HW7_Kim

Due Wednesday Oct 16, 2019

2019-10-25

For each assignment, turn in by the due date/time. Late assignments must be arranged prior to submission. In every case, assignments are to be typed neatly using proper English in Markdown.

This week, we spoke about parallelizing our R code. To do this homework, we will use ARC resources. I have added you to an "allocation" called arc-train4. If you go to ondemand.arc.vt.edu, use the Rstudio interactive app on Cascades, use the basic bio version of R, arc-train4 as the account, request 10 cores for 48 hours. The first time you do this, it will take 4-20 min to create the image being used, after that, it should be quick.

Problem 2

Bootstrapping

Recall the sensory data from five operators:
http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat

Sometimes, you really want more data to do the desired analysis, but going back to the "field" is often not an option. An often used method is bootstrapping. Check out the second answer here for a really nice and detailed description of bootstrapping: https://stats.stackexchange.com/questions/316483/manually-bootstrapping-linear-regression-in-r.

What we want to do is bootstrap the Sensory data to get non-parametric estimates of the parameters. Assume that we can neglect item in the analysis such that we are really only interested in a linear model lm(y~operator).

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.

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.

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.

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

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(knitr)
library(tidyr)


## Load the raw data via webpage

  url<-"http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat"

  Sensory_raw<-read.table(url, header=F, skip=1, fill=T, stringsAsFactors = F)

  ## see the raw data table.

    kable(Sensory_raw[c(1:8),] ,caption="Raw data")
```

Table 1: Raw data

V1	V2	V3	V4	V5	V6
Item	1.0	2.0	3.0	4.0	5.0
1	4.3	4.9	3.3	5.3	4.4
4.3	4.5	4.0	5.5	3.3	NA
4.1	5.3	3.4	5.7	4.7	NA
2	6.0	5.3	4.5	5.9	4.7
4.9	6.3	4.2	5.5	4.9	NA
6.0	5.9	4.7	6.3	4.6	NA
3	2.4	2.5	2.3	3.1	2.4

```r
    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("Operator",1:5,sep="_"))

    Sensory_tidy<-Sensory_tidy %>%

    gather(Operator,value,Operator_1:Operator_5) %>%

    mutate(Operator=parse_number(Operator))%>%

    arrange(Item)

 kable(Sensory_tidy[c(1:10),], caption="Tidy Sensory Data")
```

Table 2: Tidy Sensory Data

Item	Operator	value
1	1	4.3
1	1	4.3
1	1	4.1
1	2	4.9
1	2	4.5
1	2	5.3
1	3	3.3
1	3	4.0
1	3	3.4
1	4	5.3

```
## b)

Sensory_tidy$Operator <- as.character(Sensory_tidy$Operator)

lm.fit <- lm(value~Operator, data = Sensory_tidy)

# Fitting OLS to estimate coefficients 'Beta'.

beta_sen <- summary(lm.fit)$coefficient[2:5,1]
beta_sen
```

```
##  Operator2  Operator3  Operator4  Operator5
##  0.4700000 -0.4266667  0.6000000 -0.3266667
```

```
## bootstrapping (The number of replicates : 100)

Boot_times=100

sd.boot=data.frame(c(1:100),c(1:100),c(1:100),c(1:100))

for(i in 1:Boot_times){

  # Nonparametric bootstrap

  bootdata=Sensory_tidy[sample(nrow(Sensory_tidy), size = 150, replace = TRUE),]

  sd.boot[i,]= summary(lm(value~Operator, data = bootdata))$coefficient[2:5,1]
}

elapsed.time<- system.time(for(i in 1:Boot_times){

  # Nonparametric bootstrap

  bootdata=Sensory_tidy[sample(nrow(Sensory_tidy), size = 150, replace = TRUE),]

  sd.boot[i,]= summary(lm(value~Operator, data = bootdata))$coefficient[2:5,1]
})
```

```
elapsed.time
```

```
##    user  system elapsed
##   0.110   0.001   0.111
```

```
## c)

library(parallel)

## Using parallel

cores <- max(4, detectCores()-1) ## I use the max.number of core as 4.

cl <- makeCluster(cores)

b<- system.time(for(i in 1:Boot_times){

  # nonparametric bootstrap

  bootdata=Sensory_tidy[sample(nrow(Sensory_tidy), size = 150, replace = TRUE),]

  sd.boot[i,]= summary(lm(value~Operator, data = bootdata))$coefficient[2:5,1]
})

beta_hat_b <- apply(sd.boot, 2, mean)


cbind(beta_sen,beta_hat_b)
```

```
##             beta_sen beta_hat_b
## Operator2  0.4700000  0.4803916
## Operator3 -0.4266667 -0.4066295
## Operator4  0.6000000  0.5049498
## Operator5 -0.3266667 -0.3558552
```

Based on the above R code, the higher the number of core, the faster the speed to get the values we want.

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?

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

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

```
library(ggplot2)
library(tidyr)
library(tidyverse)
```

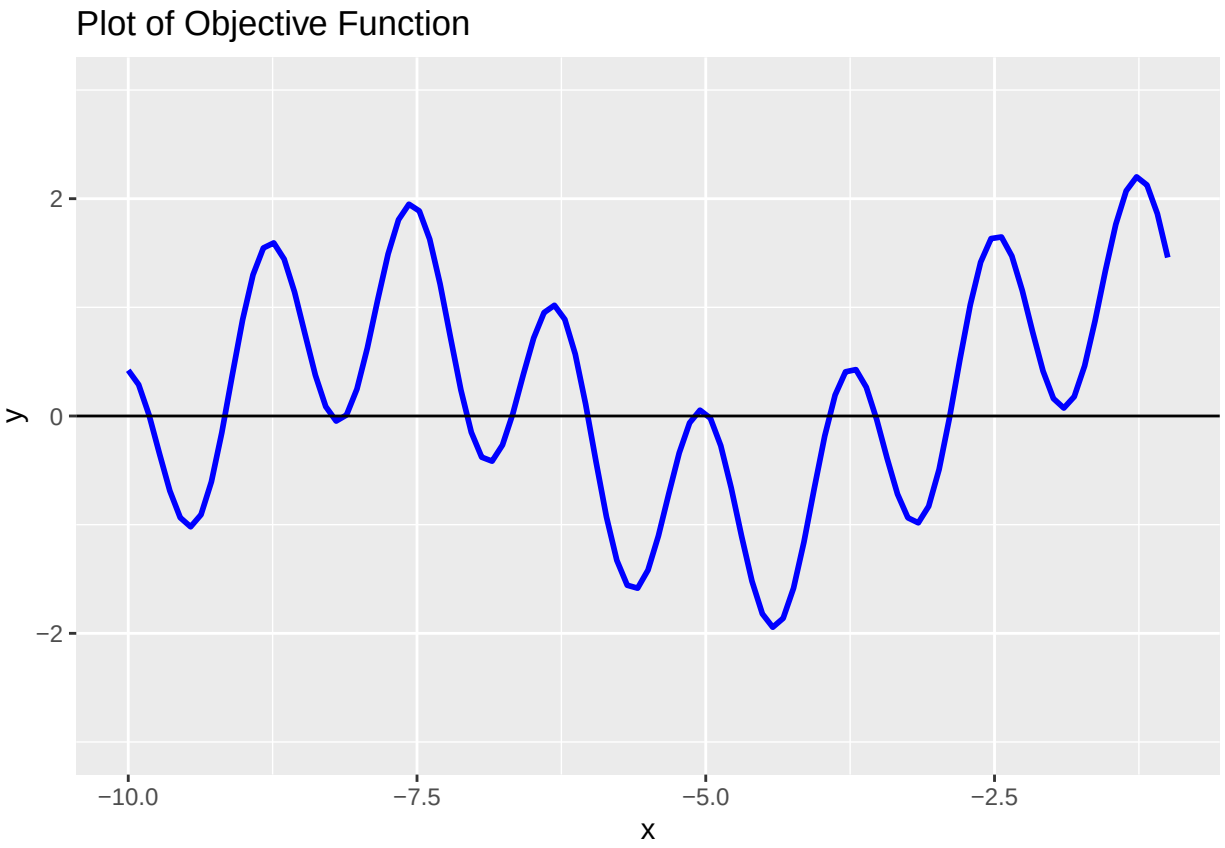
```
## -- Attaching packages ---------------------------------------------------------------------------------- t
```

```
## v tibble  2.1.3     v stringr 1.4.0
## v purrr   0.3.2     v forcats 0.4.0
```

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

```r
library(knitr)

## Let us define the objective function.

obj.function <- function(x){

  3^x - sin(x) + cos(5*x)

}



## Define the first derivative of the objective function.

dx.obj.function <- function(x) { 3^x*log(3) - cos(x) -5*sin(5*x) }



## Set the sequence denoted by x.

x <- seq(-10,-1,length.out = 1000)


#### R code to find the root(s) of the given objective function.
## stop_criteria can be changed by users
## (left_b, right_b) : the interval where we want to find the optimal solution.
## x_0 : original initial value

Newton_Method <- function(f, df, x_0, stop_criteria = 0.001,
                          max_iter = 10^3, left_b = -Inf, right_b = Inf) {


  # criteria for starting point x_0
  if (left_b > x_0 | right_b < x_0) { warning("your starting point is out of bound,
                                              please try point in the bounds")
    return(NA) }

  # variable initialization
  iters <- 0

  x_1 <- -10 ## Actual Initial value : very important
  ## if you wanna see the long progress of this algorithm.

  distance <- Inf

  # do while until

  # 1) x does not move

  # 2) exceed the maximun iteration
```

```r
  # 3) the function value become close to zero

  while ((distance > stop_criteria) &
         (iters < max_iter) &
         (abs(f(x_0)) > stop_criteria)){

     # newton method

    x_1 <- x_0 - ( f(x_0) / df(x_0) )

    # out of bound case

    if (x_1 <= left_b) {

      { x_1 <- left_b }

    }

    if (right_b <= x_1) {

      { x_1 <- right_b }

    }


    distance <- abs(x_0 - x_1)

    iters <- iters + 1

    x_0 <- x_1 }

  # return the final point

#  if (x_0 == left_b | x_0 == right_b){ warning("\n", "Final point is on the boundary", "\n") }

#  if (iters > max_iter){ warning("\n", "Cannot find the root during the iteration.", "\n") }

#  cat("Final value of function:", f(x_0), "\n")

  x_0

  }

## Drawing the shape of the objective function.

p <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))

p <- p + stat_function(fun= obj.function, color="blue", size=1.0) +
  geom_hline(yintercept=0) + xlim(-10, -1) + ylim(-3, 3)

p + ggtitle("Plot of Objective Function")
```




```
## Do simulate the following code to find the solution of objective function
## in the set : (-10,1) (This is arbitrarily set.)

## c.f) Actually, this code is really influenced by the initial value, so it is important
## to set a variety of initial values to find the global minimum of the objective function.

Newton_Method(f = obj.function, df = dx.obj.function, x_0 = -10, left_b = -8, right_b = -2)
```

```
## Warning in Newton_Method(f = obj.function, df = dx.obj.function, x_0 = -10, : your starting point is
##                                             please try point in the bounds
```

```
## [1] NA
```

```
## We can apply functions simultaneously to find the optimal solutions via NR.

roots <- sapply(x, Newton_Method, f = obj.function, df = dx.obj.function)

## Record the time spent to find the solutions via Newton Rhapson Method.
system.time(sapply(x, Newton_Method, f = obj.function, df = dx.obj.function))
```

```
##    user  system elapsed
##   0.026   0.000   0.026
```

```
## plotting the points on the objective function's graph.


roots_p <- data.frame(x_star = roots, y_star = obj.function(roots))

kable(roots_p$x_star, caption = "Solutions
```

```
    using parSapply with elapsed system.time : 0.034")
```

Table 3: Solutions using parSapply with elapsed system.time : 0.034

x
-9.1629858
-9.1629085
-10.2101808
-9.8174207
-9.8173265
-9.8175437
-9.8174661
-9.8174660
-9.8174683
-9.8174700
-9.8174708
-9.8174712
-9.8172422
-9.8173524
-9.8174147
-9.8174472
-9.8174626
-9.8174689
-9.8174709
-9.8172907
-9.8174644
-9.8174202
-9.8172156
-9.8174710
-9.8174703
-9.8174693
-9.8174679
-9.8174666
-9.8174657
-9.8174655
-9.8174663
-9.8174681
-9.8174703
-9.8175817
-9.8174683
-9.8174554
-9.8174230
-9.8173552
-9.8172249
-9.8174712
-9.8174704
-9.8174682
-9.8174613
-9.8174409
-9.8173802
-9.8174714
-9.8174707
-9.8174667

x
-9.8174406
-9.1629858
-11.2573573
-10.2103493
-10.2103357
-10.2101882
-10.2101794
-7.0684836
-11.2573350
-11.3888219
-12.9590615
-21.7293484
-3.9301520
-6.6760570
-8.1158629
-8.2471163
-8.2466079
-8.1156978
-5.1077025
-9.8174685
-9.1629858
-9.1629786
-9.1629858
-9.1629058
-9.1629732
-9.1629844
-9.1629859
-9.1629859
-9.1629858
-9.1629858
-9.1629858
-9.1629859
-9.1629859
-9.1628522
-9.1628993
-9.1629326
-9.1629548
-9.1629689
-9.1629773
-9.1629819
-9.1629843
-9.1629854
-9.1629858
-9.1629859
-9.1629458
-9.1629855
-9.1629412
-9.1628452
-9.1629859
-9.1629858
-9.1629858
-9.1629858

x
-9.1629858
-9.1629129
-9.1629858
-9.1629854
-9.1629841
-9.1629811
-9.1629749
-9.1629632
-9.1629425
-9.1629071
-9.1628485
-9.1629859
-9.1629858
-9.1629857
-9.1629854
-9.1629846
-9.1629826
-9.1629777
-9.1629660
-9.1629382
-9.1628737
-9.1629859
-9.1629858
-9.1629858
-9.1629859
-16.4933823
-7.0685345
-16.1005244
-9.8174654
-9.8172960
-9.8174695
-9.8174674
-10.2101794
-10.2101831
-11.2573607
-12.3045439
-2.8870576
-69.5077321
-3.9301323
-4.9715031
-5.1074756
-5.1075398
-7.0685877
-6.0211546
-8.2468057
-8.1152589
-8.1158581
-8.1153807
-8.1157819
-8.1152518
-8.1152661
-8.1156278

x
-8.1158321
-8.2466082
-8.2467134
-8.2468406
-8.2466084
-8.2466504
-8.2466111
-8.2466437
-8.2467079
-8.2467947
-8.2468922
-8.2469898
-8.2470795
-8.2471564
-8.2472175
-8.2466078
-8.2466081
-8.2466082
-8.2466081
-8.2466080
-8.2472470
-8.2472076
-8.2471605
-8.2471078
-8.2470515
-8.2469936
-8.2469359
-8.2468801
-8.2468275
-8.2467796
-8.2467372
-8.2467010
-8.2466713
-8.2466481
-8.2466309
-8.2466191
-8.2466117
-8.2472374
-8.2469639
-8.2467750
-8.2466643
-8.2466160
-8.2468169
-8.2466913
-8.2466182
-8.2467716
-8.2466115
-8.2467188
-8.2466170
-8.2472202
-2.8870577
-8.1153740

x
-8.1158610
-8.1157921
-8.1158636
-8.1157563
-8.1158610
-8.1158556
-8.1154671
-8.1158475
-8.1157806
-8.1156397
-8.1154097
-8.1158630
-8.1158569
-8.1158463
-8.1158298
-8.1158059
-8.1157738
-8.1157328
-8.1156828
-8.1156238
-8.1155565
-8.1154818
-8.1154010
-8.1153156
-8.1152274
-8.1158636
-8.1158626
-8.1158615
-8.1158604
-8.1158594
-8.1158584
-8.1158576
-8.1158570
-8.1158568
-8.1158568
-8.1158573
-8.1158581
-8.1158592
-8.1158606
-8.1158622
-8.1158637
-8.1153056
-8.1154621
-8.1156118
-8.1157384
-8.1158251
-8.1158629
-8.1159983
-8.1158472
-8.1158630
-8.1158603
-9.1628271

x
-8.2467057
-8.2466107
-8.2466644
-8.2466148
-8.2467321
-8.2470745
-8.2466080
-8.2469898
-8.1158346
-6.0211517
-9.1629823
-9.1629584
-9.8174674
-9.8174116
-12.3045708
-18.5876718
-8.1158621
-3.5287084
-3.9302001
-4.9714979
-4.9715041
-6.0211539
-6.0211546
-8.1154344
-6.6760570
-6.6760555
-6.6760347
-6.6759779
-6.6760568
-6.6760578
-3.9301137
-11.3882981
-7.0684836
-7.0685786
-7.0684976
-7.0684854
-7.0684837
-7.0686052
-7.0685239
-7.0684959
-7.0684869
-7.0684842
-7.0684836
-7.0685503
-7.0684878
-7.0684916
-7.0685298
-7.0685821
-7.0686360
-7.0686840
-7.0684836
-7.0684836

x
-7.0684836
-7.0684836
-7.0684836
-7.0684836
-7.0687138
-7.0686855
-7.0686545
-7.0686229
-7.0685924
-7.0685644
-7.0685400
-7.0685200
-7.0685048
-7.0684943
-7.0684879
-7.0684847
-7.0684837
-7.0685086
-7.0685448
-7.0684839
-7.0684869
-7.0684979
-7.0685275
-7.0685938
-7.0684836
-7.0684840
-7.0684851
-7.0684886
-7.0684977
-7.0685198
-7.0685671
-7.0686525
-7.0684836
-7.0686713
-7.0684850
-6.6760608
-8.2466399
-8.1158637
-11.2573150
-2.8870575
-4.9715004
-6.0211521
-2.8870029
-2.8870476
-6.6760559
-6.6760195
-6.6760598
-6.6760550
-6.6760562
-6.6760584
-6.6760599
-6.6760606

x
-6.6760609
-6.6758704
-6.6759643
-6.6760163
-6.6760427
-6.6760547
-6.6760594
-6.6760608
-6.6759595
-6.6760609
-6.6759619
-6.6760609
-6.6760605
-6.6760596
-6.6760584
-6.6760571
-6.6760558
-6.6760550
-6.6760551
-6.6760563
-6.6760583
-6.6760604
-6.6760608
-6.6760556
-6.6760384
-6.6759979
-6.6759155
-6.6760609
-6.6760606
-6.6760596
-6.6760565
-6.6760471
-6.6760190
-6.6759358
-6.6760609
-6.6760597
-6.6760549
-6.6760586
-6.6760603
-6.0211461
-7.0685213
-7.0684950
-7.0685472
-7.0685259
-6.0211515
-8.1158570
-8.2467769
-10.2102045
-50.6581242
-12.3045568
-3.9301324
-4.9715003

x
-5.1075159
-5.1079058
-4.9714640
-12.9590641
-7.0686007
-6.0211547
-6.0211524
-6.0211546
-6.0211050
-6.0211471
-6.0211539
-6.0212662
-6.0211546
-6.0211546
-6.0211546
-6.0211546
-6.0211547
-6.0211547
-6.0210358
-6.0210787
-6.0211085
-6.0211282
-6.0211404
-6.0211476
-6.0211515
-6.0211535
-6.0211543
-6.0211546
-6.0210111
-6.0211348
-6.0211497
-6.0210894
-6.0209859
-6.0211547
-6.0211546
-6.0211546
-6.0211546
-6.0211547
-6.0211847
-6.0211546
-6.0211540
-6.0211523
-6.0211486
-6.0211411
-6.0211274
-6.0211034
-6.0210627
-6.0209957
-6.0211547
-6.0211546
-6.0211544
-6.0211540

x
-6.0211530
-6.0211505
-6.0211443
-6.0211296
-6.0210946
-6.0210148
-6.0211546
-6.0211546
-6.0211532
-6.0211530
-8.2466080
-34.2957195
-7.0684973
-6.6758392
-6.6760557
-6.6760570
-9.8174679
-7.0684842
-6.6758423
-8.1158616
0.6777605
-14.3989560
-124.4857076
-4.9715047
-3.5287223
-3.5287227
-3.9301394
-3.9301495
-11.2567632
-4.9712892
-4.9715022
-4.9715031
-4.9712574
-4.9714814
-4.9713421
-4.9713659
-4.9714837
-4.9715002
-5.1075024
-5.1074623
-5.1075464
-5.1078267
-5.1075459
-5.1074448
-5.1074758
-5.1075329
-5.1076079
-5.1076909
-5.1077729
-5.1078475
-5.1079106
-5.1079598

x
-5.1079942
-5.1080138
-5.1080192
-5.1080118
-5.1079930
-5.1079645
-5.1079281
-5.1078856
-5.1078390
-5.1077901
-5.1077404
-5.1076917
-5.1076453
-5.1076024
-5.1075641
-5.1075309
-5.1075032
-5.1074812
-5.1074646
-5.1074529
-5.1074453
-5.1074408
-5.1078671
-5.1076593
-5.1075277
-5.1074611
-5.1080035
-5.1074404
-5.1079909
-5.1074909
-5.1078331
-5.1074600
-5.1077304
-5.1074882
-5.1074403
-3.5287227
-4.9712957
-4.9715048
-4.9714542
-4.9715047
-4.9714069
-4.9715015
-4.9714919
-4.9711422
-4.9714906
-4.9714313
-4.9713046
-4.9710952
-4.9715041
-4.9714992
-4.9714905
-4.9714767

x
-4.9714568
-4.9714297
-4.9713948
-4.9713520
-4.9713013
-4.9712433
-4.9711787
-4.9711086
-4.9710344
-4.9709577
-4.9715049
-4.9715042
-4.9715034
-4.9715026
-4.9715019
-4.9715012
-4.9715007
-4.9715003
-4.9715002
-4.9715003
-4.9715007
-4.9715014
-4.9715023
-4.9715033
-4.9715044
-4.9709550
-4.9710881
-4.9712212
-4.9713422
-4.9714362
-4.9714910
-4.9712468
-4.9712734
-4.9714133
-4.9714807
-4.9714046
-5.1080178
-5.1074555
-5.1075901
-5.1075697
-5.1074540
-5.1075946
-5.1079177
-5.1078914
-5.1074864
-3.5286898
-6.0211546
-6.0211547
-12.3045344
-4.9709996
-8.1158568
-10.2101799

x
-23.8234789
-3.9301147
-7.0686536
-2.8870576
-3.5287196
-3.5287226
-2.8870505
-2.8869242
-4.9714656
-3.5286393
-3.5287208
-3.5287212
-3.5287226
-3.5287224
-2.8870576
-6.0210950
-3.9301137
-3.9303192
-3.9301547
-3.9301195
-3.9301143
-3.9301136
-3.9301949
-3.9301405
-3.9301218
-3.9301158
-3.9301140
-3.9303090
-3.9301638
-3.9301158
-3.9301228
-3.9301578
-3.9302037
-3.9302500
-3.9302902
-3.9303211
-3.9303412
-3.9301137
-3.9301137
-3.9303397
-3.9303228
-3.9303005
-3.9302748
-3.9302475
-3.9302202
-3.9301945
-3.9301716
-3.9301523
-3.9301372
-3.9301262
-3.9301192
-3.9301154

x
-3.9301139
-3.9301991
-3.9301229
-3.9301137
-3.9301152
-3.9301217
-3.9301408
-3.9301860
-3.9302797
-3.9301138
-3.9301144
-3.9301162
-3.9301212
-3.9301336
-3.9301615
-3.9302163
-3.9303006
-3.9301136
-3.9301384
-3.9302979
-2.8870446
-4.9714451
-5.1078615
-10.2103020
-3.5287208
-4.9710282
-2.8870412
-14.5298997
-35.3430171
-3.5287161
-3.5286496
-3.5288054
-3.5287198
-3.5287196
-3.5287209
-3.5287219
-3.5287225
-3.5287227
-3.5285319
-3.5286233
-3.5286751
-3.5287024
-3.5287153
-3.5287206
-3.5287224
-3.5285407
-3.5287135
-3.5286819
-3.5285019
-3.5287225
-3.5287221
-3.5287213

x
-3.5287204
-3.5287197
-3.5287193
-3.5287196
-3.5287206
-3.5287220
-3.5287219
-3.5287210
-3.5287128
-3.5286911
-3.5286446
-3.5285537
-3.5287227
-3.5287224
-3.5287213
-3.5287181
-3.5287084
-3.5286796
-3.5285942
-3.5287227
-3.5287216
-3.5287200
-3.5285563
-4.9714389
-3.9301186
-3.9301246
-3.9301139
-3.9302516
-3.9301168
-6.6760593
-4.9714882
-6.0211411
-9.8174681
-3.5287221
-11.3883196
-8.1157501
-5.1077218
-3.5287052
-3.9302610
-3.9302935
-3.5287196
-2.8870574
-2.8870500
-2.8870576
-2.8869927
-2.8870495
-2.8870572
-2.8870577
-2.8870575
-2.8870574
-2.8870575
-2.8870576

x
-2.8870576
-2.8870577
-2.8869027
-2.8869593
-2.8869983
-2.8870237
-2.8870395
-2.8870487
-2.8870537
-2.8870562
-2.8870572
-2.8870576
-2.8869187
-2.8870424
-2.8870475
-2.8869679
-2.8870577
-2.8870576
-2.8870575
-2.8870575
-2.8870574
-2.8870575
-2.8870576
-2.8870017
-2.8870576
-2.8870567
-2.8870542
-2.8870488
-2.8870381
-2.8870187
-2.8869849
-2.8869286
-2.8870577
-2.8870576
-2.8870575
-2.8870572
-2.8870563
-2.8870542
-2.8870492
-2.8870373
-2.8870091
-2.8869431
-2.8870577
-2.8870575
-2.8870575
-2.8870312
-3.5287226
-6.0211321
-5.1076530
-4.9710205
-3.5287179
-3.5287207

x
-3.5287131
-6.6760606
-3.9301623
-2.8870575
-14.3989488
-7.0684837
-11.2573665
-6.6760606
-3.9301530
-45.9462003
-3.9301827
-4.9711313
-6.6760595
-2.8870574
-6.0211546
-25.9181586
-7.0685950
-2.8870577
-2.8870339
-2.8870576
-3.5287214
-2.8870577
-2.8869959
-2.8870569
-2.8870106
-6.6760602
-3.9301456
-2.8870577
-6.0211471
-2.8870575
-2.8870228
-4.9715045
-3.9301137
-4.9714396
-3.5287227
-11.2569644
-9.8174657
-9.8174046
-3.5287182
-11.3884646
-3.9301496
-3.5287226
-3.5287215
-2.8870577
-9.8174652
-3.9301426
-2.8870576
-4.9714085
-3.9301563
-3.5287226
-10.2101794
-12.3045710

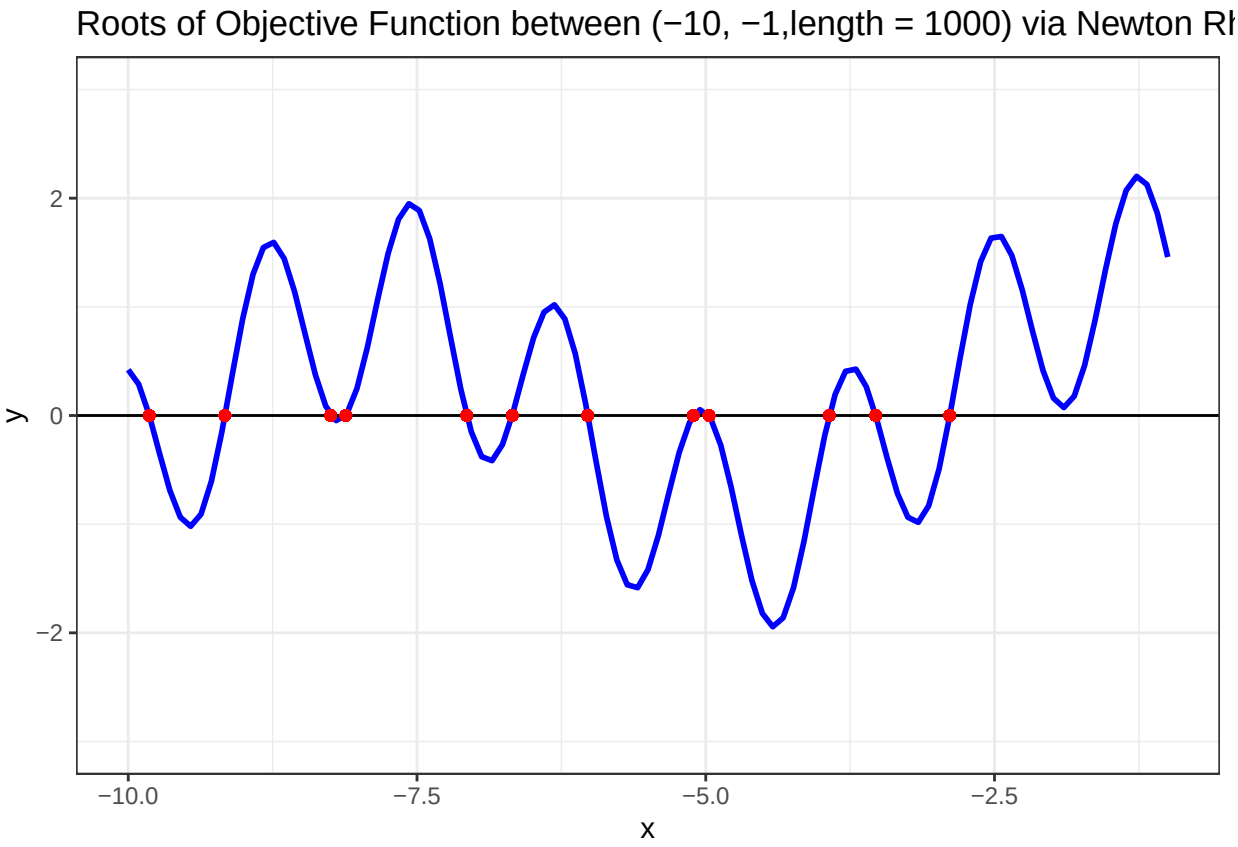
x
-4.9712651
-3.9301179
-2.8870564
-4.9713946
-2.8870574
-3.9301139
-2.8870574
-2.8870577
-5.1077664
-14.3988028
-3.5287207
-3.5286419
-5.1074516
-2.8870466
-16.1005812
0.6777605
-5.1075821
-3.9302983
-3.9301223
-2.8870182
-6.0211497
-2.8870575
-3.9301538
-12.3045690
-5.1074571
-2.8870556
-2.8870577
-2.8870499
-3.9303100
-5.1079299
-2.8870574
-16.1004393
-2.8870574
-2.8870575
-2.8870577
-3.9302890
-3.5287223
-7.0686115
-3.5287202
-3.9301631
-7.0685675
-16.4934575
-9.1629858
-2.8870577
-2.8870213
-2.8870577
-3.5287208
-2.8869830
-3.5287107
-2.8870458
-2.8870486
-3.5287222

x
-2.8870472
-4.9712600
-2.8870576
-17.6715216
-4.9715023
-6.6761943
-2.8869963
-3.9302266
-2.8870558
-9.8174680
-8.1157579
-3.5287195
-2.8870574
-4.9714419
-3.9301140
-8.2466198
-3.9301695
-2.8870576
-2.8870576
-9.1629805
-4.9715048
-17.6719714
-2.8869537
-7.0684836
-2.8870575
-2.8870562
-3.5287219
-2.8870575
-2.8870575
-3.9301136
-6.0211387
-2.8870572
-2.8869432
-2.8870449
-3.5287200
-3.9302768
-5.1079220
-8.2471614
-26.9650106
-6.0211546
-2.8870514
-3.5286691
-2.8870576
-6.6758938
-6.6758811
-3.5286534
-3.5287227
-3.5287226
-2.8870577
-3.5286903
-3.9301859
-8.2468464

x
-9.1628380
-2.8870575
-27.0962413
-2.8870476
-2.8870346
-3.9302817
-9.1629578
-20.8133126
-31.8085698
-2.8870577
-2.8870476
-3.9303129
-5.1077176
-2.8870454
-3.5286904
-6.0210994

```
p <- p + geom_point(data = roots_p, aes(x = x_star, y = y_star),

                    color = "red", size = 1.5) + theme_bw()

p + ggtitle("Roots of Objective Function between (-10, -1,length = 1000) via Newton Rhapson Method.")
```

```
## Warning: Removed 57 rows containing missing values (geom_point).
```



As you can see the above graph, the 12 red colored points are the solutions (actually, there are numerous

approximated value with 6-digits in the table 1) obtained by Newton Rhapson Method. Actually, When x value is greater than -2, there is no optimal solution such that equation meets zero value ($\because$ the shape of the graph of the objective function is greater than 0.)

Part b. Repeat the apply command using the equivelant parApply command. Use 8 workers.

`cl <- makeCluster(8).`

```
library(parallel)
library(knitr)

cl <- makeCluster(8)

kable(parSapply(cl,-10:1,Newton_Method, f = obj.function, df = dx.obj.function),caption="Solutions
      using parSapply with elapsed system.time : 0.049")
```

Table 4: Solutions using parSapply with elapsed system.time : 0.049

x
-9.162986
-9.162986
-8.115683
-7.068484
-6.020986
-4.971505
-3.930172
-2.887058
-5.107766
-6.021099
-20.682007
-3.528721

```
system.time(parSapply(cl,-10:1,Newton_Method, f = obj.function, df = dx.obj.function))
```

```
##    user  system elapsed
##   0.005   0.001   0.005
```

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

Since the size of the given data is not large, I think it is more time-saving to just use the 'sapply' function than to use the 'parsapply' function. If the size of the data is large, the amount of operations required will be heavy so that the using parsapply function will be more efficient.

Problem 4

Gradient descent, like Newton's method, has "hyperparameters" that are determined outside the algorithm and there is no set rules for determing what settings to use. For gradient descent, you need to set a start value, a step size and tolerance. Using a step size of $1e^{-7}$ and tolerance of $1e^{-9}$, try 10000 different combinations of β_0 and β_1 across the range of possible β's +/-1 from true making sure to take advantages of parallel computing opportunities. In my try at this, I found starting close to true took 1.1M iterations, so set a stopping rule for 5M and only keep a rolling 1000 iterations for both β's. If this is confusing, see the solution to the last homework.

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?

```
 set.seed(11)

x <- as.numeric(Sensory_tidy$Operator)

y <- Sensory_tidy$value

m <- length(y)

X<-cbind(rep(1, length(Sensory_tidy$value)), x)

theta<-c(0,2)



compCost<-function(X, y, theta){
   m <- length(y)
   J <- sum((X%*%theta- y)^2)/(2*m)
return(J)
}


gradDescent<-function(X, y, theta, alpha, num_iters){
       m <- length(y)
  J_hist <- rep(0, num_iters)
  for(i in 1:num_iters){

# this is a vectorized form for the gradient of the cost function
# X is a 150x5 matrix, theta is a 5x1 column vector, y is a 150x1 column vector
# X transpose is a 5x100 matrix. So t(X)%*%(X%*%theta - y) is a 5x1 column vector

    theta <- theta - alpha*(1/m)*(t(X)%*%(X%*%theta - y))

# this for-loop records the cost history for every iterative move of the gradient descent,
# and it is obtained for plotting number of iterations against cost history.

    J_hist[i]  <- compCost(X, y, theta)
  }
# for a R function to return two values, we need to use a list to store them:

  results<-list(theta, J_hist)
  return(results)
}



alpha <- exp(1)^(-7)  ## Learning Rate
```

```r
num_iters <- 5000000 ## Max.iteration number

results <- gradDescent(X, y, theta, alpha, num_iters)

theta <- results[[1]]

cost_hist <- results[[2]]


## Print the result values and Plots realted to the results.

head(cost_hist)
```

```
## [1] 7.333243 7.272895 7.213827 7.156012 7.099422 7.044033
```

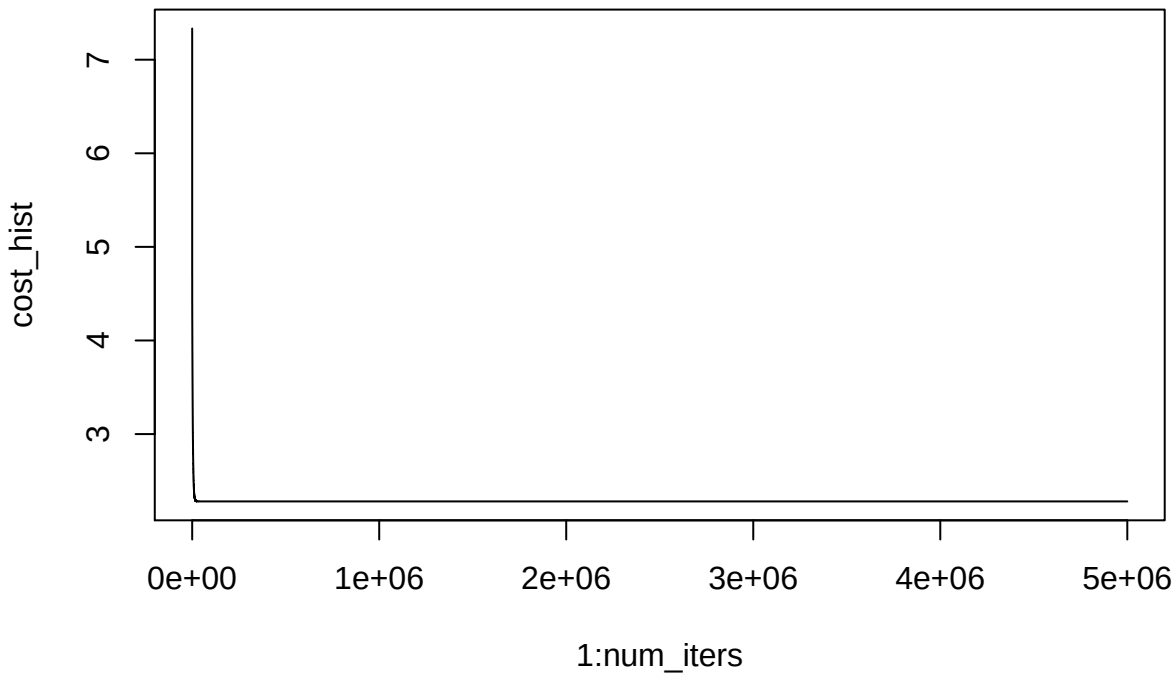
```r
print(theta)
```

```
##           [,1]
##    4.81366667
## x -0.05233333
```

```r
length(cost_hist)
```

```
## [1] 5000000
```

```r
plot(1:num_iters, cost_hist, type = 'l')
```



I think it would be a good way when we set the stopping rule to include our knowledge of the true value. It would be more efficient way to figure out where the appropriate error, and the convergence can be made.