HW7_Brown

Nick Brown

10/26/2019

Problem 2:

Part a–First, the question asked in the stackexchange was why is the supplied code not working. This question was actually never answered. What is the problem with the code? If you want to duplicate the code to test it, use the quantreg package to get the data.

Answer: It appears that the code does not work because Adam did not iterate through the bootstrapping 1000 times. He created a variable called Boot_times, but creates a for loop to run through Boot, which does not have the 1000 stored in it.

Part b–Bootstrap the analysis to get the parameter estimates using 100 bootstrapped samples. Make sure to use system.time to get total time for the analysis. You should probably make sure the samples are balanced across operators, ie each sample draws for each operator.

Answer:

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
url <- "http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat"
Sensory_raw <- read.table(url, header = F, skip = 1, fill = T,
stringsAsFactors = F)
Sensory_tidy <- Sensory_raw[-1, ]
Sensory_tidy_a <- filter(.data = Sensory_tidy, V1 %in% 1:10) %>%
rename(Item = V1, V1 = V2, V2 = V3, V3 = V4, V4 = V5,
V5 = V6)
Sensory_tidy_b <- filter(.data = Sensory_tidy, !(V1 %in%
1:10)) %>% mutate(Item = rep(as.character(1:10), each = 2)) %>%
mutate(V1 = as.numeric(V1)) %>% select(c(Item, V1:V5))
Sensory_tidy <- bind_rows(Sensory_tidy_a, Sensory_tidy_b)
colnames(Sensory_tidy) <- c("Item", paste("Person", 1:5,
sep = "_"))
Final_table <- Sensory_tidy %>% gather(Person, value, Person_1:Person_5) %>%
mutate(Person = gsub("Person_", "", Person)) %>% arrange(Item)

lm(value~Person, Final_table)
```

```
##
## Call:
## lm(formula = value ~ Person, data = Final_table)
##
```

```
## Coefficients:
## (Intercept)      Person2      Person3      Person4      Person5
##      4.5933       0.4700      -0.4267       0.6000      -0.3267
```

Part c–Redo the last problem but run the bootstraps in parallel (cl <- makeCluster(8)), don't forget to stopCluster(cl)). Why can you do this? Make sure to use system.time to get total time for the analysis.

Answer:

Create a single table summarizing the results and timing from part a and b. What are your thoughts?

Answer:

Problem 3:

Newton's method gives an answer for a root. To find multiple roots, you need to try different starting values. There is no guarantee for what start will give a specific root, so you simply need to try multiple. From the plot of the function in HW4, problem 8, how many roots are there?

Answer. Based on the plot of the function in HW4, problem 8, there are four roots, i.e., values that cross the X axis at 0.

Create a vector (length.out=1000) as a "grid" covering all the roots and extending +/-1 to either end.

Answer:

Part a. Using one of the apply functions, find the roots noting the time it takes to run the apply function.

```
newton <- function(initGuess){
fx <- 3^initGuess - sin(initGuess) + cos(5*initGuess)
fxprime <- log(3)*3^(initGuess) - cos(initGuess) - 5*sin(5*initGuess)
f <- initGuess - fx/fxprime
}
find_root<-function(x){
roots <- c(x,rep(0,1000))
i<-2
tolerance <- 0.01
move <- 2
while(move>tolerance && i < 1001){
roots[i] <- newton(roots[i-1])
move <- abs(roots[i-1]-roots[i])
if (is.na(move)) {move<-0}
i <- i+1
}
return(roots[i-1])
}
library(pracma)
```

```
##
## Attaching package: 'pracma'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     newton
```

```
## The following object is masked from 'package:purrr':
##
##     cross
```

```
x = linspace(-6,-2,1000)
transpose_x <- t(x)
find_root(-5)
```

```
## [1] -4.970865
```

```
system.time({
  roots_grid <- sapply(x, find_root)
  })
```

```
##    user  system elapsed
##   0.034   0.007   0.041
```

Part b. Repeat the apply command using the equivelant parApply command. Use 8 workers. cl <- makeCluster(8).

Create a table summarizing the roots and timing from both parts a and b. What are your thoughts?

Problem 4

Part a. What if you were to change the stopping rule to include our knowledge of the true value? Is this a good way to run this algorithm? What is a potential problem?

Part c. Make a table of each starting value, the associated stopping value, and the number of iterations it took to get to that value. What fraction of starts ended prior to 5M? What are your thoughts on this algorithm?