Code review for problem set 4

Jarrod Millman
Statistics 243
UC Berkeley

October 27, 2014

For section today, I will ask you to form a few small (< 5 people) groups. In your groups, you should discuss the following questions and then I will randomly ask one of the groups to informally summarize some of the ideas they discussed. When presenting feel free to draw on the board or ask me and/or the other students questions about parts you are unclear about.

4. Random walks

   For question 4, you were asked to implement a random walk function. Consider the following code snippet from Chris' example solution.

```
updown <- sample(c(0, 1), n, replace=TRUE)
steps <-  sample(c(-1, 1), n, replace=TRUE)
xsteps <- (updown == 0) * steps
ysteps <- (updown == 1) * steps
x <- c(0, cumsum(xsteps))
y <- c(0, cumsum(ysteps))
```

   Look through this implementation and make sure you understand how it works. In particular, make sure you can explain how this code is generating the random steps. Here are some specific questions you may wish to consider:

   - What purpose does `cumsum` serve?
   - If `cumsum` didn't exist, how would you modify this code snippet so that it still worked? Which of these implementations makes the code easier to understand and debug? Are there any implications for speed and memory use?
   - R has many functions for drawing random samples. Which of these other functions might be useful if `sample` didn't exist? Would you prefer using one of these other functions over `sample`? For example, are any of them more or less similar to how you think about the underlying mathematical model? Are there any implications for speed and memory use?

5. Memory use

   For extra credit on question 5, you were asked to rewrite a function in order to reduce its memory use. For this exercise, please consider the following portion of the original code:

```
nalineX <- is.na(xvar)
nalineY <- is.na(yvar)
xvar[nalineX | nalineY] <- 0
yvar[nalineX | nalineY] <- 0
useline <- !(nalineX | nalineY)
```

   As well as the corresponding part of Chris' example solution:

```
naline <- is.na(xvar)
naline[is.na(yvar)] <- TRUE
xvar[naline] <- 0
yvar[naline] <- 0
useline <- 1L - naline
#naline <- which(naline)
```

   Make sure you can explain why this works and why it is uses less memory. Are there any trade-offs (perhaps in terms of speed, generality, or readability) that were made for the sake of reducing memory use?

   Now consider the following alternative solutions:

   (a)
```
xvar[is.na(xvar) | is.na(yvar)] <- 0
yvar[is.na(xvar) | is.na(yvar)] <- 0
useline <- !(is.na(xvar) | is.na(yvar))
```

   (b)
```
naline <- is.na(xvar + yvar)
useline <- as.integer(!(naline))
```

   (c)
```
naline <- unique(c(which(xvar %in% NA), which(yvar %in% NA)))
useline <- as.integer(rep(1, lenght(xvar)))
useline[naline] <- as.integer(0)
```

   Do they all produce the same result? Are there similarities among the different solutions? Do these solutions make different trade-offs from each other or from Chris' example solution? When looking over the different solutions are there some that you prefer? If so, why?

6. Linear algebra and loops

   Finally, for question 6, you were asked to rewrite a function to minimize its computation time. There were several things that you could do, but the biggest reduction came from rewriting this portion of the code:

```
for (i in 1:n) {
  for (j in 1:n) {
    for (z in 1:K) {
      q[i, j, z] <- theta.old[i, z]*theta.old[j, z] /
                        Theta.old[i, j]
    }
  }
}
```

   Here is a possible solution:

```
for (z in 1:K) {
  q[ , , z] <- tcrossprod(theta.old[, z]) / Theta.old
}
```

   Does this perform the same computation? Is it more or less readable than the original? Is one of the two implementations easier to understand? Why?

   In the proposed solution, what other commands could you use to replace the body of the loop without adding or changing the loop? For example, Chris' example solution used `outer(theta.old[, z],theta.old[, z])`. Are there some that you prefer? Why? For example, are some of them easier for you to read and understand? Are some closer to the underlying mathematical expression that you would use? Are they all equivalent in terms of space and time complexity?

   Finally, consider the following alternative loop:

```
for (i in 1:n) {
  q[i, , ] <- t(theta.old[i, ] * t(theta.old[i, ])) /
                    Theta.old[i, ]
}
```

   Does it perform the same computation? Explain. Assuming it is correct, are there any cases when this loop might be preferred over the other proposed loop (or *vice versa*)?