STAT40730
Data Programming with R (Online).
Lab 11: performance enhancement

1. Here are three different functions to find the first occurrence of a specified word in a text string. Which one runs fastest?

```
fun1 <- function(string,word) {
  stringvec <- strsplit(string, ' ')[[1]]
  for(i in 1:length(stringvec)) {
    if(stringvec[i] == word) break
    }
  return(i)
}
```

```
fun2 <- function(string,word) {
  stringvec <- strsplit(string, ' ')[[1]]
  findwordcount <- 1
  foundword <- FALSE
  while(!foundword) {
    if(stringvec[findwordcount] == word) foundword <- TRUE
    findwordcount <- findwordcount + 1
    }
  return(findwordcount - 1)
}
```

```
fun3 <- function(string, word) {
  stringvec <- strsplit(string, ' ')[[1]]
  return(match(word,stringvec))
}
```

```
mystring <- 'The quick brown fox jumped over the lazy dog'
myword <- 'lazy'
fun1(mystring, myword)
fun2(mystring, myword)
fun3(mystring, myword)

# Look at timings
system.time(for(i in 1:1e5) fun1(mystring, myword))
system.time(for(i in 1:1e5) fun2(mystring, myword))
```

```
system.time(for(i in 1:1e5) fun3(mystring, myword))

# or
library(rbenchmark)
benchmark(fun1(mystring, myword), fun2(mystring, myword),
  fun3(mystring, myword),replications = 1e5)
```

2. Use `Rprof` on the three functions in question 1. Which function causes the biggest bottleneck within each of the three functions?

```
Rprof()
for(i in 1:1e5) fun1(mystring, myword)
Rprof(NULL)
summaryRprof()
Rprof()
for(i in 1:1e5) fun2(mystring, myword)
Rprof(NULL)
summaryRprof()
```

3. In the functions given above, the string is split into a character vector of single words using `strsplit` which produces a list. You can get the character vector out of the list by using the square brackets `[[1]]` (as above) or by using `unlist`. Which is faster?

```
stringvec <- strsplit(mystring, ' ')[[1]]
stringvec2 <- unlist(strsplit(mystring, ' '))
system.time(for(i in 1:1e6) strsplit(mystring, ' ')[[1]])
system.time(for(i in 1:1e6) unlist(strsplit(mystring, ' ')))
```

4. Fitting a linear regression model when you have millions of observations can be very slow, but the matrix solution for the regression coefficients after some algebra are:

$$(X^T X)^{-1} X y$$

where y is a vector of response variables of length n and X is a matrix of explanatory variables of dimensions $n \times p$. To see the speed up that can be achieved using matrix algebra over, say, using the inbuilt `lm` function (function `reg1()` below), or over using a numerical optimiser such as `optim` (function `reg2()` below), have a look at function `reg3()` below.

```
# Create a toy explanatory variable matrix X and for regression
# coefficients 3, 2 and -4 create a response variable y, for large n.
set.seed(123)
n <- 1e6
```

```r
x1 <- runif(n)
x2 <- runif(n)
y <- rnorm(n, 3 + 2 * x1 - 4 * x2, sd = 0.5)
# Version 1: use lm to estimate the regression coefficients.
reg1 <- function() lm(y ~ x1 + x2)$coefficients
# Version 2: use optim to estimate the regression coefficients.
reg2 <- function() {
  ls.fun <- function(params) {
    intercept <- params[1]
    coef.x1 <- params[2]
    coef.x2 <- params[3]
    fits <- intercept + coef.x1 * x1 + coef.x2 * x2
    ss <- sum((y - fits)^2)
  }
  optim(c(0, 0, 0),ls.fun)$par
}

# Version 3: use matrix algebra to estimate the regression coefficients.
reg3 <- function() {
  X <- cbind(1, x1, x2)
  solve(t(X) %*% X, t(X) %*% y)
}
reg1()
reg2()
reg3()
# Which is fastest?
benchmark(reg1(), reg2(), reg3(), replications = 3)
# Matrix algebra approach in reg3() wins!
```