

lavaan: a brief user's guide

Yves Rosseel

Department of Data Analysis
Ghent University – Belgium

Utrecht – April 24, 2012

Contents

1	lavaan: a brief user's guide	4
1.1	Model syntax: specifying models	4
1.2	Fitting functions: estimating models	10
1.3	Extractor functions: inspecting fitted models	21
1.4	Other functions	22
1.5	Meanstructures	23
1.6	Multiple groups	27
1.7	Missing data in lavaan	31
1.8	Standard errors	32
1.9	Test statistics	33
1.10	BootstrapLavaan	34
1.11	Constraints and defined parameters	35
1.12	Using a covariance matrix as input	37
2	Some technical details	39
2.1	Default estimator: ML	39
2.2	Estimator MLM	41

2.3 Estimator MLR	43
-----------------------------	----

1 lavaan: a brief user's guide

1.1 Model syntax: specifying models

The four main formula types, and other operators

formula type	operator	mnemonic
latent variable	= ~	is manifested by
regression	~	is regressed on
(residual) (co)variance	~ ~	is correlated with
intercept	~ 1	intercept
defined parameter	:=	is defined as
equality constraint	==	is equal to
inequality constraint	<	is smaller than
inequality constraint	>	is larger than

A typical model syntax

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

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

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

                # intercepts
                y1 ~ 1
                f1 ~ 1
                ,
```

Fixing parameters, and overriding auto-fixed parameters

```
HS.model.bis <- ' visual  =~ NA*x1 + x2 + x3
                  textual =~ NA*x4 + x5 + x6
                  speed   =~ NA*x7 + x8 + x9
                  visual  ~~ 1*visual
                  textual  ~~ 1*textual
                  speed    ~~ 1*speed
                ,
```

- pre-multiplying a model parameter with a numeric value will keep the parameter fixed to that value
- pre-multiplying a model parameter with 'NA' will force the parameter to be free
- for this piece of code: using the `std.lv=TRUE` argument has the same effect

Labels and simple equality constraints

```
model.equal <- '  
  # measurement model  
  ind60 =~ x1 + x2 + x3  
  dem60 =~ y1 + d1*y2 + d2*y3 + d3*y4  
  dem65 =~ y5 + d1*y6 + d2*y7 + d3*y8  
  
  # regressions  
  dem60 ~ ind60  
  dem65 ~ ind60 + dem60  
  
  # residual covariances  
  y1 ~~ y5  
  y2 ~~ y4 + y6  
  y3 ~~ y7  
  y4 ~~ y8  
  y6 ~~ y8  
,
```

- pre-multiplying model parameters with a string gives the model parameter a custom 'label'
- model parameters with the same label are considered to be equal

Defined parameters and mediation analysis

```
X <- rnorm(100); M <- 0.5*X + rnorm(100); Y <- 0.7*M + rnorm(100)
Data <- data.frame(X = X, Y = Y, M = M)
```

```
model <- ' # direct effect
          Y ~ c*X
          # mediator
          M ~ a*X
          Y ~ b*M
          # indirect effect (a*b)
          ab := a*b
          # total effect
          total := c + (a*b)
          ,

fit <- sem(model, data=Data)
```

- the “:=” operator defines a new parameter, as a function of existing (free) parameters, but referring to their labels
- by default, the delta rule is used to compute standard errors for these defined parameters; bootstrapping may be a better option

Linear and nonlinear equality and inequality constraints

```
Data <- data.frame(y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100),
                  x3 = rnorm(100))

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

fit <- sem(model.constr, data=Data)
```

- simple regression model, but with (nonlinear) constraints imposed on the regression coefficients
- can be used to force variances to be strictly positive
- can be used for testing interaction effects among latent variables
- for simple equality constraints (e.g. $b1 == b2$), it is much more efficient to simply provide the same label

1.2 Fitting functions: estimating models

User-friendly fitting functions

- `cfa()` for confirmatory factor analysis
- `sem()` for path analysis and SEM
- `growth()` for growth curve modeling

Arguments of the `cfa()` and `sem()` fitting functions

```
sem(model = NULL, meanstructure = "default", fixed.x = "default",
orthogonal = FALSE, std.lv = FALSE, data = NULL, std.ov = FALSE,
missing = "default", sample.cov = NULL, sample.mean = NULL,
sample.nobs = NULL, group = NULL, group.equal = "",
group.partial = "", constraints = '', estimator = "default",
likelihood = "default", information = "default", se = "default",
test = "default", bootstrap = 1000L,
mimic = "default", representation = "default",
do.fit = TRUE, control = list(), start = "default",
verbose = FALSE, warn = TRUE, debug = FALSE)
```

<code>model</code>	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See <code>model.syntax</code> for more information. Alternatively, a parameter list (eg. the output of the <code>lavaanify()</code> function) is also accepted.
<code>meanstructure</code>	If <code>TRUE</code> , the means of the observed variables enter the model. If <code>"default"</code> , the value is set based on the user-specified model, and/or the values of other arguments.
<code>fixed.x</code>	If <code>TRUE</code> , the exogenous 'x' covariates are considered fixed variables and the means, variances and covariances of these variables are fixed to their sample values. If <code>FALSE</code> , they are considered random, and the means, variances and covariances are free parameters. If <code>"default"</code> , the value is set depending on the <code>mimic</code> option.
<code>orthogonal</code>	If <code>TRUE</code> , the exogenous latent variables are assumed to be uncorrelated.
<code>orthogonal</code>	If <code>TRUE</code> , the exogenous latent variables are assumed to be uncorrelated.
<code>std.lv</code>	If <code>TRUE</code> , the metric of each latent variable is determined by fixing their variances to 1.0. If <code>FALSE</code> , the metric of each latent variable is determined by fixing the factor loading of the first indicator to 1.0.
<code>data</code>	An optional data frame containing the observed variables used in the model.
<code>std.ov</code>	If <code>TRUE</code> , all observed variables are standardized before entering the analysis.
<code>missing</code>	If <code>"listwise"</code> , cases with missing values are removed listwise from the data frame before analysis. If <code>"direct"</code> or <code>"ml"</code> or <code>"fiml"</code> and the estimator

is maximum likelihood, Full Information Maximum Likelihood (FIML) estimation is used using all available data in the data frame. This is only valid if the data are missing completely at random (MCAR) or missing at random (MAR). If "default", the value is set depending on the estimator and the mimic option.

<code>sample.cov</code>	Numeric matrix. A sample variance-covariance matrix. The rownames must contain the observed variable names. For a multiple group analysis, a list with a variance-covariance matrix for each group.
<code>sample.mean</code>	A sample mean vector. For a multiple group analysis, a list with a mean vector for each group.
<code>sample.nobs</code>	Number of observations if the full data frame is missing and only sample moments are given. For a multiple group analysis, a list or a vector with the number of observations for each group.
<code>group</code>	A variable name in the data frame defining the groups in a multiple group analysis.
<code>group.equal</code>	A vector of character strings. Only used in a multiple group analysis. Can be one or more of the following: "loadings", "intercepts", "means", "regressions", "residuals", "residual.covariances", "lv.variances" or "lv.covariances", specifying the pattern of equality constraints across multiple groups.
<code>group.partial</code>	A vector of character strings containing the labels of the parameters which

	should be free in all groups (thereby overriding the <code>group.equal</code> argument for some specific parameters).
<code>constraints</code>	Additional (in)equality constraints not yet included in the model syntax. See <code>model.syntax</code> for more information.
<code>estimator</code>	The estimator to be used. Can be one of the following: "ML" for maximum likelihood, "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, "MLF" for maximum likelihood estimation with standard errors based on first-order derivatives and a conventional test statistic, "MLR" for maximum likelihood estimation with robust 'Huber-White' standard errors and a scaled test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Note that the "MLM", "MLF" and "MLR" choices only affect the standard errors and the test statistic. They also imply <code>mimic="Mplus"</code> .
<code>likelihood</code>	Only relevant for ML estimation. If "wishart", the wishart likelihood approach is used. In this approach, the covariance matrix has been divided by N-1, and both standard errors and test statistics are based on N-1. If "normal", the normal likelihood approach is used. Here, the covariance matrix has been divided by N, and both standard errors and test statistics are based on N. If "default", it depends on the mimic option: if <code>mimic="Mplus"</code> , normal likelihood is used; otherwise, wishart likelihood is used.
<code>information</code>	If "expected", the expected information matrix is used (to compute the standard errors). If "observed", the observed information matrix is used.

	If "default", the value is set depending on the estimator and the mimic option.
se	If "standard", conventional standard errors are computed based on inverting the (expected or observed) information matrix. If "first.order", standard errors are computed based on first-order derivatives. If "robust.mlm", conventional robust standard errors are computed. If "robust.mlr", standard errors are computed based on the 'mlr' (aka pseudo ML, Huber-White) approach. If "robust", either "robust.mlm" or "robust.mlr" is used depending on the estimator, the mimic option, and whether the data are complete or not. If "boot" or "bootstrap", bootstrap standard errors are computed using standard bootstrapping (unless Bollen-Stine bootstrapping is requested for the test statistic; in this case bootstrap standard errors are computed using model-based bootstrapping). If "none", no standard errors are computed.
test	If "standard", a conventional chi-square test is computed. If "Satorra-Bentler", a Satorra-Bentler scaled test statistic is computed. If "Yuan-Bentler", a Yuan-Bentler scaled test statistic is computed. If "boot" or "bootstrap" or "bollen.stine", the Bollen-Stine bootstrap is used to compute the bootstrap probability value of the test statistic. If "default", the value depends on the values of other arguments.
bootstrap	Number of bootstrap draws, if bootstrapping is used.
mimic	If "Mplus", an attempt is made to mimic the Mplus program. If "EQS",

	<p>an attempt is made to mimic the EQS program. If "default", the value is (currently) set to "Mplus".</p>
<code>representation</code>	<p>If "LISREL" the classical LISREL matrix representation is used to represent the model (using the all-y variant).</p>
<code>do.fit</code>	<p>If FALSE, the model is not fit, and the current starting values of the model parameters are preserved.</p>
<code>control</code>	<p>A list containing control parameters passed to the optimizer. By default, lavaan uses "nlminb". See the manpage of nlminb for an overview of the control parameters. A different optimizer can be chosen by setting the value of <code>optim.method</code>. For unconstrained optimization (the model syntax does not include any "==" , ";" or ";" operators), the available options are "nlminb" (the default), "BFGS" and "L-BFGS-B". See the manpage of the <code>optim</code> function for the control parameters of the latter two options. For constrained optimization, the only available option is "nlminb.constr".</p>
<code>start</code>	<p>If it is a character string, the two options are currently "simple" and "Mplus". In the first case, all parameter values are set to zero, except the factor loadings (set to one), the variances of latent variables (set to 0.05), and the residual variances of observed variables (set to half the observed variance). If "Mplus", we use a similar scheme, but the factor loadings are estimated using the <code>fabin3</code> estimator (tsls) per factor. If <code>start</code> is a fitted object of class <code>lavaan-class</code>, the estimated values of the corresponding parameters will be extracted. If it is a model list, for example the output of the</p>

	<code>parameterEstimates()</code> function, the values of the <code>est</code> or <code>start</code> or <code>ustart</code> column (whichever is found first) will be extracted.
<code>verbose</code>	If <code>TRUE</code> , the function value is printed out during each iteration.
<code>warn</code>	If <code>TRUE</code> , some (possibly harmless) warnings are printed out during the iterations.
<code>debug</code>	If <code>TRUE</code> , debugging information is printed out.

Power-user fitting functions

- the `lavaan()` function does NOT do anything automatically
 1. no model parameters are added to the parameter table
 2. no actions are taken to make the model identifiable (e.g. setting the metric of the latent variables)

Example model syntax using the `lavaan()` function

```
HS.model.full <- ' # latent variables
                  visual  =~ 1*x1 + x2 + x3
                  textual =~ 1*x4 + x5 + x6
                  speed   =~ 1*x7 + x8 + x9

                  # factor variances
                  visual  ~~ visual
                  textual ~~ textual
                  speed   ~~ speed

                  # factor covariances
                  visual  ~~ textual
```

```
visual ~~ speed
textual ~~ speed

# residual variances observed variables
x1 ~~ x1
x2 ~~ x2
x3 ~~ x3
x4 ~~ x4
x5 ~~ x5
x6 ~~ x6
x7 ~~ x7
x8 ~~ x8
x9 ~~ x9
,

fit <- lavaan(HS.model.full, data=HolzingerSwineford1939)
```

Combining the lavaan() function with auto.* arguments

- several `auto.*` arguments are available to
 - automatically add a set of parameters (e.g. all (residual) variances)
 - take actions to make the model identifiable (e.g. set the metric of the latent variables)

Example using lavaan with an auto.* argument

```
HS.model.mixed <- ' # latent variables
                  visual  =~ 1*x1 + x2 + x3
                  textual =~ 1*x4 + x5 + x6
                  speed   =~ 1*x7 + x8 + x9
                  # factor covariances
                  visual  ~~ textual + speed
                  textual  ~~ speed
                  ,
fit <- lavaan(HS.model.mixed, data=HolzingerSwineford1939,
             auto.var=TRUE)
```

keyword	operator	parameter set
<code>auto.var</code>	~~	(residual) variances observed and latent variables
<code>auto.cov.y</code>	~~	(residual) covariances observed and latent endogenous variables
<code>auto.cov.lv.x</code>	~~	covariances among exogenous latent variables
keyword	default	action
<code>auto.fix.first</code>	TRUE	fix the factor loading of the first indicator to 1
<code>auto.fix.single</code>	TRUE	fix the residual variance of a single indicator to 0
<code>int.ov.free</code>	TRUE	freely estimate the intercepts of the observed variables (only if a mean structure is included)
<code>int.lv.free</code>	FALSE	freely estimate the intercepts of the latent variables (only if a mean structure is included)

1.3 Extractor functions: inspecting fitted models

Method	Description
<code>summary()</code>	print a long summary of the model results
<code>show()</code>	print a short summary of the model results
<code>coef()</code>	returns the estimates of the free parameters in the model as a named numeric vector
<code>fitted()</code>	returns the implied moments (covariance matrix and mean vector) of the model
<code>resid()</code>	returns the raw, normalized or standardized residuals (difference between implied and observed moments)
<code>vcov()</code>	returns the covariance matrix of the estimated parameters
<code>predict()</code>	compute factor scores
<code>logLik()</code>	returns the log-likelihood of the fitted model (if maximum likelihood estimation was used)
<code>AIC()</code> , <code>BIC()</code>	compute information criteria (if maximum likelihood estimation was used)
<code>update()</code>	update a fitted lavaan object
<code>inspect()</code>	peek into the internal representation of the model; by default, it returns a list of model matrices counting the free parameters in the model; can also be used to extract starting values, gradient values, and much more

1.4 Other functions

Function	Description
<code>lavaanify()</code>	converts a lavaan model syntax to a parameter table
<code>parameterTable()</code>	returns the parameter table
<code>parameterEstimates()</code>	returns the parameter estimates, including confidence intervals, as a data frame
<code>standardizedSolution()</code>	returns one of three types of standardized parameter estimates, as a data frame
<code>modindices()</code>	computes modification indices and expected parameter changes
<code>bootstrapLavaan()</code>	bootstrap any arbitrary statistic that can be extracted from a fitted lavaan object
<code>bootstrapLRT()</code>	bootstrap a chi-square difference test for comparing two alternative models

1.5 Meanstructures

- traditionally, SEM has focused on covariance structure analysis
- but we can also include the means
- typical situations where we would include the means are:
 - multiple group analysis
 - growth curve models
 - analysis of non-normal data, and/or missing data
- we have more data: the p -dimensional mean vector
- we have more parameters:
 - means/intercepts for the observed variables
 - means/intercepts for the latent variables (often fixed to zero)

Adding the means in lavaan

- when the `meanstructure` argument is set to `TRUE`, a meanstructure is added to the model

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

- if no restrictions are imposed on the means, the fit will be identical to the non-meanstructure fit
- we add p datapoints (the mean vector)
- we add p free parameters (the intercepts of the observed variables)
- we fix the latent means to zero
- the number of degrees of freedom does not change

Output meanstructure=TRUE

lavaan (0.4-12) converged normally after 41 iterations

Number of observations	301
Estimator	ML
Minimum Function Chi-square	85.306
Degrees of freedom	24
P-value	0.000

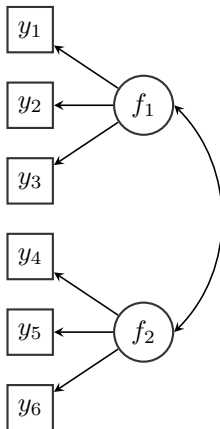
Parameter estimates:

Information	Expected					
Standard Errors	Standard					
		Estimate	Std.err	Z-value	P(> z)	
Latent variables:						
visual =~						
		x1	1.000			
		x2	0.553	0.100	5.554	0.000
		x3	0.729	0.109	6.685	0.000
textual =~						
		x4	1.000			
		x5	1.113	0.065	17.014	0.000
		x6	0.926	0.055	16.703	0.000
speed =~						
		x7	1.000			

x8	1.180	0.165	7.152	0.000
x9	1.082	0.151	7.155	0.000
Covariances:				
visual ~~				
textual	0.408	0.074	5.552	0.000
speed	0.262	0.056	4.660	0.000
textual ~~				
speed	0.173	0.049	3.518	0.000
Intercepts:				
x1	4.936	0.067	73.473	0.000
x2	6.088	0.068	89.855	0.000
x3	2.250	0.065	34.579	0.000
x4	3.061	0.067	45.694	0.000
x5	4.341	0.074	58.452	0.000
x6	2.186	0.063	34.667	0.000
x7	4.186	0.063	66.766	0.000
x8	5.527	0.058	94.854	0.000
x9	5.374	0.058	92.546	0.000
visual	0.000			
textual	0.000			
speed	0.000			
Variances:				
x1	0.549	0.114		
x2	1.134	0.102		
...				

1.6 Multiple groups

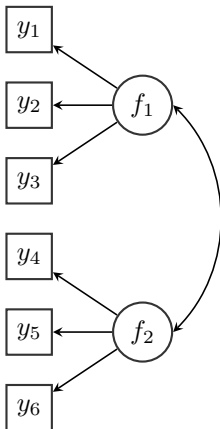
Single group analysis (CFA)



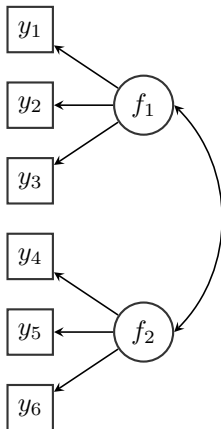
- factor means typically fixed to zero

Multiple group analysis (CFA)

GROUP 1



GROUP 2



- can we compare the means of the latent variables?

Measurement Invariance in lavaan

```
# model 1: configural invariance
fit1 <- cfa(HS.model, data=HolzingerSwineford1939, group="school")

# model 2: weak invariance
fit2 <- cfa(HS.model, data=HolzingerSwineford1939, group="school",
           group.equal="loadings")

# model 3: strong invariance
fit3 <- cfa(HS.model, data=HolzingerSwineford1939, group="school",
           group.equal=c("loadings", "intercepts"))
```

Comparing two (nested) models: the anova() function

```
anova(fit1, fit2)
```

Chi Square Difference Test

	Df	AIC	BIC	Chisq	Chisq diff	Df diff	Pr(>Chisq)
fit1	48	7484.4	7706.8	115.85			
fit2	54	7480.6	7680.8	124.04	8.1922	6	0.2244

Measurement invariance tests – all together

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

Measurement invariance tests:

Model 1: configural invariance:

chisq	df	pvalue	cfi	rmsea	bic
115.851	48.000	0.000	0.923	0.097	7706.822

Model 2: weak invariance (equal loadings):

chisq	df	pvalue	cfi	rmsea	bic
124.044	54.000	0.000	0.921	0.093	7680.771

[Model 1 versus model 2]

delta.chisq	delta.df	delta.p.value	delta.cfi
8.192	6.000	0.224	0.002

Model 3: strong invariance (equal loadings + intercepts):

chisq	df	pvalue	cfi	rmsea	bic
164.103	60.000	0.000	0.882	0.107	7686.588

[Model 1 versus model 3]

delta.chisq	delta.df	delta.p.value	delta.cfi
48.251	12.000	0.000	0.041

...

1.7 Missing data in lavaan

- if the data contain missing values, the default behavior in **lavaan** is listwise deletion
- if the missing mechanism is MCAR or MAR, the **lavaan** package provides case-wise (or ‘full information’) maximum likelihood (FIML) estimation by specifying the argument `missing="ml"` (or its alias `missing="fiml"`):

```
fit <- sem(myModel, data=myIncompleteData, missing="ml")
```

- an unrestricted (h1) model will automatically be estimated (using the EM algorithm), so that all common fit indices are available
- robust standard errors are also available, as is a scaled (‘Yuan-Bentler’) test statistic if the data are both incomplete and non-normal (`estimator="MLR"`)

1.8 Standard errors

- the `se` argument can be used to switch between different types of standard errors
- setting `se="robust"` will produce robust standard errors
 - if data is complete, lavaan will use `se="robust.mlm"`
 - if data is incomplete, lavaan will use `se="robust.mlr"`
- setting `se="boot"` or `se="bootstrap"` will produce bootstrap standard errors
- setting `se="none"` will NOT compute standard errors

1.9 Test statistics

- the `test` argument can be used to switch between different test statistics:
 - `test="standard"` (default)
 - `test="satorra.bentler"`
 - `test="yuan.bentler"`
 - `test="bootstrap"` or `test="bollen.stine"`
 - `test="none"`
- combine both robust standard errors and a scaled test statistic:
 - `estimator="MLM"`
 - `estimator="MLR"`

1.10 BootstrapLavaan

- once a lavaan model has been fitted, you can bootstrap any statistic that you can extract from a fitted lavaan object
- examples:

```
# bootstrap model parameters
PAR.boot <- bootstrapLavaan(fit, R=10, type="ordinary",
                           FUN="coef")

# bootstrap test statistic + compute p-value
T.boot <- bootstrapLavaan(fit, R=10, type="bollen.stine",
                        FUN=fitMeasures, fit.measures="chisq")

pvalue.boot <- length(which(T.boot > T.orig))/length(T.boot)

# bootstrap CFI
CFI.boot <- bootstrapLavaan(fit, R=10, type="parametric",
                          FUN=fitMeasures, fit.measures="cfi",
                          parallel="multicore", ncpus=8)
```

1.11 Constraints and defined parameters

linear and nonlinear equality and inequality constraints

```
Data <- data.frame(  y = rnorm(100),
                    x1 = rnorm(100),
                    x2 = rnorm(100),
                    x3 = rnorm(100) )

model.constr <- ' # model with labeled parameters
                 y ~ b1*x1 + b2*x2 + b3*x3

                 # constraints
                 b1 == (b2 + b3)^2
                 b1 > exp(b2 + b3)
                 ,

fit <- sem(model.constr, data=Data)
```

defined parameters and mediation analysis

```
X <- rnorm(100)
M <- 0.5*X + rnorm(100)
Y <- 0.7*M + rnorm(100)
Data <- data.frame(X = X, Y = Y, M = M)

model <- ' # direct effect
          Y ~ c*X
          # mediator
          M ~ a*X
          Y ~ b*M

          # indirect effect (a*b)
          ab := a*b
          # total effect
          total := c + (a*b)
          ,

fit <- sem(model, data=Data)
```

1.12 Using a covariance matrix as input

```
lower <- '
  11.834,
  6.947,    9.364,
  6.819,    5.091,    12.532,
  4.783,    5.028,    7.495,    9.986,
 -3.839,   -3.889,   -3.841,   -3.625,    9.610,
-21.899,  -18.831,  -21.748,  -18.775,   35.522,   450.288 '
```

```
# classic wheaton et al model
wheaton.cov <- getCov(lower,
                      names=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)
```

2 Some technical details

2.1 Default estimator: ML

- ML is the default estimator in all software packages for SEM
- the likelihood function is derived from the multivariate normal distribution (the ‘normal’ tradition) or the Wishart distribution (the ‘Wishart’ tradition)
- standard errors are usually based on the covariance matrix that is obtained by inverting the expected information matrix

$$\begin{aligned}n\text{Cov}(\hat{\theta}) &= A^{-1} \\ &= (\Delta'W\Delta)^{-1}\end{aligned}$$

- Δ is a jacobian matrix and W is a function of Σ^{-1}
- if no meanstructure:

$$\begin{aligned}\Delta &= \partial\hat{\Sigma}/\partial\hat{\theta}' \\ W &= 2D'(\hat{\Sigma}^{-1} \otimes \hat{\Sigma}^{-1})D\end{aligned}$$

- an alternative is to use the *observed* information matrix

$$\begin{aligned}n\text{Cov}(\hat{\theta}) &= A^{-1} \\ &= [-\text{Hessian}]^{-1} \\ &= \left[-\partial F(\hat{\theta})/(\partial\hat{\theta}\partial\hat{\theta}')\right]^{-1}\end{aligned}$$

where $F(\theta)$ is the function that is minimized

- overall model evaluation is based on the likelihood-ratio (LR) statistic (chi-square test): T_{ML}
 - (minus two times the) difference between loglikelihood of user-specified model H_0 and unrestricted model H_1
 - equals (in lavaan) $2 \times n$ times the minimum value of $F(\theta)$
 - T_{ML} follows (under regularity conditions) a chi-square distribution

2.2 Estimator MLM

- parameter estimates are standard ML estimates
- standard errors are robust to non-normality
 - standard errors are computed using a sandwich-type estimator:

$$\begin{aligned}n\text{Cov}(\hat{\theta}) &= A^{-1}BA^{-1} \\ &= (\Delta'W\Delta)^{-1}(\Delta'WTW\Delta)(\Delta'W\Delta)^{-1}\end{aligned}$$

- A is usually the expected information matrix (but not in Mplus)
- references: Huber (1967), Browne (1984), Shapiro (1983), Bentler (1983), ...

- chi-square test statistic is robust to non-normality
 - test statistic is ‘scaled’ by a correction factor

$$T_{SB} = T_{ML}/c$$

- the scaling factor c is computed by:

$$c = tr [UT] /df$$

where

$$U = (W^{-1} - W^{-1}\Delta(\Delta'W^{-1}\Delta)^{-1}\Delta'W^{-1})$$

- correction method described by Satorra & Bentler (1986, 1988, 1994)
- estimator MLM: for complete data only

2.3 Estimator MLR

- parameter estimates are standard ML estimates
- standard errors are robust to non-normality
 - standard errors are computed using a (different) sandwich approach:

$$\begin{aligned}n\text{Cov}(\hat{\theta}) &= A^{-1}BA^{-1} \\ &= A_0^{-1}B_0A_0^{-1} = C_0\end{aligned}$$

where

$$A_0 = - \sum_{i=1}^n \frac{\partial \log L_i}{\partial \hat{\theta} \partial \hat{\theta}'} \quad (\text{observed information})$$

and

$$B_0 = \sum_{i=1}^n \left(\frac{\partial \log L_i}{\partial \hat{\theta}} \right) \times \left(\frac{\partial \log L_i}{\partial \hat{\theta}} \right)'$$

- for both complete and incomplete data

- Huber (1967), Gouvieroux, Monfort & Trognon (1984), Arminger & Schoenberg (1989)
- chi-square test statistic is robust to non-normality
 - test statistic is ‘scaled’ by a correction factor

$$T_{MLR} = T_{ML}/c$$

- the scaling factor c is (usually) computed by

$$c = tr [M]$$

where

$$M = C_1(A_1 - A_1\Delta(\Delta'A_1\Delta)^{-1}\Delta'A_1)$$

- A_1 and C_1 are computed under the unrestricted (H_1) model
- correction method described by Yuan & Bentler (2000)
- information matrix (A) can be observed or expected
- for complete data, the MLR and MLM corrections are asymptotically equivalent