Package 'modsem'

June 13, 2025

Type Package

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

Version 1.0.10

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) (temporarily unavailable)
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 7.3.2

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, purrr, stringr, lavaan, rlang, MplusAutomation, nlme, dplyr, mvnfast, stats, fastGHQuad, mvtnorm, ggplot2, parallel, plotly

Depends R (>= 4.1.0)

URL https://modsem.org

Suggests knitr, rmarkdown

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 2025-06-13 08:10:02 UTC

Contents

`compare_fit` *compare model fit for* `modsem` *models*

Description

Compare the fit of two models using the likelihood ratio test (LRT). 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`. 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) and compares the fit of both models. The baseline model is estimated by removing all interaction terms from the model.

Usage

```
estimate_h0(object, warn_no_interaction = 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.

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)
```

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?

get_pi_data *Get data with product indicators for different approaches*

Description

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

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)
```

---

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 indicators in lavaan.

Usage

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

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
...	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)
```

`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 eacademicample, 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)

## 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-based approaches ("dblcent", "rca", "uca", "ca", "pind") 2. Distributionally 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: "rca" = residual centering approach (passed to lavaan), "uca" = unconstrained approach (passed to lavaan), "dblcent" = double centering approach (passed to lavaan), "pind" = prod ind approach, with no constraints or centering (passed to lavaan), "lms" = latent model structural equations (not passed to lavaan), "qml" = quasi maximum likelihood estimation of latent model structural equations (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, 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", EFIM.S=1000)
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")
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 since coef is not in the namespace of modsem, but stats

Usage

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

Arguments

`object`	fittet 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,
  method = "lms",
  verbose = NULL,
  optimize = NULL,
  nodes = 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,
  n.threads = NULL,
  algorithm = NULL,
  em.control = NULL,
  ...
)
```

Arguments

model.syntax	lavaan syntax
data	dataframe
method	method to use: "lms" = latent model structural equations (not passed to lavaan). "qml" = quasi maximum likelihood estimation of latent model structural equations (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. For large numbers of nodes, you might want to change the 'quad.range' argument.
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.

NOTE: It is recommended that you estimate the model normally and then standardize the output using standardize_model standardized_estimates, summary(<modsem_da-objec standardize=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 standardized_estimates.

`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 (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., 1000-10000), but it might drasticly increase computation time.

`OFIM.hessian`
: should the observed Fisher information be computed using the Hessian? If `FALSE`, it is computed using the gradient.

`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? Meant to be used for QML, can be unreliable with the LMS approach.

`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.
n.threads	number of cores 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.
...	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
```

```
est1 <- modsem_da(m1, oneInt, method = "qml")
summary(est1)

# 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
"

# LMS Approach
estTpb <- modsem_da(tpb, data = TPB, method = lms, EFIM.S = 1000)
summary(estTpb)

## End(Not run)
```

`modsem_inspect` *Inspect model information*

Description

function used to inspect fittet object. similar to 'lavInspect()' argument 'what' decides what to inspect

Usage

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

Arguments

`object`	fittet model to inspect
`what`	what to inspect
`...`	Additional arguments passed to other functions

Details

for ‘modsem_da‘, and ‘modsem_lavaan‘ for ‘modsem_lavaan‘, it is just a wrapper for ‘lavInspect()‘ for ‘modsem_da‘ and “ what can either be "all", "matrices", "optim", or just the name of what to extract.

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

Description

Estimation latent interactions through `Mplus`

Usage

```
modsem_mplus(
  model.syntax,
  data,
  estimator = "ml",
  type = "random",
  algorithm = "integration",
  process = 8,
  integration = 15,
  ...
)
```

Arguments

`model.syntax`	lavaan/modsem syntax
`data`	dataset
`estimator`	estimator argument passed to `Mplus`
`type`	type argument passed to `Mplus`
`algorithm`	algorithm argument passed to `Mplus`
`process`	process argument passed to `Mplus`
`integration`	integration argument passed to `Mplus`
`...`	arguments passed to other functions

Value

modsem_mplus object

Examples

```
# 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 Relationsships
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  BEH ~ INT:PBC
'

## Not run:
est_mplus <- modsem_mplus(tpb, data = TPB)
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 since nobs is not in the namespace of modsem, but stats

Usage

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

Arguments

`object`	fittet 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,
  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,
  constrained.res.cov.method = 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,
  ...
)
```

Arguments

`model.syntax`	lavaan syntax
`data`	dataframe

`method` method to use: `"rca"` = residual centering approach (passed to `lavaan`), `"uca"` = unconstrained approach (passed to `lavaan`), `"dblcent"` = double centering approach (passed to `lavaan`), `"pind"` = prod ind approach, with no constraints or centering (passed to `lavaan`), `"custom"` = use parameters specified in the function call (passed to `lavaan`)

`match` should the product indicators be created by using the match-strategy

`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 (overwritten by `method`, if `method != NULL`)

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

`residuals.prods` should indicator products be centered using residuals (overwritten by `method`, if `method != NULL`)

`residual.cov.syntax` should syntax for residual covariances be produced (overwritten by `method`, if `method != NULL`)

`constrained.prod.mean` should syntax for product mean be produced (overwritten by `method`, if `method != NULL`)

`constrained.loadings` should syntax for constrained loadings be produced (overwritten by `method`, if `method != NULL`)

`constrained.var` should syntax for constrained variances be produced (overwritten by `method`, if `method != NULL`)

`constrained.res.cov.method` method for constraining residual covariances

`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?

... : arguments passed to lavaan::sem()

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
est1 <- modsem_pi(m1, oneInt)
summary(est1)

## Not run:
# The Constrained Approach
est1Constrained <- modsem_pi(m1, oneInt, method = "ca")
summary(est1Constrained)

## 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
estTpb <- modsem_pi(tpb, data = TPB)
summary(estTpb)

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

## 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 since vcov is not in the namespace of modsem, but stats

Usage

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

Arguments

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

`multiplyIndicatorsCpp` *Multiply indicators*

Description

Multiply indicators

Usage

```
multiplyIndicatorsCpp(df)
```

Arguments

`df`	A data DataFrame

Value

A NumericVector

oneInt *oneInt*

Description

A simulated dataset with one interaction effect

parameter_estimates *Extract parameterEstimates from an estimated model*

Description

Extract parameterEstimates from an estimated model

Usage

```
parameter_estimates(object, ...)
```

Arguments

object	An object of class modsem_pi, modsem_da, or modsem_mplus
...	Additional arguments passed to other functions

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,
  xz = NULL,
  vals_x = seq(-3, 3, 0.001),
  vals_z,
  model,
  alpha_se = 0.15,
  digits = 2,
  ci_width = 0.95,
  ci_type = "confidence",
  rescale = TRUE,
  standardized = 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.
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.
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.
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.
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?
...	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")

# 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,
  xz = NULL,
  model,
  min_z = -3,
  max_z = 3,
  sig.level = 0.05,
  alpha = 0.2,
```

```
  detail = 1000,
  sd.line = 2,
  standardized = FALSE,
  ...
)
```

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).
xz	The name of the interaction term. If not specified, it will be created using x and z.
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 significance level 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?
...	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,
  xz = NULL,
  model,
  min_x = -3,
  max_x = 3,
  min_z = -3,
  max_z = 3,
  standardized = FALSE,
  detail = 0.01,
  ...
)
```

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.
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.
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‘.
...	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

```
## Not run:
m1 <- "
# Outer Model
  X =~ x1
```

```
  X =~ 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)

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)
  # Causal Relationsships
  INT ~ ATT + SN + PBC
  BEH ~ INT + PBC
  # BEH ~ ATT:PBC
  BEH ~ PBC:INT
  # BEH ~ PBC:PBC
"

est2 <- modsem(tpb, TPB, method = "lms")
plot_surface(x = "INT", z = "PBC", y = "BEH", model = est1)

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

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.
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.
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?
...	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. 3. If `xz` is not provided, it will be constructed by combining `x` and `z` with a colon (`":"`). In certain SEM software, 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

Get standardized estimates

Usage

```
standardized_estimates(object, ...)
```

Arguments

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

Details

For modsem_da, and modsem_mplus objects, the interaction term is not standardized such that var(xz) = 1. The interaction term is not an actual variable in the model, meaning that it does not have a variance. It must therefore be calculated from the other parameters in the model. Assuming normality and zero-means, the variance is calculated as var(xz) = var(x) * var(z) + cov(x, z)^2. Thus setting the variance of the interaction term to 1 would only be 'correct' if the correlation between x and z is zero. This means that the standardized estimates for the interaction term will be different from those using lavaan, since there the interaction term is an actual latent variable in the model, with a standardized variance of 1.

standardized_estimates.modsem_da *Get standardized estimates with Monte Carlo bootstrapped standard errors*

Description

Get standardized estimates with Monte Carlo bootstrapped standard errors

Usage

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

Arguments

object	An object of class modsem_da
monte.carlo	Should standard errors be calculated using Monte Carlo simulation. if FALSE the delta method is used instead. Default is FALSE.
mc.reps	Number of Monte Carlo repetitions
tolerance.zero	Tolerance for zero values. Standard errors smaller than this value will be set to NA.
...	Additional arguments passed to other functions

Details

The interaction term is not standardized such that var(xz) = 1. The interaction term is not an actual variable in the model, meaning that it does not have a variance. It must therefore be calculated from the other parameters in the model. Assuming normality and zero-means, the variance is calculated as var(xz) = var(x) * var(z) + cov(x, z)^2. Thus setting the variance of the interaction term to 1 would only be 'correct' if the correlation between x and z is zero. This means that the standardized estimates for the interaction term will be different from those using lavaan, since there the interaction term is an actual latent variable in the model, with a standardized variance of 1.

---

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)
```

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 = TRUE,
  verbose = interactive(),
  r.squared = TRUE,
  adjusted.stat = FALSE,
  digits = 3,
  scientific = FALSE,
  ci = FALSE,
  standardized = FALSE,
  monte.carlo = FALSE,
  mc.reps = 10000,
  loadings = TRUE,
  regressions = TRUE,
  covariances = TRUE,
  intercepts = !standardized,
  variances = TRUE,
  var.interaction = FALSE,
  ...
)

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

```
)

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

Arguments

object	modsem object to summarized
H0	should a null model be estimated (used for comparison)
verbose	print progress for the estimation of null model
r.squared	calculate R-squared
adjusted.stat	should sample size corrected/adjustes AIC and BIC be reported?
digits	number of digits to print
scientific	print p-values in scientific notation
ci	print confidence intervals
standardized	print standardized 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 (default) variances for interaction terms will be removed (if present)
...	arguments passed to lavaan::summary()
standardize	standardize estimates

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)
```

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 <- modsem(tpb, data = TPB_2SO, method = "ca")
summary(est)

## 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 replciation study of the original by Hagger et al. (2023). Obtained from https://doi.org/10.23668/psycharchiv

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)
```

`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 = FALSE,
  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
```

`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