Stats Seminar (STATS 770)
McMaster University
R programming assignment

15 November 2016

1. Consider a random sample from a mixture distribution with probability density function

$$f(y;\mu,\sigma) = \pi f_1(y;\mu,\sigma) + (1-\pi) f_2(y;\mu,\sigma), \quad y \in \mathbb{R}$$

where $f_1(.)$ and $f_2(.)$ are the PDFs of the Normal distribution with (μ=1.5, σ=1) and (μ=2, σ=1.75) respectively.

$$\pi = \begin{cases} 1 & \text{with prob. } P(U \geq 0.5) \\ 0 & \text{with prob. } P(U < 0.5) \end{cases}$$

where $U \sim$ Uniform (0,1).

- Create a numeric vector containing 10^4 values of $f(y;\mu,\sigma)$.
- Calculate the mean and standard deviation of the sample using base R functions.
- Calculate the mean and standard deviation of the sample explicitly using a `for` loop to add each term (without using default R functions).
- Compare the computation time using `system.time()` and report for both the cases above (i.e. with and without default R functions).
- Compare the computation time using a function from a benchmarking package (e.g. `microbenchmark::microbenc` or `rbenchmark::benchmark`) and report for both cases.
- Using `Rcpp::cppFunction`, compute the mean and standard deviation of the sample and report the computation time using a benchmark package. *You may use either explicit* `for` *loops or* `Rcpp` *"sugar" functions* (e.g. see here or particularly here).

2. A standard test case for simple computational methods is to estimate the value of π by the rejection method. In other words:
   - Pick N values uniformly in the unit square $[0,1] \times [0,1]$.
   - Count the fraction that fall inside the unit circle (i.e. $x^2 + y^2 < 1$) (in R you can conveniently do this via `mean(x^2+y^2<1)`, because applying `mean()` converts logical variables (`FALSE`/`TRUE`) to numeric (0/1)).
   - This value should be the area of the quarter-circle divided by the area of the square, i.e. $(\pi r^2/4)/r^2 = \pi/4$; multiplying by 4 gives π.

   - Write an R function `approx_pi` (with argument `N` specifying the number of samples) to compute this approximation. Try it out.
   - Now write R code to do this in parallel across more than one core on your machine. On Linux or MacOS `library(parallel); mclapply(c(N/2,N/2),approx_pi)` should work. If you are on Windows you will have to do something like

```r
library(parallel)
cl <- makeCluster(2)
parLapply(cl,c(N/2,N/2),approx_pi)
stopCluster(cl)
```

Use `system.time()` and/or a benchmarking package to compare the performance of the parallel and non-parallel codes.