R-package “seqampl”
A Short Introduction

Axel Gandy

May 28, 2009

This document describes briefly how to use the R-package which implements the algorithm for “Sequential implementation of monte carlo tests with uniformly bounded resampling risk.” based on Gandy [2009].

1 Installation

The installation is as for most R-packages that do not reside in CRAN. The general procedure is described in the Section 6 on “Add-on packages” in the R Manual on Istallation and Administration:
`http://cran.r-project.org/doc/manuals/R-admin.html`.

The following is merely an adaptation of those procedures to our package.

1.1 Linux/Unix

If you do not have write access to the package repository:

1. Download the package “simctest_1.0-0.tar.gz” and place it into your home directory.
2. Issue the following commands:

```
echo ".libPaths(\"$HOME/Rlibrary\")" >$HOME/.Rprofile
R CMD INSTALL -L $HOME/Rlibrary simctest_1.0-0.tar.gz
```

3. You may now delete the file “simctest_1.0-0.tar.gz”.

2 Usage

Obviously, the pacakge is loaded by typing

```
> library(simctest)
```

This document can be accessed via

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

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

```
> ? simctest
```

The following is an artificial example. By default the algorithm will report back after at most 10000 steps, work with a threshold of $\alpha = 0.05$ and use the spending sequence

$$\epsilon_n = 0.001 \frac{n}{1000 + n}.$$

A simple example of a test with true p-value 0.07.

```
> res <- simctest(function() runif(1) < 0.07)
> res

p.value: 0.06610067
Number of samples: 3298
```

One can also obtain a confidence interval (wrt the resampling procedure) of the computed p-value. By default a 95% confidence interval is computed.

```
> confint(res)

             2.5 %     97.5 %
p.value 0.05613075 0.07381973
```

2.1 Behaviour at the Threshold

Next, consider an example where the true p-value is precisely equal to the threshold α. Here, we will expect that the algorithm stops only with probability $2\epsilon = 0.002$. If the algorithm has not stopped after 10000 steps the algorithm will return.

```
> res <- simctest(function() runif(1) < 0.05)
> res

No decision reached.
Final estimate will be in [ 0.04159918 , 0.05952146 ]
Current estimate of the p.value: 0.0488
Number of samples: 10000
```

Note that a part of the output it the interval in which the final estimator will lie.

One can always take a few more steps

```
> res <- cont(res, 10000)
> res

No decision reached.
Final estimate will be in [ 0.04367672 , 0.056915 ]
Current estimate of the p.value: 0.0488
Number of samples: 20000
```

2.2 A simple bootstrap test

An example from [Davison and Hinkley, 1997, section 11.4, p. 534]:

```
> data(fir, package = "boot")
> fir.mle <- c(sum(fir$count), nrow(fir))
> fir.gen <- function(data, mle) {
+     d <- data
+     y <- sample(x = mle[2], size = mle[1], replace = TRUE)
+     d$count <- tabulate(y, mle[2])
+     d
+ }
> fir.fun <- function(data) (nrow(data) - 1) * var(data$count)/mean(data$count)
> resampl <- function() {
+     obs < fir.fun(fir.gen(data = fir, mle = fir.mle))
+ }
> obs <- fir.fun(fir)
> simctest(resampl)

p.value: 0.3809524
Number of samples: 21
```

2.3 Computing the power of a test

```
> n <- 10
> system.time(replicate(1000, {
+     obs <- mean(rnorm(n) + 0.01)
+     simctest(function() mean(rnorm(n)) > obs, maxsteps = 1000)
+ }))

   user  system elapsed 
  8.947   0.019   9.035 
```

Compared with the naive approach:

```
> system.time(replicate(1000, {
+     obs <- mean(rnorm(n) + 0.01)
+     mean(replicate(1000, mean(rnorm(n)) > obs))
+ }))

   user  system elapsed 
 30.610   0.080  31.594 
```

To reduce the overhead of computing the boundaries, they can be precomputed.

```
> alg <- getalgprecomp()
> system.time(replicate(1000, {
```

```
+     obs <- mean(rnorm(n) + 0.01)
+     run(alg, function() mean(rnorm(n)) > obs, maxsteps = 1000)
+ }))

   user  system elapsed 
  7.414   0.021   7.514 
```

For comparison purposes, the same without without pre-computation:

```
> alg <- getalgonthefly()
> system.time(replicate(1000, {
+     obs <- mean(rnorm(n) + 0.01)
+     run(alg, function() mean(rnorm(n)) > obs, maxsteps = 1000)
+ }))

   user  system elapsed 
  7.447   0.009   7.811 
```

References

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

Axel Gandy. Sequential implementation of Monte Carlo tests with uniformly bounded resampling risk. To appear in JASA, 2009.