Response to Issue #104 (Updated)

September 24, 2020

Summary of the issue

In order to reproduce the problematic results, I use the bootstrap bounds that are obtained from the `chorussell` procedure in the `lpinfer` package using the example code posted in issue #104.

```
library(lpinfer)
set.seed(5)
dgp <- mixedlogit_dgp()
df <- mixedlogit_draw(dgp, n = 4000)

lpm <- lpmodel(A.obs = mixedlogit_Aobs(dgp),
               beta.obs = function(d) mixedlogit_betaobs(d, dgp),
               A.shp = rep(1, nrow(dgp$vdist)),
               beta.shp = 1,
               A.tgt = mixedlogit_Atgt_dfelast(dgp, w2eval = 1, eeval = -1))

set.seed(5)
r <- chorussell(data = df, lpmodel = lpm, ci = TRUE, beta.tgt = .2,
                progress = FALSE)
print(r)
```

```
## 95%-confidence interval: [0.63392, 0.88549]
```

The problems here are:

- The largest bootstrap estimate of the upper bound is smaller than the upper bound of the confidence interval.
- More than half of the bootstrap estimates are smaller than the lower bound of the confidence interval.

The aim of this document is to go through the calculations I did in the `chorussell` function. For easy reference, I am also referencing the implementation notes posted in issue #79. The line where I make a reference to the implementation notes with a $\star$ symbol.

Constructing the confidence interval

Step 1. Get the point estimates of the bounds

The point estimates of the bounds are obtained by `estbounds` procedure as follows:

```
estb.return <- estbounds(df, lpm)
lb <- estb.return$lb
ub <- estb.return$ub
print(estb.return)
```

```
## Estimated bounds: [0.66418, 0.66468]
```

The default options of `estbounds` are used, i.e. setting `kappa = 0` and using the 2-norm. The results here match the point estimates from the `chorussell` procedure:

```
print(lb == r$lb)
```

```
## [1] TRUE
```

```
print(ub == r$ub)
```

```
## [1] TRUE
```

⋆ The `lb` and `ub` objects correspond to $\hat{\theta}_{\text{lb}}$ and $\hat{\theta}_{\text{ub}}$ respectively in the first bullet point of the implementation notes.

On the other hand, the length of the estimated identified set is stored as `delta`.

```
delta <- ub - lb
print(delta)
```

```
## [1] 0.0005065283
```

⋆ This corresponds to the third bullet point of the implementation notes.

Step 2. Get the bootstrap bounds

The bootstrap bounds are obtained by finding the estimated bounds on the bootstrap data. I am going to extract them from the `r` object above and store them as follows:

```
lb.bs <- r$lb.bs
ub.bs <- r$ub.bs
```

⋆ This corresponds to the second bullet point of the implementation notes.

Step 3. Get the list of candidates

The only possible solutions for c_{lb} should be inside the following set:

$$\{\sqrt{n}(\hat{\theta}^b_{\text{lb}} - \hat{\theta}_{\text{lb}})\}^B_{b=1} \cup \{\sqrt{n}(\hat{\theta}^b_{\text{lb}} - \hat{\theta}_{\text{lb}} - \Delta)\}^B_{b=1}.$$

In this document (and in the code for `chorussell`), I denote the set of candidates $\{\sqrt{n}(\hat{\theta}^b_{\text{lb}} - \hat{\theta}_{\text{lb}})\}^B_{b=1}$ and $\{\sqrt{n}(\hat{\theta}^b_{\text{lb}} - \hat{\theta}_{\text{lb}} - \Delta)\}^B_{b=1}$ as `lb.can1` and `lb.can2` respectively:

```
n <- nrow(df)
lb.can1 <- sqrt(n) * (lb.bs - lb)
lb.can2 <- sqrt(n) * (lb.bs - lb - delta)
lb.can <- c(lb.can1, lb.can2)
```

Similarly, the only possible solutions for c_{ub} should be inside the following set:

$$\{-\sqrt{n}(\hat{\theta}^b_{\text{ub}} - \hat{\theta}_{\text{ub}})\}^B_{b=1} \cup \{-\sqrt{n}(\hat{\theta}^b_{\text{ub}} - \hat{\theta}_{\text{ub}} + \Delta)\}^B_{b=1}.$$

I denote the set of candidates $\{-\sqrt{n}(\hat{\theta}^b_{\text{ub}} - \hat{\theta}_{\text{ub}})\}^B_{b=1}$ and $\{-\sqrt{n}(\hat{\theta}^b_{\text{ub}} - \hat{\theta}_{\text{ub}} + \Delta)\}^B_{b=1}$ as `ub.can1` and `ub.can2` respectively:

```
ub.can1 <- sqrt(n) * (ub.bs - ub)
ub.can2 <- sqrt(n) * (ub.bs - ub + delta)
ub.can <- -c(ub.can1, ub.can2)
```

Step 4. Simplify the list of candidates

Since the first constraint of the optimization problem is

$$\mathbf{1}\left[\sqrt{n}\left(\hat{\theta}^b_{\text{lb}} - \hat{\theta}_{\text{lb}}\right) \le c_{\text{lb}}\right] \ge \mathbf{1}\left[\sqrt{n}\left(\hat{\theta}^b_{\text{lb}} - \hat{\theta}_{\text{lb}}\right) \le c_{\text{lb}} \text{ and } -c_{\text{ub}} \le \sqrt{n}\left(\hat{\theta}^b_{\text{ub}} - \hat{\theta}_{\text{ub}} + \Delta\right)\right],$$

the list of candidates c_{lb} that satisfy

$$\frac{1}{B}\sum_{b=1}^{B}\mathbf{1}\left[\sqrt{n}\left(\hat{\theta}_{\text{lb}}^{b}-\hat{\theta}_{\text{lb}}\right)\leq c_{\text{lb}}\right]<1-\alpha$$

cannot satisfy the first constraint of the minimization problem. Hence, they can be dropped from the minimization problem. The list of c_{lb} that does not satisfy the above inequality are stored as `lb.can.new`.

```
alpha <- .05
lb.can.new <- NULL
for (x in lb.can) {
  if (mean(lb.can1 <= x) >= 1 - alpha) {
    lb.can.new <- c(lb.can.new, x)
  }
}
```

The number of elements in this new list `lb.can.new` is 11. Similarly, the list of candidates c_{ub} that satisfy

$$\frac{1}{B}\sum_{b=1}^{B}\mathbf{1}\left[-c_{\text{ub}}\leq\sqrt{n}\left(\hat{\theta}_{\text{ub}}^{b}-\hat{\theta}_{\text{ub}}\right)\right]<1-\alpha$$

cannot satisfy the second constraint of the minimization problem. Hence, they can be dropped from the minimization problem. The list of c_{ub} that does not satisfy the above inequality are stored as `ub.can.new`.

```
ub.can.new <- NULL
for (x in ub.can) {
  if (mean(-x <= ub.can1) >= 1 - alpha) {
    ub.can.new <- c(ub.can.new, x)
  }
}
```

The number of elements in this new list `ub.can.new` is 11.

$\star$ This step corresponds to the sixth bullet point of the implementation notes.

Remarks.

- If none of the candidates for the upper (resp. lower) bound satisfy the constraints, then the corresponding upper (resp. lower) bound of the $(1-\alpha)$-confidence interval will be assigned as the logical upper (resp. lower) bound.
- In addition, if the point estimate of the upper (resp. lower) bound is `Inf` (resp `-Inf`), then the upper (resp. lower) bound of the $(1-\alpha)$-confidence interval will be assigned as `Inf` (resp. `-Inf`).

Step 5. Solving the minimization problem

Based on the refined set of candidates obtained in step 4, we should choose $(c_{\text{lb}}, c_{\text{ub}})$ that solves the minimization problem. This is done by choosing $(c_{\text{lb}}, c_{\text{ub}})$ such that

- they satisfy the two constraints, and
- they minimizes $c_{\text{lb}}+c_{\text{ub}}$.

This is done as follows:

```
df.feasible <- data.frame(matrix(vector(), nrow = 0, ncol = 3))
colnames(df.feasible) <- c("lb", "ub", "len")
# Check which candidates satisfy the two constraints
for (x in lb.can.new) {
  for (y in ub.can.new) {
    cons1 <- mean((lb.can1 <= x) * (-y <= ub.can2))
    cons2 <- mean((lb.can2 <= x) * (-y <= ub.can1))
    if ((cons1 >= 1 - alpha) & (cons2 >= 1 - alpha)) {
      df.feasible[nrow(df.feasible) + 1, ] <- c(x, y, x + y)
    }
  }
}

# Choose the bounds that minimize the objective function
c.bd <- df.feasible %>% slice(which.min(len))
c.lb <- c.bd$lb
c.ub <- c.bd$ub
```

⋆ This step corresponds to the fifth bullet point of the implementation notes.

Step 6. Construct the confidence interval

Call the solution to the minimization problem as $c_{\text{lb}}(\alpha)$ and $c_{\text{ub}}(\alpha)$. The $(1-\alpha)$-confidence interval is

$$\left[\hat{\theta}_{\text{lb}} - \frac{c_{\text{lb}}(\alpha)}{\sqrt{n}}, \hat{\theta}_{\text{ub}} + \frac{c_{\text{ub}}(\alpha)}{\sqrt{n}}\right].$$

It is constructed by the following code:

```
bd <- c(lb - c.lb/sqrt(n), ub + c.ub/sqrt(n))
print(bd)
```

```
## [1] 0.6339226 0.8854928
```

This matches with the output from the `chorussell` procedure of the `lpinfer` package:

```
print(r)
```

```
## 95%-confidence interval: [0.63392, 0.88549]
```

Back to the problem again

The bounds are different from the ones posted in issue #104 becaues I have made some changes to the code. However, the same problem still appears where more than half of the bootstrap estimates of the lower bound are smaller than the lower bound of the confidence interval:

```
mean(lb.bs < bd[1])
```

```
## [1] 0.49
```

The largest bound obtained is also smaller than the upper bound of the confidence interval:

```
max(ub.bs)
```

```
## [1] 0.7401446
```

Obtaining the confidence interval without step 4

The solution $(c_{\text{lb}}(\alpha), c_{\text{ub}}(\alpha))$ can be obtained without the step of simplifying the list of candidates by building a two-dimensional grid. Recall that the list of all possible candidates for the upper and lower bounds are stored in `ub.can` and `lb.can` respectively. Hence, the bounds can be constructed as follows:

```
df.feasible.grid <- data.frame(matrix(vector(), nrow = 0, ncol = 3))
colnames(df.feasible.grid) <- c("lb", "ub", "len")
```

```
# Check the candidates through building a two-dimensional grid
for (x in lb.can) {
  for (y in ub.can) {
    cons1 <- mean((lb.can1 <= x) * (-y <= ub.can2))
    cons2 <- mean((lb.can2 <= x) * (-y <= ub.can1))
    if ((cons1 >= 1 - alpha) & (cons2 >= 1 - alpha)) {
      df.feasible.grid[nrow(df.feasible.grid) + 1, ] <- c(x, y, x + y)
    }
  }
}

# Choose the bounds that minimize the objective function
c.bd.grid <- df.feasible.grid %>% slice(which.min(len))
c.lb.grid <- c.bd.grid$lb
c.ub.grid <- c.bd.grid$ub

# Construct the confidence interval
bd.grid <- c(lb - c.lb.grid/sqrt(n), ub + c.ub.grid/sqrt(n))
print(bd.grid)
```

```
## [1] 0.6339226 0.8854928
```

The results here matches what we get from step 6 above (and hence they are the same as the confidence interval from the object `r` in the beginning).

```
print(bd.grid[1] == bd[1])
```

```
## [1] TRUE
```

```
print(bd.grid[2] == bd[2])
```

```
## [1] TRUE
```

Indeed, I have still kept the `remove.const` option in the `chorussell` procedure so that it can compute the solution without the refinement step (i.e. step 4 above) by setting `remove.const = FALSE`. The result can be obtained as follows:

```
set.seed(5)
r.grid <- chorussell(data = df, lpmodel = lpm, ci = TRUE, beta.tgt = .2,
                     progress = FALSE, remove.const = FALSE)
print(r.grid)
```

```
## 95%-confidence interval: [0.63392, 0.88549]
```

The bounds here is the same as the ones in `r`:

```
print(r$ci.df == r.grid$ci.df)
```

```
##      alpha kappa   lb   ub
## [1,]  TRUE  TRUE TRUE TRUE
```

Just to confirm again, the same problems still appear with the results from `r.grid`:

```
print(mean(lb.bs < r.grid$ci.df$lb))
```

```
## [1] 0.49
```

```
print(max(r.grid$ub.bs))
```

```
## [1] 0.7401446
```

Trying other specifications

The same problems still appear when I choose some other specifications in the `chorussell` procedure. For instance, changing `kappa` to a small constant gives:

```r
set.seed(5)
r1 <- chorussell(data = df, lpmodel = lpm, ci = TRUE, progress = FALSE,
                 kappa = 1e-3)
print(r1)
```

```
## 95%-confidence interval: [0.63385, 0.88508]
```

```r
print(mean(r1$lb.bs < r1$ci.df$lb))
```

```
## [1] 0.49
```

```r
print(max(r1$ub.bs))
```

```
## [1] 0.7415425
```

Similar observations can be made if I use the 1-norm instead:

```r
set.seed(5)
r2 <- chorussell(data = df, lpmodel = lpm, ci = TRUE, progress = FALSE,
                 norm = 1)
print(r2)
```

```
## 95%-confidence interval: [0.6454, 0.88952]
```

```r
print(mean(r2$lb.bs < r2$ci.df$lb))
```

```
## [1] 0.71
```

```r
print(max(r2$ub.bs))
```

```
## [1] 0.7313295
```

or if I use the 1-norm with a small `kappa`:

```r
set.seed(5)
r3 <- chorussell(data = df, lpmodel = lpm, ci = TRUE, progress = FALSE,
                 norm = 1, kappa = 1e-3)
print(r3)
```

```
## 95%-confidence interval: [0.64534, 0.89177]
```

```r
print(mean(r3$lb.bs < r3$ci.df$lb))
```

```
## [1] 0.71
```

```r
print(max(r3$ub.bs))
```

```
## [1] 0.7341704
```

Threshold in confidence sets

In the simulation experiments done by Cho and Russell (2019), they introduced a threshold in the length of the identified set, i.e. they use $\Delta_n^\star = \mathbf{1}[\Delta_n > b_n]$ with $b_n = [\log(n)]^{-1/2}$, instead of Δ_n in computing the critical values. They find that this helps to improve the coverage when the model is close to point identification in finite samples. In our example, the value of $\Delta_n^\star$ is:

```r
b.n <- (log(n))^(-1/2)
delta.star <- as.integer(delta > b.n)
print(delta.star)
```

```
## [1] 0
```

If I replace Δ_n by $\Delta_n^\star$, then the only possible candidates for c_{lb} are $\{\sqrt{n}(\hat{\theta}_{\text{lb}}^b - \hat{\theta}_{\text{lb}})\}_{b=1}^B$, whereas the only possible candidates for c_{ub} are $\{-\sqrt{n}(\hat{\theta}_{\text{ub}}^b - \hat{\theta}_{\text{ub}})\}_{b=1}^B$. They are already stored in `lb.can1` and `-ub.can1` above.

Therefore, the confidence interval can be constructed as follows:

```r
df.feasible.trun <- data.frame(matrix(vector(), nrow = 0, ncol = 3))
colnames(df.feasible.trun) <- c("lb", "ub", "len")
# Check the candidates through building a two-dimensional grid
for (x in lb.can1) {
  for (y in (-ub.can1)) {
    cons1 <- mean((lb.can1 <= x))
    cons2 <- mean((-y <= ub.can1))
    if ((cons1 >= 1 - alpha) & (cons2 >= 1 - alpha)) {
      df.feasible.trun[nrow(df.feasible.trun) + 1, ] <- c(x, y, x + y)
    }
  }
}

# Choose the bounds that minimize the objective function
c.bd.trun <- df.feasible.trun %>% slice(which.min(len))
c.lb.trun <- c.bd.trun$lb
c.ub.trun <- c.bd.trun$ub

# Construct the confidence interval
bd.trun <- c(lb - c.lb.trun/sqrt(n), ub + c.ub.trun/sqrt(n))
print(bd.trun)
```

```
## [1] 0.6434097 0.8706896
```

The confidence interval obtained is slightly tigher than the one in `r`, but the same problems still appear:

```r
print(mean(r$lb.bs < bd.trun[1]))
```

```
## [1] 0.56
```

```r
print(max(r$ub.bs))
```

```
## [1] 0.7401446
```

Reference

Cho, J., and T. M. Russell. 2019. "Simple Inference on Functionals of Set-Identified Parameters Defined by Linear Moments." *Working Paper*.