Package ‘mxsem’

May 5, 2024

Type Package

Title Specify 'OpenMx' Models with a 'lavaan'-Style Syntax

Version 0.0.9

Maintainer Jannik H. Orzek <jannik.orzek@mailbox.org>

Description Provides a 'lavaan'-like syntax for 'OpenMx' models. The syntax supports
definition variables, bounds, and parameter transformations. This allows for
latent growth curve models with person-specific measurement occasions, moderated
nonlinear factor analysis and much more.

License GPL (>= 3)

Depends OpenMx

Imports Rcpp (>= 1.0.10), stats, methods, dplyr, utils

LinkingTo Rcpp

RooxygenNote 7.2.3

Encoding UTF-8

Suggests testthat (>= 3.0.0), lavaan, knitr, rmarkdown

Config/testthat/edition 3

URL https://jhorzek.github.io/mxsem/

VignetteBuilder knitr

NeedsCompilation yes

Author Jannik H. Orzek [aut, cre, cph]
  (<https://orcid.org/0000-0002-3123-2248>)

Repository CRAN

Date/Publication 2024-05-05 17:20:02 UTC

R topics documented:

  clean_syntax ................................................................. 2
  get_groups .................................................................. 2
  get_individual_algebra_results ......................................... 3

1
Description
takes in a lavaan style syntax and removes comments, white space, etc.

Usage
clean_syntax(syntax)

Arguments
syntax lavaan style syntax

Value
vector of strings with cleaned syntax

Sources
mxsem ................................................................. 4
mxsem_group_by .................................................. 10
parameters ......................................................... 11
parameter_table_rcpp ............................................. 11
printmulti_group_parameters .................................... 12
set_starting_values .............................................. 13
simulate_latent_growth_curve ................................... 14
simulate_moderated_nonlinear_factor_analysis ............... 15
summarize_multi_group_model .................................. 16
unicode_directed .................................................. 17
unicode_undirected ............................................... 17

get_groups

Description
returns a list of groups for a multi group model

Usage
groups(multi_group_model)

Arguments
multi_group_model multi group model created with mxsem_group_by
**get_individual_algebra_results**

**Value**

list with data for each group

**Examples**

```r
# THE FOLLOWING EXAMPLE IS ADAPTED FROM
# https://openmx.ssri.psu.edu/docs/OpenMx/latest/_static/Rdoc/mxModel.html
library(mxsem)

model <- 'spatial =~ visual + cubes + paper
      verbal =~ general + paragrap + sentence
      math =~ numeric + series + arithmet'

mg_model <- mxsem(model = model,
                data = OpenMx::HS.ability.data) |> 
    # we want separate models for all combinations of grades and schools:
    mxsem_group_by(grouping_variables = "school") |> 
    mxTryHard()

    # let's summarize the results:
    summarize_multi_group_model(mg_model)

    # let's get the groups:
    get_groups(mg_model)
```

**Description**

evaluates algebras for each subject in the data set. This function is useful if you have algebras with definition variables (e.g., in mmifa).

**Usage**

```r
get_individual_algebra_results(
    mxModel,
    algebra_names = NULL,
    progress_bar = TRUE
)
```

**Arguments**

- `mxModel` mxModel with algebras
- `algebra_names` optional: Only compute individual algebras for a subset of the parameters
- `progress_bar` should a progress bar be shown?
Value

a list of data frames. The list contains data frames for each of the algebras. The data frames contain the individual specific algebra results as well as all definition variables used to predict said algebra.

Examples

library(mxsem)

set.seed(123)
dataset <- simulate_moderated_nonlinear_factor_analysis(N = 50)

model <- "
  xi =~ x1 + x2 + x3
  eta =~ y1 + y2 + y3
  eta ~ {a := a0 + data.k*a1}*xi
"

fit <- mxsem(model = model,
             data = dataset) |> mxTryHard()

algebra_results <- get_individual_algebra_results(mxModel = fit,
                                                  progress_bar = FALSE)

# the following plot will only show two data points because there is only two values for the definition variable k (0 or 1).
plot(x = algebra_results[["a"]]["k],
     y = algebra_results[["a"]]["algebra_result"])

Description

Create an extended SEM with OpenMx (Boker et al., 2011) using a lavaan-style (Rosseel, 2012) syntax.

Usage

mxsem(
  model,
  data, scale_loadings = TRUE, scale_latent_variances = FALSE, add_intercepts = TRUE, add_variances = TRUE, add_exogenous_latent_covariances = TRUE, add_exogenous_manifest_covariances = TRUE, lbound_variances = TRUE,
directed = unicode_directed(),
undirected = unicode_undirected(),
return_parameter_table = FALSE
)

Arguments

model  model syntax similar to lavaan's syntax

data  raw data used to fit the model. Alternatively, an object created with OpenMx::mxData can be used (e.g., OpenMx::mxData(observed = cov(OpenMx::Bollen), means = colMeans(OpenMx::Bollen), numObs = nrow(OpenMx::Bollen), type = "cov"))

scale_loadings  should the first loading of each latent variable be used for scaling?

scale_latent_variances  should the latent variances be used for scaling?

add_intercepts  should intercepts for manifest variables be added automatically? If set to false, intercepts must be added manually. If no intercepts are added, mxsem will automatically use just the observed covariances and not the observed means.

add_variances  should variances for manifest and latent variables be added automatically?

add_exogenous_latent_covariances  should covariances between exogenous latent variables be added automatically?

add_exogenous_manifest_covariances  should covariances between exogenous manifest variables be added automatically?

lbound_variances  should the lower bound for variances be set to 0.000001?

directed  symbol used to indicate directed effects (regressions and loadings)

undirected  symbol used to indicate undirected effects (variances and covariances)

return_parameter_table  if set to TRUE, the internal parameter table is returned together with the mx-Model

Details

Setting up SEM can be tedious. The lavaan (Rosseel, 2012) package provides a great syntax to make the process easier. The objective of mxsem is to provide a similar syntax for OpenMx. OpenMx is a flexible R package for extended SEM. However, note that mxsem only covers a small part of the OpenMx framework by focusing on "standard" SEM. Similar to lavaan's sem()-function, mxsem tries to set up parts of the model automatically (e.g., adding variances automatically or scaling the latent variables automatically). If you want to unlock the full potential of OpenMx, mxsem may not be the best option.

Warning: The syntax and settings of mxsem may differ from lavaan in some cases. See vignette("Syntax", package = "mxsem") for more details on the syntax and the default arguments.

Alternatives:

You will find similar functions in the following packages:
• **metaSEM** (Cheung, 2015) provides a lavaan2RAM function that can be combined with the create.mxModel function. This combination offers more features than **mxsem**. For instance, constraints of the form $a < b$ are supported. In **mxsem** such constraints require algebras (e.g., $!\text{diff}; a := b - \exp(\text{diff})$).

• **umx** (Bates et al., 2019) provides the umxRAM and umxLav2RAM functions that can parse single lavaan-style statements (e.g., $\eta = y_1 + y_2 + y_3$) or an entire lavaan models to **OpenMx** models.

• **tidySEM** (van Lissa, 2023) provides the as_ram function to translate lavaan syntax to **OpenMx** and also implements a unified syntax to specify both, lavaan and OpenMx models. Additionally, it works well with the tidyverse.

• **ezMx** (Bates, et al. 2014) simplifies fitting SEM with **OpenMx** and also provides a translation of lavaan models to **OpenMx** with the lavaan.to.OpenMx function.

Because **mxsem** implements the syntax parser from scratch, it can extend the lavaan syntax to account for specific **OpenMx** features. This enables implicit transformations with curly braces.

**Citation:**
Cite **OpenMx** (Boker et al., 2011) for the modeling and lavaan for the syntax (Rosseel, 2012). **mxsem** itself is just a very small package and lets **OpenMx** do all the heavy lifting.

**Defaults:**
By default, **mxsem** scales latent variables by setting the loadings on the first item to 1. This can be changed by setting scale_loadings = FALSE in the function call. Setting scale_latent_variances = TRUE sets latent variances to 1 for scaling. **mxsem** will add intercepts for all manifest variables as well as variances for all manifest and latent variables. A lower bound of 1e-6 will be added to all variances. Finally, covariances for all exogenous variables will be added. All of these options can be changed when calling **mxsem**.

**Syntax:**
The syntax is, for the most part, identical to that of lavaan. The following specifies loadings of a latent variable $\eta$ on manifest variables $y_1$-$y_4$:

```
eta =~ y1 + y2 + y3
```

Regressions are specified with $\sim$:

```
xi =~ x1 + x2 + x3
eta =~ y1 + y2 + y3
# predict eta with xi:
eta ~ xi
```

Add covariances with $\sim\sim$

```
xi =~ x1 + x2 + x3
eta =~ y1 + y2 + y3
# predict eta with xi:
eta ~ xi
x1 ~ x2
```

Intercepts are specified with $\sim 1$
Parameters and constraints:  

Add labels to parameters as follows:

\[
\begin{align*}
\xi & = 11x_1 + 12x_2 + 13x_3 \\
\eta & = 14y_1 + 15y_2 + 16y_3 \\
\end{align*}
\]

# predict \eta with \xi:
\[\eta \sim b\xi\]

Fix parameters by using numeric values instead of labels:

\[
\begin{align*}
\xi & = 1x_1 + 12x_2 + 13x_3 \\
\eta & = 1y_1 + 15y_2 + 16y_3 \\
\end{align*}
\]

# predict \eta with \xi:
\[\eta \sim b\xi\]

Bounds:

Lower and upper bounds allow for constraints on parameters. For instance, a lower bound can prevent negative variances.

\[
\begin{align*}
\xi & = 1x_1 + 12x_2 + 13x_3 \\
\eta & = 1y_1 + 15y_2 + 16y_3 \\
\end{align*}
\]

# predict \eta with \xi:
\[\eta \sim b\xi\]

# residual variance for \eta
\[x_1 \sim v\eta_1\]

# bound:
\[v > 0\]

Upper bounds are specified with \(v < 10\). Note that the parameter label must always come first. The following is not allowed: \(0 < v\) or \(10 > v\).

(Non-)Linear constraints:

Assume that latent construct \eta was observed twice, where \eta_1 is the first observation and \eta_2 the second. We want to define the loadings of \eta_2 on its observations as \(l_1 + \delta l_1\). If \delta l_1 is zero, we have measurement invariance.

\[
\begin{align*}
\eta_1 & = 11y_1 + 12y_2 + 13y_3 \\
\eta_2 & = 14y_4 + 15y_5 + 16y_6 \\
\end{align*}
\]

# define new delta-parameter

!delta_1; !delta_2; !delta_3

# redefine l_4-l_6

l_4 := l_1 + delta_1 \\
l_5 := l_2 + delta_2 \\
l_6 := l_3 + delta_3

```
Alternatively, implicit transformations can be used as follows:

\[
\eta_1 \sim l_1 y_1 + l_2 y_2 + l_3 y_3 \\
\eta_2 \sim (l_1 + \delta_1) y_4 + (l_2 + \delta_2) y_5 + (l_3 + \delta_3) y_6
\]

Specific labels for the transformation results can also be provided:

\[
\eta_1 \sim l_1 y_1 + l_2 y_2 + l_3 y_3 \\
\eta_2 \sim (l_4 := l_1 + \delta_1) y_4 + (l_5 := l_2 + \delta_2) y_5 + (l_6 := l_3 + \delta_3) y_6
\]

This is inspired by the approach in metaSEM (Cheung, 2015).

**Definition variables:**
Definition variables allow for person-specific parameter constraints. Use the data.-prefix to specify definition variables.

\[
I \sim 1 y_1 + 1 y_2 + 1 y_3 + 1 y_4 + 1 y_5 \\
S \sim \text{data.t}_1 y_1 + \text{data.t}_2 y_2 + \text{data.t}_3 y_3 + \text{data.t}_4 y_4 + \text{data.t}_5 y_5 \\
I \sim \text{int}*1 \\
S \sim \text{slp}*1
\]

**Starting Values:**
mxsem differs from lavaan in the specification of starting values. Instead of providing starting values in the model syntax, the set_starting_values function is used.

**References:**

**Value**
mxModel object that can be fitted with mxRun or mxTryHard. If return_parameter_table is TRUE, a list with the mxModel and the parameter table is returned.
Examples

# THE FOLLOWING EXAMPLE IS ADAPTED FROM LAVAAN
library(mxsem)

model <- '
# latent variable definitions
ind60 <- x1 + x2 + x3
dem60 <- y1 + a1*y2 + b*y3 + c1*y4
dem65 <- y5 + a2*y6 + b*y7 + c2*y8

# regressions
dem60 ~ ind60
dem65 ~ ind60 + dem60

# residual correlations
y1 ~~ y5
y2 ~~ y4 + y6
y3 ~~ y7
y4 ~~ y8
y6 ~~ y8
',

fit <- mxsem(model = model,
             data = OpenMx::Bollen) |> mxTryHard()
omxGetParameters(fit)

model_transformations <- '
# latent variable definitions
ind60 <- x1 + x2 + x3
dem60 <- y1 + a1*y2 + b1*y3 + c1*y4
dem65 <- y5 + (a2 := a1 + delta_a)*y6 + (b2 := b1 + delta_b)*y7 + c2*y8

# regressions
dem60 ~ ind60
dem65 ~ ind60 + dem60

# residual correlations
y1 ~~ y5
y2 ~~ y4 + y6
y3 ~~ y7
y4 ~~ y8
y6 ~~ y8
',

fit <- mxsem(model = model_transformations,
             data = OpenMx::Bollen) |> mxTryHard()
omxGetParameters(fit)
mxsem_group_by

**Description**

creates a multi-group model from an OpenMx model.

**Usage**

```
mxsem_group_by(
  mxModel,
  grouping_variables,
  parameters = c(".*"),
  use_grepl = TRUE
)
```

**Arguments**

- `mxModel`: mxModel with the entire data
- `grouping_variables`: Variables used to split the data in groups
- `parameters`: the parameters that should be group specific. By default all parameters are group specific.
- `use_grepl`: if set to TRUE, grepl is used to check which parameters are group specific. For instance, if parameters = "a" and use_grepl = TRUE, all parameters whose label contains the letter "a" will be group specific. If use_grepl = FALSE only the parameter that has the label "a" is group specific.

**Details**

mxsem_group_by creates a multi-group model by splitting the data found in an mxModel object using dplyr’s group_by function. The general idea is as follows:

1. The function extracts the data from mxModel 2. The data is split using the group_by function of dplyr with the variables in grouping_variables 3. a separate model is set up for each group. All parameters that match those specified in the parameters argument are group specific

**Warning**: The multi-group model may differ from **lavaan**! For instance, **lavaan** will automatically set the latent variances for all but the first group free if the loadings are fixed to equality. Such automatic procedures are not yet implemented in **mxsem**.

**Value**

mxModel with multiple groups. Use get_groups to extract the groups
Examples

# THE FOLLOWING EXAMPLE IS ADAPTED FROM
# https://openmx.ssri.psu.edu/docs/OpenMx/latest/_static/Rdoc/mxModel.html
library(mxsem)

model <- 'spatial =~ visual + cubes + paper
  verbal =~ general + paragrap + sentence
  math =~ numeric + series + arithmet'

mg_model <- mxsem(model = model,
  data = OpenMx::HS.ability.data) |>  
  # we want separate models for all combinations of grades and schools:
  mxsem_group_by(grouping_variables = "school") |>  
  mxTryHard()

  # let's summarize the results:
  summarize_multi_group_model(mg_model)

\[
\text{parameters}
\]

\textit{Description}

Returns the parameter estimates of an mxModel. Wrapper for omxGetParameters

\textit{Usage}

\texttt{parameters(mxMod)}

\textit{Arguments}

- \texttt{mxMod} : mxModel object

\textit{Value}

vector with parameter estimates

\[
\text{parameter_table_rcpp}
\]

\textit{Description}

creates a parameter table from a lavaan like syntax
print.multi_group_parameters

Usage

parameter_table_rcpp(
  syntax,
  add_intercept,
  add_variance,
  add_exogenous_latent_covariances,
  add_exogenous_manifest_covariances,
  scale_latent_variance,
  scale_loading
)

Arguments

syntax lavaan like syntax
add_intercept should intercepts for manifest variables be automatically added?
add_variance should variances for all variables be automatically added?
add_exogenous_latent_covariances should covariances between exogenous latent variables be added automatically?
add_exogenous_manifest_covariances should covariances between exogenous manifest variables be added automatically?
scale_latent_variance should variances of latent variables be set to 1?
scale_loading should the first loading of each latent variable be set to 1?

Value

parameter table

print.multi_group_parameters

print the multi_group_parameters

Description

print the multi_group_parameters

Usage

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

Arguments

x object from summarize_multi_group_model
... not used
**Description**

set the starting values of an OpenMx model. This is just an interface to omxSetParameters.

**Usage**

```r
set_starting_values(mx_model, values)
```

**Arguments**

- `mx_model`: model of class `mxModel`
- `values`: vector with labeled parameter values

**Value**

`mxModel` with changed parameter values

**Examples**

```r
library(mxsem)

model <- ' 
  # latent variable definitions
  ind60 =~ x1 + x2 + x3
  dem60 =~ y1 + a1*y2 + b*y3 + c1*y4
  dem65 =~ y5 + a2*y6 + b*y7 + c2*y8
  
  # regressions
  dem60 ~ ind60
  dem65 ~ ind60 + dem60
  
  # residual correlations
  y1 ~~ y5
  y2 ~~ y4 + y6
  y3 ~~ y7
  y4 ~~ y8
  y6 ~~ y8

fit <- mxsem(model = model, 
             data = OpenMx::Bollen) |>
set_starting_values(values = c("a1" = .4, "c1" = .6)) |>
mxTryHard()
```
**simulate_latent_growth_curve**

**Description**

simulate data for a latent growth curve model with five measurement occasions. The time-distance between these occasions differs between subjects.

**Usage**

```r
simulate_latent_growth_curve(N = 100)
```

**Arguments**

- **N**
  - sample size

**Value**

data set with columns `y1-y5` (observations) and `t_1-t_5` (time of observation)

**Examples**

```r
set.seed(123)
dataset <- simulate_latent_growth_curve(N = 100)

model <- 
  I =~ 1*y1 + 1*y2 + 1*y3 + 1*y4 + 1*y5
  S =~ data.t_1 * y1 + data.t_2 * y2 + data.t_3 * y3 + data.t_4 * y4 + data.t_5 * y5

I ~ int*1
S ~ slp*1

# set intercepts of manifest variables to zero
y1 ~ 0*1; y2 ~ 0*1; y3 ~ 0*1; y4 ~ 0*1; y5 ~ 0*1;
```

```r
mod <- mxsem(model = model,
              data = dataset) |> mxTryHard()
```
simulate_moderated_nonlinear_factor_analysis

**Description**

simulate data for a moderated nonlinear factor analysis.

**Usage**

```r
simulate_moderated_nonlinear_factor_analysis(N)
```

**Arguments**

- `N` sample size

**Value**

data set with variables x1-x3 and y1-y3 representing repeated measurements of an affect measure. It is assumed that the autoregressive effect is different depending on covariate k

**Examples**

```r
library(mxsem)
set.seed(123)
dataset <- simulate_moderated_nonlinear_factor_analysis(N = 2000)

model <- "
  xi =~ x1 + x2 + x3
  eta =~ y1 + y2 + y3
  eta ~ a*xi

  # we need two new parameters: a0 and a1. These are created as follows:
  !a0
  !a1
  # Now, we redefine a to be a0 + k*a1, where k is found in the data
  a := a0 + data.k*a1
"

mod <- mxsem(model = model,
             data = dataset) |> mxTryHard()

omxGetParameters(mod)
```
summarize_multi_group_model

Description

summarize the results of a multi group model created with mxsem_group_by

Usage

summarize_multi_group_model(multi_group_model)

Arguments

multi_group_model
  multi group model created with mxsem_group_by

Value

list with group specific parameters and common parameters

Examples

# THE FOLLOWING EXAMPLE IS ADAPTED FROM
# https://openmx.ssri.psu.edu/docs/OpenMx/latest/_static/Rdoc/mxModel.html
library(mxsem)

model <- 'spatial = visual + cubes + paper
  verbal = general + paragrap + sentence
  math = numeric + series + arithmet'

mg_model <- mxsem(model = model,
  data = OpenMx::HS.ability.data) |>  # we want separate models for all combinations of grades and schools:
  mxsem_group_by(grouping_variables = "school") |>  
mxTryHard()

# let's summarize the results:
summarize_multi_group_model(mg_model)
**unicode_directed**

---

| unicode_directed | unicode_directed |

**Description**

this function returns the unicode for directed arrows

**Usage**

```python
unicode_directed()
```

**Value**

returns unicode for directed arrows

---

**unicode_undirected**

---

| unicode_undirected | unicode_undirected |

**Description**

this function returns the unicode for undirected arrows

**Usage**

```python
unicode_undirected()
```

**Value**

returns unicode for undirected arrows
Index

clean_syntax, 2
get_groups, 2
get_individual_algebra_results, 3

mxsem, 4
mxsem_group_by, 10

parameter_table_rcpp, 11
parameters, 11
print.multi_group_parameters, 12

set_starting_values, 13
simulate_latent_growth_curve, 14
simulate_moderated_nonlinear_factor_analysis, 15
summarize_multi_group_model, 16

unicode_directed, 17
unicode_undirected, 17