Package 'modsem'

January 23, 2026

Type Package

Title Latent Interaction (and Moderation) Analysis in Structural Equation Models (SEM)

Version 1.0.16

Maintainer Kjell Solem Slupphaug <slupphaugkjell@gmail.com>

Description Estimation of interaction (i.e., moderation) effects between latent variables in structural equation models (SEM).
The supported methods are:
The constrained approach (Algina & Moulder, 2001).
The unconstrained approach (Marsh et al., 2004).
The residual centering approach (Little et al., 2006).
The double centering approach (Lin et al., 2010).
The latent moderated structural equations (LMS) approach (Klein & Moosbrugger, 2000).
The quasi-maximum likelihood (QML) approach (Klein & Muthén, 2007)
The constrained- unconstrained, residual- and double centering- approaches are estimated via 'lavaan' (Rosseel, 2012), whilst the LMS- and QML- approaches are estimated via 'modsem' it self. Alternatively model can be estimated via 'Mplus' (Muthén & Muthén, 1998-2017).
References:
Algina, J., & Moulder, B. C. (2001).
<doi:10.1207/S15328007SEM0801_3>.
``A note on estimating the Jöreskog-Yang model for latent variable interaction using 'LISREL' 8.3."
Klein, A., & Moosbrugger, H. (2000).
<doi:10.1007/BF02296338>.
``Maximum likelihood estimation of latent interaction effects with the LMS method."
Klein, A. G., & Muthén, B. O. (2007).
<doi:10.1080/00273170701710205>.
``Quasi-maximum likelihood estimation of structural equation models with multiple interaction and quadratic effects."
Lin, G. C., Wen, Z., Marsh, H. W., & Lin, H. S. (2010).
<doi:10.1080/10705511.2010.488999>.
``Structural equation models of latent interactions: Clarification of orthogonalizing and double-mean-centering strategies."
Little, T. D., Bovaird, J. A., & Widaman, K. F. (2006).

<doi:10.1207/s15328007sem1304_1>.
``On the merits of orthogonalizing powered and product terms: Implications for modeling interactions among latent variables."
Marsh, H. W., Wen, Z., & Hau, K. T. (2004).
<doi:10.1037/1082-989X.9.3.275>.
``Structural equation models of latent interactions: evaluation of alternative estimation strategies and indicator construction."
Muthén, L.K. and Muthén, B.O. (1998-2017).
``'Mplus' User's Guide. Eighth Edition."
<https://www.statmodel.com/>.
Rosseel Y (2012).
<doi:10.18637/jss.v048.i02>.
``'lavaan': An R Package for Structural Equation Modeling."

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, purrr, stringr, lavaan, rlang, MplusAutomation, nlme, dplyr, mvnfast, stats, fastGHQuad, mvtnorm, ggplot2, parallel, plotly, Deriv, MASS, Amelia, grDevices, cli, RhpcBLASctl

Depends R (>= 4.1.0)

URL https://modsem.org

Suggests knitr, rmarkdown, ggpubr, RColorBrewer

VignetteBuilder knitr

NeedsCompilation yes

Author Kjell Solem Slupphaug [aut, cre] (ORCID: <https://orcid.org/0009-0005-8324-2834>), Mehmet Mehmetoglu [ctb] (ORCID: <https://orcid.org/0000-0002-6092-8551>), Matthias Mittner [ctb] (ORCID: <https://orcid.org/0000-0003-0205-7353>)

Repository CRAN

Date/Publication 2026-01-23 13:10:02 UTC

Contents

`bootstrap_modsem` *Bootstrap a modsem Model*

Description

A generic interface for parametric and non-parametric bootstrap procedures for structural equation models estimated with the **modsem** ecosystem. The function dispatches on the class of `model`; currently dedicated methods exist for `modsem_pi` (product-indicator approach) and `modsem_da` (distributional-analytic approach).

Usage

```
bootstrap_modsem(model = modsem, FUN, ...)

## S3 method for class 'modsem_pi'
bootstrap_modsem(model, FUN = "coef", ...)

## S3 method for class 'modsem_da'
bootstrap_modsem(
  model,
  FUN = "coef",
  R = 1000L,
  P.max = 1e+05,
  type = c("nonparametric", "parametric"),
  verbose = interactive(),
  calc.se = FALSE,
  optimize = FALSE,
  ...
)

## S3 method for class '`function`'
bootstrap_modsem(
  model = modsem,
  FUN = "coef",
  data,
  R = 1000L,
  verbose = interactive(),
  FUN.args = list(),
  cluster.boot = NULL,
  ...
)
```

Arguments

model	A fitted modsem object, or a function to be bootstrapped (e.g., modsem, modsem_da and modsem_pi)
FUN	A function that returns the statistic of interest when applied to a fitted model. The function must accept a single argument, the model object, and should ideally return a numeric vector; see Value.
...	Additional arguments forwarded to lavaan::bootstrapLavaan for modsem_pi objects, or modsem_da for modsem_da objects.
R	number of bootstrap replicates.
P.max	ceiling for the simulated population size.
type	Bootstrap flavour, see Details.
verbose	Should progress information be printed to the console?
calc.se	Should standard errors for each replicate. Defaults to FALSE.
optimize	Should starting values be re-optimized for each replicate. Defaults to FALSE.

`data`	Dataset to be resampled.
`FUN.args`	Arguments passed to `FUN`
`cluster.boot`	Variable to cluster bootstrapping by

Details

A thin wrapper around `lavaan::bootstrapLavaan()` that performs the necessary book-keeping so that `FUN` receives a fully-featured `modsem_pi` object—rather than a bare `lavaan` fit—at every iteration.

The function internally resamples the observed data (non-parametric case) or simulates from the estimated parameter table (parametric case), feeds the sample to `modsem_da`, evaluates `FUN` on the refitted object and finally collates the results.

This is a more general version of `boostrap_modsem` for bootstrapping `modsem` functions, not modsem objects. `model` is now a function to be boostrapped, and `...` are now passed to the function (`model`), not `FUN`. To pass arguments to `FUN` use `FUN.args`.

Value

Depending on the return type of `FUN` either

numeric A matrix with `R` rows (bootstrap replicates) and as many columns as `length(FUN(model))`.

other A list of length `R`; each element is the raw output of `FUN`. **NOTE**: Only applies for `modsem_da` objects

Methods (by class)

- `bootstrap_modsem(modsem_pi)`: Bootstrap a `modsem_pi` model by delegating to `bootstrapLavaan`.
- `bootstrap_modsem(modsem_da)`: Parametric or non-parametric bootstrap for `modsem_da` models.
- `bootstrap_modsem(`function`)`: Non-parametric bootstrap of `modsem` functions

See Also

`bootstrapLavaan`, `modsem_pi`, `modsem_da`

Examples

```
m1 <- '
  X =~ x1 + x2
  Z =~ z1 + z2
  Y =~ y1 + y2

  Y ~ X + Z + X:Z
'

fit_pi <- modsem(m1, oneInt)
bootstrap_modsem(fit_pi, FUN = coef, R = 10L)
```

```
m1 <- '
  X =~ x1 + x2
  Z =~ z1 + z2
  Y =~ y1 + y2

  Y ~ X + Z + X:Z
'

## Not run:
fit_lms <- modsem(m1, oneInt, method = "lms")
bootstrap_modsem(fit_lms, FUN = coef, R = 10L)

## End(Not run)

tpb <- "
# Outer Model (Based on Hagger et al., 2007)
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model (Based on Steinmetz et al., 2011)
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC + INT:PBC
"

## Not run:
boot <- bootstrap_modsem(model = modsem,
                         model.syntax = tpb, data = TPB,
                         method = "dblcent", rcs = TRUE,
                         rcs.scale.corrected = TRUE,
                         FUN = "coef", R = 50L)
coef <- apply(boot, MARGIN = 2, FUN = mean, na.rm = TRUE)
se   <- apply(boot, MARGIN = 2, FUN = sd, na.rm = TRUE)

cat("Parameter Estimates:\n")
print(coef)

cat("Standard Errors: \n")
print(se)


## End(Not run)
```

`centered_estimates` *Get Centered Interaction Term Estimates*

Description

Computes centered estimates of model parameters. This is relevant when there is an interaction term in the model, as the simple main effects depend upon the mean structure of the structural model.

Currenlty only available for modsem_da and lavaan object. It is not relevant for the PI approaches (excluding the "pind" method, which is not recommended), since the indicators are centered before computing the product terms. The centering can be applied to observed variable interactions in lavaan models and latent interactions estimated using the sam function.

Usage

```
centered_estimates(object, ...)

## S3 method for class 'lavaan'
centered_estimates(
  object,
  monte.carlo = FALSE,
  mc.reps = 10000,
  tolerance.zero = 1e-10,
  ...
)

## S3 method for class 'modsem_da'
centered_estimates(
  object,
  monte.carlo = FALSE,
  mc.reps = 10000,
  tolerance.zero = 1e-10,
  ...
)
```

Arguments

object	An object of class modsem_da
...	Additional arguments passed to underlying methods. See specific method documentation for supported arguments, including:
monte.carlo	Logical. If TRUE, use Monte Carlo simulation to estimate standard errors; if FALSE, use the delta method (default).
mc.reps	Number of Monte Carlo repetitions. Default is 10000.
tolerance.zero	Threshold below which standard errors are set to NA.

Value

A data.frame with centered estimates in the est column.

Methods (by class)

- centered_estimates(lavaan): Method for lavaan objects
- centered_estimates(modsem_da): Method for modsem_da objects

Examples

```
m1 <- '
  # Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

  # Inner Model
  Y ~ X + Z + X:Z
'
## Not run:
est_lms <- modsem(m1, oneInt, method = "lms")
centered_estimates(est_lms)

## End(Not run)
```

`colorize_output` *Capture, colorise, and emit console text*

Description

Capture, colorise, and emit console text

Usage

```
colorize_output(
  expr,
  positive = MODSEM_COLORS$positive,
  negative = MODSEM_COLORS$negative,
  true = MODSEM_COLORS$true,
  false = MODSEM_COLORS$false,
  nan = MODSEM_COLORS$nan,
  na = MODSEM_COLORS$na,
  inf = MODSEM_COLORS$inf,
  string = MODSEM_COLORS$string,
  split = FALSE,
  append = "\n"
)
```

Arguments

`expr`	Expression or object with output which should be colorized.
`positive`	color of positive numbers.
`negative`	color of negative numbers.
`true`	color of `TRUE`.
`false`	color of `FALSE`.

nan	color of NaN.
na	color of NA.
inf	color of -Inf and Inf.
string	color of quoted strings.
split	Should output be splitted? If TRUE the output is printed normally (in real time) with no colorization, and the colored output is printed after the code has finished executing.
append	String appended after the colored output (default '\n').

Value

Invisibly returns the *plain* captured text.

Examples

```
set_modsem_colors(positive = "red3",
                  negative = "red3",
                  true = "darkgreen",
                  false = "red3",
                  na = "purple",
                  string = "darkgreen")

m1 <- "
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3
# Inner Model
  Y ~ X + Z + X:Z
"

est <- modsem(m1, data = oneInt)
colorize_output(summary(est))
colorize_output(est) # same as colorize_output(print(est))
colorize_output(modsem_inspect(est, what = "coef"))

## Not run:
colorize_output(split = TRUE, {
  # Get live (uncolored) output
  # And print colored output at the end of execution

  est_lms <- modsem(m1, data = oneInt, method = "lms")
  summary(est_lms)
})

colorize_output(modsem_inspect(est_lms))

## End(Not run)
```

compare_fit *compare model fit for* modsem *models*

Description

Compare the fit of two models using the likelihood ratio test (LRT). est_h0 is the null hypothesis model, and est_h1 the alternative hypothesis model. Importantly, the function assumes that est_h0 does not have more free parameters (i.e., degrees of freedom) than est_h1 (the alternative hypothesis model).

Usage

```
compare_fit(est_h1, est_h0, ...)
```

Arguments

est_h1	object of class modsem_da or modsem_pi representing the alternative hypothesis model (with interaction terms).
est_h0	object of class modsem_da or modsem_pi representing the null hypothesis model (without interaction terms).
...	additional arguments passed to the underlying comparison function. E.g., for modsem_pi models, this can be used to pass arguments to lavaan::lavTestLRT. currently only used for modsem_pi models.

Examples

```
## Not run:
m1 <- "
 # Outer Model
 X =~ x1 + x2 + x3
 Y =~ y1 + y2 + y3
 Z =~ z1 + z2 + z3

 # Inner model
 Y ~ X + Z + X:Z
"

# LMS approach
est_h1 <- modsem(m1, oneInt, "lms")
est_h0 <- estimate_h0(est_h1, calc.se=FALSE) # std.errors are not needed
compare_fit(est_h1 = est_h1, est_h0 = est_h0)

# Double centering approach
est_h1 <- modsem(m1, oneInt, method = "dblcent")
est_h0 <- estimate_h0(est_h1, oneInt)

compare_fit(est_h1 = est_h1, est_h0 = est_h0)
```

```
# Constrained approach
est_h1 <- modsem(m1, oneInt, method = "ca")
est_h0 <- estimate_h0(est_h1, oneInt)

compare_fit(est_h1 = est_h1, est_h0 = est_h0)

## End(Not run)
```

`default_settings_da` *default arguments fro LMS and QML approach*

Description

This function returns the default settings for the LMS and QML approach.

Usage

```
default_settings_da(method = c("lms", "qml"))
```

Arguments

`method`	which method to get the settings for

Value

list

Examples

```
library(modsem)
default_settings_da()
```

`default_settings_pi` *default arguments for product indicator approaches*

Description

This function returns the default settings for the product indicator approaches

Usage

```
default_settings_pi(method = c("rca", "uca", "pind", "dblcent", "ca"))
```

Arguments

`method`	which method to get the settings for

Value

list

Examples

```
library(modsem)
default_settings_pi()
```

`estimate_h0` *Estimate baseline model for* `modsem` *models*

Description

Estimates a baseline model (H0) from a given model (H1). The baseline model is estimated by removing all interaction terms from the model.

Usage

```
estimate_h0(object, warn_no_interaction = TRUE, ...)

## S3 method for class 'modsem_da'
estimate_h0(object, warn_no_interaction = TRUE, ...)

## S3 method for class 'modsem_pi'
estimate_h0(object, warn_no_interaction = TRUE, reduced = TRUE, ...)
```

Arguments

`object`	An object of class `modsem_da` or `modsem_pi`.
`warn_no_interaction`	Logical. If 'TRUE', a warning is issued if no interaction terms are found in the model.
`...`	Additional arguments passed to the 'modsem_da' function, overriding the arguments in the original model.
`reduced`	Should the baseline model be a reduced version of the model? If `TRUE`, the latent product term and its (product) indicators are kept in the model, but the interaction coefficients are constrained to zero. If `FALSE`, the interaction terms are removed completely from the model. Note that the models will no longer be nested, if the interaction terms are removed from the model completely.

Methods (by class)

- `estimate_h0(modsem_da)`: Estimate baseline model for `modsem_da` objects
- `estimate_h0(modsem_pi)`: Estimate baseline model for `modsem_pi` objects

Examples

```
## Not run:
m1 <- "
 # Outer Model
 X =~ x1 + x2 + x3
 Y =~ y1 + y2 + y3
 Z =~ z1 + z2 + z3

 # Inner model
 Y ~ X + Z + X:Z
"

# LMS approach
est_h1 <- modsem(m1, oneInt, "lms")
est_h0 <- estimate_h0(est_h1, calc.se=FALSE) # std.errors are not needed
compare_fit(est_h1 = est_h1, est_h0 = est_h0)

# Double centering approach
est_h1 <- modsem(m1, oneInt, method = "dblcent")
est_h0 <- estimate_h0(est_h1, oneInt)

compare_fit(est_h1 = est_h1, est_h0 = est_h0)

# Constrained approach
est_h1 <- modsem(m1, oneInt, method = "ca")
est_h0 <- estimate_h0(est_h1, oneInt)

compare_fit(est_h1 = est_h1, est_h0 = est_h0)

## End(Not run)
```

`extract_lavaan` *extract lavaan object from modsem object estimated using product indicators*

Description

extract lavaan object from modsem object estimated using product indicators

Usage

```
extract_lavaan(object)
```

Arguments

`object`	modsem object

Value

lavaan object

Examples

```
library(modsem)
m1 <- '
  # Outer Model
  X =~ x1 + x2 + x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'
est <- modsem_pi(m1, oneInt)
lav_est <- extract_lavaan(est)
```

`fit_modsem_da` *Fit measures for QML and LMS models*

Description

Calculates chi-sq test and p-value, as well as RMSEA for the LMS and QML models. Note that the Chi-Square based fit measures should be calculated for the baseline model, i.e., the model without the interaction effect

Usage

```
fit_modsem_da(
  model,
  chisq = TRUE,
  lav.fit = FALSE,
  drop.list.single.group = TRUE
)
```

Arguments

`model`	fitted model. Thereafter, you can use 'compare_fit()' to assess the comparative fit of the models. If the interaction effect makes the model better, and e.g., the RMSEA is good for the baseline model, the interaction model likely has a good RMSEA as well.
`chisq`	should Chi-Square based fit-measures be calculated?
`lav.fit`	Should fit indices from the `lavaan` model used to optimize the starting parameters be included (if available)? This is usually only approprioate for linear models (i.e., no interaction effects), where the parameter estimates for LMS and QML are equivalent to ML estimates from lavaan.
`drop.list.single.group`	Logical. If FALSE, (some) results are returned as a list, where each element corresponds to a group (even if there is only a single group). If TRUE, the list will be unlisted if there is only a single group

get_pi_data *Get data with product indicators for different approaches*

Description

get_pi_data() is a function for creating a dataset with product indiactors used for estimating latent interaction models using one of the product indicator approaches.

Usage

```
get_pi_data(model.syntax, data, method = "dblcent", match = FALSE, ...)
```

Arguments

model.syntax	lavaan syntax
data	data to create product indicators from
method	method to use: "rca" = residual centering approach, "uca" = unconstrained approach, "dblcent" = double centering approach, "pind" = prod ind approach, with no constraints or centering, "custom" = use parameters specified in the function call
match	should the product indicators be created by using the match-strategy
...	arguments passed to other functions (e.g., modsem_pi)

Value

data.frame

Examples

```
library(modsem)
library(lavaan)
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'
syntax <- get_pi_syntax(m1)
data <- get_pi_data(m1, oneInt)
est <- sem(syntax, data)
summary(est)
```

get_pi_syntax *Get* lavaan *syntax for product indicator approaches*

Description

`get_pi_syntax()` is a function for creating the `lavaan` syntax used for estimating latent interaction models using one of the product indicator approaches.

Usage

```
get_pi_syntax(
  model.syntax,
  method = "dblcent",
  match = FALSE,
  data = NULL,
  ...
)
```

Arguments

`model.syntax`	`lavaan` syntax
`method`	method to use: `"rca"` = residual centering approach, `"uca"` = unconstrained approach, `"dblcent"` = double centering approach, `"pind"` = prod ind approach, with no constraints or centering, `"custom"` = use parameters specified in the function call
`match`	should the product indicators be created by using the match-strategy
`data`	Optional. Dataset to use, usually not relevant.
`...`	arguments passed to other functions (e.g., modsem_pi)

Value

`character` vector

Examples

```
library(modsem)
library(lavaan)
m1 <- '
  # Outer Model
  X =~ x1 + x2 + x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'
syntax <- get_pi_syntax(m1)
```

```
data <- get_pi_data(m1, oneInt)
est <- sem(syntax, data)
summary(est)
```

`is_interaction_model` *Check if model object has interaction terms*

Description

Check if model object has interaction terms

Usage

```
is_interaction_model(object)
```

Arguments

`object`	An object of class `modsem_pi` or `modsem_da`, respectively.

Value

Logical. `TRUE` if the model has an interaction term, otherwise it returns `FALSE`.

Examples

```
m1 <- '
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

# Inner Model
  Y ~ X + Z + X:Z
'

est_dca <- modsem(m1, oneInt, method = "dblcent")
is_interaction_model(est_dca)

## Not run:
est_lms <- modsem(m1, oneInt, method = "lms")
is_interaction_model(est_lms)

## End(Not run)
```

jordan *Jordan subset of PISA 2006 data*

Description

The data stem from the large-scale assessment study PISA 2006 (Organisation for Economic Co-Operation and Development, 2009) where competencies of 15-year-old students in reading, mathematics, and science are assessed using nationally representative samples in 3-year cycles. In this example, data from the student background questionnaire from the Jordan sample of PISA 2006 were used. Only data of students with complete responses to all 15 items (N = 6,038) were considered.

Format

A data frame of fifteen variables and 6,038 observations:

enjoy1 indicator for enjoyment of science, item ST16Q01: I generally have fun when I am learning <broad science> topics.

enjoy2 indicator for enjoyment of science, item ST16Q02: I like reading about <broad science>.

enjoy3 indicator for enjoyment of science, item ST16Q03: I am happy doing <broad science> problems.

enjoy4 indicator for enjoyment of science, item ST16Q04: I enjoy acquiring new knowledge in <broad science>.

enjoy5 indicator for enjoyment of science, item ST16Q05: I am interested in learning about <broad science>.

academic1 indicator for academic self-concept in science, item ST37Q01: I can easily understand new ideas in <school science>.

academic2 indicator for academic self-concept in science, item ST37Q02: Learning advanced <school science> topics would be easy for me.

academic3 indicator for academic self-concept in science, item ST37Q03: I can usually give good answers to <test questions> on <school science> topics.

academic4 indicator for academic self-concept in science, item ST37Q04: I learn <school science> topics quickly.

academic5 indicator for academic self-concept in science, item ST37Q05: <School science> topics are easy for me.

academic6 indicator for academic self-concept in science, item ST37Q06: When I am being taught <school science>, I can understand the concepts very well.

career1 indicator for career aspirations in science, item ST29Q01: I would like to work in a career involving <broad science>.

career2 indicator for career aspirations in science, item ST29Q02: I would like to study <broad science> after <secondary school>.

career3 indicator for career aspirations in science, item ST29Q03: I would like to spend my life doing advanced <broad science>.

career4 indicator for career aspirations in science, item ST29Q04: I would like to work on <broad science> projects as an adult.

Source

This version of the dataset, as well as the description was gathered from the documentation of the 'nlsem' package (https://cran.r-project.org/package=nlsem), where the only difference is that the names of the variables were changed

Originally the dataset was gathered by the Organisation for Economic Co-Operation and Development (2009). Pisa 2006: Science competencies for tomorrow's world (Tech. Rep.). Paris, France. Obtained from: https://www.oecd.org/pisa/pisaproducts/database-pisa2006.htm

Examples

```
## Not run:
m1 <- "
  ENJ =~ enjoy1 + enjoy2 + enjoy3 + enjoy4 + enjoy5
  CAREER =~ career1 + career2 + career3 + career4
  SC =~ academic1 + academic2 + academic3 + academic4 + academic5 + academic6
  CAREER ~ ENJ + SC + ENJ:ENJ + SC:SC + ENJ:SC
"

est <- modsem(m1, data = jordan, method = "qml")
summary(est)

## End(Not run)
```

`modsem` *Estimate interaction effects in structural equation models (SEMs)*

Description

`modsem()` is a function for estimating interaction effects between latent variables in structural equation models (SEMs). Methods for estimating interaction effects in SEMs can basically be split into two frameworks:

1. Product Indicator (PI) based approaches (`"dblcent"`, `"rca"`, `"uca"`, `"ca"`, `"pind"`)
2. Distributionally (DA) based approaches (`"lms"`, `"qml"`).

For the product indicator-based approaches, `modsem()` is essentially a fancy wrapper for `lavaan::sem()` which generates the necessary syntax and variables for the estimation of models with latent product indicators.

The distributionally based approaches are implemented separately and are not estimated using `lavaan::sem()`, but rather using custom functions (largely written in `C++` for performance reasons). For greater control, it is advised that you use one of the sub-functions (modsem_pi, modsem_da, modsem_mplus) directly, as passing additional arguments to them via `modsem()` can lead to unexpected behavior.

Usage

```
modsem(model.syntax = NULL, data = NULL, method = "dblcent", ...)
```

Arguments

model.syntax	lavaan syntax
data	dataframe
method	method to use: "dblcent" double centering approach (passed to lavaan). "ca" constrained approach (passed to lavaan). "rca" residual centering approach (passed to lavaan). "uca" unconstrained approach (passed to lavaan). "pind" prod ind approach, with no constraints or centering (passed to lavaan). "lms" latent moderated structural equations (not passed to lavaan). "qml" quasi maximum likelihood estimation (not passed to lavaan). "custom" use parameters specified in the function call (passed to lavaan). "mplus" estimate model through Mplus.
...	arguments passed to other functions depending on the method (see modsem_pi, modsem_da, and modsem_mplus)

Value

modsem object with class modsem_pi, modsem_da, or modsem_mplus

Examples

```
library(modsem)
# For more examples, check README and/or GitHub.
# One interaction
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'

# Double centering approach
est1 <- modsem(m1, oneInt)
summary(est1)

## Not run:
# The Constrained Approach
est1_ca <- modsem(m1, oneInt, method = "ca")
summary(est1_ca)

# LMS approach
est1_lms <- modsem(m1, oneInt, method = "lms")
summary(est1_lms)
```

```
# QML approach
est1_qml <- modsem(m1, oneInt, method = "qml")
summary(est1_qml)

## End(Not run)

# Theory Of Planned Behavior
tpb <- '
# Outer Model (Based on Hagger et al., 2007)
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model (Based on Steinmetz et al., 2011)
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  BEH ~ INT:PBC
'

# Double centering approach
est_tpb <- modsem(tpb, data = TPB)
summary(est_tpb)

## Not run:
# The Constrained Approach
est_tpb_ca <- modsem(tpb, data = TPB, method = "ca")
summary(est_tpb_ca)

# LMS approach
est_tpb_lms <- modsem(tpb, data = TPB, method = "lms", nodes = 32)
summary(est_tpb_lms)

# QML approach
est_tpb_qml <- modsem(tpb, data = TPB, method = "qml")
summary(est_tpb_qml)

## End(Not run)
```

`modsemify` *Generate parameter table for* `lavaan` *syntax*

Description

Generate parameter table for `lavaan` syntax

Usage

```
modsemify(syntax)
```

Arguments

syntax	model syntax

Value

data.frame with columns lhs, op, rhs, mod

Examples

```
library(modsem)
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'
modsemify(m1)
```

---

modsem_coef *Wrapper for coef*

Description

wrapper for coef, to be used with modsem::modsem_coef, since coef is not in the namespace of modsem, but stats.

Usage

```
modsem_coef(object, ...)
```

Arguments

object	fitted model to inspect
...	additional arguments

modsem_da *Interaction between latent variables using LMS and QML approaches*

Description

modsem_da() is a function for estimating interaction effects between latent variables in structural equation models (SEMs) using distributional analytic (DA) approaches. Methods for estimating interaction effects in SEMs can basically be split into two frameworks: 1. Product Indicator-based approaches ("dblcent", "rca", "uca", "ca", "pind") 2. Distributionally based approaches ("lms", "qml").

modsem_da() handles the latter and can estimate models using both QML and LMS, necessary syntax, and variables for the estimation of models with latent product indicators.

NOTE: Run default_settings_da to see default arguments.

Usage

```
modsem_da(
  model.syntax = NULL,
  data = NULL,
  group = NULL,
  method = "lms",
  verbose = NULL,
  optimize = NULL,
  nodes = NULL,
  missing = NULL,
  convergence.abs = NULL,
  convergence.rel = NULL,
  optimizer = NULL,
  center.data = NULL,
  standardize.data = NULL,
  standardize.out = NULL,
  standardize = NULL,
  mean.observed = NULL,
  cov.syntax = NULL,
  double = NULL,
  calc.se = NULL,
  FIM = NULL,
  EFIM.S = NULL,
  OFIM.hessian = NULL,
  EFIM.parametric = NULL,
  robust.se = NULL,
  R.max = NULL,
  max.iter = NULL,
  max.step = NULL,
  start = NULL,
  epsilon = NULL,
```

```
  quad.range = NULL,
  adaptive.quad = NULL,
  adaptive.frequency = NULL,
  adaptive.quad.tol = NULL,
  n.threads = NULL,
  algorithm = NULL,
  em.control = NULL,
  ordered = NULL,
  ordered.probit.correction = FALSE,
  cluster = NULL,
  cr1s = FALSE,
  sampling.weights = NULL,
  sampling.weights.normalization = NULL,
  rcs = FALSE,
  rcs.choose = NULL,
  rcs.scale.corrected = TRUE,
  orthogonal.x = NULL,
  orthogonal.y = NULL,
  auto.fix.first = NULL,
  auto.fix.single = NULL,
  auto.split.syntax = NULL,
  ...
)
```

Arguments

model.syntax	lavaan syntax
data	A dataframe with observed variables used in the model.
group	Character. A variable name in the data frame defining the groups in a multiple group analysis
method	method to use: "lms" latent moderated structural equations (not passed to lavaan). "qml" quasi maximum likelihood estimation (not passed to lavaan).
verbose	should estimation progress be shown
optimize	should starting parameters be optimized
nodes	number of quadrature nodes (points of integration) used in lms, increased number gives better estimates but slower computation. How many are needed depends on the complexity of the model. For simple models, somewhere between 16-24 nodes should be enough; for more complex models, higher numbers may be needed. For models where there is an interaction effect between an endogenous and exogenous variable, the number of nodes should be at least 32, but practically (e.g., ordinal/skewed data), more than 32 is recommended. In cases where data is non-normal, it might be better to use the qml approach instead. You can also consider setting adaptive.quad = TRUE.
missing	How should missing values be handled? If "listwise" (default) missing values are removed list-wise (alias: "complete" or "casewise"). If impute values

are imputed using `Amelia::amelia`. If `"fiml"` (alias: `"ml"` or `"direct"`), full information maximum likelihood (FIML) is used. FIML can be (very) computationally intensive.

`convergence.abs`
: Absolute convergence criterion. Lower values give better estimates but slower computation. Not relevant when using the QML approach. For the LMS approach the EM-algorithm stops whenever the relative or absolute convergence criterion is reached.

`convergence.rel`
: Relative convergence criterion. Lower values give better estimates but slower computation. For the LMS approach the EM-algorithm stops whenever the relative or absolute convergence criterion is reached.

`optimizer`
: optimizer to use, can be either `"nlminb"` or `"L-BFGS-B"`. For LMS, `"nlminb"` is recommended. For QML, `"L-BFGS-B"` may be faster if there is a large number of iterations, but slower if there are few iterations.

`center.data`
: should data be centered before fitting model

`standardize.data`
: should data be scaled before fitting model, will be overridden by `standardize` if `standardize` is set to `TRUE`.

`standardize.out`
: should output be standardized (note will alter the relationships of parameter constraints since parameters are scaled unevenly, even if they have the same label). This does not alter the estimation of the model, only the output.

  NOTE: It is recommended that you estimate the model normally and then standardize the output using `standardize_model`, `standardized_estimates` or `summary(<modsem_da-object>, standardize=TRUE)`.

`standardize`
: will standardize the data before fitting the model, remove the mean structure of the observed variables, and standardize the output. Note that `standardize.data`, `mean.observed`, and `standardize.out` will be overridden by `standardize` if `standardize` is set to `TRUE`.

  NOTE: It is recommended that you estimate the model normally and then standardize the output using `standardized_estimates`.

`mean.observed`
: should the mean structure of the observed variables be estimated? This will be overridden by `standardize`, if `standardize` is set to `TRUE`.

  NOTE: Not recommended unless you know what you are doing.

`cov.syntax`
: model syntax for implied covariance matrix of exogenous latent variables (see `vignette("interaction_two_etas", "modsem")`).

`double`
: try to double the number of dimensions of integration used in LMS, this will be extremely slow but should be more similar to `mplus`.

`calc.se`
: should standard errors be computed? **NOTE**: If `FALSE`, the information matrix will not be computed either.

`FIM`
: should the Fisher information matrix be calculated using the observed or expected values? Must be either `"observed"` or `"expected"`.

`EFIM.S`
: if the expected Fisher information matrix is computed, `EFIM.S` selects the number of Monte Carlo samples. Defaults to 100. **NOTE**: This number should

likely be increased for better estimates (e.g., 1000), but it might drasticly increase computation time.

`OFIM.hessian` Logical. If `TRUE` (default), standard errors are based on the negative Hessian (observed Fisher information). If `FALSE`, they come from the outer product of individual score vectors (OPG). For correctly specified models, these two matrices are asymptotically equivalent; yielding nearly identical standard errors in large samples. The Hessian usually shows smaller finite-sample variance (i.e., it's more consistent), and is therefore the default.

Note, that the Hessian is not always positive definite, and is more computationally expensive to calculate. The OPG should always be positive definite, and a lot faster to compute. If the model is correctly specified, and the sample size is large, then the two should yield similar results, and switching to the OPG can save a lot of time. Note, that the required sample size depends on the complexity of the model.

A large difference between Hessian and OPG suggests misspecification, and `robust.se = TRUE` should be set to obtain sandwich (robust) standard errors.

`EFIM.parametric`

should data for calculating the expected Fisher information matrix be simulated parametrically (simulated based on the assumptions and implied parameters from the model), or non-parametrically (stochastically sampled)? If you believe that normality assumptions are violated, `EFIM.parametric = FALSE` might be the better option.

`robust.se` should robust standard errors be computed, using the sandwich estimator?

`R.max` Maximum population size (not sample size) used in the calculated of the expected fischer information matrix.

`max.iter` maximum number of iterations.

`max.step` maximum steps for the M-step in the EM algorithm (LMS).

`start` starting parameters.

`epsilon` finite difference for numerical derivatives.

`quad.range` range in z-scores to perform numerical integration in LMS using, when using quasi-adaptive Gaussian-Hermite Quadratures. By default `Inf`, such that `f(t)` is integrated from -Inf to Inf, but this will likely be inefficient and pointless at a large number of nodes. Nodes outside `+/- quad.range` will be ignored.

`adaptive.quad` should a quasi adaptive quadrature be used? If `TRUE`, the quadrature nodes will be adapted to the data. If `FALSE`, the quadrature nodes will be fixed. Default is `FALSE`. The adaptive quadrature does not fit an adaptive quadrature to each participant, but instead tries to place more nodes where posterior distribution is highest. Compared with a fixed Gauss Hermite quadrature this usually means that less nodes are placed at the tails of the distribution.

`adaptive.frequency`

How often should the quasi-adaptive quadrature be calculated? Defaults to 3, meaning that it is recalculated every third EM-iteration.

`adaptive.quad.tol`

Relative error tolerance for quasi adaptive quadrature. Defaults to `1e-12`.

`n.threads` number of threads to use for parallel processing. If `NULL`, it will use <= 2 threads. If an integer is specified, it will use that number of threads (e.g., `n.threads = 4` will use 4 threads). If `"default"`, it will use the default number of threads (2). If `"max"`, it will use all available threads, `"min"` will use 1 thread.

`algorithm` algorithm to use for the EM algorithm. Can be either `"EM"` or `"EMA"`. `"EM"` is the standard EM algorithm. `"EMA"` is an accelerated EM procedure that uses Quasi-Newton and Fisher Scoring optimization steps when needed. Default is `"EM"`.

`em.control` a list of control parameters for the EM algorithm. See `default_settings_da` for defaults.

`ordered` Variables to be treated as ordered. Categories for ordered variables are scored, transforming them from ordinal scale to interval scale (Chen & Wang, 2014). The underlying continous distributions are estimated analytically for indicators of exogenous variables, and using an ordered probit regression for indicators of endogenous variables. Factor scores are used as independent variables the ordered probit regressions. Interaction effects between the factor scores are included in the probit regression, if applicable. The estimates are more robust to unequal intervals in ordinal variables. I.e., the estimates should be more consistent, and less biased.

`ordered.probit.correction` Should ordered indicators be transformed such that they reproduce their (probit) polychoric correlation matrix? This can be useful for ordered variables with only a few categories, or for linear models.

`cluster` Clusters used to compute standard errors robust to non-indepence of observations. Must be paired with `robust.se = TRUE`.

`cr1s` Logical; if `TRUE`, apply the CR1S small-sample correction factor to the cluster-robust variance estimator. The CR1S factor is $(G/(G-1)) \cdot ((N-1)/(N-q))$, where G is the number of clusters, N is the total number of observations, and q is the number of free parameters. This adjustment inflates standard errors to reduce the small-sample downward bias present in the basic cluster-robust (CR0) estimator, especially when G is small. If `FALSE`, the unadjusted CR0 estimator is used. Defaults to `TRUE`. Only relevant if `cluster` is specified.

`sampling.weights` A variable name in the data frame containing sampling weight information. Depending on the sampling.weights.normalization argument, these weights may be rescaled (or not) so that their sum equals the number of observations (total or per group)

`sampling.weights.normalization` If `"none"`, the sampling weights (if provided) will not be transformed. If `"total"`, the sampling weights are normalized by dividing by the total sum of the weights, and multiplying again by the total sample size. If `"group"`, the sampling weights are normalized per group: by dividing by the sum of the weights (in each group), and multiplying again by the group size. The default is `"total"`.

`rcs` Should latent variable indicators be replaced with reliability-corrected single item indicators instead? See `relcorr_single_item`.

`rcs.choose` Which latent variables should get their indicators replaced with reliability-corrected single items? It is passed to `relcorr_single_item` as the `choose` argument.

`rcs.scale.corrected`
: Should reliability-corrected items be scale-corrected? If `TRUE` reliability-corrected single items are corrected for differences in factor loadings between the items. Default is `TRUE`.

`orthogonal.x`
: If `TRUE`, all covariances among exogenous latent variables only are set to zero. Default is `FALSE`.

`orthogonal.y`
: If `TRUE`, all covariances among endogenous latent variables only are set to zero. If `FALSE` residual covariances are added between pure endogenous variables; those that are predicted by no other endogenous variable in the structural model. Default is `FALSE`.

`auto.fix.first`
: If `TRUE` the factor loading of the first indicator, for a given latent variable is fixed to `1`. If `FALSE` no loadings are fixed (automatically). Note that that this might make it such that the model no longer is identified. Default is `TRUE`. **NOTE** this behaviour is overridden if the first loading is labelled, where it gets treated as a free parameter instead. This differs from the default behaviour in `lavaan`.

`auto.fix.single`
: If `TRUE`, the residual variance of an observed indicator is set to zero if it is the only indicator of a latent variable. If `FALSE` the residual variance is not fixed to zero, and treated as a free parameter of the model. Default is `TRUE`. **NOTE** this behaviour is overridden if the first loading is labelled, where it gets treated as a free parameter instead.

`auto.split.syntax`
: Should the model syntax automatically be split into a linear and non-linear part? This is done by moving the structural model for linear endogenous variables (used in interaction terms) into the `cov.syntax` argument. This can potentially allow interactions between two endogenous variables given that both are linear (i.e., not affected by interaction terms). This is `FALSE` by default for the LMS approach. When using the QML approach interation effects between exogenous and endogenous variables can in some cases be biased, if the model is not split beforehand. The default is therefore `TRUE` for the QML approach.

`...`
: additional arguments to be passed to the estimation function.

Value

`modsem_da` object

Examples

```
library(modsem)
# For more examples, check README and/or GitHub.
# One interaction
m1 <- "
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
```

```
  Y ~ X + Z + X:Z
"

## Not run:
# QML Approach
est_qml <- modsem_da(m1, oneInt, method = "qml")
summary(est_qml)

# Theory Of Planned Behavior
tpb <- "
# Outer Model (Based on Hagger et al., 2007)
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model (Based on Steinmetz et al., 2011)
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  BEH ~ INT:PBC
"

# LMS Approach
est_lms <- modsem_da(tpb, data = TPB, method = "lms")
summary(est_lms)

## End(Not run)
```

`modsem_inspect` *Inspect model information*

Description

function used to inspect fitted object. Similar to `lavaan::lavInspect` argument `what` decides what to inspect

`modsem_inspect.modsem_da` Lets you pull matrices, optimiser diagnostics, expected moments, or fit measures from a `modsem_da` object.

Usage

```
modsem_inspect(object, what = NULL, ...)

## S3 method for class 'lavaan'
modsem_inspect(object, what = "free", ...)

## S3 method for class 'modsem_da'
modsem_inspect(object, what = NULL, ...)
```

```
## S3 method for class 'modsem_pi'
modsem_inspect(object, what = "free", ...)
```

Arguments

`object`	A fitted object of class `"modsem_da"`.
`what`	Character scalar selecting what to return (see *Details*). If `NULL` the value `"default"` is used.
`...`	Passed straight to `modsem_inspect_da()`.

Details

For `modsem_pi` objects, it is just a wrapper for `lavaan::lavInspect`. For `modsem_da` objects an internal function is called, which takes different keywords for the `what` argument.

Below is a list of possible values for the `what` argument, organised in several sections. Keywords are *case-sensitive*.

Presets

`"default"` Everything in *Sample information*, *Optimiser diagnostics Parameter tables*, *Model matrices*, and *Expected-moment matrices* except the raw `data` slot

`"coef"` Coefficients and variance-covariance matrix of both free and constrained parameters (same as `"coef.all"`).

`"coef.all"` Coefficients and variance-covariance matrix of both free and constrained parameters (same as `"coef"`).

`"coef.free"` Coefficients and variance-covariance matrix of the free parameters.

`"all"` All items listed below, including `data`.

`"matrices"` The model matrices.

`"optim"` Only the items under *Optimiser diagnostics*.

`"fit"` A list with `fit.h0`, `fit.h1`, comparative.fit

Sample information:

`"N"` Number of analysed rows (integer).

`"ngroups"` Number of groups in model (integer).

`"group"` Group variable in model (character).

`"group.label"` Group labels (character).

`"ovs"` Observed variables used in model (character).

Parameter estimates and standard errors:

`"coefficients.free"` Free parameter values.

`"coefficients.all"` Both free and constrained parameter values.

`"vcov.free"` Variance–covariance of free coefficients only.

`"vcov.all"` Variance–covariance of both free and constrained coefficients.

Optimiser diagnostics:

`"coefficients.free"` Free parameter values.

`"vcov.free"` Variance–covariance of free coefficients only.

`"information"` Fisher information matrix.

`"loglik"` Log-likelihood.

`"iterations"` Optimiser iteration count.

`"convergence"` `TRUE`/`FALSE` indicating whether the model converged.

Parameter tables:

`"partable"` Parameter table with estimated parameters.

`"partable.input"` Parsed model syntax.

Model matrices:

`"lambda"` Λ – Factor loadings.

`"tau"` τ – Intercepts for indicators.

`"theta"` Θ – Residual (Co-)Variances for indicators.

`"gamma.xi"` Γ_ξ – Structural coefficients between exogenous and endogenous variables.

`"gamma.eta"` Γ_η – Structural coefficients between endogenous variables.

`"omega.xi.xi"` $\Omega_{\xi\xi}$ – Interaction effects between exogenous variables

`"omega.eta.xi"` $\Omega_{\eta\xi}$ – Interaction effects between exogenous and endogenous variables

`"phi"` Φ – (Co-)Variances among exogenous variables.

`"psi"` Ψ – Residual (co-)variances among engoenous variables.

`"alpha"` α – Intercepts for endogenous variables

`"beta0"` β_0 – Intercepts for exogenous variables

Model-implied matrices:

`"cov.ov"` Model-implied covariance of observed variables.

`"cov.lv"` Model-implied covariance of latent variables.

`"cov.all"` Joint covariance of observed + latent variables.

`"cor.ov"` Correlation counterpart of `"cov.ov"`.

`"cor.lv"` Correlation counterpart of `"cov.lv"`.

`"cor.all"` Correlation counterpart of `"cov.all"`.

`"mean.ov"` Expected means of observed variables.

`"mean.lv"` Expected means of latent variables.

`"mean.all"` Joint mean vector.

R-squared and standardized residual variances:

`"r2.all"` R-squared values for both observed (i.e., indicators) and latent endogenous variables.

`"r2.lv"` R-squared values for latent endogenous variables.

`"r2.ov"` R-squared values for observed (i.e., indicators) variables.

`"res.all"` Standardized residuals (i.e., `1 - R^2`) for both observed (i.e., indicators) and latent endogenous variables.

`"res.lv"` Standardized residuals (i.e., `1 - R^2`) for latent endogenous variables.

`"res.ov"` Standardized residuals (i.e., `1 - R^2`) for observed variables (i.e., indicators).

Interaction-specific caveats:

- If the model contains an *uncentred* latent interaction term it is centred internally before any `cov.*`, `cor.*`, or `mean.*` matrices are calculated.
- These matrices should not be used to compute fit-statistics (e.g., chi-square and RMSEA) if there is an interaction term in the model.

Value

A named list with the extracted information. If a single piece of information is returned, it is returned as is; not as a named element in a list.

Methods (by class)

- `modsem_inspect(lavaan)`: Inspect a `lavaan` object
- `modsem_inspect(modsem_da)`: Inspect a `modsem_da` object
- `modsem_inspect(modsem_pi)`: Inspect a `modsem_pi` object

Examples

```
## Not run:
m1 <- "
 # Outer Model
 X =~ x1 + x2 + x3
 Y =~ y1 + y2 + y3
 Z =~ z1 + z2 + z3

 # Inner model
 Y ~ X + Z + X:Z
"

est <- modsem(m1, oneInt, "lms")

modsem_inspect(est) # everything except "data"
modsem_inspect(est, what = "optim")
modsem_inspect(est, what = "phi")

## End(Not run)
```

`modsem_mimpute` *Estimate a* `modsem` *model using multiple imputation*

Description

Estimate a `modsem` model using multiple imputation

Usage

```
modsem_mimpute(
  model.syntax,
  data,
  method = "lms",
  m = 25,
  verbose = interactive(),
  se = c("simple", "full"),
  ...
)
```

Arguments

`model.syntax`	`lavaan` syntax
`data`	A dataframe with observed variables used in the model.
`method`	Method to use: `"lms"` latent moderated structural equations (not passed to `lavaan`). `"qml"` quasi maximum likelihood estimation (not passed to `lavaan`).
`m`	Number of imputations to perform. More imputations will yield better estimates but can also be (a lot) slower.
`verbose`	Should progress be printed to the console?
`se`	How should corrected standard errors be computed? Alternatives are: `"simple"` Uncorrected standard errors are only calculated once, in the first imputation. The standard errors are thereafter corrected using the distribution of the estimated coefficients from the different imputations. `"full"` Uncorrected standard errors are calculated and aggregated for each imputation. This can give more accurate results, but can be (a lot) slower. The standard errors are thereafter corrected using the distribution of the estimated coefficients from the different imputations.
`...`	Arguments passed to `modsem`.

Details

`modsem_impute` is currently only available for the DA approaches (LMS and QML). It performs multiple imputation using `Amelia::amelia` and returns aggregated coefficients from the multiple imputations, along with corrected standard errors.

Examples

```
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'

oneInt2 <- oneInt

set.seed(123)
k <- 200
I <- sample(nrow(oneInt2), k, replace = TRUE)
J <- sample(ncol(oneInt2), k, replace = TRUE)
for (k_i in seq_along(I)) oneInt2[I[k_i], J[k_i]] <- NA

## Not run:
est <- modsem_mimpute(m1, oneInt2, m = 25)
summary(est)

## End(Not run)
```

`modsem_mplus` *Estimation latent interactions through* `Mplus`

Description

Estimation latent interactions through `Mplus`

Usage

```
modsem_mplus(
  model.syntax,
  data,
  estimator = "ml",
  cluster = NULL,
  type = ifelse(is.null(cluster), yes = "random", no = "complex"),
  algorithm = "integration",
  processors = 2,
  integration = 15,
  rcs = FALSE,
  rcs.choose = NULL,
  rcs.scale.corrected = TRUE,
  output.std = TRUE,
  ...
)
```

Arguments

model.syntax	lavaan/modsem syntax
data	dataset
estimator	estimator argument passed to Mplus.
cluster	cluster argument passed to Mplus.
type	type argument passed to Mplus.
algorithm	algorithm argument passed to Mplus.
processors	processors argument passed to Mplus.
integration	integration argument passed to Mplus.
rcs	Should latent variable indicators be replaced with reliability-corrected single item indicators instead? See relcorr_single_item.
rcs.choose	Which latent variables should get their indicators replaced with reliability-corrected single items? It is passed to relcorr_single_item as the choose argument.
rcs.scale.corrected	Should reliability-corrected items be scale-corrected? If TRUE reliability-corrected single items are corrected for differences in factor loadings between the items. Default is TRUE.
output.std	Should STANDARDIZED be added to OUTPUT?
...	arguments passed to other functions

Value

modsem_mplus object

Examples

```
# Theory Of Planned Behavior
m1 <- '
# Outer Model
  X =~ x1 + x2
  Z =~ z1 + z2
  Y =~ y1 + y2

# Inner model
  Y ~ X + Z + X:Z
'

## Not run:
# Check if Mplus is installed
run <- tryCatch({MplusAutomation::detectMplus(); TRUE},
                error = \(e) FALSE)

if (run) {
  est_mplus <- modsem_mplus(m1, data = oneInt)
  summary(est_mplus)
}
```

```
## End(Not run)
```

`modsem_nobs` *Wrapper for nobs*

Description

wrapper for `nobs`, to be used with `modsem::modsem_nobs`, since `nobs` is not in the namespace of `modsem`, but `stats`.

Usage

```
modsem_nobs(object, ...)
```

Arguments

`object`	fitted model to inspect
`...`	additional arguments

`modsem_pi` *Interaction between latent variables using product indicators*

Description

`modsem_pi()` is a function for estimating interaction effects between latent variables, in structural equation models (SEMs), using product indicators. Methods for estimating interaction effects in SEMs can basically be split into two frameworks: 1. Product Indicator based approaches (`"dblcent"`, `"rca"`, `"uca"`, `"ca"`, `"pind"`), and 2. Distributionally based approaches (`"lms"`, `"qml"`). `modsem_pi()` is essentially a fancy wrapper for `lavaan::sem()` which generates the necessary syntax and variables for the estimation of models with latent product indicators. Use `default_settings_pi()` to get the default settings for the different methods.

Usage

```
modsem_pi(
  model.syntax = NULL,
  data = NULL,
  method = "dblcent",
  match = NULL,
  match.recycle = NULL,
  standardize.data = FALSE,
  center.data = FALSE,
  first.loading.fixed = FALSE,
```

```
  center.before = NULL,
  center.after = NULL,
  residuals.prods = NULL,
  residual.cov.syntax = NULL,
  constrained.prod.mean = NULL,
  constrained.loadings = NULL,
  constrained.var = NULL,
  res.cov.method = NULL,
  res.cov.across = NULL,
  auto.scale = "none",
  auto.center = "none",
  estimator = "ML",
  group = NULL,
  cluster = NULL,
  run = TRUE,
  na.rm = FALSE,
  suppress.warnings.lavaan = FALSE,
  suppress.warnings.match = FALSE,
  rcs = FALSE,
  rcs.choose = NULL,
  rcs.res.cov.xz = rcs,
  rcs.mc.reps = 1e+05,
  rcs.scale.corrected = TRUE,
  LAVFUN = lavaan::sem,
  ...
)
```

Arguments

model.syntax	lavaan syntax
data	dataframe
method	method to use: "dblcent" double centering approach (passed to lavaan). "ca" constrained approach (passed to lavaan). "rca" residual centering approach (passed to lavaan). "uca" unconstrained approach (passed to lavaan). "pind" prod ind approach, with no constraints or centering (passed to lavaan).
match	should the product indicators be created by using the match-strategy
match.recycle	should the indicators be recycled when using the match-strategy? I.e., if one of the latent variables have fewer indicators than the other, some indicators are recycled to match the latent variable with the most indicators.
standardize.data	should data be scaled before fitting model
center.data	should data be centered before fitting model
first.loading.fixed	Should the first factor loading in the latent product be fixed to one? Defaults to FALSE, as this already happens in lavaan by default. If TRUE, the first factor

loading in the latent product is fixed to one. Manually in the generated syntax (e.g., XZ =~ 1*x1z1).'

`center.before` should indicators in products be centered before computing products.

`center.after` should indicator products be centered after they have been computed?

`residuals.prods`
should indicator products be centered using residuals.

`residual.cov.syntax`
should syntax for residual covariances be produced.

`constrained.prod.mean`
should syntax for product mean be produced.

`constrained.loadings`
should syntax for constrained loadings be produced.

`constrained.var`
should syntax for constrained variances be produced.

`res.cov.method` method for constraining residual covariances. Options are

"simple" Residuals of product indicators with variables in common are allowed to covary freely. Defualt for most approches.

"ca" Residual covariances of product indicators are constrained according to the constrained approach.

"equality" Residuals of product indicators with variables in common are constrained to have equal covariances". Can be useful for models where the model is unidentifiable using `res.cov.method == "simple"`, (e.g., when there is an interaction between an observed and a latent variable).

"none" Residual covariances between product indicators are not specificed (i.e., constrained to zero). Produces the same results as `constrained.cov.syntax = FALSE`. Can be useful for models where the model is unidentifiable using `res.cov.method == "simple"`, (e.g., when there is an interaction between an observed and a latent variable).

`res.cov.across` Should residual covariances be specified/freed across different interaction terms. For example if you have two interaction terms `X:Z` and `X:W` the residuals of the generated product indicators `x1:z1` and `x1:w1` may be correlated. If `TRUE` residual covariances are allowed across different latent interaction terms. If `FALSE` residual covariances are only allowed between product indicators which belong to the same latent interaction term.

`auto.scale` methods which should be scaled automatically (usually not useful)

`auto.center` methods which should be centered automatically (usually not useful)

`estimator` estimator to use in `lavaan`

`group` group variable for multigroup analysis

`cluster` cluster variable for multilevel models

`run` should the model be run via `lavaan`, if `FALSE` only modified syntax and data is returned

`na.rm` should missing values be removed (case-wise)? Defaults to FALSE. If `TRUE`, missing values are removed case-wise. If `FALSE` they are not removed.

suppress.warnings.lavaan
: should warnings from lavaan be suppressed?

suppress.warnings.match
: should warnings from match be suppressed?

rcs
: Should latent variable indicators be replaced with reliability-corrected single item indicators instead? See relcorr_single_item.

rcs.choose
: Which latent variables should get their indicators replaced with reliability-corrected single items? It is passed to relcorr_single_item as the choose argument.

rcs.res.cov.xz
: Should the residual (co-)variances of the product indicators created from the reliability-corrected single items (created if rcs = TRUE) be specified and constrained before estimating the model? If TRUE the estimates for the constraints are approximated using a monte carlo simulation (see the rcs.mc.reps argument). If FALSE the residual variances are not specified, which usually mean that all are constrained to zero.

rcs.mc.reps
: Sample size used in monte-carlo simulation, when approximating the the estimates of the residual (co-)variances between the product indicators formed by reliabiliyt-corrected single items (see the rcs.res.cov.xz argument).

rcs.scale.corrected
: Should reliability corrected items be scale-corrected? If TRUE reliability-corrected single items are corrected for differences in factor loadings between the items. Default is TRUE.

LAVFUN
: Function used to estimate the model. Defaults to lavaan::sem.

...
: arguments passed to LAVFUN

Value

modsem object

Examples

```
library(modsem)
# For more examples, check README and/or GitHub.
# One interaction
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'

# Double centering approach
est <- modsem_pi(m1, oneInt)
summary(est)

## Not run:
```

```
# The Constrained Approach
est_ca <- modsem_pi(m1, oneInt, method = "ca")
summary(est_ca)

## End(Not run)

# Theory Of Planned Behavior
tpb <- '
# Outer Model (Based on Hagger et al., 2007)
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model (Based on Steinmetz et al., 2011)
  # Covariances
  ATT ~~ SN + PBC
  PBC ~~ SN
  # Causal Relationships
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  BEH ~ INT:PBC
'

# Double centering approach
est_tpb <- modsem_pi(tpb, data = TPB)
summary(est_tpb)

## Not run:
# The Constrained Approach
est_tpb_ca <- modsem_pi(tpb, data = TPB, method = "ca")
summary(est_tpb_ca)

## End(Not run)
```

`modsem_predict` *Predict From* `modsem` *Models*

Description

A generic function (and corresponding methods) that produces predicted values or factor scores from `modsem` models.

Usage

```
modsem_predict(object, ...)

## S3 method for class 'modsem_da'
modsem_predict(
```

```
  object,
  standardized = FALSE,
  H0 = TRUE,
  newdata = NULL,
  center.data = TRUE,
  ...
)

## S3 method for class 'modsem_pi'
modsem_predict(object, ...)
```

Arguments

object	modsem_da object
...	Further arguments passed to lavaan::predict; currently ignored by the modsem_da method.
standardized	Logical. If TRUE, return standardized factor scores.
H0	Logical. If TRUE (default), use the baseline model to compute factor scores. If FALSE, use the model specified in object. Using H0 = FALSE is not recommended!
newdata	Compute factor scores based on a different dataset, than the one used in the model estimation.
center.data	Should data be centered before computing factor scores? Default is TRUE.

Value

* For modsem_pi: whatever lavaan::predict(), which usually returns a matrix of factor scores.
* For modsem_da: a numeric matrix $n \times p$, where n is the number of (complete) observations in the dataset, and p the number of latent variables. Each column contains either raw or standardised factor scores, depending on the standardized argument.

Methods (by class)

- modsem_predict(modsem_da): Computes (optionally standardised) factor scores via the regression method using the baseline model unless H0 = FALSE.
- modsem_predict(modsem_pi): Wrapper for lavaan::predict

Examples

```
m1 <- '
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

# Inner Model
  Y ~ X + Z + X:Z
'
```

```
est_dca <- modsem(m1, oneInt, method = "dblcent")
head(modsem_predict(est_dca))

## Not run:
est_lms <- modsem(m1, oneInt, method = "lms")
head(modsem_predict(est_lms))

## End(Not run)
```

`modsem_vcov` *Wrapper for vcov*

Description

wrapper for `vcov`, to be used with `modsem::modsem_vcov`, since `vcov` is not in the namespace of `modsem`, but `stats`.

Usage

```
modsem_vcov(object, ...)
```

Arguments

`object`	fitted model to inspect
`...`	additional arguments

`oneInt` *oneInt*

Description

A simulated dataset based on the elementary interaction model.

Examples

```
m1 <- "
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3
# Inner Model
  Y ~ X + Z + X:Z
"

est <- modsem(m1, data = oneInt)
summary(est)
```

parameter_estimates *Extract parameterEstimates from an estimated model*

Description

Extract parameterEstimates from an estimated model

Usage

```
parameter_estimates(object, ...)

## S3 method for class 'lavaan'
parameter_estimates(
  object,
  colon.pi = NULL,
  high.order.as.measr = NULL,
  rm.tmp.ov = NULL,
  label.renamed.prod = NULL,
  is.public = NULL,
  ...
)

## S3 method for class 'modsem_da'
parameter_estimates(
  object,
  high.order.as.measr = TRUE,
  is.public = TRUE,
  rm.tmp.ov = is.public,
  label.renamed.prod = NULL,
  ...
)

## S3 method for class 'modsem_mplus'
parameter_estimates(
  object,
  colon.pi = NULL,
  high.order.as.measr = NULL,
  rm.tmp.ov = NULL,
  label.renamed.prod = NULL,
  is.public = NULL,
  ...
)

## S3 method for class 'modsem_pi'
parameter_estimates(
  object,
  colon.pi = FALSE,
```

```
  label.renamed.prod = FALSE,
  high.order.as.measr = NULL,
  rm.tmp.ov = NULL,
  is.public = NULL,
  ...
)
```

Arguments

object	An object of class modsem_pi, modsem_da, or modsem_mplus
...	Additional arguments passed to other functions
colon.pi	Should colons (:) be added to the interaction terms (E.g., 'XZ' -> 'X:Z')?
high.order.as.measr	Should higher order measurement model be denoted with the =~ operator? If FALSE the ~ operator is used.
rm.tmp.ov	Should temporary (hidden) variables be removed?
label.renamed.prod	Should renamed product terms keep their old (implicit) labels?
is.public	Should public version of parameter table be returned? If FALSE, the internal version of the parameter table is returned.

Methods (by class)

- parameter_estimates(lavaan): Get parameter estimates of a lavaan object
- parameter_estimates(modsem_da): Get parameter estimates of a modsem_da object
- parameter_estimates(modsem_mplus): Get parameter estimates of a modsem_mplus object
- parameter_estimates(modsem_pi): Get parameter estimates of a modsem_pi object

Examples

```
m1 <- '
  # Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

  # Inner Model
  Y ~ X + Z + X:Z
'
# Double centering approach
est_dca <- modsem(m1, oneInt)

pars <- parameter_estimates(est_dca) # no correction

# Pretty summary
summarize_partable(pars)

# Only print the data.frame
pars
```

plot_interaction *Plot Interaction Effects in a SEM Model*

Description

This function creates an interaction plot of the outcome variable (y) as a function of a focal predictor (x) at multiple values of a moderator (z). It is designed for use with structural equation modeling (SEM) objects (e.g., those from modsem). Predicted means (or predicted individual values) are calculated via simple_slopes, and then plotted with ggplot2 to display multiple regression lines and confidence/prediction bands.

Usage

```
plot_interaction(
  x,
  z,
  y,
  model,
  vals_x = seq(-3, 3, 0.001),
  vals_z,
  alpha_se = 0.15,
  digits = 2,
  ci_width = 0.95,
  ci_type = "confidence",
  rescale = TRUE,
  standardized = FALSE,
  xz = NULL,
  greyscale = FALSE,
  ...
)
```

Arguments

x	A character string specifying the focal predictor (x-axis variable).
z	A character string specifying the moderator variable.
y	A character string specifying the outcome (dependent) variable.
model	An object of class modsem_pi, modsem_da, modsem_mplus, or possibly a lavaan object. Must be a fitted SEM model containing paths for y ~ x + z + x:z.
vals_x	A numeric vector of values at which to compute and plot the focal predictor x. The default is seq(-3, 3, .001), which provides a relatively fine grid for smooth lines. If rescale=TRUE, these values are in standardized (mean-centered and scaled) units, and will be converted back to the original metric in the internal computation of predicted means.
vals_z	A numeric vector of values of the moderator z at which to draw separate regression lines. Each distinct value in vals_z defines a separate group (plotted

with a different color). If `rescale=TRUE`, these values are also assumed to be in standardized units.

`alpha_se` A numeric value in $[0, 1]$ specifying the transparency of the confidence/prediction interval ribbon. Default is `0.15`.

`digits` An integer specifying the number of decimal places to which the moderator values (`z`) are rounded for labeling/grouping in the plot.

`ci_width` A numeric value in $(0, 1)$ indicating the coverage of the confidence (or prediction) interval. The default is `0.95` for a 95% interval.

`ci_type` A character string specifying whether to compute `"confidence"` intervals (for the mean of the predicted values, default) or `"prediction"` intervals (which include residual variance).

`rescale` Logical. If `TRUE` (default), `vals_x` and `vals_z` are interpreted as standardized units, which are mapped back to the raw scale before computing predictions. If `FALSE`, `vals_x` and `vals_z` are taken as raw-scale values directly.

`standardized` Should coefficients be standardized beforehand?

`xz` A character string specifying the interaction term (`x:z`). If `NULL`, the term is created automatically as `paste(x, z, sep = ":")`. Some SEM backends may handle the interaction term differently (for instance, by removing or modifying the colon), and this function attempts to reconcile that internally.

`greyscale` Logical. If `TRUE` the plot is plotted in greyscale.

`...` Additional arguments passed on to `simple_slopes`.

Details

Computation Steps:

1. Calls `simple_slopes` to compute the predicted values of `y` at the specified grid of `x` and `z` values.
2. Groups the resulting predictions by unique `z`-values (rounded to `digits`) to create colored lines.
3. Plots these lines using `ggplot2`, adding ribbons for confidence (or prediction) intervals, with transparency controlled by `alpha_se`.

Interpretation: Each line in the plot corresponds to the regression of `y` on `x` at a given level of `z`. The shaded region around each line (ribbon) shows either the confidence interval for the predicted mean (if `ci_type = "confidence"`) or the prediction interval for individual observations (if `ci_type = "prediction"`). Where the ribbons do not overlap, there is evidence that the lines differ significantly over that range of `x`.

Value

A `ggplot` object that can be further customized (e.g., with additional `+ theme(...)` layers). By default, it shows lines for each moderator value and a shaded region corresponding to the interval type (confidence or prediction).

Examples

```
## Not run:
library(modsem)

# Example 1: Interaction of X and Z in a simple SEM
m1 <- "
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

# Inner Model
  Y ~ X + Z + X:Z
"
est1 <- modsem(m1, data = oneInt)

# Plot interaction using a moderate range of X and two values of Z
plot_interaction(x = "X", z = "Z", y = "Y", xz = "X:Z",
                 vals_x = -3:3, vals_z = c(-0.2, 0), model = est1)

# Example 2: Interaction in a theory-of-planned-behavior-style model
tpb <- "
# Outer Model
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN  =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  BEH ~ PBC:INT
"
est2 <- modsem(tpb, data = TPB, method = "lms", nodes = 32)

# Plot with custom Z values and a denser X grid
plot_interaction(x = "INT", z = "PBC", y = "BEH",
                 xz = "PBC:INT",
                 vals_x = seq(-3, 3, 0.01),
                 vals_z = c(-0.5, 0.5),
                 model = est2)

## End(Not run)
```

---

plot_jn *Plot Interaction Effect Using the Johnson-Neyman Technique*

Description

This function plots the simple slopes of an interaction effect across different values of a moderator variable using the Johnson-Neyman technique. It identifies regions where the effect of the predictor on the outcome is statistically significant.

Usage

```
plot_jn(
  x,
  z,
  y,
  model,
  min_z = -3,
  max_z = 3,
  sig.level = 0.05,
  alpha = 0.2,
  detail = 1000,
  sd.line = 2,
  standardized = FALSE,
  xz = NULL,
  greyscale = FALSE,
  plot.jn.points = TRUE,
  ...
)
```

Arguments

x	The name of the predictor variable (as a character string).
z	The name of the moderator variable (as a character string).
y	The name of the outcome variable (as a character string).
model	A fitted model object of class modsem_da, modsem_mplus, modsem_pi, or lavaan.
min_z	The minimum value of the moderator variable z to be used in the plot (default is -3). It is relative to the mean of z.
max_z	The maximum value of the moderator variable z to be used in the plot (default is 3). It is relative to the mean of z.
sig.level	The alpha-criterion for the confidence intervals (default is 0.05).
alpha	alpha setting used in ggplot (i.e., the opposite of opacity)
detail	The number of generated data points to use for the plot (default is 1000). You can increase this value for smoother plots.
sd.line	A thick black line showing +/- sd.line * sd(z). NOTE: This line will be truncated by min_z and max_z if the sd.line falls outside of [min_z, max_z].
standardized	Should coefficients be standardized beforehand?
xz	The name of the interaction term. If not specified, it will be created using x and z.
greyscale	Logical. If TRUE the plot is plotted in greyscale.

`plot.jn.points` Logical. If `TRUE`, omit the numeric annotations for the JN-points from the plot.

`...` Additional arguments (currently not used).

Details

The function calculates the simple slopes of the predictor variable `x` on the outcome variable `y` at different levels of the moderator variable `z`. It uses the Johnson-Neyman technique to identify the regions of `z` where the effect of `x` on `y` is statistically significant.

It extracts the necessary coefficients and variance-covariance information from the fitted model object. The function then computes the critical t-value and solves the quadratic equation derived from the t-statistic equation to find the Johnson-Neyman points.

The plot displays:

- The estimated simple slopes across the range of `z`.
- Confidence intervals around the slopes.
- Regions where the effect is significant (shaded areas).
- Vertical dashed lines indicating the Johnson-Neyman points.
- Text annotations providing the exact values of the Johnson-Neyman points.

Value

A `ggplot` object showing the interaction plot with regions of significance.

Examples

```
## Not run:
library(modsem)

m1 <-  '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9

  visual ~ speed + textual + speed:textual
'

est <- modsem(m1, data = lavaan::HolzingerSwineford1939, method = "ca")
plot_jn(x = "speed", z = "textual", y = "visual", model = est, max_z = 6)

## End(Not run)
```

plot_surface *Plot Surface for Interaction Effects*

Description

Generates a 3D surface plot to visualize the interaction effect of two variables (x and z) on an outcome (y) using parameter estimates from a supported model object (e.g., lavaan or modsem). The function allows specifying ranges for x and z in standardized z-scores, which are then transformed back to the original scale based on their means and standard deviations.

Usage

```
plot_surface(
  x,
  z,
  y,
  model,
  min_x = -3,
  max_x = 3,
  min_z = -3,
  max_z = 3,
  standardized = FALSE,
  detail = 0.01,
  xz = NULL,
  colorscale = "Viridis",
  reversescale = FALSE,
  showscale = TRUE,
  cmin = NULL,
  cmax = NULL,
  surface_opacity = 1,
  grid = FALSE,
  grid_nx = 12,
  grid_ny = 12,
  grid_color = "rgba(0,0,0,0.45)",
  group = NULL,
  ...
)
```

Arguments

x	A character string specifying the name of the first predictor variable.
z	A character string specifying the name of the second predictor variable.
y	A character string specifying the name of the outcome variable.
model	A model object of class modsem_pi, modsem_da, modsem_mplus, or lavaan. The model should include paths for the predictors (x, z, and xz) to the outcome (y).

min_x	Numeric. Minimum value of x in z-scores. Default is -3.
max_x	Numeric. Maximum value of x in z-scores. Default is 3.
min_z	Numeric. Minimum value of z in z-scores. Default is -3.
max_z	Numeric. Maximum value of z in z-scores. Default is 3.
standardized	Should coefficients be standardized beforehand?
detail	Numeric. Step size for the grid of x and z values, determining the resolution of the surface. Smaller values increase plot resolution. Default is 1e-2.
xz	Optional. A character string or vector specifying the interaction term between x and z. If NULL, the interaction term is constructed as paste(x, z, sep = ":") and adjusted for specific model classes.
colorscale	Character or list. Colorscale used to color the surface. - Default is "Viridis", which matches the classic Plotly default. - Can be a built-in palette name (e.g., "Greys", "Plasma", "Turbo"), or a custom two-column list with numeric stops (0–1) and color codes. Example custom scale: list(c(0, "white"), c(1, "black")) for a black-and-white gradient.
reversescale	Logical. If TRUE, reverses the color mapping so that low values become high colors and vice versa. Default is FALSE.
showscale	Logical. If TRUE, displays the colorbar legend alongside the plot. Default is TRUE.
cmin	Numeric or NULL. The minimum value for the colorscale mapping. If NULL, the minimum of the data (proj_y) is used automatically. Use this to standardize the color range across multiple plots.
cmax	Numeric or NULL. The maximum value for the colorscale mapping. If NULL, the maximum of the data (proj_y) is used automatically. Use this to standardize the color range across multiple plots.
surface_opacity	Numeric (0–1). Controls the opacity of the surface. - 1 = fully opaque (default) - 0 = fully transparent Useful when overlaying multiple surfaces or highlighting gridlines.
grid	Logical. If TRUE, draws gridlines (wireframe) directly on the surface using Plotly's contour features. Default is FALSE.
grid_nx	Integer. Approximate number of gridlines to draw along the **x-axis** direction when grid = TRUE. Higher values create a denser grid. Default is 12.
grid_ny	Integer. Approximate number of gridlines to draw along the **y-axis** direction when grid = TRUE. Higher values create a denser grid. Default is 12.
grid_color	Character. Color of the gridlines drawn on the surface. Must be a valid CSS color string, including rgba() for transparency. - Default is "rgba(0,0,0,0.45)" (semi-transparent black). Example: "rgba(255,255,255,0.8)" for semi-transparent white lines.
group	Which group to create surface plot for. Only relevant for multigroup models. Must be an integer index, representing the nth group.
...	Additional arguments passed to plotly::plot_ly.

Details

The input min_x, max_x, min_z, and max_z define the range of x and z values in z-scores. These are scaled by the standard deviations and shifted by the means of the respective variables, obtained from the model parameter table. The resulting surface shows the predicted values of y over the grid of x and z.

The function supports models of class modsem (with subclasses modsem_pi, modsem_da, modsem_mplus) and lavaan. For lavaan models, it is not designed for multigroup models, and a warning will be issued if multiple groups are detected.

Value

A plotly surface plot object displaying the predicted values of y across the grid of x and z values. The color bar shows the values of y.

Note

The interaction term (xz) may need to be manually specified for some models. For non-lavaan models, interaction terms may have their separator (:) removed based on circumstances.

Examples

```
m1 <- "
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

# Inner model
  Y ~ X + Z + X:Z
"
est1 <- modsem(m1, data = oneInt)
plot_surface("X", "Z", "Y", model = est1)

## Not run:
tpb <- "
# Outer Model (Based on Hagger et al., 2007)
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model (Based on Steinmetz et al., 2011)
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  BEH ~ PBC:INT
"

est2 <- modsem(tpb, TPB, method = "lms", nodes = 32)
plot_surface(x = "INT", z = "PBC", y = "BEH", model = est2)
```

```
## End(Not run)
```

`relcorr_single_item` *Reliability-Corrected Single-Item SEM*

Description

Replace (some of) the first-order latent variables in a lavaan measurement model by single composite indicators whose error variances are fixed from Cronbach's α. The function returns a modified lavaan model syntax together with an augmented data set that contains the newly created composite variables, so that you can fit the full SEM in a single step.

Usage

```
relcorr_single_item(
  syntax,
  data,
  choose = NULL,
  scale.corrected = TRUE,
  warn.lav = TRUE,
  group = NULL
)
```

Arguments

`syntax`	A character string containing lavaan model syntax. Must at least include the measurement relations (`=~`).
`data`	A `data.frame`, `tibble` or object coercible to a data frame that holds the raw observed indicators.
`choose`	*Optional.* Character vector with the names of the latent variables you wish to replace by single indicators. Defaults to **all** first-order latent variables in `syntax`.
`scale.corrected`	Should reliability-corrected items be scale-corrected? If `TRUE` reliability-corrected single items are corrected for differences in factor loadings between the items. Default is `TRUE`.
`warn.lav`	Should warnings from `lavaan::cfa` be displayed? If `FALSE`, they are suppressed.
`group`	Character. A variable name in the data frame defining the groups in a multiple group analysis

Details

The resulting object can be fed directly into `modsem` or `lavaan::sem` by supplying `syntax = $syntax` and `data = $data`.

Value

An object of S3 class `ModsemRelcorr` (a named list) with elements:

`syntax` Modified lavaan syntax string.

`data` Data frame with additional composite indicator columns.

`parTable` Parameter table corresponding to 'syntax'.

`reliability` Named numeric vector of reliabilities (one per latent variable).

`AVE` Named numeric vector with Average Variance Extracted values.

`lVs` Character vector of latent variables that were corrected.

`single.items` Character vector with names for the generated reliability corrected items

Examples

```
## Not run:
tpb_uk <- "
# Outer Model (Based on Hagger et al., 2007)
 ATT =~ att3 + att2 + att1 + att4
 SN =~ sn4 + sn2 + sn3 + sn1
 PBC =~ pbc2 + pbc1 + pbc3 + pbc4
 INT =~ int2 + int1 + int3 + int4
 BEH =~ beh3 + beh2 + beh1 + beh4

# Inner Model (Based on Steinmetz et al., 2011)
 INT ~ ATT + SN + PBC
 BEH ~ INT + PBC
 BEH ~ INT:PBC
"

corrected <- relcorr_single_item(syntax = tpb_uk, data = TPB_UK)
print(corrected)

syntax <- corrected$syntax
data   <- corrected$data

est_dca <- modsem(syntax, data = data, method = "dblcent")
est_lms <- modsem(syntax, data = data, method="lms", nodes=32)
summary(est_lms)

## End(Not run)
```

`set_modsem_colors` *Define or disable the color theme used by* `modsem`

Description

All arguments are optional; omitted ones fall back to the defaults below. Pass `active = FALSE` to turn highlighting off (and reset the palette).

Usage

```
set_modsem_colors(
  positive = "green3",
  negative = positive,
  true = "green3",
  false = "red",
  nan = "purple",
  na = "purple",
  inf = "purple",
  string = "cyan",
  active = TRUE
)
```

Arguments

positive	color of positive numbers.
negative	color of negative numbers.
true	color of TRUE.
false	color of FALSE.
nan	color of NaN.
na	color of NA.
inf	color of -Inf and Inf.
string	color of quoted strings.
active	Should color-theme be activated/deactived?

Value

TRUE if colors are active afterwards, otherwise FALSE.

Examples

```
set_modsem_colors(positive = "red3",
                  negative = "red3",
                  true = "darkgreen",
                  false = "red3",
                  na = "purple",
                  string = "darkgreen")

m1 <- "
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3
# Inner Model
  Y ~ X + Z + X:Z
"

est <- modsem(m1, data = oneInt)
```

```
colorize_output(summary(est))
colorize_output(est) # same as colorize_output(print(est))
colorize_output(modsem_inspect(est, what = "coef"))

## Not run:
colorize_output(split = TRUE, {
  # Get live (uncolored) output
  # And print colored output at the end of execution

  est_lms <- modsem(m1, data = oneInt, method = "lms")
  summary(est_lms)
})

colorize_output(modsem_inspect(est_lms))

## End(Not run)
```

`simple_slopes` *Get the simple slopes of a SEM model*

Description

This function calculates simple slopes (predicted values of the outcome variable) at user-specified values of the focal predictor (`x`) and moderator (`z`) in a structural equation modeling (SEM) framework. It supports interaction terms (`xz`), computes standard errors (SE), and optionally returns confidence or prediction intervals for these predicted values. It also provides p-values for hypothesis testing. This is useful for visualizing and interpreting moderation effects or to see how the slope changes at different values of the moderator.

Usage

```
simple_slopes(
  x,
  z,
  y,
  model,
  vals_x = -3:3,
  vals_z = -1:1,
  rescale = TRUE,
  ci_width = 0.95,
  ci_type = "confidence",
  relative_h0 = TRUE,
  standardized = FALSE,
  xz = NULL,
  ...
)
```

Arguments

x	The name of the variable on the x-axis (focal predictor).
z	The name of the moderator variable.
y	The name of the outcome variable.
model	An object of class modsem_pi, modsem_da, modsem_mplus, or a lavaan object. This should be a fitted SEM model that includes paths for y ~ x + z + x:z.
vals_x	Numeric vector of values of x at which to compute predicted slopes. Defaults to -3:3. If rescale = TRUE, these values are taken relative to the mean and standard deviation of x. A higher density of points (e.g., seq(-3, 3, 0.1)) will produce smoother curves and confidence bands.
vals_z	Numeric vector of values of the moderator z at which to compute predicted slopes. Defaults to -1:1. If rescale = TRUE, these values are taken relative to the mean and standard deviation of z. Each unique value of z generates a separate regression line y ~ x | z.
rescale	Logical. If TRUE (default), x and z are standardized according to their means and standard deviations in the model. The values in vals_x and vals_z are interpreted in those standardized units. The raw (unscaled) values corresponding to these standardized points will be displayed in the returned table.
ci_width	A numeric value between 0 and 1 indicating the confidence (or prediction) interval width. The default is 0.95 (i.e., 95% interval).
ci_type	A string indicating whether to compute a "confidence" interval for the predicted mean (*i.e.*, uncertainty in the regression line) or a "prediction" interval for individual outcomes. The default is "confidence". If "prediction", the residual variance is added to the variance of the fitted mean, resulting in a wider interval.
relative_h0	Logical. If TRUE (default), hypothesis tests for the predicted values (predicted - h0) assume h0 is the model-estimated mean of y. If FALSE, the null hypothesis is h0 = 0.
standardized	Should coefficients be standardized beforehand?
xz	The name of the interaction term (x:z). If NULL, it will be created by combining x and z with a colon (e.g., "x:z"). Some backends may remove or alter the colon symbol, so the function tries to account for that internally.
...	Additional arguments passed to lower-level functions or other internal helpers.

Details

Computation Steps

1. The function extracts parameter estimates (and, if necessary, their covariance matrix) from the fitted SEM model (model).
2. It identifies the coefficients for x, z, and x:z in the model's parameter table, as well as the variance of x, z and the residual variance of y.
3. If xz is not provided, it will be constructed by combining x and z with a colon (":"). Depending on the approach used to estimate the model, the colon may be removed or replaced internally; the function attempts to reconcile that.

4. A grid of x and z values is created from vals_x and vals_z. If rescale = TRUE, these values are transformed back into raw metric units for display in the output.
5. For each point in the grid, a predicted value of y is computed via (beta0 + beta_x * x + beta_z * z + beta_xz * x * z) and, if included, a mean offset.
6. The standard error (std.error) is derived from the covariance matrix of the relevant parameters, and if ci_type = "prediction", adds the residual variance.
7. Confidence (or prediction) intervals are formed using ci_width (defaulting to 95%). The result is a table-like data frame with predicted values, CIs, standard errors, z-values, and p-values.

Value

A data.frame (invisibly inheriting class "simple_slopes") with columns:

- vals_x, vals_z: The requested grid values of x and z.
- predicted: The predicted value of y at that combination of x and z.
- std.error: The standard error of the predicted value.
- z.value, p.value: The z-statistic and corresponding p-value for testing the null hypothesis that predicted == h0.
- ci.lower, ci.upper: Lower and upper bounds of the confidence (or prediction) interval.

An attribute "variable_names" (list of x, z, y) is attached for convenience.

Examples

```
## Not run:
library(modsem)

m1 <- "
# Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

# Inner model
  Y ~ X + Z + X:Z
"
est1 <- modsem(m1, data = oneInt)

# Simple slopes at X in [-3, 3] and Z in [-1, 1], rescaled to the raw metric.
simple_slopes(x = "X", z = "Z", y = "Y", model = est1)

# If the data or user wants unscaled values, set rescale = FALSE, etc.
simple_slopes(x = "X", z = "Z", y = "Y", model = est1, rescale = FALSE)

## End(Not run)
```

standardized_estimates *Get Standardized Estimates*

Description

Computes standardized estimates of model parameters for various types of modsem objects.

Usage

```
standardized_estimates(object, ...)

## S3 method for class 'lavaan'
standardized_estimates(
  object,
  monte.carlo = FALSE,
  mc.reps = 10000,
  tolerance.zero = 1e-10,
  ...
)

## S3 method for class 'modsem_da'
standardized_estimates(
  object,
  monte.carlo = FALSE,
  mc.reps = 10000,
  tolerance.zero = 1e-10,
  rm.tmp.ov = TRUE,
  ...
)

## S3 method for class 'modsem_mplus'
standardized_estimates(object, type = "stdyx", mc.reps = 1e+06, ...)

## S3 method for class 'modsem_pi'
standardized_estimates(
  object,
  correction = FALSE,
  std.errors = c("rescale", "delta", "monte.carlo"),
  mc.reps = 10000,
  colon.pi = FALSE,
  ...
)
```

Arguments

object	An object of class modsem_da, modsem_mplus, modsem_pi, or a parameter table (parTable) of class data.frame.

`...`	Additional arguments passed on to `lavaan::standardizedSolution()`.
`monte.carlo`	Logical. If `TRUE`, use Monte Carlo simulation to estimate standard errors; if `FALSE`, use the delta method (default).
`mc.reps`	Integer. Number of Monte Carlo replications to use if `std.errors = "monte.carlo"`.
`tolerance.zero`	Threshold below which standard errors are set to `NA`.
`rm.tmp.ov`	Should temporary (hidden) variables be removed?
`type`	Type of standardized estimates to retrieve. Can be one of: `"stdyx"`, `"stdy"`, `"std"`, `"un"`, `"modsem"`.
`correction`	Logical. Whether to apply a correction to the standardized estimates of the interaction term. By default, `FALSE`, which standardizes the interaction term such that $\sigma^2(XZ) = 1$, consistent with `lavaan`'s treatment of latent interactions. This is usually wrong, as it does not account for the fact that the interaction term is a product of two variables, which means that the variance of the interaction term of standardized variables (usually) is not equal to 1. Hence the scale of the interaction effect changes, such that the standardized interaction term does not correspond to one (standardized) unit change in the moderating variables. If `TRUE`, a correction is applied by computing the interaction term $b_3 = \frac{B_3\sigma(X)\sigma(Z)}{\sigma(Y)}$ (where B_3 is the unstandardized coefficient for the interaction term), and solving for $\sigma(XZ)$, which is used to correct the variance of the interaction term, and its covariances.
`std.errors`	Character string indicating the method used to compute standard errors when `correction = TRUE`. Options include: **"rescale"** Simply rescales the standard errors. Fastest option. **"delta"** Uses the delta method to approximate standard errors. **"monte.carlo"** Uses Monte Carlo simulation to estimate standard errors. Ignored if `correction = FALSE`.
`colon.pi`	Logical. If `TRUE`, the interaction terms in the output will be will be formatted using `:` to seperate the elements in the interaction term. Default is `FALSE`, using the default formatting from `lavaan`. Only relevant if `std.errors != "rescale"` and `correction = TRUE`.

Details

Standard errors can either be calculated using the delta method, or a monte.carlo simulation (`monte.carlo` is not available for `modsem_pi` objects if `correction == FALSE`.). **NOTE** that the standard errors of the standardized paramters change the assumptions of the model, and will in most cases yield different z and p-values, compared to the unstandardized solution. In almost all cases, significance testing should be based on the unstandardized solution. Since, the standardization process changes the model assumptions, it also changes what the p-statistics measure. I.e., the test statistics for the standardized and unstandardized solutions belong to different sets of hypothesis, which are not exactly equivalent to each other.

For `modsem_da` and `modsem_mplus` objects, the interaction term is not a formal variable in the model and therefore lacks a defined variance. Under assumptions of normality and zero-mean variables, the interaction variance is estimated as:

$$var(xz) = var(x) * var(z) + cov(x, z)^2$$

This means the standardized estimate for the interaction differs from approaches like lavaan, which treats the interaction as a latent variable with unit variance.

For modsem_pi objects, the interaction term is standardized by default assuming var(xz) = 1, but this can be overridden using the correction argument.

NOTE: Standardized estimates are always placed in the est column, not est.std, regardless of model type.

Value

A data.frame with standardized estimates in the est column.

Methods (by class)

- standardized_estimates(lavaan): Method for lavaan objects
- standardized_estimates(modsem_da): Method for modsem_da objects
- standardized_estimates(modsem_mplus): Retrieve standardized estimates from modsem_mplus object.
- standardized_estimates(modsem_pi): Method for modsem_pi objects

Examples

```
m1 <- '
  # Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

  # Inner Model
  Y ~ X + Z + X:Z
'
# Double centering approach
est_dca <- modsem(m1, oneInt)

std1 <- standardized_estimates(est_dca) # no correction
summarize_partable(std1)

std2 <- standardized_estimates(est_dca, correction = TRUE) # apply correction
summarize_partable(std2)

## Not run:
est_lms <- modsem(m1, oneInt, method = "lms")
standardized_estimates(est_lms) # correction not relevant for lms

## End(Not run)
```

`standardize_model` *Standardize a fitted* `modsem_da` *model*

Description

`standardize_model()` post-processes the output of `modsem_da()` (or of `modsem()`) when `method = "lms"` / `method = "qml"`), replacing the *unstandardized* coefficient vector (`$coefs`) and its variance–covariance matrix (`$vcov`) with *fully standardized* counterparts (including the measurement model).The procedure is purely algebraic— **no re-estimation is carried out** —and is therefore fast and deterministic.

Usage

```
standardize_model(model, monte.carlo = FALSE, mc.reps = 10000, ...)
```

Arguments

`model`	A fitted object of class `modsem_da`. Passing any other object triggers an error.
`monte.carlo`	Logical. If `TRUE`, the function will use Monte Carlo simulation to obtain the standard errors of the standardized estimates. If `FALSE`, the delta method is used. Default is `FALSE`.
`mc.reps`	Number of Monte Carlo replications. Default is 10,000. Ignored if `monte.carlo = FALSE`.
`...`	Arguments passed on to other functions

Value

The same object (returned invisibly) with three slots overwritten

`$parTable` Parameter table whose columns `est` and `std.error` now hold standardized estimates and their (delta-method) standard errors, as produced by `standardized_estimates()`.

`$coefs` Named numeric vector of standardized coefficients. Intercepts (operator `~1`) are removed, because a standardized variable has mean 0 by definition.

`$vcov` Variance–covariance matrix corresponding to the updated coefficient vector. Rows/columns for intercepts are dropped, and the sub-matrix associated with rescaled parameters is adjusted so that its diagonal equals the squared standardized standard errors.

The object keeps its class attributes, allowing seamless use by downstream S3 methods such as `summary()`, `coef()`, or `vcov()`.

Because the function merely transforms existing estimates, *parameter constraints imposed through shared labels remain satisfied*.

See Also

`standardized_estimates()` Obtains the fully standardized parameter table used here.

`modsem()` Fit model using LMS or QML approaches.

`modsem_da()` Fit model using LMS or QML approaches.

Examples

```
## Not run:
# Latent interaction estimated with LMS and standardized afterwards
syntax <- "
  X =~ x1 + x2 + x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3
  Y ~ X + Z + X:Z
"
fit  <- modsem_da(syntax, data = oneInt, method = "lms")
sfit <- standardize_model(fit, monte.carlo = TRUE)

# Compare unstandardized vs. standardized summaries
summary(fit)  # unstandardized
summary(sfit) # standardized

## End(Not run)
```

---

summarize_partable *Summarize a parameter table from a* modsem *model.*

Description

Summarize a parameter table from a modsem model.

Usage

```
summarize_partable(
  parTable,
  scientific = FALSE,
  ci = FALSE,
  digits = 3,
  loadings = TRUE,
  regressions = TRUE,
  covariances = TRUE,
  intercepts = TRUE,
  variances = TRUE
)
```

Arguments

parTable	A parameter table, typically obtained from a modsem model using parameter_estimates or standardized_estimates.
scientific	Logical, whether to print p-values in scientific notation.
ci	Logical, whether to include confidence intervals in the output.
digits	Integer, number of digits to round the estimates to (default is 3).

loadings	Logical, whether to include factor loadings in the output.
regressions	Logical, whether to include regression coefficients in the output.
covariances	Logical, whether to include covariance estimates in the output.
intercepts	Logical, whether to include intercepts in the output.
variances	Logical, whether to include variance estimates in the output.

Value

A summary object containing the parameter table and additional information.

Examples

```
m1 <- '
  # Outer Model
  X =~ x1 + x2 + x3
  Z =~ z1 + z2 + z3
  Y =~ y1 + y2 + y3

  # Inner Model
  Y ~ X + Z + X:Z
'
# Double centering approach
est_dca <- modsem(m1, oneInt)

std <- standardized_estimates(est_dca, correction = TRUE)
summarize_partable(std)
```

---

summary.modsem_da *summary for modsem objects*

Description

summary for modsem objects

summary for modsem objects

summary for modsem objects

Usage

```
## S3 method for class 'modsem_da'
summary(
  object,
  H0 = is_interaction_model(object),
  verbose = interactive(),
  r.squared = TRUE,
  fit = FALSE,
  adjusted.stat = FALSE,
  digits = 3,
```

```
  scientific = FALSE,
  ci = FALSE,
  standardized = FALSE,
  centered = FALSE,
  monte.carlo = FALSE,
  mc.reps = 10000,
  loadings = TRUE,
  regressions = TRUE,
  covariances = TRUE,
  intercepts = TRUE,
  variances = TRUE,
  var.interaction = FALSE,
  ...
)

## S3 method for class 'modsem_mplus'
summary(
  object,
  scientific = FALSE,
  standardized = FALSE,
  ci = FALSE,
  digits = 3,
  loadings = TRUE,
  regressions = TRUE,
  covariances = TRUE,
  intercepts = TRUE,
  variances = TRUE,
  ...
)

## S3 method for class 'modsem_pi'
summary(
  object,
  H0 = is_interaction_model(object),
  r.squared = TRUE,
  adjusted.stat = FALSE,
  digits = 3,
  scientific = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

object	modsem object to summarized
H0	Should the baseline model be estimated, and used to produce comparative fit?
verbose	Should messages be printed?
r.squared	Calculate R-squared.

fit	Print additional fit measures.
adjusted.stat	Should sample size corrected/adjustes AIC and BIC be reported?
digits	Number of digits for printed numerical values
scientific	Should scientific format be used for p-values?
ci	print confidence intervals
standardized	standardize estimates
centered	Print mean centered estimates.
monte.carlo	Should Monte Carlo bootstrapped standard errors be used? Only relevant if standardized = TRUE. If FALSE delta method is used instead.
mc.reps	Number of Monte Carlo repetitions. Only relevant if monte.carlo = TRUE, and standardized = TRUE.
loadings	print loadings
regressions	print regressions
covariances	print covariances
intercepts	print intercepts
variances	print variances
var.interaction	If FALSE variances for interaction terms will be removed from the output.
...	arguments passed to lavaan::summary()

Examples

```
## Not run:
m1 <- "
 # Outer Model
 X =~ x1 + x2 + x3
 Y =~ y1 + y2 + y3
 Z =~ z1 + z2 + z3

 # Inner model
 Y ~ X + Z + X:Z
"

est1 <- modsem(m1, oneInt, "qml")
summary(est1, ci = TRUE, scientific = TRUE)

## End(Not run)
```

TPB *TPB*

Description

A simulated dataset based on the Theory of Planned Behaviour

Examples

```
tpb <- "
# Outer Model (Based on Hagger et al., 2007)
  ATT =~ att1 + att2 + att3 + att4 + att5
  SN =~ sn1 + sn2
  PBC =~ pbc1 + pbc2 + pbc3
  INT =~ int1 + int2 + int3
  BEH =~ b1 + b2

# Inner Model (Based on Steinmetz et al., 2011)
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC + INT:PBC
"

est <- modsem(tpb, data = TPB)
summary(est)
```

TPB_1SO *TPB_1SO*

Description

A simulated dataset based on the Theory of Planned Behaviour, where INT is a higher order construct of ATT, SN, and PBC.

Examples

```
tpb <- '
  # First order constructs
  ATT =~ att1 + att2 + att3
  SN  =~ sn1 + sn2 + sn3
  PBC =~ pbc1 + pbc2 + pbc3
  BEH =~ b1 + b2

  # Higher order constructs
  INT =~ ATT + PBC + SN

  # Higher order interaction
  INTxPBC =~ ATT:PBC + SN:PBC + PBC:PBC
```

```
  # Structural model
  BEH ~ PBC + INT + INTxPBC
'

## Not run:
est_ca <- modsem(tpb, data = TPB_1SO, method = "ca")
summary(est_ca)

est_dblcent  <- modsem(tpb, data = TPB_1SO, method = "dblcent")
summary(est_dblcent)

## End(Not run)
```

TPB_2SO *TPB_2SO*

Description

A simulated dataset based on the Theory of Planned Behaviour, where INT is a higher order construct of ATT and SN, and PBC is a higher order construct of PC and PB.

Examples

```
tpb <- '
  # First order constructs
  ATT =~ att1 + att2 + att3
  SN  =~ sn1 + sn2 + sn3
  PB =~ pb1 + pb2 + pb3
  PC =~ pc1 + pc2 + pc3
  BEH =~ b1 + b2

  # Higher order constructs
  INT =~ ATT + SN
  PBC =~ PC + PB

  # Higher order interaction
  INTxPBC =~ ATT:PC + ATT:PB + SN:PC + SN:PB

  # Structural model
  BEH ~ PBC + INT + INTxPBC
'

## Not run:
est <- modsem(tpb, data = TPB_2SO, method = "ca")
summary(est)

## End(Not run)
```

TPB_UK *TPB_UK*

Description

A dataset based on the Theory of Planned Behaviour from a UK sample. 4 variables with high communality were selected for each latent variable (ATT, SN, PBC, INT, BEH), from two time points (t1 and t2).

Source

Gathered from a replication study by Hagger et al. (2023).

Obtained from doi:10.23668/psycharchives.12187

Examples

```
tpb_uk <- "
# Outer Model (Based on Hagger et al., 2007)
 ATT =~ att3 + att2 + att1 + att4
 SN =~ sn4 + sn2 + sn3 + sn1
 PBC =~ pbc2 + pbc1 + pbc3 + pbc4
 INT =~ int2 + int1 + int3 + int4
 BEH =~ beh3 + beh2 + beh1 + beh4

# Inner Model (Based on Steinmetz et al., 2011)
 # Causal Relationsships
 INT ~ ATT + SN + PBC
 BEH ~ INT + PBC
 BEH ~ INT:PBC
"

est <- modsem(tpb_uk, data = TPB_UK)
summary(est)
```

trace_path *Estimate formulas for (co-)variance paths using Wright's path tracing rules*

Description

This function estimates the path from `x` to `y` using the path tracing rules. Note that it only works with structural parameters, so `"=~"` are ignored, unless `measurement.model = TRUE`. If you want to use the measurement model, `"~"` should be in the `mod` column of `pt`.

Usage

```
trace_path(
  pt,
  x,
  y,
  parenthesis = TRUE,
  missing.cov = FALSE,
  measurement.model = TRUE,
  maxlen = 100,
  paramCol = "mod",
  ...
)
```

Arguments

pt	A data frame with columns lhs, op, rhs, and mod, from modsemify
x	Source variable
y	Destination variable
parenthesis	If TRUE, the output will be enclosed in parenthesis
missing.cov	If TRUE, covariances missing from the model syntax will be added
measurement.model	If TRUE, the function will use the measurement model
maxlen	Maximum length of a path before aborting
paramCol	The column name in pt that contains the parameter labels
...	Additional arguments passed to trace_path

Value

A string with the estimated path (simplified if possible)

Examples

```
library(modsem)
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'
pt <- modsemify(m1)
trace_path(pt, x = "Y", y = "Y", missing.cov = TRUE) # variance of Y
```

twostep *Estimate interaction effects in structural equation models (SEMs) using a twostep procedure*

Description

Estimate an interaction model using a twostep procedure. For the PI approaches, the `lavaan::sam` function is used to optimize the models, instead of `lavaan::sem`. Note that the product indicators are still used, and not the newly developed SAM approach to estimate latent interactions. For the DA approaches (LMS and QML) the measurement model is estimated using a CFA (`lavaan::cfa`). The structural model is estimated using `modsem_da`, where the estimates in the measurement model are fixed, based on the CFA estimates. Note that standard errors are uncorrected (i.e., naive), and do not account for the uncertainty in the CFA estimates. **NOTE, this is an experimental feature!**

Usage

```
twostep(model.syntax, data, method = "lms", ...)
```

Arguments

`model.syntax`	`lavaan` syntax
`data`	dataframe
`method`	method to use: `"dblcent"` double centering approach (passed to `lavaan`). `"ca"` constrained approach (passed to `lavaan`). `"rca"` residual centering approach (passed to `lavaan`). `"uca"` unconstrained approach (passed to `lavaan`). `"pind"` prod ind approach, with no constraints or centering (passed to `lavaan`). `"lms"` latent moderated structural equations (not passed to `lavaan`). `"qml"` quasi maximum likelihood estimation (not passed to `lavaan`). `"custom"` use parameters specified in the function call (passed to `lavaan`).
`...`	arguments passed to other functions depending on the method (see `modsem_pi` and `modsem_da`)

Value

`modsem` object with class `modsem_pi` or `modsem_da`.

Examples

```
library(modsem)
m1 <- '
  # Outer Model
  X =~ x1 + x2 +x3
  Y =~ y1 + y2 + y3
```

```
  Z =~ z1 + z2 + z3

  # Inner model
  Y ~ X + Z + X:Z
'

est_dblcent <- twostep(m1, oneInt, method = "dblcent")
summary(est_dblcent)

## Not run:
est_lms <- twostep(m1, oneInt, method = "lms")
summary(est_lms)

est_qml <- twostep(m1, oneInt, method = "qml")
summary(est_qml)

## End(Not run)

tpb_uk <- "
# Outer Model (Based on Hagger et al., 2007)
 ATT =~ att3 + att2 + att1 + att4
 SN =~ sn4 + sn2 + sn3 + sn1
 PBC =~ pbc2 + pbc1 + pbc3 + pbc4
 INT =~ int2 + int1 + int3 + int4
 BEH =~ beh3 + beh2 + beh1 + beh4

# Inner Model (Based on Steinmetz et al., 2011)
 # Causal Relationsships
 INT ~ ATT + SN + PBC
 BEH ~ INT + PBC
 BEH ~ INT:PBC
"

uk_dblcent <- twostep(tpb_uk, TPB_UK, method = "dblcent")
summary(uk_dblcent)

## Not run:
  uk_qml <- twostep(tpb_uk, TPB_UK, method = "qml")

  uk_lms <- twostep(tpb_uk, TPB_UK, method = "lms", nodes = 32, adaptive.quad = TRUE)
  summary(uk_lms)

## End(Not run)
```

`var_interactions` *Extract or modify parTable from an estimated model with estimated variances of interaction terms*

Description

Extract or modify parTable from an estimated model with estimated variances of interaction terms

Usage

```
var_interactions(object, ...)
```

Arguments

object	An object of class modsem_da, modsem_mplus, or a parTable of class data.frame
...	Additional arguments passed to other functions

Index