Guide No.
KUANT 021.3

KUant Guides

SEM with Lavaan 0.5-15

A guide introducing the R package *lavaan* for structural equation modeling.

Miller, P., Jorgensen, T. D., & Pornprasertmanit, S. (2013)

www.crmda.ku.edu

Introduction

R is an open source software environment for statistical computing, and one of its primary benefits is the vast number of available packages developed for advanced statistics and data analysis. Within the R programming environment there are several packages designed for latent variable analysis (including `OpenMx`, `mirt`, `lava`, and `sem`), but in this guide we will be looking at a relatively new and user-friendly package called *lavaan*, throughout which we assume a basic knowledge of R. KUant Guide #20 is devoted specifically to R beginners.

We illustrate the most salient features of *lavaan* in this guide. The developer of *lavaan* also provides a helpful, readable user's guide and more technical official software documentation (see References).

First Steps

If you have not yet installed R, follow this link: http://cran.r-project.org/

Using installation defaults should generally be sufficient for most users.

To install *lavaan*, open R and type the following command at the prompt:

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

Select a CRAN mirror when prompted (any in the US should be sufficient). Load the package by typing the following:

```
> library(lavaan)
```

You should see a warning telling you that it is beta software, which you should heed. It is reliable, but still beta.

Getting Help with *lavaan*

- Examples are provided on the web site: http://lavaan.ugent.be/
- Examples are also available in the creator's paper about *lavaan*, published in the *Journal of Statistical Software* (please cite this paper if you publish results analyzed using lavaan):
  http://www.jstatsoft.org/v48/i02
- Find help files in R by typing on the command line: `help(package = lavaan)`

A Simple Example

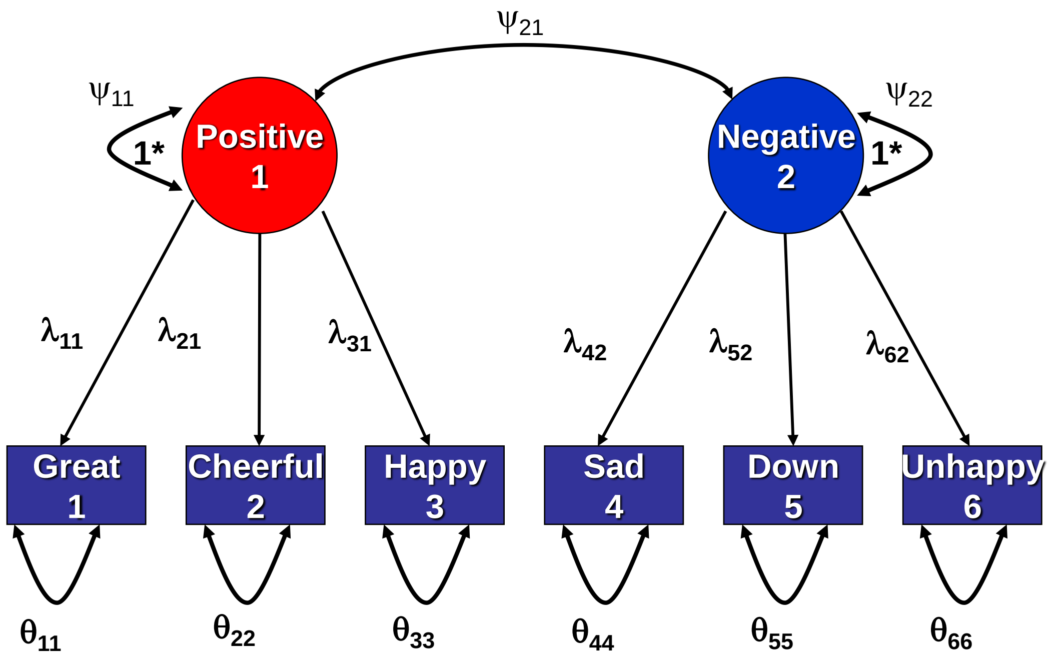
First, we need to read the data into R. In this case, the data file `("posneg.dat")` is a full data file with 6 variables.

Assuming that the data file is in the same working directory as R, we can use the following commands to store the data matrix in the object `"dat"` as a data frame, name the variables, and then verify the results.

```
> myData <- read.table("posneg.dat", header = FALSE)
> names(dat) <- c("great", "cheerful", "happy", "sad", "down",
"unhappy")
> head(myData)

   great cheerful  happy    sad down unhappy
1 3.5000   4.0000 4.0000 4.0000  4.0       4
2 2.5000   3.1667 3.0000 3.2123  2.0       3
3 1.8333   2.0000 1.5000 3.0000  3.0       2
4 2.7714   3.0602 2.3639 3.1337  4.0       3
5 3.1667   3.3333 2.8333 3.5000  4.0       4
6 2.3333   2.8333 2.3333 3.0000  2.5       3
```

As in other syntax guides, we will use the following model as an example. Positive and negative affect are each latent constructs and are each indicated by three latent variables.



We make the *lavaan* model statement and store it in the vector `PosModel`. Note that the variable names in the model statement correspond to the variable names in our data frame.

```
> PosModel <- "Positive =~ great + cheerful + happy
              Negative =~ sad + down + unhappy   "
```

The `"=~"` means “is loaded by.” To find the fit of this CFA model, call the *lavaan* function `cfa()` and store the results in a vector. In this example, we standardize the latent variances to equal 1 in order to identify the model (as opposed to the default, which is to fix the first loading to equal 1). This is specified in the `cfa()` argument `"std.lv = TRUE"`.

```
> Results <- cfa(PosModel, data = myData, std.lv = TRUE)
```

To examine the results, we will use the function `summary()`

```
> summary(Results)

lavaan (0.5-9) converged normally after  43 iterations

  Number of observations                           380

  Estimator                                         ML
  Minimum Function Chi-square                   10.708
  Degrees of freedom                                 8
  P-value                                        0.219

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  Positive =~
    great             0.465    0.021   22.474    0.000
    cheerful          0.492    0.021   23.768    0.000
    happy             0.498    0.022   22.870    0.000
  Negative =~
    sad               0.529    0.039   13.477    0.000
    down              0.579    0.044   13.314    0.000
    unhappy           0.626    0.039   16.232    0.000

Covariances:
  Positive ~~
    Negative          0.507    0.047   10.829    0.000

Variances:
    great             0.049    0.005
    cheerful          0.036    0.005
    happy             0.050    0.005
    sad               0.311    0.030
    down                 0.388    0.037
    unhappy              0.200    0.030
    Positive             1.000
    Negative             1.000
```

To include fit measures and the standardized solution, use the `fit` and `standardized` arguments in the `summary` function:

```
> summary(Results, fit = TRUE, standardized = TRUE)

lavaan (0.5-9) converged normally after  43 iterations

  Number of observations                           380

  Estimator                                         ML
  Minimum Function Chi-square                   10.708
  Degrees of freedom                                 8
  P-value                                        0.219

Chi-square test baseline model:

  Minimum Function Chi-square                 1416.263
  Degrees of freedom                                15
  P-value                                        0.000

Full model versus baseline model:

  Comparative Fit Index (CFI)                    0.998
  Tucker-Lewis Index (TLI)                       0.996

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)              -1545.377
  Loglikelihood unrestricted model (H1)      -1540.023

  Number of free parameters                         13
  Akaike (AIC)                                3116.754
  Bayesian (BIC)                              3167.976
  Sample-size adjusted Bayesian (BIC)         3126.730

Root Mean Square Error of Approximation:

  RMSEA                                          0.030
  90 Percent Confidence Interval          0.000  0.071
  P-value RMSEA <= 0.05                          0.745

Standardized Root Mean Square Residual:

  SRMR                                           0.018
(continued...)
```

In summary, the entire model can be run from raw data in the following four commands:

```
> myData <- read.table("posneg.dat", header = TRUE)
> PosModel <- 'Positive =~ X1 + X2 + X3
               Negative =~ X7 + X8 + X9'
> Results <- cfa(PosModel, data = myData, std.lv = TRUE)
> summary(Results)
```

Lavaan Model Syntax

Main Operators

With the above principles of the R programming language established, using *lavaan* is straightforward. Structural models are specified with regression-like syntax, like normal R formulas:

```
      Y1 ~ X1 + X2 + F1 + F2
```

The `"~"` operator represents "is regressed on." The left hand side represents dependent variable. The right hand side represents independent variables.

Latent variables are defined using the `"=~"` operator to represent "is loaded by":

```
     F1 =~  X4 + X5 + X6
     F2 =~  X7 + X8 + X9
```

The left hand side represents latent variable. The right hand side represents manifest variables.

To express the correlation between two residual variances or two factors, use the `"~~"` operator to represent "is correlated with":

```
     X4 ~~ X5
```

The order of variables does not matter for this operator. Users may use `"X4 ~~ X5"` instead.

Intercepts/means are specified by regressing on 1:

```
     X1 ~ 1
     F1 ~ 1
```

To build the model in its entirety, simply put enclose all of these statements in quotes `"  "` and assign it to an object.

Special parameters (such as indirect effects that are products of regression parameters) are defined by this operator: `":="` (is defined by). For example, the indirect effect of *X* to *Y* via *M* is the target parameter. First, labels should be imposed on the regression coefficients from factor *X* to factor *M* and from factor *M* to factor *Y*. A label is specified by putting an arbitrary name with an asterisk in front of an independent variable in a regression specification:

```
      M ~ mx*X
      Y ~ ym*M
```

The labels of the regression coefficients from factor *X* to factor *M* and from factor *M* to factor *Y* are `mx` and `ym`, respectively. Then, the indirect of *X* on *Y* via mediator *M* can be specified `"ymx"` like this:

```
      ymx := ym*mx
```

Parameters can be constrained using relational operators: `"=="` (is equal to), `"<"` (is less than), and `">"` (is greater than).

Free or Fixed Parameters

lavaan provides some defaults option that, sometimes, users may be not aware of it. For example, the first manifest variable loading is fixed as 1 by default. Therefore, users need to know how to fix or free parameters. To fix a parameter, users simply put a number with an asterisk in front of a target variable (use texts for labels, but numbers for actual parameter values):

```
> PosModel <- 'Positive =~ 1*V1 + V2 + V3
               Negative =~ 1*V4 + V5 + V6'
```

To free a parameter, users simply put an `NA` with an asterisk in front of a target variable:

```
> PosModel <- 'Positive =~ 1*V1 + NA*V2 + NA*V3
               Negative =~ 1*V4 + NA*V5 + NA*V6    '
```

For example, a fixed-factor-approach syntax can be specified:

```
> PosModel <- 'Positive =~ NA*V1 + NA*V2 + NA*V3
               Negative =~ NA*V4 + NA*V5 + NA*V6
               Positive ~~ 1*Positive
               Negative ~~ 1*Negative
               V1 + V2 + V3 + V4 + V5 + V6 ~ NA*1
               Positive + Negative ~ 0*1      '
```

Note that `V1 + V2 + V3 + V4 + V5 + V6 ~ NA*1` is a shortcut for freeing six measurement intercepts at once. Users may specify the intercepts one-by-one.

lavaan actually provides some functions to automatically set the default as a fixed-factor approach: the `"std.lv = TRUE"` argument of the `cfa` function as shown above. The syntax approach is the most flexible approach to specify models.

Equality Constraints

Equality constraints can be done in many ways. The easiest way is to specify a label on each parameter estimates. The label can be attached to a target parameter by adding a label with an asterisk in front of a variable name of target parameter in the syntax:

```
> PosModel <- 'Positive =~ V1 + Lambda1*V2 + Lambda1*V3
               Negative =~ V4 + V5 + V6 '
```

The variables with the same labels represent equally constrained parameters. In this case, the factor loading from the "Positive" factor on Variables 2 and 3 are equally constrained. Users may fix/free and label parameters simultaneously by specify the target parameter one more time:

```
> PosModel <- 'Positive =~ NA*V1 + eq1*V1 + V2 + eq1*V3
               Negative =~ V4 + V5 + V6 '
```

The factor loading from the "Positive" factor on Variable 1 is free and is equal to the factor loading from the "Positive" factor on Variable 3.

Multiple Groups

In multiple-group models, researchers may use the concatenate function, `c()`, to specify fixed/free parameters or labels in different groups:

```
> PosModel <- 'Positive =~ V1 + c(eq1, eq1)*V2 + c(eq1, eq1)*V3
        Negative =~ c(1, 1)*V4 + c(0.8, 1.5)*V5 + c(eq2, eq2)*V6 '
```

If the labels are the same across groups, the parameters are constrained to be equal across groups. In this example, factor loadings are fixed or constrained as follows:

1. Factor loadings from the "Positive" factor on Variable 2 and 3 in both groups are all equally constrained.
2. Factor loadings from the "Negative" factor on Variable 4 are fixed as 1 in both groups.
3. Factor loadings from the "Negative" factor on Variable 5 in Groups 1 and 2 are fixed as 0.8 and 1.5, respectively.
4. Factor loadings from the "Negative" factor on Variable 6 are equal across groups.

Categorical Indicators

Constructs defined by indicators which are measured with ordinal categories (e.g., Likert-scale responses) can be estimated appropriately if the variables are recognized as `ordered` by R:

```
> myData <-
    read.table("http://www.statmodel.com/usersguide/chap5/ex5.16.dat")
  names(myData) <- c("u1","u2","u3","u4","u5","u6","x1","x2","x3","g")
  myData$u1 <- ordered(myData$u1)
  myData$u2 <- ordered(myData$u2)
  myData$u3 <- ordered(myData$u3)
  myData$u4 <- ordered(myData$u4)
  myData$u5 <- ordered(myData$u5)
  myData$u6 <- ordered(myData$u6)
```

Ordinal factors are defined by indicators using the same syntax as for continuous indicators. Ordinal factors do not have estimated means/intercepts, but rather thresholds between categories, which are defined with the vertical bar (or “pike”, `|` , typically the same key as Shift+backslash, `\`). Thresholds are automatically labeled sequentially (e.g., for a 4-category variable `"v1"`, thresholds would be `"v1 | t1"`, `"v1 | t2"`, and `"v1 | t3"`). Let’s start with running an example with single group. `u1` through `u6` are binary indicators. If all thresholds in a variable are freely estimated, the latent scale for this variable must be fixed to 1 in both groups in order for the model to be identified, using a special operator for ordinal variables: `"~*~"`. The fixed latent scale variances as 1 are implemented in *lavaan* by default.

```
> model <- ' f1 =~ u1 + u2 + u3
             f2 =~ u4 + u5 + u6
             # mimic model with exogenous predictors
             f1 + f2 ~ x1 + x2 + x3
             # estimating thresholds for all items
             u1 | t1
             u2 | t1
             u3 | t1
             u4 | t1
             u5 | t1
             u6 | t1
             # fix the latent scale of u3 as 1, which is a default
             u3 ~*~ 1*u3 '
> fit <- cfa(model, data = myData, group = "g", group.equal =
             c("loadings", "thresholds"))
> summary(fit)
```

Note that any text to the right of pound sign, `#`, is ignored in lavaan syntax, just as it is in any other R syntax. Users can use it to make a comment within a lavaan syntax object.

The model syntax below (adapted from an example in Ch 5 of the *Mplus* User Guide) is an example of multiple-group model with categorical variables. The loadings and thresholds are both constrained to equality across groups “`g`”, allowing the scale of the ordinal indicators to be freely estimated in the second group. Only the loading and threshold of the third indicator `u3` are freely estimated in both groups (`u3|t1`) using different labels for the thresholds (`u3 | c(u3a, u3b)*t1`) and loadings (`f1 =~ c(Lambda3a, Lambda3b)*u3`). Because the threshold is freely estimated in each group, the scale for this variable must be fixed to 1 in both groups in order for the model to be identified, using a special operator for ordinal variables: `"~*~"`.

```
> modelg <- ' f1 =~ u1 + u2 + c(Lambda3a, Lambda3b)*u3
             f2 =~ u4 + u5 + u6
             # mimic
             f1 + f2 ~ x1 + x2 + x3
             # equal thresholds, but free u3|1 in second group
             u3 | c(u3a, u3b)*t1
             # fix scale of u3* to 1 in second group
             u3 ~*~ c(1, 1)*u3     '
> fitg <- cfa(modelg, data = myData, group = "g", group.equal =
             c("loadings", "thresholds"))
> summary(fitg)
```

Fitting a Model

The `cfa()` function has been introduced to fit a confirmatory factor analysis. There are two more useful functions to fit models: `growth()` for latent curve model and `sem()` for structural equation modeling. Here are some useful arguments of these functions:

- `model:` The *lavaan* syntax
- `data:` The target data to be used
- `sample.mean:` The vector of sample means (if the data argument is not used)
- `sample.cov:` The covariance matrix (if the data argument is not used)
- `sample.nobs:` The number of observations (if the data argument is not used)
- `group` The variable name of grouping variable
- `group.equal:` The set of parameters that are equally constrained. For example,
  - ❖ `"loadings"` For factor loadings
  - ❖ `"intercepts"` For measurement intercepts
  - ❖ `"residuals"` and `"residual.covariances"` For measurement residual variances or covariances, respectively
  - ❖ `"thresholds"` For thresholds of categorical indicators
  - ❖ `"regressions"` For latent regressions
  - ❖ `"means"` For latent variable means
  - ❖ `"lv.variances"` For latent variable variances (or residual variances)
  - ❖ `"lv.covariances"` For latent variable covariances (or correlations if all latent variable variances are 1)
- `group.partial:` When sets of parameters are constrained to be equal across groups, this is a list of parameters that should not be constrained to equality. For example if `group.equal = "loadings"` is used to constrain all 5 loadings across groups to equality, but you want the 4th and 5th loadings to be freely estimated across groups, specify `group.partial = c("f1=~x4", "f1=~x5")`
- `estimator:` The method of estimation. `"ML"` is the default, which is a maximum likelihood. `"MLR"` or `"MLM"` is the scaled statistics for nonnormal data adjustment. `"WLSMV"` is used when there are categorical indicators. Ex: `estimator = "MLR"`
- `se, test, bootstrap:` Along with `estimator`, these arguments can be used for more robust tests of hypotheses in the presence of nonnormal data. See the help page of the `cfa` function for details.
- `std.lv:` If `TRUE`, the fixed-factor method is used for scale identification. If `FALSE`, the manifest variable method is used for scale identification. Ex: `std.lv = FALSE`
- `ordered:` List the names of categorical variables: Ex: `ordered = c("V1", "V2")`
- `missing:` The method to handle missing data. **IMPORTANTLY**, the *lavaan* default is listwise deletion. The `"fiml"` estimator should be used for full-information maximum likelihood. Ex: `missing = "fiml"`
  - ❖ *Note*: Using lavaan with multiple-imputed data is facilitated using the `runMI()` function, available in the R package `semTools`.
- `control:` Pass options to the `nlminb()` optimizer. Ex: set convergence criterion or the number of iterations: `control = list(iter.max = 10) # see ?cfa for details`

Extract Outputs

The output is saved in a *lavaan* object. In the previous example, the `Results` is a *lavaan* object. As shown above, the `summary()` function can be used to summarize the results. Here are the useful options of the `summary()` function:

- `standardized`: If `TRUE`, standardized estimates are provided. Ex: `summary(Results, standardized = TRUE)`
- `fit`: If `TRUE`, fit indices are provided. Ex: `summary(Results, fit = TRUE)`
- `modindices`: If `TRUE`, modification indices are provided. Ex: `summary(Results, modindices = TRUE)`
- `rsquare`: If `TRUE`, R-squared are provided. Ex: `summary(Results, fit = TRUE, rsquare = TRUE)`

Other functions that can be used for the *lavaan* object:

- `inspect`: To inspect the *lavaan* object. Ex: `inspect(Results, "coef")`. This code is used to inspect parameter estimates. Here are other useful parts to be inspected:
  - `"free"` Free parameters
  - `"sampstat"` The statistics of observed variables
  - `"coef"` Parameter estimates
  - `"se"` Standard errors
  - `"std"` Standardized coefficients
  - `"r2"` R-squared
  - `"mi"` Modification indices
  - `"fit"` Fit indices
- `predict`: To provide the factor scores. Ex: `predict(Results)`
- `residuals`: To provide differences between observed sample covariances and those implied by the model. Ex: `resid(Results)`
- `anova`: To provide a model comparison statistics between two nested models. Ex: `anova(lessConstrainedModel, moreConstrainedModel)`
- `parameterEstimates`: Another way to provide the summary of the model. The fraction missing information is provided using this function, as are *p* values for hypothesis tests using bootstrapped *SE*s or for special parameters defined using the `":="` operator. Ex: `parameterEstimates(Results)`

Notable Limitations

lavaan is still beta software, and does not yet support everything other dedicated software packages for SEM do. This includes support for Bayesian analyses, discrete latent variables (e.g., for latent profile or latent class analyses), item-response (IRT) models, and hierarchical/multilevel data sets (although some options are available using the package `lavaan.survey`*).*

References

R Development Core Team. (2012). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org

Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software, 48*(2), 1–36. Retrieved from http://www.jstatsoft.org/v48/i02/

User's Guide:	http://users.ugent.be/~yrosseel/lavaan/lavaanIntroduction.pdf
Official Reference:	http://cran.r-project.org/web/packages/lavaan/lavaan.pdf