Parametric Bootstrapping of Aster Models

Charles J. Geyer

April 22, 2006

1 Introduction

This note shows how to use the parametric bootstrap to improve confidence intervals for aster models — improving on the "usual" intervals based on maximum likelihood, asymptotic normality, and Fisher information. The simulations used by the parametric bootstrap, produced using the function `raster` in the `aster` package and discussed in the last section (Simulation) of the package vignette, can also be used to check the performance of the "usual" intervals.

2 Preliminaries

We start with a large amount of preliminary computation, setting up and fitting the two aster models involved in Figure 2 of the statistical paper about aster models (`http://www.stat.umn.edu/geyer/aster`). Readers should skip this section and the following section entirely, because they leave out all explanation, only repeating what is done in Appendix D of the technical report found at that URL. Section 4 begins the new material.

```
> library(aster)

Loading required package: trust

> data(echinacea)

> pred <- c(0, 1, 2, 1, 2, 3, 4, 5, 6)
> fam <- c(1, 1, 1, 1, 1, 1, 3, 3, 3)

> vars <- c("ld02", "ld03", "ld04", "fl02", "fl03",
+     "fl04", "hdct02", "hdct03", "hdct04")
```

```
> redata <- reshape(echinacea, varying = list(vars),
+     direction = "long", timevar = "varb", times = as.factor(vars),
+     v.names = "resp")
> redata <- data.frame(redata, root = 1)

> hdct <- grep("hdct", as.character(redata$varb))
> hdct <- is.element(seq(along = redata$varb), hdct)
> redata <- data.frame(redata, hdct = as.integer(hdct))

> level <- gsub("[0-9]", "", as.character(redata$varb))
> redata <- data.frame(redata, level = as.factor(level))

> year <- gsub("[a-z]", "", as.character(redata$varb))
> year <- paste("yr", year, sep = "")
> redata <- data.frame(redata, year = as.factor(year))

> out8 <- aster(resp ~ varb + level:(nsloc + ewloc) +
+     hdct * pop - pop, pred, fam, varb, id, root,
+     data = redata)

> out10 <- aster(resp ~ varb + level:(nsloc + ewloc) +
+     hdct * pop, pred, fam, varb, id, root, data = redata)

> newdata <- data.frame(pop = levels(echinacea$pop))
> for (v in vars) newdata[[v]] <- 1
> newdata$root <- 1
> newdata$ewloc <- 0
> newdata$nsloc <- 0
> renewdata <- reshape(newdata, varying = list(vars),
+     direction = "long", timevar = "varb", times = as.factor(vars),
+     v.names = "resp")
> names(redata)

 [1] "pop"   "ewloc" "nsloc" "varb"  "resp"  "id"    "root"
 [8] "hdct"  "level" "year"

> names(renewdata)

[1] "pop"   "root"  "ewloc" "nsloc" "varb"  "resp"  "id"
```

```
> hdct <- grep("hdct", as.character(renewdata$varb))
> hdct <- is.element(seq(along = renewdata$varb), hdct)
> renewdata$hdct <- as.integer(hdct)
> level <- gsub("[0-9]", "", as.character(renewdata$varb))
> renewdata$level <- as.factor(level)
> year <- gsub("[a-z]", "", as.character(renewdata$varb))
> year <- paste("yr", year, sep = "")
> renewdata$year <- as.factor(year)

> nind <- nrow(newdata)
> nnode <- length(vars)
> amat <- array(0, c(nind, nnode, nind))
> for (i in 1:nind) amat[i, grep("hdct", vars), i] <- 1

> conf.level <- 0.95
> crit <- qnorm((1 + conf.level)/2)

> popnames <- as.character(newdata$pop)
```

3 Plot for the Paper

```
> pout8 <- predict(out8, varvar = varb, idvar = id,
+     root = root, newdata = renewdata, se.fit = TRUE,
+     amat = amat)
> pout10 <- predict(out10, varvar = varb, idvar = id,
+     root = root, newdata = renewdata, se.fit = TRUE,
+     amat = amat)

> doit <- function(list1, list2) {
+     fit8 <- list1$fit
+     y8top <- list1$top
+     y8bot <- list1$bot
+     fit10 <- list2$fit
+     y10top <- list2$top
+     y10bot <- list2$bot
+     i <- seq(along = popnames)
+     foo <- 0.1
+     plot(c(i - 1.5 * foo, i - 1.5 * foo, i + 1.5 *
+         foo, i + 1.5 * foo), c(y8top, y8bot, y10top,
+         y10bot), type = "n", axes = FALSE, xlab = "",
```

```
+         ylab = "")
+     segments(i - 1.5 * foo, y8bot, i - 1.5 * foo,
+         y8top)
+     segments(i - 2.5 * foo, y8bot, i - 0.5 * foo,
+         y8bot)
+     segments(i - 2.5 * foo, y8top, i - 0.5 * foo,
+         y8top)
+     segments(i - 2.5 * foo, fit8, i - 0.5 * foo,
+         fit8)
+     segments(i + 1.5 * foo, y10bot, i + 1.5 * foo,
+         y10top, lty = 2)
+     segments(i + 2.5 * foo, y10bot, i + 0.5 * foo,
+         y10bot)
+     segments(i + 2.5 * foo, y10top, i + 0.5 * foo,
+         y10top)
+     segments(i + 2.5 * foo, fit10, i + 0.5 * foo,
+         fit10)
+     axis(side = 2)
+     title(ylab = "unconditional mean value parameter")
+     axis(side = 1, at = i, labels = popnames)
+     title(xlab = "population")
+ }
```

Figure 1 is produced by the following code

```
> doit(list(fit = pout8$fit, bot = pout8$fit - crit *
+     pout8$se.fit, top = pout8$fit + crit * pout8$se.fit),
+     list(fit = pout10$fit, bot = pout10$fit - crit *
+         pout10$se.fit, top = pout10$fit + crit *
+         pout10$se.fit))
```

It appears on p. 5.

```
> time.start <- proc.time()
```

4 Parametric Bootstrap

Now we come to new material. We would like to redo Figure 1 using the parametric bootstrap to produce the confidence intervals. Note that this will not change the point estimates.

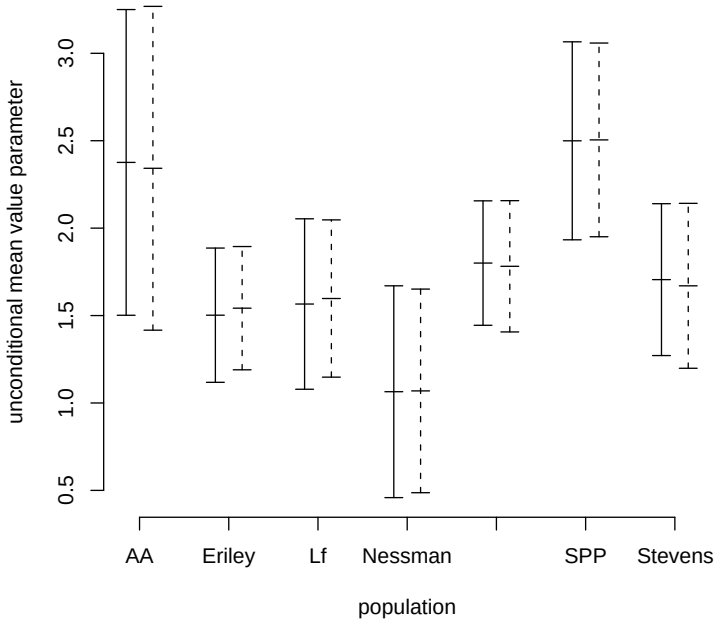

Figure 1: 95% confidence intervals for unconditional mean value parameter for fitness (sum of head count for all years) at each population for a "typical" individual having position zero-zero and having the parameterization of Model Eight (solid bar) or Model Ten (dashed bar). Tick marks in the middle of the bars are the center (the MLE). This is exactly the same as Figure 2 of the aster paper except that where R leaves off one of the labels, we restored it by hand in the figure for the paper.

This section follows the last section of the aster package vignette, which shows how to do what it calls a simulation study. Of course, there is no difference between a simulation study and a parametric bootstrap except that a parametric bootstrap simulates from a distribution given by an estimated parameter value $\hat{\theta}(x)$, whereas a regular simulation study simulates from a parameter value that is pulled out of the air and has no relevance to any particular dataset.

As always when bootstrapping, it pays to bootstrap approximately pivotal quantities. Thus our parametric bootstrap confidence intervals use the same asymptotic standard errors produced from Fisher information that the "usual" intervals use. The only difference is that here we do not assume the MLE are normally distribution and hence do not use 1.96 for 95% critical values. Instead we derive critical values from the actual simulation distribution of standardized estimates (stored in the matrix `z.star` in the code). The resulting confidence intervals are not centered at the MLE unless the simulation distribution happens to be symmetric. These intervals correctly account for any bias and skewness in the estimators (in fact, we know from theory that the estimates are unbiased, so any bias correction done is unimportant, and any asymmetry of the intervals must be due mainly to skewness of the sampling distribution of the estimators).

```
> out <- out8
> pout <- pout8
```

We start with another use of `predict`. The `raster` function wants $\boldsymbol{\theta}$ or, to be more precise, here we use $\hat{\boldsymbol{\theta}}$.

```
> theta.hat <- predict(out, model.type = "cond", parm.type = "canon")
> theta.hat <- matrix(theta.hat, nrow = nrow(out$x),
+     ncol = ncol(out$x))
> fit.hat <- pout$fit
> beta.hat <- out$coefficients
```

We also need root data, and it will be simpler if we actually don't use the forms of the `aster` and `predict.aster` functions that take formulas (because then we don't have to cram the simulated data in a data frame and we avoid a lot of repetitive parsing of the same formulas)

```
> root <- out$root
> modmat <- out$modmat
> modmat.pred <- pout$modmat
```

```
> x.pred <- matrix(1, nrow = dim(modmat.pred)[1], ncol = dim(modmat.pred)[2])
> root.pred <- x.pred
```

Now we're ready for a simulation

```
> set.seed(42)

> nboot <- 9999

> alpha <- 1 - conf.level
> ilow <- (nboot + 1) * alpha/2
> ihig <- (nboot + 1) * (1 - alpha/2)

> z.star <- matrix(NA, nboot, length(fit.hat))
> for (iboot in 1:nboot) {
+     xstar <- raster(theta.hat, pred, fam, root)
+     out.star <- aster(xstar, root, pred, fam, modmat,
+         beta.hat)
+     pout.star <- predict(out.star, x.pred, root.pred,
+         modmat.pred, amat, se.fit = TRUE)
+     z.star[iboot, ] <- (pout.star$fit - fit.hat)/pout.star$se.fit
+ }

> crit.low.8 <- apply(z.star, 2, function(x) sort(x)[ilow])
> crit.hig.8 <- apply(z.star, 2, function(x) sort(x)[ihig])
> crit.low.8

[1] -2.454192 -2.311937 -2.407913 -2.744470 -2.180884 -2.215894
[7] -2.187093

> crit.hig.8

[1] 1.738255 1.761075 1.728866 1.640237 1.817272 1.807484
[7] 1.808636
```

Note that the critical values are not symmetric (`crit.low.8` not just the negatives of `crit.hig.8`) so the resulting confidence intervals will not be centered at the MLE.

Save the simulation distribution for use in the following section.

```
> z.star.8 <- z.star
```

Now redo for Model 10.

```
> out <- out10
> pout <- pout10
> theta.hat <- predict(out, model.type = "cond", parm.type = "canon")
> theta.hat <- matrix(theta.hat, nrow = nrow(out$x),
+     ncol = ncol(out$x))
> fit.hat <- pout$fit
> beta.hat <- out$coefficients
> root <- out$root
> modmat <- out$modmat
> modmat.pred <- pout$modmat
> x.pred <- matrix(1, nrow = dim(modmat.pred)[1], ncol = dim(modmat.pred)[2])
> root.pred <- x.pred
> set.seed(42)
> z.star <- matrix(NA, nboot, length(fit.hat))
> for (iboot in 1:nboot) {
+     xstar <- raster(theta.hat, pred, fam, root)
+     out.star <- aster(xstar, root, pred, fam, modmat,
+         beta.hat)
+     pout.star <- predict(out.star, x.pred, root.pred,
+         modmat.pred, amat, se.fit = TRUE)
+     z.star[iboot, ] <- (pout.star$fit - fit.hat)/pout.star$se.fit
+ }

> crit.low.10 <- apply(z.star, 2, function(x) sort(x)[ilow])
> crit.hig.10 <- apply(z.star, 2, function(x) sort(x)[ihig])
> crit.low.10

[1] -2.413138 -2.250537 -2.231591 -2.701213 -2.147594 -2.194846
[7] -2.241582

> crit.hig.10

[1] 1.765979 1.816879 1.778429 1.631660 1.804176 1.857021
[7] 1.773137
```

Figure 2 is produced by the following code

```
> doit(list(fit = pout8$fit, bot = pout8$fit - crit.hig.8 *
+     pout8$se.fit, top = pout8$fit - crit.low.8 *
+     pout8$se.fit), list(fit = pout10$fit, bot = pout10$fit -
+     crit.hig.10 * pout10$se.fit, top = pout10$fit -
+     crit.low.10 * pout10$se.fit))
```

It appears on p. 9.

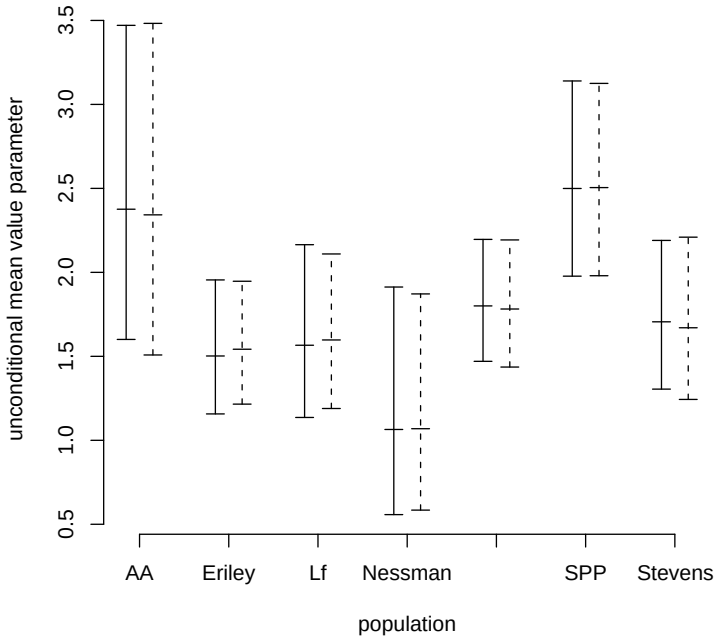

Figure 2: 95% confidence intervals for unconditional mean value parameter for fitness (sum of head count for all years) at each population for a "typical" individual having position zero-zero and having the parameterization of Model Eight (solid bar) or Model Ten (dashed bar). Tick marks in the middle of the bars are the center (the MLE). This is exactly the same as Figure 1 except that critical values for the confidence interval are calculated by the parametric bootstrap.

How much time?

```
> time.pboot <- proc.time()
> delta <- time.pboot[1] - time.start[1]

> delta.hour <- floor(delta/60^2)
> delta.minute <- floor((delta - 60^2 * delta.hour)/60)
> delta.second <- (delta - 60 * (delta.minute + 60 *
+     delta.hour))/60
```

Our parametric bootstrap estimation, including the picture, took 7 hours, 51 minutes, and 0.2 seconds.

5 Performance of the Usual Asymptotics

Our parametric bootstrap simulations can be used as a simulation study of the performance of the intervals based on the "usual asymptotics" which are based on the asymptotically standard normal quantity stored in `z.star` being exactly standard normal.

```
> apply(z.star.8, 2, function(z) mean(abs(z) < abs(qnorm(alpha/2))))

[1] 0.9353935 0.9409941 0.9402940 0.9233923 0.9459946 0.9424942
[7] 0.9443944

> apply(z.star, 2, function(z) mean(abs(z) < abs(qnorm(alpha/2))))

[1] 0.9370937 0.9434943 0.9410941 0.9265927 0.9456946 0.9425943
[7] 0.9426943
```

Oops! Before we get too pleased with this performance, we should consider whether a bias and skewness corrected interval might not do better. That is we should consider separately the probabilities of missing low and missing high.

```
> apply(z.star.8, 2, function(z) mean(z < qnorm(alpha/2)))

[1] 0.05160516 0.04530453 0.04760476 0.06800680 0.03720372
[6] 0.04060406 0.03900390

> apply(z.star, 2, function(z) mean(z < qnorm(alpha/2)))
```

```
[1] 0.04820482 0.04090409 0.04260426 0.06500650 0.03750375
[6] 0.03780378 0.04260426

> apply(z.star.8, 2, function(z) mean(z > qnorm(1 -
+     alpha/2)))

[1] 0.01300130 0.01370137 0.01210121 0.00860086 0.01680168
[6] 0.01690169 0.01660166

> apply(z.star, 2, function(z) mean(z > qnorm(1 - alpha/2)))

[1] 0.01470147 0.01560156 0.01630163 0.00840084 0.01680168
[6] 0.01960196 0.01470147
```

We see that although the "usual" intervals have fairly close to the correct coverage, the do so in the wrong way. Far from being equal tailed, they mostly miss low. Thus the parametric bootstrap intervals are better in the sense that they more closely perform as designed.