Package 'semrulesid'

September 26, 2026

Type Package

Title Evaluate Structural Equation Model Identification Rules

Version 0.4.1

Description Evaluates selected necessary and sufficient identification conditions in structural equation models (SEMs), including latent-variable scaling constraints. Output reports rule status and applicability and provides diagnostic messages to support model specification and respecification. The package is intended as a diagnostic aid and does not implement a universal identification algorithm. For more details, see Bollen (2026, ISBN:978-1009312820).

License GPL (>= 3)

Encoding UTF-8

Imports lavaan

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

Config/roxygen2/version 8.1.0

URL https://github.com/zacharyvig/semrulesid

BugReports https://github.com/zacharyvig/semrulesid/issues

NeedsCompilation no

Author Zach Vig [aut, cre, cph]

Maintainer Zach Vig <zachvig@rocketmail.com>

Repository CRAN

Date/Publication 2026-09-26 16:20:07 UTC

Contents

`get_rules` *Gather identification rules as a list*

Description

The 'semrulesid' package stores identification rule functions internally. This function makes them available to the user as a list.

Usage

```
get_rules(rule = "*", model_type = "*")
```

Arguments

`rule`	A character vector specifying the name of the rule as it's defined in the package. Use "*" to get all rules (or all rules of the defined model type). Partial matches are acceptable.
`model_type`	A character vector specifying the model sub-type from which to get rules. Sub-types include "reg" (simultaneous equations models/regression models) and "cfa" (confirmatory factor analysis models). Use "sem" to get rules that apply to all structural equation models. Use "*" to get all rules in the package.

Value

A list object with the rule function(s) specified by the user.

Examples

```
# Get all rules for a CFA model
rules <- get_rules(rule = "*", model_type = "cfa")
# Get a specific rule for a SEM model
rules <- get_rules(rule = "latent_scaling", model_type = "sem")
latent_scaling_rule <- rules[[1]]
```

id *Evaluate common Structural Equation Model (SEM) identification rules*

Description

This is the "workhorse" function of the semrulesid package. The user supplies a model string in lavaan syntax (see model.syntax for more details) and the function prints an informative table to the console about the status of the model on a variety of common identification rules.

Usage

```
id(x, print_msgs = TRUE, lav_fun = "sem", twostep = FALSE, ...)

id2(x, print_msgs = TRUE, lav_fun = "sem", ...)
```

Arguments

x	A character string model in lavaan syntax, a lavaan parameter table, or a fitted lavaan object.
print_msgs	Logical. If TRUE, the output will include why a rule does not pass or is not applicable, along with any other helpful information. Default: TRUE.
lav_fun	A character string specifying the lavaan function you intend to use to fit the model. This will ensure the correct model defaults are specified. Options currently include "lavaan", "sem", or "cfa". If a parameter table or fit object are supplied, this argument is ignored. Default: "sem".
twostep	A logical indicating whether to use the two-step identification rule instead of the usual one-step. See details. Default: FALSE.
...	Additional arguments passed to the lavaanify function from lavaan. See lavaanify for more information. If parameter tables or fitted model objects are supplied, these arguments are ignored.

Details

The primary output of id() is a table printed to the console, where rows correspond to rules, and columns include "Pass" (did the rule pass?), "Necessary" (is the rule necessary for identification?), and "Sufficient" (is the rule sufficient for identification?). These columns can take values "Yes", "No", or be left blank in the case of NA values.

If the user set the print_msgs argument to "TRUE" (which is the default), a column labeled "Messages" is appended to the table, and an output section called "Messages" is printed below the table. Messages include why a rule failed, why a rule is not applicable to the current model, or why the necessary/sufficient conditions may not apply as usual.

Messages are identified by a number, and corresponding message numbers are listed in the "Messages" column of the table.

`lav_fun` takes character values "lavaan", "sem", or "cfa", specifying which `lavaan` function the user intends to call (and thus which defaults should be used) when fitting the model in the case a model string is supplied. Supplying a parameter table or fitted model object ignores the `lav_fun` argument since defaults will have already been implemented. In these cases, you can set `lav_fun = NA` to avoid warnings about the argument being ignored. For `id_mplus()`, `lav_fun` is only needed if the user supplies an Mplus model string due to how `lavaan::lav_mplus_syntax_model()` works. If an Mplus input file is supplied, `lav_fun` is ignored since `lavaan::lav_mplus_lavaan()` always uses `sem()` defaults.

`id2` is a wrapper function for calling `id` with argument `twostep` set to `TRUE`. The two-step method parses an SEM into a CFA model and a latent variable/structural model, and evaluates the identification rules on each. If both parts are identified, the whole model is identified.

Value

An object of class `semid` or `semid2` (if `twostep = TRUE` or `id2` is called. See details.)

Examples

```
my_model <- ' L1 =~ x1 + x2 + x3
              L2 =~ x4 + x5 + x6
              L3 =~ x7 + x8 + x9
              L2 ~ L1
              L3 ~ L2 '
id(my_model, print_msgs = TRUE, lav_fun = "sem",
   meanstructure = FALSE)
id2(my_model, print_msgs = TRUE, lav_fun = "sem",
   meanstructure = FALSE)
```

`id_mplus` *Evaluate identification rules for an Mplus model*

Description

This is a wrapper function for `id` that handles Mplus model syntax (as a string) or Mplus input files (with extension ".inp"). It internally converts the Mplus model to a lavaan model, then calls `id` using the specified arguments.

Usage

```
id_mplus(x, print_msgs = TRUE, lav_fun = "sem", twostep = FALSE, ...)
```

Arguments

`x`	A character string model in Mplus syntax, or a path to an Mplus input file.
`print_msgs`	Logical. If `TRUE`, the output will include why a rule does not pass or is not applicable, along with any other helpful information. Default: `TRUE`.

lav_fun	A character string specifying the lavaan function you intend to use to fit the model. This will ensure the correct model defaults are specified. Options currently include "lavaan", "sem", or "cfa". If a parameter table or fit object are supplied, this argument is ignored. Default: "sem".
twostep	A logical indicating whether to use the two-step identification rule instead of the usual one-step. See details. Default: FALSE.
...	Additional arguments passed to the lavaanify function from lavaan. See lavaanify for more information. If parameter tables or fitted model objects are supplied, these arguments are ignored.

Value

An object of class semid or semid2 (if twostep = TRUE).

Examples

```
my_model <- ' L1 BY x1 x2 x3;
              L2 BY x4 x5 x6;
              L3 BY x7 x8 x9;
              L2 ON L1;
              L3 ON L2; '
id_mplus(my_model, print_msgs = TRUE, lav_fun = "sem",
   meanstructure = FALSE)
```

---

print.semid *Printing function for rules list*

Description

Printing function for rules list

Usage

```
## S3 method for class 'semid'
print(
  x,
  col_names = c("", "Pass", "Necessary", "Sufficient"),
  print_msgs = NULL,
  msgs_name = "Message",
  window = 56L,
  pos_lab = "Yes",
  neg_lab = "No",
  na_lab = "-",
  print_version = TRUE,
  print_lav_fun = TRUE,
  print_model_type = TRUE,
```

```
  name_value_sep = ":",
  na_rule_policy = c("footnote", "hide", "show"),
  ...
)
```

Arguments

x	A semid object
col_names	Character vector. The names of the columns of the main table
print_msgs	Logical. If TRUE messages are printed
msgs_name	Character. The name of the messages index column
window	Integer. The width of the output window
pos_lab	Character. The label for positive cells, e.g., "Yes".
neg_lab	Character. The label for negative cells, e.g., "No".
na_lab	Character. The label for NA/blank cells.
print_version	Logical. If TRUE, the version of the package is printed in a header before the rules output.
print_lav_fun	Logical. If TRUE, the lavaan function is printed in a header before the rules output.
print_model_type	Logical. If TRUE, the model type is printed in a header before the rules output.
name_value_sep	Character. The separator between the meta labels and values.
na_rule_policy	Character. How to handle rules that are not applicable to the model. Options are "hide" (default), "footnote", or "show". Option "hide" does not print them at all; option "footnote", prints their names only in a footnote after the main table; option "show" prints them in the main table with a message indicating they are not applicable.
...	Arguments passed to print.semscale if applicable.

Value

The function prints the rules list to the console and returns the semid object invisibly.

---

print.semid2 *Printing function for two-step rules lists*

Description

Printing function for two-step rules lists

Usage

```
## S3 method for class 'semid2'
print(
  x,
  ...,
  step_names = c("Measurement Model", "Latent Variable/Structural Model"),
  step_titles = c("Step 1", "Step 2"),
  print_version = TRUE,
  print_lav_fun = TRUE,
  print_model_type = TRUE,
  window = 56L,
  name_value_sep = ":"
)
```

Arguments

x	A semid2 object
...	Arguments to be passed to print.semid.
step_names	Character. The names of each step/block.
step_titles	Character. The labels to be given to each step.
print_version	Logical. If TRUE, the version of the package is printed in a header before the rules output.
print_lav_fun	Logical. If TRUE, the lavaan function is printed in a header before the rules output.
print_model_type	Logical. If TRUE, the model type is printed in a header before the rules output.
window	Integer. The width of the output window.
name_value_sep	Character. The separator between the meta labels and values.

Value

The function prints the two-step rules list to the console and returns the semid2 object invisibly.

---

print.semscale *Printing function for scaling table*

Description

Printing function for scaling table

Usage

```
## S3 method for class 'semscale'
print(
  x,
  ...,
  print_msgs = TRUE,
  window = 56L,
  sep_spaces = 3L,
  indent_lengths = c(0L, 2L, 2L),
  na_lab = "na",
  pos_lab = "Yes",
  neg_lab = "No",
  empty_lab = "None",
  bullet = "-",
  name_value_sep = ":",
  print_version = TRUE,
  print_lav_fun = TRUE
)
```

Arguments

x	A semscale object, i.e., a list of scaling information for each latent variable in the model.
...	Not currently used.
print_msgs	Logical. If TRUE messages are printed.
window	Integer. The width of the output window.
sep_spaces	Integer. The number of spaces to separate the row names from the row values.
indent_lengths	Integer vector of length 3. The number of spaces to indent for each level of information (currently there are three supported).
na_lab	Character. The label for NA/blank cells.
pos_lab	Character. The label for positive cells, e.g., "Yes".
neg_lab	Character. The label for negative cells, e.g., "No".
empty_lab	Character. The label for empty cells, e.g., "None".
bullet	Character. The bullet symbol for messages, e.g., "-".
name_value_sep	Character. The separator between the row names and row values.
print_version	Logical. If TRUE, the version of the package is printed in a header before the rules output.
print_lav_fun	Logical. If TRUE, the lavaan function is printed in a header before the rules output.

Value

The function prints the scaling table to the console and returns the semscale object invisibly.

scaling *Check if latent variables are correctly scaled*

Description

This function checks whether latent variables in the model are scaled, which is a necessary condition for model identification. A latent variable is scaled if it has a fixed loading on a scaling indicator or the latent variable has a fixed variance. When mean structure is present, the scaling indicator must also have a fixed intercept or the latent variable must also have a fixed mean. If any latent variable is not scaled, the model is not identified.

Usage

```
scaling(
  x,
  lav_fun = "sem",
  print_msgs = TRUE,
  lv = NULL,
  return_type = c("object", "logical"),
  ...
)

## S3 method for class 'lavaan'
scaling(
  x,
  lav_fun = "sem",
  print_msgs = TRUE,
  lv = NULL,
  return_type = c("object", "logical"),
  ...
)
```

Arguments

x	lavaan model syntax, a lavaan parameter table or a semrulesid ID object.
lav_fun	A character string specifying the function you intend to use to fit the model. This will ensure the correct model defaults are specified. Options currently include "lavaan", "sem", or "cfa". If a parameter table or fitted model object are supplied, this argument is ignored.
print_msgs	Logical. If TRUE the output will print why a latent variable is not scaled when applicable and the method or methods by which it is scaled when it is scaled. If FALSE, the output will only print whether the latent variable is scaled or not and a few other descriptives.
lv	An optional character vector of the specific latent variables you would like to check. Otherwise, all latent variables are extracted from the parameter table

`return_type` A character string specifying the type of output. Options include "logical" (a logical vector specifying whether each latent variable is correctly scaled) or "object" (an object with additional information about the scaling of each latent variable). The latter is of type `semscale` and has a custom print method for additional diagnosis of identification issues.

`...` Additional arguments passed to the `lavaanify` function from `lavaan`. See lavaanify for more information. These are only used when a model string is supplied. Otherwise, they are ignored.

Value

An object of class `semscale` with all scaling information (if `return_type = "object"`), or a logical vector (if `return_type = "logical"`) indicating whether each latent variable is correctly scaled.

Examples

```
my_model <- ' L1 =~ x1 + x2 + x3
              L2 =~ x4 + x5 + x6
              L3 =~ x7 + x8 + x9
              L2 ~ L1
              L3 ~ L2 '
scaling(my_model, print_msgs = TRUE, lav_fun = "sem",
        meanstructure = FALSE)
```

Index