HW7_Lian_Jiayi

Jiayi Lian

October 12, 2019

P2

a) Because generated random sample is fake random sample. It is generated by fixed algorithm. To generate different random sample, we need a new seed.

```
# b) Use Boostrap to estimate beta
## data mungo
url_sensory<-"http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat"
Sensory <-read.table(url_sensory, header=F, skip=1, fill=T, stringsAsFactors = F)
Sen_2 <- Sensory[-1, ]
Sen_2_a <- Sen_2 %>% filter(V1 %in% 1:10) %>%
                     rename(Item=V1, V1=V2, V2=V3, V3=V4, V4=V5, V5=V6)
Sen_2_b <- Sen_2 %>% filter(!(V1 %in% 1:10)) %>%
                     mutate(Item=rep(as.character(1:10), each=2)) %>%
                     mutate(V1=as.numeric(V1)) %>%
                     select(c(Item, V1:V5))
Sen_final <- Sen_2_a %>% full_join(Sen_2_b)
```

```
## Joining, by = c("Item", "V1", "V2", "V3", "V4", "V5")
```

```
y=c(Sen_final[,2],Sen_final[,3],Sen_final[,4],Sen_final[,5],Sen_final[,6])
operator<-c(rep(1,30),rep(2,30),rep(3,30),rep(4,30),rep(5,30))
sensory<-data.frame(y=y,operator=operator)
## boostrap
boost_times<-100
set.seed(992)
beta<-matrix(rep(0,2*boost_times),100,2)
time_original<-system.time(for (i in 1:boost_times) {
  bootdata=sensory[sample(nrow(sensory), size = nrow(sensory), replace = TRUE),]
  beta[i,]= coef(summary(lm(y~operator, data = bootdata)))[,1]
})

# c)parallel
## Create a cluster via makeCluster
cl <- makeCluster(8)
boost_b<-function(n)
{
  bootdata=sensory[sample(nrow(sensory), size = nrow(sensory), replace = TRUE),]
  return(coef(summary(lm(y~operator, data = bootdata)))[,1])
}
time_parallel<-system.time(for (i in 1:boost_times) {
  lapply(1:100,FUN=boost_b)
})
## Stop the cluster
stopCluster(cl)
## table
table<-rbind(time_original,time_parallel)[,1:3]
table
```

```
##                user.self sys.self elapsed
## time_original      0.10        0    0.09
## time_parallel      9.07        0    9.06
```

We can see that parallel can help us increase the efficiency of computing.

P3

```
# a) newton method with apply fun
target_fun<-function(x)
  {
  return(3^x-sin(x)+cos(5*x))
}
## There are 4 roots, which are -4.9, -3.9, -3.5, -2.8
## Interval [-6,-1.8] can cover all roots and extend -1/+1 to either end.
v<-seq(-6,-1.8,length.out = 1000)
dif<-function(x)
{
  return(log(3)*3^x-cos(x)-5*sin(5*x))
}
newton<-function(x)
{
  x0 = x
  path=0
  y=3^x0-sin(x0)+cos(5*x0)
  i=0
  while (abs(y)>1e-6 & !is.na(y)) {
    if(i>1000)
    {
      print("Out of the interval")
      break
    }
    x_new=x0-y/dif(x0)
    y=3^x_new-sin(x_new)+cos(5*x_new)
    x0=x_new
    path=x_new
    i=i+1
  }
  return(path)
}


time_original<-system.time(
{root1<-sapply(v,newton)}
)

# b) parallel
## Create a cluster via makeCluster
cl <- makeCluster(8)
time_parallel<-system.time({root2<-sapply(v,newton)})
## Stop the cluster
stopCluster(cl)
## table of time
table<-rbind(time_original,time_parallel)[,1:3]
```

```
table
```

```
##                user.self sys.self elapsed
## time_original       0.05        0    0.05
## time_parallel       0.01        0    0.02
```

```
## show roots
### Since these roots are numeric, for a single root, there are several rows. I just pick some rows out
cbind(General=root1,paralell=root2)[seq(1,1000,20),]
```

```
##            General    paralell
##  [1,]   -6.021155   -6.021155
##  [2,]   -6.021155   -6.021155
##  [3,]   -6.021155   -6.021155
##  [4,]   -6.021155   -6.021155
##  [5,]   -7.068484   -7.068484
##  [6,]   -3.930114   -3.930114
##  [7,]   -4.971508   -4.971508
##  [8,]   -5.107437   -5.107437
##  [9,]   -5.107437   -5.107437
## [10,]   -5.107438   -5.107438
## [11,]   -5.107437   -5.107437
## [12,]   -5.107437   -5.107437
## [13,]   -4.971508   -4.971508
## [14,]   -4.971508   -4.971508
## [15,]   -4.971508   -4.971508
## [16,]   -4.971508   -4.971508
## [17,]   -4.971508   -4.971508
## [18,]   -4.971508   -4.971508
## [19,]   -5.107437   -5.107437
## [20,]   -3.528723   -3.528723
## [21,]   -3.528723   -3.528723
## [22,]   -3.930114   -3.930114
## [23,]   -3.930114   -3.930114
## [24,]   -3.930114   -3.930114
## [25,]   -3.930114   -3.930114
## [26,]   -3.930114   -3.930114
## [27,]   -3.930114   -3.930114
## [28,]   -3.528723   -3.528723
## [29,]   -3.528723   -3.528723
## [30,]   -3.528723   -3.528723
## [31,]   -3.528723   -3.528723
## [32,]   -3.528723   -3.528723
## [33,]   -3.528723   -3.528723
## [34,]   -3.930114   -3.930114
## [35,]   -8.115867   -8.115867
## [36,]   -2.887058   -2.887058
## [37,]   -2.887058   -2.887058
## [38,]   -2.887058   -2.887058
## [39,]   -2.887058   -2.887058
## [40,]   -2.887058   -2.887058
## [41,]   -2.887058   -2.887058
## [42,]   -3.528723   -3.528723
## [43,]   -3.930114   -3.930114
## [44,]   -3.930114   -3.930114
```

```
## [45,]   -3.528723   -3.528723
## [46,]   -9.817471   -9.817471
## [47,]   -3.528723   -3.528723
## [48,]  -17.540559  -17.540559
## [49,]   -2.887058   -2.887058
## [50,] -131.685092 -131.685092
```

P4

a) I have a question about how to include knowledge of true value into stopping rule. Should I add some command like "abs(theta0-true_theta0)>tollerance" ?

```
## b)
cl <- makeCluster(8)

collect_data<-matrix(rep(0,5000),nrow = 1000, ncol = 5)
tollerance<-1e-9
m=100
alpha=1e-7
x=rep(1:10,10)
theta <- as.matrix(c(1,2),nrow=2)
X <- cbind(1,rep(1:10,10))
h <- X%*%theta+rnorm(100,0,0.2)

Collect_Para<-function(n){
  theta0=3*runif(1)
  theta1=3*runif(1)
  ori_beta0=theta0
  ori_beta1=theta1
  theta0_old=0
  theta1_old=0
  Z=0
  while (abs(theta0-theta0_old)>tollerance && abs(theta1-theta1_old)>tollerance && Z<5*10^6) {
    theta0_old=theta0
    theta1_old=theta1
    Z=Z+1
    theta0=theta0-alpha*sum(theta0+theta1*x-h)/m
    theta1=theta1-alpha*sum((theta0+theta1*x-h)*x)/m
  }
  return(c(ori_beta0,ori_beta1,theta0,theta1,Z))
}

collect_data<-matrix(unlist(lapply(1:1000,FUN=Collect_Para)),ncol = 5)
stopCluster(cl)
colnames(collect_data)<-c("Beta0_Ori","Beta1_Ori","Beta0","Beta1","Iteration")
# show 1st 10 iterations' results
head(collect_data)
```

```
##         Beta0_Ori    Beta1_Ori        Beta0        Beta1    Iteration
## [1,] 2.897367e+00 1.081817e+00 1.981653e+00 3.104018e-01 6.332264e-02
## [2,] 1.312002e+00 1.591457e+00 9.499060e-01 1.172635e+00 1.481945e+00
## [3,] 2.927051e+00 1.132729e+00 2.084851e+00 4.602473e-01 1.828482e-01
## [4,] 1.646077e+00 1.972317e+00 1.799204e+00 2.076906e+00 2.116534e+00
## [5,] 4.285510e+05 9.740500e+05 7.650870e+05 1.849681e+06 1.686762e+06
## [6,] 2.513073e+00 1.526063e+00 2.176077e+00 8.275866e-02 2.316210e+00
```

```r
# How many iterations have stopped within 5000000 steps
sum(collect_data[,5]<5*10^6)
```

```
## [1] 999
```