Package 'semhelpinghands'

November 2, 2024

Title Helper Functions for Structural Equation Modeling

Version 0.1.12

Description An assortment of helper functions for doing structural equation modeling, mainly by 'lavaan' for now. Most of them are time-saving functions for common tasks in doing structural equation modeling and reading the output. This package is not for functions that implement advanced statistical procedures. It is a light-weight package for simple functions that do simple tasks conveniently, with as few dependencies as possible.

URL https://sfcheung.github.io/semhelpinghands/

BugReports https://github.com/sfcheung/semhelpinghands/issues

Depends R (>= 4.1.0)

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Suggests rmarkdown, knitr, semTools, testthat (>= 3.0.0)

Config/testthat/parallel true

Config/testthat/edition 3

Imports lavaan, boot, rlang, ggplot2, ggrepel, utils

VignetteBuilder knitr

NeedsCompilation no

Author Shu Fai Cheung [aut, cre] (<https://orcid.org/0000-0002-9871-9448>)

Maintainer Shu Fai Cheung <shufai.cheung@gmail.com>

Repository CRAN

Date/Publication 2024-11-02 06:10:02 UTC

Contents

add_exo_cov . . . 2
add_sig . . . 4
annotate_matrices . . . 6
compare_estimators . . . 8
dvs_ivs . . . 9
filter_by . . . 10
fitMeasures_by_models . . . 12
group_by_groups . . . 13
group_by_models . . . 15
group_estimates . . . 17
plot_boot . . . 18
plot_models_fm . . . 22
print.est_table . . . 24
print.fit_by_models . . . 25
print.std_solution_boot . . . 26
ptable_to_syntax . . . 28
record_history . . . 30
show_cfi . . . 32
show_more_options . . . 33
show_options . . . 35
simple_mediation . . . 37
sort_by . . . 38
standardizedSolution_boot_ci . . . 39
store_boot_def . . . 42
vec_rsquare . . . 44

Index **48**

`add_exo_cov` *Add Covariances Between Exogenous Variables*

Description

It generates the 'lavaan' model syntax for exogenous variables in a `lavaan` model.

Usage

```
add_exo_cov(model, FUN = "sem", print = TRUE)

auto_exo_cov(model, FUN = "sem", print = TRUE)
```

Arguments

model	The model syntax to which the covariances are to be added.
FUN	Name (as string) of the lavaan wrapper to be called. Normally should be "sem", the default.
print	Logical. Whether the generated syntax should also be printed by cat(). Default is TRUE.

Details

The function lavaan::sem() usually will set covariances between "exogenous" variables free when fixed.x = FALSE ("exogenous" is defined here as variables that appear on the right hand side but not on the left hand side of the ~ operator‘). However, if a covariance between the residual term of an endogenous variable and an exogenous variable is manually set to free, lavaan::sem() may not set the aforementioned covariances free. Users will need to free them manually, and there may be a lot of them in some models.

This function gets a model syntax and generates the syntax for these covariances. Users can then inspect it, modify it if necessary, and then copy and paste it to the model syntax.

Value

add_exo_cov() returns a one-element character vector of the syntax, with lines separated by "\n". The generated syntax is appended to the input model syntax.

auto_exo_cov() returns a one-element character vector of the generated syntax, with lines separated by "\n".

Functions

- add_exo_cov(): Add covariances between exogenous variables to the model syntax and than return the modified model syntax.
- auto_exo_cov(): Generate the model syntax for the covariances between exogenous variables.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)
set.seed(8976223)
n <- 100
x <- rnorm(n)
m <- .5 * x + rnorm(n, 0, sqrt(.4))
z <- rnorm(n)
y <- .4 * m + .3 * z * m + rnorm(n, 0, .5)
dat <- data.frame(x, m, z, y)
dat$zm <- dat$z * dat$m
mod0 <-
```

```
"
m ~ x
y ~ m + z + zm
m ~~ z + zm
"
fit <- sem(mod0, dat, fixed.x = FALSE)

# Add covariances. Also printed by default.
mod0_cov <- add_exo_cov(mod0)

# Fit the model
fit_cov <- sem(mod0_cov, dat, fixed.x = FALSE)

# Manually adding the covariances
mod1 <-
"
m ~ x
y ~ m + z + zm
m ~~ z + zm
z  ~~ zm + x
zm ~~ x
"
fit1 <- sem(mod1, dat, meanstructure = TRUE, fixed.x = FALSE)

# Compare the results

# No manual covariances
fit

# Automatically generated covariances
fit_cov

# Manually added covariances
fit1
```

---

add_sig *Add Significant Test Results*

Description

It inserts columns to denote whether a parameter is significant.

Usage

```
add_sig(object, ..., standardized = FALSE, na_str = "", use = "pvalue")
```

Arguments

object	A 'lavaan'-class object or the output of lavaan::parameterEstimates() or lavaan::standardizedSolution(). May also work on an est_table-class object returned by functions like group_by_dvs() but there is no guarantee.
...	Optional arguments to be passed to lavaan::parameterEstimates() or lavaan::standardizedSolut
standardized	Whether standardized solution is needed. If TRUE, lavaan::standardizedSolution() will be called. If FALSE, the default, lavaan::parameterEstimates() will be called. Ignored if a table if estimates is supplied.
na_str	The string to be used for parameters with no significant tests. For example, fixed parameters. Default is "".
use	A character vector of one or more strings. If "pvalue" is in the vector, *p*-values will be used. If "ci" is in the vector, confidence intervals appeared in ci.lower and ci.upper will be used. If "boot.ci" is in the vector and the columns boot.ci.lower and boot.ci.upper are available, these columns will be used. Note that ci.lower and ci.upper can also be bootstrap confidence intervals in some tables if se = "boot" is used.

Details

The function calls lavaan::parameterEstimates() or lavaan::standardizedSolution() and checks the columns pvalue, ci.lower and ci.upper, and/or boot.ci.lower and boot.ci.upper and then inserts columns to denote for each parameter estimate whether it is significant based on the requested criteria.

Value

The output of lavaan::parameterEstimates() or lavaan::standardizedSolution(), with one or two columns inserted after the parameter estimates to denote the significant test results.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

See Also

lavaan::parameterEstimates() and lavaan::standardizedSolution()

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
```

```
y ~ b*m
ab := a*b
'
fit <- sem(model, data = dat, fixed.x = FALSE)

# Add "*" based on 'pvalue'
add_sig(fit)

# Add "*" for standardized solution
add_sig(fit, standardized = TRUE)

# Add "*" based on confidence interval
add_sig(fit, use = "ci")

# Add "*" for standardized solution based on confidence interval
add_sig(fit, standardized = TRUE, use = "ci")

# Add "*" for standardized solution based on confidence interval
# and 'pvalue'.
add_sig(fit, standardized = TRUE, use = c("ci", "pvalue"))

# Can also accept a parameter estimates table
est <- parameterEstimates(fit)
add_sig(est)

# So it can be used with some other functions in semhelpinghands
add_sig(filter_by(est, op = "~"))

# Piping can also be used
est |> filter_by(op = "~") |>
       add_sig()
```

`annotate_matrices` *Annotate the Matrices of a 'lavaan' Model*

Description

Label the elements of the model matrices in a lavaan model.

Usage

```
annotate_matrices(fit)

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

Arguments

fit	The output of lavaan::lavaan() or its wrappers, such as lavaan::cfa() and lavaan::sem().
x	A 'annotate_matrices'-class object. The output of annotate_matrices().
...	Optional arguments. To be passed to the default print method.

Details

This function annotates the model matrices in a 'lavaan'-class object. This function is not to be used in analysis. It is a learning tool, for learners to understand the relation between the model matrices and the model parameters.

It currently supports a single-level single-group model only.

Value

annotate_matrices() returns an annotate_matrices-class object, which is a list of model matrices, with elements annotated:

- If a parameter is free, then it is represented by "lhs-op-rhs" according to the parameter estimate data frame.
- If a parameter is fixed but appears in the parameter table, it is represented by "(lhs-op-rhs = x)" , x the value it is fixed to.
- If a parameter is fixed to zero but not in the parameter table, then it is represented by 0.

The print-method return the input, x. It was called for its side-effect.

Methods (by generic)

- print(annotate_matrices): The print method of the output of annotate_matrices()

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
# Adapted from https://lavaan.ugent.be/tutorial/cfa.html

library(lavaan)
HS.model <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
'
fit_cfa <- cfa(HS.model,
               data = HolzingerSwineford1939)
annotate_matrices(fit_cfa)
```

`compare_estimators` *Refit a 'lavaan'-Model by Several Estimators*

Description

Refit a model in 'lavaan' by several lavaan-supported estimators

Usage

```
compare_estimators(object, estimators = NULL)

se_ratios(fit_list, reference = NULL)
```

Arguments

`object`	A 'lavaan'-class object.
`estimators`	A character vector of the estimator supported by the `estimator` argument of `lavaan::lavaan()` and its wrappers, such as `lavaan::sem()` and `lavaan::cfa()`.
`fit_list`	The output of `compare_estimators()`.
`reference`	The name of the reference method (ratios will be equal to one). Must be one of the estimator used on `compare_estimators()`. If `NULL`, the first estimator will be used.

Details

The function simply uses `lapply()` and `update()` to rerun the analysis once for each of the estimator using `update(object, estimator = "x"`, `x` being the estimator.

The results can then be compared using `group_by_models()`.

Value

A list of `lavaan` outputs, each of them is an update of the original output using one of the estimators.

Functions

- `compare_estimators()`: Refit the model with different estimators.
- `se_ratios()`: A wrapper of `group_by_models()` that computes the ratios of standard errors of different methods to those of one method.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448.

See Also

`group_by_models()`

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
y ~ b*m
ab := a*b
'

fit <- sem(model, data = dat, fixed.x = FALSE)

# Refit the model by three different estimators
fit_more <- compare_estimators(fit, estimator = c("GLS", "MLR", "ML"))

# Use group_by_models to compare the estimates
group_by_models(fit_more, col_names = c("est", "pvalue"))

# Use se_ratios to compare standard errors
se_ratios(fit_more, reference = "ML")
```

---

dvs_ivs *Sample Dataset: 3 Predictors and 3 Outcomes*

Description

A path model with three predictors and three outcomes, for illustration.

Usage

```
dvs_ivs
```

Format

A data frame with 100 rows and 7 variables:

y1 Outcome variable 1. Numeric.

y2 Outcome variable 2. Numeric.

y3 Outcome variable 3. Numeric.

x1 Predictor 1. Numeric.

x2 Predictor 2. Numeric.

x3 Predictor 3. Numeric.

gp Group variable: "gp1" or "gp2". String.

Examples

```
data(dvs_ivs)
library(lavaan)
mod <-
"
y1 ~ x1 + x2 + x3
y2 ~ x1 + x3
y3 ~ y2 + x2
"
fit <- sem(mod, dvs_ivs)
parameterEstimates(fit)
fit_gp <- sem(mod, dvs_ivs, group = "gp")
parameterEstimates(fit_gp)
```

`filter_by` *Filter a Parameter Estimates Table*

Description

Filter parameter estimates table and similar tables in`lavaan` by common fields such as `op` (operator).

Usage

```
filter_by(object, op = NULL, lhs = NULL, rhs = NULL, group = NULL, fit = NULL)
```

Arguments

`object`	The output of `lavaan::parameterEstimates()`, `lavaan::standardizedSolution()`, or a `lavaan.data.frame` object. May also work on an `est_table`-class object returned by functions like `group_by_dvs()` but there is no guarantee.
`op`	A character vector of the operators (`op`) for filtering. Common operators are `"~"`, `"~~"`, `"=~"`, `":="`, and `"~1"`.
`lhs`	A character vector of names in the `lhs` column.
`rhs`	A character vector of names in the `rhs` column.
`group`	A vector of either the group numbers in the `group` column of the labels of the groups. If labels are supplied, the original fit object must be supplied for extracting the group labels.
`fit`	The original fit object. Usd when `group` is a vector of the group labels.

Details

This functions accepts the output of `lavaan::parameterEstimates()` and `lavaan::standardizedSolution()` and filter the rows by commonly used field.

Value

The filtered version of the input object.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
y ~ b*m
ab := a*b
'
fit <- sem(model, data = dat, fixed.x = FALSE)

model_gp <-
'
m ~ c(a1, a2)*x
y ~ c(b1, b2)*m
a1b1 := a1*b1
a2b2 := a2*b2
'
dat$gp <- sample(c("gp1", "gp2"), n, replace = TRUE)
fit_gp <- sem(model_gp, dat, group = "gp", warn = FALSE)

est <- parameterEstimates(fit)
est_gp <- parameterEstimates(fit_gp)

filter_by(est, op = "~")

filter_by(est, op = "~", lhs = "y")

filter_by(est, rhs = c("m", "x"), op = "~")

filter_by(est_gp, group = 2)

# If the fit object is supplied, can filter
# by group label
filter_by(est_gp, group = "gp2", fit = fit_gp)
filter_by(est_gp, group = "gp2", fit = fit_gp, op = "~")

# Select user-defined parameters
filter_by(est_gp, op = ":=")
```

```
# Can be used with some other functions in semhelpinghands
# Piping can also be used
est_gp |> filter_by(op = "~", group = "gp2", fit = fit_gp) |>
          add_sig()
```

`fitMeasures_by_models` *Fit Measures By Models*

Description

Groups fit measures into a table with models as columns.

Usage

```
fitMeasures_by_models(object_list, ...)
```

Arguments

`object_list`	A named list of 'lavaan'-class objects.
`...`	Optional arguments to be passed to `lavaan::fitMeasures()`.

Details

It call `lavaan::fitMeasures()` to compute for each model the fit measures supported by `lavaan`, and combine them into a data frame. Users can then use the print method (`print.fit_by_models()`) to customize the printout.

To be consist with full `lavaan` output, the names used in `lavaan::fitMeasures()` are used.

This function is intended for a simple and compact table of fit measures for quick preview. For a well-organized layout, call `lavaan::fitMeasures()` and set `output` to `"text"`.

Value

A data-frame-like object of the class `fit_by_models`, which has a `print` method (see `print.fit_by_models()`).

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448.

See Also

`lavaan::fitMeasures()`

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model1 <-
'
m ~ a*x
y ~ b*m
ab := a*b
'
fit1 <- sem(model1, data = dat, fixed.x = FALSE)
model2 <-
'
m ~ a*x
y ~ b*m + x
ab := a*b
'
fit2 <- sem(model2, data = dat, fixed.x = FALSE)

fitMeasures_by_models(list(no_direct = fit1,
                           direct = fit2))
```

---

group_by_groups *Group Estimates By Groups*

Description

Groups parameter estimates or other information such as *p*-values into a table with groups as columns and parameters as rows.

Usage

```
group_by_groups(
  object,
  ...,
  col_names = "est",
  group_first = TRUE,
  group_labels = NULL,
  fit = NULL,
  use_standardizedSolution = FALSE
)
```

Arguments

object	A 'lavaan'-class object or the output of lavaan::parameterEstimates() or lavaan::standardizedSolution().
...	Optional arguments to be passed to lavaan::parameterEstimates(). Ignored if object is an output of lavaan::parameterEstimates() or lavaan::standardizedSolution().
col_names	A vector of the column names in the parameter estimate tables to be included. Default is "est".
group_first	If TRUE, the columns will be grouped by groups first and then by columns in the parameter estimates tables. Default is TRUE.
group_labels	A character vector of group labels. Will be assigned to group id = 1, 2, 3, etc. If not provided. will try to be retrieved from object if it is a lavaan::lavaan object.
fit	Optional. A lavaan::lavaan object. If object is a parameter estimates table and group_labels is NULL, it will try to retrieve the group labels from fit is supplied.
use_standardizedSolution	If TRUE and object is not an estimates table, then lavaan::standardizedSolution() will be used to generate the table. If FALSE, the default, then lavaan::parameterEstimates() will be used if necessary.

Value

A data-frame-like object of the class est_table.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)
set.seed(5478374)
n <- 100
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
city <- sample(c("City Alpha", "City Beta"), 100,
               replace = TRUE)
dat <- data.frame(x = x, y = y, m = m, city = city)
model <-
'
m ~ c(a1, a2)*x
y ~ c(b1, b2)*m
a1b1 := a1*b1
a2b2 := a2*b2
'
fit <- sem(model, data = dat, fixed.x = FALSE,
           group = "city")
(est <- parameterEstimates(fit))
```

```
# Group them by groups
group_by_groups(fit)

# Can also work on a parameter estimates table
# To have group labels, need to supply the fit object
group_by_groups(est, fit = fit)

# Can be used with some other functions in semhelpinghands
# when used on a parameter estimates table
group_by_groups(filter_by(est, op = "~"), fit = fit)

# Also support piping
est |> filter_by(op = "~") |>
       group_by_groups(fit = fit)
```

`group_by_models` *Group Estimates By Models*

Description

Groups parameter estimates or other information such as *p*-values into a table with models as columns.

Usage

```
group_by_models(
  object_list,
  ...,
  col_names = "est",
  group_first = FALSE,
  model_first = TRUE,
  use_standardizedSolution = FALSE
)
```

Arguments

`object_list`	A named list of 'lavaan'-class objects, a named list of the output of `lavaan::parameterEstimates()`, or a named list of the output of `lavaan::standardizedSolution()`.
`...`	Optional arguments to be passed to `lavaan::parameterEstimates()`. Ignored if the elements in object_list are the results of `lavaan::parameterEstimates()` or `lavaan::standardizedSolution()`.
`col_names`	A vector of the column names in the parameter estimate tables to be included. Default is `"est"`, or `"est.std"` if `use_standardizedSolution` is `TRUE`.
`group_first`	If `TRUE`, the rows will be grouped by groups first and then by parameters. Ignored if the model has only one group. Default is `FALSE`.
`model_first`	If `TRUE`, the columns will be grouped by models first and then by columns in the parameter estimates tables. Default is `TRUE`.

use_standardizedSolution
: If TRUE and object_list is not a list of estimates tables, then lavaan::standardizedSolution() will be used to generate the table. If FALSE, the default, then lavaan::parameterEstimates() will be used if necessary.

Value

A data-frame-like object of the class est_table.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448 Inspired by the proposal Rönkkö posted in a GitHub https://github.com/simsem/semTools/issues/24#issue-235172313 for semTools. I want something simple for a quick overview and so I wrote this function.

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model1 <-
'
m ~ a*x
y ~ b*m
ab := a*b
'
fit1 <- sem(model1, data = dat, fixed.x = FALSE)
model2 <-
'
m ~ a*x
y ~ b*m + x
ab := a*b
'
fit2 <- sem(model2, data = dat, fixed.x = FALSE)
parameterEstimates(fit1)
parameterEstimates(fit2)
group_by_models(list(no_direct = fit1,
                     direct = fit2),
                col_names = c("est", "pvalue"))
# Can also be used with some other functions in
# semhelpinghands
group_by_models(list(no_direct = fit1,
                     direct = fit2),
                col_names = c("est", "pvalue")) |>
                filter_by(op = "~")
```

group_estimates *Group Estimates By Dependent or Independent Variables*

Description

Groups parameter estimates or other information such as p-values into a table with dependent variables as columns and independent variables as rows, or a transpose of this table.

Usage

```
group_by_dvs(
  object,
  ...,
  col_name = "est",
  add_prefix = TRUE,
  group_first = FALSE,
  use_standardizedSolution = FALSE
)

group_by_ivs(
  object,
  ...,
  col_name = "est",
  add_prefix = TRUE,
  group_first = FALSE,
  use_standardizedSolution = FALSE
)
```

Arguments

object	A 'lavaan'-class object or the output of lavaan::parameterEstimates() or lavaan::standardizedSolution().
...	Optional arguments to be passed to lavaan::parameterEstimates(). Ignored if object is an output of lavaan::parameterEstimates() or lavaan::standardizedSolution().
col_name	The column name of information to be grouped. Default is "est". It accepts only one name.
add_prefix	If TRUE, the default, col_name will be added as prefix to the column names of the output.
group_first	If TRUE, the rows will be grouped by groups first and then by independent variables Ignored if the model has only one group. Default is FALSE.
use_standardizedSolution	If TRUE and object is not an estimates table, then lavaan::standardizedSolution() will be used to generate the table. If FALSE, the default, then lavaan::parameterEstimates() will be used if necessary.

Details

It gets a 'lavaan'-class object or the output of `lavaan::parameterEstimates()` or `lavaan::standardizedSolution()` and group selected columns by "dependent" variables `group_by_dvs()` or by "independent" variables `group_by_ivs()`.

"Dependent" variables are defined as variables on the left hand side of the operator ~.

"Independent" variables are defined as variables on the right hand side of the operator ~.

Note that a variable can both be a "dependent" variable and an "independent" variable in a model.

Value

A data-frame-like object of the class `est_table`.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
y ~ b*m
ab := a*b
'

fit <- sem(model, data = dat, fixed.x = FALSE)
parameterEstimates(fit)

# Group by DVs
group_by_dvs(fit)

# Group by IVs
group_by_ivs(fit)
```

---

plot_boot *Diagnostic Plots of Bootstrap Estimates in 'lavaan'*

Description

Plots for examining the distribution of bootstrap estimates in a model fitted by `lavaan`.

Usage

```
plot_boot(
  object,
  param,
  standardized = NULL,
  nclass = NULL,
  hist_color = "lightgrey",
  hist_linewidth = 1,
  density_line_type = "solid",
  density_line_color = "blue",
  density_line_linewidth = 2,
  est_line_type = "dotted",
  est_line_color = "red",
  est_line_linewidth = 2,
  qq_dot_size = 2,
  qq_dot_color = "black",
  qq_dot_pch = 16,
  qq_line_linewidth = 2,
  qq_line_color = "black",
  qq_line_linetype = "solid"
)
```

Arguments

object	Either a lavaan::lavaan object with bootstrap estimates stored, or the output of standardizedSolution_boot_ci(). For standardized solution and user-defined parameters, if the object is a lavaan::lavaan object, the estimates need to be stored by store_boot_est_std() or store_boot_def().
param	String. The name of the parameter to be plotted, which should be the name as appeared in a call to coef().
standardized	Logical. Whether the estimates from the standardized solution are to be plotted. Default is NULL. If object is a lavaan::lavaan object, then this is a required parameter and users need to explicitly set it to TRUE or FALSE. If object is the output of standardizedSolution_boot_ci(), then this argument is ignored (forced to be TRUE internally).
nclass	The number of breaks. This argument will be passed to hist(). Default is NULL.
hist_color	String. The color of the bars in the histogram. It will be passed to hist() for the argument col. Default is "lightgrey".
hist_linewidth	The width of the borders of the bars in the histogram. Default is 1.
density_line_type	String. The type of the line of the density curve in the histogram. It will be passed to lines() for the argument lty. Default is "solid".
density_line_color	String. The color of the density curve in the histogram. It will be passed to lines() for the argument col. Default is "blue".

density_line_linewidth
: The width of the density curve in the histogram. It will be passed to lines() for the argument lwd. Default is 2.

est_line_type
: String. The type of the vertical line in the histogram showing the point estimate of the parameter. It will be passed to abline() for the argument lty. Default is "dotted",

est_line_color
: String. The color of the vertical line showing the point estimate in the histogram. It will be passed to abline() for the argument col. Default is "red".

est_line_linewidth
: The width of the vertical line showing the point estimate in the histogram. It will be passed to hist() for the argument lwd. Default is 2.

qq_dot_size
: The size of the points in the normal QQ-plot. It will be passed to qqnorm() for the argument cex. Default is 2.

qq_dot_color
: String. The color of the points in the normal QQ-plot. It will be passed to qqnorm() for the argument col. Default is "black".

qq_dot_pch
: Numeric. The shape of the points in the normal QQ-plot. It will be passed to qqnorm() for the argument pch. Default is 16.

qq_line_linewidth
: The width of the diagonal line to be drawn in the normal QQ-plot. It will be passed to qqline() for the argument lwd. Default is 2.

qq_line_color
: String. The color of the diagonal line to be drawn in the normal QQ-plot. It will be passed to qqline() for the argument col. Default is "black".

qq_line_linetype
: The type of the diagonal line to be drawn in the normal QQ-plot. Default is "solid".

Details

Rousselet, Pernet, and Wilcox (2021) argued that when using bootstrapping, it is necessary to examine the distribution of bootstrap estimates. This can be done when boot::boot() is used because it has a plot method for its output. This cannot be easily done in model fitted by lavaan::lavaan().

The function plot_boot() is used for plotting the distribution of bootstrap estimates for a model fitted by lavaan in a format similar to that of the output of boot::boot(), with a histogram on the left and a normal QQ-plot on the right.

For free parameters in a model (unstandardized), it can be called directly on the output of lavaan and retrieves the stored estimates.

For estimates of user-defined parameters, call store_boot_def() first to compute and store the bootstrap estimates first.

For estimates in standardized solution, for both free and user-defined parameters, call store_boot_est_std() first to compute and store the bootstrap estimates in the standardized solution.

Since Version 0.1.11.2, it can also plot bootstrap estimates in the output of standardizedSolution_boot_ci().

Value

Return the original lavaan::lavaan object invisibly. Called for its side-effect (plotting the graphs).

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

References

Rousselet, G. A., Pernet, C. R., & Wilcox, R. R. (2021). The percentile bootstrap: A primer with step-by-step instructions in R. *Advances in Methods and Practices in Psychological Science*, *4*(1), 1–10. doi:10.1177/2515245920911881

See Also

`lavaan::fitMeasures()`, `store_boot_est_std()`, and `store_boot_def()`.

Examples

```
library(lavaan)

data(simple_mediation)
mod <-
"
m ~ a * x
y ~ b * m + x
ab := a * b
"
fit <- sem(mod, simple_mediation,
           se = "bootstrap",
           bootstrap = 50,
           iseed = 985714)

# Can plot bootstrap estimates for
# free parameters directly
# Note that 'standardized' must be always be set to
# either TRUE or FALSE. No default value.
plot_boot(fit, "a", standardized = FALSE)

# For estimates of user-defined parameters,
# call store_boot_def() first.
fit <- store_boot_def(fit)
plot_boot(fit, "ab", standardized = FALSE)

# For estimates in standardized solution,
# call store_boot_est_std() first.
fit <- store_boot_est_std(fit)
plot_boot(fit, "a", standardized = TRUE)
plot_boot(fit, "ab", standardized = TRUE)

# It can also plot the estimates stored
# in the output of standardizedSolution_boot_ci().
std_boot <- standardizedSolution_boot_ci(fit)
plot_boot(std_boot, "ab")
plot_boot(fit, "ab", standardized = TRUE)
```

plot_models_fm *Plot Models on a Chi-Squares-vs-Dfs Graph*

Description

Plot models on a graph with model chi-square against model the degrees of freedom, with lines for equal fit measures.

Usage

```
plot_models_fm(
  ...,
  fit_measure = c("cfi", "tli", "rmsea"),
  fit_values,
  line_size = 1,
  label_size = 8,
  point_size = 5,
  position_dodge = 0.5,
  include_model_values = FALSE,
  include_baseline = FALSE
)
```

Arguments

...	The lavaan::lavaan objects to be plotted. Can also be a named list of the lavaan::lavaan objects. If it is as list, it must be named and the names will be used in the plot.
fit_measure	A length-one character vector of the fit measures to use to plot the lines. Only supports "cfi" (the default), "tli", and "rmsea".
fit_values	A numeric vector of the values of the fit measure used to plot the lines. The default values are c(.90, .95) for "cfi" and "tli", and c(.00, .02, .05, .08) for "rmsea".
line_size	The size of the lines. Default is 1.
label_size	The size of the model names. Default is 8.
point_size	The size of the point representing a model. Default is 2.
position_dodge	Offsetting the label of a model from the point. Default is .5. Used by ggrepel::geom_label_repel().
include_model_values	If TRUE , the values of the models on fit_measure will be added to fit_values. Default is FALSE.
include_baseline	If TRUE, the baseline model is included in the plot. Default is FALSE.

Details

This function plots models based on their model chi-squares and model degrees of freedoms.It can also add lines for chi-square-df combination with equal values on selected fit measures. Currently supports CFI, TLI, and RMSEA.

Value

Return a `ggplot2::ggplot()` output that can be further modified.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

See Also

`lavaan::fitMeasures()`

Examples

```
library(lavaan)

# From the help page of modificationIndices

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

fit <- cfa(HS.model, data = HolzingerSwineford1939)
modindices(fit, sort = TRUE, op = "=~")

fit2 <- update(fit, add = "visual =~ x9")
fit3 <- update(fit, add = "textual =~ x3\nvisual =~ x7")

models <- list(Initial = fit,
               Model_2 = fit2,
               Model_3 = fit3)
fit_cfi <- sapply(models, fitMeasures, fit.measures = "cfi")
fit_tli <- sapply(models, fitMeasures, fit.measures = "tli")
fit_rmsea <- sapply(models, fitMeasures, fit.measures = "rmsea")

# Supply the models as arguments
plot_models_fm(fit, fit2, fit3)

# Plot lines for selected values on a fit measure (CFI by default)
plot_models_fm(fit, fit2, fit3, fit_values = c(.90, .925, .95, fit_cfi))

# Plot the models' values on the fit measures
plot_models_fm(fit, fit2, fit3, include_model_values = TRUE)
```

```
# Supply the models as a named list
plot_models_fm(list(A = fit, B = fit2, C = fit3),
               fit_values = c(.90, .925, .95))

# Plot the models, fit measure set to TLI
plot_models_fm(fit, fit2, fit3, fit_measure = "tli")
plot_models_fm(fit, fit2, fit3, fit_measure = "tli",
               fit_values = c(.90, .925, .95, fit_tli))
plot_models_fm(fit, fit2, fit3, fit_measure = "tli",
               include_model_values = TRUE)

# Plot the models, fit measure set to RMSEA
plot_models_fm(fit, fit2, fit3, fit_measure = "rmsea")
plot_models_fm(fit, fit2, fit3, fit_measure = "rmsea",
               include_model_values = TRUE)
```

`print.est_table` *Print an 'est_table' Object*

Description

Print method for an 'est_table' object

Usage

```
## S3 method for class 'est_table'
print(x, ..., nd = 3, empty_cells = "--", group_first = FALSE)
```

Arguments

`x`	Object of the class `est_table`.
`...`	Optional arguments to be passed to `print()` methods.
`nd`	The number of digits to be printed. Default is 3. (Scientific notation will never be used.)
`empty_cells`	String to be printed for empty cells or cells with no values. Default is `"--"`.
`group_first`	Not used.

Value

`x` is returned invisibly. Called for its side effect.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

`print.fit_by_models` *Print a 'fit_by_models' Object*

Description

Print method for a 'fit_by_models' object

Usage

```
## S3 method for class 'fit_by_models'
print(
  x,
  ...,
  nd = 3,
  type = c("compact"),
  remove_all_na = TRUE,
 measures_compact = c("npar", "chisq", "chisq.scaled", "df", "df.scaled", "pvalue",
  "pvalue.scaled", "chisq.scaling.factor", "cfi", "cfi.robust", "tli", "tli.robust",
  "aic", "bic", "bic2", "rmsea", "rmsea.ci.level", "rmsea.ci.lower", "rmsea.ci.upper",
    "rmsea.close.h0", "rmsea.pvalue", "rmsea.robust", "rmsea.ci.lower.robust",
    "rmsea.ci.upper.robust", "rmsea.pvalue.robust", "srmr", "srmr_nomean")
)
```

Arguments

`x`	Object of the class `fit_by_models`.
`...`	Optional arguments to be passed to `print()` methods.
`nd`	The number of digits to be printed. Default is 3. (Scientific notation will never be used.)
`type`	String. The type of the output. Currently only supports one type, `"compact"`.
`remove_all_na`	Logical. Whether rows with `NA` in all columns will be removed. Default is `TRUE`.
`measures_compact`	If output `type` is `"compact"`, the character vector of fit measures to be printed. The names should be the names of the output of `lavaan::fitMeasures()`, in vector form.

Details

This function is intended to print the fit measures of one or more groups in a simple and compact table for quick preview. For a well-organized layout, call `lavaan::fitMeasures()` and set `output` to `"text"`. For comparing the models with notations on models with the best fit on each measures, use `semTools::compareFit()`.

Value

`x` is returned invisibly. Called for its side effect.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

See Also

fitMeasures_by_models()

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model1 <-
'
m ~ a*x
y ~ b*m
ab := a*b
'
fit1 <- sem(model1, data = dat, fixed.x = FALSE)
model2 <-
'
m ~ a*x
y ~ b*m + x
ab := a*b
'
fit2 <- sem(model2, data = dat, fixed.x = FALSE)

out <- fitMeasures_by_models(list(no_direct = fit1,
                                  direct = fit2))
out

print(out, nd = 4, measures_compact = c("chisq", "cfi", "rmsea"))
```

---

print.std_solution_boot

Print an 'std_solution_boot' Object

Description

Print method for an 'std_solution_boot' object, which is the output of standardizedSolution_boot_ci().

Usage

```
## S3 method for class 'std_solution_boot'
print(x, ..., nd = 3, output = c("table", "text"), standardized_only = TRUE)
```

Arguments

x	Object of the class std_solution_boot.
...	Optional arguments to be passed to print() methods.
nd	The number of digits after the decimal place. Default is 3.
output	String. How the results are printed. Default is "table" and the results are printed in a table format similar to that of lavaan::standardizedSolution(). If "text", the results will be printed in a text format similar to the printout of the output of summary() of a 'lavaan'-class object.
standardized_only	Logical. If TRUE, the default, only the results for the standardized solution will be printed. If FALSE, then the standardized solution is printed alongside the unstandardized solution, as in the printout of the output of summary() of a 'lavaan'-class object.

Details

The default format of the printout is that of lavaan::standardizedSolution(), which is compact but not easy to read. Users can request a format similar to that of the printout of the summary of a lavaan output by setting output to "text".

For the "text" format, users can also select whether only the standardized solution is printed (the default) or whether the standardized solution is appended to the right of the printout.

Value

x is returned invisibly. Called for its side effect.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

See Also

standardizedSolution_boot_ci()

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
y ~ b*m
ab := a*b
'
```

```
# Should set bootstrap to at least 2000 in real studies
fit <- sem(model, data = dat, fixed.x = FALSE,
           se = "boot",
           bootstrap = 50)
std_out <- standardizedSolution_boot_ci(fit)
std_out
print(std_out, output = "text")
print(std_out, output = "text", standardized_only = FALSE)
```

`ptable_to_syntax` *Convert a 'lavaan' Parameter Table to a 'lavaan' Model Syntax*

Description

It tries to generate a 'lavaan' model syntax from a `lavaan` parameter table.

Usage

```
ptable_to_syntax(object, allow_incomplete = FALSE)

compare_ptables(object1, object2)
```

Arguments

`object`	If set to a `lavaan` object, such as the output of `lavaan::sem()` or `lavaan::cfa()`, the parameter table will be extracted from it by `lavaan::parameterTable()`. If set to a parameter table, it will be used to generate the model syntax. It can also the output of `lavaan::lavParseModelString()` with `as.data.frame. = TRUE`, if `allow_incomplete` is set to `TRUE`. Note that `allow_incomplete` is set to `FALSE` by default because `lavaan::lavParseModelString()` only parses the model syntax and there is no guarantee that the model defined is valid.
`allow_incomplete`	Whether incomplete parameter table formed by `lavaan::lavParseModelString()` with `as.data.frame. = TRUE` is allowed. Default if `FALSE`.
`object1`	The first `lavaan` parameter table, to be compared with `object2`. If it is set to a `lavaan` object (e.g., the output of `lavaan::sem()` or `lavaan::cfa()`), then the parameter table will be extracted from it.
`object2`	The second `lavaan` parameter table, to be compared with `object1`. If it is set to a `lavaan` object (e.g., the output of `lavaan::sem()` or `lavaan::cfa()`), then the parameter table will be extracted from it.

Details

This function tries to convert a lavaan parameter table to a text representation of the lavaan model specified in model syntax.

When users call lavaan::sem(), in addition to the model syntax, other arguments not stored in the syntax are also used to produce the final model (e.g., meanstructure, fixed.x, and std.lv). To produce exactly the same model, these arguments are also needed to be specified, which is difficult to generate using only the parameter table.

Therefore, the model syntax produced will state all aspects of a model explicitly, even for those aspects that usually can be omitted due to the default values of these arguments. This approach requires users to call lavaan::lavaan() directly, instead of its wrappers (e.g, lavaan::sem()), to produce the same parameter table.

The model syntax produced this way is more difficult to read. However, it ensures that original model can be reproduced, without the need to know the arguments to set.

Due to the nearly unlimited possibilities in the form of a model, it is recommended to compare the model generated by the model syntax with the original parameter table using compare_ptables(). It only compares the forms of the two models, including user starting values, if any. It does not compare parameter estimates and standard errors.

Raw Specification From lavaan::lavParseModelString():

There may be cases in which the parameter table is the "incomplete" table generated by lavaan::lavParseModelString() with as.data.frame. = TRUE. This table is "incomplete" because it is formed merely by parsing the model syntax. There is no guarantee that the model is valid.

The function ptable_to_syntax() has basic support for this kind of tables but it is disabled by default. To process an incomplete parameter table formed by lavaan::lavParseModelString(), set allow_incomplete to TRUE.

Limitations:

The function ptable_to_syntax() does not yet support the following models:

- Multiple-group models.
- Multilevel models.
- Models with categorical variables.
- Models with user-specified lower or upper bounds.
- Models with the operator <~.
- Models with constraints imposed by equal().
- Models with labels having spaces.
- Models with labels having syntax operators (e.g., ~, =~, etc.).

Value

ptable_to_syntax() returns a length-one character vector that stores the generated lavaan model syntax.

compare_ptables() returns a length-one logical vector. TRUE if the two models are identical in form. FALSE if they are not identical.

Functions

- `ptable_to_syntax()`: Convert a lavaan parameter a lavaan model syntax.
- `compare_ptables()`: Compare two lavaan parameter tables.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448. This function is inspired by a discussion at the Google Group https://groups.google.com/g/lavaan/c/1ueFiue9qLM/m/cJhxDoqeBAAJ.

See Also

`lavaan::lavaan()`, `lavaan::parameterTable()`

Examples

```
library(lavaan)

mod <-
"
visual  =~ x3 + x1 + x2
textual =~ x4 + x6 + x5
speed   =~ x7 + x8 + x9 + start(0.1) * x6
visual ~ a*textual
speed ~ b*visual
ab := a * b
"

fit <- sem(mod, data = HolzingerSwineford1939)

mod_chk <- ptable_to_syntax(fit)
cat(mod_chk, sep = "\n")
# Need to call lavaan() directly
fit_chk <- lavaan(mod_chk, data = HolzingerSwineford1939)
fit_chk
fit
# Compare the parameter table:
(ptable1 <- parameterTable(fit))
(ptable2 <- parameterTable(fit_chk))
compare_ptables(ptable1, ptable2)
```

---

record_history *Record the Minimization History*

Description

Record the minimization history when a model is fitted by `lavaan::lavaan()` or its wrappers (e.g., `lavaan::sem()` or `lavaan::cfa()`).

Usage

```
record_history(object)

## S3 method for class 'fit_history'
plot(x, params, last_n = -1, orientation = c("horizontal", "vertical"), ...)

## S3 method for class 'fit_history'
print(x, n_iterations = 10, digits = 3, ...)
```

Arguments

object	A 'lavaan'-class object.
x	A fit_history class object, the output of record_history().
params	A character vector of the names of parameters to be plotted. Must be the names of one or more columns in x.
last_n	The lass n iterations to be plotted. Default is -1, plotting all iterations.
orientation	The orientation of the plot. Either "horizontal" (the default) or "vertical".
...	Optional arguments. To be passed to the print method of data frame.
n_iterations	The number of iterations to print. Default is 10, printing the first 10 iterations (or all iterations, if the number of iterations is less than 10).
digits	The number of digits to be displayed. Default is 3.

Details

It records the minimization history when a model is fitted by lavaan::lavaan() or its wrappers (e.g., lavaan::sem() or lavaan::cfa()). The recorded history can then be plotted or displayed, for visualizing how the estimates of free parameters is found.

It will refit the model by the update method of lavaan::lavaan, setting se = "none" and test = "standard" because they have no impact on the minimization process.

This and related functions are adapted from the package semunpack. The version in this package will be revised to be an advanced version intended for diagnostic purpose in real studies.

Value

A fit_history-class object with a plot method (plot.fit_history()).

Functions

- plot(fit_history): The plot method for the output of record_history().
- print(fit_history): The print method for the output of record_history().

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
# Adapted from the example for CFA in lavaan::cfa().
# Using only two of the factors
library(lavaan)
HS.model <-
'
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
'
fit <- cfa(HS.model, data = HolzingerSwineford1939)

# Refit the model with the history recorded
fit_h <- record_history(fit)
fit_h

# Plot the history for selected parameters
plot(fit_h, params = c("visual=~x2", "visual=~x3",
                       "visual~~textual"),
            last_n = 10)
plot(fit_h, params = c("visual=~x2", "visual=~x3",
                       "visual~~textual"),
            last_n = 10,
            orientation = "vertical")
```

`show_cfi` *Visualize How CFI and TLI Are Computed*

Description

Show how CFI and TLI are computed using a graph of model chi-square vs. model degrees of freedom.

Usage

```
show_cfi(fit, ...)

show_tli(fit, ...)

show_ifi(fit, fit_measures = c("cfi", "tli"), test = c("standard"))
```

Arguments

`fit`	An output of `lavaan::lavaan()` or its wrappers (e.g., `lavaan::cfa()` and `lavaan::sem()`).
`...`	Arguments to be passed to `show_ifi()`.
`fit_measures`	The fit measures to be plotted. Acceptable values are `"cfi"` and `"tli"`.
`test`	The type of model chi-square test. It corresponds to the `test` argument of `lavaan::lavaan()` or its wrappers. Only `"standard"` is supported for now.

Details

This function receives an output of `lavaan::lavaan()` or its wrappers (e.g., `lavaan::cfa()` and `lavaan::sem()`) and illustrates how CFI is computed.

Value

An output of `ggplot2::ggplot()` that can be further modified.

Functions

- `show_cfi()`: A wrapper of `show_ifi()` with `fit_measures = "cfi"`.
- `show_tli()`: A wrapper of `show_ifi()` with `fit_measures = "tli"`.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)

# From the help page of lavaan::cfa().

HS.model <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
'
fit <- cfa(HS.model, data = HolzingerSwineford1939)

# By default show how CFI is computed
show_ifi(fit)

# Show how TLI is computed
show_ifi(fit, fit_measures = "tli")

# Wrappers
show_cfi(fit)
show_tli(fit)
```

`show_more_options` *Show More Major Options in an Output of 'lavaan'*

Description

Display the values of more major options in a model fitted by `lavaan::lavaan()` or its wrappers (e.g., lavaan::sem or `lavaan::cfa()`).

Usage

```
show_more_options(fit)

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

Arguments

fit	An output of lavaan::lavaan() or its wrappers (e.g., lavaan::cfa() and lavaan::sem())
x	The output of show_more_options().
...	Additional arguments. Ignored.

Details

It extracts the values of major options in the output of lavaan::lavaan() or its wrappers (e.g., lavaan::sem or lavaan::cfa(). Most of the values are also reported in the summary of a 'lavaan'-class object. This function is used to show the values in one single table for a quick overview.

It checks the actual values, not the call used. This is useful for understanding how a prepackaged estimator such as ML, MLM, and MLR set other options. It supports the following options:

- Estimator (estimator)
- Standard error (se)
- Model chi-square test(s) (test)
- Missing data method (missing)
- Information matrix used for computing standard errors (information)
- Information matrix used for computing model chi-square (information)
- Whether the mean structure is included.

It is named show_more_options() to differentiate it from show_options(), originally in the semunpack package, which is intended for new users of lavaan``. The code is adapted from show_options‘ with more advanced options added.

Value

A show_more_options-class object with a print method that formats the output.

Methods (by generic)

- print(show_more_options): The print method of the output of show_more_options().

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)

# From the help page of lavaan::cfa().

HS.model <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
'
fit <- cfa(HS.model, data = HolzingerSwineford1939)

tmp <- show_more_options(fit)
tmp

fit <- cfa(HS.model, data = HolzingerSwineford1939, estimator = "MLR")
show_more_options(fit)
fit <- cfa(HS.model, data = HolzingerSwineford1939, estimator = "MLM")
show_more_options(fit)
```

---

show_options *Show Major Options in an Output of 'lavaan'*

Description

Display the values of major options in a model fitted by `lavaan::lavaan()` or its wrappers (e.g., `lavaan::sem()` or `lavaan::cfa()`).

Usage

```
show_options(fit)

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

Arguments

`fit`	An output of `lavaan::lavaan()` or its wrappers (e.g., `lavaan::cfa()` and `lavaan::sem()`)
`x`	The output of `show_options()`.
`...`	Additional arguments. Ignored.

Details

It extracts the values of major options in the output of `lavaan::lavaan()` or its wrappers (e.g., `lavaan::sem()` or `lavaan::cfa()`.

It checks the actual values, not the call used. This is useful for understanding how a prepackaged estimator such as `ML`, `MLM`, and `MLR` set other options. It supports the following options:

- Estimator (`estimator`)
- Standard error (`se`)
- Model chi-square test(s) (`test`)
- Missing data method (`missing`)
- Information matrix used for computing standard errors (`information`)
- Information matrix used for computing model chi-square (`information`)
- Whether the mean structure is included.

Value

A `show_options`-class object with a print method that formats the output.

Methods (by generic)

- `print(show_options)`: The print method of the output of `show_options()`.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)

# From the help page of lavaan::cfa().

HS.model <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
'
fit <- cfa(HS.model, data = HolzingerSwineford1939)

tmp <- show_options(fit)
tmp

fit <- cfa(HS.model, data = HolzingerSwineford1939, estimator = "MLR")
show_options(fit)
fit <- cfa(HS.model, data = HolzingerSwineford1939, estimator = "MLM")
show_options(fit)
```

`simple_mediation` *Sample Dataset: Simple Mediation*

Description

A simple mediation model.

Usage

```
simple_mediation
```

Format

A data frame with 100 rows and 5 variables:

x Predictor. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

city Group variable: "City A" or "City B". String.

Examples

```
library(lavaan)
data(simple_mediation)
mod <-
"
m ~ a * x
y ~ b * m + x
ab := a * b
"
fit <- sem(mod, simple_mediation, fixed.x = FALSE)
parameterEstimates(fit)
mod_gp <-
"
m ~ c(a1, a2) * x
y ~ c(b1, b2) * m + x
a1b1 := a1 * b1
a2b2 := a2 * b2
ab_diff := a1b1 - a2b2
"
fit_gp <- sem(mod_gp, simple_mediation, fixed.x = FALSE, group = "city")
parameterEstimates(fit_gp)
```

sort_by *Sort a Parameter Estimates Table*

Description

Sort a parameter estimates table or a similar table in lavaan by common fields such as op (operator) and lhs (left- hand side).

Usage

```
sort_by(
  object,
  by = c("op", "lhs", "rhs"),
  op_priority = c("=~", "~", "~~", ":=", "~1", "|", "~*~"),
  number_rows = TRUE
)
```

Arguments

object	The output of lavaan::parameterEstimates(), lavaan::standardizedSolution(), or a lavaan.data.frame object. May also work on an est_table-class object returned by functions like group_by_dvs() but there is no guarantee.
by	A character vector of the columns for filtering. Default is c("op", "lhs", "rhs").
op_priority	How rows are sorted by op. Default is c("=~", "~", "~~", ":=", "~1", "|", "~*~"). Can set only a few of the operators, e.g., c("~", "~~"). Other operators will be placed to the end with orders not changed.
number_rows	Whether the row names will be set to row numbers after sorting *if* the row names of object is equal to row numbers. Default is TRUE.

Details

This functions accepts the output of lavaan::parameterEstimates() and lavaan::standardizedSolution() and filter the rows by commonly used field.

Value

The sorted version of the input object.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model1 <-
'
m ~ a*x
y ~ b*m
ab := a*b
'
fit1 <- sem(model1, data = dat, fixed.x = FALSE)
model2 <-
'
m ~ a*x
y ~ b*m + x
ab := a*b
'
fit2 <- sem(model2, data = dat, fixed.x = FALSE)
parameterEstimates(fit1)
parameterEstimates(fit2)
out <- group_by_models(list(no_direct = fit1,
                            direct = fit2),
                       col_names = c("est", "pvalue"))
out
sort_by(out)
sort_by(out, op_priority = c("~", ":="))
sort_by(out, by = c("op", "rhs"))
```

---

standardizedSolution_boot_ci

Bootstrap CIs for Standardized Solution

Description

Functions for forming bootstrap confidence intervals for the standardized solution.

Usage

```
standardizedSolution_boot_ci(
  object,
  level = 0.95,
  type = "std.all",
  save_boot_est_std = TRUE,
```

```
  force_run = FALSE,
  boot_delta_ratio = FALSE,
  boot_ci_type = c("perc", "bc", "bca.simple"),
  ...
)

store_boot_est_std(object, type = "std.all", force_run = FALSE, ...)

get_boot_est_std(object)
```

Arguments

`object`	A 'lavaan'-class object, fitted with 'se = "boot"'.
`level`	The level of confidence of the confidence intervals. Default is .95.
`type`	The type of standard estimates. The same argument of `lavaan::standardizedSolution()`, and support all values supported by `lavaan::standardizedSolution()`. Default is `"std.all"`.
`save_boot_est_std`	Whether the bootstrap estimates of the standardized solution are saved. If saved, they will be stored in the attribute `boot_est_std`. Default is `TRUE`.
`force_run`	If `TRUE`, will skip checks and run models without checking the estimates. For internal use. Default is `FALSE`.
`boot_delta_ratio`	The ratio of (a) the distance of the bootstrap confidence limit from the point estimate to (b) the distance of the delta-method limit from the point estimate. Default is `FALSE`.
`boot_ci_type`	The type of the bootstrapping confidence intervals. Support percentile confidence intervals (`"perc"`, the default) and bias-corrected confidence intervals (`"bc"` or `"bca.simple"`).
`...`	Other arguments to be passed to `lavaan::standardizedSolution()`.

Details

`standardizedSolution_boot_ci()` receives a lavaan::lavaan object fitted with bootstrapping standard errors requested and forms the confidence intervals for the standardized solution.

It works by calling `lavaan::standardizedSolution()` with the bootstrap estimates of free parameters in each bootstrap sample to compute the standardized estimates in each sample.

A more reliable way is to use function like `lavaan::bootstrapLavaan()`. Nevertheless, this simple function is good enough for some simple scenarios, and does not require repeating the bootstrapping step.

`store_boot_est_std()` computes the standardized solution for each bootstrap sample, stores them the lavaan::lavaan object, and returns it. These estimates can be used by other functions, such as `plot_boot()`, to examine the estimates, without the need to repeat the computation.

`get_boot_est_std()` retrieves the bootstrap estimates of the standardized solution stored by `store_boot_est_std()`.

Value

The output of lavaan::standardizedSolution(), with bootstrap confidence intervals appended to the right, with class set to std_solution_boot (since version 0.1.8.4). It has a print method (print.std_solution_boot()) that can be used to print the standardized solution in a format similar to that of the printout of the summary() of a lavaan::lavaan object.

store_boot_est_std() returns the fit object set to object, with the bootstrap values of standardized solution in the bootstrap samples, as a matrix, stored in the slot external under the name shh_boot_est_std.

get_boot_est_std() returns a matrix of the stored bootstrap estimates of standardized solution. If none is stored, NULL is returned.

store_boot_est_std() is usually used with diagnostic functions such as plot_boot().

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448. Originally proposed in an issue at GitHub https://github.com/simsem/semTools/issues/101#issue-1021974657, inspired by a discussion at the Google group for lavaan https://groups.google.com/g/lavaan/c/qQBXSz5cd0o/m/R8YT5HxNAgAJ. boot::boot.ci() is used to form the percentile confidence intervals in this version.

See Also

lavaan::standardizedSolution(), plot_boot()

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
y ~ b*m
ab := a*b
'

# Should set bootstrap to at least 2000 in real studies
fit <- sem(model, data = dat, fixed.x = FALSE,
           se = "boot",
           bootstrap = 100)
summary(fit)

std <- standardizedSolution_boot_ci(fit)
std

# Print in a friendly format with only standardized solution
```

```
print(std, output = "text")

# Print in a friendly format with both unstandardized
# and standardized solution
print(std, output = "text", standardized_only = FALSE)

# plot_boot() can be used to examine the bootstrap estimates
# of a parameter
plot_boot(std, param = "ab")


# store_boot_est_std() is usually used with plot_boot()
# First, store the bootstrap estimates of the
# standardized solution
fit_with_boot_std <- store_boot_est_std(fit)
# Second, plot the distribution of the bootstrap estimates of
# standardized 'ab'
plot_boot(fit_with_boot_std, "ab", standardized = TRUE)
```

`store_boot_def` *Store Bootstrap Estimates of User-Defined Parameters*

Description

It receives a lavaan::lavaan object fitted with bootstrapping standard errors requested, computes the user-defined parameters in each bootstrap samples, and returns a lavaan::lavaan object with the estimates stored.

Usage

```
store_boot_def(object, force_run = FALSE)

get_boot_def(object)
```

Arguments

`object`	A 'lavaan'-class object, fitted with 'se = "boot"'.
`force_run`	If `TRUE`, will skip checks and run models without checking the estimates. For internal use. Default is `FALSE`.

Details

`lavaan::lavaan()` and its wrappers, such as `lavaan::sem()` and `lavaan::cfa()`, stores the estimates of free parameters in each bootstrap sample if bootstrapping is requested. However, if a model has user-defined parameters, their values in each bootstrap sample are not stored. `store_boot_def()` computes the retrieves the stored bootstrap estimates and computes the values of user-defined parameters. The values are then stored in the slot `external` of the object, in the element `shh_boot_def`. The bootstrap estimates can then be used by other functions for diagnostics purposes.

`get_boot_def()` extracts the bootstrap estimates of user-defined parameters from a 'lavaan'-class object. If none is stored, `NULL` is returned.

`store_boot_def()` is usually used with diagnostic functions such as `plot_boot()`.

Value

`store_boot_def()` returns the fit object set to `object`, with the bootstrap values of user-defined parameters in the bootstrap samples, as a matrix, stored in the slot `external` of `object` under the name `shh_boot_def`.

`get_boot_def()` returns a matrix of the stored bootstrap estimates of user-defined parameters

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448.

See Also

`plot_boot()`

Examples

```
library(lavaan)
set.seed(5478374)
n <- 50
x <- runif(n) - .5
m <- .40 * x + rnorm(n, 0, sqrt(1 - .40))
y <- .30 * m + rnorm(n, 0, sqrt(1 - .30))
dat <- data.frame(x = x, y = y, m = m)
model <-
'
m ~ a*x
y ~ b*m
ab := a*b
'

# Should set bootstrap to at least 2000 in real studies
fit <- sem(model, data = dat, fixed.x = FALSE,
           se = "boot",
           bootstrap = 100)
summary(fit)

# store_boot_def() is usually used with plot_boot()
# First, store the bootstrap estimates of user-defined
# parameters
fit_with_boot_def <- store_boot_def(fit)
# Second, plot the distribution of the bootstrap estimates of
# 'ab'
plot_boot(fit_with_boot_def, "ab", standardized = FALSE)
```

vec_rsquare *Wrapper Functions to Extract Information as a Vector*

Description

A set of wrapper functions to extract information from a lavaan-class object and return a named vector.

Usage

```
vec_rsquare(object)

vec_sample_vcov(object)

vec_sample_var(object)

vec_est_var(object)

vec_est_se(object)

vec_def_var(object)

vec_def_se(object)

vec_lavTestLRT(
  object,
  ...,
  method = "default",
  A.method = "delta",
  scaled.shifted = TRUE,
  H1 = TRUE,
  model.names = NULL
)

vec_lavTestScore(
  object,
  add = NULL,
  release = NULL,
  univariate = TRUE,
  information = "expected"
)

vec_lavTestWald(object, constraints = NULL, prefix = NULL)

vec_compRelSEM(object, ...)
```

Arguments

object	A lavaan-class object.
...	Additional arguments to be passed to the original function.
method	An argument to be passed to lavaan::lavTestLRT(). Please refer to the help page of lavaan::lavTestLRT().
A.method	An argument to be passed lavaan::lavTestLRT(). Please refer to the help page of lavaan::lavTestLRT().
scaled.shifted	An argument to be passed to lavaan::lavTestLRT(). Please refer to the help page of lavaan::lavTestLRT().
H1	An argument to be passed to lavaan::lavTestLRT(). Please refer to the help page of lavaan::lavTestLRT().
model.names	An argument to be passed to lavaan::lavTestLRT(). Please refer to the help page of lavaan::lavTestLRT(). Unlike lavaan::lavTestLRT(), this argument is required, for the sake of naming the vector to be returned.
add	An argument to be passed to lavaan::lavTestScore(). Please refer to the help page of lavaan::lavTestScore().
release	An argument to be passed to lavaan::lavTestScore(). Please refer to the help page of lavaan::lavTestScore().
univariate	An argument to be passed to lavaan::lavTestScore(). Please refer to the help page of lavaan::lavTestScore().
information	An argument to be passed to lavaan::lavTestScore(). Please refer to the help page of lavaan::lavTestScore().
constraints	An argument to be passed to lavaan::lavTestWald(). Please refer to the help page of lavaan::lavTestWald().
prefix	Optional. A character string to be added as a prefix to names of the output. Default is NULL.

Details

This set of wrapper functions are for functions like lavaan::bootstrapLavaan() that require users to supply a function that receives a lavaan-class object and returns a vector of values.

All wrappers functions are designed to have the same form of output: a named numeric vector.

Many of the tasks of this set of wrappers can be performed by writing our own functions. The wrapper functions are developed just to save the coding time for some commonly requested information.

The wrapper functions are designed to be as simple to use as possible, with as few arguments as possible. If advanced control is needed, users are recommended to write their own wrappers.

Value

All of them return a named numeric vector.

Functions

- `vec_rsquare()`: Get R-squares in a model.
- `vec_sample_vcov()`: Get sample variances and covariances.
- `vec_sample_var()`: Get sample variances.
- `vec_est_var()`: Sampling variances of free parameters.
- `vec_est_se()`: Standard errors of free parameters.
- `vec_def_var()`: Sampling variances of user-defined parameters.
- `vec_def_se()`: Standard errors of user-defined parameters.
- `vec_lavTestLRT()`: Get sample variances.
- `vec_lavTestScore()`: Do score tests.
- `vec_lavTestWald()`: Do a Wald test.
- `vec_compRelSEM()`: Composite reliability.

Author(s)

Shu Fai Cheung https://orcid.org/0000-0002-9871-9448

See Also

`lavaan::lavInspect()`

Examples

```
# From the help page of lavaan::cfa().

library(lavaan)
HS.model <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
'
fit <- cfa(HS.model, data = HolzingerSwineford1939)

vec_rsquare(fit)
vec_sample_vcov(fit)
vec_sample_var(fit)
vec_est_var(fit)
vec_est_se(fit)

HS.model.sem1 <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
textual ~ a * visual
speed ~ b * textual
ab := a * b
'
fit_sem1 <- sem(HS.model.sem1, data = HolzingerSwineford1939)
```

```
HS.model.sem2 <- '
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
textual ~ a * visual
speed ~ b * textual + cp * visual
ab := a * b
'
fit_sem2 <- sem(HS.model.sem2, data = HolzingerSwineford1939)

vec_def_var(fit_sem1)
vec_def_se(fit_sem1)

vec_lavTestLRT(fit_sem1, fit_sem2,
               model.names = c("No Direct", "Direct"))

vec_lavTestScore(fit_sem1,
                 add = "speed ~ visual")

vec_lavTestWald(fit_sem2,
                constraints = "cp == 0")

if (requireNamespace("semTools")) {
    vec_compRelSEM(fit)
  }
```

Index

∗ **datasets**
    dvs_ivs, 9
    simple_mediation, 37

abline(), *20*
add_exo_cov, 2
add_exo_cov(), *3*
add_sig, 4
annotate_matrices, 6
annotate_matrices(), *7*
auto_cov *(*add_exo_cov*)*, 2
auto_exo_cov *(*add_exo_cov*)*, 2
auto_exo_cov(), *3*

boot::boot(), *20*
boot::boot.ci(), *41*

cat(), *3*
compare_estimators, 8
compare_estimators(), *8*
compare_ptables *(*ptable_to_syntax*)*, 28
compare_ptables(), *29*

dvs_ivs, 9

filter_by, 10
fitMeasures_by_models, 12
fitMeasures_by_models(), *26*

get_boot_def *(*store_boot_def*)*, 42
get_boot_def(), *43*
get_boot_est_std
        *(*standardizedSolution_boot_ci*)*, 39
get_boot_est_std(), *40, 41*
ggplot2::ggplot(), *23*, *33*
ggrepel::geom_label_repel(), *22*
group_by_dvs *(*group_estimates*)*, 17
group_by_dvs(), *5*, *10*, *18*, *38*
group_by_groups, 13
group_by_ivs *(*group_estimates*)*, 17
group_by_ivs(), *18*
group_by_models, 15
group_by_models(), *8*
group_estimates, 17

hist(), *19, 20*

lapply(), *8*
lavaan::bootstrapLavaan(), *40*, *45*
lavaan::cfa(), *7, 8*, *28*, *30–36*, *42*
lavaan::fitMeasures(), *12*, *21*, *23*, *25*
lavaan::lavaan, *14*, *19, 20*, *22*, *31*, *40–42*
lavaan::lavaan(), *7, 8*, *20*, *29–36*, *42*
lavaan::lavInspect(), *46*
lavaan::lavParseModelString(), *28, 29*
lavaan::lavTestLRT(), *45*
lavaan::lavTestScore(), *45*
lavaan::lavTestWald(), *45*
lavaan::parameterEstimates(), *5*, *10*, *14–18*, *38*
lavaan::parameterTable(), *28*, *30*
lavaan::sem, *33, 34*
lavaan::sem(), *3*, *7, 8*, *28–36*, *42*
lavaan::standardizedSolution(), *5*, *10*, *14–18*, *27*, *38*, *40, 41*
lines(), *19, 20*

plot.fit_history *(*record_history*)*, 30
plot_boot, 18
plot_boot(), *20*, *40, 41*, *43*
plot_models_fm, 22
print(), *24, 25*, *27*
print.annotate_matrices
        *(*annotate_matrices*)*, 6
print.est_table, 24
print.fit_by_models, 25
print.fit_by_models(), *12*
print.fit_history *(*record_history*)*, 30
print.show_more_options
        *(*show_more_options*)*, 33

print.show_options *(*show_options*)*, 35
print.std_solution_boot, 26
print.std_solution_boot(), *41*
ptable_to_syntax, 28
ptable_to_syntax(), *29*

qqline(), *20*
qqnorm(), *20*

record_history, 30
record_history(), *31*

se_ratios *(*compare_estimators*)*, 8
semTools::compareFit(), *25*
show_cfi, 32
show_ifi *(*show_cfi*)*, 32
show_ifi(), *32, 33*
show_more_options, 33
show_more_options(), *34*
show_options, 35
show_options(), *34–36*
show_tli *(*show_cfi*)*, 32
simple_mediation, 37
sort_by, 38
standardizedSolution_boot_ci, 39
standardizedSolution_boot_ci(), *19, 20*, *26, 27*, *40*
store_boot_def, 42
store_boot_def(), *19–21*, *42, 43*
store_boot_est_std *(*standardizedSolution_boot_ci*)*, 39
store_boot_est_std(), *19–21*, *40, 41*
summary(), *27*, *41*

update(), *8*

vec_compRelSEM *(*vec_rsquare*)*, 44
vec_def_se *(*vec_rsquare*)*, 44
vec_def_var *(*vec_rsquare*)*, 44
vec_est_se *(*vec_rsquare*)*, 44
vec_est_var *(*vec_rsquare*)*, 44
vec_lavTestLRT *(*vec_rsquare*)*, 44
vec_lavTestScore *(*vec_rsquare*)*, 44
vec_lavTestWald *(*vec_rsquare*)*, 44
vec_rsquare, 44
vec_sample_var *(*vec_rsquare*)*, 44
vec_sample_vcov *(*vec_rsquare*)*, 44
vector_from_lavaan *(*vec_rsquare*)*, 44