STAT 33B Workbook 11

Yuanrui Zhu (3034615728)

Nov 12, 2020

This workbook is due **Nov 12, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

Code Performance

Watch the "Code Performance" lecture video.

No exercises for this section.

Profiling

Watch the "Profiling" lecture video.

No exercises for this section.

Profvis

Watch the "Profvis" lecture video.

Exercise 1

Read chapter 23 of Advanced R.

The flame graph for the function `rhand` function in the lecture has entries for a call to `<GC>`. What do these entries mean? Why are the listed as being "called" by the `c` function? Explain in 3-5 sentences.

YOUR ANSWER GOES HERE:

indicates the garbage collector is running. A lot of time corresponding to this entry usually means that the user is creating many short-lived objects. "c" is the way of creating vectors in R. Since vectorize using increase the efficiency of the code by taking a whole object approach to a problem. The listed can often called by vectorization to decrease the runtime.

Profiling Case Study

Watch the "Profiling Case Study" lecture video.

The next two exercises are related to the "Code Performance" lecture, but I've put them here at the end so the assigned readings are in order.

Exercise 2

Read chapter 24 of Advanced R.

What's the difference between `colSums` and `.colSums`? Explain.

YOUR ANSWER GOES HERE:

colSums is a kind of vectorised matrix functions that will always be faster than using apply(). We can sometimes use these functions to build other vectorised functions.

.colSums is the versions with an initial dot in the name are 'bare-bones' versions for use in programming: they apply only to numeric (like) matrices and do not name the result. (from R documentation)

Exercise 3

Create several test matrices of different sizes and use the microbenchmark or bench package to compare the speeds of:

- `colSums(m)`
- `.colSums(m)`
- `apply(m, 2, sum)`

Here `m` denotes a matrix. Use a plot to present your results. Is one of these consistently faster than the others? Does the size of the matrix have an effect on which is faster?

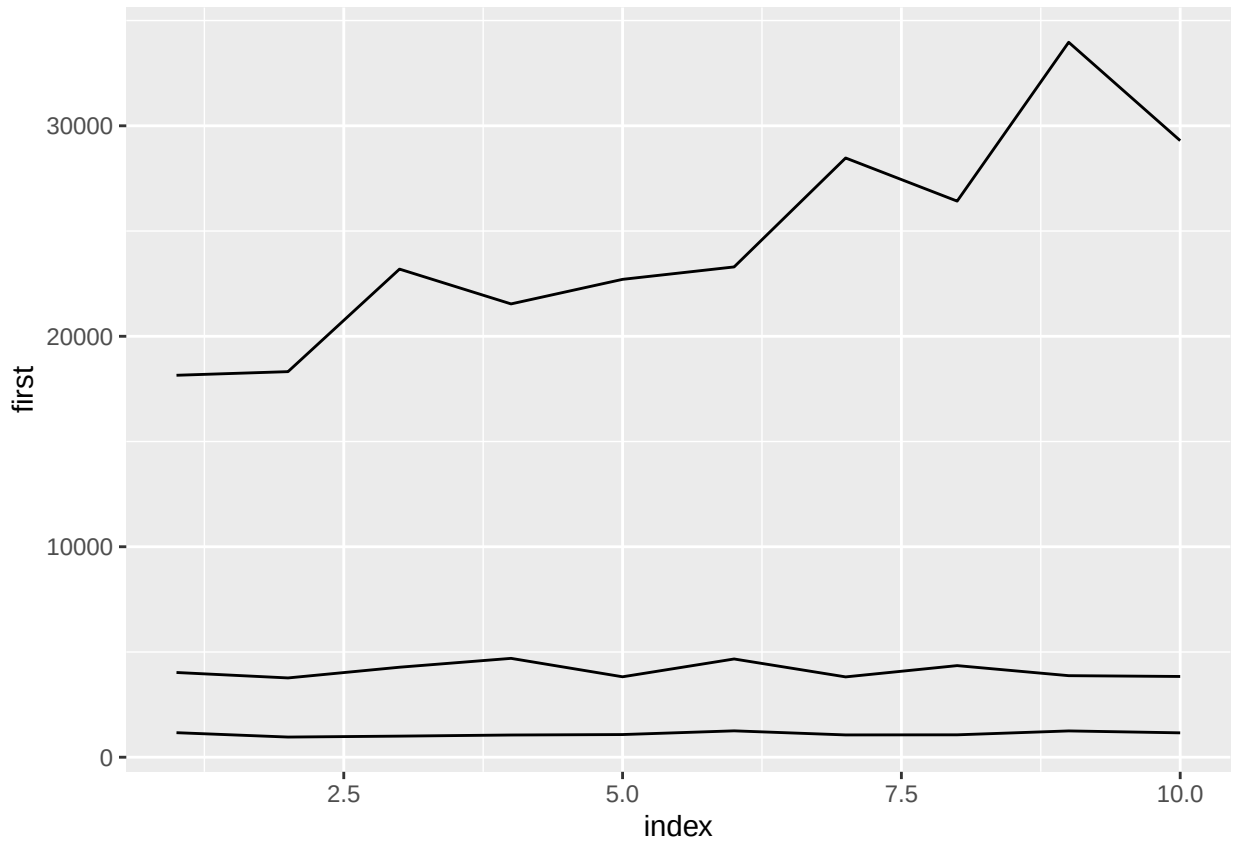
YOUR ANSWER GOES HERE:

```
library(microbenchmark)
library(ggplot2)
first = c()
second = c()
third = c()
```

```
for (i in 1:10) {
  m = matrix(1:i**2, nrow = i, ncol = i)
  first = c(first, mean(microbenchmark(colSums(m))$time))
  second = c(second,mean(microbenchmark(.colSums(m, i, i))$time))
  third = c(third, mean(microbenchmark(apply(m, 2, sum))$time))
}
index = seq(1, 10)
df = data.frame(index, first, second, third)
df
```

```
##    index   first  second    third
## 1      1 4025.07 1165.68 18147.56
## 2      2 3767.58  957.93 18319.52
## 3      3 4276.57 1001.99 23191.22
## 4      4 4697.50 1056.32 21537.51
## 5      5 3821.99 1077.11 22704.28
## 6      6 4668.41 1252.99 23295.21
## 7      7 3816.38 1059.00 28470.13
## 8      8 4352.91 1064.76 26421.53
## 9      9 3874.68 1250.01 33969.30
## 10    10 3836.34 1158.79 29295.06
```

```
ggplot(data = df, aes(x = index)) + geom_line(aes(y = first)) + geom_line(aes(y = second)) + geom_line(a
```



From the comparison created above, we can see the .colSums is consistently much faster than the other two.

Also, the larger matrix sometimes has less run time than the smaller matrix, which means the size of the matrix do not have effect on the run time.