Score-based tests in new lavaan versions

Ed Merkle

August 17, 2015

Consider the following example from Wang, Merkle, & Zeileis (2014) (note this is a custom lavaan version, as I removed the most recent changes to `lavScores()` involving the `eq.constraints.K` stuff):

```
library("lavaan", lib.loc="~/dev_package_versions")
```

```
## This is lavaan 0.5-19.872
## lavaan is BETA software! Please report any bugs.
```

```
data("YouthGratitude", package = "psychotools")
compcases <- apply(YouthGratitude[, 4:28], 1, function(x) all(x %in% 1:9))
yg <- YouthGratitude[compcases, ]
restr <- cfa("f1 = ~gq6_1 + gq6_2 + gq6_3 + gq6_4 + gq6_5",
   data = yg, group = "agegroup", meanstructure = TRUE,
   group.equal = "loadings")
```

The four loading parameters are constrained to be equal across groups, and we wish to carry out score-based tests related to these four parameters. The score-based tests can be carried out via package *strucchange* when the above model is estimated in *lavaan* 0.5-17. Starting in *lavaan* 0.5-18, equality constraints are handled differently. As a result, the following `sctest()` call throws an error:

```
library("strucchange")
sctest(restr, order.by = yg$agegroup, parm=1:4, vcov="info", functional="LMuo")
```

```
## Error in solve.default(vcov0(x) * nobs0(x)): system is computationally singular: reciprocal condition
```

This error arises when `sctest()` tries to invert the result of `vcov()`. It happens because *lavaan* no longer "reduces" parameter vectors with equality constraints, so that multiple columns of the variance/covariance matrix are associated with the same (equality-constrained) parameter. The matrix cannot be inverted because it is less than full rank. We can also see that there is a problem with the scores:

```
scores <- lavScores(restr)
colSums(scores)[1:5]
```

```
##          .p2.          .p3.          .p4.          .p5.  gq6_1~~gq6_1
## -2.019058e+01  6.045181e+00  7.095746e+00  1.097756e+01 -6.829334e-04
```

In previous versions of *lavaan*, these sums were always 0 (reflecting the gradient evaluated at the ML estimates). Now, some columns do not sum to zero because the scores associated with a single parameter are contained in multiple columns of `scores`.

Possible solutions

Two possible solutions (that do not work) involve (i) obtaining the information matrix directly from *lavaan* and maintaining the "full" model, and (ii) using the *lavaan* projection matrix `eq.constraints.K`, which projects a vector from the full parameter space to the reduced parameter space.

Pursuing the first solution, we can at least avoid the error from `sctest()` by supplying our own information matrix via `lavInspect`. First, we define the custom function `lavinfo`:

```
lavinfo <- function(fit, ...) lavInspect(fit, "information")
```

which obtains the information matrix directly from `lavaan` instead of relying on `strucchange` to do it. Second, we transform the score matrix so that each column again sums to zero. This involves obtaining the gradient of the likelihood at the ML estimates; note that the optimization function now contains a second term that involves Lagrange multipliers, so this gradient will not equal zero at the ML estimates. We apply this gradient to the scores, which accounts for the second term in the optimization function:

```
lavsc <- function(fit, ...) t(t(lavScores(fit)) + lavInspect(fit,"gradient"))
```

We now modify our call to `sctest`:

```
sctest(restr, order.by=yg$agegroup, parm=1:4, scores=lavsc,
       vcov=lavinfo, sandwich=FALSE, functional="LMuo")
```

```
##
##  M-fluctuation test
##
## data:  restr
## f(efp) = 24.112, p-value = 0.2375
```

which returns a plausible result, but the result still differs from previous versions (see Wang et al, 2014, p. 4, top first column). This is because the argument `parm=1:4` gives us only the first group's loading parameters, and we already saw above that the scores in these columns do not sum to zero. To fix this, we could find all the columns that correspond to these four loadings; these column names happen to be the only ones that have the string ".p" in them:

```
sctest(restr, order.by=yg$agegroup, parm=grep(".p", names(coef(restr))),
       scores=lavsc, vcov=lavinfo, sandwich=FALSE, functional="LMuo")
```

```
##
##  M-fluctuation test
##
## data:  restr
## f(efp) = 175.87, p-value = 0.000678
```

We now obtain a test statistic that is an order of magnitude too high; this seems related to the fact that we chose 24 parameters of the original matrix (6 groups $\times$ 4 equality-constrained parameters), with each score column summing to zero. In computing the test statistic, `sctest()` thinks we are testing 24 unique parameters instead of only 4, and the scores should not sum to zero in columns that reflect the same parameter. These results seem to imply that *strucchange* relies on the reduced model in order to produce the desired results. Related to this point, I can use the above approach with *nonnest2* and obtain the "correct" test statistics. I

cannot obtain the correct p-values for statistics that have weighted sum of χ^2 limiting distributions, because the algorithms implicitly use the number of model parameters to count degrees of freedom.

Pursuing the second solution, we can define custom functions to project the original score vector and information matrix into the "reduced" dimension. We can then use these in our call to `sctest()`:

```
lavsc <- function(fit, ...) lavScores(fit) %*% fit@Model@eq.constraints.K
lavinfo <- function(fit, ...) t(fit@Model@eq.constraints.K) %*%
                              lavInspect(fit, "information") %*%
                              fit@Model@eq.constraints.K

sctest(restr, order.by=yg$agegroup, parm=1:4, scores=lavsc, vcov=lavinfo,
       sandwich=FALSE, functional="LMuo")
```

```
##
##  M-fluctuation test
##
## data:  restr
## f(efp) = 0.081705, p-value = 1
```

This result is again implausible. It is because, after we project the original score matrix into a lower dimension, each column of the matrix no longer corresponds to a single parameter. Hence, the argument `parm=1:4` tests some function of the scores that no longer corresponds to the four loadings of interest. It is unclear how we could test specific parameters using this projection.

Preliminary solution

(*maybe you should skip to "general solution", because that supercedes this section*) There is a simple solution for models with simple equality constraints (i.e., multiple freely-estimated parameters constrained to be equal to one another), which should include all the models from Merkle & Zeileis, 2013, Merkle, Fan, & Zeileis, 2014, and Wang, Merkle, & Zeileis, 2014. These are (presumably) models that satisfy the following conditions

```
model.works <- TRUE
if(restr@Model@eq.constraints){
  ones <- (restr@Model@ceq.JAC == 1 | restr@Model@ceq.JAC == -1)
  model.works <- all(restr@Model@ceq.rhs == 0) &
                 all(apply(restr@Model@ceq.JAC != 0, 1, sum) == 2) &
                 all(apply(ones, 1, sum) == 2) &
                 length(restr@Model@ceq.nonlinear.idx) == 0
}
```

These conditions say that the constraints all involve a simple difference between two parameters that must equal 0. In retrospect, I don't think the *strucchange* tests ever worked for models with more general equality constraints (I never considered them until now).

In the simple cases outlined above, the reduced score vector can be obtained by finding columns of the full score vector that contain equality-constrained parameters, then summing across these columns. The reduced `vcov()` can be obtained by taking only one row/column that corresponds to an equality-constrained parameter, as opposed to taking all of them. The equality-constrained parameters are not entirely straightforward to find because there currently do not appear to be any indicators in *lavaan*. We find them via this code:

```
## parameters with equality constraints:
lpt <- parTable(restr)
eq.rows <- which(lpt$op == "==")
eq.pars <- tapply(lpt$rhs[eq.rows], lpt$lhs[eq.rows], unique)
```

We can then make use of the results to obtain the reduced score matrix and information matrix:

```
lavsc <- function(fit, ...){
  lpt <- parTable(fit)
  eq.rows <- which(lpt$op == "==")
  eq.pars <- tapply(lpt$rhs[eq.rows], lpt$lhs[eq.rows], unique)
  fsname <- lpt$plabel[lpt$free > 0]
  redscores <- lavScores(fit)
  rmcols <- which(fsname %in% unlist(eq.pars))
  for(i in 1:length(eq.pars)){
    sumcols <- which(fsname %in% eq.pars[[i]])
    eqloc <- match(names(eq.pars)[i], fsname)
    redscores[,eqloc] <- rowSums(redscores[,c(eqloc,sumcols)])
  }
  redscores <- redscores[,-rmcols]
  redscores
}

lavinfo <- function(fit, ...){
  lpt <- parTable(fit)
  eq.rows <- which(lpt$op == "==")
  eq.pars <- tapply(lpt$rhs[eq.rows], lpt$lhs[eq.rows], unique)
  fsname <- lpt$plabel[lpt$free > 0]
  rmcols <- which(fsname %in% unlist(eq.pars))
  fullvcov <- vcov(fit)
  redvcov <- fullvcov[-rmcols,-rmcols]
  solve(redvcov * sum(unlist(restr@SampleStats@nobs)))
}
```

Now, we can call `sctest()` on this new output:

```
sctest(restr, order.by=yg$agegroup, parm=1:4, scores=lavsc, vcov=lavinfo,
       sandwich=FALSE, functional="LMuo")
```

```
##
##  M-fluctuation test
##
## data:  restr
## f(efp) = 31.396, p-value = 0.05018
```

This matches closely with the result from Wang et al. (2014). The result may differ slightly because *lavaan* is optimizing the model in a new way, so that the 0.5-18 ML estimates may slightly differ from the 0.5-17 ML estimates.

General solution

Say that we have the parameter constraint $\theta_1 + \theta_2 + \theta_3 = 3$. Using the full version of the model, we can get the gradient (and scores) associated with all three parameters:

$$g(\theta_1) = \frac{\partial L(\theta_1, \theta_2, \theta_3, \dots | X)}{\partial \theta_1}$$
$$g(\theta_2) = \frac{\partial L(\theta_1, \theta_2, \theta_3, \dots | X)}{\partial \theta_2}$$
$$g(\theta_3) = \frac{\partial L(\theta_1, \theta_2, \theta_3, \dots | X)}{\partial \theta_3}.$$

Now, let's take $\theta_3 = (3 - \theta_1 - \theta_2)$, so that θ_3 disappears from the model, allowing us to focus on a reduced model. Let the gradient of the reduced model be denoted $g^*()$ with reduced model parameters also having asterisks. We can express the full model log-likelihood as two sub-functions, one of which includes θ_1 and one of which includes θ_3:

$$L(\theta_1^*, \theta_2, \theta_4, \dots | X) = f_1(\theta_1, \theta_2, \theta_4, \dots | X) f_2(\theta_3, \theta_2, \theta_4, \dots | X).$$

To get the gradient associated with θ_1^*, we can now use a product rule combined with a chain rule (since θ_3 is a function of θ_1):

$$g^*(\theta_1^*) = g(\theta_1) + \left(g(\theta_3) \times \frac{\partial (3 - \theta_1^* - \theta_2^*)}{\partial \theta_1^*} \right).$$

So this would be $g(\theta_1) - g(\theta_3)$, both of which we can obtain from the full model `lavScores()` (AKA `estfun()`). In general, we can implement this if we are able express one parameter involved in the constraint as a function of other parameters. This is simple for equality constraints, simple summations, and products, but it becomes more difficult to do automatically for things like

$$(\theta_1 + \theta_2)^2 == 2$$
$$\sin(\theta_1 \theta_2) == .5.$$

For constraints like these, perhaps `lavScores()` could throw an error and ask the user to specify the constraint using a single parameter on the lhs? If there is a single parameter on the lhs, we can always use numerical derivatives to obtain what we need. Or perhaps there is a way to use the results from `ceq.JAC` (for those in `ceq.linear.idx`) and `con.jac` (for those in `ceq.nonlinear.idx`) to deal with this?

Here is an example showing how we could obtain reduced model scores when we have a simple nonlinear constraint:

```
hsm2 <- ' visual  =~ x1 + a*x2 + b*x3
          textual =~ x4 + x5 + x6
          speed   =~ x7 + x8 + x9
          a*b == 1 '

fit <- cfa(hsm2, data=HolzingerSwineford1939)

ls <- lavScores(fit)
## assuming b = 1/a, convert to reduced model a parameter
## (knowing that db/da = -1/a^2)
ls[,1] <- ls[,1] - ls[,2]/(coef(fit)[1]^2)
ls <- ls[,-2]
head(colSums(ls))
```

```
##             a   textual=~x5   textual=~x6     speed=~x8     speed=~x9
## -7.050193e-05  7.784122e-05  1.339228e-04  3.602278e-04 -3.823675e-05
##        x1~~x1
## -8.792019e-05
```

Etc

Beyond this, Yves suggested that *lavaan* can give us Lagrange multiplier values associated with each constraint. These could possibly be used to augment the score matrix so that, if we then wish to test some equality-constrained parameters, we would presumably use the columns of the score matrix that correspond to the Lagrange multiplier values. I am not optimistic that this approach will work because the other packages implicitly assume that the model is in reduced form (vs full form with constraints) for counting parameters, degrees of freedom, etc. But I may not understand the suggestion.

I also notice in the "equality constraints" document that `lav_partable_df()` does not yet deal with equality constraints. I think that the solution to this issue might be relevant to the "reduced model" issues here.