Stat 8931 (Aster Models)
Lecture Slides Deck 7
Parametric Bootstrap

Charles J. Geyer

School of Statistics
University of Minnesota

October 1, 2018

R and License

- The version of R used to make these slides is 3.5.1.
- The version of R package `aster` used to make these slides is 1.0.2.

Simulation

What cannot be done by theory can be done by brute force and ignorance.

If you can simulate a probability model, then you can calculate (approximately) any probability or expectation with respect to that model by simulating the model and averaging over simulations.

The central limit theorem guarantees that eventually the error of the approximation is about $\sigma/\sqrt{n}$, where σ is the standard deviation of random variable whose expectation is being approximated and n is the number of simulations.

Simulation (cont.)

More precisely, suppose we are trying to calculate

$$\psi = E\{g(X)\} = \int g(x) f(x)\, dx$$

where f is the PDF of X (or the same with the integral replaced by a sum if X is discrete).

Suppose X_1, X_2, ... are IID simulations from the distribution of X.

Then

$$\hat{\psi}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i)$$

is the obvious approximation of ψ based on these simulations.

Simulation (cont.)

$$\hat{\psi}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i)$$

And

$$\hat{\sigma}_n^2 = \frac{1}{n} \sum_{i=1}^{n} \left[g(X_i) - \hat{\psi}_n \right]^2$$

is the obvious approximation of σ based on these simulations.

And the central limit theorem and Slutsky's theorem say

$$\frac{\hat{\psi}_n - \psi}{\hat{\sigma}_n / \sqrt{n}}$$

is approximately standard normal for large n.

Simulation (cont.)

This may look a little weird, but is straight out of intro stats. Define

$$Y = g(X)$$

and

$$Y_i = g(X_i), \qquad i = 1, 2, \ldots$$

then

- $\psi = E(Y)$ is the population mean of the distribution of Y,
- $\hat{\psi}_n = \overline{Y}_n$ is the sample mean of the Y's, and
- $\hat{\sigma}_n$ is the sample standard deviation of the Y's.

Relevant and Irrelevant Simulation

The "relevant" and "irrelevant" are my own eccentric way of talking about this subject. No one else says this.

There are two kinds of simulation that statisticians and other scientists do.

One kind — usually called just **simulation** — uses some toy model having no relevance to any particular application. The idea is that one can transfer conclusions for the toy model to real applications, but this is very questionable because the toy model will differ in many respects from the models used in real applications.

How can we know whether the toy models were chosen (consciously or unconsciously) specifically to make the author's methods look good? We can't.

I call this **irrelevant simulation**. But many authors like it. Most statistical papers have such. Many scientific papers have such.

Relevant and Irrelevant Simulation (cont.)

The other kind — usually called **the bootstrap** — uses the actual model from the actual application, which means that every user has to do their own "simulation study".

That is annoying, because it makes work for every user. We don't just have the simulation done once and for all by the original authors (on toy models, hence irrelevant).

But the real model from the real application is a **family of probability distributions** and we don't know the **true unknown parameter value**, so we don't actually know the precisely correct probability distribution to simulate from.

So we use our best guess (point estimate) of the true unknown parameter value to simulate from.

The Bootstrap

So the term **bootstrap** implies two ideas.

- We calculate by simulation rather than theory.
- We simulate from our best guess about the true unknown distribution of the data.

The bootstrap then divides into two large categories

- The **nonparametric bootstrap** assumes the data are independent and identically distributed and simulates from the **empirical distribution** $\widehat{P}_n$ of the data.
- The **parametric bootstrap** assumes the data follow a parametric model and simulates from the distribution indexed by our estimate $\hat{\psi}_n$ of the true unknown parameter ψ.

The Bootstrap (cont.)

Both kinds of bootstrap do close to but not exactly the right thing.

The empirical distribution $\widehat{P}_n$ is not the true unknown distribution of the data P.

The estimated distribution $P_{\hat{\psi}_n}$ is not the true unknown distribution P_ψ of the data.

But the estimates will be close for large sample size n.

The bootstrap is not (contrary to widespread popular opinion) an exact, small-sample methodology.

The Nonparametric Bootstrap

Many people, having heard about the nonparametric bootstrap, want to use that.

It's gotta be better. It's nonparametric! It doesn't depend on any model assumptions!

But the nonparametric bootstrap comes with many issues the parametric bootstrap doesn't have.

- It doesn't do hypothesis tests (at least not easily).
- It doesn't do regression models (at least not easily).
- If the estimators being bootstrapped are based on a parametric model — MLE for example — then they have no nonparametric interpretation anyway.

Nonparametric Bootstrap and Hypothesis Tests

In a test of statistical hypotheses, the P-value is based on simulation **under the null hypothesis**.

The empirical distribution estimates the true unknown distribution of the data, which is **not in the null hypothesis** when the **null hypothesis is false**.

Hence naive use of the nonparametric bootstrap for hypothesis tests (simulate from the empirical distribution, calculate the test statistic for both the real data and the simulations, take the P-value to be the fraction of simulated values of the test statistic that exceed the value for the real data) is completely bogus. This gives a test with the correct level but no power.

Nonparametric Bootstrap and Hypothesis Tests (cont.)

There are two correct ideas of how to do hypothesis tests with the nonparametric bootstrap.

Sometimes, in simple situations, one can cook up a nonparametric estimate (not the empirical distribution) of a distribution in the null hypothesis. Already we have left the straightforward nonparametric bootstrap. We need to use a tricky modified nonparametric bootstrap with a different trick for every application (and for most complicated applications — such as aster models — there will be no such tricks).

If the test is about a **single parameter of interest** one can always do a bootstrap confidence interval and then base a test on that (reject $H_0 : \psi = \psi_0$ at level α if a $1 - \alpha$ confidence interval does not contain ψ_0).

Nonparametric Bootstrap and Regression

In a regression model we observe pairs

$$(x_i, y_i), \qquad i = 1, 2, \ldots$$

where x_i (a vector) is the **predictor** and y_i (a scalar) is the **response**.

There are two distributions of interest.

- The joint distribution of the (x_i, y_i) which are assumed IID.
- The conditional distribution of y_i given x_i which is different for each x_i.

In regression situations we are usually interested in the latter.

Nonparametric Bootstrap and Regression (cont.)

This leads to two ideas about how to bootstrap regression.

If one is interested in studying the joint distribution, **bootstrap cases**. That is, simulate from the joint empirical distribution of the (x_i, y_i) pairs.

This simulates the joint distribution and cannot draw inference about the conditional distribution.

Nonparametric Bootstrap and Regression (cont.)

If one is interested in studying the conditional distribution, **bootstrap residuals**. This needs more structure. Assume

$$y_i = g(x_i) + e_i, \qquad i = 1, \ldots, n \tag{$*$}$$

where g is an arbitrary function (the **regression function**) and the e_i are IID mean-zero (but not necessarily normal). Let $\hat{g}$ denote the estimator of the regression function (perhaps a nonparametric estimate from some "smoothing" method). Define the **residuals**

$$\hat{e}_i = y_i - \hat{g}(x_i), \qquad i = 1, \ldots, n$$

The residuals $\hat{e}_i$ are not the errors e_i, they are only estimates of them. The residuals are not IID even though the errors are.

Nonparametric Bootstrap and Regression (cont.)

$$y_i = g(x_i) + e_i, \qquad i = 1, \ldots, n \qquad (*)$$

Nevertheless, the method of bootstrapping residuals treats the residuals as IID and simulates new errors e_i^* from the empirical distribution of the residuals and forms bootstrap data

$$y_i^* = \hat{g}(x_i) + e_i^*, \qquad i = 1, \ldots, n$$

Nonparametric Bootstrap and Regression (cont.)

$$y_i = g(x_i) + e_i, \qquad i = 1, \ldots, n \tag{$*$}$$

Bootstrapping residuals has a lot of issues

- If the method of estimating $\hat{g}$ is parametric, then it isn't completely nonparametric.
- In GLM and aster models and other complicated parametric regression models there are no IID errors like in $(*)$ so the method has nowhere to start.

Nevertheless bootstrapping residuals is the only available method for inference about the conditional distribution of the response given the predictor, which is usually what is wanted.

The Parametric Bootstrap

For all these reasons we only recommend the parametric bootstrap for aster models.

The parametric bootstrap has none of the problems of the nonparametric bootstrap.

- No problem with hypothesis tests (simulate from the MLE for the null hypothesis).
- No problem with regression (simulate from the MLE for the regression model).
- It is more accurate than the nonparametric bootstrap when the statistical model is correct.

Its only issue is when the statistical model is wrong. But then you have more worries since all your estimators are wrong too. The nonparametric bootstrap couldn't fix that even if it didn't have all the problems discussed above.

Example One Re-revisited

We revisit example one yet again. We fit this model

```
> aout <- aster(resp ~ varb + layer : (nsloc + ewloc) +
+     fit : pop, pred, fam, varb, id, root, data = redata)
```

and then did prediction like this

```
> pout.amat <- predict(aout, newdata = renewdata,
+     varvar = varb, idvar = id, root = root,
+     se.fit = TRUE, amat = amat)
```

Example One Re-revisited (cont.)

And got this table.

```
> foo <- cbind(pout.amat$fit, pout.amat$se.fit)
> rownames(foo) <- as.character(fred$pop)
> colnames(foo) <- c("estimates", "std. err.")
> round(foo, 3)
```

	estimates	std. err.
AA	2.376	0.446
Eriley	1.502	0.196
Lf	1.566	0.249
Nessman	1.064	0.309
NWLF	1.800	0.182
SPP	2.499	0.289
Stevens	1.706	0.222

Now we want to parametric bootstrap these confidence intervals.

Example One Re-revisited (cont.)

We follow the `aster` package vignette `tutor.pdf`.

The R function that simulates aster models is called `raster`. It wants the model specified by conditional canonical parameters. and it wants θ as a matrix rather than as a vector (both more bad design of R package `aster`)

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

Example One Re-revisited (cont.)

We also need data for initial nodes

```
> root <- matrix(1, nrow = nrow(theta.hat),
+     ncol = ncol(theta.hat))
```

and now we can simulate data like this

```
> resp.star <- raster(theta.hat, pred, fam, root)
```

Example One Re-revisited (cont.)

It turns out to be simpler and faster if we fit aster models to the simulated data not using formulas (using the function `aster.default`). For this we need a model matrix, and also (just for efficiency) we record the estimate of the regression coefficients to use as a starting point for the aster fitting optimization.

```
> modmat <- aout$modmat
> beta.hat <- aout$coefficients
```

and then we fit the aster model to the simulated data as follows

```
> aout.star <- aster(resp.star, root, pred, fam, modmat,
+     beta.hat)
```

Example One Re-revisited (cont.)

Now to make our "predictions" (estimate expected fitness). We also don't use formulas (cannot, since we didn't in fitting). And for this we need some more stuff.

```
> modmat.pred <- pout.amat$modmat
> root.pred <- matrix(1, nrow = dim(modmat.pred)[1],
+     ncol = dim(modmat.pred)[2])
> resp.pred <- root.pred
```

Now predict

```
> pout.amat.star <- predict(aout.star, resp.pred,
+     root.pred, modmat.pred, amat, se.fit = TRUE)
```

(We do not actually need `resp.pred` — it is ignored — but there is such an argument, so we supply it.)

Example One Re-revisited (cont.)

So that is the pattern, with the setup we have

```
> resp.star <- raster(theta.hat, pred, fam, root)
> aout.star <- aster(resp.star, root, pred, fam, modmat,
+     beta.hat)
> pout.amat.star <- predict(aout.star, resp.pred,
+     root.pred, modmat.pred, amat, se.fit = TRUE)
```

simulates new data from the model, fits the MLE for the new data, and "predicts" expected fitness for the seven populations.

All that remains is to wrap this in a loop.

Example One Re-revisited (cont.)

To make the simulation repeatable we set the seeds of the random number generator

```
> set.seed(42)
```

If this statement is removed, we get different results every time this file is run.

Example One Re-revisited (cont.)

```
> nboot <- 200
> npop <- length(pout.amat$fit)
> woof <- suppressWarnings(try(load("boot1.rda"),
+     silent = TRUE))
> if (inherits(woof, "try-error")) {
+ mu.star <- matrix(NA, nboot, npop)
+ mu.se.star <- matrix(NA, nboot, npop)
+ for (iboot in 1:nboot) {
+     resp.star <- raster(theta.hat, pred, fam, root)
+     aout.star <- aster(resp.star, root, pred, fam,
+         modmat, beta.hat)
+     pout.amat.star <- predict(aout.star, resp.pred,
+         root.pred, modmat.pred, amat, se.fit = TRUE)
+     mu.star[iboot, ] <- pout.amat.star$fit
+     mu.se.star[iboot, ] <- pout.amat.star$se.fit
+ }
+ save(mu.star, mu.se.star, file = "boot1.rda")
+ }
```

Example One Re-revisited (cont.)

That was either the hard part or the easy part (not sure which).

Bootstrapping is unfamiliar and time consuming, but we just followed the vignette.

All that is left is turning the bootstrap samples into confidence intervals. The computations are trivial but the theory is hard.

Many, many methods of doing bootstrap confidence intervals have been proposed. Most are for the nonparametric bootstrap.

But the same principles apply to the parametric bootstrap.

We illustrate just two

- bootstrap percentile intervals
- bootstrap t intervals

Bootstrap Percentile Intervals

These intervals are very simple to do (their theory is complicated).

If we have bootstrap estimators θ^*_1, ..., θ^*_{nboot} the interval between the $\alpha/2$ and $1 - \alpha/2$ sample quantiles is the $100(1 - \alpha)\%$ **bootstrap percentile interval**.

```
conf.level <- 0.95
probs <- (1 + c(-1, 1) * conf.level) / 2
quantile(theta.star, probs = probs)
```

Easy to do, but ....

Bootstrap Percentile Intervals (cont.)

Bootstrap percentile intervals were invented by Brad Efron, who is also the inventor of the bootstrap.

Peter Hall, who besides Brad Efron is perhaps the next most famous authority about the bootstrap, has said doing percentile intervals is like "looking up [in] the wrong statistical tables backwards."

Here's what he is talking about. Suppose the bootstrap distribution is highly skewed with heavy left tail. That is, θ^* is less than $\hat{\theta}$, sometimes a lot less, with high probability.

The bootstrap analogy suggests that $\hat{\theta}$ is less than θ (the true unknown parameter value), sometimes a lot less, with high probability.

To correct for this our confidence interval should be skewed in the other direction: longer to the right of $\hat{\theta}$ than to the left.

Bootstrap Percentile Intervals (cont.)

Efron is not stupid. There is an argument for percentile intervals.

Suppose there is a monotone, symmetrizing, and variance stabilizing transformation g, that is,

$$\hat{\psi} = g(\hat{\theta})$$

is symmetrically distributed, centered at $\psi = g(\theta)$, and the variance of $\hat{\psi}$ does not depend on ψ.

Then, because of the symmetry and variance stabilizing assumptions, the percentile interval for ψ makes sense.

But, because the transformation is monotone, the interval for ψ can be mapped to an interval for θ, which is the percentile interval.

Bootstrap Percentile Intervals (cont.)

The argument is a bit weird. The monotone, symmetrizing, and variance stabilizing transformation, just needs to exist.

We do not need to know what it is or use it in any way.

If it exists, then the percentile interval for θ does the right thing. If no such transformation exists, the percentile interval does the wrong thing.

Bootstrap Percentile Intervals (cont.)

But even if it is the right thing, it is not second order correct (like some other bootstrap intervals, including bootstrap t intervals).

To repeat, the bootstrap is not an exact small-sample procedure. So these confidence intervals cannot be exact.

The coverage probability is something like

$$1 - \alpha + O(n^{-1/2})$$

called **first order correct**. Better intervals have coverage probability

$$1 - \alpha + O(n^{-1})$$

called **second order correct**.

First order correct is only as good as the "usual asymptotics of maximum likelihood". Second order correct is better.

Bootstrap t Intervals

In contrast to the weird bootstrap percentile intervals, the bootstrap t intervals are very familiar. The work just like Student t confidence intervals except we replace the Student t distribution with the bootstrap distribution.

Bootstrap t Intervals (cont.)

Suppose $\hat{s}$ is an estimator of the standard error of $\hat{\theta}$, and s^* are the corresponding bootstrap estimators of standard error (like the ones we have stored in `mu.se.star`). Then

$$z = \frac{\hat{\theta} - \theta}{\hat{s}}$$

is approximately standard normal, so

$$\hat{\theta} \pm z_{\alpha/2}\hat{s}$$

is an approximate $100(1-\alpha)\%$ confidence interval for θ, where $z_{\alpha/2}$ denotes the upper $\alpha/2$ quantile of the standard normal distribution.

Bootstrap t Intervals (cont.)

But we want to do better than that. The bootstrap analog

$$z^* = \frac{\theta^* - \hat{\theta}}{s^*}$$

should also be approximately standard normal, but we don't need to know that, because we have simulated its whole distribution. Let

$$z^*_{\alpha/2} \qquad \text{and} \qquad z^*_{1-\alpha/2}$$

be the $\alpha/2$ and $1 - \alpha/2$ quantiles of the bootstrap distribution of z^* (note that these will not be $\pm$ something because, unlike the normal distribution, this bootstrap distribution distribution is not exactly symmetric about zero).

Bootstrap t Intervals (cont.)

So now we have

$$z^*_{\alpha/2} < \frac{\theta^* - \hat{\theta}}{s^*} < z^*_{1-\alpha/2}$$

with (approximate) probability $1 - \alpha$. And by the bootstrap analogy

$$z^*_{\alpha/2} < \frac{\hat{\theta} - \theta}{\hat{s}} < z^*_{1-\alpha/2}$$

Inverting these inequalities gives

$$\hat{\theta} - z^*_{1-\alpha/2}\hat{s} < \theta < \hat{\theta} - z^*_{\alpha/2}\hat{s}$$

(we don't have $\pm$ but typically $z^*_{1-\alpha/2}$ is plus and $z^*_{\alpha/2}$ is minus).

Bootstrap t Intervals (cont.)

$$\hat{\theta} - z^*_{1-\alpha/2}\hat{s} < \theta < \hat{\theta} - z^*_{\alpha/2}\hat{s}$$

Bootstrap t intervals do do the right thing when the distribution of $\hat{\theta}$ is noticeably skewed or biased because the quantiles switched ends when we inverted the pivotal quantity z to make the confidence interval.

Example One Re-revisited (cont.)

Bootstrap percentile intervals.

```
> conf.level <- 0.95
> probs <- (1 + c(-1, 1) * conf.level) / 2
> percentile <- apply(mu.star, 2, quantile,
+     probs = probs)
> percentile <- t(percentile)
```

Example One Re-revisited (cont.)

```
> mu.hat <- pout.amat$fit
> mu.se.hat <- pout.amat$se.fit
> z.star <- mu.star
> z.star <- sweep(z.star, 2, mu.hat)
> z.star <- z.star / mu.se.star
> crit <- apply(z.star, 2, quantile,
+     probs = rev(probs))
> crit <- t(crit)
> bootT <- sweep(crit, 1, mu.se.hat, "*")
> bootT <- sweep(bootT, 1, mu.hat)
> bootT <- (- bootT)
```

Example One Re-revisited (cont.)

And for comparison we do the usual normal intervals.

```
> zcrit <- qnorm((1 + conf.level) / 2)
> normint <- cbind(mu.hat - zcrit * mu.se.hat,
+     mu.hat + zcrit * mu.se.hat)
```

Example One Re-revisited (cont.)

```
> goo <- cbind(percentile, bootT, normint)
> rownames(goo) <- rownames(foo)
> colnames(goo) <- c("pct low", "pct hig",
+     "t low", "t hig", "z low", "z hig")
> round(goo, 2)
```

	pct low	pct hig	t low	t hig	z low	z hig
AA	1.60	3.25	1.62	3.32	1.50	3.25
Eriley	1.17	1.92	1.14	1.88	1.12	1.89
Lf	1.09	2.04	1.16	2.15	1.08	2.05
Nessman	0.56	1.66	0.58	1.83	0.46	1.67
NWLF	1.49	2.20	1.44	2.15	1.44	2.16
SPP	1.98	3.05	2.00	3.09	1.93	3.07
Stevens	1.33	2.11	1.35	2.14	1.27	2.14

Random Effects

Now we switch from the `aster` package vignette `tutor.pdf` to the aster models with random effects tech report (TR 692).

Everything is much the same except that we need to simulate random effects (using the function `rnorm`) as well as the conditional distribution of the response given the effects (using `raster`).

Random Effects (cont.)

Recall that in deck 6 we fit the following model.

```
> data(radish)
> pred <- c(0,1,2)
> fam <- c(1,3,2)
> rout <- reaster(resp ~ varb + fit : (Site * Region),
+     list(block = ~ 0 + fit : Block,
+         pop = ~ 0 + fit : Pop),
+     pred, fam, varb, id, root, data = radish)
```

Now we want to parametric bootstrap it.

Random Effects (cont.)

First we store (approximate) maximum likelihood estimates

```
> names(rout)

 [1] "obj"        "fixed"      "random"
 [4] "dropped"    "sigma"      "nu"
 [7] "c"          "b"          "alpha"
[10] "zwz"        "response"   "origin"
[13] "iterations" "counts"     "deviance"
[16] "formula"    "call"

> alpha.hat <- rout$alpha
> sigma.hat <- rout$sigma
> nu.hat <- rout$nu
> b.hat <- rout$b
> c.hat <- rout$c
```

Random Effects (cont.)

Standard errors are only calculated by the `summary.reaster` function, so we look in the object it returns for them

```
> sout <- summary(rout)
> se.alpha.hat <- sout$alpha[ , "Std. Error"]
> se.sigma.hat <- sout$sigma[ , "Std. Error"]
> se.nu.hat <- sout$nu[ , "Std. Error"]
```

Random Effects (cont.)

Then we collect all the model matrices into one model matrix.

```
> fixed <- rout$fixed
> random <- rout$random
> modmat.tot <- cbind(fixed, Reduce(cbind, random))
```

And we make the matrix $\widehat{A}$ that is the square root of the variance matrix of the random effects

```
> nfix <- ncol(fixed)
> nrand <- sapply(random, ncol)
> a.hat <- rep(sigma.hat, times = nrand)
```

Actually `a.hat` is the diagonal of the (diagonal) matrix $\widehat{A}$.

Random Effects (cont.)

To simulate new aster data we first need to change from unconditional canonical parameters to conditional canonical parameters (because that's what the R function `raster` requires).

```
> c.star <- rnorm(sum(nrand))
> b.star <- a.hat * c.star
> eff.star <- c(alpha.hat, b.star)
> phi.star <- as.numeric(as.vector(rout$obj$origin) +
+     modmat.tot %*% eff.star)
> theta.star <- astertransform(phi.star, rout$obj,
+     to.cond = "conditional", to.mean = "canonical")
> y.star <- raster(theta.star, pred, fam, rout$obj$root)
> y.star <- as.vector(y.star)
```

Random Effects (cont.)

Now we need to redo the above analysis on the new data. We can take the simulation truth as starting values.

```
> rout.star <- reaster(y.star ~ varb + fit:(Site * Region),
+     list(block = ~ 0 + fit:Block, pop = ~ 0 + fit:Pop),
+     pred, fam, varb, id, root, data = radish,
+     effects = c(alpha.hat, c.star), sigma = sigma.hat)
> sout.star <- summary(rout.star)
```

Random Effects (cont.)

```
> print(sout.star)

Call:
reaster.formula(fixed = y.star ~ varb + fit:(Site * Region),
    random = list(block = ~0 + fit:Block, pop = ~0 + fit:Pop),
    pred = pred, fam = fam, varvar = varb, idvar = id, root = root,
    data = radish, effects = c(alpha.hat, c.star), sigma = sigma.hat)


Fixed Effects:
                            Estimate Std. Error  z value Pr(>|z|)
(Intercept)                -465.37520    3.41630 -136.222   <2e-16 ***
varbFlowers                 472.27159    3.41731  138.200   <2e-16 ***
varbFruits                  464.40007    3.41931  135.817   <2e-16 ***
fit:SitePoint Reyes           0.04351    0.16525    0.263    0.792
fit:RegionS                  -0.09240    0.09720   -0.951    0.342
fit:SiteRiverside:RegionS     0.50187    0.01262   39.763   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Square Roots of Variance Components (P-values are one-tailed):
      Estimate Std. Error z value Pr(>|z|)/2
block  0.26083    0.05863   4.449   4.32e-06 ***
pop    0.11846    0.03604   3.287   0.000507 ***
---
```

Random Effects (cont.)

So that is the pattern. We are now ready to bootstrap. Because the whole code for the bootstrap won't fit on a slide, we break it up into functions. More straightforward coding is shown in TR 692.

```
> extract <- function(rout) {
+     stopifnot(inherits(rout, "reaster"))
+     sout <- suppressWarnings(summary(rout))
+     c(rout$alpha, rout$sigma, rout$nu,
+         sout.star$alpha[ , "Std. Error"],
+         sout.star$sigma[ , "Std. Error"],
+         sout.star$nu[ , "Std. Error"])
+ }
> e <- extract(rout.star)
```

Random Effects (cont.)

This function simulates new parametric bootstrap data

```
> generate <- function() {
+     c.star <- rnorm(sum(nrand))
+     b.star <- a.hat * c.star
+     eff.star <- c(alpha.hat, b.star)
+     phi.star <- as.numeric(as.vector(rout$obj$origin) +
+         modmat.tot %*% eff.star)
+     theta.star <- astertransform(phi.star, rout$obj,
+         to.cond = "conditional", to.mean = "canonical")
+     y.star <- raster(theta.star, pred, fam,
+         rout$obj$root)
+     as.vector(y.star)
+ }
```

Random Effects (cont.)

```
> nboot <- 199
> woof <- suppressWarnings(try(load("boot2.rda"),
+     silent = TRUE))
> if (inherits(woof, "try-error")) {
+     e.star <- matrix(NaN, nboot, length(e))
+     for (iboot in 1:nboot) {
+         y.star <- generate()
+         rout.star <- reaster(y.star ~ varb + fit:(Site*Region)
+             list(block = ~ 0 + fit:Block, pop = ~ 0 + fit:Pop)
+             pred, fam, varb, id, root, data = radish,
+             effects = c(alpha.hat, c.star), sigma = sigma.hat)
+         e.star[iboot, ] <- extract(rout.star)
+      }
+      save(e.star, file = "boot2.rda")
+ }
```

Random Effects (cont.)

Now we take the results apart.

```
> colnames(e.star) <- names(e)
> nparm <- ncol(e.star) / 2
> se.star <- e.star[ , seq(nparm + 1, 2 * nparm)]
> e.star <- e.star[ , seq(1, nparm)]
> se.hat <- e[seq(nparm + 1, 2 * nparm)]
> e.hat <- e[seq(1, nparm)]
```

Random Effects (cont.)

We are particularly interested in the region-site interaction parameter.

```
> theta.hat <- e.hat["fit:SiteRiverside:RegionS"]
> se.theta.hat <- se.hat["fit:SiteRiverside:RegionS"]
> theta.star <- e.star[ , "fit:SiteRiverside:RegionS"]
> se.theta.star <- se.star[ , "fit:SiteRiverside:RegionS"]
> sum(! is.finite(theta.star))

[1] 0

> sum(! is.finite(se.theta.star))

[1] 0
```

Random Effects (cont.)

The following code makes the figure on the next slide.

```
> hist(theta.star)
> abline(v = theta.hat, col = "red", lwd = 2)
```

Random Effects (cont.)

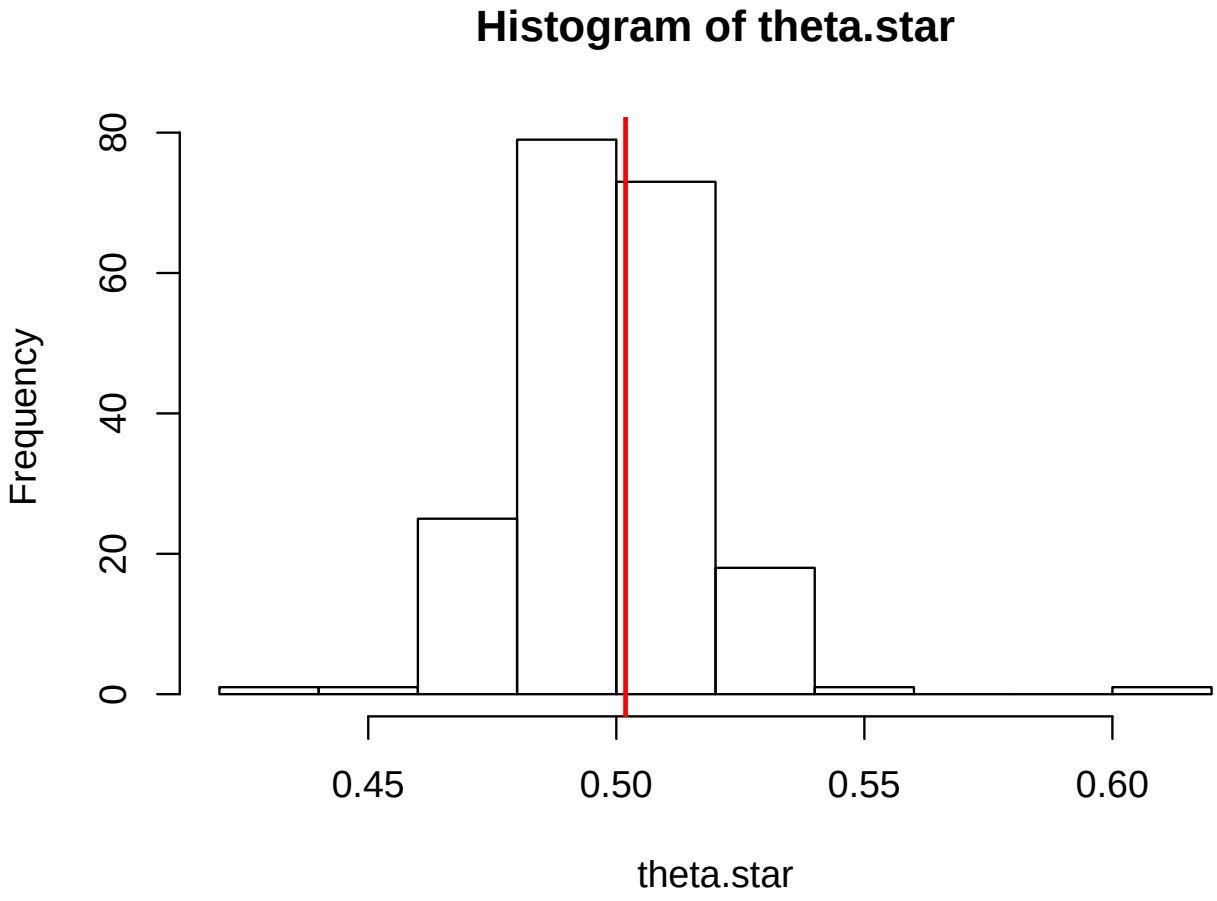

Figure: Bootstrap distribution of region-site interaction parameter. Red vertical line shows estimate for actual data.

Random Effects (cont.)

The histogram looks fairly normal and not too biased. But we compare the two kinds of bootstrap confidence intervals we know about with the asymptotic intervals printed out by the `summary` command.

First asymptotic.

```
> conf.level <- 0.95
> perc <- (1 + c(-1, 1) * conf.level) / 2
> perc
```

```
[1] 0.025 0.975
```

```
> normint <- theta.hat + qnorm(perc) * se.theta.hat
> normint
```

```
[1] 0.4771325 0.5266078
```

Random Effects (cont.)

Next percentile.

```
> percint <- sort(theta.star)[(nboot + 1) * perc]
> percint

[1] 0.4659336 0.5342814
```

Random Effects (cont.)

Next bootstrap t.

```
> z.star <- (theta.star - theta.hat) / se.theta.star
> c.star <- sort(z.star)[(nboot + 1) * perc]
> tint <- theta.hat - rev(c.star) * se.theta.hat
> tint

[1] 0.4803511 0.5257298
```

Random Effects (cont.)

```
> foo <- rbind(normint, percint, tint)
> rownames(foo) <- c("asymptotic", "percentile", "bootstrap
> colnames(foo) <- c("low", "high")
> round(foo, 4)

                low   high
asymptotic  0.4771 0.5266
percentile  0.4659 0.5343
bootstrap t 0.4804 0.5257
```

Seems the asymptotic (with all its approximations!) is at least in the ballpark. Maybe a little too narrow.

But that is just for this one parameter in just this one example.

Random Effects (cont.)

Now for variance components and their square roots.

```
> foo <- e.hat[names(e.hat) == "pop"]
> foo

      pop         pop
0.11845965 0.01403269

> sigma.hat <- foo[1]
> nu.hat <- foo[2]
> foo <- se.hat[names(e.hat) == "pop"]
> se.sigma.hat <- foo[1]
> se.nu.hat <- foo[2]
> all.equal(sigma.hat^2, nu.hat)

[1] TRUE
```

Random Effects (cont.)

```
> foo <- e.star[ , names(e.hat) == "pop"]
> sigma.star <- foo[ , 1]
> nu.star <- foo[ , 2]
> foo <- se.star[ , names(e.hat) == "pop"]
> se.sigma.star <- foo[ , 1]
> se.nu.star <- foo[ , 2]
```

Random Effects (cont.)

The following code makes the figure on the next slide.

```
> hist(sigma.star)
> abline(v = sigma.hat, col = "red", lwd = 2)
```

Random Effects (cont.)

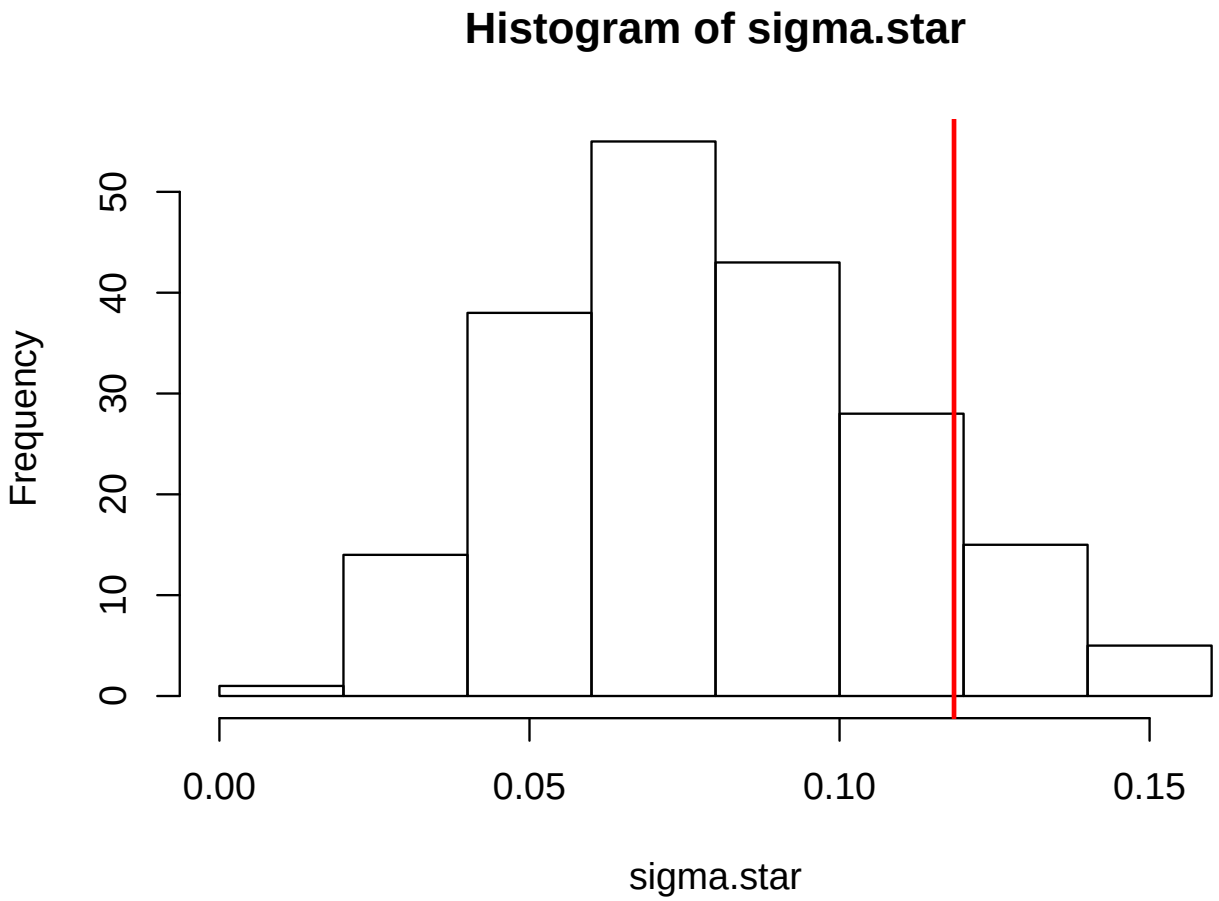

Figure: Bootstrap distribution of square root of variance component for population random effects. Red vertical line shows estimate for actual data.

Random Effects (cont.)

The following code makes the figure on the next slide.

```
> hist(nu.star)
> abline(v = nu.hat, col = "red", lwd = 2)
```

Random Effects (cont.)

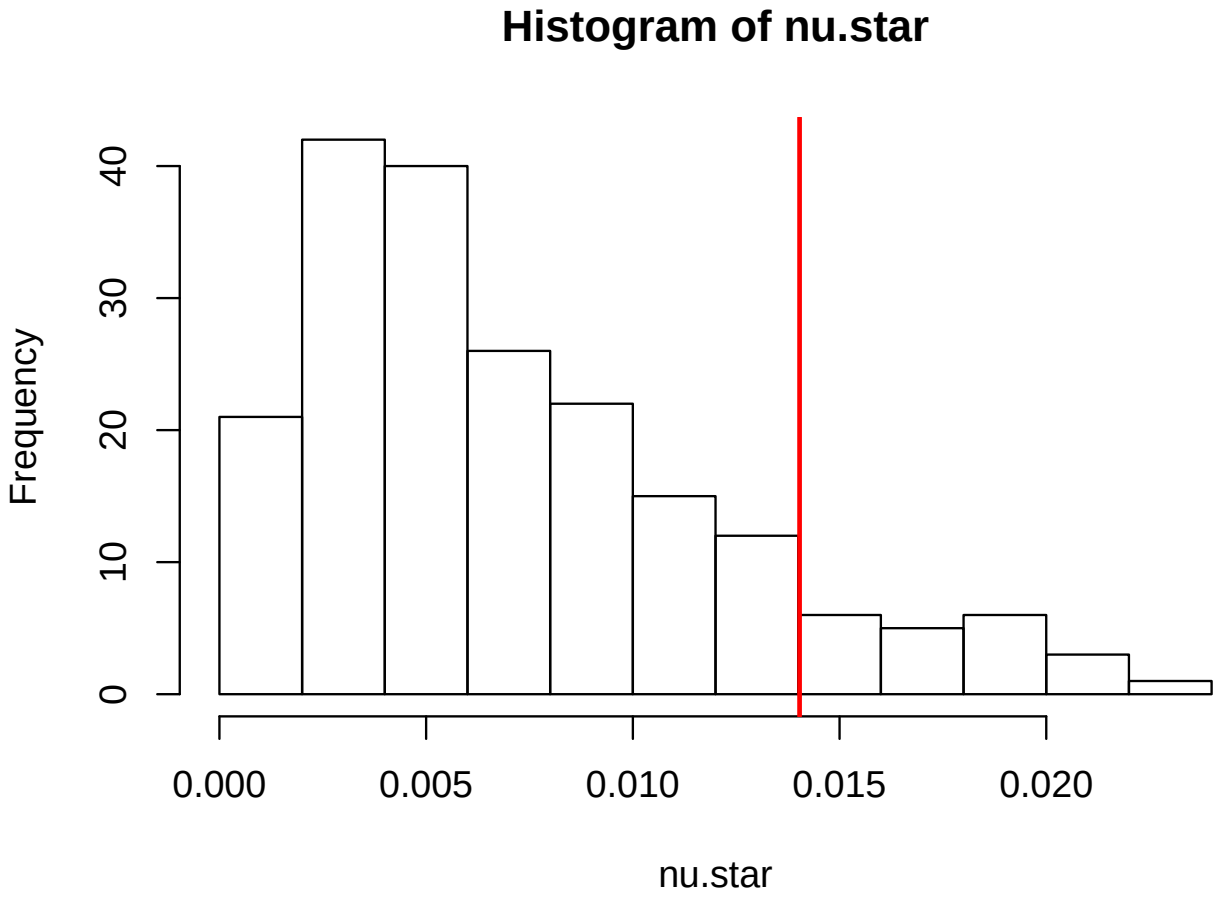

Figure: Bootstrap distribution of variance component for population random effects. Red vertical line shows estimate for actual data.

Random Effects (cont.)

Now both histograms look highly biased. Either both will be or neither will be, since squaring is a monotone transformation.

```
> normint <- sigma.hat + qnorm(perc) * se.sigma.hat
> percint <- sort(sigma.star)[(nboot + 1) * perc]
> z.star <- (sigma.star - sigma.hat) / se.sigma.star
> c.star <- sort(z.star)[(nboot + 1) * perc]
> tint <- sigma.hat - rev(c.star) * se.sigma.hat
```

Random Effects (cont.)

```
> foo <- rbind(normint, percint, tint)
> rownames(foo) <- c("asymptotic", "percentile", "bootstrap
> colnames(foo) <- c("low", "high")
> round(foo, 4)
```

```
                low   high
asymptotic   0.0478 0.1891
percentile   0.0300 0.1411
bootstrap t 0.0802 0.2680
```

Now, because of the high bias, the bootstrap percentile interval seems worst of all. (We would need a double bootstrap to confirm that.)

The bootstrap t interval is much shorter than the asymptotic interval, and presumably the best.

Still the asymptotic interval isn't really bad. Just longer than it needs to because it assumes asymptotic unbiasedness.