STATS 102B HW 6

Joshua Susanto

2023-06-09

Problem 1:

Part a:

Using the following commands

R = matrix(c(rep(0.7, p/2), rep(0.4, p/2)), p, p) diag(R) = 1

to generate a correlation matrix for p = 20 variables

Modify the code in the file stochastic-gradient-descent-regression-bls.R in folder Lecture Notes/Week 10/R code to generate data for n = 1000000 observations.

You need to modify the code so that: • the mean.vector is of dimension p = 20 with all entries equal to 0 • the βtrue vector is of dimension p + 1 and all its entries are equal to 2 • and select a starting point of dimension p + 1

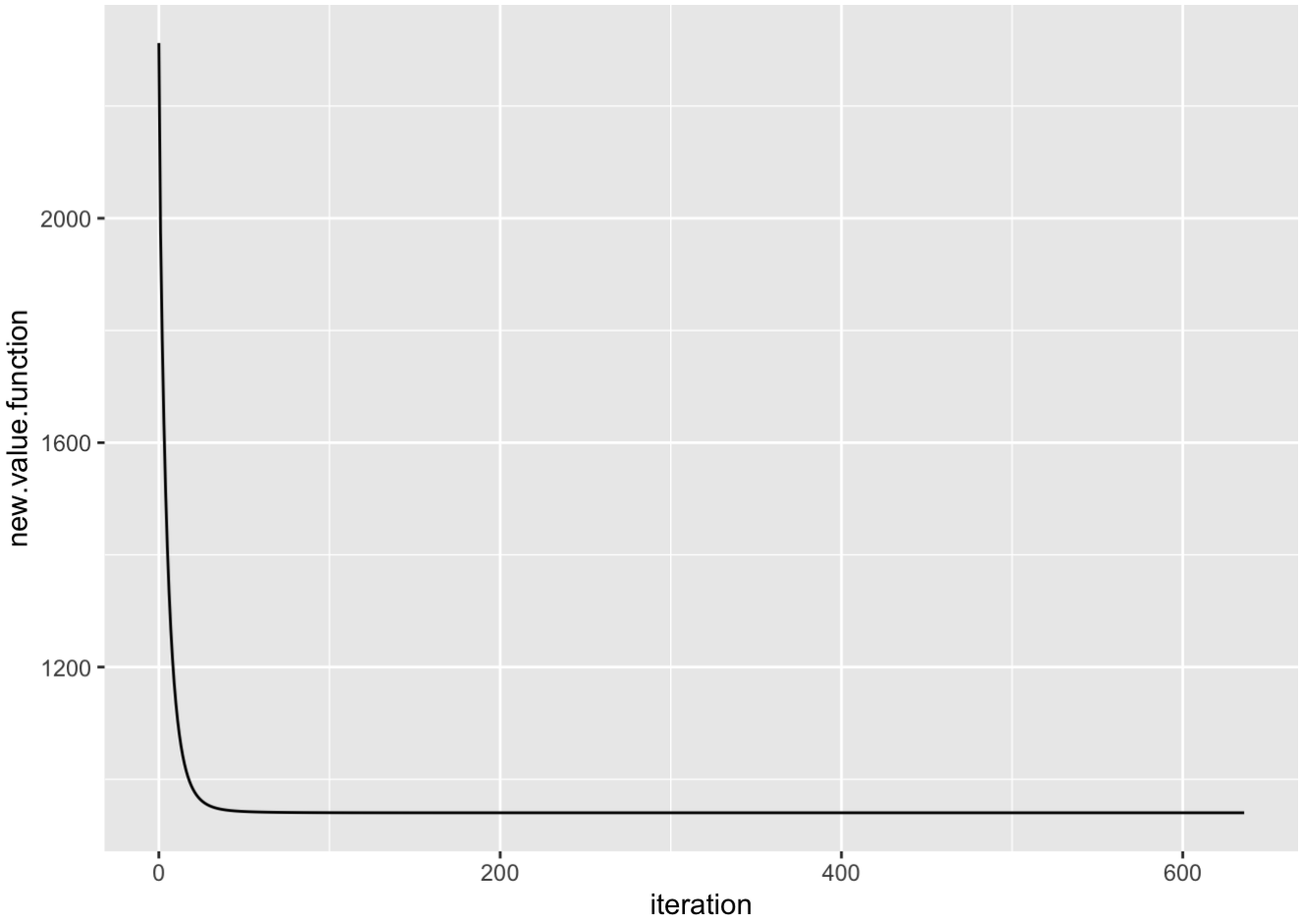
```
# stochastic gradient descent for regression
# with backtracking line search for selecting the step size
set.seed(405568250)
# simulate data
n = 1000000 # sample size
p = 20 # number of predictors
# create correlation matrix for regressors
R = matrix(c(rep(0.7, p/2), rep(0.4, p/2)), p, p)
diag(R) = 1
mean.vector = rep(0,20)
# generate design matrix X
design.orig = mvrnorm(n, mu = mean.vector, R)
intercept = rep(1, n)
design = cbind(intercept, design.orig)
# generate error term
error.term = rnorm(n,0,1)
# generate beta
beta_true = rep(2,21)
# generate response y
response = design%*%beta_true + error.term


# here we define the step size
mystepsize=5
# here we define the tolerance of the convergence criterion
mytol = 1e-15
# epsilon for backtracking line search
myepsilon = 0.5
# tau for backtracking line search
mytau = 0.5
# minibatch size
mymb = 0.001
# starting point
mystartpoint = rep(1,21)


SGD_BLS = function(y, X, startpoint, stepsize, conv_threshold,
                   epsilon, tau, mb, max_iter) {

  mini_batch=ceiling(length(y)*mb)

  z=cbind(y,X)

  # shuffle the data and select a mini batch

  shuffle.sample = z[sample(mini_batch,replace=FALSE),]

  ys=shuffle.sample[,1]
  Xs=shuffle.sample[,2:ncol(shuffle.sample)]

  old.point = startpoint
  gradient = (t(Xs)%*%Xs%*%old.point - t(Xs)%*%ys)
```

```
  # determine stepsize by backtracking line search

  while (t(ys - Xs%*%(old.point-stepsize*gradient))%*%(ys-Xs%*%(old.point-stepsize*gradi
ent)) >
               t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gr
adient) %*% gradient )
     {
    stepsize = tau * stepsize
  }

  new.point = old.point - stepsize * gradient


  old.value.function = t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point)

  converged = F
  iterations = 0

  while(converged == F) {

    # shuffle the data and select a mini batch
    shuffle.sample = z[sample(mini_batch,replace=FALSE),]

    ys=shuffle.sample[,1]
    Xs=shuffle.sample[,2:ncol(shuffle.sample)]

    ## Implement the stochastic gradient descent algorithm
    old.point = new.point

    gradient = t(Xs)%*%Xs%*%old.point - t(Xs)%*%ys

    # determine stepsize by backtracking line search

    while (t(ys - Xs%*%(old.point-stepsize*gradient))%*%(ys-Xs%*%(old.point-stepsize*gra
dient)) >
             t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradien
t) %*% gradient )
    {
      stepsize = tau * stepsize
    }

    new.point = old.point - stepsize * gradient

    new.value.function = t(ys - Xs%*%new.point)%*%(ys-Xs%*%new.point)


    if( abs(old.value.function - new.value.function) <= conv_threshold) {
      converged = T
    }

    data.output = data.frame(iteration = iterations,
```

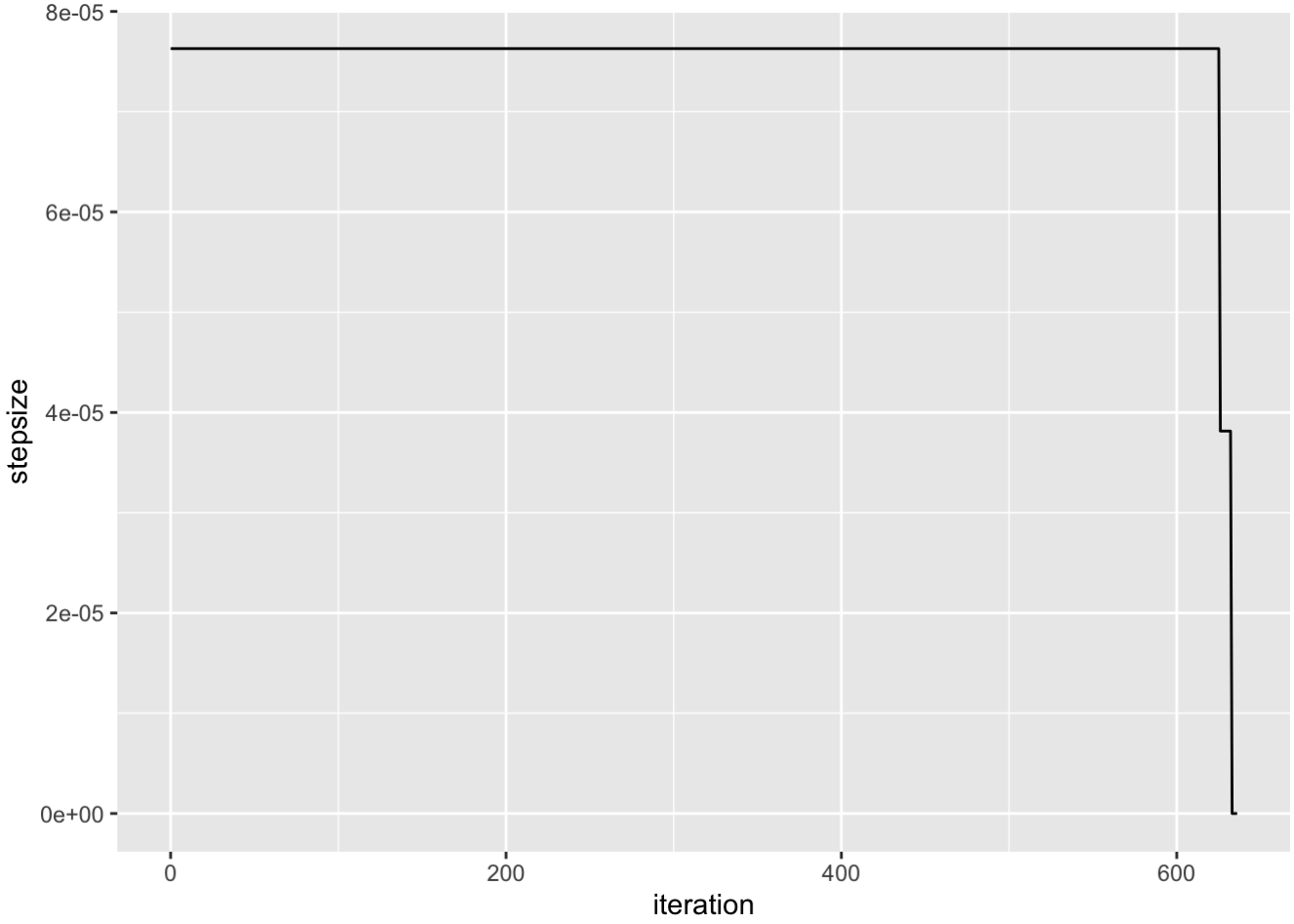
```
                        old.value.function = old.value.function,
                        new.value.function = new.value.function,
                        old.point=old.point, new.point=new.point,
                        stepsize = stepsize
                        )

    if(exists("iters")) {
      iters <- rbind(iters, data.output)
    } else {
      iters = data.output
    }

    iterations = iterations + 1
    old.value.function = new.value.function

    if(iterations >= max_iter) break
  }
  return(list(converged = converged,
              num_iterations = iterations,
              old.value.function = old.value.function,
              new.value.function = new.value.function,
              coefs = new.point,
              stepsize = stepsize,
              iters = iters))
}

start = Sys.time()
results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
                        myepsilon, mytau, mymb, 30000)


print(Sys.time()-start)
```

```
## Time difference of 4.697664 secs
```

```
par(mfrow=c(1,1))

ggplot(data = results$iters, mapping = aes(x = iteration, y = new.value.function))+
  geom_line()
```




```
par(mfrow=c(1,1))

ggplot(data = results$iters, mapping = aes(x = iteration, y = stepsize))+
  geom_line()
```




```
start = Sys.time()
summary(lm(response ~ design.orig))
```

```
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
```

```
print(Sys.time()-start)
```

```
## Time difference of 2.63357 secs
```

```
results$coefs
```

```
##                [,1]
## intercept 2.021014
##           1.981250
##           1.980711
##           2.115834
##           1.898868
##           2.025479
##           1.979566
##           1.936950
##           2.013485
##           2.011392
##           2.099715
##           2.057170
##           2.010731
##           1.951846
##           1.988558
##           2.005411
##           1.976072
##           1.981039
##           2.015348
##           1.987393
##           1.984227
```

Part b:

Use mini batch sizes Q = 100, 1000, 3000 and report the results of the stochastic gradient descent algorithm with backtracking line search and compare them with those from the least squares solution and β_{true}

```
SGD_BLS_b = function(y, X, startpoint, stepsize, conv_threshold,
                    epsilon, tau, mb, max_iter) {

  mini_batch=mb

  z=cbind(y,X)

  # shuffle the data and select a mini batch

  shuffle.sample = z[sample(mini_batch,replace=FALSE),]

  ys=shuffle.sample[,1]
  Xs=shuffle.sample[,2:ncol(shuffle.sample)]

  old.point = startpoint
  gradient = (t(Xs)%*%Xs%*%old.point - t(Xs)%*%ys)

  # determine stepsize by backtracking line search

  while (t(ys - Xs%*%(old.point-stepsize*gradient))%*%(ys-Xs%*%(old.point-stepsize*gradi
ent)) >
                t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gr
adient) %*% gradient )
     {
    stepsize = tau * stepsize
  }

  new.point = old.point - stepsize * gradient


  old.value.function = t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point)

  converged = F
  iterations = 0

  while(converged == F) {

    # shuffle the data and select a mini batch
    shuffle.sample = z[sample(mini_batch,replace=FALSE),]

    ys=shuffle.sample[,1]
    Xs=shuffle.sample[,2:ncol(shuffle.sample)]

    ## Implement the stochastic gradient descent algorithm
    old.point = new.point

    gradient = t(Xs)%*%Xs%*%old.point - t(Xs)%*%ys

    # determine stepsize by backtracking line search

    while (t(ys - Xs%*%(old.point-stepsize*gradient))%*%(ys-Xs%*%(old.point-stepsize*gra
dient)) >
```

```
           t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradien
t) %*% gradient )
    {
      stepsize = tau * stepsize
    }

    new.point = old.point - stepsize * gradient

    new.value.function = t(ys - Xs%*%new.point)%*%(ys-Xs%*%new.point)


    if( abs(old.value.function - new.value.function) <= conv_threshold) {
      converged = T
    }

    data.output = data.frame(iteration = iterations,
                        old.value.function = old.value.function,
                        new.value.function = new.value.function,
                        old.point=old.point, new.point=new.point,
                        stepsize = stepsize
                        )

    if(exists("iters")) {
      iters <- rbind(iters, data.output)
    } else {
      iters = data.output
    }

    iterations = iterations + 1
    old.value.function = new.value.function

    if(iterations >= max_iter) break
  }
  return(list(converged = converged,
              num_iterations = iterations,
              old.value.function = old.value.function,
              new.value.function = new.value.function,
              coefs = new.point,
              stepsize = stepsize,
              iters = iters))
}
```

Further, report the number of iterations required by stochastic gradient descent and the machine clock time. Compare the machine clock time for stochastic gradient descent to that required by the least squares solution.

```
mymb = c(100, 1000, 3000)

for (i in 1:length(mymb)) {
  start = Sys.time()
  results = SGD_BLS_b(response, design, mystartpoint, mystepsize, mytol,
                      myepsilon, mytau, mymb[i], 30000)


  print(paste('For Q =',mymb[i], 'our algorithm took', results$num_iterations, 'iteratio
ns'))
  print(paste('For Q =',mymb[i], 'our system took ', Sys.time()-start, 'seconds'))
  start = Sys.time()
  print(paste('Our least squares solution is'))
  print(summary(lm(response ~ design.orig)))
  print(paste('For Q =',mymb[i], 'our system took', Sys.time()-start,'seconds for the le
ast squares solution'))
  print(paste('For Q =', mymb[i], 'our results are'))
  print(results$coefs)
}
```

```
## [1] "For Q = 100 our algorithm took 788 iterations"
## [1] "For Q = 100 our system took  6.97614789009094 seconds"
## [1] "Our least squares solution is"
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
##
## [1] "For Q = 100 our system took 3.45250010490417 seconds for the least squares solut
ion"
## [1] "For Q = 100 our results are"
##               [,1]
## intercept 2.111695
##           1.686894
##           1.950650
##           2.066654
##           1.872210
##           1.888824
##           2.042422
```

```
##           2.149731
##           2.216707
##           1.760013
##           2.177844
##           2.120748
##           2.193188
##           1.810565
##           1.875865
##           2.098411
##           2.177380
##           1.902460
##           1.836141
##           2.008219
##           1.853728
## [1] "For Q = 1000 our algorithm took 708 iterations"
## [1] "For Q = 1000 our system took  5.29175996780396 seconds"
## [1] "Our least squares solution is"
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
```

```
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
##
## [1] "For Q = 1000 our system took 2.34571599960327 seconds for the least squares solu
tion"
## [1] "For Q = 1000 our results are"
##               [,1]
## intercept 2.021014
##           1.981250
##           1.980711
##           2.115834
##           1.898868
##           2.025480
##           1.979566
##           1.936950
##           2.013485
##           2.011392
##           2.099715
##           2.057170
##           2.010731
##           1.951846
##           1.988558
##           2.005411
##           1.976072
##           1.981039
##           2.015348
##           1.987393
##           1.984227
## [1] "For Q = 3000 our algorithm took 392 iterations"
## [1] "For Q = 3000 our system took  3.87149715423584 seconds"
## [1] "Our least squares solution is"
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
```

```
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
##
## [1] "For Q = 3000 our system took 2.36923003196716 seconds for the least squares solu
tion"
## [1] "For Q = 3000 our results are"
##               [,1]
## intercept 2.003840
##           1.963920
##           2.012934
##           1.982889
##           1.976446
##           2.004551
##           1.988819
##           2.005060
##           2.012948
##           2.036880
##           2.031522
##           2.034186
##           2.031982
##           2.000810
##           1.961918
##           1.991731
##           1.978451
##           1.970804
##           1.984030
##           2.000115
##           2.015914
```

We can see that an increase of mini batch sizes actually decreased our number of total iterations and total computation time. However, the computation time per iteration may be different and worth exploring. Additionally, a higher mini batch size led to more accurate results overall, with Q = 100 being relatively inaccurate and Q = 3000 drawing close. We see that finding our least squares solution consistently takes less time than performing our algorithm, being fairly consistent between our values of Q, having a consistent time difference of about 2.5 seconds. In terms of accuracy we see this gradually improve as our mini batch size is larger, with our algorithm being further away from β_{true} and least squares solution with Q = 100 and being the closest when Q = 3000.

Part c:

If you used backtrack line search only in the first iteration, do your results (in the settings of Part b) change?

```
SGD_BLS_c = function(y, X, startpoint, stepsize, conv_threshold,
                     epsilon, tau, mb, max_iter) {

  mini_batch=mb

  z=cbind(y,X)

  # shuffle the data and select a mini batch

  shuffle.sample = z[sample(mini_batch,replace=FALSE),]

  ys=shuffle.sample[,1]
  Xs=shuffle.sample[,2:ncol(shuffle.sample)]

  old.point = startpoint
  gradient = (t(Xs)%*%Xs%*%old.point - t(Xs)%*%ys)

  # determine stepsize by backtracking line search

  while (t(ys - Xs%*%(old.point-stepsize*gradient))%*%(ys-Xs%*%(old.point-stepsize*gradi
ent)) >
                 t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gr
adient) %*% gradient )
     {
    stepsize = tau * stepsize
  }

  new.point = old.point - stepsize * gradient


  old.value.function = t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point)

  converged = F
  iterations = 0

  while(converged == F) {

    # shuffle the data and select a mini batch
    shuffle.sample = z[sample(mini_batch,replace=FALSE),]

    ys=shuffle.sample[,1]
    Xs=shuffle.sample[,2:ncol(shuffle.sample)]

    ## Implement the stochastic gradient descent algorithm
    old.point = new.point

    gradient = t(Xs)%*%Xs%*%old.point - t(Xs)%*%ys

    new.point = old.point - stepsize * gradient

    new.value.function = t(ys - Xs%*%new.point)%*%(ys-Xs%*%new.point)
```

```
    if( abs(old.value.function - new.value.function) <= conv_threshold) {
      converged = T
    }

    data.output = data.frame(iteration = iterations,
                        old.value.function = old.value.function,
                        new.value.function = new.value.function,
                        old.point=old.point, new.point=new.point,
                        stepsize = stepsize
                        )

    if(exists("iters")) {
      iters <- rbind(iters, data.output)
    } else {
      iters = data.output
    }

    iterations = iterations + 1
    old.value.function = new.value.function

    if(iterations >= max_iter) break
  }
  return(list(converged = converged,
              num_iterations = iterations,
              old.value.function = old.value.function,
              new.value.function = new.value.function,
              coefs = new.point,
              stepsize = stepsize,
              iters = iters))
}
```

```
mymb = c(100, 1000, 3000)

for (i in 1:length(mymb)) {
  start = Sys.time()
  results = SGD_BLS_c(response, design, mystartpoint, mystepsize, mytol,
                        myepsilon, mytau, mymb[i], 30000)

  print(paste('For Q =',mymb[i], 'our algorithm took', results$num_iterations, 'iteratio
ns'))
  print(paste('For Q =',mymb[i], 'our system took ', Sys.time()-start, 'seconds'))
  start = Sys.time()
  print(paste('Our least squares solution is'))
  print(summary(lm(response ~ design.orig)))
  print(paste('For Q =',mymb[i], 'our system took', Sys.time()-start,'seconds for the le
ast squares solution'))
  print(paste('For Q =', mymb[i], 'our results are'))
  print(results$coefs)
}
```

```
## [1] "For Q = 100 our algorithm took 785 iterations"
## [1] "For Q = 100 our system took  4.36718702316284 seconds"
## [1] "Our least squares solution is"
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
##
## [1] "For Q = 100 our system took 2.55943202972412 seconds for the least squares solut
ion"
## [1] "For Q = 100 our results are"
##               [,1]
## intercept 2.111695
##           1.686894
##           1.950650
##           2.066654
##           1.872210
##           1.888824
##           2.042422
```

```
##           2.149731
##           2.216707
##           1.760013
##           2.177844
##           2.120748
##           2.193188
##           1.810565
##           1.875865
##           2.098411
##           2.177380
##           1.902460
##           1.836141
##           2.008219
##           1.853728
## [1] "For Q = 1000 our algorithm took 633 iterations"
## [1] "For Q = 1000 our system took  4.90429711341858 seconds"
## [1] "Our least squares solution is"
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
```

```
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
##
## [1] "For Q = 1000 our system took 2.82328486442566 seconds for the least squares solu
tion"
## [1] "For Q = 1000 our results are"
##               [,1]
## intercept 2.021014
##           1.981250
##           1.980711
##           2.115834
##           1.898868
##           2.025480
##           1.979566
##           1.936950
##           2.013485
##           2.011392
##           2.099715
##           2.057170
##           2.010731
##           1.951846
##           1.988558
##           2.005411
##           1.976072
##           1.981039
##           2.015348
##           1.987393
##           1.984227
## [1] "For Q = 3000 our algorithm took 445 iterations"
## [1] "For Q = 3000 our system took  4.33831787109375 seconds"
## [1] "Our least squares solution is"
##
## Call:
## lm(formula = response ~ design.orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7024 -0.6743  0.0001  0.6734  5.0203
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9981216  0.0009997    1999   <2e-16 ***
## design.orig1  1.9992342  0.0017403    1149   <2e-16 ***
## design.orig2  2.0025536  0.0017403    1151   <2e-16 ***
## design.orig3  1.9990254  0.0017384    1150   <2e-16 ***
## design.orig4  1.9978315  0.0017398    1148   <2e-16 ***
## design.orig5  1.9992968  0.0017401    1149   <2e-16 ***
## design.orig6  2.0001656  0.0017394    1150   <2e-16 ***
## design.orig7  1.9995259  0.0017391    1150   <2e-16 ***
## design.orig8  2.0015519  0.0017400    1150   <2e-16 ***
## design.orig9  2.0004022  0.0017403    1149   <2e-16 ***
## design.orig10 2.0011943  0.0017383    1151   <2e-16 ***
```

```
## design.orig11 1.9979712  0.0012394    1612   <2e-16 ***
## design.orig12 2.0009742  0.0012427    1610   <2e-16 ***
## design.orig13 1.9999766  0.0012414    1611   <2e-16 ***
## design.orig14 2.0012733  0.0012429    1610   <2e-16 ***
## design.orig15 1.9990334  0.0012406    1611   <2e-16 ***
## design.orig16 1.9978147  0.0012404    1611   <2e-16 ***
## design.orig17 1.9986943  0.0012408    1611   <2e-16 ***
## design.orig18 1.9995528  0.0012412    1611   <2e-16 ***
## design.orig19 1.9996696  0.0012421    1610   <2e-16 ***
## design.orig20 2.0017848  0.0012404    1614   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9997 on 999979 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.981e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
##
## [1] "For Q = 3000 our system took 2.32262802124023 seconds for the least squares solu
tion"
## [1] "For Q = 3000 our results are"
##               [,1]
## intercept 2.003840
##           1.963920
##           2.012934
##           1.982889
##           1.976446
##           2.004551
##           1.988818
##           2.005060
##           2.012949
##           2.036880
##           2.031522
##           2.034186
##           2.031982
##           2.000810
##           1.961918
##           1.991731
##           1.978451
##           1.970804
##           1.984030
##           2.000115
##           2.015914
```

For using backtracking line search for the first iteration we see a few changes. In general, the accuracy of our algorithm to our true solution and least squares solution remained fairly consistent between the two parts, with our algorithm being closer as the mini batch size grew, drawing fairly close when Q = 3000. The changes came in system time and iterations. As our mini batch size got larger, we saw that the changes we made led to a much higher total amount of iterations and an increase in system time when compared to the same results in part b. This means that the difference between the algorithm time and the least squares solution time grew as well.