HW7_Wei_Yanran

Problem 2: Sums of Squares

Four methods are used in calculating sums of squares. These methods are loop, vector operations, dopar and parSapply methods. The time for running these methods are displayed in the table below.The dataset is 1e+06 data selected from normal distribution with mean of 1 and standard deviation of 1. I shrink the dataset from 1e+7 to 1e+6 because my computer will get stuck when running parSapply method.

Table 1: Time for Different Methods

	user.self	sys.self	elapsed	Sum of Squares
Loop_Method	0.23	0	0.25	1002631
Vector_Method	0.07	0.03	0.11	1002631
Dopar_Method	0.11	0.07	0.65	1002631
ParSapply_Method	2.34	1	6.69	1002631

From the table, we can see that vector operations method perform the best, then dopar method and loop method. Parsapply method perfom worst. Sum of squares value is same for all four methods. The cluster I set in Dopar and ParSapply method is 2. The code for Dopar Method should be foreach(1:1e+06) %dopar% sum((y - mean_y)^2). It takes so long time that my computer get stuck, so I changed it to foreach(1) %dopar% sum((y - mean_y)^2).

Problem 3

Table 2: Running time for GD and Linear Methods

	user.self	sys.self	elapsed	user.child	sys.child
GD_time	0.54	0.02	2.27	NA	NA
Linear_Time	0.02	0	0.02	NA	NA

Table 3: Coefficients of GD Method

	Tolerance	theta1	theta2
result.1	1	1	2
result.2	0.1	1	2
result.3	0.01	1	2
result.4	0.001	1	2
result.5	1e-04	0.9999	2
result.6	1e-05	0.9993	1.998
result.7	1e-06	0.9743	2.001
result.8	1e-07	0.97	2.001
result.9	1e-08	0.9696	2.002
result.10	1e-09	0.9696	2.002

Table 4: Coefficients of Linear Regression Methods

	X1	X2
lm_h.coefficients	0.9696	2.002

In the first method, the parameter I made adjustment on is tolerance.

In the parallelize method, cluster 2 was used. As shown in the table *Coefficients of GD Method*, as tolerance becoming smaller, the coefficients becomes more accurate. When tolerance is less or equal than 1e-08, we get the same coefficients with that calculated by linear regression.But gradient descent method seems take more time than linear regression method.

Table 5: Running time for GD and Linear Methods

	user.self	sys.self	elapsed	user.child	sys.child
GD_time	0.05	0.03	1.77	NA	NA
Linear_Time	0.04	0	0.03	NA	NA

Table 6: Coefficients

	Beginning	GD theta1	GD theta2	LR theta1	LR theta2
result.1	1	0.9696	1.006	0.9696	1.002
result.2	2	1.966	2.005	1.992	2
result.3	3	3	3	3.066	2.993
result.4	4	4	4	4.012	3.997
result.5	5	5	5	4.908	5.017
result.6	6	6	6	5.998	6
result.7	7	7	7	7.025	6.992
result.8	8	8	8	8.048	7.993
result.9	9	9	9	8.992	9.005
result.10	10	10	10	9.996	10

In the second method, the parameter I made adjustment on is starting point of theta.

As shown in the table *Coefficients*, with different starting point, the theta calculate from GD method is almost same as that calculated from Linear Regression.

problem 4

a

Dopar function was used to bootstrop 10000 samples from Y and X. Each sample contains 10 parwise data and is chosen from the 200 pairwise dataset of X and Y. Linear regression model was built to test the relationship between x and y and to get the coefficients of the regression line.Cluster 2 was utilized.

b

Table 7: Overview of Beta

	x1	x2	x3	x4
result.1	-0.3303	2.137	2.918	-0.2862
result.2	1.313	1.017	3.129	-0.07225
result.3	0.8019	2.218	2.995	0.09269
result.4	0.6195	1.864	3.015	0.02759
result.5	1.239	2.44	2.977	-0.2977
result.6	2.257	2.358	2.645	0.197

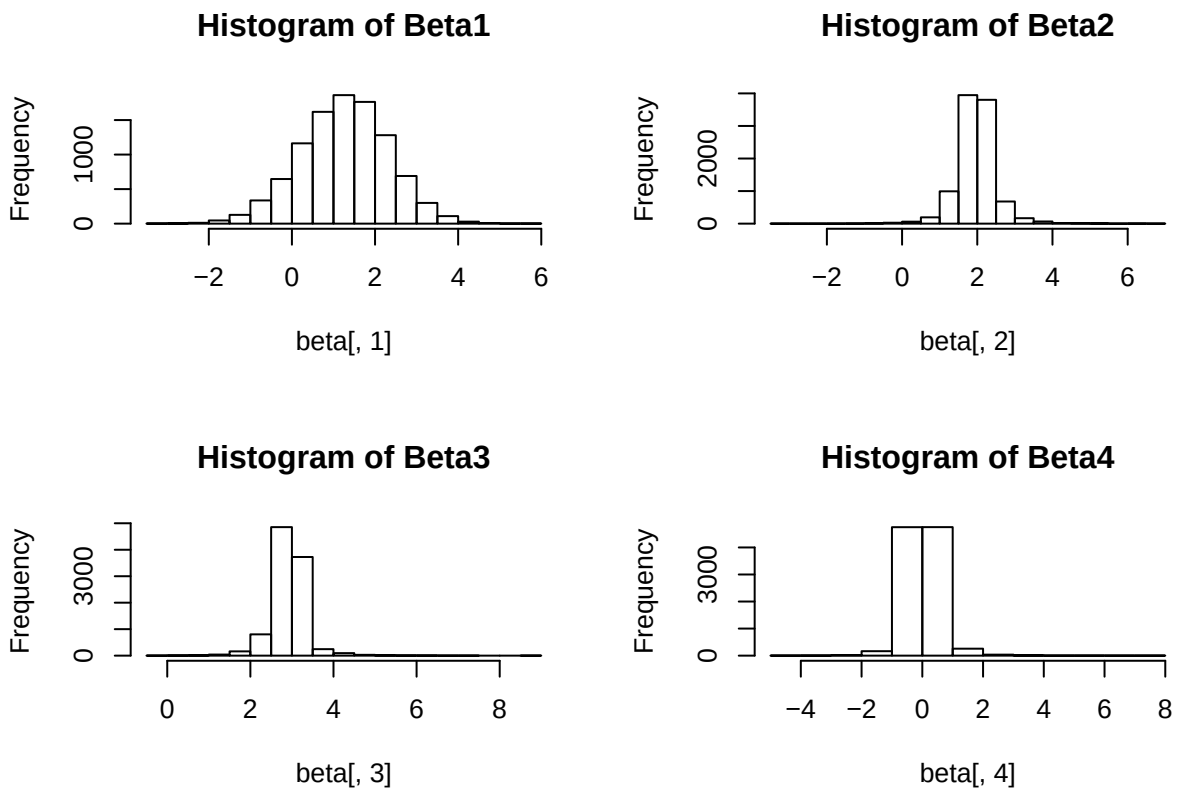
Table 8: Summary of Beta

x1	x2	x3	x4
Min. :-3.4758	Min. :-3.426	Min. :-0.2193	Min. :-4.195826
1st Qu.: 0.5619	1st Qu.: 1.736	1st Qu.: 2.7630	1st Qu.:-0.157351
Median : 1.2752	Median : 1.987	Median : 2.9696	Median : 0.000828
Mean : 1.2570	Mean : 1.960	Mean : 2.9129	Mean : 0.012373
3rd Qu.: 1.9760	3rd Qu.: 2.170	3rd Qu.: 3.0515	3rd Qu.: 0.117509
Max. : 5.9828	Max. : 6.667	Max. : 8.8908	Max. : 7.030195

As shown in the table *Overvie of Beta*, we get a matrix of beta with 10000 rows of results. Each row contains the coefficients(betas) value of the regression between x and y.

In the table *Summary of Beta*, we can see the statistics summary of these betas. Take X1 as example. The column X1 shows the summary for beta1. The minimum value of beta1 is -3.973 while the maximum value is 13.715. The median value is 1.25 and the mena value is 1.239.

c



As shown in the histogram, the values of Beta1 is maily less than 5. Beta2 is maily located between 0 and 5. Beta3 is mainly located between 2 and 4. Beta4 is mainly located between -1 and 1.

```
set.seed(12345)
y <- rnorm(n = 1e+06, mean = 1, sd = 1)
library(pander)

#################### Loop Method###############
t_lo <-
    system.time({
    mean_y <- mean(y)
    sum_difa <- 0
    for (i in 1:length(y)){
        sum_difa <- sum_difa + (y[i] - mean_y)^2
    }
})
Loop_Method <- c(t_lo[1:3], sum_difa)

#################### Vector Method###########
t_vec<-
    system.time({
  mean_y <- mean(y)
  diff_y <- y - mean_y
  sum_difb <- as.numeric(t(diff_y) %*% diff_y)
})
```

```
Vector_Method <- c(t_vec[1:3], sum_difb)

################ Dopar Method##############
library(foreach)
library(doParallel)
cl <- makeCluster(2)
registerDoParallel(cl)
t_dop <-
    system.time({
    mean_y <- mean(y)
    sum_difc <- foreach(1) %dopar% sum((y - mean_y)^2)
    })
stopCluster(cl)
Dopar_Method <- c(t_dop[1:3], sum_difc)

############### ParSapply Method###############
mean_y <- mean(y)
t_parsap <-
    system.time({
        c2 <- makeCluster(2)
        clusterExport(c2, "mean_y")
        sum_difd <- sum(parSapply(c2, y, function(y) (y-mean_y)^2))
    })
stopCluster(c2)
ParSapply_Method <- c(t_parsap[1:3], sum_difd)

#################### Summary####################
time_1 <- rbind(Loop_Method, Vector_Method, Dopar_Method, ParSapply_Method)
colnames(time_1)[4] <- c("Sum of Squares")
pander(time_1, caption = "Time for Different Methods")

################## Create data point################
set.seed(1256)
theta <- as.matrix(c(1,2),nrow=2)
X <- cbind(1,rep(1:10,10))
h <- X%*%theta+rnorm(100,0,0.2)
m <- length(h)
cl <- makeCluster(2)
registerDoParallel(cl)

############### Gradient descent################
GD_time <-
system.time({
dif1 <- 1
dif2 <- 1
alpha <- 0.001
ttheta <-
  foreach (i = 1:10, .combine = "cbind") %dopar% {
  tolerance <- 10^(1-i)
  while(dif1 > tolerance | dif2 > tolerance){
  grad_0 <- (1/m) * sum(((X%*%theta) - h))
  grad_1 <- (1/m) * t(X[, 2]) %*% ((X%*%theta) - h)
  theta_0 <- theta[1] - alpha * grad_0
```

```
  theta_1 <- theta[2] - alpha * grad_1
  dif1 <- abs(theta_0 - theta[1])
  dif2 <- abs(theta_1 - theta[2])
  theta[1] <- theta_0
  theta[2] <- theta_1
  }

  return(c(tolerance, theta[1], theta[2]))
  }
})
theta <- t(ttheta)
colnames(theta) <- c("Tolerance", "theta1", "theta2")
stopCluster(cl)

################## Linear regression##############
Linear_Time <-
system.time({
  lm_h <- lm(h~0+X)
Linear_coef <- t(data.frame(lm_h$coefficients))
})

################## Summary###################
pander(rbind(GD_time, Linear_Time), caption = "Running time for GD and Linear Methods")
pander(theta, caption = "Coefficients of GD Method")
pander(Linear_coef, caption = "Coefficients of Linear Regression Methods")
set.seed(1256)
X <- cbind(1,rep(1:10,10))
m <- length(h)
cl <- makeCluster(2)
registerDoParallel(cl)
tolerance <- 1e-9

############### Gradient descent################
GD_time <-
system.time({
dif1 <- 1
dif2 <- 1
alpha <- 0.001
ttheta <-
  foreach (i = 1:10, .combine = "cbind") %dopar% {
    theta <- as.matrix(c(i,i),nrow=2)
    h <- X%*%as.matrix(c(i,i),nrow=2)+rnorm(100,0,0.2)
  while(dif1 > tolerance | dif2 > tolerance){
  grad_0 <- (1/m) * sum(((X%*%theta) - h))
  grad_1 <- (1/m) * t(X[, 2]) %*% ((X%*%theta) - h)
  theta_0 <- theta[1] - alpha * grad_0
  theta_1 <- theta[2] - alpha * grad_1
  dif1 <- abs(theta_0 - theta[1])
  dif2 <- abs(theta_1 - theta[2])
  theta[1] <- theta_0
  theta[2] <- theta_1
  }
```

```
  return(c(i, theta[1], theta[2]))
  }
})
theta <- t(ttheta)
colnames(theta) <- c("Beginning", "GD theta1", "GD theta2")
stopCluster(cl)

################## Linear regression##############
Linear_coef <- matrix(NA, nrow = 10, ncol = 2)
Linear_Time <-
system.time({
linear <-
  for (i in 1:10) {
    theta_co <- as.matrix(c(i,i),nrow=2)
    h <- X%*%theta_co + rnorm(100,0,0.2)
  lm_h <- lm(h~0+X)
  Linear_coef[i, ] <- lm_h$coefficients
  }
})
colnames(Linear_coef) <- c("LR theta1", "LR theta2")

################## Summary####################
pander(rbind(GD_time, Linear_Time), caption = "Running time for GD and Linear Methods")
pander(cbind(theta, Linear_coef), caption = "Coefficients")
set.seed(1267)
n <- 200
X <- 1/cbind(1, rt(n, df = 1), rt(n, df = 1), rt(n, df = 1))
beta <- c(1, 2, 3, 0)
Y <- X %*% beta + rnorm(100, sd = 3)
data <- cbind(X, Y)
Y <- data[, 5]
X1 <- data[, 1]
X2 <- data[, 2]
X3 <- data[, 3]
X4 <- data[, 4]
cl <- makeCluster(2)
registerDoParallel(cl)
beta <- matrix(NA, nrow = 10000, ncol = 4)
colnames(beta) <- c("Intercept", "X2", "X3", "X4")
tbeta <-
  foreach (b = 1:10000, .combine = data.frame) %dopar% {
    i <- sample(1:200, size = 10, replace = TRUE)
    y <- Y[i]
    x1 <- X1[i]
    x2 <- X2[i]
    x3 <- X3[i]
    x4 <- X4[i]
    lm(y~0+x1+x2+x3+x4)$coef
  }
beta <- t(tbeta)
stopCluster(cl)
library(pander)
pander(head(beta), caption = "Overview of Beta")
```

```r
pander(summary(beta), caption = "Summary of Beta")
par(mfrow = c(2, 2))
hist(beta[, 1], main = "Histogram of Beta1")
hist(beta[, 2], main = "Histogram of Beta2")
hist(beta[, 3], main = "Histogram of Beta3")
hist(beta[, 4], main = "Histogram of Beta4")
```