Package `kcirt Tutorial`

Dave Zes, Jimmy Lewis, Dana Landis
@ Korn/Ferry International

October 10, 2013

0.1 Example I

We commence with an itty-bitty example.

Suppose we have 6 blocks, the first three contain 3 items each, the last three, 4 items each. We tap three states (aka scales).

```
options(stringsAsFactors=FALSE, width=140)
library(kcirt)
constructMap.ls <- list(
  c(1,2,3),
  c(1,2,3),
  c(1,2,3),
  c(1,1,2,3),
  c(1,2,2,3),
  c(1,2,3,3)
  )
```

Create true Lambda. Keying is done by signing. By default these will be the diagonal elements of the Lambda matrix, $\mathbf{\Lambda}$.

```
set.seed(99999)
mxLambda <- c(
  c(1,1,-1),
  c(1,-1,1),
  c(-1,1,1),
  c(1,-1,1,-1),
```

```
  c(1,-1,1,-1),
  c(1,-1,1,-1)
  )
```

Create true state covariance structure.

```
covEta <- diag(1, 3)
```

Define response format. For only 2 or 3 items in block, kcirt only accepts "R". The four item blocks are most/least like.

```
qTypes <- c("R", "R", "R", "M", "M", "M")
```

Create true utilities.

```
mu <- rep(0, length(mxLambda))
```

Now build model skeleton and examine contents.

```
mod1 <- kcirt.model(constructMap.ls=constructMap.ls, qTypes=qTypes, mxLambda=mxLambda,
  mu=mu, covEta=covEta)
mod1
```

Take a peek at the information $\boldsymbol{\eta}$ inherits from $\mathbf{y}^{\star}$.

```
kcirt.ystarinfo(mod1)
```

Notice, above, that inasmuch as the loadings have equal magnitude, their keying and the respective location of items within provide equal information to each state.

Let's now draw a random realization from under this parameterized model, and look at the rank data.

```
N <- 200
set.seed(99999)
mod2 <- kcirt.sim(mod1, N=N)
mod2$mxData
```

Let's now fit model. We simulate an initial guess for the loadings, $\mathbf{\Lambda}$, by adding some noise to the true values.

```
mxHatLambda <- matrix(0, nrow(mod2$mxLambda), ncol(mod2$mxLambda))
diag(mxHatLambda) <- diag(mod2$mxLambda) + rnorm(nrow(mod2$mxLambda), 0, 0.2)
```

Set an initial guess for $\boldsymbol{\mu}$ at zero. Set predicted states to zero as starting point.

```
hatMu <- rep(0, length(mxLambda))
mxHatEta <- matrix(0, N, 3)
```

Insert these three objects into model.

```
mod2$mxHatLambda <- mxHatLambda
mod2$mxHatEta <- mxHatEta
mod2$hatMu <- hatMu


mod3 <- mod2
```

Use stochastic search to locate (internally in order) $\boldsymbol{\mu}$, $\mathbf{\Lambda}$, $\mathbf{E}$. View performance (reliabilities), plot $\widehat{\boldsymbol{\eta}}$ versus $\boldsymbol{\eta}$ and $\widehat{\mathbf{\Lambda}}$ versus $\mathbf{\Lambda}$.

```
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5))
mod3$performance
mean(mod3$performance)
par(mfrow=c(1, 4))
for(i in 1:ncol(mod3$mxHatEta)) {
  plot(mod3$mxEta[ ,i], mod3$mxHatEta[ ,i],
  main = round(1000*cor(mod3$mxHatEta[ ,i], mod3$mxEta[ ,i])^2)/1000 )
  }
plot(mod3$mxLambda, mod3$mxHatLambda)
```

Notice the argument `mss.sd` in `kcirt.fitMSS()`. We have supplied it with a three-element vector. The values are the respective search function standard deviations used when locating $\boldsymbol{\mu}$, $\mathbf{\Lambda}$, $\mathbf{E}$. In the example, we have made the search for $\boldsymbol{\mu}$ comparitively *tight*, implying a belief that our starting $\widehat{\boldsymbol{\mu}}$ is close to $\boldsymbol{\mu}$. In the bigger picture — the actual fitting of our model to data — these values

have little impact as numerous MSS passes can be made over the data, allowing the hatted values to wander about to the fancy of the researcher.

That said, try running the above code block 3 or 4 more times, and notice the average reliability climbs to about 74%.

0.2 Example II

Here we will introduce the notion of item *crosstalk*, and how it might be accounted for in a `kcirt` model.

In conversational terms, suppose that the preference of items in a block is not only determined by independent comparison, but also by mutual moderation between items within the block. For example, in a block containing four items, A, B, C, D, a respondent chooses A over B based on the interplay of items B, C, D upon the perceived appropriateness of item A, and the interplay of items A, C, D, upon the perceived appropriateness of item B.

We use the term *moderation* somewhat loosely. While $\mathbf{\Lambda S \eta}$ is linear, the elements of this matrix product vector are multiplicative with respect to loadings and their respective states.

Let's start with a super itty bitty example.

```
constructMap.ls <- list(
  c(1,1,2),
  c(2,2,3),
  c(3,3,1)
  )
qTypes <- c("R", "R", "R")
mod1 <- kcirt.model(constructMap.ls=constructMap.ls, qTypes=qTypes)
```

The first row of

```
mod1$mxLambdaCTinfo
```

shows the model relationship between the first item and all items on the instrument.

```
"S"  "WT" "WF" "BF" "BF" "BF" "BF" "BF" "BT"
```

The first element gives the relationship between item 1 and item 1 — "S" for self. The second element gives the relationship between item 1 and item 2 — "WT". The W stands for "within", meaning that item 2 lives in the same question block as item 1. The "T" stands for "True", meaning that the state that item 2 points to is the same as the one item 1 points to. Notice, by looking at `constructMap.ls`, both these items point to the 1st state (the state called "1" in `constructMap.ls`). In the fourth element, the "B" stands for "between", meaning that item 1 and item 4 live in different question blocks, the "F" means that they point to different states.

0.3 Example III

Here we will deliberately mis-ID a system.

```
constructMap.ls <- list(
  c(1,2,3),
  c(1,2,3),
  c(1,2,3),
  c(1,1,2,3),
  c(1,2,2,3),
  c(1,2,3,3)
  )
mxLambda <- c(
  c(1,1,-1),
  c(1,-1,1),
  c(-1,1,1),
  c(1,-1,1,-1),
  c(1,-1,1,-1),
  c(1,-1,1,-1)
  )
covEta <- diag(1, 3)
qTypes <- c("R", "R", "R", "M", "M", "M")
mu <- rep(0, length(mxLambda))
mod1 <- kcirt.model(constructMap.ls=constructMap.ls, qTypes=qTypes, mxLambda=mxLambda,
  mu=mu, covEta=covEta)
```

Add within crosstalk.

```
set.seed(99999)
mod1$mxLambda[ mod1$mxLambdaCTinfo == "WF" ] <-
  rnorm( sum(mod1$mxLambdaCTinfo == "WF" ), 0, 0.1 )
```

Two-fold validation.

```
set.seed(99999)
N <- 1000
modX1 <- kcirt.sim(mod1, N=N)
modX2 <- kcirt.sim(mod1, N=N)
```

Fit the first data set, but incorrectly assume no crosstalk.

```
mxHatLambda <- matrix(0, nrow(modX1$mxLambda), ncol(modX1$mxLambda))
diag(mxHatLambda) <- diag(modX1$mxLambda) + rnorm(nrow(modX1$mxLambda), 0, 0.2)
mxHatEta <- matrix(0, N, 3)
hatMu <- rep(0, length(mxLambda))
modX1$mxHatLambda <- mxHatLambda
modX1$mxHatEta <- mxHatEta
modX1$hatMu <- hatMu
mod3 <- modX1
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5)) ### first MSS pass
mean(mod3$performance)
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5)) ### second MSS pass
mean(mod3$performance)
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5)) ### third MSS pass
mean(mod3$performance)
mod3$performance
par(mfrow=c(1, 4))
for(i in 1:ncol(mod3$mxHatEta)) {
  plot(mod3$mxEta[ ,i], mod3$mxHatEta[ ,i],
  main = round(1000*cor(mod3$mxHatEta[ ,i], mod3$mxEta[ ,i])^2)/1000 )
  }
plot(mod3$mxLambda, mod3$mxHatLambda)
```

Shove $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Lambda}}$ fitted from first half of data into second half of data. Prevent `kcirt.fitMSS()` from fitting (altering) $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Lambda}}$ by turning the first and second elements of `mss.sd` to zero.

```
modX2$hatMu <- mod3$hatMu
modX2$mxHatLambda <- mod3$mxHatLambda
modX2$mxHatEta <- matrix(0, N, 3)
```

```
mod4 <- modX2
mod4 <- kcirt.fitMSS(model=mod4, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0, 0, 0.5)) ### first MSS pass
mean(mod4$performance)
mod4 <- kcirt.fitMSS(model=mod4, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0, 0, 0.5)) ### second MSS pass
mean(mod4$performance)
mod4 <- kcirt.fitMSS(model=mod4, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0, 0, 0.5)) ### third MSS pass
mean(mod4$performance)
mod4$performance
par(mfrow=c(1, 4))
for(i in 1:ncol(mod4$mxHatEta)) {
  plot(mod4$mxEta[ ,i], mod4$mxHatEta[ ,i],
  main = round(1000*cor(mod4$mxHatEta[ ,i], mod4$mxEta[ ,i])^2)/1000 )
  }
plot(mod4$mxLambda, mod4$mxHatLambda)
```

0.4 Example IV

Lastly we will deliberately mis-ID a system by incorrectly assuming within block crosstalk.

```
constructMap.ls <- list(
  c(1,2,3),
  c(1,2,3),
  c(1,2,3),
  c(1,1,2,3),
  c(1,2,2,3),
  c(1,2,3,3)
  )
mxLambda <- c(
  c(1,1,-1),
  c(1,-1,1),
  c(-1,1,1),
  c(1,-1,1,-1),
  c(1,-1,1,-1),
  c(1,-1,1,-1)
  )
covEta <- diag(1, 3)
qTypes <- c("R", "R", "R", "M", "M", "M")
mu <- rep(0, length(mxLambda))
```

```
mod1 <- kcirt.model(constructMap.ls=constructMap.ls, qTypes=qTypes, mxLambda=mxLambda,
  mu=mu, covEta=covEta)
```

Again, use two-fold validation.

```
set.seed(99999)
N <- 1000
modX1 <- kcirt.sim(mod1, N=N)
modX2 <- kcirt.sim(mod1, N=N)
```

Fit the first data set, but incorrectly assume within block crosstalk.

```
mxHatLambda <- matrix(0, nrow(modX1$mxLambda), ncol(modX1$mxLambda))
diag(mxHatLambda) <- diag(modX1$mxLambda) + rnorm(nrow(modX1$mxLambda), 0, 0.2)
mxHatEta <- matrix(0, N, 3)
hatMu <- rep(0, length(mxLambda))
modX1$mxHatLambda <- mxHatLambda
modX1$mxHatEta <- mxHatEta
modX1$hatMu <- hatMu
mod3 <- modX1
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="withinx", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5)) ### first MSS pass
mean(mod3$performance)
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="withinx", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5)) ### second MSS pass
mean(mod3$performance)
mod3 <- kcirt.fitMSS(model=mod3, lambdaConstraint="withinx", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0.03, 0.03, 0.5)) ### third MSS pass
mean(mod3$performance)
mod3$performance
par(mfrow=c(1, 4))
for(i in 1:ncol(mod3$mxHatEta)) {
      plot(mod3$mxEta[ ,i], mod3$mxHatEta[ ,i],
      main = round(1000*cor(mod3$mxHatEta[ ,i], mod3$mxEta[ ,i])^2)/1000 )
  }
plot(mod3$mxLambda, mod3$mxHatLambda)
```

Again, shove $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Lambda}}$ fitted from first half of data into second half of data. Prevent `kcirt.fitMSS()` from fitting (altering) $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Lambda}}$ by turning the first and second elements of `mss.sd` to zero.

By the way, note that in the following code the argument `lambdaConstraint` makes no difference since we are in effect not actually fitting $\mathbf{\Lambda}$.

```
modX2$hatMu <- mod3$hatMu
modX2$mxHatLambda <- mod3$mxHatLambda
modX2$mxHatEta <- matrix(0, N, 3)
mod4 <- modX2
mod4 <- kcirt.fitMSS(model=mod4, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0, 0, 0.5)) ### first MSS pass
mean(mod4$performance)
mod4 <- kcirt.fitMSS(model=mod4, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0, 0, 0.5)) ### second MSS pass
mean(mod4$performance)
mod4 <- kcirt.fitMSS(model=mod4, lambdaConstraint="self", kcpus=2,
  penalty="logit", usetruesigma=TRUE, mss.sd=c(0, 0, 0.5)) ### third MSS pass
mean(mod4$performance)
mod4$performance
par(mfrow=c(1, 4))
for(i in 1:ncol(mod4$mxHatEta)) {
      plot(mod4$mxEta[ ,i], mod4$mxHatEta[ ,i],
      main = round(1000*cor(mod4$mxHatEta[ ,i], mod4$mxEta[ ,i])^2)/1000 )
  }
plot(mod4$mxLambda, mod4$mxHatLambda)
```

There seems to be some suggestion of resilience against mis-identification. Of course, a scientific argument would require many more replications, but this is just a tutorial.