Package 'manymome'

February 16, 2024

Title Mediation, Moderation and Moderated-Mediation After Model Fitting

Version 0.1.14

Description Computes indirect effects, conditional effects, and conditional indirect effects in a structural equation model or path model after model fitting, with no need to define any user parameters or label any paths in the model syntax, using the approach presented in Cheung and Cheung (2023) <doi:10.3758/s13428-023-02224-z>. Can also form bootstrap confidence intervals by doing bootstrapping only once and reusing the bootstrap estimates in all subsequent computations. Supports bootstrap confidence intervals for standardized (partially or completely) indirect effects, conditional effects, and conditional indirect effects as described in Cheung (2009) <doi:10.3758/BRM.41.2.425> and Cheung, Cheung, Lau, Hui, and Vong (2022) <doi:10.1037/hea0001188>. Model fitting can be done by structural equation modeling using lavaan() or regression using lm().

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

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

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.0

Suggests knitr, rmarkdown, semPlot, semptools, semTools, Amelia, mice, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first cond_indirect_*

Imports lavaan, boot, parallel, pbapply, stats, ggplot2, igraph, MASS, methods

Depends R (>= 3.5.0)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Shu Fai Cheung [aut, cre] (<https://orcid.org/0000-0002-9871-9448>),
Sing-Hang Cheung [aut] (<https://orcid.org/0000-0001-5182-0752>)

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

Repository CRAN

Date/Publication 2024-02-16 19:10:14 UTC

R topics documented:

`all_indirect_paths` *Enumerate All Indirect Effects in a Model*

Description

Check all indirect paths in a model and return them as a list of arguments of `x`, `y`, and `m`, to be used by `indirect_effect()`.

Usage

```
all_indirect_paths(fit = NULL, exclude = NULL, x = NULL, y = NULL)

all_paths_to_df(all_paths)
```

Arguments

`fit`	A fit object. Either the output of `lavaan::lavaan()` or its wrapper such as `lavaan::sem()`, or a list of the output of `lm()` or the output of `lm2list()`.
`exclude`	A character vector of variables to be excluded in the search, such as control variables.
`x`	A character vector of variables that will be included as the `x` variables. If supplied, only paths that start from these variables will be included in the search. If `NULL`, the default, then all variables that are one of the predictors in at least one regression equation will be included in the search.
`y`	A character vector of variables that will be included as the `y` variables. If supplied, only paths that start from these variables will be included in the search. If `NULL`, the default, then all variables that are the outcome variables in at least one regression equation will be included in the search.
`all_paths`	An `all_paths`-class object. For example, the output of `all_indirect_paths()`.

Details

It makes use of `igraph::all_simple_paths()` to identify paths in a model.

Value

`all_indirect_paths()` returns a list of the class `all_paths`. Each argument is a list of three character vectors, `x`, the name of the predictor that starts a path, `y`, the name of the outcome that ends a path, and `m`, a character vector of one or more names of the mediators, from `x` to `y`. This class has a print method.

`all_paths_to_df()` returns a data frame with three columns, `x`, `y`, and `m`, which can be used by functions such as `indirect_effect()`.

Functions

- `all_indirect_paths()`: Enumerate all indirect paths.
- `all_paths_to_df()`: Convert the output of `all_indirect_paths()` to a data frame with three columns: `x`, `y`, and `m`.

Author(s)

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

See Also

`indirect_effect()`, `lm2list()`. `many_indirect_effects()`

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)
# All indirect paths
out1 <- all_indirect_paths(fit)
out1
names(out1)

# Exclude c1 and c2 in the search
out2 <- all_indirect_paths(fit, exclude = c("c1", "c2"))
out2
names(out2)

# Exclude c1 and c2, and only consider paths start
# from x and end at y
out3 <- all_indirect_paths(fit, exclude = c("c1", "c2"),
                           x = "x",
                           y = "y")
out3
names(out3)
```

check_path *Check a Path Exists in a Model*

Description

It checks whether a path, usually an indirect path, exists in a model.

Usage

```
check_path(x, y, m = NULL, fit = NULL, est = NULL)
```

Arguments

x	Character. The name of predictor at the start of the path.
y	Character. The name of the outcome variable at the end of the path.
m	A vector of the variable names of the mediators. The path goes from the first mediator successively to the last mediator. If NULL, the default, the path goes from x to y.
fit	The fit object. Currently only supports a lavaan::lavaan object or a list of outputs of lm(). It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().
est	The output of lavaan::parameterEstimates(). If NULL, the default, it will be generated from fit. If supplied, fit will ge ignored.

Details

It checks whether the path defined by a predictor (x), an outcome (y), and optionally a sequence of mediators (m), exists in a model. It can check models in a lavaan::lavaan object or a list of outputs of lm(). It also support lavaan.mi objects returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().

For example, in the ql in lavaan syntax

```
m1 ~ x
m2 ~ m1
m3 ~ x
y ~ m2 + m3
```

This path is valid: `x = "x", y = "y", m = c("m1", "m2")`

This path is invalid: `x = "x", y = "y", m = c("m2")`

This path is also invalid: `x = "x", y = "y", m = c("m1", "m2")`

Value

A logical vector of length one. TRUE if the path is valid, FALSE if the path is invalid.

Examples

```
library(lavaan)
data(data_serial_parallel)
dat <- data_serial_parallel
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE)

# The following paths are valid
check_path(x = "x", y = "y", m = c("m11", "m12"), fit = fit)
check_path(x = "x", y = "y", m = "m2", fit = fit)
# The following paths are invalid
check_path(x = "x", y = "y", m = c("m11", "m2"), fit = fit)
check_path(x = "x", y = "y", m = c("m12", "m11"), fit = fit)
```

---

coef.cond_indirect_diff

Print the Output of 'cond_indirect_diff()'

Description

Extract the change in conditional indirect effect.

Usage

```
## S3 method for class 'cond_indirect_diff'
coef(object, ...)
```

Arguments

`object`	The output of `cond_indirect_diff()`.
`...`	Optional arguments. Ignored.

Details

The `coef` method of the `cond_indirect_diff`-class object.

Value

Scalar: The change of conditional indirect effect in `object`.

See Also

cond_indirect_diff()

---

coef.cond_indirect_effects *Estimates of Conditional Indirect Effects or Conditional Effects*

Description

Return the estimates of the conditional indirect effects or conditional effects for all levels in the output of cond_indirect_effects().

Usage

```
## S3 method for class 'cond_indirect_effects'
coef(object, ...)
```

Arguments

object	The output of cond_indirect_effects().
...	Optional arguments. Ignored by the function.

Details

It extracts and returns the column ind or std in the output of cond_indirect_effects().

Value

A numeric vector: The estimates of the conditional effects or conditional indirect effects.

See Also

cond_indirect_effects()

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ x  + w1 + x:w1
m2 ~ m1
y  ~ m2 + x + w4 + m2:w4
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
```

```
est <- parameterEstimates(fit)

# Conditional effects from x to m1 when w1 is equal to each of the levels
out1 <- cond_indirect_effects(x = "x", y = "m1",
                      wlevels = c("w1"), fit = fit)
out1
coef(out1)

# Conditional indirect effects from x1 through m1 and m2 to y,
out2 <- cond_indirect_effects(x = "x", y = "y", m = c("m1", "m2"),
                      wlevels = c("w1", "w4"), fit = fit)
out2
coef(out2)

# Standardized conditional indirect effects from x1 through m1 and m2 to y,
out2std <- cond_indirect_effects(x = "x", y = "y", m = c("m1", "m2"),
                      wlevels = c("w1", "w4"), fit = fit,
                      standardized_x = TRUE, standardized_y = TRUE)
out2std
coef(out2std)
```

`coef.delta_med` *Delta_Med in a 'delta_med'-Class Object*

Description

Return the estimate of Delta_Med in a 'delta_med'-class object.

Usage

```
## S3 method for class 'delta_med'
coef(object, ...)
```

Arguments

`object`	The output of `delta_med()`.
`...`	Optional arguments. Ignored.

Details

It just extracts and returns the element `delta_med` in the output of `delta_med()`, the estimate of the Delta_Med proposed by Liu, Yuan, and Li (2023), an R^2-like measure of indirect effect.

Value

A scalar: The estimate of Delta_Med.

Author(s)

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

References

Liu, H., Yuan, K.-H., & Li, H. (2023). A systematic framework for defining R-squared measures in mediation analysis. *Psychological Methods*. Advance online publication. https://doi.org/10.1037/met0000571

See Also

delta_med()

Examples

```
library(lavaan)
dat <- data_med
mod <-
"
m ~ x
y ~ m + x
"
fit <- sem(mod, dat)
dm <- delta_med(x = "x",
                y = "y",
                m = "m",
                fit = fit)
dm
print(dm, full = TRUE)
coef(dm)
```

---

coef.indirect *Extract the Indirect Effect or Conditional Indirect Effect*

Description

Return the estimate of the indirect effect in the output of indirect_effect() or or the conditional indirect in the output of cond_indirect().

Usage

```
## S3 method for class 'indirect'
coef(object, ...)
```

Arguments

object	The output of indirect_effect() or cond_indirect().
...	Optional arguments. Ignored by the function.

Details

It extracts and returns the element `indirect`. in an object.

If standardized effect is requested when calling `indirect_effect()` or `cond_indirect()`, the effect returned is also standardized.

Value

A scalar: The estimate of the indirect effect or conditional indirect effect.

See Also

`indirect_effect()` and `cond_indirect()`.

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ x + w1 + x:w1
m2 ~ x
y  ~ m1 + m2 + x
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

# Examples for indirect_effect():

# Inidrect effect from x through m2 to y
out1 <- indirect_effect(x = "x", y = "y", m = "m2", fit = fit)
out1
coef(out1)

# Conditional Indirect effect from x1 through m1 to y,
# when w1 is 1 SD above mean
hi_w1 <- mean(dat$w1) + sd(dat$w1)
out2 <- cond_indirect(x = "x", y = "y", m = "m1",
                      wvalues = c(w1 = hi_w1), fit = fit)
out2
coef(out2)
```

`coef.indirect_list` *Extract the Indirect Effects from a 'indirect_list' Object*

Description

Return the estimates of the indirect effects in the output of `many_indirect_effects()`.

Usage

```
## S3 method for class 'indirect_list'
coef(object, ...)
```

Arguments

`object`	The output of `many_indirect_effects()`.
`...`	Optional arguments. Ignored by the function.

Details

It extracts the estimates in each 'indirect'-class object in the list.

If standardized effect is requested when calling `many_indirect_effects()`, the effects returned are also standardized.

Value

A numeric vector of the indirect effects.

See Also

`many_indirect_effects()`

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)
# All indirect paths from x to y
paths <- all_indirect_paths(fit,
                            x = "x",
```

```
                              y = "y")
paths
# Indirect effect estimates
out <- many_indirect_effects(paths,
                               fit = fit)
out
coef(out)
```

`coef.indirect_proportion` *Extract the Proportion of Effect Mediated*

Description

Return the proportion of effect mediated in the output of `indirect_proportion()`.

Usage

```
## S3 method for class 'indirect_proportion'
coef(object, ...)
```

Arguments

`object`	The output of `indirect_proportion()`
`...`	Not used.

Details

It extracts and returns the element `proportion` in the input object.

Value

A scalar: The proportion of effect mediated.

See Also

`indirect_proportion()`

Examples

```
library(lavaan)
dat <- data_med
head(dat)
mod <-
"
m ~ x + c1 + c2
```

```
y ~ m + x + c1 + c2
"
fit <- sem(mod, dat, fixed.x = FALSE)
out <- indirect_proportion(x = "x",
                           y = "y",
                           m = "m",
                           fit = fit)
out
coef(out)
```

`coef.lm_from_lavaan` *Coefficients of an 'lm_from_lavaan'-Class Object*

Description

Returns the path coefficients of the terms in an `lm_from_lavaan`-class object.

Usage

```
## S3 method for class 'lm_from_lavaan'
coef(object, ...)
```

Arguments

`object`	A 'lm_from_lavaan'-class object.
`...`	Additional arguments. Ignored.

Details

An `lm_from_lavaan`-class object converts a regression model for a variable in a `lavaan`-class object to a `formula`-class object. This function simply extracts the path coefficients estimates. Intercept is always included, and set to zero if mean structure is not in the source `lavaan`-class object.

This is an advanced helper used by `plot.cond_indirect_effects()`. Exported for advanced users and developers.

Value

A numeric vector of the path coefficients.

See Also

`lm_from_lavaan_list()`

Examples

```
library(lavaan)
data(data_med)
mod <-
"
m ~ a * x + c1 + c2
y ~ b * m + x + c1 + c2
"
fit <- sem(mod, data_med, fixed.x = FALSE)
fit_list <- lm_from_lavaan_list(fit)
coef(fit_list$m)
coef(fit_list$y)
```

---

cond_indirect *Conditional, Indirect, and Conditional Indirect Effects*

Description

Compute the conditional effects, indirect effects, or conditional indirect effects in a structural model fitted by `lm()`, `lavaan::sem()`, or `semTools::sem.mi()`.

Usage

```
cond_indirect(
  x,
  y,
  m = NULL,
  fit = NULL,
  est = NULL,
  implied_stats = NULL,
  wvalues = NULL,
  standardized_x = FALSE,
  standardized_y = FALSE,
  boot_ci = FALSE,
  level = 0.95,
  boot_out = NULL,
  R = 100,
  seed = NULL,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE,
  save_boot_full = FALSE,
  prods = NULL,
  get_prods_only = FALSE,
  save_boot_out = TRUE,
```

```
  mc_ci = FALSE,
  mc_out = NULL,
  save_mc_full = FALSE,
  save_mc_out = TRUE,
  ci_out = NULL,
  save_ci_full = FALSE,
  save_ci_out = TRUE,
  ci_type = NULL
)

cond_indirect_effects(
  wlevels,
  x,
  y,
  m = NULL,
  fit = NULL,
  w_type = "auto",
  w_method = "sd",
  sd_from_mean = NULL,
  percentiles = NULL,
  est = NULL,
  implied_stats = NULL,
  boot_ci = FALSE,
  R = 100,
  seed = NULL,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE,
  boot_out = NULL,
  output_type = "data.frame",
  mod_levels_list_args = list(),
  mc_ci = FALSE,
  mc_out = NULL,
  ci_out = NULL,
  ci_type = NULL,
  ...
)

indirect_effect(
  x,
  y,
  m = NULL,
  fit = NULL,
  est = NULL,
  implied_stats = NULL,
  standardized_x = FALSE,
  standardized_y = FALSE,
```

```
  boot_ci = FALSE,
  level = 0.95,
  boot_out = NULL,
  R = 100,
  seed = NULL,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE,
  save_boot_full = FALSE,
  mc_ci = FALSE,
  mc_out = NULL,
  save_mc_full = FALSE,
  save_mc_out = TRUE,
  ci_out = NULL,
  save_ci_full = FALSE,
  save_ci_out = TRUE,
  ci_type = NULL
)

many_indirect_effects(paths, ...)
```

Arguments

x	Character. The name of the predictor at the start of the path.
y	Character. The name of the outcome variable at the end of the path.
m	A vector of the variable names of the mediator(s). The path goes from the first mediator successively to the last mediator. If NULL, the default, the path goes from x to y.
fit	The fit object. Can be a lavaan::lavaan object or a list of lm() outputs. It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().
est	The output of lavaan::parameterEstimates(). If NULL, the default, it will be generated from fit. If supplied, fit will be ignored.
implied_stats	Implied means, variances, and covariances of observed variables, of the form of the output of lavaan::lavInspect() with what set to "implied". The standard deviations are extracted from this object for standardization. Default is NULL, and implied statistics will be computed from fit if required.
wvalues	A numeric vector of named elements. The names are the variable names of the moderators, and the values are the values to which the moderators will be set to. Default is NULL.
standardized_x	Logical. Whether x will be standardized. Default is FALSE.
standardized_y	Logical. Whether y will be standardized. Default is FALSE.
boot_ci	Logical. Whether bootstrap confidence interval will be formed. Default is FALSE.
level	The level of confidence for the bootstrap confidence interval. Default is .95.

boot_out — If boot_ci is TRUE, users can supply pregenerated bootstrap estimates. This can be the output of do_boot(). For indirect_effect() and cond_indirect_effects(), this can be the output of a previous call to cond_indirect_effects(), indirect_effect(), or cond_indirect() with bootstrap confidence intervals requested. These stored estimates will be reused such that there is no need to do bootstrapping again. If not supplied, the function will try to generate them from fit.

R — Integer. If boot_ci is TRUE, boot_out is NULL, and bootstrap standard errors not requested if fit is a lavaan object, this function will do bootstrapping on fit. R is the number of bootstrap samples. Default is 100. For Monte Carlo simulation, this is the number of replications.

seed — If bootstrapping or Monte Carlo simulation is conducted, this is the seed for the bootstrapping or simulation. Default is NULL and seed is not set.

parallel — Logical. If bootstrapping is conducted, whether parallel processing will be used. Default is TRUE. If fit is a list of lm() outputs, parallel processing will not be used.

ncores — Integer. The number of CPU cores to use when parallel is TRUE. Default is the number of non-logical cores minus one (one minimum). Will raise an error if greater than the number of cores detected by parallel::detectCores(). If ncores is set, it will override make_cluster_args in do_boot().

make_cluster_args — A named list of additional arguments to be passed to parallel::makeCluster(). For advanced users. See parallel::makeCluster() for details. Default is list().

progress — Logical. Display bootstrapping progress or not. Default is TRUE.

save_boot_full — If TRUE, full bootstrapping results will be stored. Default is FALSE.

prods — The product terms found. For internal use.

get_prods_only — IF TRUE, will quit early and return the product terms found. The results can be passed to the prod argument when calling this function. Default is FALSE. This function is for internal use.

save_boot_out — If boot_out is supplied, whether it will be saved in the output. Default is TRUE.

mc_ci — Logical. Whether Monte Carlo confidence interval will be formed. Default is FALSE.

mc_out — If mc_ci is TRUE, users can supply pregenerated Monte Carlo estimates. This can be the output of do_mc(). For indirect_effect() and cond_indirect_effects(), this can be the output of a previous call to cond_indirect_effects(), indirect_effect(), or cond_indirect() with Monte Carlo confidence intervals requested. These stored estimates will be reused such that there is no need to do Monte Carlo simulation again. If not supplied, the function will try to generate them from fit.

save_mc_full — If TRUE, full Monte Carlo results will be stored. Default is FALSE.

save_mc_out — If mc_out is supplied, whether it will be saved in the output. Default is TRUE.

ci_out — If ci_type is supplied, this is the corresponding argument. If ci_type is "boot", this argument will be used as boot_out. If ci_type is "mc", this argument will be used as mc_out.

save_ci_full — If TRUE, full bootstrapping or Monte Carlo results will be stored. Default is FALSE.

save_ci_out — If either mc_out or boot_out is supplied, whether it will be saved in the output. Default is TRUE.

ci_type — The type of confidence intervals to be formed. Can be either "boot" (bootstrapping) or "mc" (Monte Carlo). If not supplied or is NULL, will check other arguments (e.g, boot_ci and mc_ci). If supplied, will override boot_ci and mc_ci.

wlevels — The output of merge_mod_levels(), or the moderator(s) to be passed to mod_levels_list(). If all the moderators can be represented by one variable, that is, each moderator is (a) a numeric variable, (b) a dichotomous categorical variable, or (c) a factor or string variable used in lm() in fit, then it is a vector of the names of the moderators as appeared in the data frame. If at least one of the moderators is a categorical variable represented by more than one variable, such as user-created dummy variables used in lavaan::sem(), then it must be a list of the names of the moderators, with such moderators represented by a vector of names. For example: list("w1", c("gpgp2", "gpgp3")), the first moderator w1 and the second moderator a three-categorical variable represented by gpgp2 and gpgp3.

w_type — Character. Whether the moderator is a "numeric" variable or a "categorical" variable. If "auto", the function will try to determine the type automatically. See mod_levels_list() for further information.

w_method — Character, either "sd" or "percentile". If "sd", the levels are defined by the distance from the mean in terms of standard deviation. if "percentile", the levels are defined in percentiles. See mod_levels_list() for further information.

sd_from_mean — A numeric vector. Specify the distance in standard deviation from the mean for each level. Default is c(-1, 0, 1) when there is only one moderator, and c(-1, 1) when there are more than one moderator. Ignored if w_method is not equal to "sd". See mod_levels_list() for further information.

percentiles — A numeric vector. Specify the percentile (in proportion) for each level. Default is c(.16, .50, .84) if there is one moderator, and c(.16, .84) when there are more than one moderator. Ignored if w_method is not equal to "percentile". See mod_levels_list() for further information.

output_type — The type of output of cond_indirect_effects(). If "data.frame", the default, the output will be converted to a data frame. If any other values, the output is a list of the outputs from cond_indirect().

mod_levels_list_args — Additional arguments to be passed to mod_levels_list() if it is called for creating the levels of moderators. Default is list().

... — For many_indirect_effects(), these are arguments to be passed to indirect_effect().

paths — The output of all_indirect_paths()

Details

For a model with a mediation path moderated by one or more moderators, cond_indirect_effects() can be used to compute the conditional indirect effect from one variable to another variable, at one or more set of selected value(s) of the moderator(s).

If only the effect for one set of value(s) of the moderator(s) is needed, cond_indirect() can be used.

If only the mediator(s) is/are specified (m) and no values of moderator(s) are specified, then the indirect effect from one variable (x) to another variable (y) is computed. A convenient wrapper indirect_effect() can be used to compute the indirect effect.

If only the value(s) of moderator(s) is/are specified (wvalues or wlevels) and no mediators (m) are specified when calling cond_indirect_effects() or cond_indirect(), then the conditional direct effects from one variable to another are computed.

All three functions support using nonparametric bootstrapping (for lavaan or lm outputs) or Monte Carlo simulation (for lavaan outputs only) to form confidence intervals. Bootstrapping or Monte Carlo simulation only needs to be done once. These are the possible ways to form bootstrapping:

1. Do bootstrapping or Monte Carlo simulation in the first call to one of these functions, by setting boot_ci or mc_ci to TRUE and R to the number of bootstrap samples or replications, level to the level of confidence (default .95 or 95%), and seed to reproduce the results (parallel and ncores are optional for bootstrapping). This will take some time to run for bootstrapping. The output will have all bootstrap or Monte Carlo estimates stored. This output, whether it is from indirect_effect(), cond_indirect_effects(), or cond_indirect(), can be reused by any of these three functions by setting boot_out (for bootstrapping) or mc_out (for Monte Carlo simulation) to this output. They will form the confidence intervals using the stored bootstrap or Monte Carlo estimates.
2. Do bootstrapping using do_boot() or Monte Carlo simulation us8ing do_mc(). The output can be used in the boot_out (for bootstrapping) or mc_out (for Monte Carlo simulation) argument of indirect_effect(), cond_indirect_effects() and cond_indirect().
3. For bootstrapping, if lavaan::sem() is used to fit a model and se = "boot" is used, do_boot() can extract them to generate a boot_out-class object that again can be used in the boot_out argument.

If boot_out or mc_out is set, arguments such as R, seed, and parallel will be ignored.

Value

indirect_effect() and cond_indirect() return an indirect-class object.

cond_indirect_effects() returns a cond_indirect_effects-class object.

These two classes of objects have their own print methods for printing the results (see print.indirect() and print.cond_indirect_effects()). They also have a coef method for extracting the estimates (coef.indirect() and coef.cond_indirect_effects()) and a confint method for extracting the confidence intervals (confint.indirect() and confint.cond_indirect_effects()). Addition and subtraction can also be conducted on indirect-class object to estimate and test a function of effects (see math_indirect)

Functions

- cond_indirect(): Compute conditional, indirect, or conditional indirect effects for one set of levels.
- cond_indirect_effects(): Compute the conditional effects or conditional indirect effects for several sets of levels of the moderator(s).

- `indirect_effect()`: Compute the indirect effect. A wrapper of `cond_indirect()`. Can be used when there is no moderator.
- `many_indirect_effects()`: Compute the indirect effects along more than one paths. It call `indirect_effect()` once for each of the path.

See Also

`mod_levels()` and `merge_mod_levels()` for generating levels of moderators. do_boot for doing bootstrapping before calling these functions.

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ a1 * x  + d1 * w1 + e1 * x:w1
m2 ~ a2 * x
y  ~ b1 * m1 + b2 * m2 + cp * x
"
fit <- sem(mod, dat, meanstructure = TRUE, fixed.x = FALSE, se = "none", baseline = FALSE)
est <- parameterEstimates(fit)
hi_w1 <- mean(dat$w1) + sd(dat$w1)

# Examples for cond_indirect():

# Conditional effect from x to m1 when w1 is 1 SD above mean
cond_indirect(x = "x", y = "m1",
              wvalues = c(w1 = hi_w1), fit = fit)

# Indirect effect from x1 through m2 to y
indirect_effect(x = "x", y = "y", fit = fit)

# Conditional Indirect effect from x1 through m1 to y, when w1 is 1 SD above mean
cond_indirect(x = "x", y = "y", m = "m1",
              wvalues = c(w1 = hi_w1), fit = fit)



# Examples for cond_indirect_effects():

# Create levels of w1, the moderators
w1levels <- mod_levels("w1", fit = fit)
w1levels

# Conditional effects from x to m1 when w1 is equal to each of the levels
cond_indirect_effects(x = "x", y = "m1",
                      wlevels = w1levels, fit = fit)

# Conditional Indirect effect from x1 through m1 to y,
# when w1 is equal to each of the levels
```

```
cond_indirect_effects(x = "x", y = "y", m = "m1",
                      wlevels = w1levels, fit = fit)


# Examples for many_indirect_effects():

library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)

# All indirect paths from x to y
paths <- all_indirect_paths(fit,
                            x = "x",
                            y = "y")
paths

# Indirect effect estimates
out <- many_indirect_effects(paths,
                             fit = fit)
out
```

`cond_indirect_diff` *Differences In Conditional Indirect Effects*

Description

Compute the difference in conditional indirect effects between two sets of levels of the moderators.

Usage

```
cond_indirect_diff(output, from = NULL, to = NULL, level = 0.95)
```

Arguments

`output`	A `cond_indirect_effects`-class object: The output of `cond_indirect_effects()`.
`from`	A row number of `output`.
`to`	A row number of `output`. The change in indirect effects is computed by the change in the level(s) of the moderator(s) from Row `from` to Row `to`.
`level`	The level of confidence for the confidence interval. Default is .95.

Details

Ths function takes the output of cond_indirect_effects() and computes the difference in conditional indirect effects between any two rows, that is, between levels of the moderator, or two sets of levels of the moderators when the path has more than one moderator.

The difference is meaningful when the difference between the two levels or sets of levels are meaningful. For example, if the two levels are the mean of the moderator and one standard deviation above mean of the moderator, then this difference is the change in indirect effect when the moderator increases by one standard deviation.

If the two levels are 0 and 1, then this difference is the index of moderated mediation as proposed by Hayes (2015). (This index can also be computed directly by index_of_mome(), designed specifically for this purpose.)

The function can also compute the change in the standardized indirect effect between two levels of a moderator or two sets of levels of the moderators.

This function is intended to be a general purpose function that allows users to compute the difference between any two levels or sets of levels that are meaningful in a context.

This function itself does not set the levels of comparison. The levels to be compared need to be set when calling cond_indirect_effects(). This function extracts required information from the output of cond_indirect_effects().

If bootstrap or Monte Carlo estimates are available in the input or bootstrap or Monte Carlo confidence intervals are requested in calling cond_indirect_effects(), cond_indirect_diff() will also form the percentile confidence interval for the difference in conditional indirect effects using the stored estimates.

Value

A cond_indirect_diff-class object. This class has a print method (print.cond_indirect_diff()), a coef method (coef.cond_indirect_diff()), and a confint method (confint.cond_indirect_diff()).

Functions

- cond_indirect_diff(): Compute the difference in in conditional indirect effect between two rows in the output of cond_indirect_effects().

References

Hayes, A. F. (2015). An index and test of linear moderated mediation. *Multivariate Behavioral Research, 50*(1), 1-22. doi:10.1080/00273171.2014.962683

See Also

index_of_mome() for computing the index of moderated mediation, index_of_momome() for computing the index of moderated moderated mediation, cond_indirect_effects(), mod_levels(), and merge_mod_levels() for preparing the levels to be compared.

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
dat$xw1 <- dat$x * dat$w1
mod <-
"
m1 ~ a * x  + f * w1 + d * xw1
y  ~ b * m1 + cp * x
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

# Create levels of w1, the moderators
w1levels <- mod_levels("w1", fit = fit)
w1levels

# Conditional effects from x to y when w1 is equal to each of the levels
boot_out <- fit2boot_out_do_boot(fit, R = 40, seed = 4314, progress = FALSE)
out <- cond_indirect_effects(x = "x", y = "y", m = "m1",
                             wlevels = w1levels, fit = fit,
                             boot_ci = TRUE, boot_out = boot_out)
out
out_ind <- cond_indirect_diff(out, from = 2, to = 1)
out_ind
coef(out_ind)
confint(out_ind)
```

---

confint.cond_indirect_diff

Confidence Interval of the Output of 'cond_indirect_diff()'

Description

Extract the confidence interval the output of `cond_indirect_diff()`.

Usage

```
## S3 method for class 'cond_indirect_diff'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	The output of cond_indirect_diff().
parm	Ignored.
level	The level of confidence for the confidence interval. Default is .95. Must match the level of the stored confidence interval.
...	Optional arguments. Ignored.

Details

The confint method of the cond_indirect_diff-class object.

The type of confidence intervals depends on the call used to create the object. This function merely extracts the stored confidence intervals.

Value

A one-row-two-column data frame of the confidence limits. If confidence interval is not available, the limits are NAs.

---

confint.cond_indirect_effects

Confidence Intervals of Indirect Effects or Conditional Indirect Effects

---

Description

Return the confidence intervals of the conditional indirect effects or conditional effects in the output of cond_indirect_effects().

Usage

```
## S3 method for class 'cond_indirect_effects'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	The output of cond_indirect_effects().
parm	Ignored. Always returns the confidence intervals of the effects for all levels stored.
level	The level of confidence, default is .95, returning the 95% confidence interval. Ignored for now and will use the level of the stored intervals.
...	Additional arguments. Ignored by the function.

Details

It extracts and returns the columns for confidence intervals, if available.

The type of confidence intervals depends on the call used to compute the effects. This function merely retrieves the confidence intervals stored, if any, which could be formed by nonparametric bootstrapping, Monte Carlo simulation, or other methods to be supported in the future.

Value

A data frame with two columns, one for each confidence limit of the confidence intervals. The number of rows is equal to the number of rows of `object`.

See Also

`cond_indirect_effects()`

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ x  + w1 + x:w1
m2 ~ m1
y  ~ m2 + x + w4 + m2:w4
"
fit <- sem(mod, dat, meanstructure = TRUE, fixed.x = FALSE, se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

# Examples for cond_indirect():

# Create levels of w1 and w4
w1levels <- mod_levels("w1", fit = fit)
w1levels
w4levels <- mod_levels("w4", fit = fit)
w4levels
w1w4levels <- merge_mod_levels(w1levels, w4levels)

# Conditional effects from x to m1 when w1 is equal to each of the levels
# R should be at least 2000 or 5000 in real research.
out1 <- suppressWarnings(cond_indirect_effects(x = "x", y = "m1",
                      wlevels = w1levels, fit = fit,
                      boot_ci = TRUE, R = 20, seed = 54151,
                      parallel = FALSE,
                      progress = FALSE))
confint(out1)
```

---

confint.delta_med *Confidence Interval for Delta_Med in a 'delta_med'-Class Object*

Description

Return the confidence interval of the Delta_Med in the output of `delta_med()`.

Usage

```
## S3 method for class 'delta_med'
confint(object, parm, level = NULL, ...)
```

Arguments

object	The output of delta_med().
parm	Not used because only one parameter, the Delta_Med, is allowed.
level	The level of confidence, default is NULL and the level used when the object was created will be used.
...	Optional arguments. Ignored.

Details

It returns the nonparametric bootstrap percentile confidence interval of Delta_Med, proposed byLiu, Yuan, and Li (2023). The object must be the output of delta_med(), with bootstrap confidence interval requested when calling delta_med(). However, the level of confidence can be different from that used when call delta_med().

Value

A one-row matrix of the confidence interval. All values are NA if bootstrap confidence interval was not requested when calling delta_med().

Author(s)

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

See Also

delta_med()

Examples

```
library(lavaan)
dat <- data_med
mod <-
"
m ~ x
y ~ m + x
"
fit <- sem(mod, dat)

# Call do_boot() to generate
# bootstrap estimates
# Use 2000 or even 5000 for R in real studies
# Set parallel to TRUE in real studies for faster bootstrapping
boot_out <- do_boot(fit,
                    R = 45,
```

```
                    seed = 879,
                    parallel = FALSE,
                    progress = FALSE)
# Remove 'progress = FALSE' in practice
dm_boot <- delta_med(x = "x",
                     y = "y",
                     m = "m",
                     fit = fit,
                     boot_out = boot_out,
                     progress = FALSE)
dm_boot
confint(dm_boot)
```

`confint.indirect` *Confidence Interval of Indirect Effect or Conditional Indirect Effect*

Description

Return the confidence interval of the indirect effect or conditional indirect effect stored in the output of `indirect_effect()` or `cond_indirect()`.

Usage

```
## S3 method for class 'indirect'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	The output of `indirect_effect()` or `cond_indirect()`.
`parm`	Ignored because the stored object always has only one parameter.
`level`	The level of confidence, default is .95, returning the 95% confidence interval.
`...`	Additional arguments. Ignored by the function.

Details

It extracts and returns the stored confidence interval if available.

The type of confidence interval depends on the call used to compute the effect. This function merely retrieves the stored estimates, which could be generated by nonparametric bootstrapping, Monte Carlo simulation, or other methods to be supported in the future, and uses them to form the percentile confidence interval.

Value

A numeric vector of two elements, the limits of the confidence interval.

See Also

indirect_effect() and cond_indirect()

Examples

```
dat <- modmed_x1m3w4y1

# Indirect Effect

library(lavaan)
mod1 <-
"
m1 ~ x
m2 ~ m1
y  ~ m2 + x
"
fit <- sem(mod1, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
# R should be at least 2000 or 5000 in real research.
out1 <- indirect_effect(x = "x", y = "y",
                        m = c("m1", "m2"),
                        fit = fit,
                        boot_ci = TRUE, R = 45, seed = 54151,
                        parallel = FALSE,
                        progress = FALSE)
out1
confint(out1)
```

---

confint.indirect_list *Confidence Intervals of Indirect Effects in an 'indirect_list' Object*

Description

Return the confidence intervals of the indirect effects stored in the output of many_indirect_effects().

Usage

```
## S3 method for class 'indirect_list'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

object	The output of many_indirect_effects().
parm	Ignored for now.
level	The level of confidence, default is .95, returning the 95% confidence interval.
...	Additional arguments. Ignored by the function.

Details

It extracts and returns the stored confidence interval if available.

The type of confidence intervals depends on the call used to compute the effects. This function merely retrieves the stored estimates, which could be generated by nonparametric bootstrapping, Monte Carlo simulation, or other methods to be supported in the future, and uses them to form the percentile confidence interval.

Value

A two-column data frame. The columns are the limits of the confidence intervals.

See Also

many_indirect_effects()

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)
# All indirect paths from x to y
paths <- all_indirect_paths(fit,
                            x = "x",
                            y = "y")
paths
# Indirect effect estimates
# R should be 2000 or even 5000 in real research
# parallel should be used in real research.
fit_boot <- do_boot(fit, R = 45, seed = 8974,
                    parallel = FALSE,
                    progress = FALSE)
out <- many_indirect_effects(paths,
                             fit = fit,
                             boot_ci = TRUE,
                             boot_out = fit_boot)
out
confint(out)
```

data_med *Sample Dataset: Simple Mediation*

Description

A simple mediation model.

Usage

```
data_med
```

Format

A data frame with 100 rows and 5 variables:

x Predictor. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med)
mod <-
"
m ~ a * x + c1 + c2
y ~ b * m + x + c1 + c2
ab := a * b
"
fit <- sem(mod, data_med, fixed.x = FALSE)
parameterEstimates(fit)
```

data_med_complicated *Sample Dataset: A Complicated Mediation Model*

Description

A mediation model with two predictors, two pathways,

Usage

```
data_med_complicated
```

Format

A data frame with 300 rows and 5 variables:

x1 Predictor 1. Numeric.

x2 Predictor 2. Numeric.

m11 Mediator 1 in Path 1. Numeric.

m12 Mediator 2 in Path 1. Numeric.

m2 Mediator in Path 2. Numeric.

y1 Outcome variable 1. Numeric.

y2 Outcome variable 2. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
data(data_med_complicated)
dat <- data_med_complicated
summary(lm_m11 <- lm(m11 ~ x1 + x1 + x2 + c1 + c2, dat))
summary(lm_m12 <- lm(m12 ~ m11 + x1 + x2 + c1 + c2, dat))
summary(lm_m2 <- lm(m2 ~ x1 + x2 + c1 + c2, dat))
summary(lm_y1 <- lm(y1 ~ m11 + m12 + m2 + x1 + x2 + c1 + c2, dat))
summary(lm_y2 <- lm(y2 ~ m11 + m12 + m2 + x1 + x2 + c1 + c2, dat))
```

---

data_med_mod_a *Sample Dataset: Simple Mediation with a-Path Moderated*

Description

A simple mediation model with a-path moderated.

Usage

```
data_med_mod_a
```

Format

A data frame with 100 rows and 6 variables:

x Predictor. Numeric.

w Moderator. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_a)
data_med_mod_a$xw <-
 data_med_mod_a$x *
 data_med_mod_a$w
mod <-
"
m ~ a * x + w + d * xw + c1 + c2
y ~ b * m + x + w + c1 + c2
w ~~ v_w * w
w ~ m_w * 1
ab := a * b
ab_lo := (a + d * (m_w - sqrt(v_w))) * b
ab_hi := (a + d * (m_w + sqrt(v_w))) * b
"
fit <- sem(mod, data_med_mod_a,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 11, 12, 31:33), ]
```

---

data_med_mod_ab *Sample Dataset: Simple Mediation with Both Paths Moderated (Two Moderators)*

Description

A simple mediation model with a-path and b-path each moderated by a moderator.

Usage

```
data_med_mod_ab
```

Format

A data frame with 100 rows and 7 variables:

x Predictor. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 2. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_ab)
data_med_mod_ab$xw1 <-
 data_med_mod_ab$x *
 data_med_mod_ab$w1
data_med_mod_ab$mw2 <-
 data_med_mod_ab$m *
 data_med_mod_ab$w2
mod <-
"
m ~ a * x + w1 + d1 * xw1 + c1 + c2
y ~ b * m + x + w1 + w2 + d2 * mw2 + c1 + c2
w1 ~~ v_w1 * w1
w1 ~ m_w1 * 1
w2 ~~ v_w2 * w2
w2 ~ m_w2 * 1
ab := a * b
ab_lolo := (a + d1 * (m_w1 - sqrt(v_w1))) * (b + d2 * (m_w2 - sqrt(v_w2)))
ab_lohi := (a + d1 * (m_w1 - sqrt(v_w1))) * (b + d2 * (m_w2 + sqrt(v_w2)))
ab_hilo := (a + d1 * (m_w1 + sqrt(v_w1))) * (b + d2 * (m_w2 - sqrt(v_w2)))
ab_hihi := (a + d1 * (m_w1 + sqrt(v_w1))) * (b + d2 * (m_w2 + sqrt(v_w2)))
"
fit <- sem(mod, data_med_mod_ab,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 10, 41:45), ]
```

---

data_med_mod_ab1 *Sample Dataset: Simple Mediation with Both Paths Moderated By a Moderator*

Description

A simple mediation model with a-path and b-path moderated by one moderator.

Usage

```
data_med_mod_ab1
```

Format

A data frame with 100 rows and 6 variables:

x Predictor. Numeric.

w Moderator. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_ab1)
data_med_mod_ab1$xw <-
 data_med_mod_ab1$x *
 data_med_mod_ab1$w
data_med_mod_ab1$mw <-
 data_med_mod_ab1$m *
 data_med_mod_ab1$w
mod <-
"
m ~ a * x + w + da * xw + c1 + c2
y ~ b * m + x + w + db * mw + c1 + c2
w ~~ v_w * w
w ~ m_w * 1
ab := a * b
ab_lo := (a + da * (m_w - sqrt(v_w))) * (b + db * (m_w - sqrt(v_w)))
ab_hi := (a + da * (m_w + sqrt(v_w))) * (b + db * (m_w + sqrt(v_w)))
"
fit <- sem(mod, data_med_mod_ab1,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 9, 38:40), ]
```

---

data_med_mod_b *Sample Dataset: Simple Mediation with b-Path Moderated*

Description

A simple mediation model with b-path moderated.

Usage

```
data_med_mod_b
```

Format

A data frame with 100 rows and 6 variables:

x Predictor. Numeric.

w Moderator. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_b)
data_med_mod_b$mw <-
 data_med_mod_b$m *
 data_med_mod_b$w
mod <-
"
m ~ a * x + w + c1 + c2
y ~ b * m + x + d * mw + c1 + c2
w ~~ v_w * w
w ~ m_w * 1
ab := a * b
ab_lo := a * (b + d * (m_w - sqrt(v_w)))
ab_hi := a * (b + d * (m_w + sqrt(v_w)))
"
fit <- sem(mod, data_med_mod_b,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 5, 7, 10, 11, 30:32), ]
```

---

data_med_mod_b_mod *Sample Dataset: A Simple Mediation Model with b-Path Moderated-Moderation*

Description

A simple mediation model with moderated-mediation on the b-path.

Usage

```
data_med_mod_b_mod
```

Format

A data frame with 100 rows and 5 variables:

x Predictor. Numeric.

w1 Moderator on b-path. Numeric.

w2 Moderator on the moderating effect of w1. Numeric.

m Mediator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
data(data_med_mod_b_mod)
dat <- data_med_mod_b_mod
summary(lm_m <- lm(m ~ x + c1 + c2, dat))
summary(lm_y <- lm(y ~ m*w1*w2 + x + c1 + c2, dat))
```

---

data_med_mod_parallel *Sample Dataset: Parallel Mediation with Two Moderators*

Description

A parallel mediation model with a1-path and b2-path moderated.

Usage

```
data_med_mod_parallel
```

Format

A data frame with 100 rows and 8 variables:

x Predictor. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 2. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_parallel)
data_med_mod_parallel$xw1 <-
 data_med_mod_parallel$x *
 data_med_mod_parallel$w1
data_med_mod_parallel$m2w2 <-
 data_med_mod_parallel$m2 *
 data_med_mod_parallel$w2
mod <-
"
m1 ~ a1 * x + w1 + da1 * xw1 + c1 + c2
m2 ~ a2 * x + w1 + c1 + c2
y ~ b1 * m1 + b2 * m2 + x + w1 + w2 + db2 * m2w2 + c1 + c2
w1 ~~ v_w1 * w1
w1 ~ m_w1 * 1
```

```
w2 ~~ v_w2 * w2
w2 ~ m_w2 * 1
a1b1 := a1 * b1
a2b2 := a2 * b2
a1b1_w1lo := (a1 + da1 * (m_w1 - sqrt(v_w1))) * b1
a1b1_w1hi := (a1 + da1 * (m_w1 + sqrt(v_w1))) * b2
a2b2_w2lo := a2 * (b2 + db2 * (m_w2 - sqrt(v_w2)))
a2b2_w2hi := a2 * (b2 + db2 * (m_w2 + sqrt(v_w2)))
"
fit <- sem(mod, data_med_mod_parallel,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 10, 11, 15, 48:53), ]
```

`data_med_mod_parallel_cat` *Sample Dataset: Parallel Moderated Mediation with Two Categorical Moderators*

Description

A parallel mediation model with two categorical moderators.

Usage

```
data_med_mod_parallel_cat
```

Format

A data frame with 300 rows and 8 variables:

x Predictor. Numeric.

w1 Moderator. String. Values: "group1", "group2", "group3"

w2 Moderator. String. Values: "team1", "team2"

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
data(data_med_mod_parallel_cat)
dat <- data_med_mod_parallel_cat
summary(lm_m1 <- lm(m1 ~ x*w1 + c1 + c2, dat))
summary(lm_m2 <- lm(m2 ~ x*w1 + c1 + c2, dat))
summary(lm_y <- lm(y ~ m1*w2 + m2*w2 + m1 + x + w1 + c1 + c2, dat))
```

`data_med_mod_serial` *Sample Dataset: Serial Mediation with Two Moderators*

Description

A simple mediation model with a-path and b2-path moderated.

Usage

```
data_med_mod_serial
```

Format

A data frame with 100 rows and 8 variables:

x Predictor. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 2. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_serial)
data_med_mod_serial$xw1 <-
 data_med_mod_serial$x *
 data_med_mod_serial$w1
data_med_mod_serial$m2w2 <-
 data_med_mod_serial$m2 *
 data_med_mod_serial$w2
mod <-
"
m1 ~ a * x + w1 + da1 * xw1 + c1 + c2
m2 ~ b1 * m1 + x + w1 + c1 + c2
y ~ b2 * m2 + m1 + x + w1 + w2 + db2 * m2w2 + c1 + c2
w1 ~~ v_w1 * w1
w1 ~ m_w1 * 1
w2 ~~ v_w2 * w2
w2 ~ m_w2 * 1
ab1b2 := a * b1 * b2
ab1b2_lolo := (a + da1 * (m_w1 - sqrt(v_w1))) * b1 * (b2 + db2 * (m_w2 - sqrt(v_w2)))
ab1b2_lohi := (a + da1 * (m_w1 - sqrt(v_w1))) * b1 * (b2 + db2 * (m_w2 + sqrt(v_w2)))
ab1b2_hilo := (a + da1 * (m_w1 + sqrt(v_w1))) * b1 * (b2 + db2 * (m_w2 - sqrt(v_w2)))
```

```
ab1b2_hihi := (a + da1 * (m_w1 + sqrt(v_w1))) * b1 * (b2 + db2 * (m_w2 + sqrt(v_w2)))
"
fit <- sem(mod, data_med_mod_serial,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 11, 16, 49:53), ]
```

`data_med_mod_serial_cat` *Sample Dataset: Serial Moderated Mediation with Two Categorical Moderators*

Description

A serial mediation model with two categorical moderators.

Usage

```
data_med_mod_serial_cat
```

Format

A data frame with 300 rows and 8 variables:

x Predictor. Numeric.

w1 Moderator. String. Values: "group1", "group2", "group3"

w2 Moderator. String. Values: "team1", "team2"

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
data(data_med_mod_serial_cat)
dat <- data_med_mod_serial_cat
summary(lm_m1 <- lm(m1 ~ x*w1 + c1 + c2, dat))
summary(lm_m2 <- lm(m2 ~ m1 + x + w1 + c1 + c2, dat))
summary(lm_y <- lm(y ~ m2*w2 + m1 + x + w1 + c1 + c2, dat))
```

`data_med_mod_serial_parallel` *Sample Dataset: Serial-Parallel Mediation with Two Moderators*

Description

A serial-parallel mediation model with some paths moderated.

Usage

```
data_med_mod_serial_parallel
```

Format

A data frame with 100 rows and 9 variables:

x Predictor. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 2. Numeric.

m11 Mediator 1 in Path 1. Numeric.

m12 Mediator 2 in Path 2. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_med_mod_serial_parallel)
data_med_mod_serial_parallel$xw1 <-
 data_med_mod_serial_parallel$x *
 data_med_mod_serial_parallel$w1
data_med_mod_serial_parallel$m2w2 <-
 data_med_mod_serial_parallel$m2 *
 data_med_mod_serial_parallel$w2
mod <-
"
m11 ~ a1 * x + w1 + da11 * xw1 + c1 + c2
m12 ~ b11 * m11 + x + w1 + c1 + c2
m2 ~ a2 * x + c1 + c2
y ~ b12 * m12 + b2 * m2 + m11 + x + w1 + w2 + db2 * m2w2 + c1 + c2
w1 ~~ v_w1 * w1
w1 ~ m_w1 * 1
w2 ~~ v_w2 * w2
w2 ~ m_w2 * 1
```

```
a1b11b22 := a1 * b11 * b12
a2b2 := a2 * b2
ab := a1b11b22 + a2b2
a1b11b12_w1lo := (a1 + da11 * (m_w1 - sqrt(v_w1))) * b11 * b12
a1b11b12_w1hi := (a1 + da11 * (m_w1 + sqrt(v_w1))) * b11 * b12
a2b2_w2lo := a2 * (b2 + db2 * (m_w2 - sqrt(v_w2)))
a2b2_w2hi := a2 * (b2 + db2 * (m_w2 + sqrt(v_w2)))
"
fit <- sem(mod, data_med_mod_serial_parallel,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[parameterEstimates(fit)$label != "", ]
```

`data_med_mod_serial_parallel_cat` *Sample Dataset: Serial-Parallel Moderated Mediation with Two Categorical Moderators*

Description

A serial-parallel mediation model with two categorical moderators.

Usage

```
data_med_mod_serial_parallel_cat
```

Format

A data frame with 300 rows and 8 variables:

x Predictor. Numeric.

w1 Moderator. String. Values: "group1", "group2", "group3"

w2 Moderator. String. Values: "team1", "team2"

m11 Mediator 1 in Path 1. Numeric.

m12 Mediator 2 in Path 1. Numeric.

m2 Mediator in Path 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
data(data_med_mod_serial_parallel_cat)
dat <- data_med_mod_serial_parallel_cat
summary(lm_m11 <- lm(m11 ~ x*w1 + c1 + c2, dat))
summary(lm_m12 <- lm(m12 ~ m11 + x + w1 + c1 + c2, dat))
summary(lm_m2 <- lm(m2 ~ x + w1 + c1 + c2, dat))
summary(lm_y <- lm(y ~ m12 + m2*w2 + m12 + x + c1 + c2, dat))
```

data_mod *Sample Dataset: One Moderator*

Description

A one-moderator model.

Usage

```
data_mod
```

Format

A data frame with 100 rows and 5 variables:

x Predictor. Numeric.

w Moderator. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_mod)
data_mod$xw <- data_mod$x * data_mod$w
mod <-
"
y ~ a * x + w + d * xw + c1 + c2
w ~~ v_w * w
w ~ m_w * 1
a_lo := a + d * (m_w - sqrt(v_w))
a_hi := a + d * (m_w + sqrt(v_w))
"
fit <- sem(mod, data_mod, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 7, 24, 25), ]
```

data_mod2 *Sample Dataset: Two Moderators*

Description

A two-moderator model.

Usage

```
data_mod2
```

Format

A data frame with 100 rows and 6 variables:

x Predictor. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_mod2)
data_mod2$xw1 <- data_mod2$x * data_mod2$w1
data_mod2$xw2 <- data_mod2$x * data_mod2$w2
mod <-
"
y ~ a * x + w1 + w2 + d1 * xw1 + d2 * xw2 + c1 + c2
w1 ~~ v_w1 * w1
w1 ~ m_w1 * 1
w2 ~~ v_w2 * w2
w2 ~ m_w2 * 1
a_lolo := a + d1 * (m_w1 - sqrt(v_w1)) + d2 * (m_w2 - sqrt(v_w2))
a_lohi := a + d1 * (m_w1 - sqrt(v_w1)) + d2 * (m_w2 + sqrt(v_w2))
a_hilo := a + d1 * (m_w1 + sqrt(v_w1)) + d2 * (m_w2 - sqrt(v_w2))
a_hihi := a + d1 * (m_w1 + sqrt(v_w1)) + d2 * (m_w2 + sqrt(v_w2))
"
fit <- sem(mod, data_mod2, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 4, 5, 8:11, 34:37), ]
```

---

data_mod_cat *Sample Dataset: Moderation with One Categorical Moderator*

Description

A moderation model with a categorical moderator.

Usage

```
data_mod_cat
```

Format

A data frame with 300 rows and 5 variables:

x Predictor. Numeric.

w Moderator. String. Values: "group1", "group2", "group3"

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
data(data_mod_cat)
dat <- data_mod_cat
summary(lm_y <- lm(y ~ x*w + c1 + c2, dat))
```

---

data_mome_demo *Sample Dataset: A Complicated Moderated-Mediation Model*

Description

Generated from a complicated moderated-mediation model for demonstration.

Usage

```
data_mome_demo
```

Format

A data frame with 200 rows and 11 variables:

x1 Predictor 1. Numeric.

x2 Predictor 2. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

m3 Mediator 3. Numeric.

y1 Outcome Variable 1. Numeric.

y2 Outcome Variable 2. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 21. Numeric.

c1 Control Variable 1. Numeric.

c2 Control Variable 2. Numeric.

Details

The model:

```
# w1x1 <- x1 * w1
# w2m2 <- w2 * m2
m1 ~ x1 + w1 + w1x1 + x2 + c1 + c2
m2 ~ m1 + c1 + c2
m3 ~ x2 + x1 + c1 + c2
y1 ~ m2 + w2 + w2m2 + x1 + x2 + m3 + c1 + c2
y2 ~ m3 + x2 + x1 + m2 + c1 + c2
# Covariances excluded for brevity
```

---

data_mome_demo_missing

Sample Dataset: A Complicated Moderated-Mediation Model With Missing Data

---

Description

Generated from a complicated moderated-mediation model for demonstration, with missing data

Usage

```
data_mome_demo_missing
```

Format

A data frame with 200 rows and 11 variables:

x1 Predictor 1. Numeric.

x2 Predictor 2. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

m3 Mediator 3. Numeric.

y1 Outcome Variable 1. Numeric.

y2 Outcome Variable 2. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 21. Numeric.

c1 Control Variable 1. Numeric.

c2 Control Variable 2. Numeric.

Details

A copy of data_mome_demo with some randomly selected cells changed to NA. The number of cases with no missing data is 169.

The model:

```
# w1x1 <- x1 * w1
# w2m2 <- w2 * m2
m1 ~ x1 + w1 + w1x1 + x2 + c1 + c2
m2 ~ m1 + c1 + c2
m3 ~ x2 + x1 + c1 + c2
y1 ~ m2 + w2 + w2m2 + x1 + x2 + m3 + c1 + c2
y2 ~ m3 + x2 + x1 + m2 + c1 + c2
# Covariances excluded for brevity
```

---

data_parallel *Sample Dataset: Parallel Mediation*

Description

A parallel mediation model.

Usage

```
data_parallel
```

Format

A data frame with 100 rows and 6 variables:

x Predictor. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_parallel)
mod <-
"
m1 ~ a1 * x + c1 + c2
m2 ~ a2 * x + c1 + c2
y ~ b2 * m2 + b1 * m1 + x + c1 + c2
indirect1 := a1 * b1
```

```
indirect2 := a2 * b2
indirect := a1 * b1 + a2 * b2
"
fit <- sem(mod, data_parallel,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 4, 7, 8, 27:29), ]
```

`data_sem` *Sample Dataset: A Latent Variable Mediation Model With 4 Factors*

Description

This data set is for testing functions in a four-factor structural model.

Usage

```
data_sem
```

Format

A data frame with 200 rows and 14 variables:

x01 Indicator. Numeric.

x02 Indicator. Numeric.

x03 Indicator. Numeric.

x04 Indicator. Numeric.

x05 Indicator. Numeric.

x06 Indicator. Numeric.

x07 Indicator. Numeric.

x08 Indicator. Numeric.

x09 Indicator. Numeric.

x10 Indicator. Numeric.

x11 Indicator. Numeric.

x12 Indicator. Numeric.

x13 Indicator. Numeric.

x14 Indicator. Numeric.

Examples

```
data(data_sem)
dat <- data_med_mod_b_mod
mod <-
  'f1 =~ x01 + x02 + x03
   f2 =~ x04 + x05 + x06 + x07
   f3 =~ x08 + x09 + x10
   f4 =~ x11 + x12 + x13 + x14
   f3 ~  a1*f1 + a2*f2
   f4 ~  b1*f1 + b3*f3
   a1b3 := a1 * b3
   a2b3 := a2 * b3
  '
fit <- lavaan::sem(model = mod, data = data_sem)
summary(fit)
```

`data_serial` *Sample Dataset: Serial Mediation*

Description

A serial mediation model.

Usage

```
data_serial
```

Format

A data frame with 100 rows and 6 variables:

x Predictor. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_serial)
mod <-
"
m1 ~ a * x + c1 + c2
m2 ~ b1 * m1 + x + c1 + c2
```

```
y ~ b2 * m2 + m1 + x + c1 + c2
indirect := a * b1 * b2
"
fit <- sem(mod, data_serial,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 4, 8, 28), ]
```

`data_serial_parallel` *Sample Dataset: Serial-Parallel Mediation*

Description

A mediation model with both serial and parallel components.

Usage

```
data_serial_parallel
```

Format

A data frame with 100 rows and 7 variables:

x Predictor. Numeric.

m11 Mediator 1 in Path 1. Numeric.

m12 Mediator 2 in Path 1. Numeric.

m2 Mediator in Path 2. Numeric.

y Outcome variable. Numeric.

c1 Control variable. Numeric.

c2 Control variable. Numeric.

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ a11 * x + c1 + c2
m12 ~ b11 * m11 + x + c1 + c2
m2 ~ a2 * x + c1 + c2
y ~ b12 * m12 + b2 * m2 + m11 + x + c1 + c2
indirect1 := a11 * b11 * b12
indirect2 := a2 * b2
indirect := a11 * b11 * b12 + a2 * b2
"
fit <- sem(mod, data_serial_parallel,
           meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 4, 8, 11, 12, 34:36), ]
```

`data_serial_parallel_latent` *Sample Dataset: A Latent Mediation Model With Three Mediators*

Description

Generated from a 3-mediator mediation model among eight latent factors, `fx1`, `fx2`, `fm11`, `fm12`, `fy1`, and `fy2`, each has three indicators.

Usage

```
data_serial_parallel_latent
```

Format

A data frame with 500 rows and 21 variables:

x1 Indicator of `fx1`. Numeric.

x2 Indicator of `fx1`. Numeric.

x3 Indicator of `fx1`. Numeric.

x4 Indicator of `fx2`. Numeric.

x5 Indicator of `fx2`. Numeric.

x6 Indicator of `fx2`. Numeric.

m11a Indicator of `fm11`. Numeric.

m11b Indicator of `fm11`. Numeric.

m11c Indicator of `fm11`. Numeric.

m12a Indicator of `fm12`. Numeric.

m12b Indicator of `fm12`. Numeric.

m12c Indicator of `fm12`. Numeric.

m2a Indicator of `fm2`. Numeric.

m2b Indicator of `fm2`. Numeric.

m2c Indicator of `fm2`. Numeric.

y1 Indicator of `fy1`. Numeric.

y2 Indicator of `fy1`. Numeric.

y3 Indicator of `fy1`. Numeric.

y4 Indicator of `fy2`. Numeric.

y5 Indicator of `fy2`. Numeric.

y6 Indicator of `fy2`. Numeric.

Details

The model:

```
fx1 =~ x1 + x2 + x3
fx2 =~ x4 + x5 + x6
fm11 =~ m11a + m11b + m11c
fm12 =~ m12a + m12b + m12c
fm2  =~ m2a + m2b + m2c
fy1 =~ y1 + y2 + y3
fy2 =~ y3 + y4 + y5
fm11 ~ a1 * fx1
fm12 ~ b11 * fm11 + a2m * fx2
fm2 ~ a2 * fx2
fy1 ~ b12 * fm12 + b11y1 * fm11 + cp1 * fx1
fy2 ~ b2 * fm2 + cp2 * fx2
a1b11b12 := a1 * b11 * b12
a1b11y1 := a1 * b11y1
a2b2 := a2 * b2
a2mb12 := a2m * b12
```

---

delta_med *Delta_Med by Liu, Yuan, and Li (2023)*

Description

It computes the Delta_Med proposed by Liu, Yuan, and Li (2023), an R^2-like measure of indirect effect.

Usage

```
delta_med(
  x,
  y,
  m,
  fit,
  paths_to_remove = NULL,
  boot_out = NULL,
  level = 0.95,
  progress = TRUE,
  skip_check_single_x = FALSE,
  skip_check_m_between_x_y = FALSE,
  skip_check_x_to_y = FALSE,
  skip_check_latent_variables = FALSE
)
```

Arguments

x	The name of the x variable. Must be supplied as a quoted string.
y	The name of the y variable. Must be supplied as a quoted string.
m	A vector of the variable names of the mediator(s). If more than one mediators, they do not have to be on the same path from x to y. Cannot be NULL for this function.
fit	The fit object. Must be a lavaan::lavaan object.
paths_to_remove	A character vector of paths users want to manually remove, specified in lavaan model syntax. For example, c("m2~x", "m3~m2") removes the path from x to m2 and the path from m2 to m3. The default is NULL, and the paths to remove will be determined using the method by Liu et al. (2023). If supplied, then only paths specified explicitly will be removed.
boot_out	The output of do_boot(). If supplied, the stored bootstrap estimates will be used to form the nonparametric percentile bootstrap confidence interval of Delta_Med.
level	The level of confidence of the bootstrap confidence interval. Default is .95.
progress	Logical. Display bootstrapping progress or not. Default is TRUE.
skip_check_single_x	Logical Check whether the model has one and only one x-variable. Default is TRUE.
skip_check_m_between_x_y	Logical. Check whether all m variables are along a path from x to y. Default is TRUE.
skip_check_x_to_y	Logical. Check whether there is a direct path from x to y. Default is TRUE.
skip_check_latent_variables	Logical. Check whether the model has any latent variables. Default is TRUE.

Details

It computes Delta_Med, an R^2-like effect size measure for the indirect effect from one variable (the y-variable) to another variable (the x-variable) through one or more mediators (m, or m1, m2, etc. when there are more than one mediator).

The Delta_Med of one or more mediators was computed as the difference between two R^2s:

- R_1^2, the R^2 when y is predicted by x and all mediators.
- R_2^2, the R^2 when the mediator(s) of interest is/are removed from the models, while the error term(s) of the mediator(s) is/are kept.

Delta_Med is given by $R_1^2 - R_2^2$.

Please refer to Liu et al. (2023) for the technical details.

The function can also form a nonparametric percentile bootstrap confidence of Delta_Med.

Value

A `delta_med` class object. It is a list-like object with these major elements:

- `delta_med`: The Delta_Med.
- `x`: The name of the `x`-variable.
- `y`: The name of the `y`-variable.
- `m`: A character vector of the mediator(s) along a path. The path runs from the first element to the last element.

This class has a `print` method, a `coef` method, and a `confint` method. See `print.delta_med()`, `coef.delta_med()`, and `confint.delta_med()`.

Implementation

The function identifies all the path(s) pointing to the mediator(s) of concern and fixes the path(s) to zero, effectively removing the mediator(s). However, the model is not refitted, hence keeping the estimates of all other parameters unchanged. It then uses `lavaan::lav_model_set_parameters()` to update the parameters, `lavaan::lav_model_implied()` to update the implied statistics, and then calls `lavaan::lavInspect()` to retrieve the implied variance of the predicted values of `y` for computing the R_2^2. Subtracting this R_2^2 from R_1^2 of `y` can then yield Delta_Med.

Model Requirements

For now, by defaul, it only computes Delta_Med for the types of models discussed in Liu et al. (2023):

- Having one predictor (the `x`-variable).
- Having one or more mediators, the `m`-variables, with arbitrary way to mediate the effect of `x` on the outcome variable (`y`-variable).
- Having one or more outcome variables. Although their models only have outcome variables, the computation of the Delta_Med is not affected by the presence of other outcome variables.
- Having no control variables.
- The mediator(s), `m`, and the `y`-variable are continuous.
- `x` can be continuous or categorical. If categorical, it needs to be handle appropriately when fitting the model.
- `x` has a direct path to `y`.
- All the mediators listed in the argument `m` is present in at least one path from `x` to `y`.
- None of the paths from `x` to `y` are moderated.

It can be used for other kinds of models but support for them is disabled by default. To use this function for cases not discussed in Liu et al. (2023), please disable relevant requirements stated above using the relevant `skip_check_*` arguments. An error will be raised if the models failed any of the checks not skipped by users.

References

Liu, H., Yuan, K.-H., & Li, H. (2023). A systematic framework for defining R-squared measures in mediation analysis. *Psychological Methods*. Advance online publication. https://doi.org/10.1037/met0000571

See Also

print.delta_med(), coef.delta_med(), and confint.delta_med().

Examples

```
library(lavaan)
dat <- data_med
mod <-
"
m ~ x
y ~ m + x
"
fit <- sem(mod, dat)
dm <- delta_med(x = "x",
                y = "y",
                m = "m",
                fit = fit)
dm
print(dm, full = TRUE)

# Call do_boot() to generate
# bootstrap estimates
# Use 2000 or even 5000 for R in real studies
# Set parallel to TRUE in real studies for faster bootstrapping
boot_out <- do_boot(fit,
                    R = 45,
                    seed = 879,
                    parallel = FALSE,
                    progress = FALSE)
# Remove 'progress = FALSE' in practice
dm_boot <- delta_med(x = "x",
                     y = "y",
                     m = "m",
                     fit = fit,
                     boot_out = boot_out,
                     progress = FALSE)
dm_boot
confint(dm_boot)
```

---

do_boot *Bootstrap Estimates for 'indirect_effects' and 'cond_indirect_effects'*

Description

Generate bootstrap estimates to be used by cond_indirect_effects(), indirect_effect(), and cond_indirect(),

Usage

```
do_boot(
  fit,
  R = 100,
  seed = NULL,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE
)
```

Arguments

fit	Either (a) a list of lm class objects, or the output of lm2list() (i.e., an lm_list-class object), or (b) the output of lavaan::sem().
R	The number of bootstrap samples. Default is 100.
seed	The seed for the bootstrapping. Default is NULL and seed is not set.
parallel	Logical. Whether parallel processing will be used. Default is TRUE.
ncores	Integer. The number of CPU cores to use when parallel is TRUE. Default is the number of non-logical cores minus one (one minimum). Will raise an error if greater than the number of cores detected by parallel::detectCores(). If ncores is set, it will override make_cluster_args.
make_cluster_args	A named list of additional arguments to be passed to parallel::makeCluster(). For advanced users. See parallel::makeCluster() for details. Default is list(), no additional arguments.
progress	Logical. Display progress or not. Default is TRUE.

Details

It does nonparametric bootstrapping to generate bootstrap estimates of the parameter estimates in a model fitted either by lavaan::sem() or by a sequence of calls to lm(). The stored estimates can then be used by cond_indirect_effects(), indirect_effect(), and cond_indirect() to form bootstrapping confidence intervals.

This approach removes the need to repeat bootstrapping in each call to cond_indirect_effects(), indirect_effect(), and cond_indirect(). It also ensures that the same set of bootstrap samples is used in all subsequent analysis.

It determines the type of the fit object automatically and then calls lm2boot_out(), fit2boot_out(), or fit2boot_out_do_boot().

Value

A boot_out-class object that can be used for the boot_out argument of cond_indirect_effects(), indirect_effect(), and cond_indirect() for forming bootstrap confidence intervals. The object is a list with the number of elements equal to the number of bootstrap samples. Each element is a list of the parameter estimates and sample variances and covariances of the variables in each bootstrap sample.

See Also

lm2boot_out(), fit2boot_out(), and fit2boot_out_do_boot(), which implements the bootstrapping.

Examples

```
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
lm_m <- lm(m ~ x*w + c1 + c2, dat)
lm_y <- lm(y ~ m*w + x + c1 + c2, dat)
lm_out <- lm2list(lm_m, lm_y)
# In real research, R should be 2000 or even 5000
# In real research, no need to set parallel and progress to FALSE
# Parallel processing is enabled by default and
# progress is displayed by default.
lm_boot_out <- do_boot(lm_out, R = 50, seed = 1234,
                       parallel = FALSE,
                       progress = FALSE)
wlevels <- mod_levels(w = "w", fit = lm_out)
wlevels
out <- cond_indirect_effects(wlevels = wlevels,
                             x = "x",
                             y = "y",
                             m = "m",
                             fit = lm_out,
                             boot_ci = TRUE,
                             boot_out = lm_boot_out)
out
```

---

do_mc *Monte Carlo Estimates for 'indirect_effects' and 'cond_indirect_effects'*

Description

Generate Monte Carlo estimates to be used by cond_indirect_effects(), indirect_effect(), and cond_indirect(),

Usage

```
do_mc(
  fit,
  R = 100,
  seed = NULL,
  parallel = TRUE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE
```

```
)

gen_mc_est(fit, R = 100, seed = NULL)
```

Arguments

fit	The output of lavaan::sem(). It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi(). The output of stats::lm() is not supported.
R	The number of replications. Default is 100.
seed	The seed for the generating Monte Carlo estimates. Default is NULL and seed is not set.
parallel	Not used. Kept for compatibility with do_boot().
ncores	Not used. Kept for compatibility with do_boot().
make_cluster_args	Not used. Kept for compatibility with do_boot().
progress	Logical. Display progress or not. Default is TRUE.

Details

It uses the parameter estimates and their variance-covariance matrix to generate Monte Carlo estimates of the parameter estimates in a model fitted by lavaan::sem(). The stored estimates can then be used by cond_indirect_effects(), indirect_effect(), and cond_indirect() to form Monte Carlo confidence intervals.

It also supports a model estimated by multiple imputation using semTools::runMI() or its wrapper, such as semTools::sem.mi(). The pooled estimates and their variance-covariance matrix will be used to generate the Monte Carlo estimates.

This approach removes the need to repeat Monte Carlo simulation in each call to cond_indirect_effects(), indirect_effect(), and cond_indirect(). It also ensures that the same set of Monte Carlo estimates is used in all subsequent analysis.

Value

A mc_out-class object that can be used for the mc_out argument of cond_indirect_effects(), indirect_effect(), and cond_indirect() for forming Monte Carlo confidence intervals. The object is a list with the number of elements equal to the number of Monte Carlo replications. Each element is a list of the parameter estimates and sample variances and covariances of the variables in each Monte Carlo replication.

Functions

- do_mc(): A general purpose function for creating Monte Carlo estimates to be reused by other functions. It returns a mc_out-class object.
- gen_mc_est(): Generate Monte Carlo estimates and store them in the external slot: external$manymome$mc. For advanced users.

See Also

fit2mc_out(), which implements the Monte Carlo simulation.

Examples

```
library(lavaan)
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
mod <-
"
m ~ x + w + x:w + c1 + c2
y ~ m + w + m:w + x + c1 + c2
"
fit <- sem(mod, dat)
# In real research, R should be 5000 or even 10000
mc_out <- do_mc(fit, R = 100, seed = 1234)
wlevels <- mod_levels(w = "w", fit = fit)
wlevels
out <- cond_indirect_effects(wlevels = wlevels,
                             x = "x",
                             y = "y",
                             m = "m",
                             fit = fit,
                             mc_ci = TRUE,
                             mc_out = mc_out)
out
```

---

factor2var *Create Dummy Variables*

Description

Create dummy variables from a categorical variable.

Usage

```
factor2var(
  x_value,
  x_contrasts = "contr.treatment",
  prefix = "",
  add_rownames = TRUE
)
```

Arguments

x_value	The vector of categorical variable.
x_contrasts	The contrast to be used. Default is "constr.treatment".
prefix	The prefix to be added to the variables to be created. Default is "".
add_rownames	Whether row names will be added to the output. Default is TRUE.

Details

Its main use is for creating dummy variables (indicator variables) from a categorical variable, to be used in lavaan::sem().

Optionally, the other contrasts can be used through the argument x_contrasts.

Value

It always returns a matrix with the number of rows equal to the length of the vector (x_value). If the categorical has only two categories and so only one dummy variable is needed, the output is still a one-column "matrix" in R.

Examples

```
dat <- data_mod_cat
dat <- data.frame(dat,
                  factor2var(dat$w, prefix = "gp", add_rownames = FALSE))
head(dat[, c("w", "gpgroup2", "gpgroup3")], 15)
```

---

fit2boot_out *Bootstrap Estimates for a* lavaan *Output*

Description

Generate bootstrap estimates from the output of lavaan::sem().

Usage

```
fit2boot_out(fit)

fit2boot_out_do_boot(
  fit,
  R = 100,
  seed = NULL,
  parallel = FALSE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE,
  internal = list()
)
```

Arguments

fit	The fit object. This function only supports a lavaan::lavaan object.
R	The number of bootstrap samples. Default is 100.
seed	The seed for the random resampling. Default is NULL.
parallel	Logical. Whether parallel processing will be used. Default is NULL.
ncores	Integer. The number of CPU cores to use when parallel is TRUE. Default is the number of non-logical cores minus one (one minimum). Will raise an error if greater than the number of cores detected by parallel::detectCores(). If ncores is set, it will override make_cluster_args.
make_cluster_args	A named list of additional arguments to be passed to parallel::makeCluster(). For advanced users. See parallel::makeCluster() for details. Default is list().
progress	Logical. Display progress or not. Default is TRUE.
internal	A list of arguments to be used internally for debugging. Default is list().

Details

This function is for advanced users. do_boot() is a function users should try first because do_boot() has a general interface for input-specific functions like this one.

If bootstrapping confidence intervals was requested when calling lavaan::sem() by setting se = "boot", fit2boot_out() can be used to extract the stored bootstrap estimates so that they can be reused by indirect_effect(), cond_indirect_effects() and related functions to form bootstrapping confidence intervals for effects such as indirect effects and conditional indirect effects.

If bootstrapping confidence was not requested when fitting the model by lavaan::sem(), fit2boot_out_do_boot() can be used to generate nonparametric bootstrap estimates from the output of lavaan::sem() and store them for use by indirect_effect(), cond_indirect_effects(), and related functions.

This approach removes the need to repeat bootstrapping in each call to indirect_effect(), cond_indirect_effects(), and related functions. It also ensures that the same set of bootstrap samples is used in all subsequent analyses.

Value

A boot_out-class object that can be used for the boot_out argument of indirect_effect(), cond_indirect_effects(), and related functions for forming bootstrapping confidence intervals.

The object is a list with the number of elements equal to the number of bootstrap samples. Each element is a list of the parameter estimates and sample variances and covariances of the variables in each bootstrap sample.

Functions

- fit2boot_out(): Process stored bootstrap estimates for functions such as cond_indirect_effects().
- fit2boot_out_do_boot(): Do bootstrapping and store information to be used by cond_indirect_effects() and related functions. Support parallel processing.

See Also

do_boot(), the general purpose function that users should try first before using this function.

Examples

```
library(lavaan)
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
dat$"x:w" <- dat$x * dat$w
dat$"m:w" <- dat$m * dat$w
mod <-
"
m ~ x + w + x:w + c1 + c2
y ~ m + w + m:w + x + c1 + c2
"

# Bootstrapping not requested in calling lavaan::sem()
fit <- sem(model = mod, data = dat, fixed.x = FALSE,
           se = "none", baseline = FALSE)
fit_boot_out <- fit2boot_out_do_boot(fit = fit,
                                     R = 40,
                                     seed = 1234,
                                     progress = FALSE)
out <- cond_indirect_effects(wlevels = "w",
                             x = "x",
                             y = "y",
                             m = "m",
                             fit = fit,
                             boot_ci = TRUE,
                             boot_out = fit_boot_out)
out
```

---

fit2mc_out *Monte Carlo Estimates for a* lavaan *Output*

Description

Generate Monte Carlo estimates from the output of lavaan::sem().

Usage

```
fit2mc_out(fit, progress = TRUE)
```

Arguments

fit	The fit object. This function only supports a lavaan::lavaan object. It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().

progress Logical. Display progress or not. Default is TRUE.

Details

This function is for advanced users. do_mc() is a function users should try first because do_mc() has a general interface for input-specific functions like this one.

fit2mc_out() can be used to extract the stored Monte Carlo estimates so that they can be reused by indirect_effect(), cond_indirect_effects() and related functions to form Monte Carlo confidence intervals for effects such as indirect effects and conditional indirect effects.

This approach removes the need to repeat Monte Carlo simulation in each call to indirect_effect(), cond_indirect_effects(), and related functions. It also ensures that the same set of Monte Carlo estimates is used in all subsequent analyses.

Value

A mc_out-class object that can be used for the mc_out argument of indirect_effect(), cond_indirect_effects(), and related functions for forming Monte Carlo confidence intervals.

The object is a list with the number of elements equal to the number of Monte Carlo replications. Each element is a list of the parameter estimates and sample variances and covariances of the variables in each Monte Carlo replication.

See Also

do_mc(), the general purpose function that users should try first before using this function.

Examples

```
library(lavaan)
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
dat$"x:w" <- dat$x * dat$w
dat$"m:w" <- dat$m * dat$w
mod <-
"
m ~ x + w + x:w + c1 + c2
y ~ m + w + m:w + x + c1 + c2
"

fit <- sem(model = mod, data = dat, fixed.x = FALSE,
           baseline = FALSE)
# In real research, R should be 5000 or even 10000.
fit <- gen_mc_est(fit, R = 100, seed = 453253)
fit_mc_out <- fit2mc_out(fit)
out <- cond_indirect_effects(wlevels = "w",
                             x = "x",
                             y = "y",
                             m = "m",
                             fit = fit,
                             mc_ci = TRUE,
```

```
                              mc_out = fit_mc_out)
out
```

`get_one_cond_indirect_effect` *Get The Conditional Indirect Effect for One Row of 'cond_indirect_effects' Output*

Description

Return the conditional indirect effect of one row of the output of `cond_indirect_effects()`.

Usage

```
get_one_cond_indirect_effect(object, row)

get_one_cond_effect(object, row)
```

Arguments

`object`	The output of `cond_indirect_effects()`.
`row`	The row number of the row to be retrieved.

Details

It just extracts the corresponding output of `cond_indirect()` from the requested row.

Value

An `indirect`-class object, similar to the output of `indirect_effect()` and `cond_indirect()`. See [indirect_effect)] and `cond_indirect()` for details on these classes.

[indirect_effect)]: R:indirect_effect) `cond_indirect()`: R:cond_indirect()

Functions

- `get_one_cond_effect()`: An alias to `get_one_cond_indirect_effect()`

See Also

cond_indirect_effects

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ x  + w1 + x:w1
m2 ~ m1
y  ~ m2 + x + w4 + m2:w4
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

# Examples for cond_indirect():

# Conditional effects from x to m1
# when w1 is equal to each of the default levels
out1 <- cond_indirect_effects(x = "x", y = "m1",
                              wlevels = c("w1", "w4"), fit = fit)
get_one_cond_indirect_effect(out1, 3)

# Conditional Indirect effect from x1 through m1 to y,
# when w1 is equal to each of the levels
out2 <- cond_indirect_effects(x = "x", y = "y", m = c("m1", "m2"),
                              wlevels = c("w1", "w4"), fit = fit)
get_one_cond_indirect_effect(out2, 4)
```

`get_prod` *Product Terms (if Any) Along a Path*

Description

Identify the product term(s), if any, along a path in a model and return the term(s), with the variables involved and the coefficient(s) of the term(s).

Usage

```
get_prod(
  x,
  y,
  operator = ":",
  fit = NULL,
  est = NULL,
  data = NULL,
  expand = FALSE
)
```

Arguments

x	Character. Variable name.
y	Character. Variable name.
operator	Character. The string used to indicate a product term. Default is ":", used in both lm() and lavaan::sem() for observed variables.
fit	The fit object. Currently only supports a lavaan::lavaan object. It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().
est	The output of lavaan::parameterEstimates(). If NULL, the default, it will be generated from fit. If supplied, fit will ge ignored.
data	Data frame (optional). If supplied, it will be used to identify the product terms.
expand	Whether products of more than two terms will be searched. FALSE by default.

Details

This function is used by several functions in manymome to identify product terms along a path. If possible, this is done by numerically checking whether a column in a dataset is the product of two other columns. If not possible (e.g., the "product term" is the "product" of two latent variables, formed by the products of indicators), then it requires the user to specify an operator.

The detailed workflow of this function can be found in the article https://sfcheung.github.io/manymome/articles/get_prod.html

This function is not intended to be used by users. It is exported such that advanced users or developers can use it.

Value

If at least one product term is found, it returns a list with these elements:

- prod: The names of the product terms found.
- b: The coefficients of these product terms.
- w: The variable, other than x, in each product term.
- x: The x-variable, that is, where the path starts.
- y: The y-variable, that is, where the path ends.

It returns NA if no product term is found along the path.

Examples

```
dat <- modmed_x1m3w4y1
library(lavaan)
mod <-
"
m1 ~ x   + w1 + x:w1
m2 ~ m1  + w2 + m1:w2
m3 ~ m2
```

```
y  ~ m3  + w4 + m3:w4 + x + w3 + x:w3 + x:w4
"
fit <- sem(model = mod,
           data = dat,
           meanstructure = TRUE,
           fixed.x = FALSE)

# One product term
get_prod(x = "x", y = "m1", fit = fit)
# Two product terms
get_prod(x = "x", y = "y", fit = fit)
# No product term
get_prod(x = "m2", y = "m3", fit = fit)
```

---

index_of_mome *Index of Moderated Mediation and Index of Moderated Moderated Mediation*

Description

It computes the index of moderated mediation and the index of moderated moderated mediation proposed by Hayes (2015, 2018).

Usage

```
index_of_mome(
  x,
  y,
  m = NULL,
  w = NULL,
  fit = NULL,
  boot_ci = FALSE,
  level = 0.95,
  boot_out = NULL,
  R = 100,
  seed = NULL,
  progress = TRUE,
  mc_ci = FALSE,
  mc_out = NULL,
  ci_type = NULL,
  ci_out = NULL,
  ...
)

index_of_momome(
  x,
  y,
```

```
  m = NULL,
  w = NULL,
  z = NULL,
  fit = NULL,
  boot_ci = FALSE,
  level = 0.95,
  boot_out = NULL,
  R = 100,
  seed = NULL,
  progress = TRUE,
  mc_ci = FALSE,
  mc_out = NULL,
  ci_type = NULL,
  ci_out = NULL,
  ...
)
```

Arguments

x	Character. The name of the predictor at the start of the path.
y	Character. The name of the outcome variable at the end of the path.
m	A vector of the variable names of the mediator(s). The path goes from the first mediator successively to the last mediator. If NULL, the default, the path goes from x to y.
w	Character. The name of the moderator.
fit	The fit object. Can be a lavaan::lavaan object, a list of lm() outputs, or an object created by lm2list(). It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().
boot_ci	Logical. Whether bootstrap confidence interval will be formed. Default is FALSE.
level	The level of confidence for the bootstrap confidence interval. Default is .95.
boot_out	If boot_ci is TRUE, users can supply pregenerated bootstrap estimates. This can be the output of do_boot(). For indirect_effect() and cond_indirect_effects(), this can be the output of a previous call to cond_indirect_effects(), indirect_effect(), or cond_indirect() with bootstrap confidence intervals requested. These stored estimates will be reused such that there is no need to do bootstrapping again. If not supplied, the function will try to generate them from fit.
R	Integer. If boot_ci is TRUE, boot_out is NULL, and bootstrap standard errors not requested if fit is a lavaan object, this function will do bootstrapping on fit. R is the number of bootstrap samples. Default is 100. For Monte Carlo simulation, this is the number of replications.
seed	If bootstrapping or Monte Carlo simulation is conducted, this is the seed for the bootstrapping or simulation. Default is NULL and seed is not set.
progress	Logical. Display bootstrapping progress or not. Default is TRUE.
mc_ci	Logical. Whether Monte Carlo confidence interval will be formed. Default is FALSE.

mc_out	If mc_ci is TRUE, users can supply pregenerated Monte Carlo estimates. This can be the output of do_mc(). For indirect_effect() and cond_indirect_effects(), this can be the output of a previous call to cond_indirect_effects(), indirect_effect(), or cond_indirect() with Monte Carlo confidence intervals requested. These stored estimates will be reused such that there is no need to do Monte Carlo simulation again. If not supplied, the function will try to generate them from fit.
ci_type	The type of confidence intervals to be formed. Can be either "boot" (bootstrapping) or "mc" (Monte Carlo). If not supplied or is NULL, will check other arguments (e.g, boot_ci and mc_ci). If supplied, will override boot_ci and mc_ci.
ci_out	If ci_type is supplied, this is the corresponding argument. If ci_type is "boot", this argument will be used as boot_out. If ci_type is "mc", this argument will be used as mc_out.
...	Arguments to be passed to cond_indirect_effects()
z	Character. The name of the second moderator, for computing the index of moderated moderated mediation.

Details

The function index_of_mome() computes the *index of moderated mediation* proposed by Hayes (2015). It supports any path in a model with one (and only one) component path moderated. For example, x->m1->m2->y with x->m1 moderated by w. It measures the change in indirect effect when the moderator increases by one unit.

The function index_of_momome() computes the *index of moderated moderated mediation* proposed by Hayes (2018). It supports any path in a model, with two component paths moderated, each by one moderator. For example, x->m1->m2->y with x->m1 moderated by w and m2->y moderated by z. It measures the change in the index of moderated mediation of one moderator when the other moderator increases by one unit.

Value

It returns a cond_indirect_diff-class object. This class has a print method (print.cond_indirect_diff()), a coef method for extracting the index (coef.cond_indirect_diff()), and a confint method for extracting the confidence interval if available (confint.cond_indirect_diff()).

Functions

- index_of_mome(): Compute the index of moderated mediation.
- index_of_momome(): Compute the index of moderated moderated mediation.

References

Hayes, A. F. (2015). An index and test of linear moderated mediation. *Multivariate Behavioral Research, 50*(1), 1-22. doi:10.1080/00273171.2014.962683

Hayes, A. F. (2018). Partial, conditional, and moderated moderated mediation: Quantification, inference, and interpretation. *Communication Monographs, 85*(1), 4-40. doi:10.1080/03637751.2017.1352100

See Also

cond_indirect_effects()

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
dat$xw1 <- dat$x * dat$w1
mod <-
"
m1 ~ a * x  + f * w1 + d * xw1
y  ~ b * m1 + cp * x
ind_mome := d * b
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

# R should be at least 2000 or even 5000 in real research.
# parallel is set to TRUE by default.
# Therefore, in research, the argument parallel can be omitted.
out_mome <- index_of_mome(x = "x", y = "y", m = "m1", w = "w1",
                          fit = fit,
                          boot_ci = TRUE,
                          R = 42,
                          seed = 4314,
                          parallel = FALSE,
                          progress = FALSE)
out_mome
coef(out_mome)
# From lavaan
print(est[19, ], nd = 8)
confint(out_mome)



library(lavaan)
dat <- modmed_x1m3w4y1
dat$xw1 <- dat$x * dat$w1
dat$m1w4 <- dat$m1 * dat$w4
mod <-
"
m1 ~ a * x  + f1 * w1 + d1 * xw1
y  ~ b * m1 + f4 * w4 + d4 * m1w4 + cp * x
ind_momome := d1 * d4
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)
```

```
# See the example of index_of_mome on how to request
# bootstrap confidence interval.
out_momome <- index_of_momome(x = "x", y = "y", m = "m1",
                              w = "w1", z = "w4",
                              fit = fit)
out_momome
coef(out_momome)
print(est[32, ], nd = 8)
```

`indirect_effects_from_list` *Coefficient Table of an 'indirect_list' Class Object*

Description

Create a coefficient table for the point estimates and confidence intervals (if available) in the output of `many_indirect_effects()`.

Usage

```
indirect_effects_from_list(object, add_sig = TRUE, pvalue = FALSE, se = FALSE)
```

Arguments

`object`	The output of `indirect_effect()` or `cond_indirect()`.
`add_sig`	Whether a column of significance test results will be added. Default is `TRUE`.
`pvalue`	Logical. If `TRUE`, asymmetric *p*-values based on bootstrapping will be added available. Default is `FALSE`.
`se`	Logical. If `TRUE` and confidence intervals are available, the standard errors of the estimates are also added. They are simply the standard deviations of the bootstrap estimates or Monte Carlo simulated values, depending on the method used to form the confidence intervals.

Details

If bootstrapping confidence interval was requested, this method has the option to add *p*-values computed by the method presented in Asparouhov and Muthén (2021). Note that these *p*-values is asymmetric bootstrap *p*-values based on the distribution of the bootstrap estimates. They are not computed based on the distribution under the null hypothesis.

For a *p*-value of *a*, it means that a 100(1 - *a*)% bootstrapping confidence interval will have one of its limits equal to 0. A confidence interval with a higher confidence level will include zero, while a confidence interval with a lower confidence level will exclude zero.

Value

A data frame with the indirect effect estimates and confidence intervals (if available). It also has A string column, "Sig", for #' significant test results if add_sig is TRUE and confidence intervals are available.

References

Asparouhov, A., & Muthén, B. (2021). Bootstrap p-value computation. Retrieved from https://www.statmodel.com/download Bootstrap%20-%20Pvalue.pdf

See Also

many_indirect_effects()

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)

# All indirect paths from x to y
paths <- all_indirect_paths(fit,
                            x = "x",
                            y = "y")
paths

# Indirect effect estimates
out <- many_indirect_effects(paths,
                             fit = fit)
out

# Create a data frame of the indirect effect estimates

out_df <- indirect_effects_from_list(out)
out_df
```

indirect_i *Indirect Effect (No Bootstrapping)*

Description

It computes an indirect effect, optionally conditional on the value(s) of moderator(s) if present.

Usage

```
indirect_i(
  x,
  y,
  m = NULL,
  fit = NULL,
  est = NULL,
  implied_stats = NULL,
  wvalues = NULL,
  standardized_x = FALSE,
  standardized_y = FALSE,
  computation_digits = 5,
  prods = NULL,
  get_prods_only = FALSE,
  data = NULL,
  expand = TRUE,
  warn = TRUE,
  allow_mixing_lav_and_obs = TRUE
)
```

Arguments

x	Character. The name of the predictor at the start of the path.
y	Character. The name of the outcome variable at the end of the path.
m	A vector of the variable names of the mediator(s). The path goes from the first mediator successively to the last mediator. If NULL, the default, the path goes from x to y.
fit	The fit object. Currently only supports lavaan::lavaan objects. Support for lists of lm() output is implemented by high level functions such as indirect_effect() and cond_indirect_effects(). It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().
est	The output of lavaan::parameterEstimates(). If NULL, the default, it will be generated from fit. If supplied, fit will be ignored.
implied_stats	Implied means, variances, and covariances of observed variables and latent variables (if any), of the form of the output of lavaan::lavInspect() with what set to "implied", but with means extracted with what set to "mean.ov" and

"mean.lv". The standard deviations are extracted from this object for standardization. Default is NULL, and implied statistics will be computed from fit if required.

wvalues A numeric vector of named elements. The names are the variable names of the moderators, and the values are the values to which the moderators will be set to. Default is NULL.

standardized_x Logical. Whether x will be standardized. Default is FALSE.

standardized_y Logical. Whether y will be standardized. Default is FALSE.

computation_digits
The number of digits in storing the computation in text. Default is 3.

prods The product terms found. For internal use.

get_prods_only IF TRUE, will quit early and return the product terms found. The results can be passed to the prod argument when calling this function. Default is FALSE. For internal use.

data Data frame (optional). If supplied, it will be used to identify the product terms. For internal use.

expand Whether products of more than two terms will be searched. TRUE by default. For internal use.

warn If TRUE, the default, the function will warn against possible misspecification, such as not setting the value of a moderator which moderate one of the component path. Set this to FALSE will suppress these warnings. Suppress them only when the moderators are omitted intentionally.

allow_mixing_lav_and_obs
If TRUE, it accepts a path with both latent variables and observed variables. Default is TRUE.

Details

This function is a low-level function called by indirect_effect(), cond_indirect_effects(), and cond_indirect(), which call this function multiple times if bootstrap confidence interval is requested.

This function usually should not be used directly. It is exported for advanced users and developers

Value

It returns an indirect-class object. This class has the following methods: coef.indirect(), print.indirect(). The confint.indirect() method is used only when called by cond_indirect() or cond_indirect_effects().

See Also

indirect_effect(), cond_indirect_effects(), and cond_indirect(), the high level functions that should usually be used.

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ a1 * x   + b1 * w1 + d1 * x:w1
m2 ~ a2 * m1  + b2 * w2 + d2 * m1:w2
m3 ~ a3 * m2  + b3 * w3 + d3 * m2:w3
y  ~ a4 * m3  + b4 * w4 + d4 * m3:w4
"
fit <- sem(mod, dat, meanstructure = TRUE,
           fixed.x = FALSE, se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

wvalues <- c(w1 = 5, w2 = 4, w3 = 2, w4 = 3)

# Compute the conditional indirect effect by indirect_i()
indirect_1 <- indirect_i(x = "x", y = "y", m = c("m1", "m2", "m3"), fit = fit,
                         wvalues = wvalues)

# Manually compute the conditional indirect effect
indirect_2 <- (est[est$label == "a1", "est"] +
                 wvalues["w1"] * est[est$label == "d1", "est"]) *
               (est[est$label == "a2", "est"] +
                 wvalues["w2"] * est[est$label == "d2", "est"]) *
               (est[est$label == "a3", "est"] +
                 wvalues["w3"] * est[est$label == "d3", "est"]) *
               (est[est$label == "a4", "est"] +
                 wvalues["w4"] * est[est$label == "d4", "est"])
# They should be the same
coef(indirect_1)
indirect_2
```

---

indirect_proportion *Proportion of Effect Mediated*

Description

It computes the proportion of effect mediated along a pathway.

Usage

```
indirect_proportion(x, y, m = NULL, fit = NULL)
```

Arguments

x	The name of the x variable. Must be supplied as a quoted string.

y	The name of the y variable. Must be supplied as a quoted string.
m	A vector of the variable names of the mediator(s). The path goes from the first mediator successively to the last mediator. Cannot be NULL for this function.
fit	The fit object. Can be a lavaan::lavaan object or a list of lm() outputs. It can also be a lavaan.mi object returned by semTools::runMI() or its wrapper, such as semTools::sem.mi().

Details

The proportion of effect mediated along a path from x to y is the indirect effect along this path divided by the total effect from x to y (Alwin & Hauser, 1975). This total effect is equal to the sum of all indirect effects from x to y and the direct effect from x to y.

To ensure that the proportion can indeed be interpreted as a proportion, this function computes the the proportion only if the signs of all the indirect and direct effects from x to y are same (i.e., all effects positive or all effects negative).

Value

An indirect_proportion class object. It is a list-like object with these major elements:

- proportion: The proportion of effect mediated.
- x: The name of the x-variable.
- y: The name of the y-variable.
- m: A character vector of the mediator(s) along a path. The path runs from the first element to the last element.

This class has a print method and a coef method.

References

Alwin, D. F., & Hauser, R. M. (1975). The decomposition of effects in path analysis. *American Sociological Review, 40*(1), 37. doi:10.2307/2094445

See Also

print.indirect_proportion() for the print method, and coef.indirect_proportion() for the coef method.

Examples

```
library(lavaan)
dat <- data_med
head(dat)
mod <-
"
m ~ x + c1 + c2
y ~ m + x + c1 + c2
"
```

```
fit <- sem(mod, dat, fixed.x = FALSE)
out <- indirect_proportion(x = "x",
                           y = "y",
                           m = "m",
                           fit = fit)
out
```

---

lm2boot_out *Bootstrap Estimates for* lm *Outputs*

Description

Generate bootstrap estimates for models in a list of 'lm' outputs.

Usage

```
lm2boot_out(outputs, R = 100, seed = NULL, progress = TRUE)

lm2boot_out_parallel(
  outputs,
  R = 100,
  seed = NULL,
  parallel = FALSE,
  ncores = max(parallel::detectCores(logical = FALSE) - 1, 1),
  make_cluster_args = list(),
  progress = TRUE
)
```

Arguments

outputs	A list of lm class objects, or the output of lm2list() (i.e., an lm_list-class object).
R	The number of bootstrap samples. Default is 100.
seed	The seed for the random resampling. Default is NULL.
progress	Logical. Display progress or not. Default is TRUE.
parallel	Logical. Whether parallel processing will be used. Default is NULL.
ncores	Integer. The number of CPU cores to use when parallel is TRUE. Default is the number of non-logical cores minus one (one minimum). Will raise an error if greater than the number of cores detected by parallel::detectCores(). If ncores is set, it will override make_cluster_args.
make_cluster_args	A named list of additional arguments to be passed to parallel::makeCluster(). For advanced users. See parallel::makeCluster() for details. Default is list().

Details

This function is for advanced users. do_boot() is a function users should try first because do_boot() has a general interface for input-specific functions like this one.

It does nonparametric bootstrapping to generate bootstrap estimates of the regression coefficients in the regression models of a list of lm() outputs, or an lm_list-class object created by lm2list(). The stored estimates can be used by indirect_effect(), cond_indirect_effects(), and related functions in forming bootstrapping confidence intervals for effects such as indirect effect and conditional indirect effects.

This approach removes the need to repeat bootstrapping in each call to indirect_effect(), cond_indirect_effects(), and related functions. It also ensures that the same set of bootstrap samples is used in all subsequent analyses.

Value

A boot_out-class object that can be used for the boot_out argument of indirect_effect(), cond_indirect_effects(), and related functions for forming bootstrapping confidence intervals. The object is a list with the number of elements equal to the number of bootstrap samples. Each element is a list of the parameter estimates and sample variances and covariances of the variables in each bootstrap sample.

Functions

- lm2boot_out(): Generate bootstrap estimates using one process (serial, without parallelization).
- lm2boot_out_parallel(): Generate bootstrap estimates using parallel processing.

See Also

do_boot(), the general purpose function that users should try first before using this function.

Examples

```
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
lm_m <- lm(m ~ x*w + c1 + c2, dat)
lm_y <- lm(y ~ m*w + x + c1 + c2, dat)
lm_out <- lm2list(lm_m, lm_y)
# In real research, R should be 2000 or even 5000
# In real research, no need to set progress to FALSE
# Progress is displayed by default.
lm_boot_out <- lm2boot_out(lm_out, R = 100, seed = 1234,
                           progress = FALSE)
out <- cond_indirect_effects(wlevels = "w",
                             x = "x",
                             y = "y",
                             m = "m",
                             fit = lm_out,
                             boot_ci = TRUE,
```

```
                                boot_out = lm_boot_out)
out
```

`lm2list` *Join 'lm()' Output to Form an 'lm_list'-Class Object*

Description

The resulting model can be used by `indirect_effect()`, `cond_indirect_effects()`, or `cond_indirect()` as a path method, as if fitted by `lavaan::sem()`.

Usage

```
lm2list(...)
```

Arguments

`...`	The `lm()` outputs to be grouped in a list.

Details

If a path model with mediation and/or moderation is fitted by a set of regression models using `lm()`, this function can combine them to an object of the class `lm_list` that represents a path model, as one fitted by structural equation model functions such as `lavaan::sem()`. This class of object can be used by some functions, such as `indirect_effect()`, `cond_indirect_effects()`, and `cond_indirect()` as if they were the output of fitting a path model, with the regression coefficients treated as path coefficients.

The regression outputs to be combined need to meet the following requirements:

- All models must be connected to at least one another model. That is, a regression model must either have (a) at least on predictor that is the outcome variable of another model, or (b) its outcome variable the predictor of another model.
- All models must be fitted to the same sample. This implies that the sample size must be the same in all analysis.

Value

It returns an `lm_list`-class object that forms a path model represented by a set of regression models. This class has a `summary` method that shows the summary of each regression model stored (see `summary.lm_list()`), and a `print` method that prints the models stored (see `print.lm_list()`).

See Also

`summary.lm_list()` and `print.lm_list()` for related methods, `indirect_effect()` and `cond_indirect_effects()` which accept `lm_list`-class objects as input.

Examples

```
data(data_serial_parallel)
lm_m11 <- lm(m11 ~ x + c1 + c2, data_serial_parallel)
lm_m12 <- lm(m12 ~ m11 + x + c1 + c2, data_serial_parallel)
lm_m2 <- lm(m2 ~ x + c1 + c2, data_serial_parallel)
lm_y <- lm(y ~ m11 + m12 + m2 + x + c1 + c2, data_serial_parallel)
# Join them to form a lm_list-class object
lm_serial_parallel <- lm2list(lm_m11, lm_m12, lm_m2, lm_y)
lm_serial_parallel
summary(lm_serial_parallel)

# Compute indirect effect from x to y through m11 and m12
outm11m12 <- cond_indirect(x = "x", y = "y",
                           m = c("m11", "m12"),
                           fit = lm_serial_parallel)
outm11m12
# Compute indirect effect from x to y
# through m11 and m12 with bootstrapping CI
# R should be at least 2000 or even 5000 in read study.
# In real research, parallel and progress can be omitted.
# They are est to TRUE by default.
outm11m12 <- cond_indirect(x = "x", y = "y",
                           m = c("m11", "m12"),
                           fit = lm_serial_parallel,
                           boot_ci = TRUE,
                           R = 100,
                           seed = 1234,
                           parallel = FALSE,
                           progress = FALSE)
outm11m12
```

---

lm_from_lavaan_list *'lavaan'-class to 'lm_from_lavaan_list'-Class*

Description

Converts the regression models in a lavaan-class model to an lm_from_lavaan_list-class object.

Usage

```
lm_from_lavaan_list(fit)
```

Arguments

fit	A lavaan-class object, usually the output of lavaan::lavaan() or its wrappers.

Details

It identifies all dependent variables in a lavaan model and creates an lm_from_lavaan-class object for each of them.

This is an advanced helper used by plot.cond_indirect_effects(). Exported for advanced users and developers.

Value

An lm_from_lavaan_list-class object, which is a list of lm_from_lavaan objects. It has a predict-method (predict.lm_from_lavaan_list()) for computing the predicted values from one variable to another.

See Also

predict.lm_from_lavaan_list

Examples

```
library(lavaan)
data(data_med)
mod <-
"
m ~ a * x + c1 + c2
y ~ b * m + x + c1 + c2
"
fit <- sem(mod, data_med, fixed.x = FALSE)
fit_list <- lm_from_lavaan_list(fit)
tmp <- data.frame(x = 1, c1 = 2, c2 = 3, m = 4)
predict(fit_list, x = "x", y = "y", m = "m", newdata = tmp)
```

---

math_indirect *Math Operators for 'indirect'-Class Objects*

Description

Mathematic operators for 'indirect'-class object, the output of indirect_effect() and cond_indirect().

Usage

```
## S3 method for class 'indirect'
e1 + e2

## S3 method for class 'indirect'
e1 - e2
```

Arguments

e1	An 'indirect'-class object.
e2	An 'indirect'-class object.

Details

For now, only + operator and - operator are supported. These operators can be used to estimate and test a function of effects between the same pair of variables but along different paths.

For example, they can be used to compute and test the total effects along different paths. They can also be used to compute and test the difference between the effects along two paths.

The operators will check whether an operation is valid. An operation is not valid if

1. the two paths do not start from the same variable,
2. the two paths do not end at the same variable, (c) a path appears in both objects,
3. moderators are involved but they are not set to the same values in both objects, and
4. bootstrap estimates stored in boot_out, if any, are not identical.
5. Monte Carlo simulated estimates stored in mc_out, if any, are not identical.

Value

An 'indirect'-class object with a list of effects stored. See indirect_effect() on details for this class.

See Also

indirect_effect() and cond_indirect()

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ a1 * x  + d1 * w1 + e1 * x:w1
m2 ~ m1 + a2 * x
y  ~ b1 * m1 + b2 * m2 + cp * x
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)
hi_w1 <- mean(dat$w1) + sd(dat$w1)

# Examples for cond_indirect():

# Conditional effect from x to m1 when w1 is 1 SD above mean
out1 <- cond_indirect(x = "x", y = "y", m = c("m1", "m2"),
              wvalues = c(w1 = hi_w1), fit = fit)
out2 <- cond_indirect(x = "x", y = "y", m = c("m2"),
```

```
              wvalues = c(w1 = hi_w1), fit = fit)
out3 <- cond_indirect(x = "x", y = "y",
              wvalues = c(w1 = hi_w1), fit = fit)

out12 <- out1 + out2
out12
out123 <- out1 + out2 + out3
out123
coef(out1) + coef(out2) + coef(out3)
```

`merge_mod_levels` *Merge the Generated Levels of Moderators*

Description

Merge the levels of moderators generated by `mod_levels()` into a data frame.

Usage

```
merge_mod_levels(...)
```

Arguments

...	The output from `mod_levels()`, or a list of levels generated by `mod_levels_list()`.

Details

It merges the levels of moderators generated by `mod_levels()` into a data frame, with each row represents a combination of the levels. The output is to be used by `cond_indirect_effects()`.

Users usually do not need to use this function because `cond_indirect_effects()` will merge the levels internally if necessary. This function is used when users need to customize the levels for each moderator and so cannot use `mod_levels_list()` or the default levels in `cond_indirect_effects()`.

Value

A `wlevels`-class object, which is a data frame of the combinations of levels, with additional attributes about the levels.

See Also

`mod_levels()` on generating the levels of a moderator.

Examples

```
data(data_med_mod_ab)
dat <- data_med_mod_ab
# Form the levels from a list of lm() outputs
lm_m <- lm(m ~ x*w1 + c1 + c2, dat)
lm_y <- lm(y ~ m*w2 + x + w1 + c1 + c2, dat)
lm_out <- lm2list(lm_m, lm_y)
w1_levels <- mod_levels(lm_out, w = "w1")
w1_levels
w2_levels <- mod_levels(lm_out, w = "w2")
w2_levels
merge_mod_levels(w1_levels, w2_levels)
```

---

modmed_x1m3w4y1 *Sample Dataset: Moderated Serial Mediation*

Description

Generated from a serial mediation model with one predictor, three mediators, and one outcome variable, with one moderator in each stage.

Usage

```
modmed_x1m3w4y1
```

Format

A data frame with 200 rows and 11 variables:

x Predictor. Numeric.

w1 Moderator 1. Numeric.

w2 Moderator 2. Numeric.

w3 Moderator 3. Numeric.

w4 Moderator 4. Numeric.

m1 Mediator 1. Numeric.

m2 Mediator 2. Numeric.

m3 Mediator 3. Numeric.

y Outcome variable. Numeric.

gp Three values: "earth", "mars", "venus". String.

city Four values: "alpha", "beta", "gamma", "sigma". String.

mod_levels *Create Levels of Moderators*

Description

Create levels of moderators to be used by `indirect_effect()`, `cond_indirect_effects()`, and `cond_indirect()`.

Usage

```
mod_levels(
  w,
  fit,
  w_type = c("auto", "numeric", "categorical"),
  w_method = c("sd", "percentile"),
  sd_from_mean = c(-1, 0, 1),
  percentiles = c(0.16, 0.5, 0.84),
  extract_gp_names = TRUE,
  prefix = NULL,
  values = NULL,
  reference_group_label = NULL,
  descending = TRUE
)

mod_levels_list(
  ...,
  fit,
  w_type = "auto",
  w_method = "sd",
  sd_from_mean = NULL,
  percentiles = NULL,
  extract_gp_names = TRUE,
  prefix = NULL,
  descending = TRUE,
  merge = FALSE
)
```

Arguments

w	Character. The names of the moderator. If the moderator is categorical with 3 or more groups, this is the vector of the indicator variables.
fit	The fit object. Can be a lavaan::lavaan object or a list of `lm()` outputs. It can also be a `lavaan.mi` object returned by `semTools::runMI()` or its wrapper, such as `semTools::sem.mi()`.
w_type	Character. Whether the moderator is a `"numeric"` variable or a `"categorical"` variable. If `"auto"`, the function will try to determine the type automatically.

w_method
: Character, either "sd" or "percentile". If "sd", the levels are defined by the distance from the mean in terms of standard deviation. if "percentile", the levels are defined in percentiles.

sd_from_mean
: A numeric vector. Specify the distance in standard deviation from the mean for each level. Default is c(-1, 0, 1) for mod_levels(). For mod_levels_list(), the default is c(-1, 0, 1) when there is only one moderator, and c(-1, 1) when there are more than one moderator. Ignored if w_method is not equal to "sd".

percentiles
: A numeric vector. Specify the percentile (in proportion) for each level. Default is c(.16, .50, .84) for mod_levels(), corresponding approximately to one standard deviation below mean, mean, and one standard deviation above mean in a normal distribution. For mod_levels_list(), default is c(.16, .50, .84) if there is one moderator, and c(.16, .84) when there are more than one moderator. Ignored if w_method is not equal to "percentile".

extract_gp_names
: Logical. If TRUE, the default, the function will try to determine the name of each group from the variable names.

prefix
: Character. If extract_gp_names is TRUE and prefix is supplied, it will be removed from the variable names to create the group names. Default is NULL, and the function will try to determine the prefix automatically.

values
: For numeric moderators, a numeric vector. These are the values to be used and will override other options. For categorical moderators, a named list of numeric vector, each vector has length equal to the number of indicator variables. If the vector is named, the names will be used to label the values. For example, if set to list(gp1 = c(0, 0), gp3 = c(0, 1), two levels will be returned, one named gp1 with the indicator variables equal to 0 and 0, the other named gp3 with the indicator variables equal to 0 and 1. Default is NULL.

reference_group_label
: For categorical moderator, if the label for the reference group (group with all indicators equal to zero) cannot be determined, the default label is "Reference". To change it, set reference_group_label to the desired label. Ignored if values is set.

descending
: If TRUE (default), the rows are sorted in descending order for numerical moderators: The highest value on the first row and the lowest values on the last row. For user supplied values, the first value is on the last row and the last value is on the first row. If FALSE, the rows are sorted in ascending order.

...
: The names of moderators variables. For a categorical variable, it should be a vector of variable names.

merge
: If TRUE, mod_levels_list() will call merge_mod_levels() and return the merged levels. Default is FALSE.

Details

It creates values of a moderator that can be used to compute conditional effect or conditional indirect effect. By default, for a numeric moderator, it uses one standard deviation below mean, mean, and one standard deviation above mean. The percentiles of these three levels in a normal distribution (16th, 50th, and 84th) can also be used. For categorical variable, it will simply collect the unique categories in the data.

The generated levels are then used by cond_indirect() and cond_indirect_effects().

If a model has more than one moderator, mod_levels_list() can be used to generate combinations of levels. The output can then passed to cond_indirect_effects() to compute the conditional effects or conditional indirect effects for all the combinations.

Value

mod_levels() returns a wlevels-class object which is a data frame with additional attributes about the levels.

mod_levels_list() returns a list of wlevels-class objects, or a wlevels-class object which is a data frame of the merged levels if merge = TRUE.

Functions

- mod_levels(): Generate levels for one moderator.
- mod_levels_list(): Generate levels for several moderators.

See Also

cond_indirect_effects() for computing conditional indiret effects; merge_mod_levels() for merging levels of moderators.

Examples

```
library(lavaan)
data(data_med_mod_ab)
dat <- data_med_mod_ab
# Form the levels from a list of lm() outputs
lm_m <- lm(m ~ x*w1 + c1 + c2, dat)
lm_y <- lm(y ~ m*w2 + x + w1 + c1 + c2, dat)
lm_out <- lm2list(lm_m, lm_y)
w1_levels <- mod_levels(lm_out, w = "w1")
w1_levels
w2_levels <- mod_levels(lm_out, w = "w2")
w2_levels
# Indirect effect from x to y through m, at the first levels of w1 and w2
cond_indirect(x = "x", y = "y", m = "m",
              fit = lm_out,
              wvalues = c(w1 = w1_levels$w1[1],
                          w2 = w2_levels$w2[1]))
# Can form the levels based on percentiles
w1_levels2 <- mod_levels(lm_out, w = "w1", w_method = "percentile")
w1_levels2
# Form the levels from a lavaan output
# Compute the product terms before fitting the model
dat$mw2 <- dat$m * dat$w2
mod <-
"
m ~ x + w1 + x:w1 + c1 + c2
y ~ m + x + w1 + w2 + mw2 + c1 + c2
```

```
"
fit <- sem(mod, dat, fixed.x = FALSE)
cond_indirect(x = "x", y = "y", m = "m",
              fit = fit,
              wvalues = c(w1 = w1_levels$w1[1],
                          w2 = w2_levels$w2[1]))
# Can pass all levels to cond_indirect_effects()
# First merge the levels by merge_mod_levels()
w1w2_levels <- merge_mod_levels(w1_levels, w2_levels)
cond_indirect_effects(x = "x", y = "y", m = "m",
                      fit = fit,
                      wlevels = w1w2_levels)




# mod_levels_list() forms a combinations of levels in one call
# It returns a list, by default.
# Form the levels from a list of lm() outputs
# "merge = TRUE" is optional. cond_indirect_effects will merge the levels
# automatically.
w1w2_levels <- mod_levels_list("w1", "w2", fit = fit, merge = TRUE)
w1w2_levels
cond_indirect_effects(x = "x", y = "y", m = "m",
                      fit = fit, wlevels = w1w2_levels)
# Can work without merge = TRUE:
w1w2_levels <- mod_levels_list("w1", "w2", fit = fit)
w1w2_levels
cond_indirect_effects(x = "x", y = "y", m = "m",
                      fit = fit, wlevels = w1w2_levels)
```

`plot.cond_indirect_effects` *Plot Conditional Effects*

Description

Plot the conditional effects for different levels of moderators.

Usage

```
## S3 method for class 'cond_indirect_effects'
plot(
  x,
  x_label,
  w_label = "Moderator(s)",
  y_label,
```

```
  title,
  x_from_mean_in_sd = 1,
  x_method = c("sd", "percentile"),
  x_percentiles = c(0.16, 0.84),
  x_sd_to_percentiles = NA,
  note_standardized = TRUE,
  no_title = FALSE,
  line_width = 1,
  point_size = 5,
  graph_type = c("default", "tumble"),
  ...
)
```

Arguments

x	The output of cond_indirect_effects(). (Named x because it is required in the naming of arguments of the plot generic function.)
x_label	The label for the X-axis. Default is the value of the predictor in the output of cond_indirect_effects().
w_label	The label for the legend for the lines. Default is "Moderator(s)".
y_label	The label for the Y-axis. Default is the name of the response variable in the model.
title	The title of the graph. If not supplied, it will be generated from the variable names or labels (in x_label, y_label, and w_label). If "", no title will be printed. This can be used when the plot is for manuscript submission and figures are required to have no titles.
x_from_mean_in_sd	How many SD from mean is used to define "low" and "high" for the focal variable. Default is 1.
x_method	How to define "high" and "low" for the focal variable levels. Default is in terms of the standard deviation of the focal variable, "sd". If equal to "percentile", then the percentiles of the focal variable in the dataset is used.
x_percentiles	If x_method is "percentile", then this argument specifies the two percentiles to be used, divided by 100. It must be a vector of two numbers. The default is c(.16, .84), the 16th and 84th percentiles, which corresponds approximately to one SD below and above mean for a normal distribution, respectively.
x_sd_to_percentiles	If x_method is "percentile" and this argument is set to a number, this number will be used to determine the percentiles to be used. The lower percentile is the percentile in a normal distribution that is x_sd_to_percentiles SD below the mean. The upper percentile is the percentile in a normal distribution that is x_sd_to_percentiles SD above the mean. Therefore, if x_sd_to_percentiles is set to 1, then the lower and upper percentiles are 16th and 84th, respectively. Default is NA.
note_standardized	If TRUE, will check whether a variable has SD nearly equal to one. If yes, will report this in the plot. Default is TRUE.

no_title	If TRUE, title will be suppressed. Default is FALSE.
line_width	The width of the lines as used in ggplot2::geom_segment(). Default is 1.
point_size	The size of the points as used in ggplot2::geom_point(). Default is 5.
graph_type	If "default", the typical line-graph with equal end-points will be plotted. If "tubmle", then the tumble graph proposed by Bodner (2016) will be plotted. Default is "default".
...	Additional arguments. Ignored.

Details

This function is a plot method of the output of cond_indirect_effects(). It will use the levels of moderators in the output.

It plots the conditional effect from x to y in a model for different levels of the moderators.

It does not support conditional indirect effects. If there is one or more mediators in x, it will raise an error.

Value

A ggplot2 graph. Plotted if not assigned to a name. It can be further modified like a usual ggplot2 graph.

References

Bodner, T. E. (2016). Tumble graphs: Avoiding misleading end point extrapolation when graphing interactions from a moderated multiple regression analysis. *Journal of Educational and Behavioral Statistics, 41*(6), 593-604. doi:10.3102/1076998616657080

See Also

cond_indirect_effects()

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
n <- nrow(dat)
set.seed(860314)
dat$gp <- sample(c("gp1", "gp2", "gp3"), n, replace = TRUE)
dat <- cbind(dat, factor2var(dat$gp, prefix = "gp", add_rownames = FALSE))

# Categorical moderator

mod <-
"
m3 ~ m1 + x + gpgp2 + gpgp3 + x:gpgp2 + x:gpgp3
y ~ m2 + m3 + x
"
fit <- sem(mod, dat, meanstructure = TRUE, fixed.x = FALSE)
out_mm_1 <- mod_levels(c("gpgp2", "gpgp3"),
```

```
                        sd_from_mean = c(-1, 1),
                        fit = fit)
out_1 <- cond_indirect_effects(wlevels = out_mm_1, x = "x", y = "m3", fit = fit)
plot(out_1)
plot(out_1, graph_type = "tumble")

# Numeric moderator

dat <- modmed_x1m3w4y1
mod2 <-
"
m3 ~ m1 + x + w1 + x:w1
y ~ m3 + x
"
fit2 <- sem(mod2, dat, meanstructure = TRUE, fixed.x = FALSE)
out_mm_2 <- mod_levels("w1",
                        w_method = "percentile",
                        percentiles = c(.16, .84),
                        fit = fit2)
out_mm_2
out_2 <- cond_indirect_effects(wlevels = out_mm_2, x = "x", y = "m3", fit = fit2)
plot(out_2)
plot(out_2, graph_type = "tumble")
```

`predict.lm_from_lavaan` *Predicted Values of a 'lm_from_lavaan'-Class Object*

Description

Compute the predicted values based on the model stored in a 'lm_from_lavaan'-class object.

Usage

```
## S3 method for class 'lm_from_lavaan'
predict(object, newdata, ...)
```

Arguments

`object`	A 'lm_from_lavaan'-class object.
`newdata`	Required. A data frame of the new data. It must be a data frame.
`...`	Additional arguments. Ignored.

Details

An `lm_from_lavaan`-class method that converts a regression model for a variable in a `lavaan` model to a `formula` object. This function uses the stored model to compute predicted values using user-supplied data.

This is an advanced helper used by `plot.cond_indirect_effects()`. Exported for advanced users and developers.

Value

A numeric vector of the predicted values, with length equal to the number of rows of user-supplied data.

See Also

`lm_from_lavaan_list()`

Examples

```
library(lavaan)
data(data_med)
mod <-
"
m ~ a * x + c1 + c2
y ~ b * m + x + c1 + c2
"
fit <- sem(mod, data_med, fixed.x = FALSE)
fit_list <- lm_from_lavaan_list(fit)
tmp <- data.frame(x = 1, c1 = 2, c2 = 3, m = 4)
predict(fit_list$m, newdata = tmp)
predict(fit_list$y, newdata = tmp)
```

---

predict.lm_from_lavaan_list

Predicted Values of an 'lm_from_lavaan_list'-Class Object

Description

It computes the predicted values based on the models stored in an 'lm_from_lavaan_list'-class object.

Usage

```
## S3 method for class 'lm_from_lavaan_list'
predict(object, x = NULL, y = NULL, m = NULL, newdata, ...)
```

Arguments

object	A 'lm_from_lavaan'-class object.
x	The variable name at the start of a path.
y	The variable name at the end of a path.
m	Optional. The mediator(s) from x to y. A numeric vector of the names of the mediators. The path goes from the first element to the last element. For example, if m = c("m1", "m2"), then the path is x -> m1 -> m2 -> y.
newdata	Required. A data frame of the new data. It must be a data frame.
...	Additional arguments. Ignored.

Details

An lm_from_lavaan_list-class object is a list of lm_from_lavaan-class objects.

This is an advanced helper used by plot.cond_indirect_effects(). Exported for advanced users and developers.

Value

A numeric vector of the predicted values, with length equal to the number of rows of user-supplied data.

See Also

lm_from_lavaan_list()

Examples

```
library(lavaan)
data(data_med)
mod <-
"
m ~ a * x + c1 + c2
y ~ b * m + x + c1 + c2
"
fit <- sem(mod, data_med, fixed.x = FALSE)
fit_list <- lm_from_lavaan_list(fit)
tmp <- data.frame(x = 1, c1 = 2, c2 = 3, m = 4)
predict(fit_list, x = "x", y = "y", m = "m", newdata = tmp)
```

`predict.lm_list` *Predicted Values of an 'lm_list'-Class Object*

Description

Compute the predicted values based on the models stored in an 'lm_list'-class object.

Usage

```
## S3 method for class 'lm_list'
predict(object, x = NULL, y = NULL, m = NULL, newdata, ...)
```

Arguments

`object`	An 'lm_list'-class object.
`x`	The variable name at the start of a path.
`y`	The variable name at the end of a path.
`m`	Optional. The mediator(s) from `x` to `y`. A numeric vector of the names of the mediators. The path goes from the first element to the last element. For example, if `m = c("m1", "m2")`, then the path is `x -> m1 -> m2 -> y`.
`newdata`	Required. A data frame of the new data. It must be a data frame.
`...`	Additional arguments. Ignored.

Details

An `lm_list`-class object is a list of `lm`-class objects, this function is similar to the `stats::predict()` method of `lm()` but it works on a system defined by a list of regression models.

This is an advanced helper used by some functions in this package. Exported for advanced users.

Value

A numeric vector of the predicted values, with length equal to the number of rows of user-supplied data.

See Also

`lm2list()`

Examples

```
data(data_serial_parallel)
lm_m11 <- lm(m11 ~ x + c1 + c2, data_serial_parallel)
lm_m12 <- lm(m12 ~ m11 + x + c1 + c2, data_serial_parallel)
lm_m2 <- lm(m2 ~ x + c1 + c2, data_serial_parallel)
lm_y <- lm(y ~ m11 + m12 + m2 + x + c1 + c2, data_serial_parallel)
# Join them to form a lm_list-class object
```

```
lm_serial_parallel <- lm2list(lm_m11, lm_m12, lm_m2, lm_y)
lm_serial_parallel
summary(lm_serial_parallel)
newdat <- data_serial_parallel[3:5, ]
predict(lm_serial_parallel,
        x = "x",
        y = "y",
        m = "m2",
        newdata = newdat)
```

`print.all_paths` *Print 'all_paths' Class Object*

Description

Print the content of 'all_paths'-class object, which can be generated by `all_indirect_paths()`.

Usage

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

Arguments

`x`	A 'all_paths'-class object.
`...`	Optional arguments.

Details

This function is used to print the paths identified in a readable format.

Value

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

Author(s)

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

See Also

`all_indirect_paths()`

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)
# All indirect paths
out1 <- all_indirect_paths(fit)
out1
```

---

print.boot_out *Print a* boot_out*-Class Object*

Description

Print the content of the output of do_boot() or related functions.

Usage

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

Arguments

x	The output of do_boot(), or any boot_out-class object returned by similar functions.
...	Other arguments. Not used.

Value

x is returned invisibly. Called for its side effect.

Examples

```
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
lm_m <- lm(m ~ x*w + c1 + c2, dat)
lm_y <- lm(y ~ m*w + x + c1 + c2, dat)
lm_out <- lm2list(lm_m, lm_y)
# In real research, R should be 2000 or even 5000
# In real research, no need to set parallel to FALSE
```

```
# In real research, no need to set progress to FALSE
# Progress is displayed by default.
lm_boot_out <- do_boot(lm_out, R = 100,
                       seed = 1234,
                       progress = FALSE,
                       parallel = FALSE)
# Print the output of do_boot()
lm_boot_out

library(lavaan)
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
dat$"x:w" <- dat$x * dat$w
dat$"m:w" <- dat$m * dat$w
mod <-
"
m ~ x + w + x:w + c1 + c2
y ~ m + w + m:w + x + c1 + c2
"
fit <- sem(model = mod, data = dat, fixed.x = FALSE,
           se = "none", baseline = FALSE)
# In real research, R should be 2000 or even 5000
# In real research, no need to set progress to FALSE
# In real research, no need to set parallel to FALSE
# Progress is displayed by default.
fit_boot_out <- do_boot(fit = fit,
                        R = 40,
                        seed = 1234,
                        parallel = FALSE,
                        progress = FALSE)
# Print the output of do_boot()
fit_boot_out
```

`print.cond_indirect_diff` *Print the Output of 'cond_indirect_diff'*

Description

Print the output of `cond_indirect_diff()`.

Usage

```
## S3 method for class 'cond_indirect_diff'
print(x, digits = 3, pvalue = FALSE, pvalue_digits = 3, se = FALSE, ...)
```

Arguments

x	The output of cond_indirect_diff().
digits	The number of decimal places in the printout.
pvalue	Logical. If TRUE, asymmetric *p*-value based on bootstrapping will be printed if available. Default is FALSE.
pvalue_digits	Number of decimal places to display for the *p*-value. Default is 3.
se	Logical. If TRUE and confidence intervals are available, the standard errors of the estimates are also printed. They are simply the standard deviations of the bootstrap estimates or Monte Carlo simulated values, depending on the method used to form the confidence intervals.
...	Optional arguments. Ignored.

Details

The print method of the cond_indirect_diff-class object.

If bootstrapping confidence interval was requested, this method has the option to print a *p*-value computed by the method presented in Asparouhov and Muthén (2021). Note that this *p*-value is asymmetric bootstrap *p*-value based on the distribution of the bootstrap estimates. It is not computed based on the distribution under the null hypothesis.

For a *p*-value of *a*, it means that a 100(1 - *a*)% bootstrapping confidence interval will have one of its limits equal to 0. A confidence interval with a higher confidence level will include zero, while a confidence interval with a lower confidence level will exclude zero.

Value

It returns x invisibly. Called for its side effect.

References

Asparouhov, A., & Muthén, B. (2021). Bootstrap p-value computation. Retrieved from https://www.statmodel.com/download
Bootstrap%20-%20Pvalue.pdf

See Also

cond_indirect_diff()

---

print.cond_indirect_effects

Print a 'cond_indirect_effects' Class Object

Description

Print the content of the output of cond_indirect_effects()

Usage

```
## S3 method for class 'cond_indirect_effects'
print(
  x,
  digits = 3,
  annotation = TRUE,
  pvalue = FALSE,
  pvalue_digits = 3,
  se = FALSE,
  ...
)
```

Arguments

x	The output of cond_indirect_effects().
digits	Number of digits to display. Default is 3.
annotation	Logical. Whether the annotation after the table of effects is to be printed. Default is TRUE.
pvalue	Logical. If TRUE, asymmetric *p*-values based on bootstrapping will be printed if available. Default is FALSE.
pvalue_digits	Number of decimal places to display for the *p*-values. Default is 3.
se	Logical. If TRUE and confidence intervals are available, the standard errors of the estimates are also printed. They are simply the standard deviations of the bootstrap estimates or Monte Carlo simulated values, depending on the method used to form the confidence intervals.
...	Other arguments. Not used.

Details

The print method of the cond_indirect_effects-class object.

If bootstrapping confidence intervals were requested, this method has the option to print *p*-values computed by the method presented in Asparouhov and Muthén (2021). Note that these *p*-values are asymmetric bootstrap *p*-values based on the distribution of the bootstrap estimates. They not computed based on the distribution under the null hypothesis.

For a *p*-value of *a*, it means that a 100(1 - *a*)% bootstrapping confidence interval will have one of its limits equal to 0. A confidence interval with a higher confidence level will include zero, while a confidence interval with a lower confidence level will exclude zero.

Value

x is returned invisibly. Called for its side effect.

References

Asparouhov, A., & Muthén, B. (2021). Bootstrap p-value computation. Retrieved from https://www.statmodel.com/download
Bootstrap%20-%20Pvalue.pdf

See Also

cond_indirect_effects()

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ a1 * x  + d1 * w1 + e1 * x:w1
m2 ~ a2 * x
y  ~ b1 * m1 + b2 * m2 + cp * x
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE, se = "none", baseline = FALSE)

# Conditional effects from x to m1 when w1 is equal to each of the default levels
cond_indirect_effects(x = "x", y = "m1",
                      wlevels = "w1", fit = fit)

# Conditional Indirect effect from x1 through m1 to y,
# when w1 is equal to each of the default levels
out <- cond_indirect_effects(x = "x", y = "y", m = "m1",
                      wlevels = "w1", fit = fit)
out

print(out, digits = 5)

print(out, annotation = FALSE)
```

---

print.delta_med *Print a 'delta_med' Class Object*

Description

Print the content of a delta_med-class object.

Usage

```
## S3 method for class 'delta_med'
print(x, digits = 3, level = NULL, full = FALSE, ...)
```

Arguments

x	A delta_med-class object.
digits	The number of digits after the decimal. Default is 3.
level	The level of confidence of bootstrap confidence interval, if requested when created. If NULL, the default, the level requested when calling delta_med() is used. If not null, then this level will be used.
full	Logical. Whether additional information will be printed. Default is FALSE.
...	Optional arguments. Ignored.

Details

It prints the output of delta_med(), which is a delta_med-class object.

Value

x is returned invisibly. Called for its side effect.

Author(s)

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

See Also

delta_med()

Examples

```
library(lavaan)
dat <- data_med
mod <-
"
m ~ x
y ~ m + x
"
fit <- sem(mod, dat)
dm <- delta_med(x = "x",
                y = "y",
                m = "m",
                fit = fit)
dm
print(dm, full = TRUE)

# Call do_boot() to generate
# bootstrap estimates
# Use 2000 or even 5000 for R in real studies
# Set parallel to TRUE in real studies for faster bootstrapping
boot_out <- do_boot(fit,
                    R = 45,
                    seed = 879,
```

```
                    parallel = FALSE,
                    progress = FALSE)
# Remove 'progress = FALSE' in practice
dm_boot <- delta_med(x = "x",
                     y = "y",
                     m = "m",
                     fit = fit,
                     boot_out = boot_out,
                     progress = FALSE)
dm_boot
confint(dm_boot)
confint(dm_boot,
        level = .90)
```

`print.indirect` *Print an 'indirect' Class Object*

Description

Print the content of the output of `indirect_effect()` or `cond_indirect()`.

Usage

```
## S3 method for class 'indirect'
print(x, digits = 3, pvalue = FALSE, pvalue_digits = 3, se = FALSE, ...)
```

Arguments

`x`	The output of `indirect_effect()` or `cond_indirect()`.
`digits`	Number of digits to display. Default is 3.
`pvalue`	Logical. If `TRUE`, asymmetric *p*-value based on bootstrapping will be printed if available.
`pvalue_digits`	Number of decimal places to display for the *p*-value. Default is 3.
`se`	Logical. If `TRUE` and confidence interval is available, the standard error of the estimate is also printed. This is simply the standard deviation of the bootstrap estimates or Monte Carlo simulated values, depending on the method used to form the confidence interval.
`...`	Other arguments. Not used.

Details

The `print` method of the `indirect`-class object.

If bootstrapping confidence interval was requested, this method has the option to print a *p*-value computed by the method presented in Asparouhov and Muthén (2021). Note that this *p*-value is asymmetric bootstrap *p*-value based on the distribution of the bootstrap estimates. It is not computed based on the distribution under the null hypothesis.

For a *p*-value of *a*, it means that a 100(1 - *a*)% bootstrapping confidence interval will have one of its limits equal to 0. A confidence interval with a higher confidence level will include zero, while a confidence interval with a lower confidence level will exclude zero.

We recommend using confidence interval directly. Therefore, *p*-value is not printed by default. Nevertheless, users who need it can request it by setting `pvalue` to `TRUE`.

Value

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

References

Asparouhov, A., & Muthén, B. (2021). Bootstrap p-value computation. Retrieved from https://www.statmodel.com/download Bootstrap%20-%20Pvalue.pdf

See Also

`indirect_effect()` and `cond_indirect()`

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ a1 * x   + b1 * w1 + d1 * x:w1
m2 ~ a2 * m1  + b2 * w2 + d2 * m1:w2
m3 ~ a3 * m2  + b3 * w3 + d3 * m2:w3
y  ~ a4 * m3  + b4 * w4 + d4 * m3:w4
"
fit <- sem(mod, dat,
           meanstructure = TRUE, fixed.x = FALSE,
           se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

wvalues <- c(w1 = 5, w2 = 4, w3 = 2, w4 = 3)

indirect_1 <- cond_indirect(x = "x", y = "y",
                            m = c("m1", "m2", "m3"),
                            fit = fit,
                            wvalues = wvalues)
indirect_1

dat <- modmed_x1m3w4y1
mod2 <-
"
m1 ~ a1 * x
m2 ~ a2 * m1
m3 ~ a3 * m2
y  ~ a4 * m3 + x
"
```

```
fit2 <- sem(mod2, dat,
            meanstructure = TRUE, fixed.x = FALSE,
            se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

indirect_2 <- indirect_effect(x = "x", y = "y",
                              m = c("m1", "m2", "m3"),
                              fit = fit2)
indirect_2
print(indirect_2, digits = 5)
```

`print.indirect_list` *Print an 'indirect_list' Class Object*

Description

Print the content of the output of `many_indirect_effects()`.

Usage

```
## S3 method for class 'indirect_list'
print(
  x,
  digits = 3,
  annotation = TRUE,
  pvalue = FALSE,
  pvalue_digits = 3,
  se = FALSE,
  ...
)
```

Arguments

`x`	The output of `many_indirect_effects()`.
`digits`	Number of digits to display. Default is 3.
`annotation`	Logical. Whether the annotation after the table of effects is to be printed. Default is `TRUE`.
`pvalue`	Logical. If `TRUE`, asymmetric *p*-values based on bootstrapping will be printed if available.
`pvalue_digits`	Number of decimal places to display for the *p*-values. Default is 3.
`se`	Logical. If `TRUE` and confidence intervals are available, the standard errors of the estimates are also printed. They are simply the standard deviations of the bootstrap estimates or Monte Carlo simulated values, depending on the method used to form the confidence intervals.
`...`	Other arguments. Not used.

Details

The `print` method of the `indirect_list`-class object.

If bootstrapping confidence interval was requested, this method has the option to print a *p*-value computed by the method presented in Asparouhov and Muthén (2021). Note that this *p*-value is asymmetric bootstrap *p*-value based on the distribution of the bootstrap estimates. It is not computed based on the distribution under the null hypothesis.

For a *p*-value of *a*, it means that a 100(1 - *a*)% bootstrapping confidence interval will have one of its limits equal to 0. A confidence interval with a higher confidence level will include zero, while a confidence interval with a lower confidence level will exclude zero.

Value

x is returned invisibly. Called for its side effect.

References

Asparouhov, A., & Muthén, B. (2021). Bootstrap p-value computation. Retrieved from https://www.statmodel.com/download Bootstrap%20-%20Pvalue.pdf

See Also

`many_indirect_effects()`

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)
# All indirect paths from x to y
paths <- all_indirect_paths(fit,
                            x = "x",
                            y = "y")
paths
# Indirect effect estimates
out <- many_indirect_effects(paths,
                             fit = fit)
out
```

print.indirect_proportion *Print an 'indirect_proportion'-Class Object*

Description

Print the content of an 'indirect_proportion'-class object, the output of `indirect_proportion()`.

Usage

```
## S3 method for class 'indirect_proportion'
print(x, digits = 3, annotation = TRUE, ...)
```

Arguments

x	An 'indirect_proportion'-class object.
digits	Number of digits to display. Default is 3.
annotation	Logical. Whether additional information should be printed. Default is TRUE.
...	Optional arguments. Not used.

Details

The print method of the indirect_proportion-class object, which is produced by `indirect_proportion()`. In addition to the proportion of effect mediated, it also prints additional information such as the path for which the proportion is computed, and all indirect path(s) from the x-variable to the y-variable.

To get the proportion as a scalar, use the coef method of indirect_proportion objects.

Value

x is returned invisibly. Called for its side effect.

See Also

`indirect_proportion()`

Examples

```
library(lavaan)
dat <- data_med
head(dat)
mod <-
"
m ~ x + c1 + c2
y ~ m + x + c1 + c2
"
fit <- sem(mod, dat, fixed.x = FALSE)
```

```
out <- indirect_proportion(x = "x",
                           y = "y",
                           m = "m",
                           fit = fit)
out
print(out, digits = 5)
```

`print.lm_list` *Print an* `lm_list`*-Class Object*

Description

Print the content of the output of `lm2list()`.

Usage

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

Arguments

`x`	The output of `lm2list()`.
`...`	Other arguments. Not used.

Value

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

Examples

```
data(data_serial_parallel)
lm_m11 <- lm(m11 ~ x + c1 + c2, data_serial_parallel)
lm_m12 <- lm(m12 ~ m11 + x + c1 + c2, data_serial_parallel)
lm_m2 <- lm(m2 ~ x + c1 + c2, data_serial_parallel)
lm_y <- lm(y ~ m11 + m12 + m2 + x + c1 + c2, data_serial_parallel)
# Join them to form a lm_list-class object
lm_serial_parallel <- lm2list(lm_m11, lm_m12, lm_m2, lm_y)
lm_serial_parallel
```

print.mc_out *Print a* mc_out*-Class Object*

Description

Print the content of the output of do_mc() or related functions.

Usage

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

Arguments

x	The output of do_mc(), or any mc_out-class object returned by similar functions.
...	Other arguments. Not used.

Value

x is returned invisibly. Called for its side effect.

Examples

```
library(lavaan)
data(data_med_mod_ab1)
dat <- data_med_mod_ab1
mod <-
"
m ~ x + w + x:w + c1 + c2
y ~ m + w + m:w + x + c1 + c2
"
fit <- sem(mod, dat)
# In real research, R should be 5000 or even 10000
mc_out <- do_mc(fit, R = 100, seed = 1234)

# Print the output of do_boot()
mc_out
```

simple_mediation_latent *Sample Dataset: A Simple Latent Mediation Model*

Description

Generated from a simple mediation model among xthree latent factors, fx, fm, and fy, xeach has three indicators.

Usage

```
simple_mediation_latent
```

Format

A data frame with 200 rows and 11 variables:

x1 Indicator of fx. Numeric.

x2 Indicator of fx. Numeric.

x3 Indicator of fx. Numeric.

m1 Indicator of fm. Numeric.

m2 Indicator of fm. Numeric.

m3 Indicator of fm. Numeric.

y1 Indicator of fy. Numeric.

y2 Indicator of fy. Numeric.

y3 Indicator of fy. Numeric.

Details

The model:

```
fx =~ x1 + x2 + x3
fm =~ m1 + m2 + m3
fy =~ y1 + y2 + y3
fm ~ a * fx
fy ~ b * fm + cp * fx
indirect := a * b
```

`subsetting_cond_indirect_effects` *Extraction Methods for 'cond_indirect_effects' Outputs*

Description

For subsetting a 'cond_indirect_effects'-class object.

Usage

```
## S3 method for class 'cond_indirect_effects'
x[i, j, drop = if (missing(i)) TRUE else length(j) == 1]
```

Arguments

`x`	A 'cond_indirect_effects'-class object.
`i`	A numeric vector of row number(s), a character vector of row name(s), or a logical vector of row(s) to be selected.
`j`	A numeric vector of column number(s), a character vector of column name(s), or a logical vector of column(s) to be selected.
`drop`	Whether dropping a dimension if it only have one row/column.

Details

Customized `[` for 'cond_indirect_effects'-class objects, to ensure that these operations work as they would be on a data frame object, while information specific to conditional effects is modified correctly.

Value

A 'cond_indirect_effects'-class object. See `cond_indirect_effects()` for details on this class.

Examples

```
library(lavaan)
dat <- modmed_x1m3w4y1
mod <-
"
m1 ~ x  + w1 + x:w1
m2 ~ m1
y  ~ m2 + x + w4 + m2:w4
"
fit <- sem(mod, dat, meanstructure = TRUE, fixed.x = FALSE, se = "none", baseline = FALSE)
est <- parameterEstimates(fit)

# Examples for cond_indirect():
```

```
# Conditional effects from x to m1 when w1 is equal to each of the levels
out1 <- cond_indirect_effects(x = "x", y = "m1",
                     wlevels = "w1", fit = fit)
out1[2, ]

# Conditional Indirect effect from x1 through m1 to y,
# when w1 is equal to each of the levels
out2 <- cond_indirect_effects(x = "x", y = "y", m = c("m1", "m2"),
                     wlevels = c("w1", "w4"), fit = fit)
out2[c(1, 3), ]
```

`subsetting_wlevels` *Extraction Methods for a 'wlevels'-class Object*

Description

For subsetting a 'wlevels'-class object. Attributes related to the levels will be preserved if appropriate.

Usage

```
## S3 method for class 'wlevels'
x[i, j, drop = if (missing(i)) TRUE else length(j) == 1]

## S3 replacement method for class 'wlevels'
x[i, j] <- value

## S3 replacement method for class 'wlevels'
x[[i, j]] <- value
```

Arguments

`x`	A 'wlevels'-class object.
`i`	A numeric vector of row number(s), a character vector of row name(s), or a logical vector of row(s) to be selected.
`j`	A numeric vector of column number(s), a character vector of column name(s), or a logical vector of column(s) to be selected.
`drop`	Whether dropping a dimension if it only have one row/column.
`value`	Ignored.

Details

Customized `[` for 'wlevels'-class objects, to ensure that these operations work as they would be on a data frame object, while information specific to a `wlevels`-class object modified correctly.

The assignment methods `[<-` and `[[<-` for `wlevels`-class objects will raise an error. This class of objects should be created by `mod_levels()` or related functions.

Subsetting the output of mod_levels() is possible but not recommended. It is more reliable to generate the levels using mod_levels() and related functions. Nevertheless, there are situations in which subsetting is preferred.

Value

A 'wlevels'-class object. See mod_levels() and merge_mod_levels() for details on this class.

See Also

mod_levels(), mod_levels_list(), and merge_mod_levels()

Examples

```
data(data_med_mod_ab)
dat <- data_med_mod_ab
# Form the levels from a list of lm() outputs
lm_m <- lm(m ~ x*w1 + c1 + c2, dat)
lm_y <- lm(y ~ m*w2 + x + w1 + c1 + c2, dat)
lm_out <- lm2list(lm_m, lm_y)
w1_levels <- mod_levels(lm_out, w = "w1")
w1_levels
w1_levels[2, ]
w1_levels[c(2, 3), ]

dat <- data_med_mod_serial_cat
lm_m1 <- lm(m1 ~ x*w1 + c1 + c2, dat)
lm_y <- lm(y ~ m1 + x + w1 + c1 + c2, dat)
lm_out <- lm2list(lm_m1, lm_y)
w1gp_levels <- mod_levels(lm_out, w = "w1")
w1gp_levels
w1gp_levels[2, ]
w1gp_levels[3, ]

merged_levels <- merge_mod_levels(w1_levels, w1gp_levels)
merged_levels

merged_levels[4:6, ]
merged_levels[1:3, c(2, 3)]
merged_levels[c(1, 4, 7), 1, drop = FALSE]
```

---

summary.lm_list *Summary of an* lm_list*-Class Object*

Description

The summary of content of the output of lm2list().

Usage

```
## S3 method for class 'lm_list'
summary(object, ...)

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

Arguments

object	The output of lm2list().
...	Other arguments. Not used.
x	An object of class summary_lm_list.
digits	The number of significant digits in printing numerical results.

Value

summary.lm_list() returns a summary_lm_list-class object, which is a list of the summary() outputs of the lm() outputs stored.

print.summary_lm_list() returns x invisibly. Called for its side effect.

Functions

- print(summary_lm_list): Print method for output of summary for lm_list.

Examples

```
data(data_serial_parallel)
lm_m11 <- lm(m11 ~ x + c1 + c2, data_serial_parallel)
lm_m12 <- lm(m12 ~ m11 + x + c1 + c2, data_serial_parallel)
lm_m2 <- lm(m2 ~ x + c1 + c2, data_serial_parallel)
lm_y <- lm(y ~ m11 + m12 + m2 + x + c1 + c2, data_serial_parallel)
# Join them to form a lm_list-class object
lm_serial_parallel <- lm2list(lm_m11, lm_m12, lm_m2, lm_y)
lm_serial_parallel
summary(lm_serial_parallel)
```

---

terms.lm_from_lavaan *Model Terms of an 'lm_from_lavaan'-Class Object*

Description

It extracts the terms object from an lm_from_lavaan-class object.

Usage

```
## S3 method for class 'lm_from_lavaan'
terms(x, ...)
```

Arguments

x	An 'lm_from_lavaan'-class object.
...	Additional arguments. Ignored.

Details

A method for lm_from_lavaan-class that converts a regression model for a variable in a lavaan model to a formula object. This function simply calls stats::terms() on the formula object to extract the predictors of a variable.

Value

A terms-class object. See terms.object for details.

See Also

terms.object, lm_from_lavaan_list()

Examples

```
library(lavaan)
data(data_med)
mod <-
"
m ~ a * x + c1 + c2
y ~ b * m + x + c1 + c2
"
fit <- sem(mod, data_med, fixed.x = FALSE)
fit_list <- lm_from_lavaan_list(fit)
terms(fit_list$m)
terms(fit_list$y)
```

---

total_indirect_effect *Total Indirect Effect Between Two Variables*

Description

Compute the total indirect effect between two variables in the paths estimated by many_indirect_effects().

Usage

```
total_indirect_effect(object, x, y)
```

Arguments

object	The output of many_indirect_effects(), or a list of indirect-class objects.
x	Character. The name of the x variable. All paths start from x will be included.
y	Character. The name of the y variable. All paths end at y will be included.

Details

It extracts the indirect-class objects of relevant paths and then add the indirect effects together using the + operator.

Value

An indirect-class object.

See Also

many_indirect_effects()

Examples

```
library(lavaan)
data(data_serial_parallel)
mod <-
"
m11 ~ x + c1 + c2
m12 ~ m11 + x + c1 + c2
m2 ~ x + c1 + c2
y ~ m12 + m2 + m11 + x + c1 + c2
"
fit <- sem(mod, data_serial_parallel,
           fixed.x = FALSE)

# All indirect paths, control variables excluded
paths <- all_indirect_paths(fit,
                            exclude = c("c1", "c2"))
paths

# Indirect effect estimates
out <- many_indirect_effects(paths,
                             fit = fit)
out

# Total indirect effect from x to y
total_indirect_effect(out,
                      x = "x",
                      y = "y")
```

Index