describes group membership.
within_latent	The type of within-person latent contemporaneous model to be used.
within_residual	The type of within-person residual model to be used.
between_latent	The type of between-person latent model to be used.
between_residual	The type of between-person residual model to be used.
beta_within	A model matrix encoding the within-cluster structural. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Defaults to "zero".
beta_between	A model matrix encoding the between-cluster structural. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Defaults to "zero".
omega_zeta_within	Only used when within_latent = "ggm". Can be "full", "zero", or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
delta_zeta_within	Only used when within_latent = "ggm". Can be "diag", "zero" (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
kappa_zeta_within	Only used when within_latent = "prec". Can be "full", "diag", or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
sigma_zeta_within	Only used when within_latent = "cov". Can be "full", "diag", or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_zeta_within`
: Only used when `within_latent = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_epsilon_within`
: Only used when `within_residual = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_epsilon_within`
: Only used when `within_residual = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_epsilon_within`
: Only used when `within_residual = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_epsilon_within`
: Only used when `within_residual = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_epsilon_within`
: Only used when `within_residual = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_zeta_between`
: Only used when `between_latent = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_zeta_between`
: Only used when `between_latent = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

kappa_zeta_between
: Only used when `between_latent = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

sigma_zeta_between
: Only used when `between_latent = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

lowertri_zeta_between
: Only used when `between_latent = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

omega_epsilon_between
: Only used when `between_residual = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

delta_epsilon_between
: Only used when `between_residual = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

kappa_epsilon_between
: Only used when `between_residual = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

sigma_epsilon_between
: Only used when `between_residual = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

lowertri_epsilon_between
: Only used when `between_residual = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

nu	Optional vector encoding the intercepts of the observed variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.
nu_eta	Optional vector encoding the intercepts of the latent variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.
identify	Logical, should the model be automatically identified?
identification	Type of identification used. "loadings" to fix the first factor loadings to 1, and "variance" to fix the diagonal of the latent variable model matrix (sigma_zeta, lowertri_zeta, delta_zeta or kappa_zeta) to 1.
vars	An optional character vector with names of the variables used.
latents	An optional character vector with names of the latent variables.
groups	An optional string indicating the name of the group variable in data.
equal	A character vector indicating which matrices should be constrained equal across groups.
baseline_saturated	A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.
estimator	The estimator to be used. "ML" and "FIML" maximize the *same* (exact) two-level maximum-likelihood objective and converge to the same solution; they differ only in how that likelihood is *computed*, exactly as for the single-level model families in this package, where "ML" fits the model to summary statistics and "FIML" fits it row by row to the raw data. "ML" (experimental) evaluates the likelihood from two-level sufficient statistics (pooled within-cluster covariance and per-cluster-size between moments; McDonald & Goldstein, 1989), which for complete data is much faster and does not slow down as clusters grow. "FIML" (wide-format full-information maximum likelihood) stacks each cluster's members into one wide observation; it was the only estimator before version 0.15.31. Both handle within-cluster missing data ("ML" through a per-pattern, per-cluster likelihood; see Details). This is NOT the approximate "MUML" / pseudo-balanced estimator: the sufficient-statistics evaluation is exact for unbalanced designs, and the two estimators' fit-function values coincide to numerical precision (verified to ~1e-13). "default" selects "ML" when the data are complete and the largest cluster contains more than 5 units, and "FIML" otherwise; with missing data the default stays on the long-standing "FIML" path, so the (R-only) missing-data "ML" estimator is used only when estimator = "ML" is requested explicitly.
optimizer	The optimizer to be used. Can be one of "nlminb" (the default R nlminb function), "ucminf" (from the optimr package), "nloptr_TNEWTON" (preconditioned truncated Newton via nloptr), and "LBFGS++" (pure C++ L-BFGS-B). Defaults to "nlminb".
storedata	Logical, should the raw data be stored? Needed for bootstrapping (see bootstrap).
verbose	Logical, should progress be printed to the console?

`standardize` Which standardization method should be used? `"none"` (default) for no standardization, `"z"` for z-scores, and `"quantile"` for a non-parametric transformation to the quantiles of the marginal standard normal distribution.

`sampleStats` An optional sample statistics object. Mostly used internally.

`bootstrap` Should the data be bootstrapped? If `TRUE` the data are resampled and a bootstrap sample is created. These must be aggregated using `aggregate_bootstraps`! Can be `TRUE` or `FALSE`. Can also be `"nonparametric"` (which sets `boot_sub = 1` and `boot_resample = TRUE`) or `"case"` (which sets `boot_sub = 0.75` and `boot_resample = FALSE`).

`boot_sub` Proportion of cases to be subsampled (`round(boot_sub * N)`).

`boot_resample` Logical, should the bootstrap be with replacement (`TRUE`) or without replacement (`FALSE`)

`rho_zeta_within` Only used when `within_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix; see `omega_zeta_within`).

`SD_zeta_within` Only used when `within_latent = "cor"`. Diagonal standard-deviation matrix specification (`"full"`, `"diag"`, or a pattern matrix; see `delta_zeta_within`).

`rho_epsilon_within` Only used when `within_residual = "cor"`. Correlation matrix specification (`"zero"`, `"full"`, or a pattern matrix).

`SD_epsilon_within` Only used when `within_residual = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix).

`rho_zeta_between` Only used when `between_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).

`SD_zeta_between` Only used when `between_latent = "cor"`. Diagonal standard-deviation matrix specification (`"full"`, `"diag"`, or a pattern matrix).

`rho_epsilon_between` Only used when `between_residual = "cor"`. Correlation matrix specification (`"zero"`, `"full"`, or a pattern matrix).

`SD_epsilon_between` Only used when `between_residual = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix).

`...` Arguments sent to 'ml_lvm'

Details

The `"ML"` and `"FIML"` estimators are two computational routes to the *same* exact two-level maximum-likelihood estimate, not two different statistical methods and not an exact-versus-approximate pair: they maximize the same log-likelihood and (up to optimizer tolerance) return identical estimates, standard errors and fit. The naming follows the convention used throughout psychonetrics – `"ML"` fits to summary statistics while `"FIML"` fits row by row to the raw data – and does not by itself

indicate complete versus missing data, since both routes can accommodate within-cluster missingness. Choose `"ML"` for speed on (near-)complete data with non-trivial cluster sizes and `"FIML"` (or `"default"`) otherwise.

With `estimator = "FIML"`, every cluster is treated as one wide-format observation (members stacked side by side) and the model is estimated with full-information maximum likelihood over the missingness patterns. This handles missing data, but its cost grows quickly with the cluster size.

With `estimator = "ML"` (experimental), the same two-level random-intercept likelihood is instead evaluated from sufficient statistics (McDonald & Goldstein, 1989; Muthen, 1990) when the data are complete. Writing Σ_W and Σ_B for the implied within- and between-cluster covariance matrices and μ for the implied mean vector, minus twice the log-likelihood equals

$$(N-J)\left[\ln|\Sigma_W| + \mathrm{tr}(\Sigma_W^{-1} S_{PW})\right] + \sum_s m_s \left[\ln|\Sigma_s| + n_s \mathrm{tr}(\Sigma_s^{-1} A_s)\right] + Np\ln(2\pi),$$

in which N is the number of units, J the number of clusters, S_{PW} the pooled within-cluster covariance matrix, and, per distinct cluster size n_s with m_s clusters, $\Sigma_s = \Sigma_W + n_s\Sigma_B$ and A_s the second-moment matrix of the corresponding cluster means around μ. The two estimators optimize numerically identical objectives (the fit function values coincide at identical parameter values), so they converge to the same solution; the same likelihood is also used by `lavaan` for two-level models (Rosseel, 2012). The sufficient statistics are computed once, making `"ML"` much faster than `"FIML"` when clusters are large, and its cost does not depend on the cluster sizes. For complete data, standard errors and modification indices for `"ML"` use the analytic EXPECTED Fisher information. Note that `lavaan` uses the OBSERVED information by default for multilevel models, so standard errors reported by `lavaan` typically differ by a few percent; with `lavaan::sem(..., information = "expected")` the standard errors agree.

When the data contain within-cluster missing values (assumed missing at random), `estimator = "ML"` switches to a per-pattern, per-cluster evaluation of the same two-level likelihood: for each missingness pattern the within-cluster precision and log-determinant are obtained by a symmetric (Schur) inverse update of Σ_W^{-1}, and these are accumulated per cluster into the quantities entering minus twice the log-likelihood,

$$\sum_P f_P \ln|\Sigma_W^{(P)}| + \sum_j \ln|I + \Sigma_B A_j| + \sum_i (y_i-\mu)'\Sigma_W^{(P_i)-1}(y_i-\mu) - \sum_j p_j'(I+\Sigma_B A_j)^{-1}\Sigma_B\, p_j + n_{\mathrm{obs}}\ln(2\pi),$$

with $A_j = \sum_{i\in j}\Sigma_W^{(P_i)-1}$, $p_j = \sum_{i\in j}\Sigma_W^{(P_i)-1}(y_i-\mu)$ (observed coordinates only), f_P the number of units in pattern P, and n_{obs} the number of observed values. This is the same two-level missing-data likelihood that `lavaan::sem(..., cluster = , missing = "ml")` maximizes (Rosseel, 2012), and the two agree to optimizer tolerance. The missing-data path is evaluated in R (no C++ acceleration) and reduces exactly to the sufficient-statistics likelihood above when no values are missing; its standard errors use the numeric (observed-information-based) Fisher information.

The saturated (unrestricted) and baseline (independence) reference models used by `runmodel` for the fit measures are themselves estimated as two-level models: the saturated model freely estimates the means and the full within- and between-cluster covariance matrices, and the baseline model only estimates means and the variances at both levels. These are the same reference models `lavaan` uses for two-level SEM, so the saturated and baseline log-likelihoods, the model and baseline chi-square, the degrees of freedom, the CFI and the TLI agree with `lavaan` (up to optimization tolerances), provided the `lavaan` model uses the same parameterization (`ml_lvm` shares the factor loadings across the within and between levels, so the matching `lavaan` model must constrain its

level-1 and level-2 loadings to be equal). One convention difference remains: psychonetrics evaluates sample-size-dependent fit measures, such as the RMSEA (with its confidence interval) and the BIC, using the number of independent observations, which in these models is the number of *clusters* J, whereas `lavaan` uses the total number of level-1 *units* N. Because the RMSEA equals $\sqrt{\max((\chi^2/n)/df - 1/n, 0)}$ with n the number of independent observations, and the chi-square and degrees of freedom otherwise agree, the RMSEA reported by psychonetrics is larger than the value reported by `lavaan` by exactly a factor $\sqrt{N/J}$. This applies to both estimators of the `ml_lvm` family.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

McDonald, R. P., & Goldstein, H. (1989). Balanced versus unbalanced designs for linear structural relations in two-level data. British Journal of Mathematical and Statistical Psychology, 42(2), 215-232.

Muthen, B. O. (1990). Mean and covariance structure analysis of hierarchical data. UCLA Statistics Series #62.

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

Examples

```
# Simulate two-level data with 2 latent variables on both levels:
set.seed(1)
J <- 50 # clusters
nj <- 10 # units per cluster
p <- 6
Lambda <- matrix(0, p, 2)
Lambda[1:3,1] <- c(1, 0.8, 0.7)
Lambda[4:6,2] <- c(1, 0.9, 0.6)
PsiW <- matrix(c(1,0.3,0.3,1), 2)
PsiB <- matrix(c(0.5,0.2,0.2,0.4), 2)
data <- do.call(rbind, lapply(seq_len(J), function(j){
  bj <- MASS::mvrnorm(1, rep(0,2), PsiB)
  ej <- rnorm(p, sd = 0.4) # between-level residuals
  t(replicate(nj, as.vector(Lambda %*% (bj + MASS::mvrnorm(1, rep(0,2), PsiW))) +
      ej + rnorm(p, sd = 0.7)))
}))
data <- as.data.frame(data)
names(data) <- paste0("y", 1:p)
data$cluster <- rep(seq_len(J), each = nj)

# Lambda model matrix (shared across levels):
lambda <- matrix(0, p, 2)
```

```
lambda[1:3,1] <- lambda[4:6,2] <- 1

# Two-level ML estimator (fast, complete data):
mod <- ml_lvm(data, lambda = lambda, clusters = "cluster", estimator = "ML")
mod <- runmodel(mod)
mod

# The FIML estimator (slower; supports missing data) converges to the
# same solution:
mod_fiml <- ml_lvm(data, lambda = lambda, clusters = "cluster",
                   estimator = "FIML")
mod_fiml <- runmodel(mod_fiml)
```

`ml_tsdlvm1` *Multi-level Lag-1 dynamic latent variable model family of psychonetrics models for time-series data*

Description

Deprecated. Use `dlvm1` directly with long-format data instead (set `vars` to a character vector and provide `idvar`).

This function is a deprecated wrapper around `dlvm1` that allows for specifying the model using a long format data and similar input as the `mlVAR` package. The `ml_ts_lvgvar` simply sets `within_latent = "ggm"` and `between_latent = "ggm"` by default. These functions now call `dlvm1()` directly and emit deprecation warnings. (The observed-variable wrappers `ml_var` and `ml_gvar` are now aliases for `ml_var1` and `ml_gvar1`, and are documented there.)

Usage

```
ml_tsdlvm1(data, beepvar, idvar, vars, groups, estimator = "FIML",
  standardize = c("none", "z", "quantile"), ...)

ml_ts_lvgvar(...)
```

Arguments

`data`	The data to be used. Must be raw data in long format (each row indicates one person at one time point).
`beepvar`	Optional string indicating assessment beep per day. Adding this argument will cause non-consecutive beeps to be treated as missing!
`idvar`	String indicating the subject ID
`vars`	Vectors of variables to include in the analysis
`groups`	An optional string indicating the name of the group variable in `data`.
`estimator`	Estimator to be used. Must be `"FIML"`.

standardize	Which standardization method should be used? "none" (default) for no standardization, "z" for z-scores, and "quantile" for a non-parametric transformation to the quantiles of the marginal standard normal distribution.
...	Arguments sent to dlvm1

Value

An object of the class psychonetrics (psychonetrics-class).

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

---

ml_var1 *Multi-level lag-1 vector auto-regression family (random intercepts)*

Description

This family models a multi-level (random intercept) lag-1 vector auto-regression (VAR) / graphical VAR for intensive longitudinal data (many repeated measures nested in subjects), estimated by a two-level summary-statistics pseudo maximum likelihood. Each subject follows a within-person stationary VAR(1) process (temporal matrix beta and contemporaneous structure sigma_zeta_within) around a person-specific random intercept, and the random intercepts have a between-person covariance structure (sigma_zeta_between). The ml_var1 function models the contemporaneous and between-person structures as variance–covariance matrices by default, and the ml_gvar1 wrapper models them as Gaussian graphical models (GGMs), yielding the three multi-level GVAR networks: the temporal network (beta / partial directed correlations), the contemporaneous network (omega_zeta_within) and the between-person network (omega_zeta_between).

This model targets the same population model as panelvar / panelgvar (a multi-level VAR with random intercepts), but is designed for intensive longitudinal / experience-sampling (ESM) data, where the number of repeated measures per subject is large and the full-likelihood panelvar route is infeasible. See Details for the estimator and its statistical caveats.

ml_var and ml_gvar are the primary names of this family (ml_var1 / ml_gvar1 are identical and remain fully supported; the historical "1" referred to the lag order). In psychonetrics 0.15 ml_var / ml_gvar were wrappers around dlvm1 with estimator = "FIML"; the default estimator = "auto" reproduces that behavior for short (panel-like) measurement-occasion grids (at most 10 occasions, now through the faster panelvar framework with identical estimates) and uses the two-level summary-statistics estimator for longer (ESM-like) series, where full-information ML is not feasible. Their argument names contemporaneous and between are mapped to within_latent and between_latent.

This estimator is experimental (a message is shown once per session).

Usage

```
ml_var1(data, idvar, vars, dayvar, beepvar, groups, groupvar,
        within_latent = c("cov", "chol", "prec", "ggm", "cor"),
        between_latent = c("cov", "chol", "prec", "ggm", "cor"),
        beta = "full",
        omega_zeta_within = "full", delta_zeta_within = "diag",
        kappa_zeta_within = "full", sigma_zeta_within = "full",
        lowertri_zeta_within = "full", rho_zeta_within = "full",
        SD_zeta_within = "full",
        omega_zeta_between = "full", delta_zeta_between = "diag",
        kappa_zeta_between = "full", sigma_zeta_between = "full",
        lowertri_zeta_between = "full", rho_zeta_between = "full",
        SD_zeta_between = "full",
        mu, equal = "none", baseline_saturated = TRUE,
        estimator = c("auto", "ML", "FIML"),
        optimizer, storedata = FALSE,
        standardize = c("none", "z", "quantile"),
        sampleStats, verbose = FALSE, toeplitz = TRUE,
        within, between)

ml_gvar1(..., within_latent = "ggm", between_latent = "ggm")

ml_var(..., contemporaneous = c("cov", "chol", "prec", "ggm", "cor"),
       between = c("cov", "chol", "prec", "ggm", "cor"))

ml_gvar(..., contemporaneous = c("ggm", "cov", "chol", "prec", "cor"),
        between = c("ggm", "cov", "chol", "prec", "cor"))
```

Arguments

data	A data frame encoding the (long-format) data used in the analysis. Must be a raw dataset with one row per measurement occasion, containing the subject id variable (idvar) and the variables in vars.
idvar	Required string indicating the subject/cluster ID variable in data. Each subject's rows form one within-person time series; the random intercepts vary over subjects. (Use var1 for single-subject time-series models.)
vars	An optional character vector encoding the variables used in the analysis. Must equal names in data. If missing, all columns except idvar, dayvar, beepvar and the group variable are used.
dayvar	Optional string indicating assessment day. Adding this argument makes sure that the first measurement of a day is not regressed on the last measurement of the previous day. IMPORTANT: only add this if the data has multiple observations per day.
beepvar	Optional string indicating assessment beep per day. Adding this argument will cause non-consecutive beeps to be treated as missing (the corresponding lag pairs are dropped).

`groups` Deprecated. Use `groupvar` instead. An optional string indicating the name of the group variable in `data`.

`groupvar` An optional string indicating the name of the group variable in `data` for multi-group analysis. Replaces the deprecated `groups` argument.

`within_latent` The type of within-person contemporaneous model to be used: `"cov"` (covariance, default), `"chol"` (Cholesky), `"prec"` (precision), `"ggm"` (Gaussian graphical model, giving the contemporaneous network `omega_zeta_within`), or `"cor"` (correlation).

`between_latent` The type of between-person (random-intercept) model to be used: `"cov"` (default), `"chol"`, `"prec"`, `"ggm"` (giving the between-person network `omega_zeta_between`), or `"cor"`.

`beta` A model matrix encoding the temporal relationships (transpose of the temporal network; from = row, to = column, as in `var1`). A 0 encodes a fixed to zero element, a 1 a free to estimate element, and higher integers equality constraints. For multiple groups, this can be a list or array. Can also be `"full"` (default) for a full temporal network or `"zero"` for an empty temporal network.

`omega_zeta_within`
Only used when `within_latent` = `"ggm"`. Can be `"full"`, `"zero"`, or a model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this can be a list or array.

`delta_zeta_within`
Only used when `within_latent` = `"ggm"`. Scaling (delta) matrix. Can be `"diag"`, `"zero"` (not recommended), or a model matrix.

`kappa_zeta_within`
Only used when `within_latent` = `"prec"`. Precision matrix. Can be `"full"`, `"diag"`, or a model matrix.

`sigma_zeta_within`
Only used when `within_latent` = `"cov"`. Covariance matrix. Can be `"full"`, `"diag"`, or a model matrix.

`lowertri_zeta_within`
Only used when `within_latent` = `"chol"`. Cholesky (lower triangular) matrix. Can be `"full"`, `"diag"`, or a model matrix.

`rho_zeta_within`
Only used when `within_latent` = `"cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).

`SD_zeta_within` Only used when `within_latent` = `"cor"`. Diagonal standard-deviation matrix specification (`"full"`, `"diag"`, or a pattern matrix).

`omega_zeta_between`
Only used when `between_latent` = `"ggm"`. Between-person network. Can be `"full"`, `"zero"`, or a model matrix.

`delta_zeta_between`
Only used when `between_latent` = `"ggm"`. Scaling (delta) matrix. Can be `"diag"`, `"zero"` (not recommended), or a model matrix.

`kappa_zeta_between`
Only used when `between_latent` = `"prec"`. Precision matrix. Can be `"full"`, `"diag"`, or a model matrix.

`sigma_zeta_between`
: Only used when `between_latent = "cov"`. Covariance matrix. Can be `"full"`, `"diag"`, or a model matrix.

`lowertri_zeta_between`
: Only used when `between_latent = "chol"`. Cholesky (lower triangular) matrix. Can be `"full"`, `"diag"`, or a model matrix.

`rho_zeta_between`
: Only used when `between_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).

`SD_zeta_between`
: Only used when `between_latent = "cor"`. Diagonal standard-deviation matrix specification (`"full"`, `"diag"`, or a pattern matrix).

`mu`
: Optional vector encoding the stationary means. Set elements to 0 to indicate fixed to zero constraints, 1 to indicate free means, and higher integers to indicate equality constraints. For multiple groups, this can be a list or matrix.

`equal`
: A character vector indicating which matrices should be constrained equal across groups (e.g. `"beta"`).

`baseline_saturated`
: A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT recommended to be used manually.

`estimator`
: The estimator to be used. The default `"auto"` selects `"FIML"` when the measurement-occasion grid is short (at most 10 occasions per subject, the panel-data regime, where full-information ML is feasible and matches the behavior of `ml_var` in psychonetrics 0.15) and `"ML"` otherwise, with a message stating the choice. `"ML"` uses the two-level summary-statistics pseudo maximum likelihood described in Details. `"FIML"` dispatches to the `panelvar` framework: the measurement occasions become the waves of a panel model that is estimated with full-information maximum likelihood (missing responses and missed beeps are handled by FIML; a night break, when `dayvar` is used, is bridged by one structurally missing occasion, so the model links the last beep of a day to the first beep of the next day through two VAR steps). The FIML route is exact but its cost grows with the number of measurement occasions, so it is only feasible for a modest number of occasions per subject.

`optimizer`
: The optimizer to be used. Can be one of `"nlminb"` (the default R `nlminb` function), `"ucminf"`, `"nloptr_TNEWTON"`, and `"LBFGS++"`.

`storedata`
: Logical, should the raw data be stored?

`standardize`
: Which standardization method should be used on the raw variables before the lag embedding? `"none"` (default), `"z"` for z-scores, and `"quantile"` for a non-parametric quantile transformation. Standardization is grand (over all observations), not within-person, so that the between-person variance survives.

`sampleStats`
: An optional sample statistics object. Mostly used internally (for the recursive baseline/saturated calls).

`verbose`
: Logical, should progress be printed to the console?

`toeplitz`
: Internal. If `TRUE` (default) the stationary Toeplitz VAR(1) within structure is imposed; `FALSE` selects the free-within (saturated) mode used internally to build the reference model. Not intended for direct use.

`...`	Arguments sent to `ml_var1` (used by `ml_gvar1`, `ml_var` and `ml_gvar`).
`within`	Alias for `within_latent` (there are no latent variables in this model, so the shorter name is preferred).
`between`	Alias for `between_latent`; also the name used by `ml_var` and `ml_gvar` since psychonetrics 0.15.
`contemporaneous`	Alias for `within_latent`, used by `ml_var` and `ml_gvar` (matching the `mlVAR` package).

Details

For each subject the observed p-variate series is augmented into complete lag pairs $z_{it} = (y'_{i,t-1}, y'_{it})'$ (the $2p$-variate "Toeplitz" embedding also used by `var1`; pairs are broken across days/beeps via `dayvar` / `beepvar`). These lag pairs are then treated, *within subject*, as conditionally independent given the subject's random intercept b_i:

$$z_{it} \mid b_i \sim N(\mu_z + (b'_i, b'_i)',\ \Sigma_W)\,, \qquad b_i \sim N(0, \Sigma_B),$$

with $\mu_z = (\mu', \mu')'$ the stationary mean repeated across the two halves, the within-person block a stationary Toeplitz structure $\Sigma_W = [[\Sigma_0, \Sigma_0\beta'], [\beta\Sigma_0, \Sigma_0]]$ with $\mathrm{vec}(\Sigma_0) = (I - \beta \otimes \beta)^{-1}\mathrm{vec}(\Sigma_\zeta)$, and the between-person block $\Sigma_{Bf} = 1_{2\times 2} \otimes \Sigma_B$ (a rank-p, $2p \times 2p$ matrix). Minus twice the log-likelihood is the two-level Gaussian objective of McDonald & Goldstein (1989) / Muthen (1990) evaluated at $(\mu_z, \Sigma_W, \Sigma_{Bf})$ with the subject as cluster.

Because consecutive lag pairs share y_{it}, this is a *working-independence pseudo (composite) likelihood*, exactly the move `var1` makes at a single subject. Its key advantage is that the data are compressed *once* into the two-level sufficient statistics of the lag pairs, so every fit / gradient / Fisher-information evaluation works only with $2p \times 2p$ matrices, and the per-evaluation cost is *independent of the number of time points*. `ml_var1` therefore uses the same data augmentation as `var1` and the same two-level estimator as `ml_lvm` (`estimator = "ML"`).

`ml_var1` shares the estimand with `panelvar` / `panelgvar` (a multi-level VAR(1) with random intercepts) but not the estimator: `panelvar` forms the full $(Tp) \times (Tp)$ block-Toeplitz likelihood, which is statistically ideal for panel data (few waves, many subjects) but grows with the number of waves T and becomes infeasible for intensive longitudinal / ESM data with dozens to hundreds of measurements per subject. *Use* `panelvar` / `panelgvar` *for few-wave panel data, and* `ml_var1` / `ml_gvar1` *for many repeated measures where* `panelvar` *is infeasible.* In this first version, incomplete lag pairs (rows with any missing value, including the structural lag rows at day/beep breaks) are dropped (complete-pairs, listwise analysis); a full-information treatment of the augmented rows is planned for a later version.

Statistical caveats. Because the estimator treats consecutive lag pairs as independent (working-independence pseudo-likelihood, as in `var1`), two caveats apply. First, under positive temporal autocorrelation the between-person covariance Σ_B carries a small upward finite-length bias of order $O(1/T)$: in simulation ($p = 4$, $J = 60$, β spectral radius 0.5) the between-person variances were biased about +9% at $T = 25$ and +3% at $T = 50$, and negligibly by $T = 100$; a full-likelihood `panelvar` / `panelgvar` fit does not show this bias. Second, the Fisher-based standard errors are pseudo-likelihood standard errors and can be anti-conservative – most noticeably for the contemporaneous (within) network, where reported standard errors were about 75–80% of the empirical standard deviation, an under-statement that grows with temporal dependence and does not

vanish as T grows. Point estimates of the temporal network `beta` and the contemporaneous network `omega_zeta_within` are consistent and closely track the full-likelihood `panelvar` estimates.

The saturated (unrestricted) and baseline (independence) reference models used by `runmodel` for the fit measures are estimated within the `ml_var1` framework itself (a free two-level model over the augmented $2p$ variables and a two-level independence model, respectively). The number of independent observations used in sample-size-dependent fit measures (BIC, RMSEA) is the number of *subjects*. The model layer is currently implemented in R only (no C++ acceleration), but estimation is still fast because every evaluation touches only $2p \times 2p$ matrices.

Value

An object of the class psychonetrics (psychonetrics-class).

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

McDonald, R. P., & Goldstein, H. (1989). Balanced versus unbalanced designs for linear structural relations in two-level data. British Journal of Mathematical and Statistical Psychology, 42(2), 215-232.

Muthen, B. O. (1990). Mean and covariance structure analysis of hierarchical data. UCLA Statistics Series #62.

Epskamp, S. (2020). Psychometric network models from time-series and panel data. Psychometrika, 85(1), 206-231. doi:10.1007/s11336020096973

See Also

`var1` for the single-subject (or pooled) VAR(1)/GVAR, `panelvar` / `panelgvar` for the full-likelihood random-intercept VAR on panel data, `ml_lvm` for the two-level latent variable model that supplies the estimator, and `dlvm1` for the dynamic latent variable model.

Examples

```
## Not run:
library("dplyr")

# Simulate a tiny two-level VAR(1) data set (random intercepts):
set.seed(1)
p <- 3        # nodes
J <- 40       # subjects
TT <- 25      # measurements per subject

# Temporal (beta) and contemporaneous (within) structure:
beta <- diag(0.3, p); beta[1, 2] <- 0.2; beta[3, 1] <- -0.15
SigZ <- diag(p) * 0.8; SigZ[1, 2] <- SigZ[2, 1] <- 0.2
Sig0 <- matrix(solve(diag(p^2) - kronecker(beta, beta), as.vector(SigZ)), p, p)
SigB <- diag(p) * 0.4; SigB[2, 3] <- SigB[3, 2] <- 0.1
```

```
# Generate:
long <- do.call(rbind, lapply(seq_len(J), function(i){
  bi  <- MASS::mvrnorm(1, rep(0, p), SigB)          # random intercept
  eta <- MASS::mvrnorm(1, rep(0, p), Sig0)
  Y <- matrix(NA_real_, TT, p)
  Y[1, ] <- eta + bi
  for (t in 2:TT){
    eta <- as.vector(beta %*% eta) + MASS::mvrnorm(1, rep(0, p), SigZ)
    Y[t, ] <- eta + bi
  }
  data.frame(id = i, beep = seq_len(TT), Y)
}))
names(long)[3:5] <- paste0("V", 1:p)

# Fit the multi-level graphical VAR (GGM within and between):
mod <- ml_gvar1(long, vars = paste0("V", 1:p), idvar = "id", beepvar = "beep")
mod <- runmodel(mod)

# The three multi-level GVAR networks:
getmatrix(mod, "beta")                # temporal network
getmatrix(mod, "omega_zeta_within")   # contemporaneous network
getmatrix(mod, "omega_zeta_between")  # between-person network

## End(Not run)
```

`ml_varcov` *Multi-level variance-covariance family of psychonetrics models*

Description

This is the multi-level (two-level) analogue of the `varcov` family: it models the within-cluster and between-cluster variance-covariance matrices of clustered data directly, without a latent-variable layer. Each level can be parameterized independently through the `within` and `between` arguments: `"cov"` models a covariance matrix directly, `"chol"` a Cholesky decomposition, `"prec"` a precision matrix, `"ggm"` a Gaussian graphical model (the multi-level GGM; alias `ml_ggm()` sets both levels to `"ggm"`) and `"cor"` a correlation matrix (alias `ml_corr()` sets both levels to `"cor"`).

Two estimators are supported, exactly as in `ml_lvm`: `"ML"`, the two-level sufficient-statistics Gaussian maximum-likelihood estimator (its computational cost does not depend on the number of units per cluster), and `"FIML"`, wide-format full-information maximum likelihood (the natural choice with missing data). Both maximize the same likelihood and give identical estimates on complete data. The default picks `"ML"` for complete data with more than five units per cluster and `"FIML"` otherwise.

The multi-level GGM (`ml_ggm`) is also implemented, using a different (nodewise multilevel regression) estimator, in the `mlGGM` function of the `mlVAR` package. The `psychonetrics` implementation here estimates the full two-level covariance structure by maximum likelihood instead.

This framework is experimental (a message is shown once per session).

Usage

```
ml_varcov(data, clusters, vars,
          within  = c("cov", "chol", "prec", "ggm", "cor"),
          between = c("cov", "chol", "prec", "ggm", "cor"),
          groups, groupvar,
          sigma_within = "full", lowertri_within = "full",
          omega_within = "full", delta_within = "diag",
          kappa_within = "full", rho_within = "full", SD_within = "full",
          sigma_between = "full", lowertri_between = "full",
          omega_between = "full", delta_between = "diag",
          kappa_between = "full", rho_between = "full", SD_between = "full",
          mu, equal = "none", baseline_saturated = TRUE,
          estimator = c("default", "ML", "FIML"),
          optimizer, storedata = FALSE,
          standardize = c("none", "z", "quantile"), sampleStats,
          verbose = FALSE)

ml_ggm(...)

ml_corr(...)
```

Arguments

data	A data frame in long format: one row per unit, with a column identifying the cluster each unit belongs to.
clusters	String indicating the name of the cluster (grouping) variable in data. May not be missing.
vars	Character vector of the variables to include. If missing, all columns except clusters and the grouping variable are used.
within	The parameterization of the within-cluster covariance matrix. One of "cov", "chol", "prec", "ggm" or "cor".
between	The parameterization of the between-cluster covariance matrix. One of "cov", "chol", "prec", "ggm" or "cor".
groups	Deprecated alias for groupvar.
groupvar	An optional string indicating the name of a variable for a multi-group analysis.
sigma_within	Only used when within = "cov". Either "full", "diag" or a matrix specifying the covariance structure of the within block (0 = fixed to 0, 1 = free, higher integers = equality constraints).
lowertri_within	Only used when within = "chol". As sigma_within, for the lower-triangular Cholesky factor.
omega_within	Only used when within = "ggm". As sigma_within, for the within partial-correlation network (only the off-diagonal is used).
delta_within	Only used when within = "ggm". As sigma_within, for the within diagonal scaling matrix.

kappa_within	Only used when within = "prec". As sigma_within, for the within precision matrix.
rho_within	Only used when within = "cor". As sigma_within, for the within correlation matrix.
SD_within	Only used when within = "cor". As sigma_within, for the within standard-deviation scaling matrix.
sigma_between	As sigma_within, for the between-cluster block (used when between = "cov").
lowertri_between	As lowertri_within, for the between-cluster block.
omega_between	As omega_within, for the between-cluster block.
delta_between	As delta_within, for the between-cluster block.
kappa_between	As kappa_within, for the between-cluster block.
rho_between	As rho_within, for the between-cluster block.
SD_between	As SD_within, for the between-cluster block.
mu	Optional vector encoding the mean structure. Set elements to 0 to indicate fixed zeroes.
equal	A character vector indicating which matrices should be constrained equal across groups.
baseline_saturated	A logical indicating whether the baseline and saturated models should be included. Mostly used internally and should be kept to TRUE.
estimator	"ML" (two-level sufficient-statistics ML), "FIML" (wide-format full-information ML) or "default" ("ML" for complete data with more than five units per cluster, "FIML" otherwise). The estimator is fixed at model creation and cannot be switched afterwards. Note that the EBIC fit measures under "FIML" inherit the wide-format variable count from the sample statistics (as in ml_lvm), whereas under "ML" they use the number of modelled variables.
optimizer	The optimizer to be used. Usually left missing to use the default.
storedata	Logical, should the raw data be stored? Needed for bootstrapping.
standardize	Optional grand standardization of the raw variables before estimation: "none", "z" (z-scores) or "quantile".
sampleStats	An optional sample statistics object. Mostly used internally.
verbose	Logical, should progress be printed to the console?
...	Arguments passed to ml_varcov by ml_ggm and ml_corr.

Details

For unit j in cluster i the observed response is decomposed into a within-cluster and a between-cluster component,

$$y_{ij} = \mu + \eta_{ij}^{(W)} + \eta_{i}^{(B)},$$

with $\eta_{ij}^{(W)} \sim N(0, \Sigma_W)$ and $\eta_{i}^{(B)} \sim N(0, \Sigma_B)$ independent, so that $\mathrm{var}(y_{ij}) = \Sigma_W + \Sigma_B$ and $\mathrm{cov}(y_{ij}, y_{ij'}) = \Sigma_B$ for two units in the same cluster. Σ_W (sigma_within) and Σ_B (sigma_between)

are each parameterized according to the `within` and `between` arguments; for the `"ggm"` parameterization $\Sigma = \Delta(I - \Omega)^{-1}\Delta$ with partial-correlation network Ω (`omega_within` / `omega_between`) and scaling Δ (`delta_within` / `delta_between`).

This is the special case of `ml_lvm` in which the loadings matrix equals the identity and there is no residual variance, but it is implemented as its own framework with its own matrix names (`sigma_within`, `omega_within`, `sigma_between`, `omega_between`, ...) and its own (simpler) analytic derivatives.

Value

An object of the class psychonetrics (psychonetrics-class).

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

Epskamp, S., Rhemtulla, M., & Borsboom, D. (2017). Generalized network psychometrics: Combining network and latent variable models. *Psychometrika, 82*(4), 904-927. doi:10.1007/s11336-0179557x

Hamaker, E. L., & Muthen, B. (2020). The fixed versus random effects debate and how it relates to centering in multilevel modeling. *Psychological Methods, 25*(3), 365-379.

McDonald, R. P., & Goldstein, H. (1989). Balanced versus unbalanced designs for linear structural relations in two-level data. *British Journal of Mathematical and Statistical Psychology, 42*(2), 215-232.

See Also

`varcov` for the single-level variance-covariance family, `ggm` for the single-level GGM, and `ml_lvm` for the multi-level latent variable model that supplies the estimator.

Examples

```
## Not run:
library("dplyr")

# Simulate two-level data (200 clusters of 8 units, 4 variables):
set.seed(1)
nCluster <- 200; perCluster <- 8; p <- 4
cl <- rep(seq_len(nCluster), each = perCluster)
# True within and between covariance matrices:
SigmaW <- diag(p); SigmaW[1,2] <- SigmaW[2,1] <- 0.4
SigmaB <- diag(0.5, p); SigmaB[3,4] <- SigmaB[4,3] <- 0.2
b <- MASS::mvrnorm(nCluster, rep(0, p), SigmaB)
Y <- b[cl, ] + MASS::mvrnorm(nCluster * perCluster, rep(0, p), SigmaW)
Data <- as.data.frame(Y)
names(Data) <- paste0("V", 1:p)
Data$cluster <- cl
```

```
# Fit a multi-level GGM (both levels a Gaussian graphical model):
mod <- ml_ggm(Data, vars = paste0("V", 1:p), clusters = "cluster")
mod <- mod %>% runmodel

# Inspect the estimated within- and between-cluster networks:
getmatrix(mod, "omega_within")
getmatrix(mod, "omega_between")

# Prune non-significant edges and re-fit:
mod <- mod %>% prune(alpha = 0.05)

## End(Not run)
```

`modelsearch` *Stepwise model search*

Description

This function performs stepwise model search to find an optimal model that (locally) minimizes some criterion (by default, the BIC).

Usage

```
modelsearch(x, criterion = "bic", matrices, prunealpha = 0.01,
                    addalpha = 0.01, verbose, ...)
```

Arguments

`x`	A `psychonetrics` model.
`criterion`	String indicating the criterion to minimize. Any criterion from `fit` can be used.
`matrices`	Vector of strings indicating which matrices should be searched. Will default to network structures and factor loadings.
`prunealpha`	Minimal alpha used to consider edges to be removed
`addalpha`	Maximum alpha used to consider edges to be added
`verbose`	Logical, should messages be printed?
`...`	Arguments sent to `runmodel`

Details

The full algorithm is as follows:

1. Evaluate all models in which an edge is removed that has p > prunealpha, or an edge is added that has a modification index with p < addalpha

2. If none of these models improve the criterion, return the previous model and stop the algorithm

3. Update the model to the model that improved the criterion the most

4. Evaluate all other considered models that improved the criterion

5. If none of these models improve the criterion, go to 1, else go to 3

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

prune, stepup

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars)

# Run model:
mod <- mod %>% runmodel

# Model search
mod <- mod %>% prune(alpha= 0.01) %>% modelsearch
```

NA2020 *Network Analysis 2020 Self-Report Data*

Description

A subset of self-report data collected in 2020 by Adela-Maria Isvoranu as part of a graduate-level course on network analysis at the University of Amsterdam. Students in the course collected the data from a convenience sample. The full dataset contains responses to 87 items covering topics such as sleep, well-being, and social functioning. This subset contains 8 items used in Chapter 6 of the textbook *Network Psychometrics with R* (Epskamp, Isvoranu, & Haslbeck, 2022) to demonstrate Gaussian Graphical Model estimation with psychonetrics. All items are rated on a 1–7 Likert scale. The dataset contains some missing values, making it suitable for demonstrating FIML estimation.

Usage

```
data("NA2020")
```

Format

A data frame with 501 observations on 8 variables.

`regular_sleep` I try to keep a regular sleep pattern (Q10)

`worried_sleep` I am worried about my current sleeping behavior (Q13)

`sleep_interfere` My sleep interferes with my daily functioning (Q14)

`happy_health` I am happy with my physical health (Q68)

`optimistic_future` I feel optimistic about the future (Q70)

`very_happy` I am very happy (Q75)

`feel_alone` I often feel alone (Q77)

`happy_love_life` I am happy with my love life (Q80)

Source

The full dataset is available at https://osf.io/45n6d/. Data collected by Adela-Maria Isvoranu as part of a graduate course on network analysis at the University of Amsterdam (2020).

References

Epskamp, S., Haslbeck, J. M. B., Isvoranu, A. M., & Van Borkulo, C. D. (2022). Pairwise Markov random fields. In A. M. Isvoranu, S. Epskamp, L. J. Waldorp, & D. Borsboom (Eds.), Network psychometrics with R: A guide for behavioral and social scientists (pp. 95–118). Routledge.

Examples

```
data(NA2020)


# Estimate a GGM using FIML (as in Chapter 6):
library(dplyr)
mod <- ggm(NA2020, estimator = "FIML") %>% runmodel

# Prune non-significant edges:
mod_pruned <- mod %>% prune(alpha = 0.05)

# Inspect parameters:
mod_pruned %>% parameters
```

panelvar *Lag-1 panel vector autoregression family of psychonetrics models*

Description

This is the family of models that models a lag-1 vector autoregression (VAR) process with random intercepts on panel data of observed variables: a within-person stationary VAR(1) process (temporal matrix `beta` and contemporaneous structure `sigma_zeta_within`) combined with a between-person (random intercept) structure (`sigma_zeta_between`). The `panelvar` function models the contemporaneous and between-person structures as variance–covariance matrices by default, and the `panelgvar` wrapper models them as Gaussian graphical models (GGMs), yielding the panel graphical VAR model.

This model family is the observed-variable special case of `dlvm1` (a factor loading matrix fixed to identity, zero residual variances and observed means). Since psychonetrics 0.16.2 it is implemented as its own model framework: results are identical to routing through `dlvm1` with dummy matrices, but estimation skips all factor-loading algebra and is faster. The mean structure is a single `mu` matrix of observed stationary means (in older versions these appeared as `nu`); with `stationary_means = FALSE` `mu` instead holds wave-specific means, one per variable per wave (the trend-stationary case, see the details below). Latent panel models (`lambda` not identity) are still handled by `dlvm1` and its wrapper `panellvgvar`.

All functions accept both wide-format data (with a design matrix for `vars`) and long-format data (with a character vector for `vars` plus `idvar` and optionally `beepvar`). The format is auto-detected from the type of `vars`.

Usage

```
panelvar(data, vars, within_latent = c("cov", "chol", "prec",
                    "ggm", "cor"), between_latent = c("cov", "chol",
                    "prec", "ggm", "cor"), datatype = c("auto", "wide",
                    "long"), idvar, beepvar, standardize = c("none",
                    "z", "quantile", "z_per_wave"), beta = "full",
                    omega_zeta_within = "full", delta_zeta_within =
                    "diag", kappa_zeta_within = "full",
                    sigma_zeta_within = "full", lowertri_zeta_within =
                    "full", omega_zeta_between = "full",
                    delta_zeta_between = "diag", kappa_zeta_between =
                    "full", sigma_zeta_between = "full",
                    lowertri_zeta_between = "full", mu, groups,
                    groupvar, covs, cors, means, nobs, corinput, start
                    = "version2", covtype = c("choose", "ML", "UB"),
                    missing = "auto", equal = "none",
                    baseline_saturated = TRUE, estimator = "ML",
                    optimizer, storedata = FALSE, verbose = FALSE,
                    sampleStats, baseline =
                    c("stationary_random_intercept", "stationary",
                    "independence", "none"), bootstrap = FALSE, boot_sub,
```

```
                    boot_resample, within, between,
                    penalty_lambda = NA, penalty_alpha = 1,
                    penalize_matrices,
                    rho_zeta_within = "full", SD_zeta_within = "diag",
                    rho_zeta_between = "full", SD_zeta_between = "diag",
                    temporal = c("raw","PDC"), PDC = "full",
                    stationary_means = TRUE)

panelgvar(data, vars, within_latent = c("ggm","chol","cov","prec","cor"),
          between_latent = c("ggm","chol","cov","prec","cor"), ...)
```

Arguments

`data`	A data frame encoding the data used in the analysis. Can be missing if `covs` and `nobs` are supplied.
`vars`	Required argument. For wide-format data, this must be a *matrix* with each row indicating a variable and each column indicating a measurement. The matrix must be filled with names of the variables in the dataset corresponding to variable i at wave j. NAs can be used to indicate missing waves. The rownames of this matrix will be used as variable names. For long-format data, this should be a character vector of variable names in the dataset.
`within_latent`	The type of within-person contemporaneous model to be used. Since there are no latent variables in panelvar, the shorter alias `within` is preferred.
`between_latent`	The type of between-person model to be used. Since there are no latent variables in panelvar, the shorter alias `between` is preferred.
`datatype`	Data format: `"auto"` (default) auto-detects based on whether `vars` is a matrix (wide) or character vector (long). `"wide"` forces wide-format interpretation. `"long"` forces long-format interpretation with automatic reshape to wide.
`idvar`	Optional string indicating the subject/cluster ID variable in `data`. Required when `datatype = "long"` or when `vars` is a character vector.
`beepvar`	Optional string indicating the time point / measurement occasion variable. If missing when `datatype = "long"`, a sequential measurement variable is created per subject.
`standardize`	Standardization method: `"none"` (default), `"z"` for global z-scores per variable across all waves (uses the overall mean and standard deviation for each variable computed across all waves of data), `"quantile"` for quantile normalization across all waves, `"z_per_wave"` for independent z-scores per wave column (note: this imposes stationarity as means become 0 and variances become 1 at each wave).
`beta`	A model matrix encoding the temporal relationships (transpose of temporal network). A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Can also be `"full"` for a full temporal network or `"zero"` for an empty temporal network.

omega_zeta_within
: Only used when `within_latent = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

delta_zeta_within
: Only used when `within_latent = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

kappa_zeta_within
: Only used when `within_latent = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

sigma_zeta_within
: Only used when `within_latent = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

lowertri_zeta_within
: Only used when `within_latent = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

omega_zeta_between
: Only used when `between_latent = "ggm"`. Can be `"full"`, `"zero"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

delta_zeta_between
: Only used when `between_latent = "ggm"`. Can be `"diag"`, `"zero"` (not recommended), or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

kappa_zeta_between
: Only used when `between_latent = "prec"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_zeta_between`
: Only used when `between_latent = "cov"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_zeta_between`
: Only used when `between_latent = "chol"`. Can be `"full"`, `"diag"`, or a typical model matrix with 0s indicating parameters constrained to zero, 1s indicating free parameters, and higher integers indicating equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`mu`
: Optional vector encoding the observed stationary means. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free means, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector. When `stationary_means = FALSE` this is instead a `nVar` by `nTime` matrix (variables in the rows, waves in the columns), or an array/list of such matrices for multiple groups; elements at variable–wave combinations that are not in the `vars` design matrix are always fixed to zero.

`groups`
: Deprecated. Use `groupvar` instead. An optional string indicating the name of the group variable in `data`.

`groupvar`
: An optional string indicating the name of the group variable in `data`. Replaces the deprecated `groups` argument; if both are supplied, `groupvar` takes precedence with a warning.

`covs`
: A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure the `covtype` argument is set correctly to the type of covariances used.

`cors`
: Not supported in `panelvar()`; supplying correlation input produces an error.

`means`
: A vector of sample means, or a list/matrix containing such vectors for multiple groups. This argument always refers to the sample means; the mean *structure* of the model is set with `stationary_means`.

`nobs`
: The number of observations used in `covs` and `means`, or a vector of such numbers of observations for multiple groups.

`corinput`
: Logical. Not supported in `panelvar()` and will produce an error if set to `TRUE`.

`start`
: Start value specification. Can be either a string or a psychonetrics model. If it is a string, `"version2"` indicates the latest version of start value computation, `"version1"` indicates start values as they were computed up to version 0.11, and `"simple"` indicate simple starting values. If this is a psychonetrics model the starting values will be based on the output. This can be useful, for example, if you first estimate a model with matrices set to a Cholesky decomposition, then use those values as start values for estimating Gaussian graphical models.

`covtype`
: If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to `"ML"` for maximum likelihood estimates (denominator n) and `"UB"` to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.

`missing` How should missingness be handled in computing the sample covariances and number of observations when `data` is used. Can be `"auto"` (default) for automatic detection, `"listwise"` for listwise deletion, or `"pairwise"` for pairwise deletion. When `"auto"`, the function checks for missing data and switches ML to FIML, PML to PFIML, or defaults to listwise for LS estimators.

`equal` A character vector indicating which matrices should be constrained equal across groups.

`baseline_saturated`
A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

`estimator` The estimator to be used. Currently implemented are `"ML"` for maximum likelihood estimation, `"FIML"` for full-information maximum likelihood estimation, `"PML"` for penalized maximum likelihood estimation, `"PFIML"` for penalized full-information maximum likelihood estimation, `"ULS"` for unweighted least squares estimation, `"WLS"` for weighted least squares estimation, and `"DWLS"` for diagonally weighted least squares estimation. When `missing = "auto"` (default), `"ML"` is automatically switched to `"FIML"` and `"PML"` to `"PFIML"` if missing data is detected.

`optimizer` The optimizer to be used. Can be one of `"nlminb"` (the default R `nlminb` function), `"ucminf"` (from the `optimr` package), `"nloptr_TNEWTON"` (preconditioned truncated Newton via `nloptr`), and `"LBFGS++"` (pure C++ L-BFGS-B). Defaults to `"nlminb"`.

`storedata` Logical, should the raw data be stored? Needed for bootstrapping (see `bootstrap`).

`verbose` Logical, should progress be printed to the console?

`sampleStats` An optional sample statistics object. Mostly used internally.

`baseline` What baseline model should be used? `"stationary_random_intercept"` includes both within- and between person variances constrained equal across time (default), `"stationary"` only includes within-person variances constrained equal across time, `"independence"` (default up to version 0.11) includes a variance for every variable at every time point (not constrained equal across time), and `"none"` includes no baseline model. The baseline model always uses the same mean structure as the model itself (see the `stationary_means` argument and the details below).

`bootstrap` Should the data be bootstrapped? If `TRUE` the data are resampled and a bootstrap sample is created. These must be aggregated using `aggregate_bootstraps`! Can be `TRUE` or `FALSE`. Can also be `"nonparametric"` (which sets `boot_sub = 1` and `boot_resample = TRUE`) or `"case"` (which sets `boot_sub = 0.75` and `boot_resample = FALSE`).

`boot_sub` Proportion of cases to be subsampled (`round(boot_sub * N)`).

`boot_resample` Logical, should the bootstrap be with replacement (`TRUE`) or without replacement (`FALSE`)

`within` Preferred name for `within_latent` (there are no latent variables in panelvar). If both `within` and `within_latent` are explicitly specified, `within_latent` takes precedence and a warning is issued.

`between` Preferred name for `between_latent` (there are no latent variables in panelvar). If both `between` and `between_latent` are explicitly specified, `between_latent` takes precedence and a warning is issued.

`penalty_lambda` Numeric penalty strength for penalized ML estimation (PML/PFIML). `NA` (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See `find_penalized_lambda` and `penalize`.

`penalty_alpha` Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

`penalize_matrices`
Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.

`rho_zeta_within`
Only used when `within_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix; see `omega_zeta_within`).

`SD_zeta_within` Only used when `within_latent = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix; see `delta_zeta_within`).

`rho_zeta_between`
Only used when `between_latent = "cor"`. Correlation matrix specification (`"full"`, `"zero"`, or a pattern matrix).

`SD_zeta_between`
Only used when `between_latent = "cor"`. Diagonal standard-deviation matrix specification (`"diag"`, `"full"`, or a pattern matrix).

`temporal` Parameterization of the temporal effects: `"raw"` (default) models `beta` directly; `"PDC"` models the partial directed correlations directly, with the `PDC` matrix encoding from = row and to = column (the transposed orientation relative to `beta`, matching `getmatrix(..., "PDC")`). A (pattern-constrained) PDC model is fit-equivalent to the corresponding raw model, but standard errors, confidence intervals, equality constraints and modification indices then apply directly on the PDC scale.

`PDC` Only used when `temporal = "PDC"`. Model matrix for the partial directed correlations (from = row, to = column): `"full"`, `"zero"`, `"diag"`, or a pattern matrix with 0s indicating elements fixed to zero, 1s indicating free elements, and higher integers indicating equality constraints.

`stationary_means`
Logical. `TRUE` (default, and the behavior of all earlier versions) models a single stationary mean per variable, constrained equal across waves. `FALSE` models a separate mean for every variable at every wave (the `mu` matrix becomes `nVar` by `nTime`), which saturates the mean structure and adds `nVar * (nTime - 1)` parameters. See the details below.

`...` Arguments sent to `panelvar`.

Details

Stationary versus wave-specific means. By default the panel VAR model is fully stationary: every variable has a single mean, and the implied means are therefore equal across waves. With `stationary_means = FALSE` the means become wave-specific, which is the classical *trend-stationary*

case. The level shifts are then period effects that are exogenous to the dynamics: they are deterministic offsets added to all cases at a given wave, the within-person deviation process

$$y_{i,t} = \mu_t + \eta_{i,t} + \zeta_i, \quad \eta_{i,t} = B\eta_{i,t-1} + \varepsilon_{i,t}$$

remains covariance-stationary, and wave 1 is still drawn from that stationary distribution. Consequently the implied covariance matrix is *identical* to that of the stationary-means model, and only the mean part of the model (and of its Jacobian) changes. Fitting `stationary_means = FALSE` to raw data is equivalent to fitting the default model to per-wave mean-centered data, except that the free means are estimated (and tested) rather than removed, and the degrees of freedom are adjusted accordingly.

Identification. The mean structure under `stationary_means = FALSE` is exactly saturated: one free mean per variable–wave combination present in the `vars` design matrix (variable–wave combinations that are absent have no mean parameter). The estimated means therefore equal the observed per-wave means, and the likelihood-ratio test of `stationary_means = TRUE` against `FALSE` is a test of equal means over time with `nVar * (nTime - 1)` degrees of freedom. Cross-group equality constraints (`equal = "mu"`) equate the means *per wave* across groups.

No free innovation variances. Only the means can be freed over time in this framework, not the innovation (co)variances: a wave-varying innovation covariance has no stationary distribution for wave 1 to be drawn from, so the Kronecker/stationarity machinery on which `panelvar` is built does not apply. Models with wave-specific innovation structures are covered by the RI-CLPM / RI-CLPN framework instead, see `ri_clpm` (which also leaves the exogenous wave-1 covariance block free).

Baseline model. The baseline model always uses the same mean structure as the fitted model: with `stationary_means = FALSE` the baseline gets wave-specific means as well. A baseline model must be nested in the fitted model, otherwise it can fit better than the fitted model on the mean side (a free-means baseline against an equal-means target would beat it), which makes incremental fit indices meaningless or negative. A corollary is that incremental fit indices (CFI, TLI, NFI, ...) are **not** directly comparable between a `stationary_means = TRUE` and a `stationary_means = FALSE` fit of the same data, because the two use different baselines by design; compare those two models with a likelihood-ratio test (`compare`) instead. The saturated model already has a saturated mean structure and is unchanged.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

`dlvm1` for the latent variable generalization of this model family.

Examples

```
library("dplyr")

# Smoke data cov matrix, based on LISS data panel https://www.dataarchive.lissdata.nl
```

```
smoke <- structure(c(47.2361758611759, 43.5366809116809, 41.0057465682466,
                     43.5366809116809, 57.9789886039886, 47.6992521367521,
                     41.0057465682466,
                     47.6992521367521, 53.0669434731935), dim = c(3L, 3L),
                   dimnames = list(
                       c("smoke2008", "smoke2009", "smoke2010"), c("smoke2008",
                   "smoke2009", "smoke2010")))

# Design matrix:
design <- matrix(rownames(smoke),1,3)

# Form model:
mod <- panelvar(vars = design,
                covs = smoke, nobs = 352
)


# Run model:
mod <- mod %>% runmodel

# Evaluate fit:
mod %>% fit


# panelgvar and panelvar also accept long-format data.
# For long-format data, pass vars as a character vector
# and provide idvar (and optionally beepvar):

# Simulate some long-format panel data:
longdata <- data.frame(
  id = rep(1:50, each = 3),
  time = rep(1:3, 50),
  x = rnorm(150),
  y = rnorm(150)
)

# Use panelgvar with long-format data:
mod_long <- panelgvar(data = longdata,
                      vars = c("x", "y"),
                      idvar = "id",
                      beepvar = "time")

# The same model with wave-specific (trend-stationary) means, which can be
# compared to the stationary-means model with a chi-square difference test:
mod_free <- panelgvar(data = longdata,
                      vars = c("x", "y"),
                      idvar = "id",
                      beepvar = "time",
                      stationary_means = FALSE)
```

`parameters` *Print parameter estimates*

Description

This function will print a list of parameters of the model

Usage

```
parameters(x, standardized = FALSE)
```

Arguments

`x`	A `psychonetrics` model.
`standardized`	Logical. If `TRUE`, the completely standardized (`std.all`) solution is computed and two extra columns are added next to the unstandardized estimate: `std` (the standardized point estimate) and `se_std` (its standard error, obtained by the delta method). Standardization rescales every variable to unit variance using the model-implied standard deviations, so that loadings, regression coefficients and (co)variances are expressed on a standardized scale (covariances become correlations; residual/disturbance variances become proportions of variance; residual/disturbance covariances become residual correlations), matching `lavaan::standardizedSolution(type = "std.all")` (Rosseel, 2012). Currently implemented for the `lvm` and `varcov` model families only; for all other families the `std`/`se_std` columns are returned as `NA` (with a warning). Default `FALSE` (unstandardized estimates only, exactly as before).

Details

The standardized standard errors are obtained post-fit by differentiating the standardization map with respect to the free parameters (`numDeriv::jacobian`) and applying the delta method to the parameter covariance matrix; a fixed parameter (e.g. a marker loading) has `se_std = NA` because it has no sampling variance.

Value

Invisibly returns a data frame containing information on all parameters (including the `std` and `se_std` columns when `standardized = TRUE`).

Author(s)

Sacha Epskamp

References

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

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars, omega = "zero")

# Run model:
mod <- mod %>% runmodel

# Parameter estimates:
mod %>% parameters

# Standardized (std.all) solution with standard errors:
mod %>% parameters(standardized = TRUE)
```

---

parequal *Set equality constraints across parameters*

Description

This function can be used to set parameters equal

Usage

```
parequal(x, ..., inds = integer(0), verbose, log = TRUE,
                    runmodel = FALSE)
```

Arguments

x	A psychonetrics model.
...	Vectors of parameter numbers to constrain equal (combined with inds)
inds	Parameter indices of parameters to be constrained equal
verbose	Logical, should messages be printed?
log	Logical, should the log be updated?
runmodel	Logical, should the model be updated?

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

`partialprune` *Partial pruning of multi-group models*

Description

This function will search for a multi-group model with equality constraints on some but not all parameters. This is called partial pruning (Epskamp, Isvoranu, & Cheung, 2020; Haslbeck, 2020). The algorithm is as follows: 1. remove all parameters not significant at alpha in all groups (without equality constraints), 2. create a union model with all parameters included in any group included in all groups and constrained equal. 3. Stepwise free equality constraints of the parameter that features the largest sum of modification indices until BIC can no longer be improved. 4. Select and return (by default) the best model according to BIC (original model, pruned model, union model and partially pruned model).

Usage

```
partialprune(x, alpha = 0.01, matrices, verbose,
             combinefun = c("unionmodel", "intersectionmodel", "identity"),
             return = c("partialprune","best","union_equal","prune"),
             criterion = "bic",  best = c("lowest","highest"),
             final_prune = c("saturated","partialprune"),
             useMIs = c("joint","simple"),
             release_model = c("pruned","saturated"),
             identify = TRUE, ...)
```

Arguments

`x`	A `psychonetrics` model.
`alpha`	Significance level to use.
`matrices`	Vector of strings indicating which matrices should be pruned. Will default to network structures.
`verbose`	Logical, should messages be printed?
`combinefun`	Function used to combine models of different groups.
`return`	What model to return? `"best"` for best fitting model (according to BIC), `"partialprune"` for the partialpruned model, `"union_equal"` for the union model with equality constraints, and `"prune"` for the originally pruned model without equality constraints.

best	Should the lowest or the highest index of criterion be used to select the final model?
criterion	What criterion to use for the model selection in the last step? Defaults to "bic" for BIC selection.
final_prune	Should the last prune step be based on removing edges not significant in the last model in the partialprune algorithm or the first model (saturated model) in the algorithm? Defaults to "saturated". Set to "partialprune" to mimic psychonetrics < 0.13.1 behavior.
useMIs	How to rank candidate equality constraints in the stepwise release loop. The default "joint" (introduced in 0.15.4) uses the joint multivariate score / Lagrange-multiplier test (one statistic per equality-constrained parameter, df = G - 1; see equalityScoreTest). The legacy "simple" option sums the per-group univariate modification indices instead, which reproduces the behavior of psychonetrics <= 0.15.3. For two-group models the two options are equivalent; they differ for G >= 3.
release_model	Specifies the alternative model used in the BIC comparison when deciding whether to release an equality constraint in the stepwise loop. With the default "pruned" (new in 0.15.5), the just-released parameter is immediately fixed to zero in any group where the initial per-group prune (Step 1) had already removed it, BEFORE refitting and comparing BIC. The equality-constrained model is therefore tested against the asymmetric structure suggested by the initial per-group prune, rather than against a fully saturated per-group alternative. This improves specificity at low sample sizes by penalizing the alternative less heavily for spurious cross-group differences. Set to "saturated" to reproduce the psychonetrics 0.15.4 behavior in which the released parameter is freed in all groups with no per-group zero pattern applied until the final prune step.
identify	Logical, passed to the internal prune, unionmodel, intersectionmodel, groupequal, and groupfree calls. When FALSE, identify is not invoked on the intermediate models, so user-fixed parameters (e.g. fixpar("beta", 1, 1, value = 1) in a multi-group Ising model) are not re-freed when equality constraints are introduced on matrices. Defaults to TRUE, which preserves the historical behavior.
...	Arguments sent to prune.

Value

An object of the class psychonetrics (psychonetrics-class); which model is returned is controlled by the return argument (by default the model selected via the chosen criterion).

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

References

Epskamp, S., Isvoranu, A. M., & Cheung, M. (2020). Meta-analytic gaussian network aggregation. PsyArXiv preprint. DOI:10.31234/osf.io/236w8.

Haslbeck, J. (2020). Estimating Group Differences in Network Models using Moderation Analysis. PsyArXiv preprint. DOI:10.31234/osf.io/926pv.

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the extroversion items, and gender:
ExData <- bfi %>%
  select(E1:E5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ExData)[1:5]

# Saturated estimation:
mod_saturated <- ggm(ExData,
                     vars = vars,
                     groups = "gender")

# Partial prune model:
mod_partial <- mod_saturated %>%
  partialprune

# Obtain the networks:
getmatrix(mod_partial, "omega")

# Differences:
getmatrix(mod_partial, "omega")[[1]] -
  getmatrix(mod_partial, "omega")[[2]]

# Difference detected in edge 4 - 5
```

`penalize` *Penalized Maximum Likelihood Functions*

Description

Functions for penalized maximum likelihood (PML) and penalized full-information maximum likelihood (PFIML) estimation. `penalize` adds an L1 (LASSO) or elastic net penalty to specified model parameters. `unpenalize` removes the penalty. `penaltyVector` extracts the per-free-parameter penalty strengths. `refit` performs post-selection inference by fixing penalized zeros and re-running with standard ML (or FIML for PFIML) estimation to obtain standard errors.

Usage

```
penalize(x, matrix, row, col, lambda, group,
```

```
         what = c("free", "fixed"), verbose, log = TRUE)
unpenalize(x, matrix, row, col, group, verbose, log = TRUE)
penaltyVector(x)
refit(x, threshold = 1e-8, verbose, log = TRUE, ...)
```

Arguments

x	A psychonetrics model.
matrix	Character vector of matrix name(s) to penalize/unpenalize. If missing, defaults to the matrices returned by defaultPenalizeMatrices(x) for penalize, or all currently penalized matrices for unpenalize.
row	Integer or character indicating specific row(s) of the matrix. If missing, the entire matrix is penalized/unpenalized.
col	Integer or character indicating specific column(s) of the matrix. If missing, the entire matrix is penalized/unpenalized.
lambda	The penalty strength. Can be a scalar (applied to all selected parameters) or a matrix with dimensions matching the model matrix. When a matrix is provided: 0 means do not penalize, NA means auto-select via EBIC, and a positive number means use that fixed penalty strength. If missing, uses the global x@penalty$lambda.
group	Integer indicating specific group(s). If missing, all groups are included.
what	Character, either "free" (default) or "fixed". Controls which parameters are targeted: "free" penalizes parameters that are already free; "fixed" first frees parameters that are currently fixed to zero, then penalizes them (starting value set to 0). Parameters fixed to non-zero values (e.g., marker loadings fixed to 1) are never affected by what = "fixed". When using a matrix lambda, mismatched parameters are silently skipped. When using a scalar lambda with specific row/col, a warning is issued for mismatched parameters.
verbose	Logical, should messages be printed?
log	Logical, should the log be updated?
threshold	For refit: threshold below which penalized parameters are considered zero and fixed.
...	Additional arguments passed to runmodel in refit.

Details

Penalized ML estimation uses the "PML" estimator, which wraps the standard ML fit function and gradient with an additional penalty term. For models with missing data, the "PFIML" estimator wraps the FIML fit function and gradient with the same penalty. The ISTA (Iterative Shrinkage-Thresholding Algorithm) proximal gradient optimizer is used automatically for both PML and PFIML models.

For symmetric matrices (e.g., omega), only off-diagonal elements are penalized by default. For non-symmetric matrices (e.g., beta), all elements are penalized.

The what argument is particularly useful for regularized confirmatory factor analysis (CFA). For example, starting from a simple-structure CFA model, what = "fixed" can free and penalize all cross-loadings, allowing LASSO to select which cross-loadings to retain.

The `refit` function is used for post-selection inference: after a penalized model is estimated, `refit` fixes parameters that were shrunk to zero and re-estimates the model with standard ML (for PML) or FIML (for PFIML) to obtain valid standard errors and fit indices.

Model constructors (e.g., `ggm`, `lvm`, `var1`) accept `penalty_lambda`, `penalty_alpha`, and `penalize_matrices` arguments for convenient PML/PFIML setup. When `missing = "auto"` (the default), specifying `estimator = "PML"` with missing data will automatically switch to `"PFIML"`.

Value

`penalize`, `unpenalize`, and `refit` return an object of class `psychonetrics` (psychonetrics-class). `penaltyVector` returns a numeric vector of penalty strengths per free parameter.

Author(s)

Sacha Epskamp

See Also

`find_penalized_lambda` for automatic lambda selection via EBIC grid search, `runmodel` for running models (calls `find_penalized_lambda` automatically when `penalty_lambda = NA`), `ggm` and other model constructors for the `penalty_lambda`, `penalty_alpha`, and `penalize_matrices` arguments.

Examples

```
# Load bfi data from psych package:
library("psychTools")
library("dplyr")
data(bfi)

ConsData <- bfi %>%
  select(A1:A5) %>%
  na.omit

vars <- names(ConsData)

# Penalized GGM estimation with manual lambda:
mod <- ggm(ConsData, vars = vars, estimator = "PML", penalty_lambda = 0.1)
mod <- mod %>% runmodel

# Post-selection refit for standard errors:
mod_refit <- mod %>% refit
mod_refit %>% parameters

# Manual unpenalize: free a specific edge from the penalty:
mod2 <- ggm(ConsData, vars = vars, estimator = "PML", penalty_lambda = 0.1)
mod2 <- unpenalize(mod2, "omega", row = 1, col = 2)
mod2 <- mod2 %>% runmodel
```

`plot.psychonetrics` *Plot the networks of a psychonetrics model*

Description

A wrapper around `qgraph` that draws one or more model matrices of a `psychonetrics` model as networks, in the spirit of the plot method for `estimateNetwork` objects in the `bootnet` package: the same overwritten defaults (`cut = 0`, `theme = "colorblind"`, `layout = "spring"`, `parallelEdge = TRUE`) and everything else passed on to `qgraph` through `...`. Which matrices are plotted, and how, is derived from the model, so that a plain `plot(mod)` gives a sensible picture for every model family.

Usage

```
## S3 method for class 'psychonetrics'
plot(x, matrix, ..., time, group,
     threshold = FALSE, alpha = 0.01,
     adjust = c("none", "holm", "hochberg", "hommel",
                "bonferroni", "BH", "BY", "fdr"),
     mode = c("tested", "all"),
     covariances = FALSE, labels, layout = "spring",
     cut = 0, theme = "colorblind", parallelEdge = TRUE,
     titles = TRUE, equalLayout = TRUE, verbose = TRUE)
```

Arguments

`x`	A `psychonetrics` model.
`matrix`	A character vector of matrices to plot; each element becomes a panel, in the given order. Unlike `getmatrix`, more than one matrix may be given. When omitted, a default is chosen from the model family (see Details).
`...`	Arguments passed to `qgraph` (or to `qgraph.loadings` for a factor loading matrix, or to `semPaths` for a latent variable model). Anything given here overrides the defaults set by this method.
`time`	Only for `ri_clpm` / `ri_clpn` models: the wave whose block should be plotted, either an integer or a column name of the design matrix. When omitted, every wave gets its own panel. See `getmatrix`.
`group`	The group(s) to plot, as labels or numbers. Defaults to all groups, one row of panels per group.
`threshold`	Logical (or a numeric absolute threshold), as in `getmatrix`: should non-significant edges be set to zero?
`alpha`	Significance level used when `threshold = TRUE`.
`adjust`	p-value adjustment method used when `threshold = TRUE`. See `p.adjust`.
`mode`	Mode for adjusting for multiple comparisons, as in `getmatrix`.

covariances	Logical. By default, covariance matrices are drawn as correlation networks and precision matrices as partial correlation networks, so that the edge widths are comparable across differently scaled variables. Set to TRUE to draw the unstandardized matrix instead, with self-loops showing the diagonal (the variances, respectively the diagonal of the precision matrix).
labels	Node labels. Defaults to the variable names of the model. May be a character vector (used for every panel) or a list with one element per panel.
layout	The qgraph layout. Defaults to "spring".
cut	The qgraph cut value. Defaults to 0 (no cut), as in bootnet.
theme	The qgraph theme. Defaults to "colorblind".
parallelEdge	Should reciprocal directed edges be drawn as parallel lines? Defaults to TRUE.
titles	TRUE (default) to title each panel with what it shows, FALSE for no titles, or a character vector with one title per panel.
equalLayout	Logical. When several panels share the same nodes, should they be constrained to a common layout (through averageLayout) so that they are visually comparable? Defaults to TRUE. Ignored when layout is given as a matrix.
verbose	Logical, should messages about how a matrix is being drawn be printed?

Details

Default matrices. When matrix is omitted, the matrices are chosen from the model family, in the parameterization that was estimated:

- varcov family (ggm, precision, corr, ...): the (co)variance matrix itself, one panel.
- Ising and BlumeCapel: the network omega, one panel.
- var1 / gvar: two panels, the temporal and the contemporaneous network.
- panelvar / panelgvar, dlvm1, ml_var1 and ml_lvm: three panels, the temporal, the contemporaneous (within-person) and the between-person network.
- tsdlvm1: the temporal and the contemporaneous network.
- ml_varcov: the within-person and the between-person network.
- meta_varcov: the pooled (co)variance matrix.
- ri_clpm / ri_clpn: the temporal, contemporaneous and between-person *blocks* (see getmatrix); one panel per wave, with waves that were equated by ri_clpm_stationary collapsed into one.
- lnm, rnm and lrnm: the latent and/or residual network.
- Any other lvm model has no natural network representation, so the full path diagram is drawn with semPaths from the semPlot package (via tolavaan). Give a matrix to plot a model matrix as a network instead.

If the model uses the "PDC" temporal parameterization, the temporal panel shows PDC rather than beta.

How each matrix is drawn. The type of graph follows from the matrix:

- omega (Gaussian graphical model): an undirected network of partial correlations.

- kappa (precision): an undirected network. Standardized to partial correlations unless covariances = TRUE.
- sigma (covariance): a network with two arrowheads per edge. Standardized to correlations unless covariances = TRUE, in which case the variances are added as self-loops.
- rho (correlation): a network with two arrowheads per edge.
- beta and PDC (temporal): a directed network, with the autoregressive effects drawn as self-loops.
- lambda (factor loadings): a reflective measurement diagram, drawn with qgraph.loadings.
- lowertri and exo_cholesky (Cholesky decompositions) have no network interpretation and raise an error; plot the corresponding (co)variance or partial correlation matrix instead.
- Diagonal scaling matrices (delta, SD), mean vectors (mu, nu, tau) and the Ising inverse temperature (beta) are not networks and raise an error.

Value

Invisibly, the qgraph object of the plot, or a named list of such objects when more than one panel is drawn.

Author(s)

Sacha Epskamp

See Also

getmatrix, CIplot, qgraph

Examples

```
# Simulate data from a chain graph:
set.seed(1)
omega <- matrix(0, 5, 5)
for (i in 1:4) omega[i, i + 1] <- omega[i + 1, i] <- 0.35
Sigma <- cov2cor(solve(diag(5) - omega))
data  <- as.data.frame(MASS::mvrnorm(500, rep(0, 5), Sigma))
names(data) <- paste0("V", 1:5)

# Fit a GGM and plot it:
mod <- runmodel(ggm(data))
plot(mod)

# Only the significant edges:
plot(mod, threshold = TRUE)

# The same data as a covariance model: drawn as a correlation network ...
mod2 <- runmodel(varcov(data, type = "cov"))
plot(mod2)
# ... or as covariances, with the variances as self-loops:
plot(mod2, covariances = TRUE)
```

```
# qgraph arguments are passed through:
plot(mod, layout = "circle", vsize = 12, labels = paste0("x", 1:5))
```

---

prune *Stepdown model search by pruning non-significant parameters*

Description

This function will (recursively) remove parameters that are not significant and refit the model.

Usage

```
prune(x, alpha = 0.01, adjust = c("none", "holm",
                    "hochberg", "hommel", "bonferroni", "BH", "BY",
                    "fdr"), matrices, runmodel = TRUE, recursive = FALSE,
                    verbose, log = TRUE, identify = TRUE, startreduce = 1,
                    limit = Inf, mode = c("tested","all"), ...)
```

Arguments

x	A psychonetrics model.
alpha	Significance level to use.
adjust	p-value adjustment method to use. See p.adjust.
matrices	Vector of strings indicating which matrices should be pruned. Will default to network structures.
runmodel	Logical, should the model be evaluated after pruning?
recursive	Logical, should the pruning process be repeated?
verbose	Logical, should messages be printed?
log	Logical, should the log be updated?
identify	Logical, should models be identified automatically?
startreduce	A numeric value indicating a factor with which the starting values should be reduced. Can be useful when encountering numeric problems.
limit	The maximum number of parameters to be pruned.
mode	Mode for adjusting for multiple comparisons. Should all parameters be considered as the total number of tests or only the tested parameters (parameters of interest)?
...	Arguments sent to runmodel

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

stepup

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars, omega = "full")

# Run model:
mod <- mod %>% runmodel

# Prune model:
mod <- mod %>% prune(adjust = "fdr", recursive = FALSE)
```

---

psychonetrics-class *Class* "psychonetrics"

Description

Main class for psychonetrics results.

Objects from the Class

Objects can be created by calls of the form `new("psychonetrics", ...)`.

Slots

`model`: Object of class `"character"`: the model framework (e.g. `"varcov"`, `"lvm"`, `"var1"`).

`submodel`: Object of class `"character"`: the submodel/type within the framework (e.g. `"ggm"`).

`parameters`: Object of class `"data.frame"`: the parameter table (one row per model parameter, with estimates, standard errors, matrix/row/col indices and parameter numbers).

`matrices`: Object of class `"data.frame"`: bookkeeping table of the model matrices and their dimensions.

`meanstructure`: Object of class `"logical"`: whether a mean structure is included in the model.

`computed`: Object of class `"logical"`: whether the model has been estimated (run).

`sample`: Object of class `"psychonetrics_samplestats"`: the sample statistics used in estimation.

`modelmatrices`: Object of class `"list"`: the model-implied matrices in list form (per group).

`log`: Object of class `"psychonetrics_log"`: the logbook of actions performed on the model.

`optim`: Object of class `"list"`: the raw output of the optimizer.

`fitmeasures`: Object of class `"list"`: computed fit measures.

`baseline_saturated`: Object of class `"list"`: the baseline and saturated models used for fit computation.

`equal`: Object of class `"character"`: names of matrices constrained equal across groups (legacy representation; the parameters table is authoritative).

`objective`: Object of class `"numeric"`: the value of the fit function at the solution.

`information`: Object of class `"matrix"`: the Fisher information matrix.

`identification`: Object of class `"character"`: the identification method used (e.g. `"loadings"` or `"variance"`).

`optimizer`: Object of class `"character"`: the optimizer used.

`optim.args`: Object of class `"list"`: arguments passed to the optimizer.

`estimator`: Object of class `"character"`: the estimator used (e.g. `"ML"`, `"FIML"`).

`distribution`: Object of class `"character"`: the assumed distribution (e.g. `"Gaussian"`).

`extramatrices`: Object of class `"list"`: additional helper matrices used internally.

`rawts`: Object of class `"logical"`: whether raw time-series data are used.

`Drawts`: Object of class `"list"`: duplication/derivative matrices for raw time-series estimation.

`types`: Object of class `"list"`: the type of each model matrix per group.

`cpp`: Object of class `"logical"`: whether the C++ backend is used.

`verbose`: Object of class `"logical"`: whether progress messages are printed.

`penalty`: Object of class `"list"`: the penalty configuration for penalized (PML/PFIML) estimation, holding the penalty strength (`lambda`) and elastic-net mixing parameter (`alpha`).

Methods

resid `signature(object = "psychonetrics")`: ...

residuals `signature(object = "psychonetrics")`: ...

show `signature(object = "psychonetrics")`: ...

summary `signature(object = "psychonetrics")`: The print output extended with the main fit measures (log-likelihood, number of parameters, AIC, BIC, RMSEA, CFI, TLI). Use `fit` for the complete list of fit measures.

Author(s)

Sacha Epskamp

Examples

```
showClass("psychonetrics")
```

---

psychonetrics_bootstrap-class

Class "psychonetrics_bootstrap"

Description

Class for aggregated bootstrap results.

Objects from the Class

Objects can be created by calls of the form `new("psychonetrics_bootstrap", ...)`.

Slots

model: Object of class "character" ~~

submodel: Object of class "character" ~~

parameters: Object of class "data.frame" ~~

models: Object of class "list" ~~

matrices: Object of class "data.frame" ~~

fitmeasures: Object of class "data.frame" ~~

distribution: Object of class "character" ~~

verbose: Object of class "logical" ~~

boot_sub: Object of class "numeric" ~~

boot_resample: Object of class "logical" ~~

n_fail: Object of class "numeric" ~~

n_success: Object of class "numeric" ~~

types: Object of class "list" ~~

Methods

show `signature(object = "psychonetrics_bootstrap")`: ...

Author(s)

Sacha Epskamp

Examples

```
showClass("psychonetrics_bootstrap")
```

psychonetrics_log-class *Class* "psychonetrics_log"

Description

A logbook entry in the psychonetrics logbook

Objects from the Class

Objects can be created by calls of the form `new("psychonetrics_log", ...)`.

Slots

`event`: Object of class "`character`": a description of the logged event.

`time`: Object of class "`POSIXct`": the time at which the event was logged.

`sessionInfo`: Object of class "`sessionInfo`": the R session information at the time of the event.

Methods

show `signature(object = "psychonetrics_log")`: ...

Author(s)

Sacha Epskamp

Examples

```
showClass("psychonetrics_log")
```

psychonetrics_update *Model updating functions*

Description

These functions update a psychonetrics model. Typically they are not required.

Usage

```
addMIs(x, matrices = "all", type = c("normal", "free",
                    "equal"), verbose, analyticFisher = TRUE)

addSEs(x, verbose, approximate_SEs = FALSE)

addfit(x, verbose)

identify(x)
```

Arguments

x	A psychonetrics model.
matrices	Optional vector of matrices to include in MIs.
type	String indicating which modification indices should be updated.
verbose	Logical, should messages be printed?
analyticFisher	Logical indicating if an analytic Fisher information matrix should be used.
approximate_SEs	Logical, should standard errors be approximated? If true, an approximate matrix inverse of the Fisher information is used to obtain the standard errors.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

---

ri_clpm *Random intercept cross-lagged panel models and networks*

Description

The ri_clpm function sets up a random intercept cross-lagged panel model (RI-CLPM) as a special case of the latent variable model (lvm) framework. The ri_clpm_stationary function adds stationarity (equality over time) constraints to such a model, and ri_clpm_search runs a guarded decision-tree search over these constraints. The functions ri_clpn, ri_clpn_stationary and ri_clpn_search are the network-parameterization counterparts (see Details).

Usage

```
ri_clpm(data, vars,
        datatype = c("auto", "wide", "long"),
        idvar, beepvar,
        standardize = c("none", "z", "quantile", "z_per_wave"),
        lambda,
        type = c("cor", "cov", "chol", "prec", "ggm"),
        verbose = FALSE, ...)

ri_clpn(data, vars,
        type = c("ggm", "cor", "cov", "chol", "prec"), ...)

ri_clpm_stationary(x,
        stationary = c("intercepts", "contemporaneous",
                       "innovation", "temporal"))

ri_clpn_stationary(...)

ri_clpm_search(x,
        criterion = c("Chisq", "AIC", "BIC"),
        alpha = 0.05,
        include_panelvar = TRUE,
        verbose = TRUE,
        on_rejection = c("release", "stop"), ...)

ri_clpn_search(...)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied (via ...).
vars	Required argument. For wide-format data (the default), a *matrix* with each row indicating a variable and each column indicating a measurement (wave); each cell must contain the name of the variable in data corresponding to that variable at that wave. The row names of this matrix are used as variable names. For long-format data, a character vector of variable names (see datatype, idvar and beepvar).
datatype	The data format. "auto" (default) detects "wide" when vars is a matrix and "long" when vars is a character vector.
idvar	Only used for long-format data: the name of the variable encoding the subject/cluster ID.
beepvar	Optional, only used for long-format data: the name of the variable encoding the measurement occasion (wave). If omitted, observations are assumed to be ordered within each subject.
standardize	Standardization to apply to the data before estimation. "none" (default) applies none, "z" standardizes each variable to mean 0 and SD 1 pooling across all waves, "z_per_wave" standardizes each variable separately at each wave,

and "quantile" applies a (pooled) nonparametric quantile transformation per variable.

lambda — A model matrix encoding the factor loading structure. Latent variables are not yet supported for the RI-CLPM; supplying lambda currently raises an error. When omitted, an observed-variable model is created.

type — The parameterization used for the latent (innovation and between-person random intercept) covariance structure: "cor" (correlations with separate standard deviations; the default of ri_clpm), "cov" (covariances), "chol" (Cholesky), "prec" (precision), or "ggm" (Gaussian graphical model / partial correlations; the default of ri_clpn). Because the latent covariance matrix is block-diagonal over time, this parameterization is applied to both the within-person (innovation) blocks and the between-person (random intercept) block. The default "cor" parameterization separates the innovation correlations from the innovation standard deviations, so that ri_clpm_search can test equality of the contemporaneous relations and equality of the innovation variances as two distinct, nested steps.

verbose — Logical, should progress be printed to the console?

x — A psychonetrics RI-CLPM model (as returned by ri_clpm).

stationary — The part of the model to constrain to be stationary (equal over time): "intercepts" (also frees the random-intercept means), "temporal" (the cross-lagged/autoregressive effects), "contemporaneous" (the innovation covariances/network, waves 2 and later), or "innovation" (the innovation variances, waves 2 and later).

criterion — The criterion used by ri_clpm_search to decide whether to retain each constraint: "Chisq" (the default) retains it when the likelihood-ratio test is not significant at level alpha, while "AIC" and "BIC" retain it when that criterion does not increase. The default was "BIC" up to and including psychonetrics 0.17.4; pass criterion = "BIC" explicitly to reproduce analyses run with those versions.

alpha — Significance level used when criterion = "Chisq".

include_panelvar — Logical. If TRUE (default) and the covariance-stationarity constraints are retained, ri_clpm_search additionally tests the panel(G)VAR model, which treats the first wave as the stationary distribution of the process (wave 1 endogenous). See the "wave1_endogenous" step in the Details section.

on_rejection — What ri_clpm_search does when a constraint is rejected. "release" (default) runs the guarded decision tree: only a rejected *temporal* constraint stops the search, and a rejection of any other constraint releases that constraint (it is not imposed) but the search continues with the remaining steps. "stop" reproduces the pre-0.17.2 behavior of stopping at the first rejection, and is provided for reproducibility of analyses run with earlier versions.

... — Arguments sent to lvm (for ri_clpm), to ri_clpm (for ri_clpn), to ri_clpm_stationary (for ri_clpn_stationary), or to runmodel (for ri_clpm_search) / ri_clpm_search (for ri_clpn_search).

Details

The RI-CLPM decomposes each observed score into a stable between-person random intercept and a within-person deviation (innovation). The within-person deviations follow a lag-1 (cross-lagged) structure over time. In this implementation the innovations and the random intercepts are modeled as latent variables under the `lvm` framework, with the latent covariance matrix being block-diagonal over time, so that the within-person (contemporaneous) and between-person covariance structures can each be modeled as a covariance matrix, a Cholesky decomposition, a precision matrix, or a Gaussian graphical model (GGM).

The RI-CLPN. When the innovation and between-person structures are parameterized as Gaussian graphical models (`type = "ggm"`), the RI-CLPM becomes the *random-intercept cross-lagged panel network model* (RI-CLPN): the contemporaneous and the between-person relations are then partial-correlation networks rather than covariance matrices, while the temporal (cross-lagged) structure is the directed lag-1 network in either case. The function `ri_clpn` is a thin wrapper around `ri_clpm` that uses `type = "ggm"` as its default (the `type` argument can still be overridden), mirroring how `panelgvar` wraps `panelvar`. Because `ri_clpm_stationary` and `ri_clpm_search` operate on an already specified model object, no parameterization is involved there, and `ri_clpn_stationary` and `ri_clpn_search` are pure aliases of those functions, provided so that a network-parameterized analysis can be written entirely in RI-CLPN notation.

The stationarity search. `ri_clpm_search` formalizes the typical analysis workflow as a *guarded decision tree* over the four stationarity constraints, closed off by the panel(G)VAR model. The steps are visited in the order `"temporal"`, `"contemporaneous"`, `"intercepts"`, `"innovation"`, `"wave1_endogenous"`, and each step is tested against the current working model (the base model plus all constraints retained so far). A constraint is retained if it does not worsen fit according to `criterion`: with `criterion = "Chisq"` when the likelihood-ratio test has $p \geq$ `alpha`, and with `"AIC"` or `"BIC"` when that criterion does not increase. A candidate model that did not converge, or that produced a negative or non-finite chi-square or a non-finite criterion, is never retained; such a fit counts as a rejection of its constraint and the reason is reported in the verdict table.

The tree treats the steps differently according to the targets of inference:

- `"temporal"` is the only hard **gate**. The temporal (cross-lagged) effects are the primary target of inference in a cross-lagged panel analysis. If they are not stationary there is no single lag-1 temporal network to report, and the target of inference changes to a *set* of per-transition temporal networks (one for each wave-to-wave step). The search therefore stops entirely, the unconstrained base model is selected, and the remaining steps are recorded as `"not tested"`, since each of them would only be interpretable conditional on a temporal structure that does not hold. When this happens, check whether the waves are equally spaced in time: unequal measurement intervals produce wave-varying lag-1 effects even when the underlying process is perfectly stationary, and so mimic temporal non-stationarity.
- `"contemporaneous"`, `"intercepts"` and `"innovation"` are nuisance- or secondary-structure questions, so a rejection is **released** rather than fatal: the constraint is not imposed (that set of parameters stays wave-specific) but the search continues to the next step. Each of these constraints is therefore tested exactly once, conditional only on the constraints that survived their own test. This reaches endpoints that a strict chain cannot reach, for example wave-varying contemporaneous relations combined with stationary intercepts and stationary innovation variances.
- `"wave1_endogenous"` tests the panel(G)VAR model, which additionally treats the first wave as a draw from the stationary distribution of the within-person process. It is gated on *co-*

variance stationarity and therefore requires the temporal, contemporaneous *and* innovation constraints to all have been retained: a within-person process whose innovation covariance structure varies over waves has no stationary distribution, so there is nothing for an endogenous first wave to be drawn from and the step is undefined. In that case the step is reported as `"not applicable"` and the search finishes. When the step does apply, the working model is compared to `panelvar(vars = ..., sampleStats = ..., within_latent = type, between_latent = type, baseline = "none")`; if the intercept constraint was released, that panel model is fit with `stationary_means = FALSE`, so that both models carry a saturated (wave-specific) mean structure and the comparison isolates the wave-1 block. The panel model is the *constrained* special case here: it has $m(m+1)/2$ fewer free parameters, and hence $m(m+1)/2$ *more* degrees of freedom, than the working RI-CLPM, namely the freed exogenous wave-1 within-person covariance block (m = number of variables). The likelihood-ratio test is thus the chi-square of the panel model minus the chi-square of the working model on $m(m+1)/2$ degrees of freedom. This degrees-of-freedom gap is verified at run time and a warning is issued when it differs, which indicates that the constraint bookkeeping of the working model was disturbed upstream (for example by `prune` or `fixpar`).

Setting `on_rejection = "stop"` reproduces the pre-0.17.2 behavior exactly: the same tree is walked but the search breaks at the first rejection, so no constraint is ever released and the panel(G)VAR step is only reached when all four constraints are retained.

With the default `type = "cor"` parameterization the contemporaneous (correlation) step and the innovation-variance (standard deviation) step are cleanly separated and nested; the innovation variances are never equated with the exogenous wave-1 variances nor with the between-person random-intercept variances.

Extracting and plotting the networks. Because the model is set up in the `lvm` framework, its matrices span all waves at once and the temporal, contemporaneous and between-person networks are blocks of those matrices. Those blocks can be addressed directly, using the same matrix names as `panelvar` / `panelgvar` together with a `time` argument:

```
getmatrix(x, "beta", time = 2)              # temporal network, wave 2 -> 3
getmatrix(x, "PDC",  time = 2)            # ... as partial directed correlations
getmatrix(x, "omega_zeta_within", time = 3)  # contemporaneous network at wave 3
getmatrix(x, "omega_zeta_between")           # between-person network
getmatrix(x, "nu_between")                   # random-intercept means
CIplot(x, "omega_zeta_within", time = 3)    # confidence intervals of that block
```

Blocks carry the variable names of the design matrix, so they can be passed straight to `qgraph`. Omitting `time` returns one block per wave, with waves that were equated by `ri_clpm_stationary` collapsed into a single entry. See `getmatrix` and `CIplot` for the details.

Structural missing waves (`NA` entries in the `vars` design matrix, i.e. a variable not measured at a particular wave) are not yet supported. Incomplete cases (missing data for some subjects) are handled automatically through full-information maximum likelihood.

Value

`ri_clpm`, `ri_clpn` and `ri_clpm_stationary` return a `psychonetrics` model object. `ri_clpm_search` returns a list of class `"ri_clpm_search"` with elements `selected` (the chosen `psychonetrics` model), `selected_name` (a compact description of the selected model, for example `"temporal+contemporaneous+innovat`

(means free)" or "panelVAR (means free)"), models (all fitted models), comparison (a compare table; note that its rows are sorted by degrees of freedom and that consecutive rows need not be nested when constraints were released), path (a data frame describing each fitted model and decision), verdicts, criterion, alpha and on_rejection.

The verdicts element is a data frame with one row per step and the columns step ("temporal", "contemporaneous", "intercepts", "innovation", "wave1_endogenous"), tested (logical), decision ("retained", "released", "rejected (gate)", "not tested" or "not applicable"), Chisq_diff, DF_diff, p_value, dAIC, dBIC (each relative to the working model that the step was tested against) and note (a short explanation, e.g. why a step was not applicable or why a fit was discarded).

Author(s)

Sacha Epskamp

See Also

lvm, dlvm1, panelvar, panelgvar, getmatrix, CIplot

Examples

```
# Simulate a small panel data set (3 variables, 4 waves):
set.seed(1)
nPerson <- 300; nVar <- 3; nWave <- 4

# Between-person random intercepts:
RI <- matrix(rnorm(nPerson * nVar), nPerson, nVar)

# Lag-1 temporal structure:
beta <- diag(0.3, nVar); beta[2, 1] <- 0.2; beta[3, 2] <- 0.2

# Within-person process:
within <- array(0, c(nPerson, nVar, nWave))
within[, , 1] <- matrix(rnorm(nPerson * nVar), nPerson, nVar)
for (w in 2:nWave){
  within[, , w] <- within[, , w - 1] %*% t(beta) +
                   matrix(rnorm(nPerson * nVar, sd = 0.8), nPerson, nVar)
}

# Observed scores = within-person deviation + random intercept:
data <- do.call(cbind, lapply(1:nWave, function(w) within[, , w] + RI))
colnames(data) <- as.vector(outer(paste0("V", 1:nVar), 1:nWave, paste, sep = "_"))
data <- as.data.frame(data)

# Design matrix (rows = variables, columns = waves):
design <- matrix(colnames(data), nVar, nWave)
rownames(design) <- paste0("V", 1:nVar)

# Form the unconstrained RI-CLPN (= RI-CLPM with GGM innovation/between
# structures). Equivalent to ri_clpm(data, design, type = "ggm"):
mod <- ri_clpn(data, design)
```

```
# Run the unconstrained model:
mod <- runmodel(mod)

# Add stationarity constraints step by step:
mod_temp <- runmodel(ri_clpm_stationary(mod,      "temporal"))
mod_con  <- runmodel(ri_clpm_stationary(mod_temp, "contemporaneous"))
compare(base = mod, temporal = mod_temp, contemporaneous = mod_con)

# Or run the whole guarded decision-tree search automatically
# (chi-square difference tests at alpha = 0.05 by default):
res <- ri_clpn_search(mod)
res
res$verdicts
res$selected

# Information criteria can be used instead:
ri_clpn_search(mod, criterion = "BIC")

# Extract the networks as nVar x nVar blocks (rather than the full
# latent-sized matrices of the underlying lvm model):
getmatrix(mod, "omega_zeta_between")           # between-person network
getmatrix(mod, "beta", time = 1)               # temporal network, T1 -> T2
getmatrix(mod, "PDC", time = 1)                # ... as partial directed correlations
getmatrix(mod, "omega_zeta_within", time = 2)  # contemporaneous network at wave 2

# The pre-0.17.2 behavior: BIC, stopping at the first rejection:
ri_clpn_search(mod, criterion = "BIC", on_rejection = "stop")
```

---

runmodel *Run a psychonetrics model*

Description

This is the main function used to run a psychonetrics model.

Usage

```
runmodel(x, level = c("gradient", "fitfunction"), addfit =
                   TRUE, addMIs = TRUE, addSEs = TRUE, addInformation =
                   TRUE, log = TRUE, verbose, optim.control,
                   analyticFisher = TRUE, warn_improper = FALSE,
                   warn_gradient = TRUE, warn_bounds = TRUE,
                   return_improper = TRUE, bounded = TRUE,
                   approximate_SEs = FALSE,
                   criterion = "ebic.5",
                   beta_min = c("numerical", "theoretical"),
                   saturated = c("default", "analytic", "model"))
```

Arguments

x	A psychonetrics model.
level	Level at which the model should be estimated. Defaults to "gradient" to indicate the analytic gradient should be used.
addfit	Logical, should fit measures be added?
addMIs	Logical, should modification indices be added?
addSEs	Logical, should standard errors be added?
addInformation	Logical, should the Fisher information be added?
log	Logical, should the log be updated?
verbose	Logical, should messages be printed?
optim.control	A list with options for optimr
analyticFisher	Logical, should the analytic Fisher information be used? If FALSE, numeric information is used instead.
return_improper	Should a result in which improper computation was used be return? Improper computation can mean that a pseudoinverse of small spectral shift was used in computing the inverse of a matrix.
warn_improper	Logical. Should a warning be given when at some point in the estimation a pseudoinverse was used?
warn_gradient	Logical. Should a warning be given when the average absolute gradient is > 1?
bounded	Logical. Should bounded estimation be used (e.g., variances should be positive)?
approximate_SEs	Logical, should standard errors be approximated? If true, an approximate matrix inverse of the Fisher information is used to obtain the standard errors.
warn_bounds	Should a warning be given when a parameter is estimated near its bounds?
criterion	Character string specifying the information criterion for automatic lambda selection in PML/PFIML models. One of "bic", "ebic.25", "ebic.5" (default), "ebic.75", or "ebic1". Only used when the model contains auto-select penalty parameters (penalty_lambda = NA). See find_penalized_lambda for details.
beta_min	Threshold for zeroing out small penalized parameters during automatic lambda selection. Can be "numerical" (default; uses 1e-05), "theoretical" (uses sqrt(log(p)/n)), or a numeric value. See find_penalized_lambda for details.
saturated	Method for computing the saturated model log-likelihood. "default" runs the saturated varcov model numerically, with automatic fallback to the analytical formula if the optimizer fails (saturated LL < model LL). "analytic" skips the saturated model optimization and uses the analytical pattern-specific formula directly (fast, useful for high-dimensional panel data where optimization is slow or fails). "model" always runs the saturated varcov model numerically with no fallback. The method that was ultimately used is recorded in the result: x@baseline_saturated$satMethodUsed is "numeric", "analytic", or

"analytic_fallback", and the fit measure satLL_analytic is a 0/1 indicator of whether the analytic formula was used (forced or as fallback), which is retained by aggregate_bootstraps (its bootstrap average is the proportion of samples using the analytic saturated log-likelihood).

Details

For penalized models (PML or PFIML) with auto-select penalty parameters (penalty_lambda = NA, the default), runmodel automatically calls find_penalized_lambda to select the optimal penalty strength via EBIC grid search before fitting. The criterion and beta_min arguments control this search. After the lambda search, use refit for post-selection inference with standard errors.

For the robust maximum likelihood estimators ("MLM", "MLMV", "MLMVS", "MLR"; see setestimator), the point estimates are obtained by plain maximum likelihood, while runmodel additionally computes robust (sandwich) standard errors, a scaled test statistic (Satorra-Bentler family or Yuan-Bentler) and robust RMSEA/CFI/TLI. These are stored in the model fit measures (chisq.scaled, chisq.scaling.factor, df.scaled, chisq.shift.parameters, rmsea.robust, cfi.robust, tli.robust). The Satorra-Bentler family ("MLM"/"MLMV"/"MLMVS") requires complete (continuous) raw data. "MLR" additionally supports missing data through full-information maximum likelihood (FIML): when the raw data contain missing values, point estimation uses FIML and the robust standard errors (Huber-White), the Yuan-Bentler-Mplus scaled test statistic and the FIML-corrected robust RMSEA/CFI/TLI are computed under missingness.

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

find_penalized_lambda for manual control over the lambda search, penalize for setting penalty parameters, refit for post-selection inference after penalized estimation.

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
```

```
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars, omega = "full")

# Run model:
mod <- mod %>% runmodel
```

---

setestimator *Convenience functions*

Description

These functions can be used to change some estimator options.

Usage

```
setestimator(x, estimator)

setoptimizer(x, optimizer = c("default", "nlminb", "ucminf",
                 "nloptr_TNEWTON", "LBFGS++"), optim.args)


usecpp(x, use = TRUE)
```

Arguments

x	A psychonetrics model.
estimator	A string indicating the estimator to be used. Maximum likelihood ("ML"), full-information maximum likelihood ("FIML"), and the least-squares estimators ("WLS", "DWLS", "ULS"; "WLSMV" is a synonym for "DWLS" with a mean-and-variance adjusted scaled test) are supported. In addition, the following *robust maximum likelihood* estimators are available for complete continuous data (raw data required). They use plain ML for the point estimates but report robust (sandwich) standard errors, a scaled test statistic and robust RMSEA/CFI/TLI:
	"MLM" Browne (1984) "robust.sem" sandwich standard errors and the Satorra-Bentler (1994) mean-adjusted scaled chi-square.
	"MLMV" Robust.sem standard errors and the scaled-and-shifted chi-square (Satorra 2000; Asparouhov & Muthen 2010).
	"MLMVS" Robust.sem standard errors and the mean-and-variance adjusted (Satterthwaite) scaled chi-square with fractional degrees of freedom.
	"MLR" Huber-White (sandwich) standard errors and the Yuan-Bentler (2000) scaled chi-square (the Mplus variant). Requires the raw data to be stored; the model constructors set storedata = TRUE automatically when estimator = "MLR". "MLR" also supports *missing* data: when the raw data contain missing values it estimates the point estimates by full-information maximum likelihood (FIML) and computes the Huber-White standard errors,

the Yuan-Bentler-Mplus scaled chi-square and the FIML-corrected (Savalei 2010) robust RMSEA/CFI/TLI under missingness.

These robust estimators map internally to `estimator = "ML"` (or, for `"MLR"` with missing data, `estimator = "FIML"`) plus robust standard-error and test flags, so the point estimates are identical to plain ML (or FIML). The scaled statistics are stored in the fit measures (`chisq.scaled`, `chisq.scaling.factor`, `df.scaled`, `chisq.shift.parameters`, `rmsea.robust`, `cfi.robust`, `tli.robust`). The Satorra-Bentler family (`"MLM"`/`"MLMV"`/`"MLMVS"`) requires complete data; for missing data use `"MLR"`.

`optimizer` The optimizer to be used. The following optimizers are available:

R-based optimizers:

`"nlminb"` Port trust-region Newton-like optimizer from base R. Uses analytic gradients and a trust-region method that adaptively controls step sizes, making it very stable for SEM problems. This is the default and recommended optimizer for most models.

`"ucminf"` Unconstrained minimization using a quasi-Newton method (via the `optimr` package). Does not support box constraints. Fast for unconstrained problems.

C++ based optimizer:

`"LBFGS++"` L-BFGS-B optimizer from the LBFGSpp library (Yixuan Qiu). This is a pure C++ implementation that computes the objective function and gradient in a single combined call, avoiding redundant computation via internal caching. Supports box constraints. Recommended when speed is important.

NLopt-based optimizer (via nloptr):

`"nloptr_TNEWTON"` Preconditioned truncated Newton with restarts from the NLopt library (via the `nloptr` package). Uses the `NLOPT_LD_TNEWTON_PRECOND_RESTART` algorithm, which builds a local quadratic model of the objective and solves the Newton system approximately using a preconditioned conjugate-gradient method. Supports box constraints and uses analytic gradients. Can handle large parameter spaces efficiently. See Dembo & Steihaug (1982) for the underlying method.

Defaults to `"nlminb"`.

`use` Logical indicating if C++ should be used (currently only used in FIML)

`optim.args` List of arguments to sent to the optimizer.

Details

The default optimizer is nlminb with the following arguments:

- eval.max=20000L
- iter.max=10000L
- trace=0L
- abs.tol=sqrt(.Machine$double.eps)
- rel.tol=sqrt(.Machine$double.eps)

- step.min=1.0
- step.max=1.0
- x.tol=1.5e-8
- xf.tol=2.2e-14

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

References

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.

Satorra, A., & Bentler, P. M. (1994). Corrections to test statistics and standard errors in covariance structure analysis. In A. von Eye & C. C. Clogg (Eds.), *Latent variables analysis: Applications for developmental research* (pp. 399-419). Sage.

Yuan, K.-H., & Bentler, P. M. (2000). Three likelihood-based methods for mean and covariance structure analysis with nonnormal missing data. *Sociological Methodology, 30*(1), 165-200.

Savalei, V. (2010). Expected versus observed information in SEM with incomplete normal and nonnormal data. *Psychological Methods, 15*(4), 352-367.

Brosseau-Liard, P. E., & Savalei, V. (2014). Adjusting incremental fit indices for nonnormality. *Multivariate Behavioral Research, 49*(5), 460-470.

Asparouhov, T., & Muthen, B. (2010). *Simple second order chi-square correction.* Mplus Technical Appendix.

Examples

```
# Robust maximum likelihood (MLR) on the Holzinger-Swineford data:
library("lavaan")
data("HolzingerSwineford1939")
vars <- paste0("x", 1:9)

# Three-factor CFA loadings matrix:
Lambda <- matrix(0, 9, 3)
Lambda[1:3, 1] <- Lambda[4:6, 2] <- Lambda[7:9, 3] <- 1

# Robust ML (Huber-White SEs + Yuan-Bentler scaled chi-square). The point
# estimates are identical to plain ML; only the SEs, the scaled test and the
# robust fit indices differ:
mod <- lvm(HolzingerSwineford1939, lambda = Lambda, vars = vars,
           identification = "variance", estimator = "MLR")
mod <- runmodel(mod)

# Robust fit measures (chisq.scaled, rmsea.robust, cfi.robust, ...):
```

```
fit(mod)

# Equivalent via setestimator() (the model must store the raw data):
mod2 <- lvm(HolzingerSwineford1939, lambda = Lambda, vars = vars,
            identification = "variance", estimator = "ML", storedata = TRUE)
mod2 <- runmodel(setestimator(mod2, "MLM"))

# MLR also handles missing data via FIML. With missing values in the raw data,
# estimator = "MLR" estimates the point estimates by FIML and reports
# Huber-White standard errors, the Yuan-Bentler-Mplus scaled chi-square and the
# FIML-corrected robust RMSEA/CFI/TLI:
HSmis <- HolzingerSwineford1939
set.seed(1)
for (v in vars) HSmis[[v]][runif(nrow(HSmis)) < 0.1] <- NA
modM <- lvm(HSmis, lambda = Lambda, vars = vars,
            identification = "variance", estimator = "MLR")
modM <- runmodel(modM)   # estimator is "FIML" internally, with MLR robust output
fit(modM)
```

`setverbose` *Should messages of computation progress be printed?*

Description

This function controls if messages should be printed for a psychonetrics model.

Usage

```
setverbose(x, verbose = TRUE)
```

Arguments

`x`	A `psychonetrics` model.
`verbose`	Logical indicating if verbose should be enabled

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

simplestructure *Generate factor loadings matrix with simple structure*

Description

This function generates the input for `lambda` arguments in latent variable models using a simple structure. The input is a vector with an element for each variable indicating the factor the variable loads on.

Usage

```
simplestructure(x)
```

Arguments

`x`	A vector indicating which factor each indicator loads on.

Value

A binary `lambda` (factor loadings) matrix with one row per indicator and one column per factor, with a 1 in each cell where the indicator loads on the factor and 0 otherwise.

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

StarWars *Star Wars dataset*

Description

This questionnaire was constructed by Carolin Katzera, Charlotte Tanis, Esther Niehoff, Myrthe Veenman, and Jason Nak as part of an assignment for a course on confirmatory factor analysis (http://sachaepskamp.com/SEM2018). They also collected the data among fellow psychology students as well as through social media.

Usage

```
data("StarWars")
```

Format

A data frame with 271 observations on the following 13 variables.

`Q1` I am a huge Star Wars fan! (star what?)

`Q2` I would trust this person with my democracy <picture of Jar Jar Binks>

`Q3` I enjoyed the story of Anakin's early life

`Q4` The special effects in this scene are awful <video of the Battle of Geonosis>

`Q5` I would trust this person with my life <picture of Han Solo>

`Q6` I found Darth Vader's big reveal in "Empire" one of the greatest moments in movie history

`Q7` The special effects in this scene are amazing <video of the Death Star explosion>

`Q8` If possible, I would definitely buy this droid <picture of BB-8>

`Q9` The story in the Star Wars sequels is an improvement to the previous movies

`Q10` The special effects in this scene are marvellous <video of the Starkiller Base firing>

`Q11` What is your gender?

`Q12` How old are you?

`Q13` Have you seen any of the Star Wars movies?

Details

The questionnaire is online at https://github.com/SachaEpskamp/SEM-code-examples/blob/master/CFA_fit_examples/StarW
The authors of the questionnaire defined a measurement model before collecting data: Q2 - Q4 are expected to load on a "prequel" factor, Q5 - Q7 are expected to load in a "originals" factor, and Q8 - Q10 are expected to load on a "sequel" factor. Finally, Q1 is expected to load on all three.

Source

https://github.com/SachaEpskamp/SEM-code-examples/blob/master/CFA_fit_examples

Examples

```
data(StarWars)
```

---

stepup *Stepup model search along modification indices*

Description

This function automatically performs step-up search by adding the parameter with the largest modification index until some criterion is reached or no modification indices are significant at alpha.

Usage

```
stepup(x, alpha = 0.01, criterion = "bic", matrices, mi =
                    c("mi", "mi_free", "mi_equal"), greedyadjust =
                    c("bonferroni", "none", "holm", "hochberg", "hommel",
                    "fdr", "BH", "BY"), stopif, greedy = FALSE, verbose,
                    checkinformation = TRUE, singularinformation =
                    c("tryfix", "skip", "continue", "stop"), startEPC =
                    TRUE, ...)
```

Arguments

x	A psychonetrics model.
alpha	Significance level to use.
criterion	String indicating the criterion to minimize. Any criterion from fit can be used.
matrices	Vector of strings indicating which matrices should be searched. Will default to network structures and factor loadings.
mi	String indicating which kind of modification index should be used ("mi" is the typical MI, "mi_free" is the modification index free from equality constraints across groups, and "mi_equal" is the modification index if the parameter is added constrained equal across all groups).
greedyadjust	String indicating which p-value adjustment should be used in greedy start. Any method from p.adjust can be used.
stopif	An R expression, using objects from fit, which will break stepup search if it evaluates to TRUE. For example, stopif = rmsea < 0.05 will lead to search to stop if rmsea is below 0.05.
greedy	Logical, should a greedy start be used? If TRUE, the first step adds any parameter that is significant (after adjustment)
verbose	Logical, should messages be printed?
checkinformation	Logical, should the Fisher information be checked for potentially non-identified models?
singularinformation	String indicating how to proceed if the information matrix is singular. "tryfix" will adjust starting values to try to fix the proble, "skip" will lead to the algorithm to skip the current parameter, "continue" will ignore the situation, and "stop" will break the algorithm and return a list with the last two models.
startEPC	Logical, should the starting value be set at the expected parameter change?
...	Arguments sent to runmodel

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

See Also

prune

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Let's fit a full GGM:
mod <- ggm(ConsData, vars = vars, omega = "full")

# Run model:
mod <- mod %>%runmodel %>%prune(alpha = 0.05)

# Remove an edge (example):
mod <- mod %>%fixpar("omega",1,2) %>%runmodel

# Stepup search
mod <- mod %>%stepup(alpha = 0.05)
```

---

tolavaan *Convert a psychonetrics lvm to a lavaan model*

Description

Converts a psychonetrics latent variable model (lvm) with `latent = "cov"` and `residual = "cov"` into a fitted `lavaan` object, or into a lavaan parameter table. Only the public lavaan API (`lavaan::lavaan`) is used; no lavaan source code is reused.

Usage

```
tolavaan(x, fit = TRUE, verbose = FALSE, ...)
```

Arguments

x	A psychonetrics object created with lvm (or a wrapper such as lnm) using latent = "cov" and residual = "cov".
fit	Logical. If TRUE (default), the model is fitted with lavaan::lavaan and the resulting lavaan object is returned. If FALSE, the lavaan parameter table (a data.frame) is returned without fitting.
verbose	Logical. If TRUE, an experimental-feature note is shown.
...	Further arguments passed to lavaan::lavaan. For example, pass do.fit = FALSE to obtain an unfitted lavaan object that uses the psychonetrics estimates as starting values.

Details

Only models with latent = "cov" and residual = "cov" can be converted, because network (ggm/prec) and Cholesky (chol) parameterizations have no direct lavaan equivalent. The function emits one lavaan parameter-table row for every element of the psychonetrics matrices lambda, sigma_zeta, sigma_epsilon, beta, nu and nu_eta (including elements that are fixed to zero, so that lavaan adds no default parameters). Free parameters are given sequential free indices and fixed parameters their fixed value via ustart. Equality constraints (shared psychonetrics par indices) are encoded with per-row plabels and explicit == rows, which is the only encoding lavaan honors for cross-row equality.

Warm start. When the psychonetrics model has been estimated (x@computed is TRUE), the estimates are also written as starting values (ustart) for the free parameters. lavaan then typically polishes this warm start by a few optimizer steps, so the refitted log-likelihood matches to within optimizer tolerance (about 10^{-5}) rather than exactly.

Data. If the psychonetrics model stored its raw data (fitted with storedata = TRUE), that raw data is passed to lavaan (with the grouping variable for multi-group models, and missing = "ml" for FIML). Otherwise the sample covariance matrix (and the sample means, when a mean structure is present) is passed with sample.cov.rescale = FALSE and likelihood = "normal", since psychonetrics covariances already use the n (maximum-likelihood) normalization. FIML conversion therefore requires the raw data: re-fit the lvm with storedata = TRUE first.

Unsupported. The following raise an informative error: non-lvm models, lvm models with latent or residual not equal to "cov", correlation-input or ordinal models, estimators other than "ML" or "FIML", and FIML models fitted without storedata = TRUE.

Value

A fitted lavaan object when fit = TRUE, or a lavaan parameter table (data.frame) when fit = FALSE.

Author(s)

Sacha Epskamp

References

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

See Also

fromlavaan, lvm

Examples

```
if (requireNamespace("lavaan", quietly = TRUE)){
  library("lavaan")
  library("dplyr")

  # A psychonetrics CFA:
  data(HolzingerSwineford1939, package = "lavaan")
  HS <- lavaan::HolzingerSwineford1939
  Lambda <- matrix(0, 9, 3)
  Lambda[1:3, 1] <- Lambda[4:6, 2] <- Lambda[7:9, 3] <- 1

  mod <- lvm(HS, lambda = Lambda, latent = "cov", residual = "cov",
             vars = paste0("x", 1:9),
             latents = c("visual", "textual", "speed"),
             identification = "loadings") %>%
    runmodel

  # Convert to a fitted lavaan object:
  lavfit <- tolavaan(mod)
  c(psychonetrics = mod@fitmeasures$logl,
    lavaan = as.numeric(logLik(lavfit)))

  # Or obtain just the parameter table:
  PT <- tolavaan(mod, fit = FALSE)
  head(PT)
}
```

---

transmod *Transform between model types*

Description

This function allows to transform a model variance–covariance structure from one type to another. Its main uses are to (1) use a Cholesky decomposition to estimate a saturated covariance matrix or GGM, and (2) to transform between conditional (ggm) and marginal associations (cov).

Usage

```
transmod(x, ..., verbose, keep_computed = FALSE, log = TRUE,
         identify = TRUE)
```

Arguments

x	A psychonetrics model
...	Named arguments with the new types to use (e.g., between = "ggm" or y = "cov")
verbose	Logical, should messages be printed?
keep_computed	Logical: keep the model marked as computed after the transformation? Defaults to FALSE (the model must be re-run, although the transformed estimates are already at the ML solution). Cannot be TRUE with identification = 'variance'.
log	Logical, should a logbook entry be made?
identify	Logical, should the model be identified after transforming?

Details

Transformations are only possible if the model is diagonal (e.g., no partial correlations) or saturated (e.g., all covariances included).

Value

An object of the class psychonetrics (psychonetrics-class) with the requested variance–covariance structure(s).

Author(s)

Sacha Epskamp

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
  select(A1:A5, gender) %>%
  na.omit # Let's remove missingness (otherwise use Estimator = "FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Model with Cholesky decompositon:
mod <- varcov(ConsData, vars = vars, type = "chol")

# Run model:
mod <- mod %>% runmodel

# Transform to GGM:
```

```
mod_trans <- transmod(mod, type = "ggm") %>% runmodel
# Note: runmodel often not needed

# Obtain thresholded GGM:
getmatrix(mod_trans, "omega", threshold = TRUE)
```

`tsdlvm1` *Lag-1 dynamic latent variable model family of psychonetrics models for time-series data*

Description

This is the family of models that models a dynamic factor model on time-series. There are two covariance structures that can be modeled in different ways: `contemporaneous` for the contemporaneous model and `residual` for the residual model. These can be set to `"cov"` for covariances, `"prec"` for a precision matrix, `"ggm"` for a Gaussian graphical model and `"chol"` for a Cholesky decomposition. The `ts_lvgvar` wrapper function sets `contemporaneous = "ggm"` for the graphical VAR model. `tsdlvm()` is the primary short name of `tsdlvm1()` (identical function; the historical "1" referred to the lag order).

Usage

```
tsdlvm1(data, lambda, contemporaneous = c("cov", "chol",
                   "prec", "ggm", "cor"), residual = c("cov", "chol", "prec",
                  "ggm", "cor"), beta = "full", omega_zeta = "full", delta_zeta
                  = "diag", kappa_zeta = "full", sigma_zeta = "full",
                  lowertri_zeta = "full", omega_epsilon = "zero",
                  delta_epsilon = "diag", kappa_epsilon = "diag",
                  sigma_epsilon = "diag", lowertri_epsilon = "diag", nu,
                  mu_eta, identify = TRUE, identification =
                  c("loadings", "variance"), latents, beepvar, dayvar,
                  idvar, vars, groups, groupvar, covs, cors,
                  means, nobs, corinput, missing =
                  "auto", equal = "none", baseline_saturated = TRUE,
                  estimator = "ML", optimizer, storedata = FALSE,
                  sampleStats, covtype = c("choose", "ML", "UB"),
                  centerWithin = FALSE, standardize = c("none", "z",
                  "quantile"), verbose = FALSE, bootstrap = FALSE,
                  boot_sub, boot_resample, penalty_lambda = NA,
                  penalty_alpha = 1, penalize_matrices,
                  rho_zeta = "full", SD_zeta = "diag",
                  rho_epsilon = "zero", SD_epsilon = "diag",
                  temporal = c("raw","PDC"), PDC = "full")


tsdlvm(...)

ts_lvgvar(...)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied.
lambda	A model matrix encoding the factor loading structure. Each row indicates an indicator and each column a latent. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
contemporaneous	The type of contemporaneous model used. See description.
residual	The type of residual model used. See description.
beta	A model matrix encoding the temporal relationships (transpose of temporal network) between latent variables. A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Can also be "full" for a full temporal network or "zero" for an empty temporal network.
omega_zeta	Only used when contemporaneous = "ggm". Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
delta_zeta	Only used when contemporaneous = "ggm". Either "diag" or "zero" (not recommended), or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
kappa_zeta	Only used when contemporaneous = "prec". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
sigma_zeta	Only used when contemporaneous = "cov". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
lowertri_zeta	Only used when contemporaneous = "chol". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`omega_epsilon` Only used when `residual = "ggm"`. Either `"full"` to estimate every element freely, `"zero"` to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta_epsilon` Only used when `residual = "ggm"`. Either `"diag"` or `"zero"` (not recommended), or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`kappa_epsilon` Only used when `residual = "prec"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`sigma_epsilon` Only used when `residual = "cov"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri_epsilon`
Only used when `residual = "chol"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`nu` Optional vector encoding the intercepts of the observed variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.

`mu_eta` Optional vector encoding the means of the latent variables. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free intercepts, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.

`identify` Logical, should the model be automatically identified?

`identification` Type of identification used. `"loadings"` to fix the first factor loadings to 1, and `"variance"` to fix the diagonal of the latent variable model matrix (sigma_zeta, lowertri_zeta, delta_zeta or kappa_zeta) to 1.

`latents` An optional character vector with names of the latent variables.

`beepvar` Optional string indicating assessment beep per day. Adding this argument will cause non-consecutive beeps to be treated as missing!

dayvar — Optional string indicating assessment day. Adding this argument makes sure that the first measurement of a day is not regressed on the last measurement of the previous day. IMPORTANT: only add this if the data has multiple observations per day.

idvar — Optional string indicating the subject ID

vars — An optional character vector encoding the variables used in the analysis. Must equal names of the dataset in data.

groups — Deprecated. Use groupvar instead. An optional string indicating the name of the group variable in data.

groupvar — An optional string indicating the name of the group variable in data. Replaces the deprecated groups argument; if both are supplied, groupvar takes precedence with a warning.

covs — A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure covtype argument is set correctly to the type of covariances used.

cors — A sample correlation matrix, or a list/array of such matrices for multiple groups. When supplied, the matrix is treated as a covariance matrix with a warning (appropriate for standardized data). Requires nobs. Note: corinput = TRUE is not supported in tsdlvm1.

means — A vector of sample means, or a list/matrix containing such vectors for multiple groups.

nobs — The number of observations used in covs and means, or a vector of such numbers of observations for multiple groups.

corinput — Logical. Not supported in tsdlvm1() and will produce an error if set to TRUE.

missing — How should missingness be handled in computing the sample covariances and number of observations when data is used. Can be "auto" (default) for automatic detection, "listwise" for listwise deletion, or "pairwise" for pairwise deletion. When "auto", the function checks for missing data and switches ML to FIML, PML to PFIML, or defaults to listwise for LS estimators.

equal — A character vector indicating which matrices should be constrained equal across groups.

baseline_saturated — A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

estimator — The estimator to be used. Currently implemented are "ML" for maximum likelihood estimation, "FIML" for full-information maximum likelihood estimation, "PML" for penalized maximum likelihood estimation, "PFIML" for penalized full-information maximum likelihood estimation, "ULS" for unweighted least squares estimation, "WLS" for weighted least squares estimation, and "DWLS" for diagonally weighted least squares estimation. When missing = "auto" (default), "ML" is automatically switched to "FIML" and "PML" to "PFIML" if missing data is detected.

optimizer — The optimizer to be used. Can be one of "nlminb" (the default R nlminb function), "ucminf" (from the optimr package), "nloptr_TNEWTON" (preconditioned truncated Newton via nloptr), and "LBFGS++" (pure C++ L-BFGS-B). Defaults to "nlminb".

storedata	Logical, should the raw data be stored? Needed for bootstrapping (see bootstrap).
standardize	Which standardization method should be used? "none" (default) for no standardization, "z" for z-scores, and "quantile" for a non-parametric transformation to the quantiles of the marginal standard normal distribution.
sampleStats	An optional sample statistics object. Mostly used internally.
centerWithin	Logical, should data be within-person centered?
covtype	If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to "ML" for maximum likelihood estimates (denominator n) and "UB" to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.
verbose	Logical, should messages be printed?
bootstrap	Should the data be bootstrapped? If TRUE the data are resampled and a bootstrap sample is created. These must be aggregated using aggregate_bootstraps! Can be TRUE or FALSE. Can also be "nonparametric" (which sets boot_sub = 1 and boot_resample = TRUE) or "case" (which sets boot_sub = 0.75 and boot_resample = FALSE).
boot_sub	Proportion of cases to be subsampled (round(boot_sub * N)).
boot_resample	Logical, should the bootstrap be with replacement (TRUE) or without replacement (FALSE)
penalty_lambda	Numeric penalty strength for penalized ML estimation (PML/PFIML). NA (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See find_penalized_lambda and penalize.
penalty_alpha	Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.
penalize_matrices	Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.
rho_zeta	Only used when contemporaneous = "cor". Either "full" or "zero", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the contemporaneous (innovation) correlations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
SD_zeta	Only used when contemporaneous = "cor". Either "full" or "diag", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the diagonal matrix of contemporaneous (innovation) standard deviations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
rho_epsilon	Only used when residual = "cor". Either "full" or "zero", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the residual correlations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

SD_epsilon	Only used when residual = "cor". Either "full" or "diag", or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the diagonal matrix of residual standard deviations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
temporal	Parameterization of the temporal effects: "raw" (default) models beta directly; "PDC" models the partial directed correlations directly, with the PDC matrix encoding from = row and to = column (the transposed orientation relative to beta, matching getmatrix(..., "PDC")). A (pattern-constrained) PDC model is fit-equivalent to the corresponding raw model, but standard errors, confidence intervals, equality constraints and modification indices then apply directly on the PDC scale.
PDC	Only used when temporal = "PDC". Model matrix for the partial directed correlations (from = row, to = column): "full", "zero", "diag", or a pattern matrix with 0s indicating elements fixed to zero, 1s indicating free elements, and higher integers indicating equality constraints.
...	Arguments sent to tsdlvm1

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

Examples

```
# Note: this example is wrapped in a dontrun environment because the data is not
# available locally.
## Not run:
# Obtain the data from:
#
# Epskamp, S., van Borkulo, C. D., van der Veen, D. C., Servaas, M. N., Isvoranu, A. M.,
# Riese, H., & Cramer, A. O. (2018). Personalized network modeling in psychopathology:
# The importance of contemporaneous and temporal connections. Clinical Psychological
# Science, 6(3), 416-427.
#
# Available here: https://osf.io/c8wjz/
tsdata <- read.csv("Supplementary2_data.csv")

# Encode time variable in a way R understands:
tsdata$time <- as.POSIXct(tsdata$time, tz = "Europe/Amsterdam")

# Extract days:
tsdata$Day <- as.Date(tsdata$time, tz = "Europe/Amsterdam")

# Variables to use:
vars <- c("relaxed", "sad", "nervous", "concentration", "tired", "rumination",
```

```
          "bodily.discomfort")

# Create lambda matrix (in this case: one factor):
Lambda <- matrix(1,7,1)

# Estimate dynamical factor model:
model <- tsdlvm1(
  tsdata,
  lambda = Lambda,
  vars = vars,
  dayvar = "Day",
  estimator = "FIML"
)

# Run model:
model <- model %>% runmodel

# Look at fit:
model %>% print
model %>% fit # Pretty bad fit

## End(Not run)
```

---

unionmodel *Unify models across groups*

Description

The unionmodel will add all parameters to all groups that are free in at least one group, and the intersectionmodel will constrain all parameters across groups to zero unless they are free to estimate in all groups.

Usage

```
unionmodel(x, runmodel = FALSE, verbose, log = TRUE, identify =
                    TRUE, matrices, ...)

intersectionmodel(x, runmodel = FALSE, verbose, log = TRUE, identify =
                    TRUE, matrices, ...)
```

Arguments

x	A psychonetrics model.
runmodel	Logical, should the model be updated?
verbose	Logical, should messages be printed?
log	Logical, should the log be updated?
identify	Logical, should the model be identified?
matrices	Which matrices should be used to form the union/intersection model?
...	Arguments sent to runmodel

Value

An object of the class psychonetrics (psychonetrics-class)

Author(s)

Sacha Epskamp

`update_baseline_saturated` *Set a custom baseline or saturated reference model*

Description

Convenience setters that overwrite the `$baseline` or `$saturated` slot of a `psychonetrics` model's `@baseline_saturated` list. Together with the `baseline_saturated = FALSE` argument exposed by every model constructor, they make it straightforward to attach hand-built reference models before calling `runmodel`.

Usage

```
update_baseline(x, baseline)
update_saturated(x, saturated)
```

Arguments

`x`	A `psychonetrics` model.
`baseline`	A `psychonetrics` model to use as the baseline reference.
`saturated`	A `psychonetrics` model to use as the saturated reference.

Details

Pass `baseline_saturated = FALSE` to the model constructor to skip auto-construction of the default reference models, attach your own with these setters, and run the target. `runmodel` will run any attached baseline / saturated that is not yet `@computed`.

Value

An object of the class psychonetrics (psychonetrics-class) with the relevant slot overwritten.

Author(s)

Sacha Epskamp

Examples

```
## Not run:
mod <- modelfun(..., baseline_saturated = FALSE)
mod <- update_baseline(mod, mod_baseline)
mod <- update_saturated(mod, mod_saturated)
mod <- runmodel(mod)

## End(Not run)
```

---

var1 *Lag-1 vector autoregression family of psychonetrics models*

Description

This is the family of models that models time-series data using a lag-1 vector autoregressive model (VAR; Epskamp,Waldorp, Mottus, Borsboom, 2018). The model is fitted to the Toeplitz matrix, but unlike typical SEM software the block of covariances of the lagged variables is not used in estimating the temporal and contemporaneous relationships (the block is modeled completely separately using a cholesky decomposition, and does not enter the model otherwise). The `contemporaneous` argument can be used to define what contemporaneous model is used: `contemporaneous = "cov"` (default) models a variance-covariance matrix, `contemporaneous = "chol"` models a Cholesky decomposition, `contemporaneous = "prec"` models a precision matrix, `contemporaneous = "ggm"` (alias: `gvar()`) models a Gaussian graphical model, also then known as a graphical VAR model, and `contemporaneous = "cor"` models a correlation matrix with separate standard deviations.

Usage

```
var1(data, contemporaneous = c("cov", "chol", "prec",
                  "ggm", "cor"), beta = "full", omega_zeta = "full", delta_zeta
                   = "full", kappa_zeta = "full", sigma_zeta = "full",
                   lowertri_zeta = "full", mu, beepvar, dayvar, idvar,
                   vars, groups, groupvar, covs, cors, means,
                   nobs, corinput, missing = "auto",
                   centerWithin = FALSE,
                   equal = "none", baseline_saturated = TRUE, estimator =
                   "ML", optimizer, storedata = FALSE, covtype =
                   c("choose", "ML", "UB"), standardize = c("none", "z",
                   "quantile"), sampleStats, verbose = FALSE, bootstrap =
                   FALSE, boot_sub, boot_resample,
                   penalty_lambda = NA, penalty_alpha = 1,
                   penalize_matrices, rho_zeta = "full", SD_zeta = "full",
                   temporal = c("raw","PDC"), PDC = "full")

gvar(...)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied.
contemporaneous	The type of contemporaneous model used. See description.
beta	A model matrix encoding the temporal relationships (transpose of temporal network). A 0 encodes a fixed to zero element, a 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix. Can also be "full" for a full temporal network or "zero" for an empty temporal network.
omega_zeta	Only used when contemporaneous = "ggm". Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
delta_zeta	Only used when contemporaneous = "ggm". The scaling (standard deviation) matrix, which is diagonal. Either "full" (the default) or "diag" to freely estimate all diagonal scalings, "zero" (not recommended) to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
kappa_zeta	Only used when contemporaneous = "prec". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
sigma_zeta	Only used when contemporaneous = "cov". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
lowertri_zeta	Only used when contemporaneous = "chol". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
mu	Optional vector encoding the mean structure. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free means, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.

beepvar — Optional string indicating assessment beep per day. Adding this argument will cause non-consecutive beeps to be treated as missing!

dayvar — Optional string indicating assessment day. Adding this argument makes sure that the first measurement of a day is not regressed on the last measurement of the previous day. IMPORTANT: only add this if the data has multiple observations per day.

idvar — Optional string indicating the subject ID

vars — An optional character vector encoding the variables used in the analysis. Must equal names of the dataset in data.

groups — Deprecated. Use groupvar instead. An optional string indicating the name of the group variable in data.

groupvar — An optional string indicating the name of the group variable in data. Replaces the deprecated groups argument; if both are supplied, groupvar takes precedence with a warning.

covs — A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure covtype argument is set correctly to the type of covariances used.

cors — A sample correlation matrix, or a list/array of such matrices for multiple groups. When supplied, the matrix is treated as a covariance matrix with a warning (appropriate for standardized data). Requires nobs. Note: corinput = TRUE is not supported in var1.

means — A vector of sample means, or a list/matrix containing such vectors for multiple groups.

nobs — The number of observations used in covs and means, or a vector of such numbers of observations for multiple groups.

corinput — Logical. Not supported in var1() and will produce an error if set to TRUE.

missing — How should missingness be handled in computing the sample covariances and number of observations when data is used. Can be "auto" (default) for automatic detection, "listwise" for listwise deletion, or "pairwise" for pairwise deletion. When "auto", the function checks for missing data and switches ML to FIML, PML to PFIML, or defaults to listwise for LS estimators.

centerWithin — Logical, should the data be centered within persons (per idvar) before estimation? Defaults to FALSE (the historical behavior). Only used when raw data is supplied.

equal — A character vector indicating which matrices should be constrained equal across groups.

baseline_saturated — A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

estimator — The estimator to be used. Currently implemented are "ML" for maximum likelihood estimation, "FIML" for full-information maximum likelihood estimation, "PML" for penalized maximum likelihood estimation, "PFIML" for penalized full-information maximum likelihood estimation, "ULS" for unweighted least squares estimation, "WLS" for weighted least squares estimation, and "DWLS"

for diagonally weighted least squares estimation. When `missing = "auto"` (default), `"ML"` is automatically switched to `"FIML"` and `"PML"` to `"PFIML"` if missing data is detected.

`optimizer` The optimizer to be used. Can be one of `"nlminb"` (the default R `nlminb` function), `"ucminf"` (from the `optimr` package), `"nloptr_TNEWTON"` (preconditioned truncated Newton via `nloptr`), and `"LBFGS++"` (pure C++ L-BFGS-B). Defaults to `"nlminb"`.

`storedata` Logical, should the raw data be stored? Needed for bootstrapping (see `bootstrap`).

`standardize` Which standardization method should be used? `"none"` (default) for no standardization, `"z"` for z-scores, and `"quantile"` for a non-parametric transformation to the quantiles of the marginal standard normal distribution.

`sampleStats` An optional sample statistics object. Mostly used internally.

`covtype` If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to `"ML"` for maximum likelihood estimates (denominator n) and `"UB"` to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.

`verbose` Logical, should messages be printed?

`bootstrap` Should the data be bootstrapped? If `TRUE` the data are resampled and a bootstrap sample is created. These must be aggregated using `aggregate_bootstraps`! Can be `TRUE` or `FALSE`. Can also be `"nonparametric"` (which sets `boot_sub = 1` and `boot_resample = TRUE`) or `"case"` (which sets `boot_sub = 0.75` and `boot_resample = FALSE`).

`boot_sub` Proportion of cases to be subsampled (`round(boot_sub * N)`).

`boot_resample` Logical, should the bootstrap be with replacement (`TRUE`) or without replacement (`FALSE`)

`penalty_lambda` Numeric penalty strength for penalized ML estimation (PML/PFIML). `NA` (default) triggers automatic selection via EBIC-based grid search when a penalized estimator is used; set to a specific numeric value to use a fixed penalty strength (0 = no penalty). See `find_penalized_lambda` and `penalize`.

`penalty_alpha` Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

`penalize_matrices`
Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.

`rho_zeta` Only used when `contemporaneous = "cor"`. Either `"full"` or `"zero"`, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the contemporaneous (innovation) correlations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`SD_zeta` Only used when `contemporaneous = "cor"`. Either `"full"` or `"diag"`, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. Encodes the diagonal matrix of contemporaneous (innovation) standard deviations. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

temporal	Parameterization of the temporal effects: "raw" (default) models beta directly; "PDC" models the partial directed correlations directly, with the PDC matrix encoding from = row and to = column (the transposed orientation relative to beta, matching getmatrix(..., "PDC")). A (pattern-constrained) PDC model is fit-equivalent to the corresponding raw model, but standard errors, confidence intervals, equality constraints and modification indices then apply directly on the PDC scale.
PDC	Only used when temporal = "PDC". Model matrix for the partial directed correlations (from = row, to = column): "full", "zero", "diag", or a pattern matrix with 0s indicating elements fixed to zero, 1s indicating free elements, and higher integers indicating equality constraints.
...	Arguments sent to var1

Details

The lag-1 vector autoregression (VAR(1)) model describes a multivariate time series through a lag-0 (contemporaneous) and a lag-1 (temporal) structure. The temporal structure is captured by the regression matrix beta (whose transpose is the temporal network), relating each observation to the previous measurement occasion. The contemporaneous structure captures the associations among the residuals (innovations) within the same measurement occasion, and can be parameterized in several ways through the contemporaneous argument: as a covariance matrix ("cov"), a Cholesky decomposition ("chol"), a precision matrix ("prec"), a Gaussian graphical model ("ggm"; the partial correlation network of the residuals), or a correlation matrix with separate standard deviations ("cor"). The gvar wrapper fixes contemporaneous = "ggm" to directly estimate a graphical VAR model. Combined, the temporal and contemporaneous networks imply the lag-0 and lag-1 (block-Toeplitz) covariance structure that is fitted to the data.

Value

An object of the class psychonetrics

Author(s)

Sacha Epskamp

References

Epskamp, S., Waldorp, L. J., Mottus, R., & Borsboom, D. (2018). The Gaussian graphical model in cross-sectional and time-series data. Multivariate Behavioral Research, 53(4), 453-480.

See Also

lvm, varcov, dlvm1

Examples

```
library("dplyr")
library("graphicalVAR")

beta <- matrix(c(
```

```
  0,0.5,
  0.5,0
),2,2,byrow=TRUE)
kappa <- diag(2)
simData <- graphicalVARsim(50, beta, kappa)

# Form model:
model <- gvar(simData)

# Evaluate model:
model <- model %>% runmodel

# Parameter estimates:
model %>% parameters

# Plot the CIs:
CIplot(model,  "beta")

# Note: this example is wrapped in a dontrun environment because the data is not
# available locally.
## Not run:
# Longer example:
#
# Obtain the data from:
#
# Epskamp, S., van Borkulo, C. D., van der Veen, D. C., Servaas, M. N., Isvoranu, A. M.,
# Riese, H., & Cramer, A. O. (2018). Personalized network modeling in psychopathology:
# The importance of contemporaneous and temporal connections. Clinical Psychological
# Science, 6(3), 416-427.
#
# Available here: https://osf.io/c8wjz/

tsdata <- read.csv("Supplementary2_data.csv")

# Encode time variable in a way R understands:
tsdata$time <- as.POSIXct(tsdata$time, tz = "Europe/Amsterdam")

# Extract days:
tsdata$Day <- as.Date(tsdata$time, tz = "Europe/Amsterdam")

# Variables to use:
vars <- c("relaxed", "sad", "nervous", "concentration", "tired", "rumination",
          "bodily.discomfort")

# Estimate, prune with FDR, and perform stepup search:
model_FDRprune <- gvar(
  tsdata,
  vars = vars,
  dayvar = "Day",
  estimator = "FIML"
  ) %>%
  runmodel %>%
  prune(adjust = "fdr", recursive = FALSE) %>%
```

```
  stepup(criterion = "bic")

# Estimate with greedy stepup search:
model_stepup <- gvar(
  tsdata,
  vars = vars,
  dayvar = "Day",
  estimator = "FIML",
  omega_zeta = "zero",
  beta = "zero"
) %>%
  runmodel %>%
  stepup(greedy = TRUE, greedyadjust = "bonferroni", criterion = "bic")

# Compare models:
compare(
  FDRprune = model_FDRprune,
  stepup = model_stepup
)
# Very similar but not identical. Stepup is prefered here according to AIC and BIC

# Stepup results:
temporal <- getmatrix(model_stepup, "PDC") # PDC = Partial Directed Correlations
contemporaneous <- getmatrix(model_stepup, "omega_zeta")

# Average layout:
library("qgraph")
L <- averageLayout(temporal, contemporaneous)

# Labels:
labs <- gsub("\\.","\n",vars)

# Plot:
layout(t(1:2))
qgraph(temporal, layout = L, theme = "colorblind", directed=TRUE, diag=TRUE,
       title = "Temporal", vsize = 12, mar = rep(6,4), asize = 5,
       labels = labs)
qgraph(contemporaneous, layout = L, theme = "colorblind",
       title = "Contemporaneous", vsize = 12, mar = rep(6,4), asize = 5,
       labels = labs)

## End(Not run)
```

---

varcov *Variance-covariance family of psychonetrics models*

Description

This is the family of models that models only a variance-covariance matrix with mean structure. The `type` argument can be used to define what model is used: `type = "cov"` (default) models a

variance-covariance matrix directly, type = "chol" (alias: cholesky()) models a Cholesky decomposition, type = "prec" (alias: precision()) models a precision matrix, type = "ggm" (alias: ggm()) models a Gaussian graphical model (Epskamp, Rhemtulla and Borsboom, 2017), and type = "cor" (alias: corr()) models a correlation matrix.

Usage

```
varcov(data, type = c("cov", "chol", "prec", "ggm", "cor"),
                    sigma = "full", kappa = "full", omega = "full",
                    lowertri = "full", delta = "diag", rho = "full", SD =
                    "full", mu, tau, vars, ordered = character(0), groups,
                    groupvar, covs, cors, means, nobs, missing = "auto", equal =
                    "none", baseline_saturated = TRUE, estimator =
                    "default", likelihood = c("normal", "wishart"),
                    fixed_x = character(0),
                    optimizer, storedata = FALSE, WLS.W,
                    sampleStats, meanstructure, corinput, verbose = FALSE,
                    covtype = c("choose", "ML", "UB"), standardize =
                    c("none", "z", "quantile"), fullFIML = FALSE,
                    bootstrap = FALSE, boot_sub, boot_resample,
                    penalty_lambda = NA, penalty_alpha = 1,
                    penalize_matrices, sampling_weights)
cholesky(...)
precision(...)
prec(...)
ggm(...)
corr(...)
```

Arguments

data	A data frame encoding the data used in the analysis. Can be missing if covs and nobs are supplied.
type	The type of model used. See description.
sigma	Only used when type = "cov". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
kappa	Only used when type = "prec". Either "full" to estimate every element freely, "diag" to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.
omega	Only used when type = "ggm". Either "full" to estimate every element freely, "zero" to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element,

and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`lowertri` Only used when `type = "chol"`. Either `"full"` to estimate every element freely, `"diag"` to only include diagonal elements, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`delta` Only used when `type = "ggm"`. Either `"diag"` or `"zero"` (not recommended), or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`rho` Only used when `type = "cor"`. Either `"full"` to estimate every element freely, `"zero"` to set all elements to zero, or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`SD` Only used when `type = "cor"`. Either `"diag"` or `"zero"` (not recommended), or a matrix of the dimensions node x node with 0 encoding a fixed to zero element, 1 encoding a free to estimate element, and higher integers encoding equality constraints. For multiple groups, this argument can be a list or array with each element/slice encoding such a matrix.

`mu` Optional vector encoding the mean structure. Set elements to 0 to indicate fixed to zero constrains, 1 to indicate free means, and higher integers to indicate equality constraints. For multiple groups, this argument can be a list or array with each element/column encoding such a vector.

`tau` Optional list encoding the thresholds per variable.

`vars` An optional character vector encoding the variables used in the analysis. Must equal names of the dataset in `data`.

`groups` Deprecated. Use `groupvar` instead. An optional string indicating the name of the group variable in `data`.

`groupvar` An optional string indicating the name of the group variable in `data`. Replaces the deprecated `groups` argument; if both are supplied, `groupvar` takes precedence with a warning.

`covs` A sample variance–covariance matrix, or a list/array of such matrices for multiple groups. Make sure `covtype` argument is set correctly to the type of covariances used.

`cors` A sample correlation matrix, or a list/array of such matrices for multiple groups. When supplied, `corinput` defaults to `TRUE` and the matrix is used in place of `covs`. Requires `nobs`.

`means` A vector of sample means, or a list/matrix containing such vectors for multiple groups.

`nobs` The number of observations used in `covs` and `means`, or a vector of such numbers of observations for multiple groups.

`covtype` If 'covs' is used, this is the type of covariance (maximum likelihood or unbiased) the input covariance matrix represents. Set to `"ML"` for maximum likelihood estimates (denominator n) and `"UB"` to unbiased estimates (denominator n-1). The default will try to find the type used, by investigating which is most likely to result from integer valued datasets.

`missing` How should missingness be handled in computing the sample covariances and number of observations when `data` is used. Can be `"auto"` (default) for automatic detection, `"listwise"` for listwise deletion, or `"pairwise"` for pairwise deletion. When `"auto"`, the function checks for missing data and switches ML to FIML, PML to PFIML, or defaults to listwise for LS estimators.

`equal` A character vector indicating which matrices should be constrained equal across groups.

`baseline_saturated`
A logical indicating if the baseline and saturated model should be included. Mostly used internally and NOT Recommended to be used manually.

`estimator` The estimator to be used. Currently implemented are `"ML"` for maximum likelihood estimation, `"FIML"` for full-information maximum likelihood estimation, `"PML"` for penalized maximum likelihood estimation, `"PFIML"` for penalized full-information maximum likelihood estimation, `"ULS"` for unweighted least squares estimation, `"WLS"` for weighted least squares estimation, and `"DWLS"` for diagonally weighted least squares estimation. `"WLSMV"` is accepted as a synonym for `"DWLS"` (both use DWLS estimation with a mean-and-variance adjusted scaled test statistic). When `missing = "auto"` (default), `"ML"` is automatically switched to `"FIML"` and `"PML"` to `"PFIML"` if missing data is detected. Defaults to `"ML"` for continuous data and `"DWLS"` when ordinal variables are specified via `ordered`.

`likelihood` The Gaussian likelihood scaling for maximum-likelihood estimation. `"normal"` (the default) uses the n (biased) sample-covariance denominator, so the chi-square is $N\hat{F}$ and the parameter covariance is Info^{-1}/N. `"wishart"` uses the $n-1$ (unbiased) sample covariance per group, so the chi-square uses $(n_g - 1)$ multipliers and the standard errors are inflated by $\sqrt{n_g/(n_g-1)}$, matching `lavaan`'s `likelihood = "wishart"` (Rosseel, 2012). Only available for complete-data maximum likelihood (`estimator = "ML"`); it errors for `FIML`, least-squares and ordinal estimators and for raw time-series input. Default `"normal"` reproduces the behaviour of previous versions exactly.

`fixed_x` Character vector of exogenous variable names whose means and mutual (co)variances are fixed to their sample values and excluded from the free-parameter count and the degrees-of-freedom statistic count, matching `lavaan`'s `sem(..., fixed.x = TRUE)` (Rosseel, 2012). The model is thereby conditioned on these variables, which relaxes the distributional (multivariate-normality) assumption on them; the cross-covariances between the fixed-x and the endogenous variables remain free. Only supported for `type = "cov"` and complete-data `estimator = "ML"`. The reported log-likelihood is the conditional log-likelihood (of the endogenous variables given x). Default `character(0)` (no fixed.x; behaviour unchanged). The baseline and saturated reference models are conditioned on the exogenous block as well, so the incremental fit indices (CFI/TLI/NFI/...) and the absolute fit measures (chi-square, df, RMSEA, AIC, BIC, log-likelihood) and all

estimates and standard errors match lavaan.

sampling_weights
: Optional single column name in data giving (survey) SAMPLING weights, enabling pseudo-maximum-likelihood estimation. The moments are computed as weighted means/covariances (weights normalized so that they sum, over all groups, to the total sample size – lavaan's sampling.weights.normalization = "total"), and the model is fit with robust (MLR) Huber-White sandwich standard errors and a Yuan-Bentler-Mplus scaled test statistic, exactly matching lavaan's sampling.weights=. Point estimates, standard errors, the scaled chi-square and the robust fit indices agree with lavaan to numerical precision (single- and multi-group). Requires complete-data continuous input (raw data); not supported for ordinal data, missing data (FIML), or likelihood = "wishart". References: Asparouhov (2005); Rosseel (2012).

optimizer
: The optimizer to be used. Can be one of "nlminb" (the default R nlminb function), "ucminf" (from the optimr package), "nloptr_TNEWTON" (preconditioned truncated Newton via nloptr), and "LBFGS++" (pure C++ L-BFGS-B). Defaults to "nlminb".

storedata
: Logical, should the raw data be stored? Needed for bootstrapping (see bootstrap).

standardize
: Which standardization method should be used? "none" (default) for no standardization, "z" for z-scores, and "quantile" for a non-parametric transformation to the quantiles of the marginal standard normal distribution.

WLS.W
: Optional WLS weights matrix.

sampleStats
: An optional sample statistics object. Mostly used internally.

verbose
: Logical, should progress be printed to the console?

ordered
: A vector with strings indicating the variables that are ordered categorical, or set to TRUE to model all variables as ordered categorical.

meanstructure
: Logical, should the meanstructure be modeled explicitly?

corinput
: Logical, is the input a correlation matrix?

fullFIML
: Logical, should row-wise FIML be used? Not recommended!

bootstrap
: Should the data be bootstrapped? If TRUE the data are resampled and a bootstrap sample is created. These must be aggregated using aggregate_bootstraps! Can be TRUE or FALSE. Can also be "nonparametric" (which sets boot_sub = 1 and boot_resample = TRUE) or "case" (which sets boot_sub = 0.75 and boot_resample = FALSE).

boot_sub
: Proportion of cases to be subsampled (round(boot_sub * N)).

boot_resample
: Logical, should the bootstrap be with replacement (TRUE) or without replacement (FALSE)

penalty_lambda
: Numeric penalty strength for penalized ML estimation (PML/PFIML). Default is NA, which triggers automatic lambda selection via EBIC grid search when estimator = "PML" or "PFIML" (see find_penalized_lambda). Set to a specific numeric value (e.g., 0.1) for manual lambda, or 0 for no penalty.

penalty_alpha
: Elastic net mixing parameter: 1 = LASSO (default), 0 = ridge.

penalize_matrices
: Character vector of matrix names to penalize. If missing, defaults are selected based on the model type.

...
: Arguments sent to varcov

Details

The model used in this family is:

$\text{var}(\boldsymbol{y}) = \boldsymbol{\Sigma}$

$\mathcal{E}(\boldsymbol{y}) = \boldsymbol{\mu}$

in which the covariance matrix can further be modeled in four ways. With `type = "chol"` as Cholesky decomposition:

$\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$,

with `type = "prec"` as Precision matrix:

$\boldsymbol{\Sigma} = \boldsymbol{K}^{-1}$,

with `type = "ggm"` as Gaussian graphical model:

$\boldsymbol{\Sigma} = \boldsymbol{\Delta}(\boldsymbol{I} - \boldsymbol{\Omega})^(- 1)\boldsymbol{\Delta}$,

and finally with `type = "cor"` as a correlation matrix scaled by standard deviations:

$\boldsymbol{\Sigma} = \boldsymbol{D}\boldsymbol{R}\boldsymbol{D}$,

in which $\boldsymbol{D}$ is a diagonal matrix of standard deviations and $\boldsymbol{R}$ a correlation matrix.

Value

An object of the class psychonetrics

Author(s)

Sacha Epskamp

References

Epskamp, S., Rhemtulla, M., & Borsboom, D. (2017). Generalized network psychometrics: Combining network and latent variable models. Psychometrika, 82(4), 904-927.

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

See Also

`lvm`, `var1`, `dlvm1`, `penalize` for manual penalty control, `find_penalized_lambda` for automatic lambda selection, `refit` for post-selection inference after penalized estimation.

Examples

```
# Load bfi data from psych package:
library("psychTools")
data(bfi)

# Also load dplyr for the pipe operator:
library("dplyr")

# Let's take the agreeableness items, and gender:
ConsData <- bfi %>%
```

```
  select(A1:A5, gender) %>%
 na.omit # Let's remove missingness (or use missing = "auto" default which auto-selects FIML)

# Define variables:
vars <- names(ConsData)[1:5]

# Saturated estimation:
mod_saturated <- ggm(ConsData, vars = vars)

# Run the model:
mod_saturated <- mod_saturated %>% runmodel

# We can look at the parameters:
mod_saturated %>% parameters

# Labels:
labels <- c(
  "indifferent to the feelings of others",
  "inquire about others' well-being",
  "comfort others",
  "love children",
  "make people feel at ease")

# Plot CIs:
CIplot(mod_saturated,  "omega", labels = labels, labelstart = 0.2)



# We can also fit an empty network:
mod0 <- ggm(ConsData, vars = vars, omega = "zero")

# Run the model:
mod0 <- mod0 %>% runmodel

# We can look at the modification indices:
mod0 %>% MIs

# To automatically add along modification indices, we can use stepup:
mod1 <- mod0 %>% stepup

# Let's also prune all non-significant edges to finish:
mod1 <- mod1 %>% prune

# Look at the fit:
mod1 %>% fit

# Compare to original (baseline) model:
compare(baseline = mod0, adjusted = mod1)

# We can also look at the parameters:
mod1 %>% parameters

# Or obtain the network as follows:
```

```
getmatrix(mod1, "omega")

# Penalized GGM estimation with automatic lambda selection:
mod_pml <- ggm(ConsData, vars = vars, estimator = "PML")
mod_pml <- mod_pml %>% runmodel

# Check selected lambda:
mod_pml@optim$lambda_search

# Obtain the sparse network:
getmatrix(mod_pml, "omega")

# Wishart likelihood (n-1 denominator), matching lavaan likelihood = "wishart":
mod_wis <- varcov(ConsData, vars = vars, type = "cov", likelihood = "wishart")
mod_wis <- mod_wis %>% runmodel

# fixed.x: condition on A1 and A2 as exogenous covariates (their means and
# mutual covariances are fixed to the sample values, excluded from npar/df):
mod_fx <- varcov(ConsData, vars = vars, type = "cov",
                 fixed_x = c("A1", "A2"))
mod_fx <- mod_fx %>% runmodel
```

`write_psychonetrics` *Write comprehensive model output to a text file*

Description

Writes a comprehensive plain-text output file containing all model information, similar to Mplus or LISREL output. The file includes model specification, sample information, parameter estimates, fit measures, model matrices, modification indices, and the model logbook. This file can be shared as supplementary material in publications.

Usage

```
write_psychonetrics(x, file = "psychonetrics_output.txt",
                    matrices = TRUE, MIs = TRUE, logbook = TRUE)
```

Arguments

`x`	A `psychonetrics` model.
`file`	Character string specifying the path to the output file. Defaults to `"psychonetrics_output.txt"`.
`matrices`	Logical. Should the full estimated model matrices be included in the output? Defaults to `TRUE`. Set to `FALSE` for more compact output.
`MIs`	Logical. Should modification indices be included? Only included if modification indices have been computed via `addMIs`. Defaults to `TRUE`.
`logbook`	Logical. Should the model logbook be included? Defaults to `TRUE`.

Value

Invisibly returns the file path of the written output.

Author(s)

Sacha Epskamp

Examples

```
library("dplyr")

# Load data:
data(StarWars)

# Simple CFA:
Lambda <- matrix(1, 4)
mod <- lvm(StarWars, lambda = Lambda, vars = c("Q1","Q5","Q6","Q7"),
           identification = "variance", latents = "Originals")
mod <- mod %>% runmodel %>% addfit

# Write output:
write_psychonetrics(mod, file = tempfile(fileext = ".txt"))
```

Index

∗ **classes**
psychonetrics-class, 130
psychonetrics_bootstrap-class, 132
psychonetrics_log-class, 133
∗ **datasets**
Jonas, 54
NA2020, 109
StarWars, 147
∗ **utilities**
loop_psychonetrics, 57

addfit *(*psychonetrics_update*)*, 133
addMIs, *175*
addMIs *(*psychonetrics_update*)*, 133
addSEs *(*psychonetrics_update*)*, 133
aggregate_bootstraps, 6, *11*, *28*, *52*, *58*, *64*, *86*, *94*, *115*, *142*, *158*, *165*, *172*
allequal, 7
averageLayout, *127*

bifactor, 8
BlumeCapel, *6*, 9, *53*
bootstrap, 13, *28*, *58*, *64*, *93*, *115*

changedata, 14
checkFisher *(*diagnostics*)*, 21
checkJacobian *(*diagnostics*)*, 21
cholesky *(*varcov*)*, 168
CIplot, 15, *58*, *128*, *138*, *139*
compare, 17, *31*, *117*, *139*
corr *(*varcov*)*, 168
cov, *14*
covML, 20
covMLtoUB *(*covML*)*, 20
covUBtoML *(*covML*)*, 20

diagnostics, 21
diagonalizationMatrix *(*duplicationMatrix*)*, 32
dlvm *(*dlvm1*)*, 22
dlvm1, *6*, 22, *35*, *36*, *89*, *97*, *98*, *103*, *111*, *117*, *127*, *139*, *166*, *173*
duplicationMatrix, 32

eliminationMatrix *(*duplicationMatrix*)*, 32
emergencystart, 33
equalityScoreTest, 33, *122*
esa, 35
esa_manual *(*esa*)*, 35

factorscores, 37
find_penalized_lambda, *12*, *29*, 37, *52*, *65*, *116*, *125*, *141*, *142*, *158*, *165*, *172*, *173*
fit, 40, *108*, *131*, *149*
fixpar, *8*, 41, *138*
fixstart, 42
freepar *(*fixpar*)*, 41
fromlavaan, 43, *152*

generate, 45
getmatrix, *29*, 46, *126–128*, *138*, *139*
getVCOV, 48
ggm, *38*, *107*, *125*, *127*
ggm *(*varcov*)*, 168
groupequal, *7*, *8*, 49, *122*
groupfree, *122*
groupfree *(*groupequal*)*, 49
gvar *(*var1*)*, 162

identify, *122*
identify *(*psychonetrics_update*)*, 133
intersectionmodel, *122*
intersectionmodel *(*unionmodel*)*, 160
Ising, *6*, *9*, *10*, *12*, 50, *127*

Jonas, 54

latentgrowth, 55
lnm, *127*

lnm *(lvm)*, 59
logbook, 57
loop_psychonetrics, 57
lrnm *(lvm)*, 59
lvm, *6*, *9*, *16*, *38*, *43–45*, *47*, *55*, 59, *77*, *89*, *127*, *134*, *136–139*, *150–152*, *166*, *173*

meta_ggm *(meta_varcov)*, 82
meta_gvar *(meta_var1)*, 78
meta_lvm, *6*, 73, *81*
meta_var *(meta_var1)*, 78
meta_var1, *6*, 78
meta_varcov, *6*, *77*, *81*, 82
MIs, *35*, 87
ml_corr *(ml_varcov)*, 104
ml_ggm *(ml_varcov)*, 104
ml_gvar *(ml_var1)*, 98
ml_gvar1 *(ml_var1)*, 98
ml_lnm *(ml_lvm)*, 89
ml_lrnm *(ml_lvm)*, 89
ml_lvm, *6*, 89, *102–104*, *107*, *127*
ml_rnm *(ml_lvm)*, 89
ml_ts_lvgvar *(ml_tsdlvm1)*, 97
ml_tsdlvm1, *6*, 97
ml_var *(ml_var1)*, 98
ml_var1, *81*, *97*, 98
ml_varcov, 104
modelsearch, 108

NA2020, 109

panel_lvgvar *(dlvm1)*, 22
panelgvar, *16*, *22*, *47*, *127*, *137–139*
panelgvar *(panelvar)*, 111
panellvgvar *(dlvm1)*, 22
panelvar, *16*, *22*, *30*, *31*, *35*, *36*, *47*, *98*, *101–103*, 111, *127*, *137–139*
parameters, 119
parequal, *8*, 120
partialprune, *35*, 121
penalize, *12*, *29*, *39*, *52*, *65*, *116*, 123, *142*, *158*, *165*, *173*
penaltyVector *(penalize)*, 123
plot.esa *(esa)*, 35
plot.esa_manual *(esa)*, 35
plot.psychonetrics, 126
prec *(varcov)*, 168
precision *(varcov)*, 168
print.esa *(esa)*, 35
print.esa_manual *(esa)*, 35
print.psychonetrics_compare *(compare)*, 17
print.ri_clpm_search *(ri_clpm)*, 134
prune, *109*, *122*, 129, *138*, *150*
psychonetrics *(psychonetrics-package)*, 3
psychonetrics-class, *8*, *9*, *14*, *31*, *39*, *42*, *43*, *49*, *56*, *66*, *76*, *81*, *86*, *96*, *98*, *103*, *107*, *109*, *117*, *121*, *122*, *125*, *129*, 130, *134*, *142*, *145*, *146*, *149*, *153*, *159*, *161*
psychonetrics-package, 3
psychonetrics_bootstrap-class, 132
psychonetrics_log-class, 133
psychonetrics_samplestats-class *(psychonetrics-class)*, 130
psychonetrics_update, 133

qgraph, *126*, *128*
qgraph.loadings, *126*, *128*

refit, *39*, *142*, *173*
refit *(penalize)*, 123
resid,psychonetrics-method *(psychonetrics-class)*, 130
residuals,psychonetrics-method *(psychonetrics-class)*, 130
ri_clpm, *6*, *15*, *16*, *31*, *46*, *47*, *117*, *126*, *127*, 134
ri_clpm_search *(ri_clpm)*, 134
ri_clpm_stationary, *16*, *47*, *127*
ri_clpm_stationary *(ri_clpm)*, 134
ri_clpn, *6*
ri_clpn *(ri_clpm)*, 134
ri_clpn_search *(ri_clpm)*, 134
ri_clpn_stationary *(ri_clpm)*, 134
rnm *(lvm)*, 59
runmodel, *37*, *39*, *43*, *95*, *103*, *108*, *125*, *129*, *136*, 140, *149*, *161*

sessionInfo-class *(psychonetrics-class)*, 130
setestimator, *40*, *142*, 143
setoptimizer *(setestimator)*, 143
setverbose, 146
show,psychonetrics-method *(psychonetrics-class)*, 130

show,psychonetrics_bootstrap-method
(psychonetrics_bootstrap-class), 132
show,psychonetrics_log-method
(psychonetrics_log-class), 133
simplestructure, *90*, 147
StarWars, 147
stepup, *109*, *130*, 148
summary,psychonetrics-method
(psychonetrics-class), 130

tolavaan, *45*, *127*, 150
transmod, 152
ts_lvgvar (tsdlvm1), 154
tsdlvm (tsdlvm1), 154
tsdlvm1, *6*, *127*, 154

unionmodel, *122*, 160
unpenalize (penalize), 123
update_baseline
(update_baseline_saturated), 161
update_baseline_saturated, 161
update_saturated
(update_baseline_saturated), 161
usecpp (setestimator), 143

var1, *6*, *81*, *99*, *100*, *102*, *103*, *127*, 162, *173*
varcov, *6*, *62*, *63*, *104*, *107*, *127*, *166*, 168

write_psychonetrics, 175