Package ‘manymome’

September 9, 2022

Title Mediation, Moderation and Moderated-Mediation After Model Fitting

Version 0.1.4.3

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. 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 7.2.1

Suggests knitr, rmarkdown, semPlot, semptools, 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

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  2022-09-09 10:32:57 UTC

R topics documented:

check_path  ................................................................. 3
coef.cond_indirect_diff  .................................................. 5
coef.cond_indirect_effects  .............................................. 5
coef.indirect  ............................................................. 7
coef.lm_from_lavaan  ...................................................... 8
cond_indirect  ............................................................ 9
cond_indirect_diff  ....................................................... 15
confint.cond_indirect_diff  ............................................ 17
confint.cond_indirect_effects  ...................................... 17
confint.indirect  ......................................................... 19
data_med  ................................................................. 20
data_med_complicated  .................................................. 21
data_med_mod_a  .......................................................... 22
data_med_mod_ab  ......................................................... 23
data_med_mod_ab1  ....................................................... 24
data_med_mod_b  .......................................................... 25
data_med_mod_b_mod  ..................................................... 26
data_med_mod_parallel  .................................................. 26
data_med_mod_parallel_cat  .......................................... 28
data_med_mod_serial  ...................................................... 28
data_med_mod_serial_cat  .............................................. 29
data_med_mod_serial_parallel  ....................................... 30
data_med_mod_serial_parallel_cat  .................................. 32
data_mod  ................................................................. 33
data_mod2  ............................................................... 33
data_mod_cat  ............................................................. 34
data_mome_demo  .......................................................... 35
data_mome_demo_missing  ............................................... 36
data_parallel  ............................................................. 37
data_sem  ................................................................. 38
data_serial  ............................................................... 39
data_serial_parallel  .................................................... 40
do_boot  ................................................................. 41
factor2var  .............................................................. 43
fit2boot_out  ............................................................ 44
get_one_cond_indirect_effect  ......................................... 46
index_of_mome  ........................................................... 47
indirect_i  ............................................................... 50
lm2boot_out  ............................................................. 53
lm2list  ................................................................. 54
lm_from_lavaan_list  .................................................... 56
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().
- **est**: The output of lavaan::parameterEstimates(). If NULL, the default, it will be generated from fit. If supplied, fit will be ignored.
check_path

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 \texttt{lm()}.

For example, in the \texttt{ql} in lavaan syntax

\[
\begin{align*}
m1 & \sim x \\
m2 & \sim m1 \\
m3 & \sim x \\
y & \sim m2 + m3
\end{align*}
\]

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. \texttt{TRUE} if the path is valid, \texttt{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().
...

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

```r
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.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

```r
## 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

```r
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():

# Indirect effect from x through m2 to y
out1 <- indirect_effect(x = "x", y = "y", m = "m2", fit = fit)
out1
coeff(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
coeff(out2)

---

**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

```r
## 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()`
cond_indirect

Examples

```r
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()` or `lavaan::sem()`.

Usage

```r
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
)```
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(),
  ...
)

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
)
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.
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.
seed  If bootstrapping is conducted, this is the seed for the bootstrapping. 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.

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().

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 to form percentile confidence intervals. Bootstrapping only needs to be done once. These are the possible ways to form bootstrapping:

1. Do bootstrapping in the first call to one of these functions, by setting `boot_ci` to TRUE and R to the number of bootstrap samples, level to the level of confidence (default .95 or 95%), and seed to reproduce the results (parallel and ncores are optional). This will take some time to run. The output will have all bootstrap 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` to this output. They will form the confidence intervals using the stored bootstrap estimates.

2. Do bootstrapping using `do_boot()`. The output can be used in the `boot_out` argument of `indirect_effect()`, `cond_indirect_effects()` and `cond_indirect()`.

3. 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` 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.
See Also

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

Examples

```r
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():
cond_indirect(x = "x", y = "m1", wvalues = c(w1 = hi_w1), fit = fit)

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

# Conditional Indirect effect from x 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 x 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)
```

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 bootstrap confidence interval. Default is .95.

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

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 conditional indirect effect between two rows in the output of cond_indirect_effects().

References


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
coeff(out_ind)
confint(out_ind)
**Description**

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

**Usage**

```r
## 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 bootstrap 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.

**Value**

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

---

**Description**

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

**Usage**

```r
## 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 bootstrap confidence interval 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.

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

```r
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_indirectEffects(x = "x", y = "m1",
  wlevels = w1levels, fit = fit,
  boot_ci = TRUE, R = 20, seed = 54151,
  parallel = FALSE,
...)
```
confint.indirect

confint(out1)

---

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

**Description**

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

**Usage**

```r
## 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 bootstrap confidence interval if available.

**Value**

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

**See Also**

`indirect_effect()` and `cond_indirect()`

**Examples**

```r
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)

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
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))
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

```r
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), ]
```
**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.

**Examples**

```r
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)))
```

```r
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**

```r
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)))
  "
```

data_med_mod_b

```r
fit <- sem(mod, data_med_mod_ab1,
    meanstructure = TRUE, fixed.x = FALSE)
parameterEstimates(fit)[c(1, 3, 6, 9, 38:40), ]
```

**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**

```r
library(lavaan)
data(data_med_mod_b)
data_med_mod_b$mw <- data_med_mod_b$m * data_med_mod_b$w
data_med_mod_b$v_w <- data_med_mod_b$m
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

```r
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**

```r
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),]
```

---

**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**

```r
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

```r
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$m2 * data_med_mod_serial_parallel$m2
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
a1b1b22 := a1 * b11 * b12
a2b2 := a2 * b2
ab := a1b1b22 + a2b2
a1b1b2_w1lo := (a1 + da11 * (m_w1 - sqrt(v_w1))) * b11 * b12
a1b1b2_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 != "", ]
```
**Sample Dataset: Serial-Parallel Moderated Mediation with Two Categorical Moderators**

**Description**

A serial-parallel mediation model with two categorical moderators.

**Usage**

```r
data(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**

```r
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))
```
**Sample Dataset: One Moderator**

**Description**

A one-moderator model.

**Usage**

```r
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.

**Examples**

```r
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), ]
```

---

**Sample Dataset: Two Moderators**

**Description**

A two-moderator model.

**Usage**

```r
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

```r
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
data_mome_demo

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:

```r
# 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
```

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 2. Numeric.
- `c1` Control Variable 1. 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:

```r
# 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

```r
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"
```
```r
indirect2 := a2 * b2
direct := 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


Data serial parallel

\[
y = b_2 \times m_2 + m_1 + x + c_1 + c_2
\]
\[
\text{indirect} := a \times b_1 \times b_2
\]

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), ]
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. 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.
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**

```r
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
```
factor2var  

Create Dummy Variables

Description

Create dummy variables from a categorical variable.

Usage

```r
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 "contr.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

```r
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
)

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.

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**

```r
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")
```
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
index_of_mome

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)

---

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_c1 = FALSE,
  level = 0.95,
  ...)
index_of_mome:

```r
index_of_mome(
  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,
  ...
)
```

**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().
- **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.
- **seed** If bootstrapping is conducted, this is the seed for the bootstrapping. Default is NULL and seed is not set.
progress Logical. Display bootstrapping progress or not. Default is TRUE.

... 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 bootstrap 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


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
\[
\text{ind\_mome} := d \times b
\]

\[
\text{fit} <- \text{sem(mod, dat, meanstructure = TRUE, fixed.x = FALSE, se = "none", baseline = FALSE)}
\]

\[
\text{est} <- \text{parameterEstimates(fit)}
\]

# 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.
\[
\text{out\_mome} <- \text{index\_of\_mome(x = "x", y = "y", m = "m1", w = "w1", fit = fit, boot\_ci = TRUE, R = 42, seed = 4314, parallel = FALSE, progress = FALSE)}
\]

\[
\text{out\_mome}
\]

\[
\text{coef(out\_mome)}
\]

# From lavaan
\[
\text{print(est[19, ], nd = 8)}
\]

\[
\text{confint(out\_mome)}
\]

\[
\text{library(lavaan)}
\]

\[
\text{dat} <- \text{modmed\_x1m3w4y1}
\]

\[
\text{dat}\_xw1 \leftarrow \text{dat}\_x \times \text{dat}\_w1
\]

\[
\text{dat}\_mlw4 \leftarrow \text{dat}\_m1 \times \text{dat}\_w4
\]

\[
\text{mod} <-
\]

\[
\text{m1} \sim a \times x + f1 \times w1 + d1 \times xw1
\]

\[
\text{y} \sim b \times m1 + f4 \times w4 + d4 \times mlw4 + cp \times x
\]

\[
\text{ind\_momome} := d1 \times d4
\]

\[
\text{fit} <- \text{sem(mod, dat, meanstructure = TRUE, fixed.x = FALSE, se = "none", baseline = FALSE)}
\]

\[
\text{est} <- \text{parameterEstimates(fit)}
\]

# See the example of index\_of\_mome on how to request bootstrap confidence interval.
\[
\text{out\_momome} <- \text{index\_of\_momome(x = "x", y = "y", m = "m1", w = "w1", fit = fit, boot\_ci = TRUE, R = 42, seed = 4314, parallel = FALSE, progress = FALSE)}
\]

\[
\text{out\_momome}
\]

\[
\text{coef(out\_momome)}
\]

\[
\text{print(est[32, ], nd = 8)}
\]

\[
\text{Indirect Effect (No Bootstrapping)}
\]
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
)
```

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()`.
- **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 compo-
   nent path. Set this to FALSE will suppress these warnings. Suppress them only
   when the moderators are omitted intentionally.

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 func-
tions 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

---

**lm2boot_out**

*Bootstrap Estimates for lm Outputs*

**Description**

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

**Usage**

```r
lm2boot_out(outputs, R = 100, seed = NULL, 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 bootstrapping. Default is NULL and seed is not set.
- **progress**: Whether progress will be displayed. Default is TRUE.

**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.

**See Also**

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

**Examples**

```r
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 \texttt{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 \texttt{lm()}, this function can combine them to an object of the class \texttt{lm_list} that represents a path model, as one fitted by structural equation model functions such as \texttt{lavaan::sem()}. This class of object can be used by some functions, such as \texttt{indirect_effect()}, \texttt{cond_indirect_effects()}, and \texttt{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 one 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 \texttt{lm_list}-class object that forms a path model represented by a set of regression models. This class has a \texttt{summary} method that shows the summary of each regression model stored (see \texttt{summary.lm_list()}), and a \texttt{print} method that prints the models stored (see \texttt{print.lm_list()}).

See Also

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

Examples

```r
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)
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

```r
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)
```

**Description**

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

**Usage**

```r
## 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` are not identical.
merge_mod_levels

Merge the Generated Levels of Moderators

Description

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

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

```r
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)
```
Usage

merge_mod_levels(...)

Arguments

... The output from \texttt{mod\_levels()}, or a list of levels generated by \texttt{mod\_levels\_list()}.

Details

It merges the levels of moderators generated by \texttt{mod\_levels()} into a data frame, with each row represents a combination of the levels. The output is to be used by \texttt{cond\_indirect\_effects()}.

Users usually do not need to use this function because \texttt{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 \texttt{mod\_levels\_list()} or the default levels in \texttt{cond\_indirect\_effects()}.

Value

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

See Also

\texttt{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 \texttt{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)
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.

Description

Create levels of moderators to be used by `indirect_effect()`, `cond INDIRECT_effects()`, and `cond INDIRECT()`. 

Usage

```r
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
)
```

```r
mod_levels_list(
```

```r
```
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.
- **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, .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.
For a categorical variable, the reference group is the group with all indicators equal to zero. If the label for the reference group 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 indirect effects; `merge_mod_levels()` for merging levels of moderators.

Examples

```r
library(lavaan)
data(data_med_mod_ab)
```
mod_levels

```r
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",
```

**plot.cond_indirect_effects**

Plot Conditional Effects

**Description**

Plot the conditional effects for different levels of moderators.

**Usage**

```r
## 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.

details
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
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.

See Also

- cond_indirect_effects()
predict.lm_from_lavaan_list

Details

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

This is an advanced helper used by \texttt{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

\texttt{lm_from_lavaan_list()}

Examples

\begin{verbatim}
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)
\end{verbatim}
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 \( x \rightarrow m1 \rightarrow m2 \rightarrow y \).
- **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

```r
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**
```r
## 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**
```r
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.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, ...)

Arguments

x The output of cond_indirect_diff().
digits The number of decimal places in the printout.
... Optional arguments. Ignored.

Details

The print method of the cond_indirect_diff-class object.

Value

It returns x invisibly. Called for its side effect.

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, ...)

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.
... Other arguments. Not used.

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

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.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, ...)

Arguments

x  
The output of indirect_effect() or cond_indirect().
digits  
Number of digits to display. Default is 3.
...
Other arguments. Not used.

Value

x is returned invisibly. Called for its side effect.

See Also

indirect_effect() and cond_indirect()

Examples

library(lavaan)
dat <- med.mod_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"
```r
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.lm_list**

*Print an lm_list-Class Object*

**Description**

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

**Usage**

```r
## 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)

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:

\[
\begin{align*}
fx &= x_1 + x_2 + x_3 \\
fm &= m_1 + m_2 + m_3 \\
fy &= y_1 + y_2 + y_3 \\
fm &\sim a \times fx \\
fy &\sim b \times fm + cp \times fx \\
\text{indirect} &:= a \times b
\end{align*}
\]

Extraction Methods for 'cond_indirect_effects' Outputs

Description

For subsetting a 'cond_indirect_effects'-class object.

Usage

## S3 method for class 'cond_indirect_effects'

```r
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

```r
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**

```r
## 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[2, ]
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.
**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**

```r
## 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**

```r
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)
```
Examples

```r
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)
```
Index

* datasets
  data_med, 20
  data_med_complicated, 21
  data_med_mod_a, 22
  data_med_mod_ab, 23
  data_med_mod_ab1, 24
  data_med_b, 25
  data_med_mod_b_mod, 26
  data_med_mod_parallel, 26
  data_med_mod_parallel_cat, 28
  data_med_mod_serial, 28
  data_med_mod_serial_cat, 29
  data_med_mod_serial_parallel, 30
  data_med_mod_serial_parallel_cat, 32
  data_mod, 33
  data_mod2, 33
  data_mod_cat, 34
  data_mome_demo, 35
  data_mome_demo_missing, 36
  data_parallel, 37
  data_sem, 38
  data_serial, 39
  data_serial_parallel, 40
  modmed_x1m3w4y1, 59
  simple_mediation_latent, 74
  +.indirect (math_indirect), 57
  -.indirect (math_indirect), 57
  [.cond_indirect_effects
     (subsetting_cond_indirect_effects), 75
  [.wlevels (subsetting_wlevels), 76
  [<-wlevels (subsetting_wlevels), 76
  [<<-wlevels (subsetting_wlevels), 76
  check_path, 3
  coef.cond_indirect_diff, 5
  coef.cond_indirect_diff(), 15, 49
  coef.cond_indirect_effects, 5
  coef.cond_indirect_effects(), 13
  coef.indirect, 7
  coef.indirect(), 13, 52
  coef.im_from_lavaan, 8
  cond_indirect, 9
  cond_indirect(), 7, 11–13, 19, 41, 42, 46, 48, 52, 54, 55, 57, 58, 60, 62, 72
  cond_indirect_diff, 15
  cond_indirect_diff(), 5, 15, 17, 70
  cond_indirect_effects, 46
  cond_indirect_effects (cond_indirect), 9
  cond_indirect_effects(), 5, 6, 11–13, 15–18, 41, 42, 44–46, 48, 49, 51–55, 59, 60, 62, 64–66, 71, 75
  confint.cond_indirect_diff, 17
  confint.cond_indirect_diff(), 15, 49
  confint.cond_indirect_effects, 17
  confint.cond_indirect_effects(), 13
  confint.indirect, 19
  confint.indirect(), 13, 52
  data_med, 20
  data_med_complicated, 21
  data_med_mod_a, 22
  data_med_mod_ab, 23
  data_med_mod_ab1, 24
  data_med_b, 25
  data_med_mod_b_mod, 26
  data_med_mod_parallel, 26
  data_med_mod_parallel_cat, 28
  data_med_mod_serial, 28
  data_med_mod_serial_cat, 29
  data_med_mod_serial_parallel, 30
  data_med_mod_serial_parallel_cat, 32
  data_mod, 33
  data_mod2, 33
  data_mod_cat, 34
  data_mome_demo, 35, 37
  data_mome_demo_missing, 36
  data_parallel, 37
  data_sem, 38

81
data_serial, 39
data_serial_parallel, 40
do_boot, 14, 41
do_boot(), 11, 13, 44, 45, 48, 53, 54
factor2var, 43
fit2boot_out, 44
fit2boot_out(), 42, 44
fit2boot_out_do_boot(fit2boot_out), 44
fit2boot_out_do_boot(), 42, 45
ggplot2, 65
ggplot2::geom_point(), 65
ggplot2::geom_segment(), 65
index_of_mome, 47
index_of_mome(), 15, 16, 49
index_of_momome(index_of_mome), 47
index_of_momome(), 16, 49
indirect_effect(cond indirect), 9
indirect_effect(), 7, 11, 13, 19, 41, 42, 44–46, 48, 51–55, 57, 58, 60, 72
indirect_i, 50
lavaan, 4, 11, 48
lavaan::lavaan, 3, 4, 11, 44, 48, 51, 61
lavaan::lavaan(), 56
lavaan::lavInspect(), 11, 51
lavaan::parameterEstimates(), 3, 11, 51
lavaan::sem(), 9, 12, 13, 41, 43–45, 54, 55
lm(), 3, 4, 9, 11, 12, 41, 48, 51, 53, 55, 61, 69, 78
lm2boot_out, 53
lm2boot_out(), 42
lm2list, 54
lm2list(), 41, 48, 53, 69, 73, 78
lm_from_lavaan_list, 56
lm_from_lavaan_list(), 8, 67, 68, 79
math_indirect, 13, 57
merge_mod_levels, 58
merge_mod_levels(), 12, 14, 16, 62, 77
mod_levels, 60
mod_levels(), 14, 16, 58, 59, 61, 62, 77
mod_levels_list(mod_levels), 60
mod_levels_list(), 12, 59, 61, 62, 77
modmed_x1m3w4y1, 59
parallel::detectCores(), 11, 41, 44
parallel::makeCluster(), 11, 41, 44
plot.cond_indirect_effects, 64
plot.cond indirect effects, 8, 56, 67, 68
predict.lm_from_lavaan, 66
predict.lm_from_lavaan_list, 56, 67
predict.lm_from_lavaan_list(), 56
predict.lm_list, 69
print.cond indirect diff, 70
print.cond indirect diff(), 15, 49
print.cond indirect effects, 71
print.cond indirect effects(), 13
print.indirect, 72
print.indirect(), 13, 52
print.lm, 73
print.lm_list, 55
print.summary lm list
(summary.lm_list), 78
print.summary_lm_list(), 78
simple_mediation latent, 74
stats::predict(), 69
stats::terms(), 79
subsetting cond indirect effects, 75
subsetting wlevels, 76
summary(), 78
summary.lm_list, 78
summary.lm_list(), 55, 78
terms.lm_from_lavaan, 79
terms.object, 79