Score-based tests in lavaan 0.5-18

Ed Merkle

08/03/2015

Consider the following example from Wang, Merkle, & Zeileis (2014):

```
library("lavaan")
```

```
## This is lavaan 0.5-19.871
## 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 (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. We now modify our call to `sctest`:

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

```
##
##  M-fluctuation test
##
## data:  restr
## f(efp) = 28.441, p-value = 0.09936
```

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))), vcov=lavinfo,
       sandwich=FALSE, functional="LMuo")
```

```
##
##  M-fluctuation test
##
## data:  restr
## f(efp) = 211.38, p-value = 5.244e-07
```

We now obtain a test statistic that is an order of magnitude too high; this is related to the fact that we chose 24 parameters of the original matrix (6 groups $\times$ 4 equality-constrained parameters). In computing the test statistic, `sctest()` thinks we are testing 24 unique parameters instead of only 4, and the scores do not sum to zero in any of these columns.

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.

Preliminary solution

There exists 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 these cases, 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.

Potential full solution

If we wish to apply the tests to models with more general constraints, Yves suggests that *lavaan* can give us Lagrange multiplier values associated with each constraint. These could 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. An augmented information matrix can already be obtained for these purposes via

```
auginfo <- lavInspect(restr, "augmented.information")
```

Yves suggests looking at the `lavTestScore()` function for hints on augmenting the scores. This option has not yet been explored.