**Introduction**

[introduced **MCHT**](https://wp.me/p27liZ-TD) two weeks ago and presented it as a package for Monte Carlo and boostrap hypothesis testing. Last week, and delved into important technical details and showed how to [make self-contained MCHTest objects](https://wp.me/p27liZ-TG) that don’t suffer side effects from changes in the global namespace. In this article I show how to perform maximized Monte Carlo hypothesis testing using **MCHT**, as described in [1].

**Maximized Monte Carlo Hypothesis Testing**

The usual procedure for Monte Carlo hypothesis testing is:

1. Compute a test statistic for the data on which you wish to test a hypothesis
2. Generate Nrandom datasets like the one of interest but with the data generating process (DGP) being the one prescribed by the null hypothesis, and compute the test statistic on each of these datasets
3. Use the [empirical distribution function](https://en.wikipedia.org/wiki/Empirical_distribution_function) of the simulated test statistics to compute the p-value of the test

Monte Carlo tests often make strong distributional assumptions, such as what distribution generated the dataset being tested, but when those assumptions hold, they are exact tests (see [2]). They are not as powerful as if we had the exact distribution of the test statistic under those assumptions but the power increases with N(see [3]) and given the power of modern computers getting a large Nis usually not a problem. Thus Monte Carlo tests are attractive in small sample situations where we do not want to rely on an asymptotic distribution for inference.

However, the procedure outlined above does not allow for nuisance parameters (parameters that are not the subject of interest but whose values are needed in order to conduct inference). In the introductory blog post, while one may view the population standard deviation \sigmaas a nuisance parameter, the test statistic \frac{\bar{X} - \mu_0}{\frac{S}{\sqrt{n}}}does not depend on it when the data follows a [Gaussian distribution](https://en.wikipedia.org/wiki/Normal_distribution) so there was no need to worry about it. In the case when we switched to data following the exponential distribution, it was still not a problem since its value was specified under the null hypothesis (\sigma = \mu_0). Thus it was no longer a nuisance parameter.

That said, nuisance parameters can still appear when we need to perform inference. Suppose, for example, that our data follows a [Weibull distribution](https://en.wikipedia.org/wiki/Weibull_distribution), denoted by X_i \sim \text{WEI}(k, \lambda), with kbeing the shape parameter and \lambdathe scale parameter. We want to test the set of hypotheses:

H_0: \lambda = \lambda_0

H_A: \lambda \neq \lambda_0

We can use the generalized likelihood ratio test to form a test statistic (which I won’t repeat hear but does appear in the code below). While [Wilks’ theorem](https://en.wikipedia.org/wiki/Wilks%27_theorem) tells us about the asymptotic distribution of the test statistic, it says nothing about the exact distribution of the test statistic at a particular sample size, and it’s not given that the test statistic is [pivotal](https://en.wikipedia.org/wiki/Pivotal_quantity) and thus independent of the value of nuisance parameters under the null hypothesis (the nuisance parameter, in this case, being the shape parameter k).

What then can we do? A bootstrap test would estimate the value of the nuisance parameter under the null hypothesis and use that estimate as the actual value when generating new, simulated test statistics. Bootstrap tests, however, are not exact tests (see [2]) and we’ve decided that we want a test with stronger guarantees.

[1] introduced the maximized Monte Carlo (MMC) test, which proceeds as described below:

1. Compute the test statistic from the data.
2. Generate Ncollections of random numbers, such as uniformly distributed random numbers, and use those random numbers for generating random copies of test statistics that depend on the values of nuisance parameters (notice that the random numbers are effectively *not* discarded)
3. Use an optimization procedure ([1] suggested simulated annealing) to pick values for the nuisance parameters such that the p-value is maximized; the maximally chosen p-value is the p-value of the test

[1] showed that this procedure yields p-values that, while not as precise as if we knew the values of the nuisance parameters that produced the data, are at least *conservative*, in the sense that they’re no smaller than they should be (thus biasing our conclusions in favor of the null hypothesis). This is the best we can hope for in this context.

MMC is intuitive and compelling, and the theoretical guarantee gives us confidence in our conclusions, but it’s not a panacea. First, the optimization procedure is costly in work and time. Second (and, in my opinion, the biggest problem), the procedure may be *too* conservative. There’s a strong chance that the procedure will find *some* values for nuisance parameters that yield a large p-value, perhaps a combination not at all resembling the actual values of the nuisance parameters that produced the data. In short, MMC can be severely lacking in [power](https://en.wikipedia.org/wiki/Power_(statistics)). When it does reject the null hypothesis, it’s compelling, but otherwise it’s not convincing that the alternative hypothesis is false.

**MMC in MCHT**

Creating an implementation of MMC in R was my original goal in developing **MCHT**, and all that needs to be done to perform MMC is pass a value the nuisance\_params parameter and an appropriate list to optim\_control.

Let’s take the hypothesis test mentioned above and prepare to implement it using **MCHT**. I will be using **fitdistrplus** for maximum likelihood estimation, as required by the test statistic (see [[4]](https://www.jstatsoft.org/article/view/v064i04)).

library(MCHT)

## .------..------..------..------.

## |M.--. ||C.--. ||H.--. ||T.--. |

## | (\/) || :/\: || :/\: || :/\: |

## | :\/: || :\/: || (\_\_) || (\_\_) |

## | '--'M|| '--'C|| '--'H|| '--'T|

## `------'`------'`------'`------' v. 0.1.0

## Type citation("MCHT") for citing this R package in publications

library(fitdistrplus)

registerDoParallel(detectCores())

# To be passed to test\_stat

ts <- function(x, scale = 1) {

fit\_null <- coef(fitdist(x, "weibull", fix.arg = list("scale" = scale)))

kt <- fit\_null[["shape"]]

l0 <- scale

fit\_all <- coef(fitdist(x, "weibull"))

kh <- fit\_all[["shape"]]

lh <- fit\_all[["scale"]]

n <- length(x)

# Test statistic, based on the negative-log-likelihood ratio

suppressWarnings(n \* ((kt - 1) \* log(l0) - (kh - 1) \* log(lh) -

log(kt/kh) - log(lh/l0)) - (kt - kh) \* sum(log(x)) + l0^(-kt) \*

sum(x^kt) - lh^(-kh) \* sum(x^kh))

}

# To be passed to stat\_gen; localize\_functions will be TRUE

sg <- function(x, scale = 1, shape = 1) {

x <- qweibull(x, shape = shape, scale = scale)

test\_stat(x, scale = scale)

}

The MCHTest() parameter nuisance\_params accepts a character vector giving the names of nuisance parameters the distribution of the test statistic may depend upon, and those names must be among the arguments of the function passed to stat\_gen; that function is expected to know how to handle those parameters. In this case, rand\_gen will not be specified since by default it gives uniformly distributed random variables. It’s a well-known fact in probability that the inverse of the CDF of a random variable (which are the q functions in R) applied to a uniformly distributed random variable yields a random variable that follows the distribution prescribed by the CDF. Hence the use of qweibull() above, which is being applied to datasets of uniformly distributed random variables that are effectively fixed when stat\_gen will be called. Then the test statistic computed will be computed from data that follows the scale parameter \lambdaprescribed by the null hypothesis but for some set value of k, the shape parameter.

The MCHTest object will then perform simulated annealing to choose the value of the nuisance parameter that maximizes the p-value under the null hypothesis for the given dataset. Simulated annealing is implemented in the GenSA() function provided by the **GenSA** package (see [[5]](https://journal.r-project.org/archive/2013/RJ-2013-002/index.html)). GenSA() needs a description of what set of parameter values over which to optimize and there is no general method for choosing this, so MCHTest() will require that a list be passed to optim\_control that effectively contains the parameters that will be passed to GenSA(). At minimum, this list must contain an upper and a lower element, each of which are numeric vectors with names exactly like the character vector passed to nuisance\_params; these vectors specify the space GenSA() will search to find the optima. Other elements can be passed to control GenSA(), and I highly recommend reading the function’s documentation for more details.

There’s an additional parameter of MCHTest(), threshold\_pval, that matters to the optimization. GenSA() will take many steps to make sure it reaches a good optimal value, perhaps taking too long. The authors of **GenSA** recommend specifying an additional terminating condition to speed up the process. threshold\_pval will alter the threshold.stop parameter in the control list of optim\_control so that the algorithm terminates when the estimated p-value crosses threshold\_pval‘s value. Effectively, this means that we know that whatever the true p-value of the test is, it’s larger than that threshold, and if the threshold is chosen appropriately, we know that we should not reject the null hypothesis based on the results of this test.

While giving threshold\_pval a number less than 1 can help terminate the algorithm in the case of not rejecting the null hypothesis, the algorithm can still run for a long time if the test will eventually return a statistically significant result. For this reason, I recommend that optim\_list contain a list called control and that the list should have a max.time element telling the algorithm the maximum running time (in seconds) it should have.

With all this in mind, we create the MCHTest object below:

mc.wei.scale.test <- MCHTest(ts, sg, N = 1000, seed = 123,

test\_params = "scale", nuisance\_params = "shape",

optim\_control = list("lower" = c("shape" = 0),

"upper" = c("shape" = 100),

"control" = list(

"max.time" = 10

)), threshold\_pval = .2,

localize\_functions = TRUE)

mc.wei.scale.test(rweibull(10, 2, 2), scale = 4)

##

## Monte Carlo Test

##

## data: rweibull(10, 2, 2)

## S = 7.2983, p-value < 2.2e-16

**Conclusion**

The MMC procedure is intersting and I don’t think any package implements it to the level I have in **MCHT**. The power of the procedure itself concerns me, though. Fortunately, the package also supports bootstrap testing, which I will discuss [next week](https://wp.me/p27liZ-TU).

**References**

1. J-M Dufour, *Monte Carlo tests with nuisance parameters: A general approach to finite-sample inference and nonstandard asymptotics*, Journal of Econometrics, vol. 133 no. 2 (2006) pp. 443-477
2. J. G. MacKinnon, *Bootstrap hypothesis testing* in *Handbook of computational econometrics* (2009) pp. 183-213
3. A. C. A. Hope, *A simplified Monte Carlo test procedure*, JRSSB, vol. 30 (1968) pp. 582-598
4. M. L. Delignette-Muller and C. Dulag, *fitdistrplus: an R package for fitting distributions*, J. Stat. Soft., vol. 64 no. 4 (2015)
5. Y. Xiang et. al., *Generalized simulated annealing for global optimization: the GenSA package*, R Journal, vol. 5 no. 1 (2013) pp. 13-28