R-package “simctest”
R-class “mctest”
A Short Introduction

Dong Ding, Axel Gandy and Georg Hahn

March 23, 2017

This document describes briefly how to use the class “mctest”, included in the R-package “simctest”. It implements the methods from “Implementing Monte Carlo Tests with Multiple Thresholds” of Dong et al. (2017).

The class can be used to evaluate the statistical significance of a hypothesis H_0 with respect to multiple thresholds. It is assumed that the p-value p of H_0 is not known analytically and can only be approximated via Monte Carlo simulation. To this end, a function `gen()` provided by the user is used to draw one sample under H_0 at a time in order to approximate the p-value corresponding to H_0. By means of an appropriate (default) choice of the thresholds, the R-class can be used to either obtain an exact decision for H_0 with respect to all given thresholds (in expected infinite time), or to obtain a finite time decision in extended star notation, see Dong et al. (2017).

1 Installation

The functions described in this document are included in the R-package “simctest”. Please see the documentation of “simctest” on how to install the package.

2 Usage

The package is loaded by typing

```
> library(simctest)
```

This document can be accessed via

```
> vignette("simctest-mctest-intro")
```

Documentation of the most useful commands can be obtained as follows:

```
> ? simctest
> ? mctest
```

0	***	10^{-3}	**	0.01	*	0.05		1
	10^{-5}	**~	$4 \cdot 10^{-3}$	*~	0.03	~	0.1	

Table 1: Reporting the significance of a hypothesis: The classical way (first row) from * (weakly significant) to *** (highly significant) and a proposed alternative approach using auxiliary thresholds (second row). Taken from Dong et al. (2017).

3 Testing with respect to multiple thresholds

Dong et al. (2017) define a general scenario to test a hypothesis with respect to a given set of multiple thresholds. More generally, the authors describe an algorithm to find the *p-value bucket* containing the p-value p of interest (among a finite set of p-value buckets specified by the user).

The classical choice of p-value buckets, given by

$$\mathcal{J} = \mathcal{J}^0 := \{[0, 10^{-3}], (10^{-3}, 0.01], (0.01, 0.05], (0.05, 1]\},$$

is equivalent to deciding where p lies in relation to the three thresholds 0.001, 0.01 and 0.05 traditionally employed in hypothesis testing.

However, the concept of p-value buckets is more general: The overlapping intervals

$$\mathcal{J}^* = \mathcal{J}^0 \cup \{(10^{-5}, 4 \cdot 10^{-3}], (4 \cdot 10^{-3}, 0.03], (0.03, 0.1]\}$$

make it possible to extend the classical way of reporting significances to finite time decisions, however at the expense that the exact location of p with respect to the buckets in $\mathcal{J}^*$ is only known for all but one bucket. For instance, in case the threshold 0.01 will remain as the last undecided one, p will be reported to be definitely (up to a pre-specified error probability) significant at the 0.05 level, with possible significance at 0.01 as well. This is summarised in Table 1.

A refined (more narrow) choice of overlapping thresholds used in Dong et al. (2017) is given by

$$\mathcal{J}^n = \mathcal{J}^0 \cup \{(10^{-4}, 3 \cdot 10^{-3}], (6 \cdot 10^{-3}, 0.015], (0.04, 0.06]\}.$$

These three threshold sets (sets of p-value buckets) are provided as default choices in *R* upon loading `simctest` as variables `J`, `Jstar` and `Jnarrow`.

4 Using the R-class

The following gives a step-by-step introduction to the R-class `mctest` and provides examples.

4.1 Providing a sampler interface

To implement a given test, it suffices to specify a function `gen()` (without input parameters) which returns one sample under H_0, thus allowing to approximate the p-value p of H_0.

For simplicity, we assume $p = 0.04$ for now and we simulate samples under H_0 by drawing independent Bernoulli samples, thus

```
> gen <- function() { runif(1)<0.04 }
```

4.2 Defining p-value buckets

A set of r p-value buckets one wishes to test p against is specified as an $r \times 2$ matrix. For instance, the traditional choice of thresholds 0.001, 0.01 and 0.05 already seen in the definition of `J` is given as

```
> J <- matrix(nrow=2,
+             c(0,   1e-3,
+               1e-3,1e-2,
+               1e-2,0.05,
+               0.05,1))
> colnames(J) <- c("***","**","*","")
```

Additionally, the significance classifier symbols corresponding to each bucket are specified as column names.

4.3 Testing with respect to a given set of p-value buckets

Given a generating mechanism for samples under H_0 is provided (see Section 4.1), the main algorithm can be called using

```
> res <- mctest(gen,J=Jnarrow,epsilon=0.001,batch=10,
+               batchincrement=1.1,maxbatch=100,method="simctest")
```

where the parameter `method` can take the arguments ”simctest” and ”RL”. These two options refer to the method used to compute confidence intervals for p, see Dong et al. (2017). The choice of confidence intervals does not affect the accuracy of the result. The choice ”simctest”, however, seems to be computationally more favourable since it leads to faster decisions on p.

The other parameters are as follows:

- `J` is the matrix of p-value buckets (default choice is `Jnarrow`).
- `epsilon` is the allowed resampling error for the (simultaneous) decision on all buckets.
- `batch` is the batch size for new samples drawn in each iteration.
- `batchincrement` is the geometric increment used to increase the batch size of samples drawn (use value 1 for no increase).
- `maxbatch` is the upper limit of samples drawn in each iteration.
- `method` is the method used to compute confidence intervals for p, use ”simctest” or ”RL”.

In fact, the function `mctest` is just a wrapper function for the actual routines carrying out the testing, precisely

```
> mctest.RL(gen,J=Jnarrow,epsilon=0.001,batch=10,
+           batchincrement=1.1,maxbatch=100)
```

for the ”RL” (Robbins-Lai) approach and

```
> mctest.simctest(gen,J=Jnarrow,epsilon=0.001,batch=10,
+                 batchincrement=1.1,maxbatch=100)
```

for the ”simctest” approach. These two functions can also be called directly.

4.4 Testing result

Once `mctest` (or `mctest.RL` or `mctest.simctest`, respectively) have finished their computation, a result object of the class `mctestres` is returned. This class provides a print function for its objects.

First,

```
> res
```

```
Interval for p-value: [0.01,0.05]
Decision: *
Estimate of p-value: 0.0338345864661654
Batched number of samples: 2926
Actual number of samples: 2953
```

prints a summary of the computation, consisting of the final interval (bucket) from `J` the p-value is determined to lie in, the decision (taken from the definition of `J`, see Section 4.2) in (extended) star notation, an estimate of the p, as well as the batched number of samples actually drawn in the run and the actual (non-batched) number of samples needed to reach a decision (due to batching, a few samples more are drawn in the last batch than would have been needed to reach a decision).

Similarly, the individual elements of the results object can be accessed by the user:

```
> res$decision.interval
```

```
[1] 0.01 0.05
```

```
> res$decision
```

```
[1] "*"
```

```
> res$est.p
```

```
[1] 0.03383459
```

```
> res$batchedSamples
```

```
[1] 2926
```

```
> res$actualSamples
```

```
[1] 2953
```

They can be obtained as list entries *decision.interval*, *decision*, *est.p*, *batchedSamples* and *actualSamples*.

4.5 An extended example

The following example is a likelihood ratio test of contingency table data which can be found in Dong et al. (2017). It consists of 39 multinomial counts for two categorical variables in a 5×7 contingency table. We wish to test for independence of these two variables.

We first enter the data example found in Gandy (2009), Newton and Geyer (1994) or Davison and Hinkley (1997) as well as the likelihood ratio test statistic:

```
> dat <- matrix(nrow=5,ncol=7,byrow=TRUE,
+   c(1,2,2,1,1,0,1,
+     2,0,0,2,3,0,0,
+     0,1,1,1,2,7,3,
+     1,1,2,0,0,0,1,
+     0,1,1,1,1,0,0))
> loglikrat <- function(data) {
+   cs <- colSums(data)
+   rs <- rowSums(data)
+   mu <- outer(rs,cs)/sum(rs)
+   2*sum(ifelse(data<=0.5, 0,data*log(data/mu)))
+ }
```

Davison and Hinkley (1997) propose to use a parametric bootstrap test. The following is a function to resample the dataset:

```
> resample <- function(data){
+   cs <- colSums(data)
+   rs <- rowSums(data)
+   n <- sum(rs)
+   mu <- outer(rs,cs)/n/n
+   matrix(rmultinom(1,n,c(mu)),nrow=dim(data)[1],ncol=dim(data)[2])
+ }
```

After evaluating the test statistic on the (original) data,

```
> t <- loglikrat(dat)
```

and storing the result in t, we can define a p-value of a right-sided test as the proportion of exceedances of the test statistic evaluated on the resampled data over t. The following function returns binary samples corresponding to these exceedances:

```
> gen <- function(){loglikrat(resample(dat))>=t}
```

Using the function `gen`, we can test for independence in the contingency table as

```
> res <- mctest(gen,method="simctest")
> mctest.simctest(gen)

Interval for p-value: [0.01,0.05]
Decision: *
Estimate of p-value: 0.0434801820511687
Batched number of samples: 22631
Actual number of samples: 22653

> mctest.RL(gen)

Interval for p-value: [0.01,0.05]
Decision: *
Estimate of p-value: 0.0413206669766824
Batched number of samples: 15053
Actual number of samples: 15053
```

using the wrapper `mctest` as well as `mctest.simctest` or `mctest.RL`. The decision interval for the p-value as well as its significance (for the first instance of `mctest` which saved the test result) can be queried using

```
> res$decision.interval

[1] 0.01 0.05

> res$decision

[1] "*"
```

References

Davison, A. and Hinkley, D. (1997). *Bootstrap methods and their application.* Cambridge University Press.

Dong, D., Gandy, A., and Hahn, G. (2017). Implementing monte carlo tests with multiple thresholds. arXiv:0000.0000.

Gandy, A. (2009). Sequential implementation of Monte Carlo tests with uniformly bounded resampling risk. *Journal of the American Statistical Association*, 104(488):1504–1510.

Newton, M. A. and Geyer, C. J. (1994). Bootstrap recycling: A monte carlo alternative to the nested bootstrap. *Journal of the American Statistical Association*, 89(427):905–912.