Package 'semeqmodels'

September 23, 2026

Title Equivalent Models in Structural Equation Models

Version 0.1.0

Description For identifying the sets of empirically equivalent models for structural equation models fitted by the 'lavaan' package developed by Rosseel (2012) <doi:10.18637/jss.v048.i02>.

License GPL (>= 3)

Encoding UTF-8

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

Config/testthat/edition 3

Config/testthat/parallel true

Imports lavaan (>= 0.7.2), modelbpp (>= 0.4.0), digest, manymome (>= 0.3.7), igraph, semPlot, semptools (>= 0.4.0), RColorBrewer, cli

Depends R (>= 4.1.0)

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

LazyData true

Config/roxygen2/version 8.1.0

VignetteBuilder knitr

NeedsCompilation no

Author Shu Fai Cheung [aut, cre] (ORCID: <https://orcid.org/0000-0002-9871-9448>), Borui Yang [aut] (ORCID: <https://orcid.org/0009-0006-7532-8950>), Wenting Xu [aut] (ORCID: <https://orcid.org/0000-0002-6671-3308>), Wendie Yang [aut] (ORCID: <https://orcid.org/0009-0000-8388-6481>)

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

Repository CRAN

Date/Publication 2026-09-23 03:50:02 UTC

Contents

`auto_curve_covariance` *Make a Covariance a Curve*

Description

Identify covariances in a `qgraph` object and set their curve values.

Usage

```
auto_curve_covariance(qgraph_obj, base_curve = 3)
```

Arguments

`qgraph_obj`	A qgraph object.
`base_curve`	The curve value used to make a line a curve. The same value used by the argument `curve` of `semPlot::semPaths()`.

Details

This function can be used to automatically identify covariances in a model, which are represented by bidirectional edges, and set each of the line as a curve.

Value

An object of the same class as the `qgraph_obj`, with lines of covariances set to curves, if any.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit1 <- sem(
          model = mod1,
          data = data_test_3_factor_3_item
        )
fit1_1_more <- drop_k(fit1)
fit1_1_more_1_less <- lapply(
  fit1_1_more,
  add_k
)

partables1 <- combine_partables(fit1_1_more_1_less)
eq_out_1 <- eq_models(
          partables1,
          original_model = fit1,
          parallel = FALSE
        )

eq_out_1 <- eq_models(
          partables1,
          original_model = fit1,
          parallel = FALSE
        )

layout_i <- matrix(c(  NA, "fm",  NA,
                     "fx",   NA, "fy"),
                   ncol = 3,
                   nrow = 2,
                   byrow = TRUE)
layout_i
p <- partables_plots(
  eq_out_1,
  original_model = fit1,
  layout = layout_i,
  label.cex = 1.5,
  sizeLat = 15,
  edge.width = 5,
  asize = 5,
  structural = TRUE,
  par_diff_settings = list(
```

```
            color = "blue",
            width = 10
          )
)
plot(
  p,
  ncol = 3,
  nrow = 2
)

# Process the covariances
p2 <- p %p>% auto_curve_covariance()
plot(
  p2,
  ncol = 3,
  nrow = 2
)
```

`data_test_3obvs` *Test Dataset: 3 Observed Variables*

Description

A dataset for testing.

Usage

```
data_test_3obvs
```

Format

A data frame with 200 rows and 3 variables:

fx Numeric.

fm Numeric.

fy Numeric.

Examples

```
library(lavaan)
data(data_test_3obvs)
mod <-
"
fm ~ fx
fy ~ fm + fx
"
fit <- sem(mod, data_test_3obvs)
parameterEstimates(fit)
```

`data_test_3_factor_3_item` *Test Dataset: 3-Factor-3-Item*

Description

A dataset for testing.

Usage

```
data_test_3_factor_3_item
```

Format

A data frame with 200 rows and 3 variables:

x1 Numeric.

x2 Numeric.

x3 Numeric.

m1 Numeric.

m2 Numeric.

m3 Numeric.

y1 Numeric.

y2 Numeric.

y3 Numeric.

Examples

```
library(lavaan)
data(data_test_3_factor_3_item)
mod <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
"
fit <- sem(mod, data_test_3_factor_3_item)
parameterEstimates(fit)
```

data_test_4obvs *Test Dataset: 4 Observed Variables*

Description

A dataset for testing.

Usage

```
data_test_4obvs
```

Format

A data frame with 500 rows and 4 variables:

fx Numeric.

fm1 Numeric.

fm2 Numeric.

fy Numeric.

Examples

```
library(lavaan)
data(data_test_4obvs)
mod <-
"
fm1 ~~ fm2
fm1 ~ fx
fm2 ~ fx
fy ~ fm1 + fm2 + fx
"
fit <- sem(mod, data_test_4obvs)
parameterEstimates(fit)
```

data_test_4_factor_3_item *Test Dataset: 4-Factor-3-Item*

Description

A dataset for testing.

Usage

```
data_test_4_factor_3_item
```

Format

A data frame with 500 rows and 4 variables:

x1 Numeric.

x2 Numeric.

x3 Numeric.

m1 Numeric.

m2 Numeric.

m3 Numeric.

m4 Numeric.

m5 Numeric.

m6 Numeric.

y1 Numeric.

y2 Numeric.

y3 Numeric.

Examples

```
library(lavaan)
data(data_test_4_factor_3_item)
mod <-
"
fx =~ x1 + x2 + x3
fm1 =~ m1 + m2 + m3
fm2 =~ m4 + m5 + m6
fy =~ y1 + y2 + y3
"
fit <- sem(mod, data_test_4_factor_3_item)
parameterEstimates(fit)
```

---

digest_partable *Helpers to Compare Models*

Description

Helper functions that compute and use hash digests for models (parameter tables) to facilitate comparisons.

Usage

```
digest_partable(
  partable,
 cols = getOption("semeqmodels.digest_cols", default = c("lhs", "op", "rhs", "block",
    "group", "free", "ustart", "start")),
  digits = getOption("semeqmodels.digest_digits", default = 6),
  sort_rows = TRUE,
 sort_by = getOption("semeqmodels.digest_sort_by", default = c("lhs", "op", "rhs",
    "block", "group")),
  algo = getOption("semeqmodels.algo", default = "xxhash32"),
  ...
)

add_digest(partable, ..., overwrite = FALSE)

get_digest(partable, ...)

add_digest_partables(partables, ..., overwrite = FALSE)

get_digest_partables(partables, ...)
```

Arguments

partable	A lavaan parameter table. If it is a lavaan output, the parameter table will be retrieved by lavaan::parameterTable().
cols	The columns to be used to compute the hash value. Note that start and ustart will be rounded based on digits, free will be converted to 0s and 1s (any values greater than 0 will be converted to 0), and start and ustart will be set to NA for free parameters.
digits	The number of digits when rounding ustart and start by round().
sort_rows	Whether the parameter tables will be sorted by cols before computing the hash value.
sort_by	The columns used when sorting the rows, if sort_rows is TRUE.
algo	The algorithm used to by digest::digest(). Note that the default value is different from that of digest::digest()
...	For digest_partable(), these are optional arguments to be passed to digest::digest(). For add_digest(), get_digest(), add_digest_partables(), and get_digest_partables(), these are arguments to be passed to digest_partable().
overwrite	Logical. If TRUE, the stored hash value, if it exists, will be overwritten. If FALSE, the stored hash value, if it exists, will not be overwritten.
partables	A list of lavaan parameter tables.

Details

The function digest_partable() uses digest::digest() to compute a hash value for a parameter, after sorting some essential columns (see the cols argument) If two lavaan parameter tables

are identical on the values for these columns, they should have the same hash value and will be considered as identical.

Note that there may be cases in which two models cannot be correctly identified using this approach. Nevertheless, they should be sufficient for typical models used in this package.

The function `add_digest()` computes the hash value of a parameter table and adds it to the attribute `"digest"` of the table. If the attribute is already set, it will not overwrite it unless `overwrite` is set to `TRUE`.

The function `get_digest()` retrieves the stored hash value from a parameter table, if available. If not available, it will call `digest_partable()` to compute the hash value.

The function add_digest_partables calls `add_digest()` on a list of parameter tables.

The function get_digest_partables calls `get_digest()` on a list of parameter tables.

Value

The function `digest_partable()` returns a character string, the output of `digest::digest()`.

The function `add_digest()` returns the parameter table, with the hash value stored in the attribute `"digest"`.

The function `get_digest()` returns the hash value of a parameter table.

The function add_digest_partables returns the list of parameter tables, with hash values stored.

The function get_digest_partables returns a character vector, the outputs of `get_digest()` for the parameter tables.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit1 <- sem(
          model = mod1,
          data = data_test_3_factor_3_item,
          do.fit = FALSE
        )

mod2 <-
"
fy =~ y1 + y2 + y3
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy ~ fm + fx
```

```
fm ~ fx
"
fit2 <- sem(
          model = mod2,
          data = data_test_3_factor_3_item,
          do.fit = FALSE
        )

pt1 <- parameterTable(fit1)
pt2 <- parameterTable(fit2)

pt1
pt2

digest_partable(pt1)
digest_partable(pt2)
```

`eq_df_models` *Generate Models with the Same Degrees of Freedom*

Description

Generate a list of models with *df* equal to a fitted model.

Usage

```
eq_df_models(
  sem_out,
  fit_models = FALSE,
  loadings_to_exclude_from_drop = "all",
  must_not_drop = NULL,
  must_not_add = NULL,
  se = "none",
  exclude_x_y_ecov = TRUE,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  progress = interactive(),
  gen_models_progress = FALSE,
  short_names = TRUE,
  must_not_add_nil_parameters = TRUE,
  save_history = FALSE
)
```

Arguments

`sem_out`	A `lavaan` object, which is usually the output of `lavaan::sem()` or similar wrappers. Models will be generated from this model.

`fit_models` Whether the models will be fitted to the data. To be passed to `drop_k()` and `add_k()`. Usually should be left as `FALSE`, the default, to speed up the search because usually only the final set of models need to be fitted.

`loadings_to_exclude_from_drop`
How factor loadings will be handled. Default is `"all"` and no factor loadings will be dropped. To be passed to `modelbpp::gen_models()`. This argument should not be changed. Included for internal use.

`must_not_drop` A character vector of parameters that must not be removed, and so will not be modified. To be passed to `modelbpp::gen_models()`.

`must_not_add` A character vector of parameters that must not be added. To be passed to `modelbpp::gen_models()`.

`se` Whether standard error will be computed. This argument will be passed to `lavaan::lavaan()`. Default is `"none"`, and this setting overrides the setting in `object`. The standard errors are irrelevant in checking whether two models are equivalent.

`exclude_x_y_ecov`
If `TRUE`, models with a covariance between a variable (latent or observed) and the error term of another variable it predicts, either directly or indirectly, will be excluded. The screening is implemented by `remove_x_y_ecov()`.

`parallel` Whether parallel processing will be used. If possible, should be set to `TRUE` to speed up the search.

`ncores` The number of CPU cores to be used if `parallel` is `TRUE`.

`progress` If `TRUE`, messages will be displayed to report the progress of the search.

`gen_models_progress`
If `TRUE`, then progress in each call to `drop_k()` or `add_k()` will also be displayed.

`short_names` If `TRUE`, then short names (from the digests generated by `digest_partable()`) will be used to name the models. Though these names are not meaningful words, the full names describing the changes can be very long.

`must_not_add_nil_parameters`
If `TRUE`, nil parameters (paths or covariances fixed to zero) will not be added in the search, implemented by including these parameters in `must_not_add`.

`save_history` Logical. If `TRUE`, the search history will be saved.

Details

The following steps will be repeated to generate models with the same *df* as a fitted model:

First, models with one more *df* than the fitted model will be generated, by fixing more free parameters to zero. This step is conducted by `drop_k()`.

Second, for each of the one-more-*df* model, models with one less *df* will be generated, usually by setting one fixed parameter to free (e.g., adding a regression path). These models will then have the same model *df* as the fitted model. This step is conducted by `add_k()`.

These two steps will be repeated until no more new models are found.

Value

The function eq_df_models() returns an eq_partables object (a subclass of partables), which is a list of models represented by parameter tables.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fm ~ fx
fy ~ fm
"
fit1 <- sem(
          model = mod1,
          data = data_test_3obvs,
          fixed.x = FALSE
        )

# Remove 'parallel = FALSE' or set parallel to TRUE
# for faster generation.
out <- eq_df_models(
  sem_out = fit1,
  parallel = FALSE
)
out
```

---

eq_models *Empirical Equivalent Models*

Description

Identify model(s) in a list of models (parameter tables) empirically equivalent to the original model.

Usage

```
eq_models(
  partables = NULL,
  original_model = NULL,
  ...,
  se = "none",
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = interactive(),
  tolerance = c(chisq = 1e-05),
```

```
  eq_df_models_args = list()
)

is_eq(
  partables = NULL,
  original_model = NULL,
  ...,
  se = "none",
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = interactive(),
  tolerance = c(chisq = 1e-05),
  eq_df_models_args = list()
)
```

Arguments

partables	A list of the class partables. If NULL, eq_models() will try to generate the models by calling eq_df_models() on the argument of original_model. For [is_eq()], this argument cannot be NULL.
original_model	The original model, fitted by lavaan::lavaan() or its wrapper, such as lavaan::sem(). If it is a lavaan parameter table (the output of lavaan::parameterTable()), data will be simulated to fit the model. If it is NULL, then the first model in partables will be used.
...	Optional arguments to be used when fitting models to the data, to be passed to lavaan::sem(). Usually can be omitted.
se	How standard errors are to be computed. To be passed to lavaan::sem(). The default, "none", is sufficient because the standard errors are not needed to check whether two models are empirically equivalent.
parallel	Whether parallel processing will be used when fitting models. Default is TRUE. To be passed to modelbpp::fit_many().
ncores	The number of CPU cores to use when parallel processing is used. To be passed to modelbpp::fit_many().
make_cluster_args	Additional arguments to be passed to modelbpp::fit_many() when creating a cluster for parallel processing. To be passed to modelbpp::fit_many().
progress	Whether the testing progress will be displayed on screen.
tolerance	The maximum absolute difference in a fit measure for two models to be considered empirically equivalent. It should be a named numeric vector, with the names being an acceptable value of the name of fit measures in lavaan::fitMeasures(). For example, the default fit measure is "chisq", model chi-square. If set to c(chisq = 1e-5, cfi = .01), then two models are considered empirically equivalent if their differences on model chi-square and CFI are at most 1e-5 and .01, respectively.

`eq_df_models_args`
: If `partables` is not supplied (`NULL`) but `original_model` is set, `eq_df_models()` will be called to generate the models. This argument must be a named list of additional arguments to be passed to `eq_df_models()`.

Details

`eq_models()`:

The function `eq_models()` checks the model degrees of freedom and model chi-squares of a list of models (represented by `lavaan` parameter tables) against an original model, fitted to a sample, to identify models that are *empirically equivalent* to the original model *in this sample*.

Two models are empirically equivalent if they (a) have the same model degrees of freedom and (b) have a difference in model chi-squares equal to or less than a tolerance (controlled by the argument `tolerance`).

If two models are mathematically equivalent, then they must be empirically equivalent.

However, even if two models are not mathematically equivalent, they may still be empirically equivalent for a sample.

How to Generate the List of Models to Check:

Usually, the alternative models can be generated automatically by leaving `partables` at its default value (`NULL`). The function `eq_df_models()` will be called using `original_model` as the original model. The generation can be customized by setting the argument `eq_df_models_args`.

Alternatively, the function `eq_df_models()` can be called directly to generate an initial list of models. This list can then be filtered by helpers in partable_select, such as `must_be_y()` or `must_not_have_paths()`, and use the resulting list as `partables`.

`is_eq()`:

The function `is_eq()` is similar to `eq_models()`, but returns a logical vector to indicate which models in `partables` are empirically equivalent to the original model.

Value

The function `eq_models()` returns a list of the class `eq_partables` (a subclass of `partables`) of models that are empirically equivalent to the original model.

The function `is_eq()` returns a logical vector of the same length of `patables`, with `TRUE` denoting that a model is empirically equivalent to `original_model`.

References

Pesigan, I. J. A., Cheung, S. F., Wu, H., Chang, F., & Leung, S. O. (2026). How plausible is my model? Assessing model plausibility of structural equation models using Bayesian posterior probabilities (BPP). *Behavior Research Methods*, *58*(3), 73. doi:10.3758/s1342802502921x

See Also

`modelbpp::fit_many()` for the function used to fit the models.

Examples

```
library(lavaan)

# For illustration, only a few models are generated below,
# using drop_k() and add_k manually.
# These two functions are usually not used directly.

# Model 1

mod1 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit1 <- sem(
          model = mod1,
          data = data_test_3_factor_3_item
        )
fit1_1_more <- drop_k(fit1)
fit1_1_more_1_less <- lapply(
  fit1_1_more,
  add_k
)

# Model 3

mod3 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm
"
fit3 <- sem(
          model = mod3,
          data = data_test_3_factor_3_item
        )
fit3_1_more <- drop_k(fit3)
fit3_1_more_1_less <- lapply(
  fit3_1_more,
  add_k
)

# All equivalent
partables1 <- combine_partables(fit1_1_more_1_less)

# Some equivalent
partables3 <- combine_partables(fit3_1_more_1_less)
```

```
eq_out_1 <- eq_models(
          partables1,
          original_model = fit1,
          parallel = FALSE
        )

eq_out_3 <- eq_models(
          partables3,
          original_model = fit3,
          parallel = FALSE
        )

# The usual way to use eq_models:
# 'parallel' should be set to TRUE or omitted
# eq_out_all <- eq_models(
#           original_model = fit1
#         )
# eq_out_all


# Using is_eq()

is_eq(
  partables1,
  original_model = fit1,
  parallel = FALSE
)

is_eq(
  partables3,
  original_model = fit3,
  parallel = FALSE
)
```

`eq_partables_helpers` *Helpers for 'eq_partables' Object*

Description

Helpers to work with an `eq_partables` object.

Usage

```
eq_lavInspect(object, ..., simplify = FALSE)

eq_fitMeasures(object, ..., output_format = c("data.frame", "list"))

eq_df(object)
```

```
eq_chisq(object)

eq_fits(object)

## S3 method for class 'eq_partables'
x[i]

## S3 replacement method for class 'eq_partables'
x[i] <- value

## S3 replacement method for class 'eq_partables'
x[[i]] <- value

## S3 method for class 'eq_partables'
print(
  x,
  max_models = NULL,
  names_to_use = c("default", "long"),
  wrap_long_names = TRUE,
  readable_long_names = TRUE,
  ...
)

## S3 method for class 'eq_partables'
c(..., drop_duplicated = TRUE)

eq_partables(...)

as_eq_partables(sem_out = NULL, model_name = "original")

## S3 method for class 'eq_partables'
duplicated(x, incomparables = FALSE, ...)

## S3 method for class 'eq_partables'
unique(x, incomparables = FALSE, ...)
```

Arguments

object	An eq_partables object.
...	For eq_lavInspect() and eq_fitMeasures(), these are optional arguments to be passed to the lavaan functions to be called. For the print-method of eq_partables objects, these arguments are not used. For the c-method of eq_partables objects, these are eq_partables objects to be combined. For eq_partables(), it should be lavaan outputs or lavaan parameter tables.
simplify	To be passed to sapply(). Note that the default is FALSE, different from sapply().
output_format	The format of the output of eq_fitMeasures(). If output_format is "data.frame, then the output will be a data frame with the number of columns equal to

	the number of models, and rows equal to the number of values returned by `lavaan::fitMeasures()`. If `output_format` is `"list"`, then the output will be a list of numeric vectors.
`x`	An `eq_partables` object.
`i`	A numeric vector of model position(s), a character vector of model name(s), or a logical vector of model(s) to be selected.
`value`	The value(s) to be assigned to the `eq_partables` object.
`max_models`	The maximum number of models to print. If `NULL`, all models will be printed.
`names_to_use`	If `"default"`, the names in `x`, which may not be descriptive, will be used. If `"long"`, the names from `modelbpp::gen_models()` will be used if available. They can be very long, but describe the changes leading to a model.
`wrap_long_names`	If `TRUE`, long names will be wrapped when printed. Used only when `names_to_use` is `"long"`.
`readable_long_names`	If `TRUE`, the long names will be modified to make them more readable.
`drop_duplicated`	Logical. Whether duplicated models will be removed.
`sem_out`	A `lavaan` object or a `lavaan` parameter table. Can be `NULL`.
`model_name`	The name of the model in the output.
`incomparables`	Not used.

Details

Although the functions are designed to work with the output of `eq_models()`, they also work for a list of models (parameter tables), except for the methods specifically for an `eq_partables` object.

The function `eq_lavInspect()` calls `lavaan::lavInspect()` on the elements of an `eq_partables` object.

The function `eq_fitMeasures()` calls `lavaan::fitMeasures()` on the elements of an `eq_partables` object.

The function `eq_df()` is a wrapper that calls `eq_fitMeasures()` with `fit.measures` set to `"df"`. It always returns a numeric vector.

The function `eq_chisq()` is a wrapper that calls `eq_fitMeasures()` with `fit.measures` set to `"chisq"`. It always returns a numeric vector.

The function `eq_fits()` extracts the `lavaan` outputs stored for each model, if present.

The class `eq_partables` has `[`, `[<-`, and `[[<-` methods for extracting and changing elements.

Though available, it is not advised to assign models to an `eq_partables` object because there is no guarantee that the names still reflect how the models are created.

The `print`-method of `eq_partables` object handles a zero-length list. If not of zero length, the `print`-method for `partables` will be used.

The `c`-method of `eq_partables` objects combines `eq_partables` elements into one single `eq_partables` element.

The function eq_partables() creates an eq_partables object from lavaan parameter tables or lavaan outputs.

The function as_eq_partables() is not a usual as function. It works only on a lavaan output or lavaan parameter table. It converts sem_out to a one-element list of the class eq_partables, with the parameter table as the element and the lavaan output, if sem_out is a lavaan output, in the attribute "fit". If sem_out is not a lavaan object nor a lavaan parameter table, a zero-length eq_partables object will be returned.

The duplicated-method of eq_partables checks whether any models are identical. If yes, the duplicated models, except for the first one, will be denoted as duplicated.

The unique-method of eq_partables returns an eq_partables object with duplicated models, if any, removed.

Value

The function eq_lavInspect() returns the output of lavaan::lavInspect(). Whether it is a vector, list, or other type of objects depends on the argument simplify, used by sapply().

The function eq_fitMeasures() returns the output of lavaan::fitMeasures(). The format is determined by output_format.

The function eq_fits() returns a list of lavaan outputs for the models in object. If absent for a model, the value returned is NULL.

The [, [<-, and [[<- methods return an eq_partables object.

The print-method of eq_partables returns x invisibly. It is called for its side-effect.

The c-method of eq_partables returns a list of the class eq_partables.

The function eq_partables() returns an eq_partables object created from one or more lavaan outputs or lavaan parameter tables.

The function as_eq_partables() returns a one-element eq_partables object if sem_out is a lavaan output or a lavaan parameter table. It returns a zero-length eq_partables object otherwise.

The duplicated-method of eq_partables returns a logical vector to indicate which models, if any, are identical to other models earlier in the list.

The unique-method of eq_partables returns an eq_partables object.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fm ~ fx
fy ~ fm
"
fit1 <- sem(
          model = mod1,
          data = data_test_3obvs,
```

```
          fixed.x = FALSE
        )

# Remove 'parallel = FALSE' or set parallel to TRUE
# for faster generation.
out <- eq_models(
  original_model = fit1,
  parallel = FALSE
)
out

eq_lavInspect(out, "implied")


eq_fitMeasures(out, c("cfi", "tli"))



eq_df(out)


eq_fits(out)


out1 <- out[2:3]

out1
```

`model_diff` *Compare Two Models*

Description

Helper functions to compare two models (parameter tables) and find the differences.

Usage

```
model_diff(
  model_x,
  model_y,
 cols = getOption("semeqmodels.digest_cols", default = c("lhs", "op", "rhs", "block",
    "group", "free", "ustart", "start")),
  digits = 6,
  model_x_name = NULL,
  model_y_name = NULL
)

model_diff_many(
```

```
  target_model,
  other_models,
  ...,
  target_model_name = NULL,
  other_models_names = NULL
)

## S3 method for class 'model_diff'
print(x, format = c("summary", "data.frame"), ...)

## S3 method for class 'model_diff_many'
print(x, format = "summary", ...)
```

Arguments

model_x, model_y	Models as parameter tables (the output lavaan::parameterTable) to be compared. They can also be lavaan objects, the output of functions such as lavaan::sem().
cols	The columns to be compared. Two parameters are considered to be identical if they are identical on these columns (but see Details on how free, ustart, and start are compared).
digits	The number of decimal places used to round ustart and start when doing the comparison.
model_x_name, model_y_name	The names of the models used in the output. If NULL, the name will be generated automatically.
target_model	A model (lavaan parameter table or lavaan output) which other models will be compared with.
other_models	A list of models (lavaan parameter tables or lavaan outputs) to be compared to the target_model by model_diff().
...	For model_diff_many(), these are arguments to be passed to model_diff(). For the print-methods, these arguments are ignored.
target_model_name, other_models_names	Names of the models to be passed to model_diff(). If NULL, they will be generated automatically.
x	The object to be printed.
format	The format of the output when printing model differences. Either a user-friendly summary ("summary") or the original parameter table ("data.frame").

Details

model_diff():

The functions model_diff() takes two models (parameter tables) and identifies their differences, if any.

The columns compared are specified by the argument cols.

For the `free` column, the actual values are ignored. Two parameters are considered identical if they are both free (have non-zero values on `free`).

For the `ustart` and `start` columns, the values are used only if a parameter is fixed. If a parameter is free, they will be recoded to `NA` when being compared.

`model_diff_many()`:

The function `model_diff_many()` compares one model (`target_model`) against several other models (`other_models`) using `model_diff()`.

The `print` method of the output of `model_diff()`:

The `print` method of the output of `model_diff()` prints the differences between models in a user-friendly way.

The `print` method of the output of `model_diff_many()`:

The `print` method of the output of `model_diff_many()` prints the list of model differences in a user-friendly way.

Value

The function `model_diff()` returns a list with two elements, `model_x_only` and `model_y_only`. Each element is a parameter table with parameters that are present only in one of the models. If there is no parameter that is present only in a model, then the element is still a parameter table, though with zero row. The list is of the class `model_diff`, with a `print` method.

The function `model_diff_many()` returns a list of the results of `model_diff()`. The list is of the class `model_diff_many`, with a `print` method.

The `print`-method of the output of `model_diff()` returns `x` invisibly. It is called for its side effect.

The `print`-method of the output of `model_diff_many()` returns `x` invisibly. It is called for its side effect.

Examples

```
library(lavaan)


mod1 <-
"
m ~ x
y ~ m + x
"

mod2 <-
"
m ~ x
y ~ m
"

mod3 <-
"
y ~ m + x
"
```

```
pt1 <- parameterTable(sem(mod1, do.fit = FALSE))
pt2 <- parameterTable(sem(mod2, do.fit = FALSE))
pt3 <- parameterTable(sem(mod3, do.fit = FALSE))

model_diff(pt1, pt2)
model_diff(pt1, pt3)
model_diff(pt2, pt3)
```

`modified_models` *Modified Models*

Description

Generate a list of models with a certain number of degrees different from an original model.

Usage

```
drop_k(
  object,
  ...,
  sem_out = NULL,
  loadings_to_exclude_from_drop = "all",
  df_change_drop = 1,
  must_not_drop = NULL,
  se = "none",
  progress = interactive(),
  fit_models = FALSE,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  drop_original = TRUE,
  add_digest = TRUE,
  dat = NULL
)

add_k(
  object,
  ...,
  sem_out = NULL,
  df_change_add = 1,
  must_not_add = NULL,
  exclude_x_y_ecov = TRUE,
  partable_name = NULL,
  se = "none",
  progress = interactive(),
  fit_models = FALSE,
```

```
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  remove_dropped = TRUE,
  remove_zeros = FALSE,
  add_name = FALSE,
  add_digest = TRUE,
  dat = NULL,
  return_error_msg = FALSE
)
```

Arguments

object	The original model. It can be a lavaan-class object (the output of lavaan::lavaan() or its wrappers, such as lavaan::sem()). It can also be a parameter table generated in lavaan.
...	Optional arguments to be passed to modelbpp::gen_models().
sem_out	A lavaan object. If supplied and fit_models is TRUE, the generate models will be fitted by updating this object.
loadings_to_exclude_from_drop	How factor loadings will be handled. Default is "all" and no factor loadings will be dropped. To be passed to modelbpp::gen_models(). This argument should not be changed. Included for internal use.
df_change_drop	The change in the degrees of freedom when generating simplified models. Default is one. To be passed to modelbpp::gen_models().
must_not_drop	A character vector of parameters that must not be removed, and so will not be modified. To be passed to modelbpp::gen_models().
se	Whether standard error will be computed. This argument will be passed to lavaan::lavaan(). Default is "none", and this setting overrides the setting in object. The standard errors are irrelevant in checking whether two models are equivalent.
progress	Whether the model generation process will be displayed on screen.
fit_models	Whether the models will be fitted to the data.
parallel	Whether parallel processing will be used when fitting the models. To be passed to modelbpp::fit_many().
ncores	The number of CPU cores to be used if parallel is TRUE. To be passed to modelbpp::fit_many().
make_cluster_args	An optional named list of arguments to be used in parallel::makeCluster(). To be passed to modelbpp::fit_many().
drop_original	Logical. Whether the original model will be dropped from the output. Default is TRUE.
add_digest	Logical. If TRUE, add_digest() will be called to add hash values to the parameter table.

dat — The dataset to be used when sem_out is NULL and object is not a lavaan output with data.

df_change_add — The change in the degrees of freedom when adding free parameters. To be passed to modelbpp::gen_models(). Default to one. Should not be changed except for experimental use of this function.

must_not_add — A character vector of parameters that must not be added. To be passed to modelbpp::gen_models().

exclude_x_y_ecov — If TRUE, models with a covariance between a variable (latent or observed) and the error term of another variable it predicts, either directly or indirectly, will be excluded. The screening is implemented by remove_x_y_ecov().

partable_name — The name of the original model. Used only if it cannot be generated from object.

remove_dropped — Whether the previously dropped parameter, if stored, will be removed from the original parameter table. This is necessary for reversing a path. For internal use. Should not be changed.

remove_zeros — Whether a parameter explicitly fixed to zero will be removed before generating modified models. For internal use. Should not be changed.

add_name — Whether the name of the original model will be added as a prefix to the names of the generated models.

return_error_msg — If an error occurred when calling modelbpp::gen_models(), whether it will throw an error or return the error message. Set to TRUE when being called by some functions, to defer error handling to the calling function.

Details

The function drop_k() is a helper to generate a list of models *k* more degrees of freedom different from an original model fitted by lavaan, such as lavaan::sem().

The function add_k() is a helper to generate a list of models *k* less degrees of freedom different from an original model fitted by lavaan, such as lavaan::sem().

Value

The function drop_k() returns a list of the class eq_partables, a subclass of partables. It is an output of modelbpp::gen_models(), which are simplified versions of the original model, usually with one or more paths removed (fixed to zero).

The function add_k() returns a list of the class eq_partables, a subclass of the output of modelbpp::gen_models(), which are more complicated versions of the original model, usually with one or more paths added (set to free).

References

Pesigan, I. J. A., Cheung, S. F., Wu, H., Chang, F., & Leung, S. O. (2026). How plausible is my model? Assessing model plausibility of structural equation models using Bayesian posterior probabilities (BPP). *Behavior Research Methods*, *58*(3), 73. doi:10.3758/s1342802502921x

See Also

modelbpp::gen_models() for how the model generation is implemented.

Examples

```
library(lavaan)

mod <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit <- sem(
          model = mod,
          data = data_test_3_factor_3_item
        )
pt <- parameterTable(fit)

# ==== Generate models one-less-df ====

# ==== drop_k ====

fit_1_more1 <- drop_k(fit)
fit_1_more1

fit_1_more2 <- drop_k(pt)
fit_1_more2


# ==== add_k ====

# Remove 'parallel = FALSE' or use 'parallel = TRUE'
# to enable parallel processing, which is recommended.
fit_1_less <- add_k(
                fit_1_more1[[1]],
                add_name = TRUE,
                parallel = FALSE
              )

fit_1_less
```

---

partable_helpers *Parameter Table Helpers*

Description

Helper functions to manipulate parameter tables. They are exported for advanced users.

Usage

```
combine_partables(object_list, drop_duplicated = TRUE)

setdiff_eq_partables(x, y)

setdiff_partables(x, y)

union_eq_partables(x, y)

intersect_eq_partables(x, y)

setequal_eq_partables(x, y)

setequal_partables(x, y)

is_element_eq_partables(el, set)

is_element_partables(el, set)

match_eq_partables(x, table, nomatch = NA_integer_)

match_partables(x, table, nomatch = NA_integer_)

x %pt_in% table

x %pt_notin% table

is_partable(object, colchk = c("id", "lhs", "op", "rhs"))

is_partables(object, colchk = c("id", "lhs", "op", "rhs"))
```

Arguments

object_list	A list of objects of the class partables or eq_partables.
drop_duplicated	Logical. Whether duplicated models will be removed.
x, y	List of parameter tables, such as eq_partables objects.
el	A parameter table.
set	A list of parameter tables.
table	A list of parameter tables.
nomatch	The same argument from match().
object	The object to be checked whether it is a parameter table.
colchk	The columns to be checked.

Details

The function combine_partables() combines a list of partables objects or eq_partables objects to one single object of the same type.

The function setdiff_eq_partables() (and setdiff_partables()) removes from x models that are also in y.

The function union_eq_partables() combines the models in x and y, with duplicated models removed.

The function intersect_eq_partables() finds the models common in x and y.

The function setequal_eq_partables() (and setequal_partables()) checks whether x and y has the same set of models. Orders are ignored.

The function is_element_eq_partables() checks whether el is one of the models in set.

The function match_eq_partables() (and match_partables()) is similar to match(), but check matches based on the results of get_digest() applied to the parameter tables.
%pt_in% is similar to %in%, but works on parameter tables using match_eq_partables().
%pt_notin% is similar to %notin%, but works on parameter tables using match_eq_partables().

The function is_partable() checks whether an object is probably a parameter table. It checks whether (a) the object is a data.frame-like object (by is.data.frame()) and (b) the columns in colchk exist. If both conditions are met, then the object is considered a parameter table.

The function is_partables() checks whether a list is likely a list of parameter tables. It simply calls is_partable() on all the elements.

Value

The function combine_partables() always returns an object of the class eq_partables, which is a subclass of partables.

The function setdiff_eq_partables() returns a list of parameter tables, of the same class of x, with models present in y removed.

The function union_eq_partables() returns a list of parameter tables, of the class eq_partables.

The function intersect_eq_partables() returns a list of parameter tables, of the class eq_partables, common in both x and y.

The function setequal_eq_partables() returns TRUE or FALSE, based on the results of get_digest_partables() applied to x and y.

The function is_element_eq_partables() (and is_element_partables()) returns TRUE or FALSE, based on the results of is.element() applied to the hash values from get_digest() and get_digest_partables().

The function match_eq_partables() (and match_partables()) returns the results of match() applied to the hash values generated by get_digest().
%pt_in% returns a logical vector, the output of %in% applied to the has values of x and table.
%pt_notin% returns a logical vector, the output of %notin% applied to the has values of x and table.

The function is_partable() returns either TRUE or FALSE. It is TRUE if the two conditions mentioned in Details are met.

The function is_partables() returns either TRUE or FALSE. It is TRUE only if is_partable() returns TRUE for all its elements.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit1 <- sem(
          model = mod1,
          data = data_test_3_factor_3_item
        )
fit1_1_more <- drop_k(fit1)
fit1_1_more_1_less <- lapply(
  fit1_1_more,
  add_k
)

partables1 <- combine_partables(fit1_1_more_1_less)
partables1
```

---

partable_select *Select Models (Parameter Tables)*

Description

Helper functions to select models (parameter tables) based on various criteria.

Usage

```
have_pars_all(
  partables,
  pars = NULL,
  output = c("models", "partables", "logical")
)

have_pars_any(
  partables,
  pars = NULL,
  output = c("models", "partables", "logical")
)

have_pars_none(
```

```
  partables,
  pars = NULL,
  output = c("models", "partables", "logical")
)

remove_x_y_ecov(
  partables,
  output = c("models", "partables", "logical"),
  cl = NULL
)

must_not_be_y(
  partables,
  vars = NULL,
  output = c("models", "partables", "logical")
)

must_be_y(partables, vars = NULL, output = c("models", "partables", "logical"))

must_not_have_paths(
  partables,
  y_on_x = NULL,
  output = c("models", "partables", "logical")
)

must_have_paths(
  partables,
  y_on_x = NULL,
  output = c("models", "partables", "logical")
)
```

Arguments

partables	A list of parameter tables, such as a partables or eq_partables object. It can also be the output of partables_plots().
pars	A character vector of lavaan model syntax that can be converted to a parameter table. It can be a vector of parameters, such as c("y ~ x", "m ~~ x"), but can also be of other forms as long as lavaan::lavParseModelString() can process it. Covariances such as "m ~~ x" and "x ~~ m" are treated as the same and so only one of them is necessary.
output	The type of output. If "models" or "partables", the output is a subset of the partables, which can be a list of parameter tables or a list of plots of parameter tables. If "logical", a logical vector of the same length as partables is returned to indicate models that match the selection criteria.
cl	A cluster created by parallel::makeCluster(). Used internally. Do not set this argument.
vars	A character vector of variables to be checked.

y_on_x A character vector of pairs of variables, specified as "y ~ x", for which paths will be checked.

Details

The select functions can be used both for a list of models in the form of parameter tables (the output of lavaan::parameterTable()) or a list of plots based on these models, generated by partables_plots(). They return an object of the same type. Therefore, they can be used to filter both models and plots of models.

Select by free parameters:

The function have_pars_all() selects models that have all the free parameters specified in pars.

The function have_pars_any() identifies models that have any of the free parameters specified in pars.

The function have_pars_none() identifies models that have none of the free parameters specified in pars.

Select by covariances with an error term:

The function remove_x_y_ecov() removes models from partables that have a covariance between an observed or latent variable (or its error term) and the error term of another variable it predicts, either directly or indirectly.

Select by the role of a variable:

The function must_not_be_y() keeps only models with selected variables (observed or latent) not appearing as the outcome in a regression equation (indicators not counted). They are defined as variables not in "eqs.y" as returned by lavaan::lavNames().

The function must_be_y() keeps only models with selected variables (observed or latent) appearing as the outcome in at least one regression equation (indicators not counted). They are defined as variables in "eqs.y" as returned by lavaan::lavNames().

Select by paths:

The function must_not_have_paths() keeps only models that do not have any paths, direct or indirect, between selected pairs of variables.

The function must_have_paths() keeps only models with at least one path, direct or indirect, between selected pairs of variables.

Value

Each of the select functions, by default, returns a list of parameter tables (models) that meet the criterion of that function, or a list of plots of the parameter tables if partables is the output of partables_plots(). If output is set to "logical", each of these functions returns a logical vector to indicate models or plots that meet the criterion. The vector can then be used to extract objects meeting the criterion.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit1 <- sem(
          model = mod1,
          data = data_test_3_factor_3_item
        )
fit1_1_more <- drop_k(fit1)
fit1_1_more_1_less <- lapply(
  fit1_1_more,
  add_k
)

partables1 <- combine_partables(fit1_1_more_1_less)
partables1

# === have_pars_all ====

have_pars_all(partables1, c("fx ~~ fm", "fy ~ fx"))
have_pars_all(partables1, c("fx ~~ fm", "fy ~ fx"),
              output = "logical")


# === have_pars_any ====

have_pars_any(partables1, c("fx ~~ fm", "fm ~ fx"))
have_pars_any(partables1, c("fx ~~ fm", "fm ~ fx"),
              output = "logical")


# === have_pars_none ====

have_pars_none(partables1, c("fx ~~ fm", "fm ~ fx"))
have_pars_none(partables1, c("fx ~~ fm", "fm ~ fx"),
               output = "logical")


# === must_not_be_y ====

must_not_be_y(partables1, c("fx"))
must_not_be_y(partables1, c("fx"),
              output = "logical")
```

```
# === must_be_y ====

must_be_y(partables1, c("fm"))
must_be_y(partables1, c("fm"),
          output = "logical")


# === must_not_have_paths ====

must_not_have_paths(partables1, y_on_x = c("fx ~ fm"))
must_not_have_paths(partables1, y_on_x = c("fx ~ fm"),
                    output = "logical")


# === must_not_have_paths ====

must_have_paths(partables1, y_on_x = c("fm ~ fx"))
must_have_paths(partables1, y_on_x = c("fm ~ fx"),
                output = "logical")
```

---

plot_partables *Plot Functions for a List of Models*

Description

Various plot functions for the output of eq_models(), eq_df_models(), and similar functions.

Usage

```
partables_plots(
  object,
  ...,
  partables = NULL,
  original_model = NULL,
  auto_node_color = TRUE,
  par_diff_settings = list(color = "blue", width = 2),
  fix_pars_fixed_zero = TRUE,
  par_fixed_zero_settings = list(color = "white", width = 0),
  exclude_original_model = FALSE,
  curve_cov = TRUE,
  curve_cov_settings = list(base_curve = 1.5)
)

## S3 method for class 'partables_plots'
plot(
```

```
  x,
  ...,
  title_mode = c("digest", "name", "none"),
  title_adj = 1.4,
  title_args = list(),
  ncol = 1,
  nrow = 1,
  scale_plots = c("auto", "always"),
  scale = NULL,
 elements_to_scale = c("Nodes:width", "Nodes:height", "Edges:width", "Edges:asize"),
  original_model_mode = c("include", "exclude", "side_by_side")
)

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

lhs %p>% rhs
```

Arguments

object	If it is a list of models (parameter tables), such as the output of eq_models(), it will be used as the value for partables. If it is a partables_plots object (i.e., an output of partables_plots(), then it will be updated with any new values for other arguments.
...	For plot.partables_plots(), these are optional arguments to be passed to semPlot::semPaths(). For print.partables_plots(), they are not used.
partables	It should be a list of models in the form of parameter tables, such as the output of eq_models() or eq_df_models().
original_model	The model to which models in partables will be compared to. If NULL, then the plots will be generated without checking for differences between a model and original_model. Model comparison is conducted by model_diff().
auto_node_color	If TRUE, and the number of nodes is 12 or less, they will be automatically set to different colors. Ignored if color of nodes is in ....
par_diff_settings	A named list of settings to configure the change of a parameter from original_model to a model in partables. For now, two settings are supported: color for the color and width for the width of an edge (arrow/path).
fix_pars_fixed_zero	If TRUE, parameters fixed to zero will be modified based on par_fixed_zero_settings.
par_fixed_zero_settings	A named settings to configure edges (arrows) fixed to zero. For now, two settings are supported: color for the color and width for the width of an edge (arrow/path). Setting the width to zero, the default, effectively hides an arrow/path.
exclude_original_model	If TRUE and original_model is set, the output will not include the plot of the original model, though this plot will be stored in the attribute "original_model", as a one-element list.

curve_cov
: If TRUE, lines denoting covariances will be automatically curve. Implemented by calling auto_curve_covariance().

curve_cov_settings
: A named list of arguments to be passed to auto_curve_covariance(). Used only if curve_cov is TRUE.

x
: The output of partables_plots(), a partables_plots object.

title_mode
: What will be used as the title. If "digest", then the digest value of a model, generated by digest_partable(), will be used as its title. If "name", then its name in x will be used, which may also be the digest value. If "none", then no title will be drawn with the plot.

title_adj
: Adjust the position of the title. Increase this value if the plot is too close to the title, and decrease this value if the space between the plot and its title is too large.

title_args
: A named list of arguments to be passed to title() when printing the title.

ncol, nrow
: The number of columns and rows when drawing the models. Used by mfrow in par().

scale_plots
: How the models will be scaled. If "auto", then the models will be scaled based on ncol and nrow, using scale as a reference. If "always", then scale will always be used to scale the models, even if they are drawn one by one.

scale
: How the model will be further scaled when drawn. If this value is greater than one, then elements in elements_to_scale will be increased by this ratio. If this value is less than one, then elements in elements_to_scale will be decreased by this ratio.

elements_to_scale
: Elements in the qgraph object that will be scaled based on the values of scale_plots and scale.

original_model_mode
: How original model, if present, will be handled when drawing the models. If "exclude", then the original model will not be drawn. If "include", the original model will be drawn as the first model, along with other models. If "side_by_side", then the number of columns is always two and the number of rows is always one. In each plot, the original model will be drawn on the left and the other model will be drawn on the right.

lhs
: A partables_plots object.

rhs
: A function call to be applied to all stored plots. Each plot will be inserted as the first argument.

Details

The functions provide different ways to visualize the models. Basic knowledge of semPlot::semPaths() from the package semPlot is required to customize the plots.

Value

The function partables_plots() returns a list of qgraph objects generated from semPlot::semPaths(), of the class partables_plots.

The plot method of the output of partables_plots() returns x invisibly. Called for its side effect.

The print method of the output of partables_plots() returns x invisibly. Called for its side effect.

The %p>% operator returns a partables_plots object, processed by the right-hand side function.

See Also

See eq_models() on the type of output supported.

Examples

```
library(lavaan)

# Model 1

mod1 <-
"
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ fx
fy ~ fm + fx
"
fit1 <- sem(
          model = mod1,
          data = data_test_3_factor_3_item
        )
fit1_1_more <- drop_k(fit1)
fit1_1_more_1_less <- lapply(
  fit1_1_more,
  add_k
)

partables1 <- combine_partables(fit1_1_more_1_less)
eq_out_1 <- eq_models(
          partables1,
          original_model = fit1,
          parallel = FALSE
        )

eq_out_1 <- eq_models(
          partables1,
          original_model = fit1,
          parallel = FALSE
        )

layout_i <- matrix(c(  NA, "fm",  NA,
                     "fx",   NA, "fy"),
                   ncol = 3,
                   nrow = 2,
                   byrow = TRUE)
layout_i
```

```
p <- partables_plots(
  eq_out_1,
  original_model = fit1,
  layout = layout_i,
  label.cex = 1.5,
  sizeLat = 15,
  edge.width = 5,
  asize = 5,
  structural = TRUE,
  par_diff_settings = list(
            color = "blue",
            width = 10
          )
)
plot(p)
```

Index