**Introduction**

Let’s now talk about bootstrap testing. Not much is different when we’re doing bootstrap testing; the main difference is that the replicates used to generate test statistics depend on the data we feed to the test, and thus are not completely independent of it. You can read more about bootstrap testing in [1].

**Bootstrap Hypothesis Testing**

Let Srepresent our test statistic. For bootstrap hypothesis testing, we will construct Ntest statistics from data generated using our sample. Call these test statistics S^{*}_1, \ldots, S^{*}_N. These statistics are generated in such a way that we know that the null hypothesis holds for them. Suppose for the sake of demonstration that large values of Sconstitute evidence against the null hypothesis. Then the p-value for the bootstrap hypothesis test is

\hat{p} = \frac{1}{N} \sum_{j = 1}^{N} I_{\{S^{*}_j > S\}}

Here, Iis the [indicator function](https://en.wikipedia.org/wiki/Indicator_function).

There are many ways to generate the data used to compute S^{*}_j. There’s the [parametric bootstrap](https://en.wikipedia.org/wiki/Bootstrapping_(statistics)#Parametric_bootstrap), where the data is used to estimate the parameters of a distribution, then those parameters are plugged into that distribution and then the distribution is used to generate new samples. There’s also the nonparametric bootstrap that doesn’t make such strong assumptions about the data, perhaps sampling from the data itself to generate new samples. Either of these methods can be used in bootstrap testing, and MCHTest() supports both.

Unlike Monte Carlo tests, bootstrap tests cannot claim to be exact tests for any sample size; they’re better for larger sample sizes. That said, they often work well even in small sample sizes and thus are still a good alternative to inference based on asymptotic results. They also could serve as an alternative approach to the nuisance parameter problem, as MMC often has weak power.

**Bootstrap Testing in MCHT**

In **MCHT**, there is little difference between bootstrap testing and Monte Carlo testing. Bootstrap tests need the original dataset to generate replicates; Monte Carlo tests do not. So the difference here is that the function passed to rand\_gen needs to accept a parameter x rather than n, with x representing the original dataset, like that passed to test\_stat.

That’s the only difference. All else is the same.

**Nonparametric Bootstrap Example**

Suppose we wish to test for the location of the mean. Our nonparametric bootstrap procedure is as follows:

1. Generate Nsamples of data from the demeaned dataset.
2. Suppose our mean under the null hypothesis is \mu_0. Add this mean to each generated dataset and compute the tstatistic for each of those datasets; these will be the simulated test statistics S^{*}_j.
3. Compute the test statistic on the main data and use the [empirical distribution function](https://en.wikipedia.org/wiki/Empirical_distribution_function) of the simulated test statistics to compute a p-value.

The code below implements this procedure.

library(MCHT)

library(doParallel)

registerDoParallel(detectCores())

ts <- function(x, mu = 0) {

sqrt(length(x)) \* (mean(x) - mu)/sd(x)

}

rg <- function(x) {

x\_demeaned <- x - mean(x)

sample(x\_demeaned, replace = TRUE, size = length(x))

}

sg <- function(x, mu = 0) {

x <- x + mu

test\_stat(x, mu = mu) # Will be localizing

}

b.t.test <- MCHTest(ts, sg, rg, seed = 123, N = 1000, lock\_alternative = FALSE,

test\_params = "mu", localize\_functions = TRUE)

dat <- c(2.3, 1.1, 8.1, -0.2, -0.8, 4.7, -1.9)

b.t.test(dat, alternative = "two.sided", mu = 1)

##

## Monte Carlo Test

##

## data: dat

## S = 0.68164, p-value = 0.432

## alternative hypothesis: true mu is not equal to 1

b.t.test(dat, alternative = "less", mu = 7)

##

## Monte Carlo Test

##

## data: dat

## S = -3.8626, p-value = 0.025

## alternative hypothesis: true mu is less than 7

**Parametric Bootstrap Test**

The parametric bootstrap test assumes that the observed data was generated using a specific distribution, such as the [Gaussian distribution](https://en.wikipedia.org/wiki/Normal_distribution). All that’s missing, in essence, is the parameters of that distribution. The procedure thust starts by estimating all nuisance parameters of the assumed distribution using the data. Then the first step of the process mentioned above (which admittedly was specific to a test for the mean but still strongly resembles the general process) is replaced with simulating data from the assumed distribution using any parameters assumed under the null hypothesis and the estimated values of any nuisance parameters. The other two steps of the above process are unchanged.

We can use the parametric bootstrap to test for goodness of fit with the [Kolmogorov-Smirnov test](https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test#Kolmogorov_distribution). Without going into much detail, suppose that Frepresents a distribution that is known except maybe for the values of its parameters. Assume that X_1, \ldots, X_nis an independently and identically distributed dataset, and we have observed values x_1, \ldots, x_n. We wish to use the dataset to decide between the hypotheses

H_0: X_1 \sim F

H_A: H_0 \text{ is false}

That is, we want to test whether our data was drawn from the distribution For whether it was drawn from a different distribution. This is what the Kolmogorov-Smirnov test checks.

R implements this test in ks.test() but that function does not allow for any nuisance parameters. It will only check for an exact match between distributions. This is often not what we want; we want to check whether out data was drawn from any member of the family of distributions F, not a particular member with a particular combination of parameters. It’s tempting to plug in the estimated values of these parameters, but then the p-value needs to be computed differently, not in the way that is prescribed by ks.test(). Thus we will need to approach the test differently.

Since the distribution of the data is known under the null hypothesis, this is a good situation to use a bootstrap test. We’ll use [maximum likelihood estimation](https://en.wikipedia.org/wiki/Maximum_likelihood_estimation) to estimate the values of the missing parameters, as implemented by **fitdistrplus** (see [2]). Then we generate samples from this distribution using the estimated parameter values and use those samples to generate simulated test statistic values that follow the distribution prescribed by the null hypothesis.

Suppose we wished to test whether the data was drawn from a Weibull distribution. The result is the following test.

library(fitdistrplus)

ts <- function(x) {

param <- coef(fitdist(x, "weibull"))

shape <- param[['shape']]; scale <- param[['scale']]

ks.test(x, pweibull, shape = shape, scale = scale,

alternative = "two.sided")$statistic[[1]]

}

rg <- function(x) {

n <- length(x)

param <- coef(fitdist(x, "weibull"))

shape <- param[['shape']]; scale <- param[['scale']]

rweibull(n, shape = shape, scale = scale)

}

b.ks.test <- MCHTest(test\_stat = ts, stat\_gen = ts, rand\_gen = rg,

seed = 123, N = 1000)

b.ks.test(rweibull(1000, 2, 2))

##

## Monte Carlo Test

##

## data: rweibull(1000, 2, 2)

## S = 0.021907, p-value = 0.275

b.ks.test(rbeta(1000, 2, 2))

##

## Monte Carlo Test

##

## data: rbeta(1000, 2, 2)

## S = 0.047165, p-value < 2.2e-16

**Conclusion**

Given the choice between a MMC test and a bootstrap test, which should you prefer? If you’re concerned about speed and power, go with the bootstrap test. If you’re concerned about precision and getting an “exact” test that’s at least conservative, then go with a MMC test. I think most of the time, though, the bootstrap test will be good enough, even with small samples, but that’s mostly a hunch.