

# Package: mxsem (via r-universe)

August 27, 2024

**Type** Package

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

**Version** 0.1.0

**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

**RoxygenNote** 7.3.1

**Encoding** UTF-8

**Suggests** knitr, rmarkdown

**URL** <https://jhorzek.github.io/mxsem/>,  
<https://github.com/jhorzek/mxsem/>

**BugReports** <https://github.com/jhorzek/mxsem/issues>

**VignetteBuilder** knitr

**Repository** <https://jhorzek.r-universe.dev>

**RemoteUrl** <https://github.com/jhorzek/mxsem>

**RemoteRef** HEAD

**RemoteSha** 628402a8afce862a430c48a59315bb04a1787a36

## Contents

clean_syntax . . . . .	2
get_groups . . . . .	3
get_individual_algebra_results . . . . .	4
mxsem . . . . .	5
mxsem_group_by . . . . .	10
parameters . . . . .	11
parameter_table_rcpp . . . . .	12
print.multi_group_parameters . . . . .	13
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
<b>Index</b>	<b>18</b>

---

clean_syntax	<i>clean_syntax</i>
--------------	---------------------

---

### 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

---

get_groups	<i>get_groups</i>
------------	-------------------

---

### Description

returns a list of groups for a multi group model

### Usage

```
get_groups(multi_group_model)
```

### Arguments

multi\_group\_model  
multi group model created with mxsem\_group\_by

### Value

list with data for each group

### 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 + paragraf + 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)
```

---

```
get_individual_algebra_results
      get_individual_algebra_results
```

---

**Description**

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

**Usage**

```
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"]] $algebra\_result$ )
```

---

 mxsem

*mxsem*


---

## 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 <b>lavaan</b> 's syntax
data	raw data used to fit the model. Alternatively, an object created with <code>OpenMx::mxData</code> can be used (e.g., <code>OpenMx::mxData(observed = cov(OpenMx::Bollen), means = colMeans(OpenMx::Bollen), numObs = nrow(OpenMx::Bollen), type = "cov")</code> ).
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, <b>mxsem</b> 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?

<code>add_exogenous_manifest_covariances</code>	should covariances between exogenous manifest variables be added automatically?
<code>lbound_variances</code>	should the lower bound for variances be set to 0.000001?
<code>directed</code>	symbol used to indicate directed effects (regressions and loadings)
<code>undirected</code>	symbol used to indicate undirected effects (variances and covariances)
<code>return_parameter_table</code>	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., `!diff; a := b - exp(diff)`).
- **umx** (Bates et al., 2019) provides the `umxRAM` and `umxLav2RAM` functions that can parse single **lavaan**-style statements (e.g., `eta =~ y1 + y2 + y3`) 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 y1-y4:

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

Regressions are specified with ~:

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

Add covariances with ~~

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

Intercepts are specified with ~1

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

```
eta ~ 1
```

### Parameter labels and constraints:

Add labels to parameters as follows:

```
xi =~ l1*x1 + l2*x2 + l3*x3
eta =~ l4*y1 + l5*y2 + l6*y3
# predict eta with xi:
eta ~ b*xi
```

Fix parameters by using numeric values instead of labels:

```
xi =~ 1*x1 + 12*x2 + 13*x3
eta =~ 1*y1 + 15*y2 + 16*y3
# predict eta with xi:
eta ~ b*xi
```

### Bounds:

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

```

xi =~ 1*x1 + 12*x2 + 13*x3
eta =~ 1*y1 + 15*y2 + 16*y3
# predict eta with xi:
eta ~ b*xi
# residual variance for x1
x1 ~~ v*x1
# 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 eta1 is the first observation and eta2 the second. We want to define the loadings of eta2 on its observations as  $l_1 + \text{delta}_{11}$ . If delta\_11 is zero, we have measurement invariance.

```

eta1 =~ 11*y1 + 12*y2 + 13*y3
eta2 =~ 14*y4 + 15*y5 + 16*y6
# define new delta-parameter
!delta_1; !delta_2; !delta_3
# redefine l4-l6
l4 := 11 + delta_1
l5 := 12 + delta_2
l6 := 13 + delta_3

```

Alternatively, implicit transformations can be used as follows:

```

eta1 =~ 11*y1 + 12*y2 + 13*y3
eta2 =~ {11 + delta_1} * y4 + {12 + delta_2} * y5 + {13 + delta_3} * y6

```

Specific labels for the transformation results can also be provided:

```

eta1 =~ 11*y1 + 12*y2 + 13*y3
eta2 =~ {l4 := 11 + delta_1} * y4 + {l5 := 12 + delta_2} * y5 + {l6 := 13 + delta_3} * y6

```

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 =~ 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

```

### 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:**

- Bates, T. C., Maes, H., & Neale, M. C. (2019). umx: Twin and Path-Based Structural Equation Modeling in R. *Twin Research and Human Genetics*, 22(1), 27–41. <https://doi.org/10.1017/thg.2019.2>
- Bates, T. C., Prindle, J. J. (2014). ezMx. <https://github.com/OpenMx/ezMx>
- Boker, S. M., Neale, M., Maes, H., Wilde, M., Spiegel, M., Brick, T., Spies, J., Estabrook, R., Kenny, S., Bates, T., Mehta, P., & Fox, J. (2011). OpenMx: An Open Source Extended Structural Equation Modeling Framework. *Psychometrika*, 76(2), 306–317. <https://doi.org/10.1007/s11336-010-9200-6>
- Cheung, M. W.-L. (2015). metaSEM: An R package for meta-analysis using structural equation modeling. *Frontiers in Psychology*, 5. <https://doi.org/10.3389/fpsyg.2014.01521>
- Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, 48(2), 1–36. <https://doi.org/10.18637/jss.v048.i02>
- van Lissa, C. J. (2023). tidySEM: Tidy Structural Equation Modeling. R package version 0.2.4, <https://cjvanlissa.github.io/tidySEM/>.

**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

*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_grep = 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 + paragraf + 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)
```

---

parameters

*parameters*

---

**Description**

Returns the parameter estimates of an `mxModel`. Wrapper for `omxGetParameters`

**Usage**

```
parameters(mxMod)
```

**Arguments**

`mxMod`                      `mxModel` object

**Value**

vector with parameter estimates

---

parameter\_table\_rcpp    *parameter\_table\_rcpp*

---

**Description**

creates a parameter table from a lavaan like syntax

**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

**Value**

nothing

---

```
set_starting_values    set_starting_values
```

---

**Description**

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

**Usage**

```
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**

```

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
      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**

```
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**

```

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;
  "

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

```

---

```

simulate_moderated_nonlinear_factor_analysis
      simulate_moderated_nonlinear_factor_analysis

```

---

**Description**

simulate data for a moderated nonlinear factor analysis.

**Usage**

```
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**

```

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

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

```

```

eta ~ a*x1

# 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
      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 + paragraf + 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*

---

**Description**

this function returns the unicode for directed arrows

**Usage**

```
unicode_directed()
```

**Value**

returns unicode for directed arrows

---

*unicode\_undirected*      *unicode\_undirected*

---

**Description**

this function returns the unicode for undirected arrows

**Usage**

```
unicode_undirected()
```

**Value**

returns unicode for undirected arrows

# Index

`clean_syntax`, [2](#)

`get_groups`, [3](#)

`get_individual_algebra_results`, [4](#)

`mxsem`, [5](#)

`mxsem_group_by`, [10](#)

`parameter_table_rcpp`, [12](#)

`parameters`, [11](#)

`print.multi_group_parameters`, [13](#)

`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](#)