lavaan: an R package for structural equation modeling and more Version 0.4-9 (BETA)

Yves Rosseel
Department of Data Analysis
Ghent University (Belgium)

June 14, 2011

Abstract

The lavaan package is developed to provide useRs, researchers and teachers a free, open-source, but commercial-quality package for latent variable analysis. The long-term goal of lavaan is to implement all the state-of-the-art capabilities that are currently available in commercial packages, including support for various data types, discrete latent variables (aka mixture models) and multilevel datasets. Currently, the lavaan package provides support for confirmatory factor analysis, structural equation modeling, and latent growth curve models. In this document, we illustrate the use of lavaan by providing several examples. If you are new to lavaan, this is the first document to read.

Contents

1 Before you start

Before you start, please read these points carefully:

- First of all, you must have a fairly recent ($\geq$ 2.10.1) version of `R` installed. You can download the latest version of `R` from this page: `http://cran.r-project.org/`.
- The lavaan package is not finished yet. But it is already very useful for most users, or so we hope. There are a number of known minor issues (listed on the website), and some features are simply not implemented yet. Some important features that are currently *not* available in lavaan are:
  - support for categorical/censored variables
  - support for discrete latent variables (mixture models)
  - support for hierarchical/multilevel datasets

  We hope to add these features in the next year or so.
- We do not expect you to be an expert in `R`. In fact, the lavaan package is designed to be used by users that would normally never use `R`. Nevertheless, it may help to familiarize yourself a bit with `R`, just to be comfortable with it. Perhaps the most important skill that you may need to learn is how to import your own datasets (perhaps in an SPSS format) into `R`. There are many tutorials on the web to teach you just that. Once you have your data in `R`, you can start specifying your `lavaan` model. We have tried very hard to make it as easy as possible for users to fit their models. Of course, if you have suggestions on how we can improve things, please let us know.
- This document is written for first-time users (and beta-testers) of the lavaan package. It is not a reference manual, nor does it contain technical material on how things are done in the lavaan package. These documents are currently under preparation.
- The lavaan package is free open-source software. This means (among other things) that there is no warranty whatsoever.
- The numerical results of the lavaan package are typically very close, if not identical, to the results of the commercial package Mplus. If you wish to compare the results with those obtained by other SEM packages, you can use the optional argument `mimic="EQS"` when calling the `cfa`, `sem` or `growth` functions (see section 8.2).

2 Installation of the lavaan package

Since may 2010, the lavaan package is available on CRAN. Therefore, to install lavaan, simply start up R, and type:

```
> install.packages("lavaan")
```

You can check if the installation was succesful by typing

```
> library(lavaan)

This is lavaan 0.4-9
lavaan is BETA software! Please report any bugs.
```

When the package is loaded, a startup message will be displayed showing the version number, and a reminder that this is beta software.

3 The model syntax

At the heart of the lavaan package is the 'model syntax'. The model syntax is a description of the model to be estimated. In this section, we briefly explain the elements of the lavaan model syntax. More details are given in the examples that follow.
In the R environment, a regression formula has the following form:

```
y ~ x1 + x2 + x3 + x4
```

In lavaan, a typical model is simply a set (or system) of regression formulas, where some variables (starting with an 'f' below) may be latent. For example:

```
 y ~ f1 + f2 + x1 + x2
f1 ~ f2 + f3
f2 ~ f3 + x1 + x2
```

If we have latent variables in any of the regression formulas, we must 'define' them by listing their manifest indicators. We do this by using the special operator `"=~"`, which can be read as *is manifested by.* For example, to define the three latent variabels `f1`, `f2` and `f3`, we can use something like:

```
f1 =~ y1 + y2 + y3
f2 =~ y4 + y5 + y6
f3 =~ y7 + y8 + y9 + y10
```

Furthermore, variances and covariances are specified using a 'double tilde' operator, for example:

```
y1 ~~ y1
y1 ~~ y2
f1 ~~ f2
```

And finally, intercepts for observed and latent variables are simple regression formulas with only an intercept (explicitly denoted by the number '1') as the only predictor:

```
y1 ~ 1
f1 ~ 1
```

Using these four *formula types*, a large variety of latent variable models can be described. But new formula types may be added in the future. The current set of formula types is summarized in the table below.

formula type	operator	mnemonic
latent variable definition	=~	is measured by
regression	~	is regressed on
(residual) (co)variance	~~	is correlated with
intercept	~ 1	intercept

3.1 Entering the model syntax as a string literal

If the model syntax is fairly short, you can specify it interactively at the R prompt by enclosing the formulas with single quotes. For example:

```
> myModel <- ' # regressions
                 y1 + y2 ~ f1 + f2 + x1 + x2
                      f1 ~ f2 + f3
                      f2 ~ f3 + x1 + x2

               # latent variable definitions
                 f1 =~ y1 + y2 + y3
                 f2 =~ y4 + y5 + y6
                 f3 =~ y7 + y8 +
                       y9 + y10

               # variances and covariances
                 y1 ~~ y1
                 y1 ~~ y2
                 f1 ~~ f2

               # intercepts
                 y1 ~ 1
                 f1 ~ 1
             '
```

Of course, instead of typing this interactively at the R prompt, you may prefer to type the whole model first in an external text editor, and when you are done, you can copy/paste it to the R console. This piece of code will produce a model syntax object, called `myModel` that can be used later when calling a function that actually estimates this model given a dataset. Note that formulas can be split over multiple lines, and you can use comments (starting with the `#` character) and blank lines within the single quotes to improve readability of the model syntax.

3.2 Reading the model syntax from an external file

If your model syntax is rather long, or you need to reuse the model syntax over and over again, you may prefer to type it in a separate text file called, say, `myModel.lav`. This text file should be in a human readable format (not a Word document). Within R, you can then read the model syntax from the file as follows:

```
> myModel <- readLines("/mydirectory/myModel.lav")
```

The argument of `readLines` is the full path to the file containing the model syntax. Again, the model syntax object `myModel` can be used later to fit this model given a dataset.

4 Fitting latent variable models: two examples

4.1 A first example: confirmatory factor analysis (CFA)

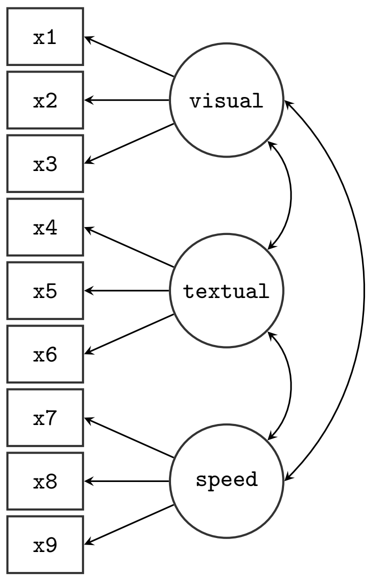
We start with a simple example of confirmatory factor analysis, using the `cfa` function, which is a user-friendly function for fitting CFA models. The lavaan package contains a built-in dataset called `HolzingerSwineford1939`. See the help page for this dataset by typing

```
> ?HolzingerSwineford1939
```

at the `R` prompt. This is a 'classic' dataset that is used in many papers and books on Structural Equation Modeling (SEM), including some manuals of commercial SEM software packages. The data consists of mental ability test scores of seventh- and eighth-grade children from two different schools (Pasteur and Grant-White). In our version of the dataset, only 9 out of the original 26 tests are included. A CFA model that is often proposed for these 9 variables consists of three latent variables (or factors), each with three indicators:

- a *visual* factor measured by 3 variables: `x1`, `x2` and `x3`
- a *textual* factor measured by 3 variables: `x4`, `x5` and `x6`
- a *speed* factor measured by 3 variables: `x7`, `x8` and `x9`

The left panel of the figure below contains a simplified graphical representation of the three-factor model. The right panel contains the corresponding lavaan syntax for specifying this model.



lavaan syntax

```
visual  =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
```

In this example, the model syntax only contains three 'latent variable definitions'. Each formula has the following format:

```
latent variable =~ indicator1 + indicator2 + indicator3
```

We call these expressions *latent variable definitions* because they define how the latent variables are 'manifested by' a set of observed (or manifest) variables, often called 'indicators'. Note that the special `"=~"` operator in the middle consists of a sign (`"="`) character and a tilde (`"~"`) character next to each other. The reason why this model syntax is so short, is that behind the scenes, the `cfa` function will take care of several things. First, by default, the factor loading of the first indicator of a latent variable is fixed to 1, thereby fixing the scale of the latent variable. Second, residual variances are added automatically. And third, all exogenous latent variables are correlated by default. This way, the model syntax can be kept concise. On the other hand, the user remains in control, since all this 'default' behavior can be overriden and/or switched off.
We can enter the model syntax using the single quotes:

```
> HS.model <- '
+   visual  =~ x1 + x2 + x3
+   textual =~ x4 + x5 + x6
+   speed   =~ x7 + x8 + x9
+ '
```

We can now fit the model as follows:

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939)
```

The lavaan function `cfa` is a dedicated function for fitting confirmatory factor analysis models. The first argument is the user-specified model. The second argument is the dataset that contains the observed variables. Once the model has been fitted, the `summary` method provides a nice summary of the fitted model:

```
> summary(fit, fit.measures = TRUE)

Lavaan (0.4-9) converged normally after 36 iterations

  Number of observations                           301

  Estimator                                         ML
  Minimum Function Chi-square                   85.306
  Degrees of freedom                                24
  P-value                                        0.000

Chi-square test baseline model:
  Minimum Function Chi-square                  918.852
  Degrees of freedom                                36
  P-value                                        0.000

Full model versus baseline model:

  Comparative Fit Index (CFI)                    0.931
  Tucker-Lewis Index (TLI)                       0.896

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)              -3737.745
  Loglikelihood unrestricted model (H1)      -3695.092

  Number of free parameters                         21
  Akaike (AIC)                                7517.490
  Bayesian (BIC)                              7595.339
  Sample-size adjusted Bayesian (BIC)         7528.739

Root Mean Square Error of Approximation:

  RMSEA                                          0.092
  90 Percent Confidence Interval          0.071  0.114
  P-value RMSEA <= 0.05                          0.001

Standardized Root Mean Square Residual:

  SRMR                                           0.065

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.554    0.100    5.554    0.000
    x3                0.729    0.109    6.685    0.000
  textual =~
    x4                1.000
    x5                1.113    0.065   17.014    0.000
    x6                0.926    0.055   16.703    0.000
  speed =~
    x7                1.000
    x8                1.180    0.165    7.152    0.000
    x9                1.082    0.151    7.155    0.000

Covariances:
  visual ~~
    textual           0.408    0.074    5.552    0.000
    speed             0.262    0.056    4.660    0.000
  textual ~~
    speed             0.173    0.049    3.518    0.000

Variances:
    x1                0.549    0.114    4.833    0.000
    x2                1.134    0.102   11.146    0.000
    x3                0.844    0.091    9.317    0.000
    x4                0.371    0.048    7.779    0.000
    x5                0.446    0.058    7.642    0.000
    x6                0.356    0.043    8.277    0.000
    x7                0.799    0.081    9.823    0.000
    x8                0.488    0.074    6.573    0.000
    x9                0.566    0.071    8.003    0.000
    visual            0.809    0.145    5.564    0.000
    textual           0.979    0.112    8.737    0.000
    speed             0.384    0.086    4.451    0.000
```

The output should look familiar to users of other SEM software. If you find it confusing or esthetically unpleasing, again, please let us know, and we will try to improve it. To wrap up this first example, we summarize the code that was needed to fit this three-factor model:

R code

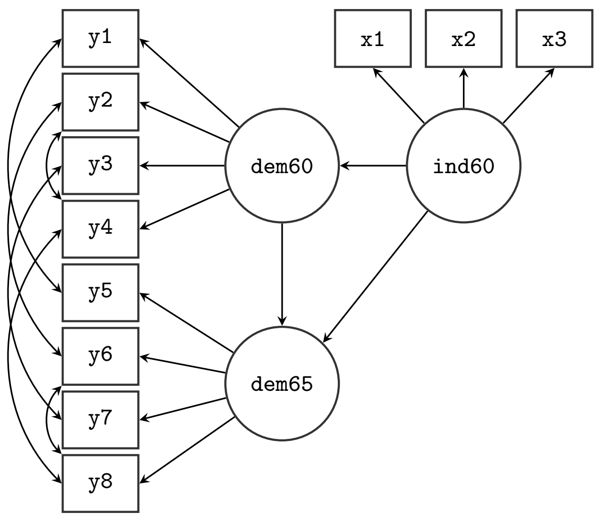
```
# load the lavaan package (only needed once per session)
library(lavaan)

# specify the model
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

# fit the model
fit <- cfa(HS.model, data=HolzingerSwineford1939)

# display summary output
summary(fit, fit.measures=TRUE)
```

Simply copying this code and pasting it in `R` should work. The syntax illustrates the typical workflow in the lavaan package:

1. specify your model using the lavaan model syntax. In this example, only *latent variable definitions* have been used. In the following examples, other formula types will be used.
2. fit the model. This requires a dataset containing the observed variables (or alternatively the sample covariance matrix and the number of observations; see section 8.1). In this example, we have used the `cfa` function. Other funcions in the lavaan package are `sem` and `growth` for fitting full structural equation models and growth curve models respectively. All three functions are so-called user-friendly functions, in the sense that they take care of many details automatically, so we can keep the model syntax simple and concise. If you wish to fit non-standard models or if you don't like the idea that things are done for you automatically, you can use the lower-level function `lavaan`, where you have full control.
3. extract information from the fitted model. This can be a long verbose summary, or it can be a single number only (say, the RMSEA value). In the spirit of `R`, you only get what you asked for. We do not print out unnecessary information that you would ignore anyway.

4.2 A second example: a structural equation model

In our second example, we will use the built-in `PoliticalDemocracy` dataset. This is a dataset that has been used by Bollen in his 1989 book on structural equation modeling (and elsewhere). To learn more about the dataset, see the help page and the references therein.
The left panel of the figure below contains a graphical representation of the model that we want to fit. The right panel contains the corresponding model syntax.



lavaan syntax

```
# latent variable definitions
  ind60 =~ x1 + x2 + x3
  dem60 =~ y1 + y2 + y3 + y4
  dem65 =~ y5 + y6 + y7 + y8

# regressions
  dem60 ~ ind60
  dem65 ~ ind60 + dem60

# residual covariances
  y1 ~~ y5
  y2 ~~ y4 + y6
  y3 ~~ y7
  y4 ~~ y8
  y6 ~~ y8
```

In this example, we use three different formula types: latent variabele definitions, regression formulas, and (co)variance formulas. The regression formulas are similar to ordinary formulas in R. The (co)variance formulas typically have the following form:

```
variable ~~ variable
```

The variables can be either observed or latent variables. If the two variable names are the same, the expression refers to the variance (or residual variance) of that variable. If the two variable names are different, the expression refers to the (residual) covariance among these two variables. The lavaan package automatically makes the distinction between variances and residual variances.
In our example, the expression `y1 ~~ y5` allows the residual variances of the two observed variables to be correlated. This is sometimes done if it is believed that the two variables have something in common that is not captured by the latent variables. In this case, the two variables refer to identical scores, but measured in two different years (1960 and 1965 respectively). Note that the two expressions `y2 ~~ y4` and `y2 ~~ y6` can be combined into the expression `y2 ~~ y4 + y6`. This is just a shorthand notation.
We enter the model syntax as follows:

```
> model <- '
+   # measurement model
+     ind60 =~ x1 + x2 + x3
+     dem60 =~ y1 + y2 + y3 + y4
+     dem65 =~ y5 + y6 + y7 + y8
+
+   # regressions
+     dem60 ~ ind60
+     dem65 ~ ind60 + dem60
+
+   # residual correlations
+     y1 ~~ y5
+     y2 ~~ y4 + y6
+     y3 ~~ y7
+     y4 ~~ y8
+     y6 ~~ y8
+ '
```

To fit the model and see the results we can type:

```
> fit <- sem(model, data = PoliticalDemocracy)
> summary(fit, standardized = TRUE)

Lavaan (0.4-9) converged normally after 68 iterations

  Number of observations                            75

  Estimator                                         ML
  Minimum Function Chi-square                   38.125
  Degrees of freedom                                35
  P-value                                        0.329

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)   Std.lv  Std.all
Latent variables:
  ind60 =~
    x1                1.000                               0.670    0.920
    x2                2.180    0.139   15.742    0.000    1.460    0.973
    x3                1.819    0.152   11.967    0.000    1.218    0.872
  dem60 =~
    y1                1.000                               2.223    0.850
    y2                1.257    0.182    6.889    0.000    2.794    0.717
    y3                1.058    0.151    6.987    0.000    2.351    0.722
    y4                1.265    0.145    8.722    0.000    2.812    0.846
  dem65 =~
    y5                1.000                               2.103    0.808
    y6                1.186    0.169    7.024    0.000    2.493    0.746
    y7                1.280    0.160    8.002    0.000    2.691    0.824
    y8                1.266    0.158    8.007    0.000    2.662    0.828
```

```
Regressions:
  dem60 ~
    ind60             1.483    0.399    3.715    0.000    0.447    0.447
  dem65 ~
    ind60             0.572    0.221    2.586    0.010    0.182    0.182
    dem60             0.837    0.098    8.514    0.000    0.885    0.885

Covariances:
  y1 ~~
    y5                0.624    0.358    1.741    0.082    0.624    0.296
  y2 ~~
    y4                1.313    0.702    1.871    0.061    1.313    0.273
    y6                2.153    0.734    2.934    0.003    2.153    0.356
  y3 ~~
    y7                0.795    0.608    1.308    0.191    0.795    0.191
  y4 ~~
    y8                0.348    0.442    0.787    0.431    0.348    0.109
  y6 ~~
    y8                1.356    0.568    2.386    0.017    1.356    0.338

Variances:
    x1                0.082    0.019    4.184    0.000    0.082    0.154
    x2                0.120    0.070    1.718    0.086    0.120    0.053
    x3                0.467    0.090    5.177    0.000    0.467    0.239
    y1                1.891    0.444    4.256    0.000    1.891    0.277
    y2                7.373    1.374    5.366    0.000    7.373    0.486
    y3                5.067    0.952    5.324    0.000    5.067    0.478
    y4                3.148    0.739    4.261    0.000    3.148    0.285
    y5                2.351    0.480    4.895    0.000    2.351    0.347
    y6                4.954    0.914    5.419    0.000    4.954    0.443
    y7                3.431    0.713    4.814    0.000    3.431    0.322
    y8                3.254    0.695    4.685    0.000    3.254    0.315
    ind60             0.448    0.087    5.173    0.000    1.000    1.000
    dem60             3.956    0.921    4.295    0.000    0.800    0.800
    dem65             0.172    0.215    0.803    0.422    0.039    0.039
```

The function `sem` is very similar to the `cfa` function. In fact, the two functions are currently almost identical, but this may change in the future. In the `summary` method, we omitted the `fit.measures=TRUE` argument. Therefore, you only get the basic chi-square statistic. The argument `standardized=TRUE` augments the output with standardized parameter values. Two extra columns of standardized parameter values are printed. In the first column (labeled `Std.lv`), only the latent variables are standardized. In the second column (labeled `Std.all`), both latent and observed variables are standardized. The latter is often called the 'completely standardized solution'.
The complete code to specify and fit this model is printed again below:

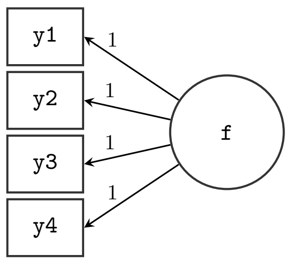
R code

```r
library(lavaan) # only needed once per session
model <- '
  # measurement model
    ind60 =~ x1 + x2 + x3
    dem60 =~ y1 + y2 + y3 + y4
    dem65 =~ y5 + y6 + y7 + y8

  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60

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

fit <- sem(model, data=PoliticalDemocracy)
summary(fit, standardized=TRUE)
```

5 Fixing parameters, starting values and equality constraints

5.1 Fixing parameters

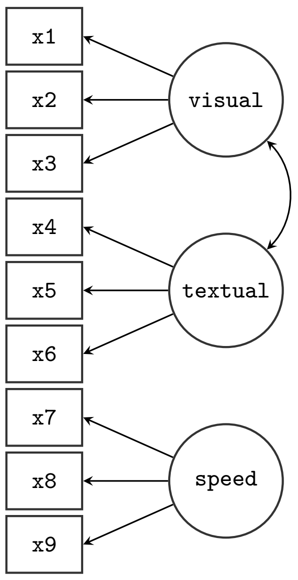
Consider a simple one-factor model with 4 indicators. By default, lavaan will always fix the factor loading of the first indicator to 1. The other three factor loadings are free, and their values are estimated by the model. But suppose that you have good reasons the fix all the factor loadings to 1. The syntax below illustrates how this can be done:



lavaan syntax

```
f =~ y1 + 1*y2 + 1*y3 + 1*y4
```

In general, to fix a parameter in a lavaan formula, you need to pre-multiply the corresponding variable in the formula by a numerical value. This is called the pre-multiplication mechanism and will be used for many purposes. As another example, consider again the three-factor Holzinger and Swineford CFA model. Recall that, by default, all exogenous latent variables in a CFA model are correlated. But if you wish to fix the correlation (or covariance) between a pair of latent variables to zero, you need to explicity add a covariance-formula for this pair, and fix the parameter to zero. In the figure below, we allow the covariance between the latent variables `visual` and `textual` to be free, but the two other covariances are fixed to zero. In addition, we fix the variance of the `speed` factor to unity. Therefore, there is no need anymore to set the factor loading of its first indicator (`x7`) equal to one. To force this factor loading to be free, we pre-multiply it with `NA`, as a hint to lavaan that the value of this parameter is still unknown.



lavaan syntax

```
# three-factor model
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ NA*x7 + x8 + x9

# orthogonal factors
   visual ~~ 0*speed
  textual ~~ 0*speed

# fix variance of speed factor
    speed ~~ 1*speed
```

If you need to constrain all covariances of the latent variables in a CFA model to be orthogonal, there is a shortcut. You can omit the covariance formulas in the model syntax and simply add an `orthogonal=TRUE` argument to the `cfa` function call:

```
> HS.model <- ' visual  =~ x1 + x2 + x3
+               textual =~ x4 + x5 + x6
+               speed   =~ x7 + x8 + x9 '
> fit.HS.ortho <- cfa(HS.model, data=HolzingerSwineford1939, orthogonal=TRUE)
```

Similarly, if you want to fix the variances of *all* the latent variables in a CFA model to unity, there is again a shortcut. Simply add a `std.lv=TRUE` argument to the `cfa` function call:

```
> HS.model <- ' visual  =~ x1 + x2 + x3
+               textual =~ x4 + x5 + x6
+               speed   =~ x7 + x8 + x9 '
> fit <- cfa(HS.model, data=HolzingerSwineford1939, std.lv=TRUE)
```

If the `std.lv=TRUE` argument is used, the factor loadings of the first indicator of each latent variable will no longer be fixed to 1.

5.2 Starting values

The lavaan package automatically generates starting values for all free parameters. Normally, this works fine. But if you must provide your own starting values, you are free to do so. The way it works is based on the pre-multiplication mechanism that we discussed before. But the numeric constant is now the argument of a special function `start()`. An example will make this clear:

lavaan syntax

```
visual  =~ x1 + start(0.8)*x2 + start(1.2)*x3
textual =~ x4 + start(0.5)*x5 + start(1.0)*x6
speed   =~ x7 + start(0.7)*x8 + start(1.8)*x9
```

The factor loadings of the first indicators (`x1`, `x4` and `x7`) are fixed, so no starting values are needed. But for all other factor loadings, starting values are provided in this example.

5.3 Parameter names

A nice property of the lavaan package is that all free parameters are automatically named according to a simple set of rules. This is convenient, for example, if equality constraints are needed (see the next subsection). To see how the naming mechanism works, we will use the model that we used for the Politcal Democracy data.

```
> model <- '
+   # latent variable definitions
+     ind60 =~ x1 + x2 + x3
+     dem60 =~ y1 + y2 + y3 + y4
+     dem65 =~ y5 + y6 + y7 + y8
+   # regressions
+     dem60 ~ ind60
+     dem65 ~ ind60 + dem60
+   # residual (co)variances
+     y1 ~~ y5
+     y2 ~~ y4 + y6
+     y3 ~~ y7
+     y4 ~~ y8
+     y6 ~~ y8
+ '
> fit <- sem(model, data=PoliticalDemocracy)
> coef(fit)

   ind60=~x2    ind60=~x3    dem60=~y2    dem60=~y3    dem60=~y4    dem65=~y6
       2.180        1.819        1.257        1.058        1.265        1.186
   dem65=~y7    dem65=~y8  dem60~ind60  dem65~ind60  dem65~dem60       y1~~y5
       1.280        1.266        1.483        0.572        0.837        0.624
      y2~~y4       y2~~y6       y3~~y7       y4~~y8       y6~~y8       x1~~x1
       1.313        2.153        0.795        0.348        1.356        0.082
      x2~~x2       x3~~x3       y1~~y1       y2~~y2       y3~~y3       y4~~y4
       0.120        0.467        1.891        7.373        5.067        3.148
      y5~~y5       y6~~y6       y7~~y7       y8~~y8 ind60~~ind60 dem60~~dem60
       2.351        4.954        3.431        3.254        0.448        3.956
dem65~~dem65
       0.172
```

The `coef` function extracts the estimated values of the free parameters in the model, together with their names. Each name consists of three parts and reflects the part of the formula where the parameter was involved. The first part is the variable name that appears on the left-hand side of the formula. The middle part is the operator type of the formula, and the third part is the variable in the right-hand side of the formula that corresponds with the parameter.
If you want, you can provide custom parameter names or *labels* by using the `label()` modifier. An example will make this clear:

```
> model <- '
+   # latent variable definitions
+     ind60 =~ x1 + x2 + label("myLabel")*x3
+     dem60 =~ y1 + y2 + y3 + y4
+     dem65 =~ y5 + y6 + y7 + y8
+   # regressions
+     dem60 ~ ind60
+     dem65 ~ ind60 + dem60
+   # residual (co)variances
+     y1 ~~ y5
+     y2 ~~ y4 + y6
+     y3 ~~ y7
+     y4 ~~ y8
+     y6 ~~ y8
+ '
```

The default name of the parameter associated with the factor loading of the `x3` indicator is by default `"ind60=~x3"`. But the `label()` modifier will change it to the custom name `"myLabel"`. If we need to constrain some other parameters to be equal to this parameter, we can refer to it by its new label.
Since version 0.4-8, there is an alternative way to 'label' a parameter. Instead of using the `label()` modifier, you can simply multiply the variable with its label (without the quotes), for example:

```
> model <- '
+   # latent variable definitions
+     ind60 =~ x1 + x2 + myLabel*x3
+   ...
+ '
```

Internally, the '`myLabel`' modifier is simply replaced by '`label("myLabel")`', so both methods can be used interchangeably. It is important, however, that labels start with a letter (a-zA-Z), and certainly not with a digit. For example '13bis' is not a valid label, and will confuse the lavaan syntax parser.

5.4 Simple equality constraints

In some applications, it is useful to impose equality constraints on one or more otherwise free parameters. Consider again the three-factor H&S CFA model. Suppose a user has a priori reasons to believe that the factor loadings of the `x2` and `x3` indicators are equal to each other. Instead of estimating two free parameters, lavaan should only estimate a single free parameter, and use that value for both factor loadings. Another way of thinking about this is that the factor loading for the `x2` variable will be freely estimated, but that the factor loading of the `x3` variable will be set equal to the factor loading of the `x2` variable. We call the factor loading for `x2` the 'target parameter', and the factor loading for `x3` the 'constrained' parameter. In the lavaan model syntax, we again need to use the pre-multiplication mechanism using a special function called `equal()`. The single argument of this function is the name of the target parameter. This is illustrated in the following syntax:

```
lavaan syntax

visual  =~ x1 + x2 + equal("visual=~x2")*x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
```

The parameter corresponding to the factor loading of the `x2` variable is (automatically) called `"visual=~x2"`. By using the `equal()` modifier for `x3`, the corresponding parameter value will be set equal to the factor loading of `x2`. This mechanism can be used for any free parameter in a lavaan model.

Again from version 0.4-8 onwards, a second method can be used. If we explicitly label the parameters with the same name, the parameters will be constrained to be equal. For example, giving the same label (say, `v2`) to both the factor loading of `x2` and `x3` will constrain the two parameters to be equal:

```
lavaan syntax

visual  =~ x1 + v2*x2 + v2*x3
textual =~ x4 + x5 + x6
speed   =~ x7 + x8 + x9
```

Especially if many constraints need to be specified, this may be a much more convenient way to specify the constraints than using the `equal()` modifier.

5.5 Nonlinear equality and inequality constraints

In version 0.4-8, initial (and experimental) support has been added for general nonlinear equality and inequality constraints. Consider for example the following regression:

```
lavaan syntax

y ~ b1*x1 + b2*x2 + b3*x3
```

where we have explicitly labeled the regression coefficients as b1, b2 and b3. We create a toy dataset containing these four variables and fit the regression model:

```
> set.seed(1234)
> Data <- data.frame(y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100),
+                    x3 = rnorm(100))
> model <- ' y ~ b1*x1 + b2*x2 + b3*x3 '
> fit <- sem(model, data=Data)
> coef(fit)

    b1     b2     b3   y~~y
-0.052  0.084  0.139  0.970
```

Suppose that we need to impose the following two (nonlinear) constraints: $b_1 = (b_2 + b_3)^2$ and $b_1 \geq \exp(b_2 + b_3)$. The first constraint is an equality constraint. The second is an inequality constraint. To specify these constraints, you can use the following syntax:

```
lavaan syntax

model.constr <- ' # model with labeled parameters
                    y ~ b1*x1 + b2*x2 + b3*x3
                  # constraints
                    b1 == (b2 + b3)^2
                    b1 > exp(b2 + b3) '
```

To see the effect of the constraints, we refit the model:

```
> model.constr <- ' # model with labeled parameters
+                     y ~ b1*x1 + b2*x2 + b3*x3
+                   # constraints
+                     b1 == (b2 + b3)^2
+                     b1 > exp(b2 + b3) '
> fit <- sem(model.constr, data=Data)
> coef(fit)

    b1     b2     b3   y~~y
 0.495 -0.405 -0.299  1.610
```

The reader can verify that the constraints are indeed respected. The equality constraint holds exactly. The inequality constraint has resulted in an equality between the left-hand side (b_1) and the right-hand side ($\exp(b_2 + b_3)$). Note: robust standard errors are not (yet) available in this situation.

6 Meanstructures and multiple groups

6.1 Bringing in the means

By and large, structural equation models are used to model the covariance matrix of the observed variables in a dataset. But in some applications, it is useful to bring in the means of the observed variables too. One way to do this is to explicitly refer to intercepts in the lavaan syntax. This can be done by including 'intercept formulas' in the model syntax. An intercept formula has the following form:

```
variable ~ 1
```

The left part of the expression contains the name of the observed or latent variable. The right part contains the number 1, representing the intercept. For example, in the three-factor H&S CFA model, we can add the intercepts of the observed variables as follows:

```
lavaan syntax

# three-factor model
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9

# intercepts
  x1 ~ 1
  x2 ~ 1
  x3 ~ 1
  x4 ~ 1
  x5 ~ 1
  x6 ~ 1
  x7 ~ 1
  x8 ~ 1
  x9 ~ 1
```

However, it is more convenient to omit the intercept formulas in the model syntax (unless you want to fix their values), and to add the `meanstructure = TRUE` argument in the `cfa` and `sem` function calls. For example, we can refit the three-factor H&S CFA model as follows:

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939, meanstructure = TRUE)
> summary(fit)

Lavaan (0.4-9) converged normally after 36 iterations

  Number of observations                           301

  Estimator                                         ML
  Minimum Function Chi-square                   85.306
  Degrees of freedom                                24
  P-value                                        0.000

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.554    0.100    5.554    0.000
    x3                0.729    0.109    6.685    0.000
  textual =~
    x4                1.000
    x5                1.113    0.065   17.014    0.000
    x6                0.926    0.055   16.703    0.000
  speed =~
    x7                1.000
    x8                1.180    0.165    7.152    0.000
    x9                1.082    0.151    7.155    0.000
```

```
Covariances:
  visual ~~
    textual           0.408    0.074    5.552    0.000
    speed             0.262    0.056    4.660    0.000
  textual ~~
    speed             0.173    0.049    3.518    0.000

Intercepts:
    x1                4.936    0.067   73.473    0.000
    x2                6.088    0.068   89.855    0.000
    x3                2.250    0.065   34.579    0.000
    x4                3.061    0.067   45.694    0.000
    x5                4.341    0.074   58.452    0.000
    x6                2.186    0.063   34.667    0.000
    x7                4.186    0.063   66.766    0.000
    x8                5.527    0.058   94.854    0.000
    x9                5.374    0.058   92.546    0.000
    visual            0.000
    textual           0.000
    speed             0.000

Variances:
    x1                0.549    0.114    4.833    0.000
    x2                1.134    0.102   11.146    0.000
    x3                0.844    0.091    9.317    0.000
    x4                0.371    0.048    7.779    0.000
    x5                0.446    0.058    7.642    0.000
    x6                0.356    0.043    8.277    0.000
    x7                0.799    0.081    9.823    0.000
    x8                0.488    0.074    6.573    0.000
    x9                0.566    0.071    8.003    0.000
    visual            0.809    0.145    5.564    0.000
    textual           0.979    0.112    8.737    0.000
    speed             0.384    0.086    4.451    0.000
```

As you can see in the output, the model includes intercept parameters for both the observed and latent variables. By default, the `cfa` and `sem` functions fix the latent variable intercepts (which in this case correspond to the latent *means*) to zero. Otherwise, the model would not be estimable. Note that the chi-square statistic and the number of degrees of freedom is the same as in the original (non-meanstructure) model. The reason is that we brought in some new data (a mean value for each of the 9 observed variables), but we also added 9 additional parameters to the model (an intercept for each of the 9 observed variables). The end result is an identical fit. In practice, the only reason why a user would add intercept-formulas in the model syntax, is because some constraints must be specified on them. For example, suppose that we wish to fix the intercepts of the variables `x1`, `x2`, `x3` and `x4` to, say, 0.5. We would write the model syntax as follows:

lavaan syntax

```
# three-factor model
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9

# intercepts with fixed values
  x1 + x2 + x3 + x4 ~ 0.5*1
```

where we have used the left-hand side of the formula to 'repeat' the right-hand side for each element of the left-hand side.

6.2 Multiple groups

The lavaan package has full support for multiple groups. To request a multiple group analysis, you need to add the name of the group variable in your dataset to the `group` argument in the `cfa` and `sem` function calls. By default, the same model is fitted in all groups. In the following example, we fit the H&S CFA model for the two schools (Pasteur and Grant-White).

```
> HS.model <- ' visual  =~ x1 + x2 + x3
+               textual =~ x4 + x5 + x6
+                 speed   =~ x7 + x8 + x9 '
> fit <- cfa(HS.model, data=HolzingerSwineford1939, group="school")
> summary(fit)

Lavaan (0.4-9) converged normally after 56 iterations

  Number of observations per group
  Pasteur                                            156
  Grant-White                                        145

  Estimator                                           ML
  Minimum Function Chi-square                    115.851
  Degrees of freedom                                  48
  P-value                                          0.000

Chi-square for each group:

  Pasteur                                         64.309
  Grant-White                                     51.542

Parameter estimates:

  Information                                   Expected
  Standard Errors                               Standard

Group 1 [Pasteur]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.394    0.122    3.220    0.001
    x3                0.570    0.140    4.076    0.000
  textual =~
    x4                1.000
    x5                1.183    0.102   11.613    0.000
    x6                0.875    0.077   11.421    0.000
  speed =~
    x7                1.000
    x8                1.125    0.277    4.057    0.000
    x9                0.922    0.225    4.104    0.000

Covariances:
  visual ~~
    textual           0.479    0.106    4.531    0.000
    speed             0.185    0.077    2.397    0.017
  textual ~~
    speed             0.182    0.069    2.628    0.009

Variances:
    x1                0.298    0.232    1.286    0.198
    x2                1.334    0.158    8.464    0.000
    x3                0.989    0.136    7.271    0.000
    x4                0.425    0.069    6.138    0.000
    x5                0.456    0.086    5.292    0.000
    x6                0.290    0.050    5.780    0.000
    x7                0.820    0.125    6.580    0.000
    x8                0.510    0.116    4.406    0.000
    x9                0.680    0.104    6.516    0.000
    visual            1.097    0.276    3.967    0.000
    textual           0.894    0.150    5.963    0.000
    speed             0.350    0.126    2.778    0.005



Group 2 [Grant-White]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.736    0.155    4.760    0.000
    x3                0.925    0.166    5.583    0.000
  textual =~
    x4                1.000
    x5                0.990    0.087   11.418    0.000
    x6                0.963    0.085   11.377    0.000
  speed =~
    x7                1.000
    x8                1.226    0.187    6.569    0.000
    x9                1.058    0.165    6.429    0.000

Covariances:
  visual ~~
    textual           0.408    0.098    4.153    0.000
    speed             0.276    0.076    3.639    0.000
  textual ~~
    speed             0.222    0.073    3.022    0.003

Variances:
    x1                0.715    0.126    5.676    0.000
    x2                0.899    0.123    7.339    0.000
    x3                0.557    0.103    5.409    0.000
    x4                0.315    0.065    4.870    0.000
    x5                0.419    0.072    5.812    0.000
    x6                0.406    0.069    5.880    0.000
    x7                0.600    0.091    6.584    0.000
    x8                0.401    0.094    4.249    0.000
    x9                0.535    0.089    6.010    0.000
    visual            0.604    0.160    3.762    0.000
    textual           0.942    0.152    6.177    0.000
    speed             0.461    0.118    3.910    0.000
```

If you want to fix parameters, or provide starting values, you can use the same pre-multiplication techniques, but the single argument is now replaced by a vector of arguments, one for each group. If you use a single element instead of a vector, that element will be applied for all groups. For example:

lavaan syntax

```
HS.model <- ' visual  =~ x1 + 0.5*x2 + c(0.6, 0.8)*x3
              textual =~ x4 + start(c(1.2, 0.6))*x5 + a*x6
              speed   =~ x7 + equal(c("textual=~x5","textual=~x5.g2"))*x8
                            + equal("a")*x9 '
```

In the definition of the latent factor `visual`, we have fixed the factor loading of the `x3` indicator to the value '0.6' in the first group, and to the value '0.8' in the second group, while the factor loading of the `x2` indicator is fixed to the value '0.5' in both groups. In the definition of the `textual` factor, two different starting values are provided for the `x5` indicator; one for each group. In addition, we have labeled the factor loading of the `x6` indicator as 'a', so that we can easily refer to it later on. Finally, in the definition of the `speed` factor, we constrained the factor loadings of the `x8` indicator to be the same as the factor loadings of the `x5` indicator, in both groups. Carefully note that the second element in the vector argument of the `equal()` modifier (ie `textual=~x5.g2"`) refers to the second group (the `".g2"` is added automatically). For the `x9` indicator, we again use the `equal()` modifier, but now with a single argument (the label 'a'). Therefore, the corresponding parameter in both groups will be set equal to the value of the parameter 'a'. To verify the effects of these modifiers, we refit the model:

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939, group = "school")
> summary(fit)

Lavaan (0.4-9) converged normally after 35 iterations

  Number of observations per group
  Pasteur                                          156
  Grant-White                                      145

  Estimator                                         ML
  Minimum Function Chi-square                  121.339
  Degrees of freedom                                56
  P-value                                        0.000

Chi-square for each group:

  Pasteur                                       65.154
  Grant-White                                   56.185
```

```
Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

Group 1 [Pasteur]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.500
    x3                0.600
  textual =~
    x4                1.000
    x5                1.196    0.095   12.626    0.000
    x6                0.905    0.064   14.119    0.000
  speed =~
    x7                1.000
    x8                1.196    0.095   12.626    0.000
    x9                0.905    0.064   14.119    0.000

Covariances:
  visual ~~
    textual           0.455    0.101    4.524    0.000
    speed             0.173    0.068    2.531    0.011
  textual ~~
    speed             0.173    0.062    2.806    0.005

Variances:
    x1                0.388    0.129    3.005    0.003
    x2                1.304    0.155    8.432    0.000
    x3                0.965    0.120    8.016    0.000
    x4                0.435    0.068    6.381    0.000
    x5                0.460    0.085    5.386    0.000
    x6                0.281    0.050    5.664    0.000
    x7                0.828    0.111    7.459    0.000
    x8                0.482    0.092    5.251    0.000
    x9                0.695    0.093    7.499    0.000
    visual            1.000    0.172    5.802    0.000
    textual           0.867    0.138    6.283    0.000
    speed             0.335    0.070    4.772    0.000



Group 2 [Grant-White]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.500
    x3                0.800
  textual =~
    x4                1.000
    x5                1.045    0.076   13.759    0.000
    x6                0.989    0.085   11.688    0.000
  speed =~
    x7                1.000
    x8                1.045    0.076   13.759    0.000
    x9                0.905    0.064   14.119    0.000

Covariances:
  visual ~~
    textual           0.442    0.096    4.589    0.000
    speed             0.339    0.081    4.184    0.000
  textual ~~
    speed             0.247    0.078    3.182    0.001

Variances:
    x1                0.640    0.116    5.528    0.000
    x2                0.966    0.120    8.070    0.000
    x3                0.599    0.091    6.563    0.000
    x4                0.331    0.062    5.296    0.000
    x5                0.404    0.072    5.636    0.000
    x6                0.406    0.069    5.881    0.000
    x7                0.568    0.089    6.409    0.000
    x8                0.422    0.080    5.251    0.000
    x9                0.549    0.082    6.663    0.000
    visual            0.734    0.133    5.541    0.000
    textual           0.894    0.142    6.320    0.000
    speed             0.578    0.100    5.805    0.000
```

6.2.1 Constraining a single parameter to be equal across groups

If you want to constrain one or more parameters to be equal across groups, we can again use the `equal()` modifier. For example, to constrain the factor loading of the `x3` indicator to be equal across groups, we can use the `equal()` modifier as follows:

```
> HS.model <- ' visual  =~ x1 + x2 + equal(c("","visual=~x3"))*x3
+               textual =~ x4 + x5 + x6
+               speed   =~ x7 + x8 + x9 '
```

The argument of the `equal()` modifier is a vector with two elements, one for each group. The first element is empty, since we do not want to constrain the parameter in the first group.

6.2.2 Constraining groups of parameters to be equal across groups

Although the `equal()` modifier is very flexible, there is a more convenient way to impose equality constraints on a whole set of parameters (for example: all factor loadings, or all intercepts). We call these type of constraints *group equality constraints* and they can be specified by the `group.equal` argument in the `cfa` or `sem` function calls. For example, to constrain (all) the factor loadings to be equal across groups, you can proceed as follows:

```
> HS.model <- ' visual  =~ x1 + x2 + x3
+               textual =~ x4 + x5 + x6
+               speed   =~ x7 + x8 + x9 '
> fit <- cfa(HS.model, data=HolzingerSwineford1939, group="school",
+            group.equal=c("loadings"))
> summary(fit)

Lavaan (0.4-9) converged normally after 41 iterations

  Number of observations per group
  Pasteur                                          156
  Grant-White                                      145

  Estimator                                         ML
  Minimum Function Chi-square                  124.044
  Degrees of freedom                                54
  P-value                                        0.000

Chi-square for each group:

  Pasteur                                       68.825
  Grant-White                                   55.219

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

Group 1 [Pasteur]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                1.000
    x2                0.599    0.100    5.979    0.000
    x3                0.784    0.108    7.267    0.000
  textual =~
    x4                1.000
    x5                1.083    0.067   16.049    0.000
    x6                0.912    0.058   15.785    0.000
  speed =~
    x7                1.000
    x8              1.201    0.155    7.738    0.000
    x9              1.038    0.136    7.629    0.000

Covariances:
  visual ~~
    textual         0.416    0.097    4.271    0.000
    speed           0.169    0.064    2.643    0.008
  textual ~~
    speed           0.176    0.061    2.882    0.004

Variances:
    x1              0.551    0.137    4.010    0.000
    x2              1.258    0.155    8.117    0.000
    x3              0.882    0.128    6.884    0.000
    x4              0.434    0.070    6.238    0.000
    x5              0.508    0.082    6.229    0.000
    x6              0.266    0.050    5.294    0.000
    x7              0.849    0.114    7.468    0.000
    x8              0.515    0.095    5.409    0.000
    x9              0.658    0.096    6.865    0.000
    visual          0.805    0.171    4.714    0.000
    textual         0.913    0.137    6.651    0.000
    speed           0.305    0.078    3.920    0.000



Group 2 [Grant-White]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1              1.000
    x2              0.599    0.100    5.979    0.000
    x3              0.784    0.108    7.267    0.000
  textual =~
    x4              1.000
    x5              1.083    0.067   16.049    0.000
    x6              0.912    0.058   15.785    0.000
  speed =~
    x7              1.000
    x8              1.201    0.155    7.738    0.000
    x9              1.038    0.136    7.629    0.000

Covariances:
  visual ~~
    textual         0.437    0.099    4.423    0.000
    speed           0.314    0.079    3.958    0.000
  textual ~~
    speed           0.226    0.072    3.144    0.002

Variances:
    x1              0.645    0.127    5.084    0.000
    x2              0.933    0.121    7.732    0.000
    x3              0.605    0.096    6.282    0.000
    x4              0.329    0.062    5.279    0.000
    x5              0.384    0.073    5.270    0.000
    x6              0.437    0.067    6.576    0.000
    x7              0.599    0.090    6.651    0.000
    x8              0.406    0.089    4.541    0.000
    x9              0.532    0.086    6.202    0.000
    visual          0.722    0.161    4.490    0.000
    textual         0.906    0.136    6.646    0.000
    speed           0.475    0.109    4.347    0.000
```

More ‘group equality constraints’ can be added. In addition to the factor loadings, the following keywords are currently supported:

- `"intercepts"`: the intercepts of the observed variables
- `"means"`: the intercepts/means of the latent variables
- `"residuals"`: the residual variances of the observed variables
- `"residual.covariances"`: the residual covariances of the observed variables

- "lv.variances": the (residual) variances of the latent variables
- "lv.covariances": the (residual) covariances of the latent varibles
- "regressions": all regression coefficients in the model

If you omit the group.equal arguments, all parameters are freely estimated in each group (but the model structure is the same).

But what if you want to constrain a whole group of parameters (say all factor loadings and intercepts) across groups, except for one or two parameters that need to stay free in all groups. For this scenario, you can use the argument group.partial, containing the names of those parameters that need to remain free. For example:

```
> fit <- cfa(HS.model, data=HolzingerSwineford1939, group="school",
+             group.equal=c("loadings", "intercepts"),
+             group.partial=c("visual=~x2", "x7~1"))
```

6.2.3 Measurement Invariance

If you are interested in testing the measurement invariance of a CFA model across several groups, you can use the measurementInvariance function which performs a number of multiple group analyses in a particular sequence, with increasingly more restrictions on the parameters. Each model is compared to the baseline model and the previous model using chi-square difference tests. In addition, the difference in the cfi fit measure is also shown. Although the current implementation of the function is still a bit primitive, it does illustrate how the various components of the lavaan package can be used as building blocks for constructing higher level functions (such as the measurementInvariance function), something that is often very hard to accomplish with commercial software.

```
> measurementInvariance(HS.model, data = HolzingerSwineford1939,
+      group = "school")

Measurement invariance tests:

Model 1: configural invariance:
   chisq       df   pvalue      cfi    rmsea      bic
 115.851   48.000    0.000    0.923    0.097 7604.094

Model 2: weak invariance (equal loadings):
   chisq       df   pvalue      cfi    rmsea      bic
 124.044   54.000    0.000    0.921    0.093 7578.043

[Model 1 versus model 2]
  delta.chisq      delta.df delta.p.value     delta.cfi
        8.192         6.000         0.224         0.002

Model 3: strong invariance (equal loadings + intercepts):
   chisq       df   pvalue      cfi    rmsea      bic
 164.103   60.000    0.000    0.882    0.107 7686.588

[Model 1 versus model 3]
  delta.chisq      delta.df delta.p.value     delta.cfi
       48.251        12.000         0.000         0.041

[Model 2 versus model 3]
  delta.chisq      delta.df delta.p.value     delta.cfi
       40.059         6.000         0.000         0.038

Model 4: equal loadings + intercepts + means:
   chisq       df   pvalue      cfi    rmsea      bic
 204.605   63.000    0.000    0.854    0.122 7709.969

[Model 1 versus model 4]
  delta.chisq      delta.df delta.p.value     delta.cfi
       88.754        15.000         0.000         0.069

[Model 3 versus model 4]
  delta.chisq      delta.df delta.p.value     delta.cfi
       40.502         3.000         0.000         0.028
```

By adding the equal.partial argument, you can test for partial measurement invariance by allowing a few parameters to remain free.

7 Growth curve models

Another important type of latent variable models are latent growth curve models. Growth modeling is often used to analyze longitudinal or developmental data. In this type of data, an outcome measure is measured on several occasions, and we want to study the change over time. In many cases, the trajectory over time can be modeled as a simple linear or quadratic curve. Random effects are used to capture individual differences. The random effects are conveniently represented by (continuous) latent variables, often called *growth factors*. In the example below, we use an artifical toy dataset called `Demo.growth` where a score (say, a standardized score on a reading ability scale) is measured on 4 time points. To fit a linear growth model for these four time points, we need to specify a model with two latent variables: a random intercept, and a random slope:

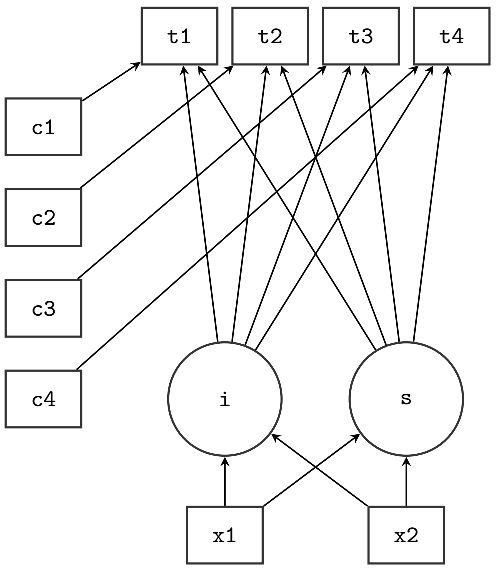
lavaan syntax

```
# linear growth model with 4 timepoints
# intercept and slope with fixed coefficients
 i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
 s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4
```

In this model, we have fixed all the coefficients of the growth functions. To fit this model, the lavaan package provides a special `growth` function:

```
> model <- ' i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
+            s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4 '
> fit <- growth(model, data=Demo.growth)
> summary(fit)

Lavaan (0.4-9) converged normally after 30 iterations

  Number of observations                           400

  Estimator                                         ML
  Minimum Function Chi-square                    8.069
  Degrees of freedom                                 5
  P-value                                        0.152

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  i =~
    t1                1.000
    t2                1.000
    t3                1.000
    t4                1.000
  s =~
    t1                0.000
    t2                1.000
    t3                2.000
    t4                3.000

Covariances:
  i ~~
    s                 0.618    0.071    8.686    0.000

Intercepts:
    t1                0.000
    t2                0.000
    t3                0.000
    t4                0.000
    i                 0.615    0.077    8.007    0.000
    s                 1.006    0.042   24.076    0.000

Variances:
    t1                0.595    0.086    6.944    0.000
    t2                0.676    0.061   11.061    0.000
    t3                0.635    0.072    8.761    0.000
    t4                0.508    0.124    4.090    0.000
    i                 1.932    0.173   11.194    0.000
    s                 0.587    0.052   11.336    0.000
```

Technically, the **growth** function is almost identical to the **sem** function. But a meanstructure is automatically assumed, and the observed intercepts are fixed to zero by default, while the latent variable intercepts/means are freely estimated. A slightly more complex model adds two regressors (`x1` and `x2`) that influence the latent growth factors. In addition, a time-varying covariate that influences the outcome measure at the four time points has been added to the model. A graphical representation of this model together with the corresponding lavaan syntax is presented below.



lavaan syntax

```
# intercept and slope
# with fixed coefficients
  i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
  s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4

# regressions
  i ~ x1 + x2
  s ~ x1 + x2

# time-varying covariates
  t1 ~ c1
  t2 ~ c2
  t3 ~ c3
  t4 ~ c4
```

For ease of copy/pasting, the complete R code needed to specify and fit this linear growth model with a time-varying covariate is printed again below:

R code

```
# a linear growth model with a time-varying covariate

model <- '
  # intercept and slope with fixed coefficients
    i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
    s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4

  # regressions
    i ~ x1 + x2
    s ~ x1 + x2

  # time-varying covariates
    t1 ~ c1
    t2 ~ c2
    t3 ~ c3
    t4 ~ c4
'

fit <- growth(model, data=Demo.growth)
summary(fit)
```

8 Additional information

8.1 Using a covariance matrix as input

If you have no full dataset, but you do have a sample covariance matrix, you can still fit your model. If you need a meanstructure, you will need to provide a mean vector too. Importantly, you also need to specify the number of observations that were used to compute the sample moments. The following example illustrates the use of a sample covariance matrix as input:

```
> wheaton.cov <- matrix(c(
+        11.834,     0,        0,        0,       0,        0,
+         6.947,    9.364,     0,        0,       0,        0,
+         6.819,    5.091,   12.532,     0,       0,        0,
+         4.783,    5.028,    7.495,    9.986,    0,        0,
+        -3.839,   -3.889,   -3.841,   -3.625,   9.610,     0,
+       -21.899,  -18.831,  -21.748,  -18.775,  35.522,  450.288),
+       6, 6, byrow=TRUE)
> colnames(wheaton.cov) <- rownames(wheaton.cov) <-
+   c("anomia67", "powerless67", "anomia71",
+     "powerless71", "education", "sei")
> wheaton.model <- '
+   # measurement model
+     ses     =~ education + sei
+     alien67 =~ anomia67 + powerless67
+     alien71 =~ anomia71 + powerless71
+
+   # equations
+     alien71 ~ alien67 + ses
+     alien67 ~ ses
+
+   # correlated residuals
+     anomia67 ~~ anomia71
+     powerless67 ~~ powerless71
+ '
> fit <- sem(wheaton.model, sample.cov=wheaton.cov, sample.nobs=932)
> summary(fit, standardized=TRUE)
```

```
Lavaan (0.4-9) converged normally after 82 iterations

  Number of observations                           932

  Estimator                                         ML
  Minimum Function Chi-square                    4.735
  Degrees of freedom                                 4
  P-value                                        0.316

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)   Std.lv  Std.all
Latent variables:
  ses =~
    education         1.000                               2.607    0.842
    sei               5.219    0.422   12.364    0.000   13.609    0.642
  alien67 =~
    anomia67          1.000                               2.663    0.774
    powerless67       0.979    0.062   15.895    0.000    2.606    0.852
  alien71 =~
    anomia71          1.000                               2.850    0.805
    powerless71       0.922    0.059   15.498    0.000    2.628    0.832

Regressions:
  alien71 ~
    alien67           0.607    0.051   11.898    0.000    0.567    0.567
    ses              -0.227    0.052   -4.334    0.000   -0.207   -0.207
  alien67 ~
    ses              -0.575    0.056  -10.195    0.000   -0.563   -0.563
```

```
Covariances:
  anomia67 ~~
    anomia71          1.623    0.314    5.176    0.000    1.623    0.356
  powerless67 ~~
    powerless71       0.339    0.261    1.298    0.194    0.339    0.121

Variances:
    education         2.801    0.507    5.525    0.000    2.801    0.292
    sei             264.597   18.126   14.597    0.000  264.597    0.588
    anomia67          4.731    0.453   10.441    0.000    4.731    0.400
    powerless67       2.563    0.403    6.359    0.000    2.563    0.274
    anomia71          4.399    0.515    8.542    0.000    4.399    0.351
    powerless71       3.070    0.434    7.070    0.000    3.070    0.308
    ses               6.798    0.649   10.475    0.000    1.000    1.000
    alien67           4.841    0.467   10.359    0.000    0.683    0.683
    alien71           4.083    0.404   10.104    0.000    0.503    0.503
```

Only the lower half of the covariance matrix (including the diagonal) is used. The rownames (and optionally the colnames) must contain the names of the observed variables that are used in the model syntax.
If you have multiple groups, the `sample.cov` argument must be a list containing the sample variance-covariance matrix of each group as a seperate element in the list. If a meanstructure is needed, the `sample.mean` argument must be a list containing the sample means of each group. Finally, the `sample.nobs` argument can be either a list or a integer vector containing the number of observations for each group.

8.2 Estimators, Standard errors and Missing Values

8.2.1 Estimators

The default estimator in the lavaan package is maximum likelihood (`estimator = "ML"`). Alternative estimators currently available in `lavaan` are:

- `"GLS"` for generalized least squares. For complete data only.
- `"WLS"` for weighted least squares (sometimes called ADF estimation). For complete data only.
- `"MLM"` for maximum likelihood estimation with robust standard errors and a Satorra-Bentler scaled test statistic. For complete data only.
- `"MLF"` for maximum likelihood estimation with standard errors based on the first-order derivatives, and a conventional test statistic. For both complete and incomplete data.
- `"MLR"` maximum likelihood estimation with robust (Huber-White) standard errors and a scaled test statistic that is (asymptotically) equal to the Yuan-Bentler test statistic. For both complete and incomplete data.

If maximum likelihood estimation is used (`"ML"`, `"MLM"`, `"MLF"` or `"MLR"`), the default behavior of lavaan is to base the analysis on the so-called *biased* sample covariance matrix, where the elements are divided by n instead of $n-1$. This is done internally, and should not be done by the user. In addition, the chi-square statistic is computed by multiplying the minimum function value with a factor n (instead of $n-1$). This is similar to the Mplus program. If you prefer to use an unbiased covariance, and $n-1$ as the multiplier to compute the chi-square statistic, you need to specify the `likelihood="wishart"` argument when calling the fitting functions. For example:

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939, likelihood = "wishart")
> fit

Lavaan (0.4-9) converged normally after 35 iterations

  Number of observations                           301

  Estimator                                         ML
  Minimum Function Chi-square                   85.022
  Degrees of freedom                                24
  P-value                                        0.000
```

The value of the test statistic will be closer to the value reported by programs like EQS, LISREL or AMOS, since they all use the 'Wishart' approach when using the maximum likelihood estimator. The program Mplus, on the other hand, uses the 'normal' approach to maximum likelihood estimation.

8.2.2 Missing values

If the data contain missing values, the default behavior is listwise deletion. If the missing mechanism is MCAR (missing completely at random) or MAR (missing at random), the lavaan package provides case-wise (or 'full information') maximimum likelihood estimation. You can turn this feature on, by using the argument `missing="ml"` when calling the fitting function. An unrestricted (h1) model will automatically be estimated, so that all common fit indices are available.

8.2.3 Standard Errors

Standard errors are (by default) based on the expected information matrix. The only exception is when data are missing and full information ML is used (via `missing="ml"`). In this case, the observed information matrix is used to compute the standard errors. The user can change this behavior by using the `information` argument, which can be set to `"expected"` or `"observed"`. If the estimator is simply `"ML"`, you request robust standard errors by using the `se` argument, which can be set to `"robust.mlm"`, `"robust.mlr"` or `"first.order"`. Or simply to `"none"` if you don't need them. This will not affect the test statistic. In fact, you can choose the test statistic independently by using the `"test"` argument, which can be set to `"Satorra-Bentler"` of `"Yuan-Bentler"`.

8.3 Modification Indices

Modification indices can be requested by adding the `modindices=TRUE` argument in the `summary` call, or by calling the `modindices` function directly. The `modindices` function returns a data frame. For example, to see only the modification indices for the factor loadings, you can use something like this:

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939)
> mi <- modindices(fit)
> mi[mi$op == "=~", ]
```

```
       lhs op rhs     mi    epc sepc.lv sepc.all
1   visual =~  x1     NA     NA      NA       NA
2   visual =~  x2  0.000  0.000   0.000    0.000
3   visual =~  x3  0.000  0.000   0.000    0.000
4   visual =~  x4  1.211  0.077   0.069    0.059
5   visual =~  x5  7.441 -0.210  -0.189   -0.147
6   visual =~  x6  2.843  0.111   0.100    0.092
7   visual =~  x7 18.631 -0.422  -0.380   -0.349
8   visual =~  x8  4.295 -0.210  -0.189   -0.187
9   visual =~  x9 36.411  0.577   0.519    0.515
10 textual =~  x1  8.903  0.350   0.347    0.297
11 textual =~  x2  0.017 -0.011  -0.011   -0.010
12 textual =~  x3  9.151 -0.272  -0.269   -0.238
13 textual =~  x4     NA     NA      NA       NA
14 textual =~  x5  0.000  0.000   0.000    0.000
15 textual =~  x6  0.000  0.000   0.000    0.000
16 textual =~  x7  0.098 -0.021  -0.021   -0.019
17 textual =~  x8  3.359 -0.121  -0.120   -0.118
18 textual =~  x9  4.796  0.138   0.137    0.136
19   speed =~  x1  0.014  0.024   0.015    0.013
20   speed =~  x2  1.580 -0.198  -0.123   -0.105
21   speed =~  x3  0.716  0.136   0.084    0.075
22   speed =~  x4  0.003 -0.005  -0.003   -0.003
23   speed =~  x5  0.201 -0.044  -0.027   -0.021
24   speed =~  x6  0.273  0.044   0.027    0.025
25   speed =~  x7     NA     NA      NA       NA
26   speed =~  x8  0.000  0.000   0.000    0.000
27   speed =~  x9  0.000  0.000   0.000    0.000
```

Modification indices are printed out for each nonfree (or nonredundant) parameter. The modification indices are supplemented by the expected parameter change values (column `epc`). The last two columns are the standardized and completely standardized EPC values respectively.

8.4 Extracting information from a fitted model

The `summary` function gives a nice overview of a fitted model, but is for display only. If you need the actual numbers for further processing, you may prefer to use one of several 'extractor' functions. We have already seen the `coef` function which extracts the estimated parameters of a fitted model. Other extractor functions are discussed below.

8.4.1 parameterEstimates

The `parameterEstimates` function extracts not only the values of the estimated parameters, but also the standard errors, the z-values, the standardized parameter values, as a convenient data frame. For example

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939)
> parameterEstimates(fit)

       lhs op     rhs   est    se      z est.std est.std.all
1   visual =~      x1 1.000 0.000     NA   0.900       0.772
2   visual =~      x2 0.554 0.100  5.554   0.498       0.424
3   visual =~      x3 0.729 0.109  6.685   0.656       0.581
4  textual =~      x4 1.000 0.000     NA   0.990       0.852
5  textual =~      x5 1.113 0.065 17.014   1.102       0.855
6  textual =~      x6 0.926 0.055 16.703   0.917       0.838
7    speed =~      x7 1.000 0.000     NA   0.619       0.570
8    speed =~      x8 1.180 0.165  7.152   0.731       0.723
9    speed =~      x9 1.082 0.151  7.155   0.670       0.665
10      x1 ~~      x1 0.549 0.114  4.833   0.549       0.404
11      x2 ~~      x2 1.134 0.102 11.146   1.134       0.821
12      x3 ~~      x3 0.844 0.091  9.317   0.844       0.662
13      x4 ~~      x4 0.371 0.048  7.779   0.371       0.275
14      x5 ~~      x5 0.446 0.058  7.642   0.446       0.269
15      x6 ~~      x6 0.356 0.043  8.277   0.356       0.298
16      x7 ~~      x7 0.799 0.081  9.823   0.799       0.676
17      x8 ~~      x8 0.488 0.074  6.573   0.488       0.477
18      x9 ~~      x9 0.566 0.071  8.003   0.566       0.558
19  visual ~~  visual 0.809 0.145  5.564   1.000       1.000
20 textual ~~ textual 0.979 0.112  8.737   1.000       1.000
21   speed ~~   speed 0.384 0.086  4.451   1.000       1.000
22  visual ~~ textual 0.408 0.074  5.552   0.459       0.459
23  visual ~~   speed 0.262 0.056  4.660   0.471       0.471
24 textual ~~   speed 0.173 0.049  3.518   0.283       0.283
```

8.4.2 standardizedSolution

The `standardizedSolution` function is similar to the `parameterEstimates` function, but only shows the unstandardized and standardized parameter estimates.

8.4.3 fitted.values

The `fitted` and `fitted.values` functions return the implied (fitted) covariance matrix (and mean vector) of a fitted model.

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939)
> fitted(fit)

$cov
   x1    x2    x3    x4    x5    x6    x7    x8    x9   
x1 1.358                                                
x2 0.448 1.382                                          
x3 0.590 0.327 1.275                                    
x4 0.408 0.226 0.298 1.351                              
x5 0.454 0.252 0.331 1.090 1.660                        
x6 0.378 0.209 0.276 0.907 1.010 1.196                  
x7 0.262 0.145 0.191 0.173 0.193 0.161 1.183            
x8 0.309 0.171 0.226 0.205 0.228 0.190 0.453 1.022      
x9 0.284 0.157 0.207 0.188 0.209 0.174 0.415 0.490 1.015

$mean
x1 x2 x3 x4 x5 x6 x7 x8 x9 
 0  0  0  0  0  0  0  0  0 
```

8.4.4 residuals

The `resid` or `residuals` functions return (unstandardized) residuals of a fitted model. This is simply the difference between the observed and implied covariance matrix and mean vector. If the estimator is maximum likelihood, it is also possible to obtain the normalized and the standardized residuals.

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939)
> resid(fit, type = "standardized")
```

```
$cov
   x1     x2     x3     x4     x5     x6     x7     x8     x9
x1     NA
x2 -2.196  0.000
x3 -1.199  2.692  0.000
x4  2.465 -0.283 -1.948  0.000
x5 -0.362 -0.610 -4.443  0.856  0.000
x6  2.032  0.661 -0.701     NA  0.633  0.000
x7 -3.787 -3.800 -1.882  0.839 -0.837 -0.321     NA
x8 -1.456 -1.137 -0.305 -2.049 -1.100 -0.635  3.804  0.000
x9  4.062  1.517  3.328  1.237  1.723  1.436 -2.772     NA  0.000

$mean
x1 x2 x3 x4 x5 x6 x7 x8 x9
 0  0  0  0  0  0  0  0  0
```

8.4.5 vcov

The `vcov` function returns the estimated covariance matrix of the parameter estimates.

8.4.6 AIC and BIC

The `AIC` and `BIC` functions return the AIC and BIC values of a fitted model.

8.4.7 fitMeasures

The `fitMeasures` function returns all the fit measures computed by lavaan as a named numeric vector. If you only want the value of a single fit measure, say, the CFI, you give the name (in lower case) as the second argument:

```
> fit <- cfa(HS.model, data = HolzingerSwineford1939)
> fitMeasures(fit, "cfi")

  cfi
0.931
```

8.4.8 inspect

If you want to peek inside a fitted `lavaan` object (the object that is returned by a call to `cfa`, `sem` or `growth`), you can use the `inspect` function, with a variety of options. By default, calling `inspect` on a fitted `lavaan` object returns a list of the model matrices that are used internally to represent the model. The free parameters are nonzero integers.

```
> inspect(fit)

$lambda
   visual textul speed
x1      0      0     0
x2      1      0     0
x3      2      0     0
x4      0      0     0
x5      0      3     0
x6      0      4     0
x7      0      0     0
x8      0      0     5
x9      0      0     6

$theta
   x1 x2 x3 x4 x5 x6 x7 x8 x9
x1  7
x2  0  8
x3  0  0  9
x4  0  0  0 10
x5  0  0  0  0 11
x6  0  0  0  0  0 12
x7  0  0  0  0  0  0 13
x8  0  0  0  0  0  0  0 14
x9  0  0  0  0  0  0  0  0 15

$psi
        visual textul speed
visual  16
textual 19     17
speed   20     21     18
```

To see the starting values of parameters in each model matrix, type

```
> inspect(fit, what = "start")

$lambda
   visual textul speed
x1  1.000  0.000 0.000
x2  0.778  0.000 0.000
x3  1.107  0.000 0.000
x4  0.000  1.000 0.000
x5  0.000  1.133 0.000
x6  0.000  0.924 0.000
x7  0.000  0.000 1.000
x8  0.000  0.000 1.225
x9  0.000  0.000 0.854

$theta
   x1    x2    x3    x4    x5    x6    x7    x8    x9
x1 0.681
x2 0.000 0.693
x3 0.000 0.000 0.640
x4 0.000 0.000 0.000 0.678
x5 0.000 0.000 0.000 0.000 0.833
x6 0.000 0.000 0.000 0.000 0.000 0.600
x7 0.000 0.000 0.000 0.000 0.000 0.000 0.594
x8 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.513
x9 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.509

$psi
        visual textul speed
visual  0.05
textual 0.00   0.05
speed   0.00   0.00   0.05
```

To see how lavaan internally represents a model, you can type

```
> inspect(fit, what = "list")

   id     lhs op     rhs user group free ustart fixed.x label eq.id free.uncon
1   1  visual =~      x1    1     1    0      1       0           0          0
2   2  visual =~      x2    1     1    1     NA       0           0          1
3   3  visual =~      x3    1     1    2     NA       0           0          2
4   4 textual =~      x4    1     1    0      1       0           0          0
5   5 textual =~      x5    1     1    3     NA       0           0          3
6   6 textual =~      x6    1     1    4     NA       0           0          4
7   7   speed =~      x7    1     1    0      1       0           0          0
8   8   speed =~      x8    1     1    5     NA       0           0          5
9   9   speed =~      x9    1     1    6     NA       0           0          6
10 10      x1 ~~      x1    0     1    7     NA       0           0          7
11 11      x2 ~~      x2    0     1    8     NA       0           0          8
12 12      x3 ~~      x3    0     1    9     NA       0           0          9
13 13      x4 ~~      x4    0     1   10     NA       0           0         10
14 14      x5 ~~      x5    0     1   11     NA       0           0         11
15 15      x6 ~~      x6    0     1   12     NA       0           0         12
16 16      x7 ~~      x7    0     1   13     NA       0           0         13
17 17      x8 ~~      x8    0     1   14     NA       0           0         14
18 18      x9 ~~      x9    0     1   15     NA       0           0         15
19 19  visual ~~  visual    0     1   16     NA       0           0         16
20 20 textual ~~ textual    0     1   17     NA       0           0         17
21 21   speed ~~   speed    0     1   18     NA       0           0         18
22 22  visual ~~ textual    0     1   19     NA       0           0         19
23 23  visual ~~   speed    0     1   20     NA       0           0         20
24 24 textual ~~   speed    0     1   21     NA       0           0         21
```

For more inspect options, see the help page for the `lavaan` class which you can find by typing the following:

```
> class?lavaan
```

9 Report a bug, or give use feedback

If you have found a bug, or something unpleasant happened, please let us now. You can send an email to Yves.Rosseel@UGent.be. Start the subject line with [lavaan], and it will get the proper attention. To help us with your problem (and fix our bugs), we need two types of information from you:

1. a detailed description of the problem: what happened, which error message or warning message did you see, and when does it occur. If possible at all, provide a reproducable example of the syntax that generated the error.
2. the output of the following command in R:

   ```
   > sessionInfo()
   ```

   This will show vital information about your platform, the version of R, and other stuff that might help us with identifying the problem.

We also welcome all suggestions, both on the software and the documentation.

A Examples from the Mplus User's Guide

Below, we provide some examples of lavaan model syntax to mimic the examples in the Mplus User's guide. The datafiles can be downloaded from http://www.statmodel.com/ugexcerpts.shtml.

A.1 Chapter 3: Regression and Path Analysis

```
# ex3.1
Data <- read.table("ex3.1.dat")
names(Data) <- c("y1","x1","x2")

model.ex3.1 <- ' y1 ~ x1 + x2 '
fit <- sem(model.ex3.1, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


# ex3.11
Data <- read.table("ex3.11.dat")
names(Data) <- c("y1","y2","y3",
                 "x1","x2","x3")

model.ex3.11 <- ' y1 + y2 ~ x1 + x2 + x3
                       y3 ~ y1 + y2 + x2 '

fit <- sem(model.ex3.11, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)
```

A.2 Chapter 5: Confirmatory factor analysis and structural equation modeling

```
# ex5.1
Data <- read.table("ex5.1.dat")
names(Data) <- paste("y", 1:6, sep="")

model.ex5.1 <- ' f1 =~ y1 + y2 + y3
                 f2 =~ y4 + y5 + y6 '

fit <- cfa(model.ex5.1, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


# ex5.6
Data <- read.table("ex5.6.dat")
names(Data) <- paste("y", 1:12, sep="")

model.ex5.6 <- ' f1 =~ y1  + y2  + y3
```

```
                 f2 =~ y4  + y5  + y6
                 f3 =~ y7  + y8  + y9
                 f4 =~ y10 + y11 + y12
                 f5 =~ f1  + f2  + f3  + f4 '

fit <- cfa(model.ex5.6, data=Data, estimator="ML")
summary(fit, standardized=TRUE, fit.measures=TRUE)


# ex5.8
Data <- read.table("ex5.8.dat")
names(Data) <- c(paste("y", 1:6, sep=""), paste("x", 1:3, sep=""))

model.ex5.8 <- ' f1 =~ y1 + y2 + y3
                 f2 =~ y4 + y5 + y6
                 f1 + f2 ~ x1 + x2 + x3 '

fit <- cfa(model.ex5.8, data=Data, estimator="ML")
summary(fit, standardized=TRUE, fit.measures=TRUE)


# ex5.9
Data <- read.table("ex5.9.dat")
names(Data) <- c("y1a","y1b","y1c","y2a","y2b","y2c")

model.ex5.9 <- ' f1 =~ 1*y1a + 1*y1b + 1*y1c
                 f2 =~ 1*y2a + 1*y2b + 1*y2c
                 y1a + y1b + y1c ~ i1*1
                 y2a + y2b + y2c ~ i2*1 '

fit <- cfa(model.ex5.9, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


# ex5.11
Data <- read.table("ex5.11.dat")
names(Data) <- paste("y", 1:12, sep="")

model.ex5.11 <- ' f1 =~ y1 + y2 + y3
                  f2 =~ y4 + y5 + y6
                  f3 =~ y7 + y8 + y9
                  f4 =~ y10 + y11 + y12
                  f3 ~ f1 + f2
                  f4 ~ f3 '

fit <- sem(model.ex5.11, data=Data, estimator="ML")
summary(fit, standardized=TRUE, fit.measures=TRUE)


# ex5.14
Data <- read.table("ex5.14.dat")
names(Data) <- c("y1","y2","y3","y4","y5","y6", "x1","x2","x3", "g")

model.ex5.14 <- ' f1 =~ y1 + y2 + y3
                  f2 =~ y4 + y5 + y6
                  f1 + f2 ~ x1 + x2 + x3 '

fit <- cfa(model.ex5.14, data=Data, group="g", meanstructure=FALSE,
           group.equal=c("loadings"), group.partial=c("f1=~y3"))
summary(fit, standardized=TRUE, fit.measures=TRUE)

# ex5.15
Data <- read.table("ex5.15.dat")
names(Data) <- c("y1","y2","y3","y4","y5","y6", "x1","x2","x3", "g")
```

```
model.ex5.15 <- ' f1 =~ y1 + y2 + y3
                  f2 =~ y4 + y5 + y6
                  f1 + f2 ~ x1 + x2 + x3 '

fit <- cfa(model.ex5.15, data=Data, group="g", meanstructure=TRUE,
           group.equal=c("loadings", "intercepts"),
           group.partial=c("f1=~y3", "y3~1"))
summary(fit, standardized=TRUE, fit.measures=TRUE)

# ex5.20
Data <- read.table("ex5.20.dat")
names(Data) <- paste("y", 1:6, sep="")

model.ex5.20 <- ' f1 =~ y1 + lam2*y2 + lam3*y3
                  f2 =~ y4 + lam5*y5 + lam6*y6
                  f1 ~~ vf1*f1 + start(1.0)*f1   ## otherwise, neg vf2
                  f2 ~~ vf2*f2 + start(1.0)*f2   ##
                  y1 ~~ ve1*y1
                  y2 ~~ ve2*y2
                  y3 ~~ ve3*y3
                  y4 ~~ ve4*y4
                  y5 ~~ ve5*y5
                  y6 ~~ ve6*y6

                  # constraints
                  lam2^2*vf1/(lam2^2*vf1 + ve2) ==
                      lam5^2*vf2/(lam5^2*vf2 + ve5)
                  lam3*sqrt(vf1)/sqrt(lam3^2*vf1 + ve3) ==
                      lam6*sqrt(vf2)/sqrt(lam6^2*vf2 + ve6)
                  ve2 > ve5
                  ve4 > 0
                '

fit <- cfa(model.ex5.20, data=Data, estimator="ML")
summary(fit, standardized=TRUE, fit.measures=TRUE)
```

A.3 Chapter 6: Growth modeling

```
# 6.1
Data <- read.table("ex6.1.dat")
names(Data) <- c("y11","y12","y13","y14")

model.ex6.1 <- ' i =~ 1*y11 + 1*y12 + 1*y13 + 1*y14
                 s =~ 0*y11 + 1*y12 + 2*y13 + 3*y14 '

fit <- growth(model.ex6.1, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


#6.8
Data <- read.table("ex6.8.dat")
names(Data) <- c("y11","y12","y13","y14")

model.ex6.8 <- ' i =~ 1*y11 + 1*y12 + 1*y13 + 1*y14
                 s =~ 0*y11 + 1*y12 + start(2)*y13 + start(3)*y14 '

fit <- growth(model.ex6.8, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


#6.9
Data <- read.table("ex6.9.dat")
names(Data) <- c("y11","y12","y13","y14")

model.ex6.9 <- ' i =~ 1*y11 + 1*y12 + 1*y13 + 1*y14
```

```
                 s =~ 0*y11 + 1*y12 + 2*y13 + 3*y14
                 q =~ 0*y11 + 1*y12 + 4*y13 + 9*y14 '

fit <- growth(model.ex6.9, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


#6.10
Data <- read.table("ex6.10.dat")
names(Data) <- c("y11","y12","y13","y14","x1","x2","a31","a32","a33","a34")

model.ex6.10 <- ' i =~ 1*y11 + 1*y12 + 1*y13 + 1*y14
                  s =~ 0*y11 + 1*y12 + 2*y13 + 3*y14
                  i + s ~ x1 + x2
                  y11 ~ a31
                  y12 ~ a32
                  y13 ~ a33
                  y14 ~ a34 '

fit <- growth(model.ex6.10, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)


#6.11
Data <- read.table("ex6.11.dat")
names(Data) <- c("y1","y2","y3","y4","y5")

modelex6.11 <- ' i  =~ 1*y1 + 1*y2 + 1*y3 + 1*y4 + 1*y5
                 s1 =~ 0*y1 + 1*y2 + 2*y3 + 2*y4 + 2*y5
                 s2 =~ 0*y1 + 0*y2 + 0*y3 + 1*y4 + 2*y5 '

fit <- growth(modelex6.11, data=Data)
summary(fit, standardized=TRUE, fit.measures=TRUE)
```