```
[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    0.840     0.122 7709.969

[Model 1 versus model 4]
  delta.chisq      delta.df delta.p.value     delta.cfi
       88.754        15.000         0.000         0.083

[Model 3 versus model 4]
  delta.chisq      delta.df delta.p.value     delta.cfi
       40.502         3.000         0.000         0.042
```

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

9 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 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:

```
# 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 (0.5-13) converged normally after  44 iterations

  Number of observations                           400

  Estimator                                         ML
  Minimum Function Test Statistic                8.069
  Degrees of freedom                                 5
  P-value (Chi-square)                           0.152

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard
```

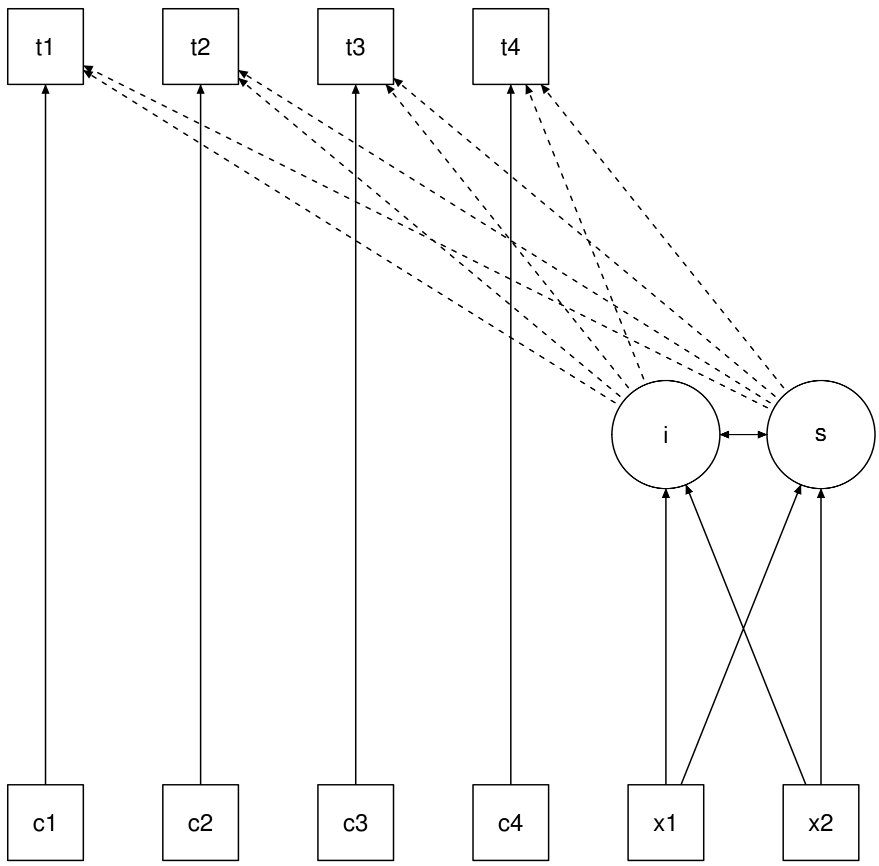
```
                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  i =~
    t1                1.000
    t2                1.000
    t3                1.000
    t4                1.000
  s =~
    t1                0.000
    t2                1.000
    t3                2.000
    t4                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
    t2                0.676    0.061
    t3                0.635    0.072
    t4                0.508    0.124
    i                 1.932    0.173
    s                 0.587    0.052
```

Technically, the `growth()` function is almost identical to the `sem()` function. But a mean structure 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 `c` that influences the outcome measure at the four time points has been added to the model. A graphical representation of this model is presented below.



The corresponding syntax is the following:

```
# 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:

```
# 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)
```

10 Using categorical variables

Binary, ordinal and nominal variables are considered categorical (not continuous). It makes a big difference if these categorical variables are exogenous (independent) or endogenous (dependent) in the model.

Exogenous categorical variables If you have a binary exogenous covariate (say, gender), all you need to do is to recode it as a dummy (0/1) variable. Just like you would do in a classic regression model. If you have an exogenous ordinal variable, you can use a coding scheme reflecting the order (say, 1,2,3,...) and treat it as any other (numeric) covariate. If you have a nominal categorical variable with $K > 2$ levels, you need to replace it by a set of $K - 1$ dummy variables, again, just like you would do in classical regression.

Endogenous categorical variables The lavaan 0.5 series can deal with binary and ordinal (but not nominal) endogenous variables. Only the three-stage WLS approach is currently supported, including some 'robust' variants. To use binary/ordinal data, you have two choices:

1. declare them as 'ordered' (using the `ordered` function, which is part of base R) in your data.frame before you run the analysis; for example, if you need to declare four variables (say, `item1`, `item2`, `item3`, `item4`) as ordinal in your data.frame (called `Data`), you can use something like:

```
Data[,c("item1",
        "item2",
        "item3",
        "item4")] <-
    lapply(Data[,c("item1",
                   "item2",
                   "item3",
                   "item4")], ordered)
```

2. use the `ordered` argument when using one of the fitting functions (cfa/sem/growth/lavaan), for example, if you have four binary or ordinal variables (say, `item1`, `item2`, `item3`, `item4`), you can use:

```
fit <- cfa(myModel, data = myData,
           ordered=c("item1","item2",
                     "item3","item4"))
```

In both cases, lavaan will automatically switch to the `WLSMV` estimator: it will use diagonally weighted least squares (`DWLS`) to estimate the model parameters, but it will use the full weight matrix to compute robust standard errors, and a mean- and variance-adjusted test stastistic.

11 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 wish to add a mean structure, you need to provide a mean vector too. Importantly, if only sample statistics are provided, you must 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. First, we read in the lower half of the covariance matrix (including the diagonal):

```
lower <- '
 11.834
  6.947   9.364
```