lavaan: an R package for structural equation modeling and more Version 0.3-1 (BETA)

Yves Rosseel
Department of Data Analysis
Ghent University (Belgium)

August 17, 2010

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.

1 Before you start

Before you start, read these points carefully:

- First of all, you must have a 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 (see section 9), and some features are simply not implemented yet. Some important features that are currently *not* available in lavaan are:
  - support for categorical/censored variables (this will be available in the next release of lavaan)
  - 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 was 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 other SEM packages, you can use the optional argument `mimic.Mplus=FALSE` 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 0.3-1
lavaan is BETA software! Please report any bugs.
```

If you see the startup message (showing the version number, and a reminder that this is beta software), you're all set. Move on to the next section. If you get an error, or nothing happens at all, please let us know. See section 11 for how to submit a bug report.

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 need to '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
```

Further more, 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 near 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
                 y ~ 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
            '
```

This 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, you may prefer to type it in a separate text file called, say, `myModel.lms`. 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.lms")
```

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 variables models: two examples

4.1 A first example: confirmatory factor analysis (CFA)

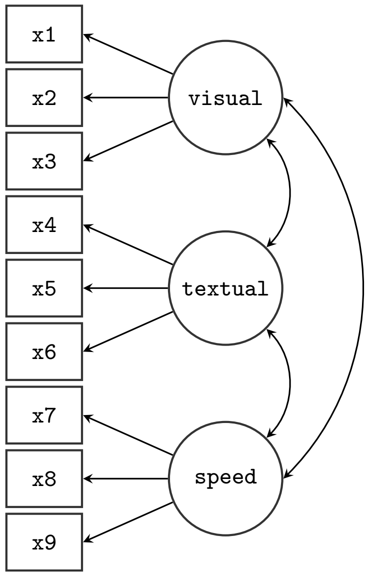
We start with a simple example of confirmatory factor analysis. The `lavaan` package contains a built-in dataset called `HolzingerSwineford1939`. See the help page for this dataset by typing

```
> ?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 simple 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, lavaan 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, residuals variances are added automatically. And third, all 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. More on this later.
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)

Model converged normally after 35 iterations using ML

  Minimum Function Chi-square                 85.306
  Degrees of freedom                              24
  P-value                                     0.0000

Chi-square test baseline model:

  Minimum Function Chi-square                918.852
  Degrees of freedom                              36
  P-value                                     0.0000
```

```
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

  Akaike (AIC)                                7517.490
  Bayesian (BIC)                              7595.339

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


Model estimates:

                   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

Latent 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

Latent variances:
    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

Residual 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    7.778    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
```

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

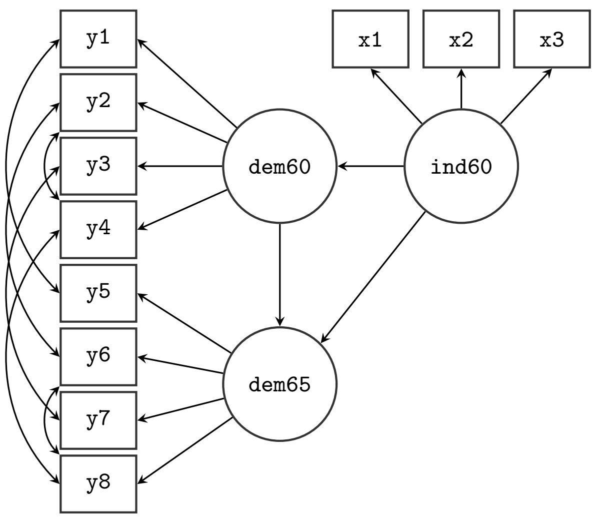
```
# load the lavaan package (only needed once per session)
library(lavaan)

# 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.
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)

Model converged normally after 95 iterations using ML

  Minimum Function Chi-square              38.125
  Degrees of freedom                           35
  P-value                                  0.3292

                   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    6.888    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

Residual covariances:
  y1 ~~
    y5                0.624    0.358    1.741    0.082    0.624    0.092
  y2 ~~
    y4                1.313    0.702    1.870    0.061    1.313    0.101
    y6                2.153    0.734    2.934    0.003    2.153    0.165
  y3 ~~
    y7                0.795    0.608    1.308    0.191    0.795    0.075
  y4 ~~
    y8                0.348    0.442    0.787    0.431    0.348    0.033
  y6 ~~
    y8                1.356    0.568    2.386    0.017    1.356    0.126

Latent variances:
    ind60             0.448    0.087    5.173    0.000    1.000    1.000

Residual 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                5.068    0.952    5.325    0.000    5.068    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
    dem60             3.956    0.921    4.294    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:

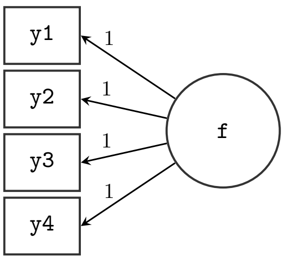
R code

```
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

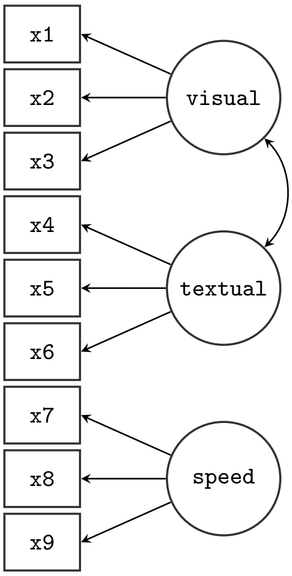
Consider a simple one-factor model with 4 indicators. By default, lavaan will always fix the factor loading of the first indicator to 1. The other three factor loadings are free, and their values are estimated by the model. But suppose that you have good reasons the fix all the factor loadings to 1. 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 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
  2.18036714   1.81851074   1.25674943   1.05771682   1.26479406   1.18569755
   dem65=~y7    dem65=~y8       x1~~x1       x2~~x2       x3~~x3       y1~~y1
  1.27951120   1.26594723   0.08154936   0.11980662   0.46670208   1.89140112
      y5~~y1       y2~~y2       y4~~y2       y6~~y2       y3~~y3       y7~~y3
  0.62366956   7.37297430   1.31303762   2.15291322   5.06753538   0.79500700
      y4~~y4       y8~~y4       y5~~y5       y6~~y6       y8~~y6       y7~~y7
  3.14779771   0.34822664   2.35096513   4.95394085   1.35617088   3.43142160
      y8~~y8  dem60~ind60  dem65~ind60  dem65~dem60 ind60~~ind60 dem60~~dem60
  3.25413086   1.48300197   0.57233362   0.83734605   0.44843768   3.95598568
dem65~~dem65
  0.17248060
```

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 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"`.

5.4 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.

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)

Model converged normally after 35 iterations using ML

  Minimum Function Chi-square                 85.306
  Degrees of freedom                              24
  P-value                                     0.0000

                   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

Latent 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

Latent means/intercepts:
    visual            0.000
    textual           0.000
    speed             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

Latent variances:
    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

Residual 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    7.778    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
```

As you can see in the output, the model includes intercept parameters for both the observed and latent variables. By default, the latent variable intercepts (which in this case correspond to the latent *means*) are fixed 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 ~ 0.5*1
  x2 ~ 0.5*1
  x3 ~ 0.5*1
  x4 ~ 0.5*1
```

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)

Model converged normally after 56 iterations using ML

  Minimum Function Chi-square             115.851
  Degrees of freedom                           48
  P-value                                  0.0000

Chi-square for each group:

  Grant-White                              51.542
  Pasteur                                  64.309

Group 1 [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

Latent 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

Latent variances:
    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

Residual 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    4.248    0.000
    x9                0.535    0.089    6.010    0.000


Group 2 [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    4.058    0.000
    x9                0.922    0.225    4.104    0.000

Latent 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

Latent variances:
    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

Residual 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
```

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. For example:

lavaan syntax

```
HS.model <- ' visual  =~             x1 +
                                      x2 +
                            c(0.6, 0.8)*x3

              textual =~                    x4 +
                            start(c(1.2, 0.6))*x5 +
                                            x6
```

```
            speed   =~                        x7 +
                                              x8 +
                      label(c("x9.group1",
                              "x9.group2"))*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. In the definition of the `textual` factor, two different starting values are provided for the `x5` indicator; one for each group. Finally, in the definition of the `speed` factor, we changed the labels of the parameters associated with the factor loading of the `x9` indicator. For the last modification, we can see the effect by requesting the values of the estimated parameters:

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

     Grant-White.visual=~x2      Grant-White.textual=~x5
                  0.5880695                    0.9904237
    Grant-White.textual=~x6       Grant-White.speed=~x8
                  0.9617101                    1.2276221
                  x9.group1          Grant-White.x1~~x1
                  1.0892346                    0.5712898
         Grant-White.x2~~x2          Grant-White.x3~~x3
                  0.9408347                    0.6825272
         Grant-White.x4~~x4          Grant-White.x5~~x5
                  0.3146871                    0.4171026
         Grant-White.x6~~x6          Grant-White.x7~~x7
                  0.4084114                    0.6115976
         Grant-White.x8~~x8          Grant-White.x9~~x9
                  0.4158980                    0.5169119
  Grant-White.visual~~visual  Grant-White.textual~~visual
                  0.8256916                    0.4706483
   Grant-White.speed~~visual Grant-White.textual~~textual
                  0.3353609                    0.9425314
  Grant-White.speed~~textual     Grant-White.speed~~speed
                  0.2233279                    0.4502038
         Pasteur.visual=~x2         Pasteur.textual=~x5
                  0.5287217                    1.1895757
        Pasteur.textual=~x6           Pasteur.speed=~x8
                  0.8777832                    1.1373990
                  x9.group2              Pasteur.x1~~x1
                  0.9525108                    0.5387143
             Pasteur.x2~~x2              Pasteur.x3~~x3
                  1.2742905                    0.8786946
             Pasteur.x4~~x4              Pasteur.x5~~x5
                  0.4304867                    0.4496137
             Pasteur.x6~~x6              Pasteur.x7~~x7
                  0.2893181                    0.8308872
             Pasteur.x8~~x8              Pasteur.x9~~x9
                  0.5134012                    0.6699607
      Pasteur.visual~~visual      Pasteur.textual~~visual
                  0.8208162                    0.4023822
       Pasteur.speed~~visual     Pasteur.textual~~textual
                  0.1742155                    0.8892006
      Pasteur.speed~~textual         Pasteur.speed~~speed
                  0.1784486                    0.3390281
```

If multiple groups are involved, the 'default' parameter names include the name of the group. The labels for the `x9` indicator are changed to the custom names provided via the `label()` modifier.

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("","Grant-White.visual=~x2"))*x3
+               textual =~ x4 + x5 + x6
+               speed   =~ x7 + x8 + x9 '
```

The first element of the `equal` modifier is the empty string: we do *not* want to impose any equality constraints on the factor loading in the first group. All we want is to set the value of the factor loading (of `x3`) in the second group (the Pasteur school) equal to the freely estimated value in first group (the Grant-White school). That is why we take `"Grant-White.visual= x2"` as the label of the 'target' parameter in this 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 constraints* and they can be specified by the `group.constraints` argument in the `cfa` or `sem` function call. 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.constraints=c("loadings"))
> summary(fit)

Model converged normally after 42 iterations using ML

  Minimum Function Chi-square              124.044
  Degrees of freedom                            54
  P-value                                   0.0000

Chi-square for each group:

  Grant-White                               55.219
  Pasteur                                   68.825

Group 1 [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

Latent 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

Latent variances:
    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

Residual 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


Group 2 [Pasteur]:

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                 1.000
    x2                 0.599
    x3                 0.784
  textual =~
    x4                 1.000
    x5                 1.083
    x6                 0.912
  speed =~
    x7                 1.000
    x8                 1.201
    x9                 1.038

Latent 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

Latent variances:
    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

Residual 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
```

More 'group' constraints can be added. In addition to the factor loadings, you can also constrain the `"intercepts"` of the observed variables, `"means"` of latent variables, and `"residuals"` (residual variances of observed variables) to be equal across groups, simply by adding them to the `group.constraints` argument. If you omit the `group.constraints` arguments, all parameters are freely estimated in each group (but the model structure is the same).

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 `measurement.invariance` 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 `measurement.invariance` function), something that is often very hard to accomplish with commercial software.

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

Measurement invariance tests:

Model 1: configural invariance
   chisq        df    pvalue       cfi       tli     rmsea       bic
 115.851    48.000     0.000     0.924     0.900     0.097  7604.094

Model 2: weak invariance (equal loadings):
   chisq        df    pvalue       cfi       tli     rmsea       bic
 124.044    54.000     0.000     0.922     0.909     0.093  7578.043

[Model 1 versus model 2]
delta.chisq     df   p.value  delta.cfi
      8.19       6   0.22436     0.0025

Model 3: strong invariance (equal loadings + equal intercepts):
   chisq        df    pvalue       cfi       tli     rmsea       bic
 164.103    60.000     0.000     0.884     0.878     0.107  7686.588

[Model 1 versus model 3]
delta.chisq     df   p.value  delta.cfi
     48.25      12   0.00000     0.0405

[Model 2 versus model 3]
delta.chisq     df   p.value  delta.cfi
     40.06       6   0.00000     0.0381

Model 4: equal loadings + intercepts + means:
   chisq        df    pvalue       cfi       tli     rmsea       bic
 204.605    63.000     0.000     0.855     0.835     0.122  7709.969

[Model 1 versus model 4]
delta.chisq     df   p.value  delta.cfi
     88.75      15   0.00000     0.0688

[Model 3 versus model 4]
delta.chisq     df   p.value  delta.cfi
     40.50       3   0.00000     0.0283
```

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 an 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:

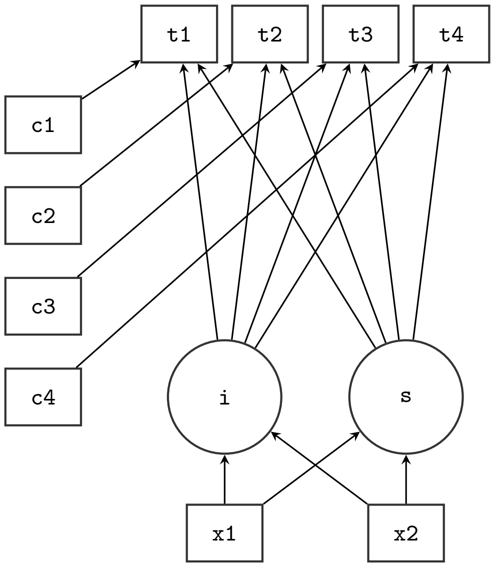
lavaan syntax

```
# 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)

Model converged normally after 37 iterations using ML

  Minimum Function Chi-square                8.069
  Degrees of freedom                             5
  P-value                                   0.1525

                   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

Latent covariances:
  i ~~
    s                 0.618    0.071    8.686    0.000

Latent means/intercepts:
    i                 0.615    0.077    8.007    0.000
    s                 1.006    0.042   24.076    0.000

Intercepts:
    t1                0.000
    t2                0.000
    t3                0.000
    t4                0.000

Latent variances:
    i                 1.932    0.173   11.194    0.000
    s                 0.587    0.052   11.336    0.000

Residual 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
```

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)

Model converged normally after 112 iterations using ML

  Minimum Function Chi-square               4.735
  Degrees of freedom                            4
  P-value                                  0.3156

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

Residual covariances:
  anomia67 ~~
    anomia71          1.623    0.314    5.176    0.000    1.623    0.133
  powerless67 ~~
    powerless71       0.339    0.261    1.298    0.194    0.339    0.035

Latent variances:
    ses               6.798    0.649   10.475    0.000    1.000    1.000

Residual 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
    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 elements 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
- `"WLS"` for weighted least squares (sometimes called ADF estimation)
- `"MLM"` for maximum likelihood estimation with robust standard errors and a Satorra-Bentler scaled test statistic.

If maximum likelihood estimation is used (`"ML"` or `"MLM"`), 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 `mimic.Mplus=FALSE` argument when calling the fitting functions.

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 `na.rm=FALSE` 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 `na.rm=FALSE`). 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 `"MLM"` estimator is used, the standard errors are based on the expected information matrix and corrected using the Satorra-Bentler approach.

8.3 Modification Indices

Modification indices can be requested by adding the `modindices=TRUE` argument in the `summary` call. For example:

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

Model converged normally after 35 iterations using ML

  Minimum Function Chi-square               85.306
  Degrees of freedom                            24
  P-value                                   0.0000

                   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

Latent 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

Latent variances:
    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

Residual 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    7.778    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

Modification Indices:

  Parameter label                 M.I.    E.P.C.    Std.lv   Std.all

        visual=~x4               1.211     0.077     0.069     0.059
        visual=~x5               7.441    -0.210    -0.189    -0.147
        visual=~x6               2.843     0.111     0.100     0.092
        visual=~x7              18.631    -0.422    -0.380    -0.349
        visual=~x8               4.295    -0.210    -0.189    -0.187
        visual=~x9              36.411     0.577     0.519     0.515
       textual=~x1               8.903     0.350     0.347     0.297
       textual=~x2               0.017    -0.011    -0.011    -0.010
       textual=~x3               9.151    -0.272    -0.269    -0.238
       textual=~x7               0.098    -0.021    -0.021    -0.019
       textual=~x8               3.359    -0.121    -0.120    -0.118
       textual=~x9               4.796     0.138     0.137     0.136
         speed=~x1               0.014     0.024     0.015     0.013
         speed=~x2               1.580    -0.198    -0.123    -0.105
         speed=~x3               0.716     0.136     0.084     0.075
         speed=~x4               0.003    -0.005    -0.003    -0.003
         speed=~x5               0.201    -0.044    -0.027    -0.021
         speed=~x6               0.273     0.044     0.027     0.025

            x2~~x1               3.606    -0.184    -0.184    -0.134
            x3~~x1               0.935    -0.139    -0.139    -0.105
            x3~~x2               8.532     0.218     0.218     0.164
            x4~~x1               3.554     0.078     0.078     0.058
            x4~~x2               0.534    -0.034    -0.034    -0.025
            x4~~x3               0.142    -0.016    -0.016    -0.012
            x5~~x1               0.522    -0.033    -0.033    -0.022
            x5~~x2               0.023    -0.008    -0.008    -0.005
```

```
x5~~x3              7.858   -0.130   -0.130   -0.089
x5~~x4              2.534    0.186    0.186    0.124
x6~~x1              0.048    0.009    0.009    0.007
x6~~x2              0.785    0.039    0.039    0.031
x6~~x3              1.855    0.055    0.055    0.044
x6~~x4              6.221   -0.235   -0.235   -0.185
x6~~x5              0.916    0.101    0.101    0.072
x7~~x1              5.420   -0.129   -0.129   -0.102
x7~~x2              8.918   -0.183   -0.183   -0.143
x7~~x3              0.638   -0.044   -0.044   -0.036
x7~~x4              5.920    0.098    0.098    0.078
x7~~x5              1.233   -0.049   -0.049   -0.035
x7~~x6              0.259   -0.020   -0.020   -0.017
x8~~x1              0.634   -0.041   -0.041   -0.035
x8~~x2              0.054   -0.012   -0.012   -0.010
x8~~x3              0.059   -0.012   -0.012   -0.011
x8~~x4              3.805   -0.069   -0.069   -0.059
x8~~x5              0.347    0.023    0.023    0.018
x8~~x6              0.275    0.018    0.018    0.016
x8~~x7             34.145    0.536    0.536    0.488
x9~~x1              7.335    0.138    0.138    0.117
x9~~x2              1.895    0.075    0.075    0.063
x9~~x3              4.126    0.102    0.102    0.089
x9~~x4              0.196   -0.016   -0.016   -0.014
x9~~x5              0.999    0.040    0.040    0.031
x9~~x6              0.097   -0.011   -0.011   -0.010
x9~~x7              5.183   -0.187   -0.187   -0.170
x9~~x8             14.946   -0.423   -0.423   -0.415
```

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

8.4 Extracting information from a fitted model

If you want to peek inside a fitted `semModel` object (the object that is returned by a call to `cfa`, `sem` or `growth`), or you want to 'extract' specific information from a fitted object, you can use the `inspect` function, with a variety of options. By default, calling `inspect` on a fitted `semModel` 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 textual 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  0  0  0  0  0  0  0  0
x2  0  8  0  0  0  0  0  0  0
x3  0  0  9  0  0  0  0  0  0
x4  0  0  0 10  0  0  0  0  0
x5  0  0  0  0 11  0  0  0  0
x6  0  0  0  0  0 12  0  0  0
x7  0  0  0  0  0  0 13  0  0
x8  0  0  0  0  0  0  0 14  0
x9  0  0  0  0  0  0  0  0 15
```

```
$psi
        visual textual speed
visual      16       0     0
textual     17      19     0
speed       18      20    21
```

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

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

```
$lambda
   visual textual speed
x1      1       0     0
x2      1       0     0
x3      1       0     0
x4      0       1     0
x5      0       1     0
x6      0       1     0
x7      0       0     1
x8      0       0     1
x9      0       0     1

$theta
          x1        x2        x3        x4        x5        x6        x7
x1 0.6814489 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
x2 0.0000000 0.6931949 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
x3 0.0000000 0.0000000 0.6395572 0.0000000 0.0000000 0.0000000 0.0000000
x4 0.0000000 0.0000000 0.0000000 0.6775834 0.0000000 0.0000000 0.0000000
x5 0.0000000 0.0000000 0.0000000 0.0000000 0.8326592 0.0000000 0.0000000
x6 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.6001731 0.0000000
x7 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.5935416
x8 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
x9 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
          x8        x9
x1 0.0000000 0.0000000
x2 0.0000000 0.0000000
x3 0.0000000 0.0000000
x4 0.0000000 0.0000000
x5 0.0000000 0.0000000
x6 0.0000000 0.0000000
x7 0.0000000 0.0000000
x8 0.5126947 0.0000000
x9 0.0000000 0.5091936

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

To extract a single fit index, say, `CFI` from a fitted model, you can use

```
> fit.indices <- inspect(fit, what = "fit")
> fit.indices["cfi"]
```

```
      cfi
0.9305597
```

The `inspect` function always returns the information as a vector or a list, so that the information can be captured for further processing. For more inspect options, see the help page for the `semModel` class which you can find by typing the following:

```
> class?semModel
```

Other 'extractor' functions are `coef`, `fitted.values`, `residuals` and its alias `resid`. We have already seen that `coef` returns the values of the free parameters (as a vector). The `fitted.values` function returns a list containing the *implied* covariance matrix and mean vector. The `residuals` and `resid` functions return a list containing the raw difference between the observed and implied covariance matrix and mean vector.

9 Known Issues

There are a number of known issues with the current beta version of the lavaan package. If you can help us out with any of these, we would be very grateful.

```
~= lavaan version 0.3-1 (march 2010) ~=

New issues:

 * multigroup WLS in Mplus: if each group has its own set of free parameters,
   the X2 of the first group is not same, as if you would fit the model
   separately using that group alone; strangely, the X2 of the second group
   is ok...   Is this a bug in Mplus? (confirmed in 4.1 and 5.21)

 * In Mplus: SRMR index is (sometimes much) smaller if
   'information=observed' option is used? (when?)

Old Issues:

 * Satorra-Bentler correction if mimic.Mplus=TRUE:
  - the standard errors and scaled chi-square test statistic are not
    the same as in Mplus (4.1 and 5.21); Mplus must be doing something
    different here, but what?
    Note: if mimic.Mplus=FALSE, the results are the same as in EQS.

 * Modification indices:
  - EPC's for equality constraints are not the same as in Mplus
    (for example, the NU EPC's in Mplus example 5.9); I need a reference
    containing the proper formulas!
  - the MI's for multiple group models with equality constraints are not
    identical to Mplus (but mostly close); again, a reference containing
    some formulas would be useful

 * SRMR value is slightly off in multiple-groups analyses with equality
   constraints (for example in ex5.14 with equality.constraints=c("loadings",
   "intercepts","means", "residuals"), the SRMR in semplus is 0.133, while
   Mplus reports 0.142)
```

10 New features and changes compared to 'semplus 0.9-10' (december 2009)

```
~= lavaan version 0.3-1 (march 2010) ~=

User-visible changes (compared to semplus 0.9-1 december 2009 version):

 * the name of the package has changed to `lavaan' (for latent variable
   analysis)

 * the 'ML.N' option is replaced by a 'mimic.Mplus' option; if TRUE, an attempt
   is made to mimic Mplus results as much as possible; if FALSE, results
   are more similar to EQS/LISREL results

 * if do.fit=FALSE, a full summary (including standard errors) is now available

 * if a correlation matrix is supplied (instead of a covariance matrix),
   only a (big) warning is now spit out (instead of an error and stopping)


New features (compared to semplus 0.9-1 december 2009 version):

 * the model syntax can now be specified as a string literal enclosed in
   single quotes, allowing for arbitrary blank lines and comments; this
   will be the preferred way to provide a model syntax; the specfiy.Model()
   function will be deprecated in the next release
```

```
 * multiple values are now accepted within pre-multiplication commands when
   analyzing multiple groups; for example: "start( c(0.5, 0.8) )" will give
   different starting values for each of the two groups

 * in a multiple group analysis, the sample moments can be provided using
   a list

 * 'automatic' naming of free parameters is now group-dependent

 * using NA*x in a formula forces the corresponding parameter to be free

 * a new modifier 'label' can now be used to specify custom labels
   for parameters, eg. f =~ x1 + x2 + label("mylabel")*x3

 * added 'information' argument to the cfa/sem/growth functions, so that
   the user can choose between the 'expected' or the 'observed' information
   matrix to be used when calculating standard errors; the observed information
   matrix is currently computed using a numerical approximation (not
   analytically); this produces accurate results, but is fairly slow

 * if na.rm=FALSE and estimator="ML", full information ML (FIML) is used
   to estimate the parameters; a new 'missing' slot is provided in the
   Sample slot of a fitted object, containing information about the
   missing patterns and their sufficient statistics


Bug fixes (compared to semplus 0.9-1 december 2009 version):

 * the std.lv=TRUE argument is now working again

 * fixed two bugs in the specify.Model() function:
    - the comment character '#' can now appear in the first column
    - avoid confusion with lv defs containing equal("f1=~x1") statements
   but the specify.Model() function is deprecated and will be removed
   in the next release

 * fixed WLS + meanstructure issue: function value of 1st iteration is now
   identical to Mplus 4.1

 * symmetric matrices returned by inpect() are fully symmetric (not only
   showing the lower half)

 * Satorra-Bentler correction: if mimic.Mplus=FALSE, the scaled chi-square
   statistic and standard errors should now be identical to EQS in all cases

 * corrected the nomenclature of H0 and H1 in the output of
   'summary(fit, fit.measures=TRUE)'

 * Residual covariances in 'summary' output does not show the user-fixed
   residual variances anymore
```

11 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 ~ x1 + x2 + x3
                  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 ~ x1 + x2 + x3
                 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 ~ 1
                 y1b ~ equal("y1a~1") * 1
                 y1c ~ equal("y1a~1") * 1
                 y2a ~ 1
                 y2b ~ equal("y2a~1") * 1
                 y2c ~ equal("y2a~1") * 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 + equal(c("","1.f1=~y2"))*y2 + y3
                  f2 =~ y4 + equal(c("","1.f2=~y5"))*y5
                           + equal(c("","1.f2=~y6"))*y6
                  f1 ~ x1 + x2 + x3
                  f2 ~ x1 + x2 + x3 '

fit <- cfa(model.ex5.14, data=Data, group="g", meanstructure=FALSE)
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.ex5.14 <- ' f1 =~ y1 + equal(c("","1.f1=~y2"))*y2 + y3
                  f2 =~ y4 + equal(c("","1.f2=~y5"))*y5
                           + equal(c("","1.f2=~y6"))*y6
                  f1 ~ x1 + x2 + x3
                  f2 ~ x1 + x2 + x3
```

```
                  f1 ~ c(0,NA)*1
                  f2 ~ c(0,NA)*1
                  y1 ~ equal(c("","1.y1~1"))*1
                  y2 ~ equal(c("","1.y2~1"))*1
                  y3 ~ 1
                  y4 ~ equal(c("","1.y4~1"))*1
                  y5 ~ equal(c("","1.y5~1"))*1
                  y6 ~ equal(c("","1.y6~1"))*1 '

fit <- cfa(model.ex5.14, data=Data, group="g", meanstructure=TRUE)
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 ~ x1 + x2
  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)
```