Stats 102B HW 6

Tori Wang

2023-06-10

Question 1

Part a)

```
p=20;
R = matrix( c( rep(0.7,p/2) , rep( 0.4 , p/2) ) , p , p )
diag(R) = 1
```

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.

```
# stochastic gradient descent for regression
# with backtracking line search for selecting the step size
library(MASS)
set.seed(200)
# simulate data
n = 1000000 # sample size
#p = 4  # number of predictors
# create correlation matrix for regressors
p = 20
mean.vector = c(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 = c(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 = c(rep(0,21))


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*gradient)) >
               t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradient) %*% grad
     {
    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
```

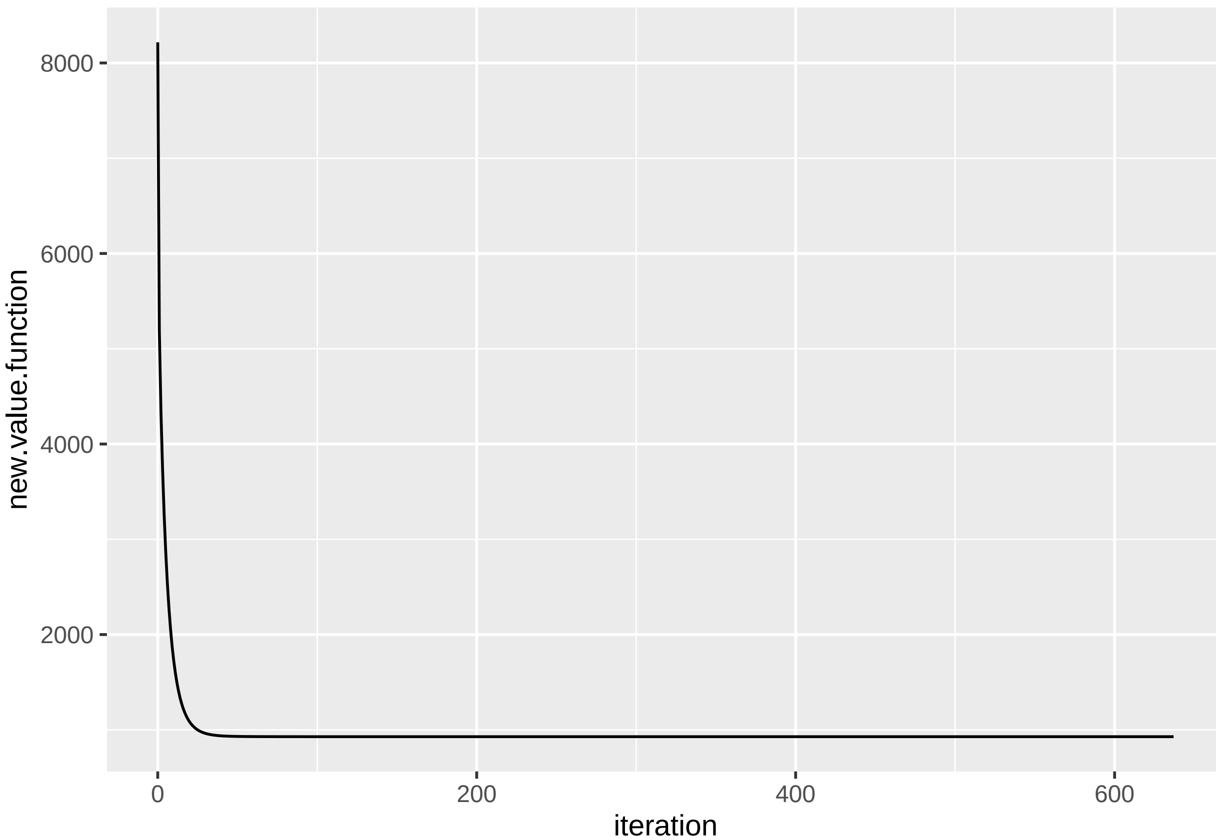
```
    while (t(ys - Xs%*%(old.point-stepsize*gradient))%*%(ys-Xs%*%(old.point-stepsize*gradient)) >
           t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradient) %*% gradient
    {
      stepsize = tau * stepsize
    }

    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 2.288752 secs
```

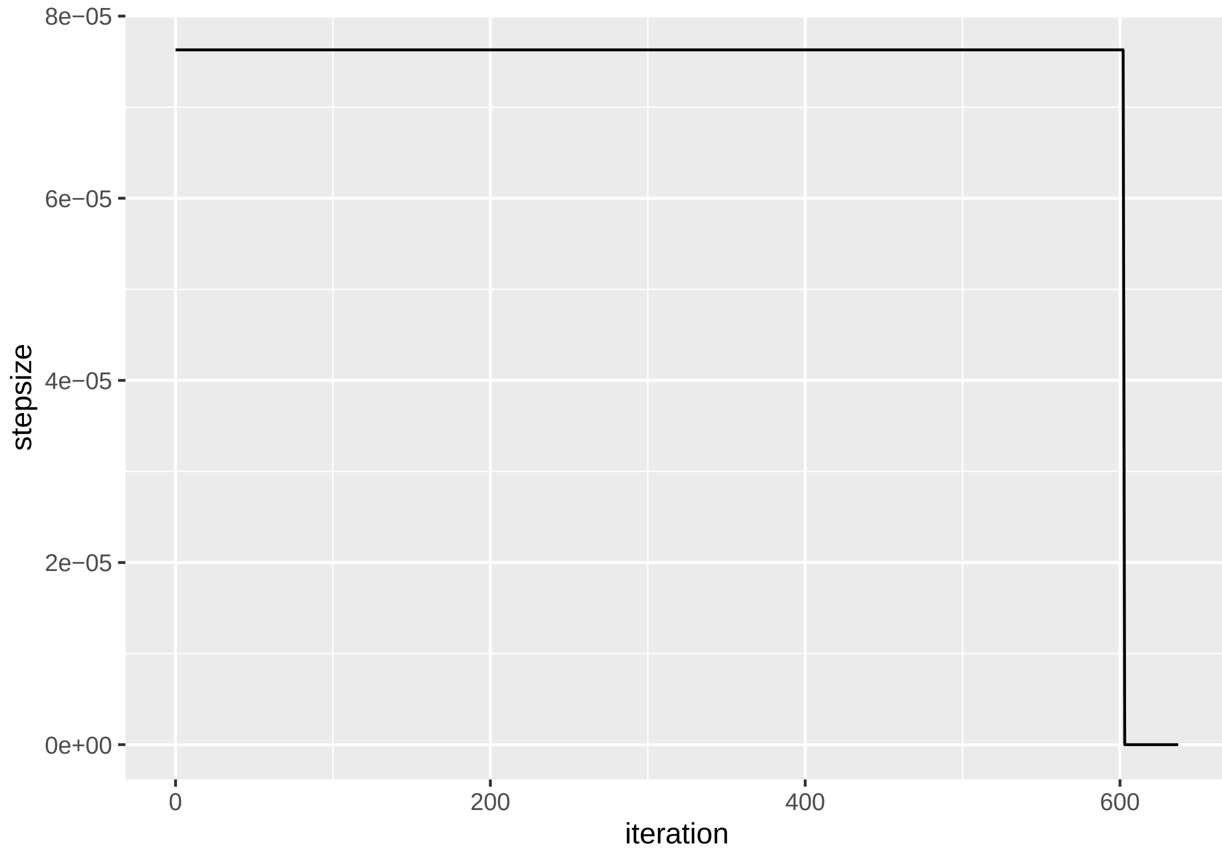
```
print(results$num_iterations)
```

```
## [1] 638
```

```
library(ggplot2)

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
## -5.3117 -0.6754  0.0002  0.6748  4.8110
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.001106   0.001000    2000   <2e-16 ***
## design.orig1   1.998384   0.001741    1148   <2e-16 ***
## design.orig2   2.002671   0.001740    1151   <2e-16 ***
## design.orig3   2.002004   0.001741    1150   <2e-16 ***
## design.orig4   2.001254   0.001739    1151   <2e-16 ***
## design.orig5   2.002853   0.001741    1150   <2e-16 ***
## design.orig6   1.995253   0.001741    1146   <2e-16 ***
## design.orig7   2.000170   0.001740    1149   <2e-16 ***
## design.orig8   1.996076   0.001741    1147   <2e-16 ***
## design.orig9   1.999886   0.001740    1149   <2e-16 ***
## design.orig10 2.001473   0.001739    1151   <2e-16 ***
## design.orig11 1.998633   0.001243    1607   <2e-16 ***
## design.orig12 1.998914   0.001241    1611   <2e-16 ***
```

```
## design.orig13 1.997663   0.001242    1609   <2e-16 ***
## design.orig14 1.999507   0.001242    1610   <2e-16 ***
## design.orig15 2.000793   0.001241    1612   <2e-16 ***
## design.orig16 2.000533   0.001242    1610   <2e-16 ***
## design.orig17 2.000533   0.001242    1611   <2e-16 ***
## design.orig18 2.001047   0.001242    1612   <2e-16 ***
## design.orig19 2.001977   0.001243    1611   <2e-16 ***
## design.orig20 1.999560   0.001241    1611   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1 on 999979 degrees of freedom
## Multiple R-squared:  0.9987, Adjusted R-squared:  0.9987
## F-statistic: 3.973e+07 on 20 and 999979 DF,  p-value: < 2.2e-16
```

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

```
## Time difference of 1.893773 secs
```

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 true Beta 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.

Discussion:

As seen below, we implement the mini batch stochastic gradient descent. The results for the original Stochastic Gradient descent is 2.294078 machine time seconds and 638 iterations. For the mini batch sizes, we have the following results: Q=100: 2.545432 seconds, 663 iterations. Q=1000: 2.192693 seconds and 636 iterations. Q= 3000: 5.973731 seconds and 1186 iterations.

We observe that the machine clock time for Q = 100 and Q=1000 is not very different from the original Stochastic gradient descent, however for the very large batch size of 3000, we observe a much higher machine clock time and number of iterations: 1186.

```
# stochastic gradient descent for regression
# with backtracking line search for selecting the step size
library(MASS)
set.seed(200)
# simulate data
n = 1000000 # sample size
#p = 4  # number of predictors
# create correlation matrix for regressors
p = 20
mean.vector = c(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 = c(rep(2, 21))
# generate response y
```

```r
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 = c(rep(0,21))


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

  mini_batch=ceiling(length(y)*mb)

  z=cbind(y,X)

  # shuffle the data and select a mini batch

  shuffle.sample = z[sample(mini_batch,size = minibatchsize,replace=TRUE),]

  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*gradient)) >
              t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradient) %*% grac
     {
    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),]

  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*gradient)) >
         t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradient) %*% 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))
```

```
}
```

Size Q = 100

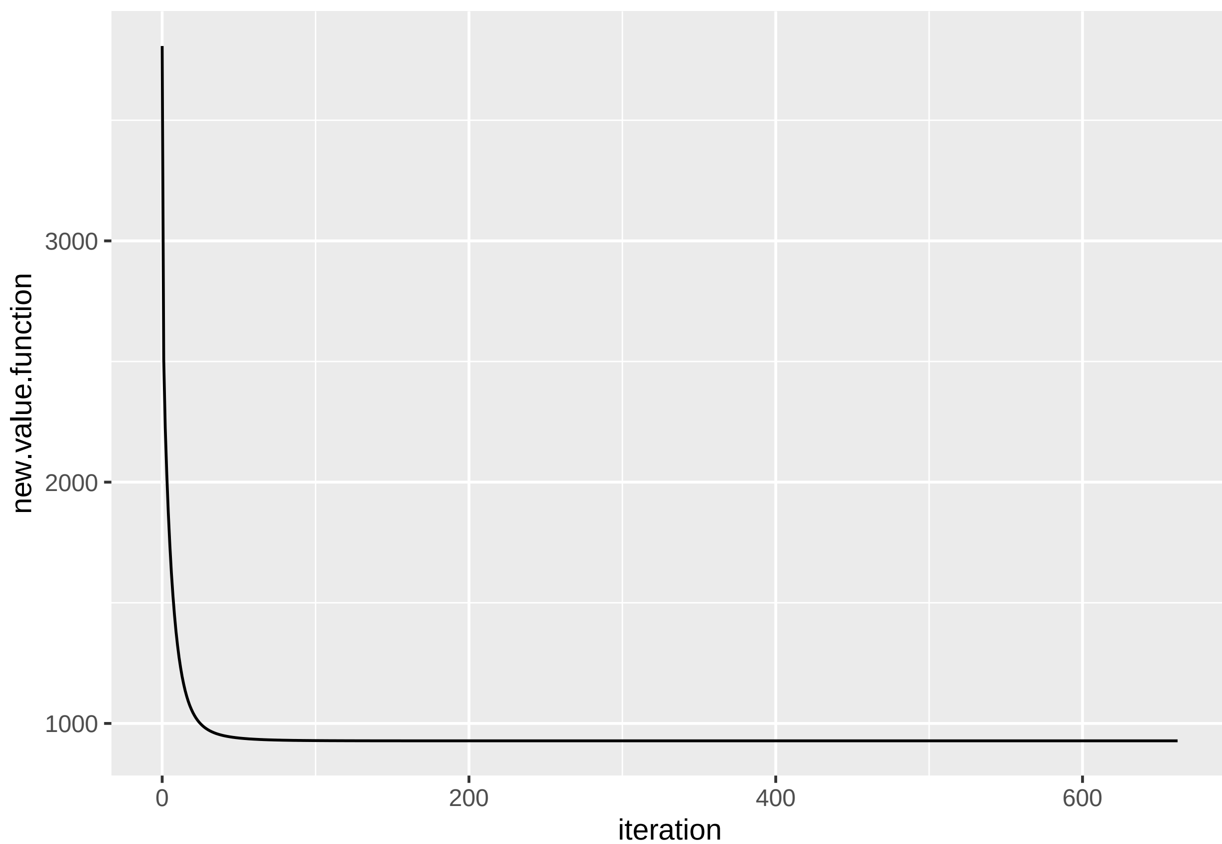
```
start = Sys.time()
results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
                       myepsilon, mytau, mymb, 30000,100)
print(Sys.time()-start)
```

```
## Time difference of 2.930541 secs
```

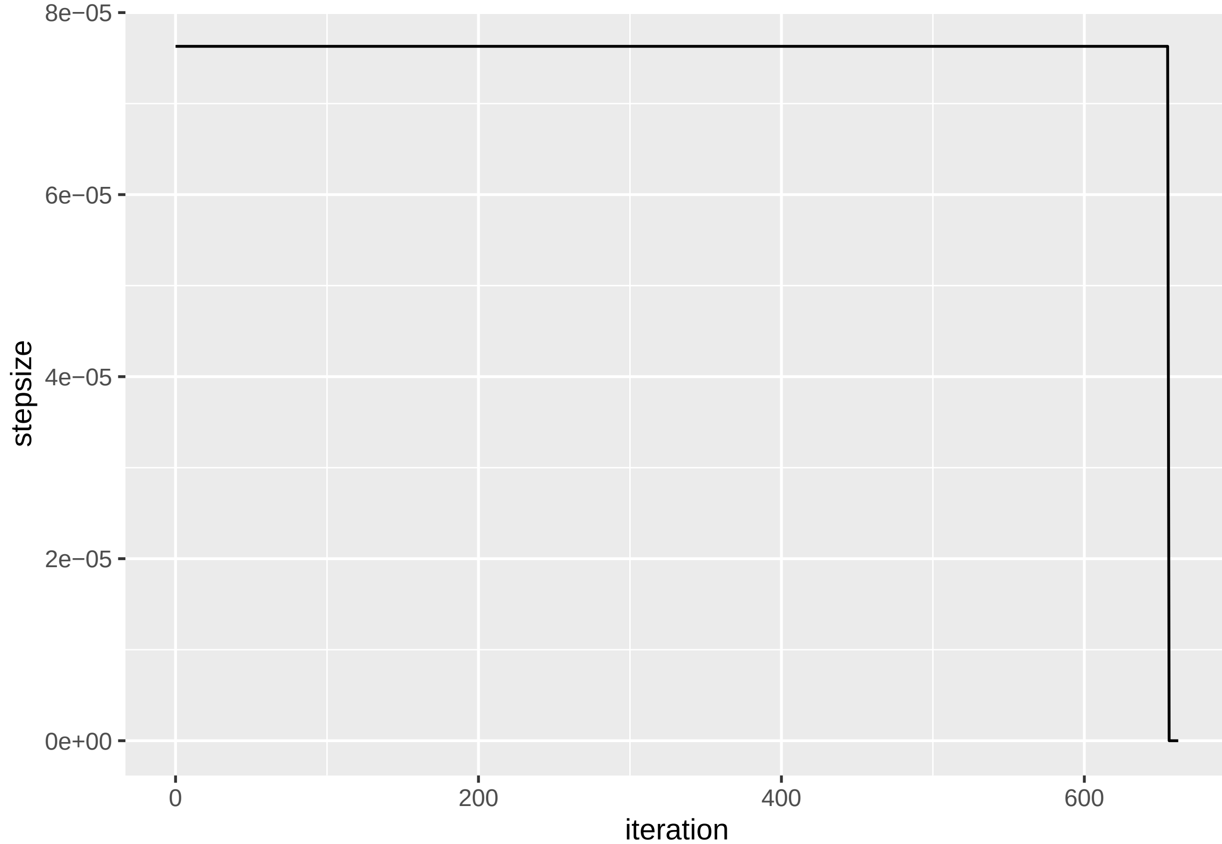
```
print(results$num_iterations)
```

```
## [1] 663
```

```
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()
```



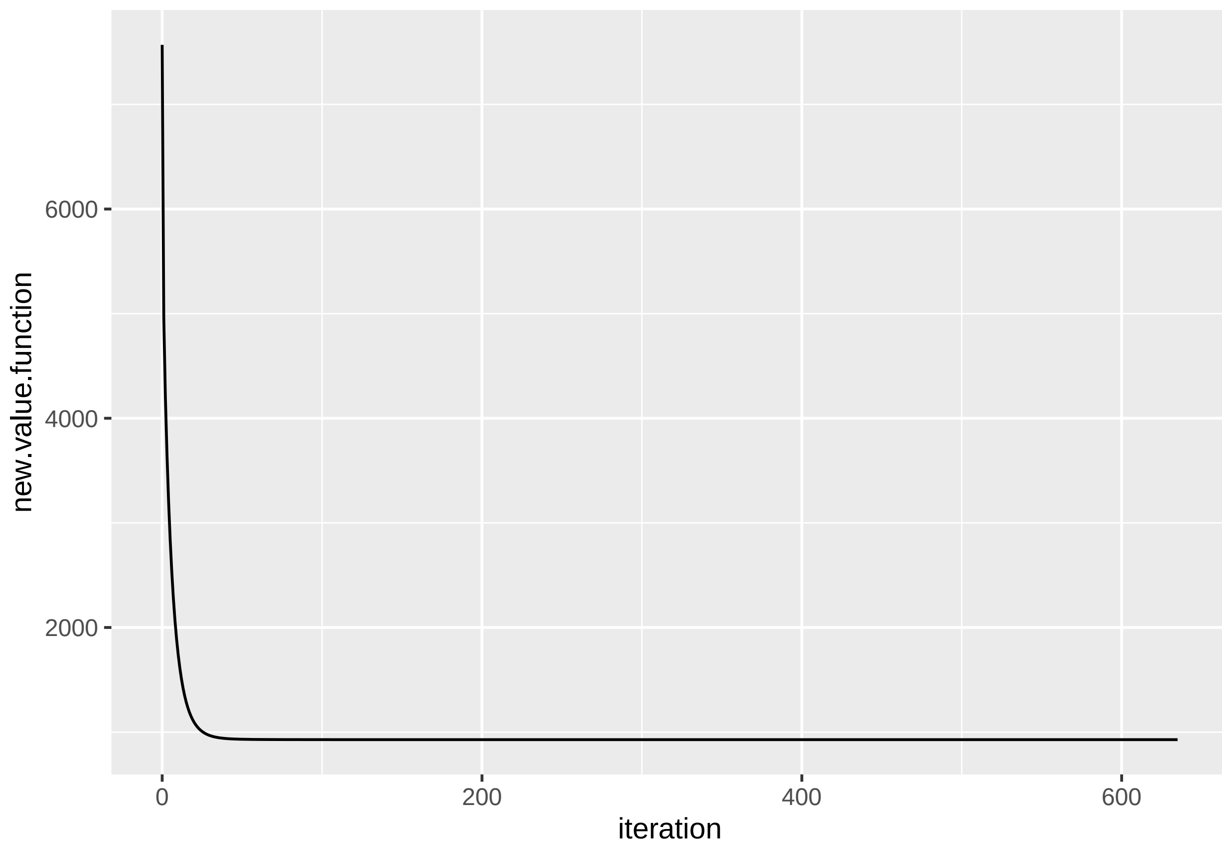

```r
start = Sys.time()

results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
                        myepsilon, mytau, mymb, 30000,1000)
print(Sys.time()-start)
```

```
## Time difference of 2.449257 secs
```

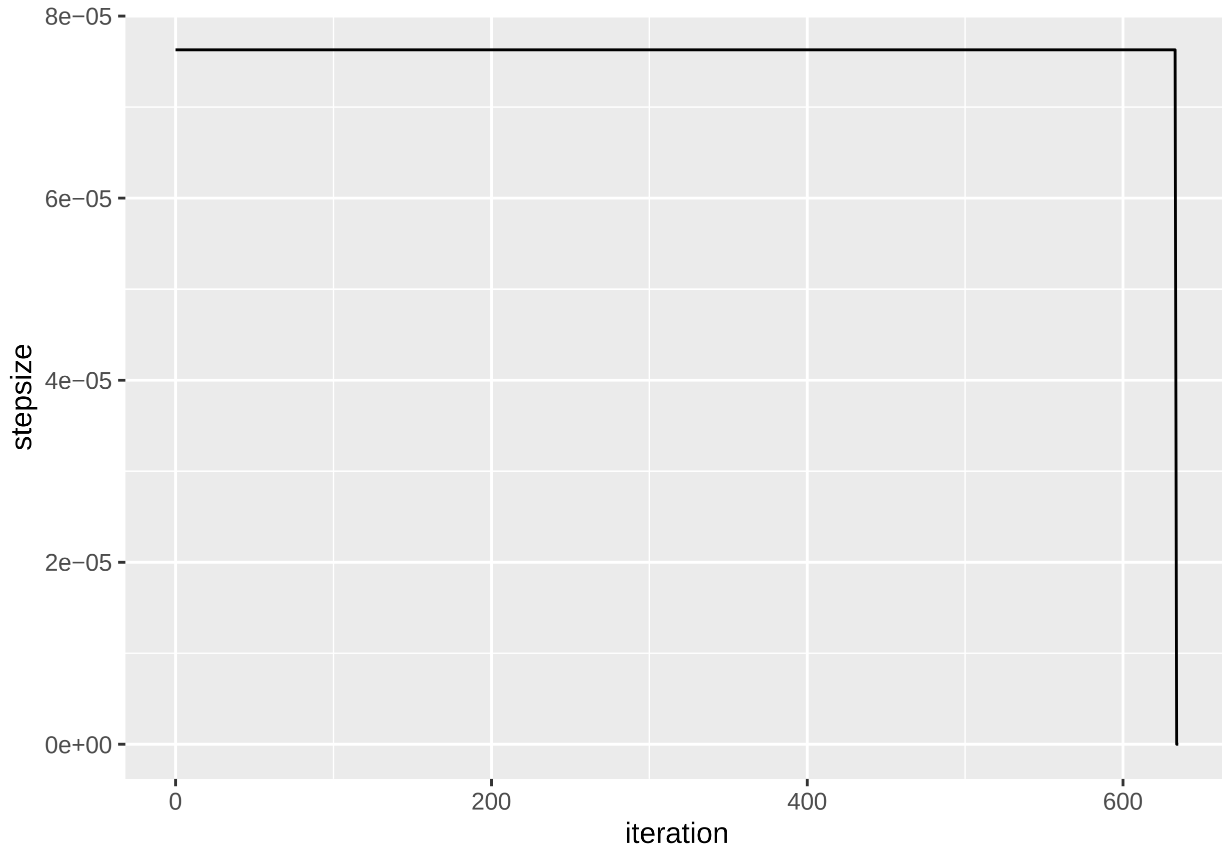
```r
print(results$num_iterations)
```

```
## [1] 636
```

```r
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()
```



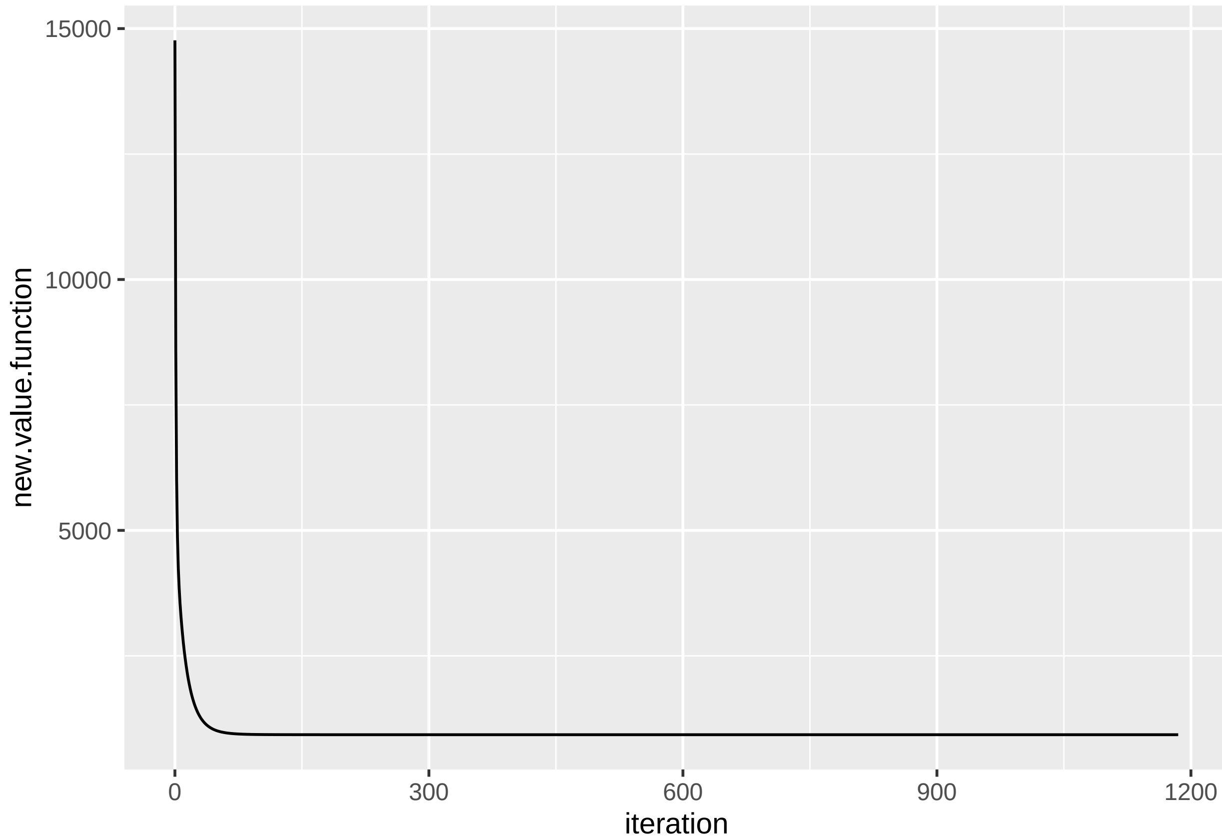

```
start = Sys.time()
results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
                       myepsilon, mytau, mymb, 30000,3000)
print(Sys.time()-start)
```

```
## Time difference of 6.469507 secs
```

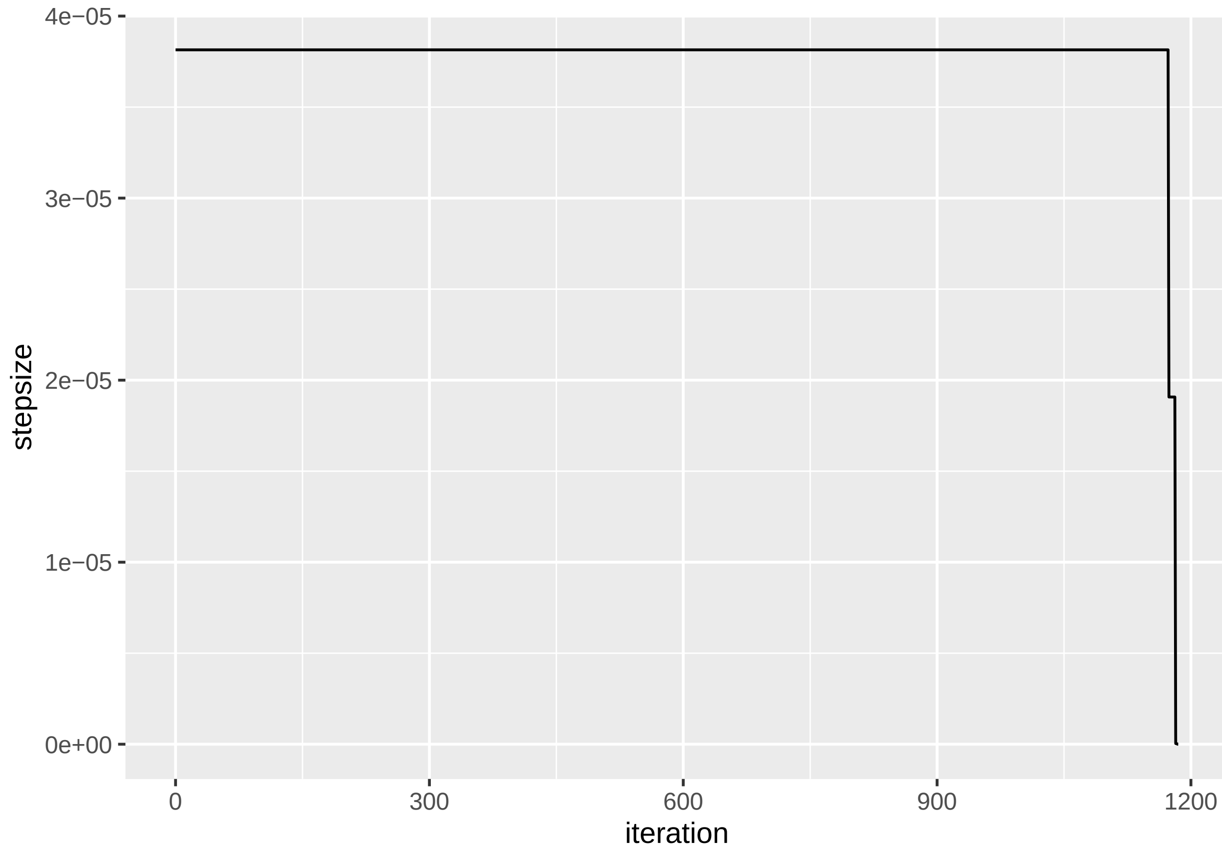
```
print(results$num_iterations)
```

```
## [1] 1186
```

```
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()
```



Part c)

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

Below, we change the settings in order to make the backtracking line search only occur in the first iteration. As seen below, I had to comment out the setting for Q=100 mini batch size because it would not converge and resulted in an error in my R program. For mini batch sizes Q=1000, the resulting machine clock time is 0.972682 and the number of iterations is 355. For Q = 3000 mini batch size, the resulting machine clock time is 6.490675 and the number of iterations is 1258.

```
# stochastic gradient descent for regression
# with backtracking line search for selecting the step size
library(MASS)
set.seed(200)
# simulate data
n = 1000000 # sample size
#p = 4  # number of predictors
# create correlation matrix for regressors
p = 20
mean.vector = c(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 = c(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 = c(rep(0,21))

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

  mini_batch=ceiling(length(y)*mb)

  z=cbind(y,X)

  # shuffle the data and select a mini batch

  shuffle.sample = z[sample(mini_batch,size = minibatchsize,replace=TRUE),]

  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*gradient)) >
              t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) - epsilon * stepsize * t(gradient) %*% grad
    {
    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),]

  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*gradient)) >
  #        t(ys - Xs%*%old.point)%*%(ys-Xs%*%old.point) -
  #        epsilon * stepsize * t(gradient) %*% 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))
}
```

Size Q = 100

```
# start = Sys.time()
# results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
#                         myepsilon, mytau, mymb, 30000,100)
# print(Sys.time()-start)
#
# print(results$num_iterations)
#
# par(mfrow=c(1,1))
#
# ggplot(data = results$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()
```

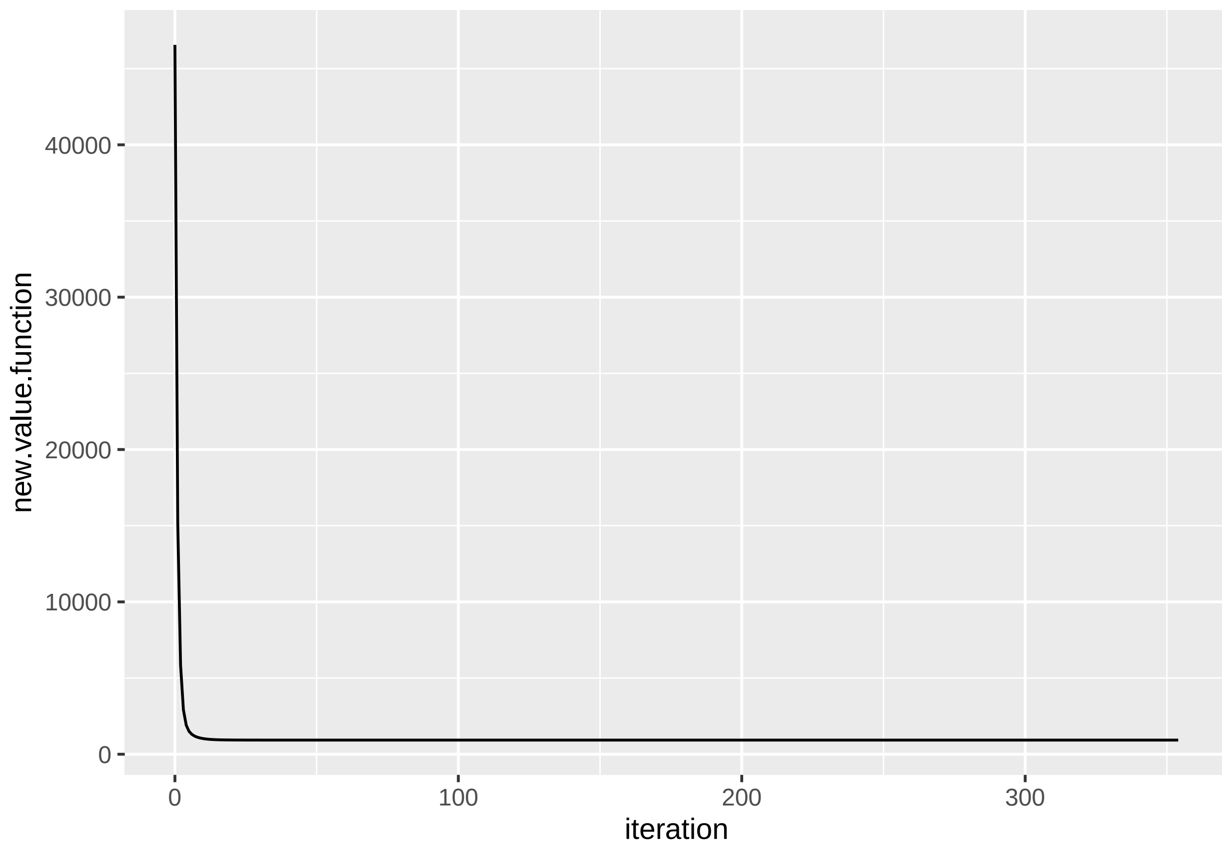
```
start = Sys.time()

results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
                       myepsilon, mytau, mymb, 30000,1000)
print(Sys.time()-start)
```

```
## Time difference of 0.971081 secs
```

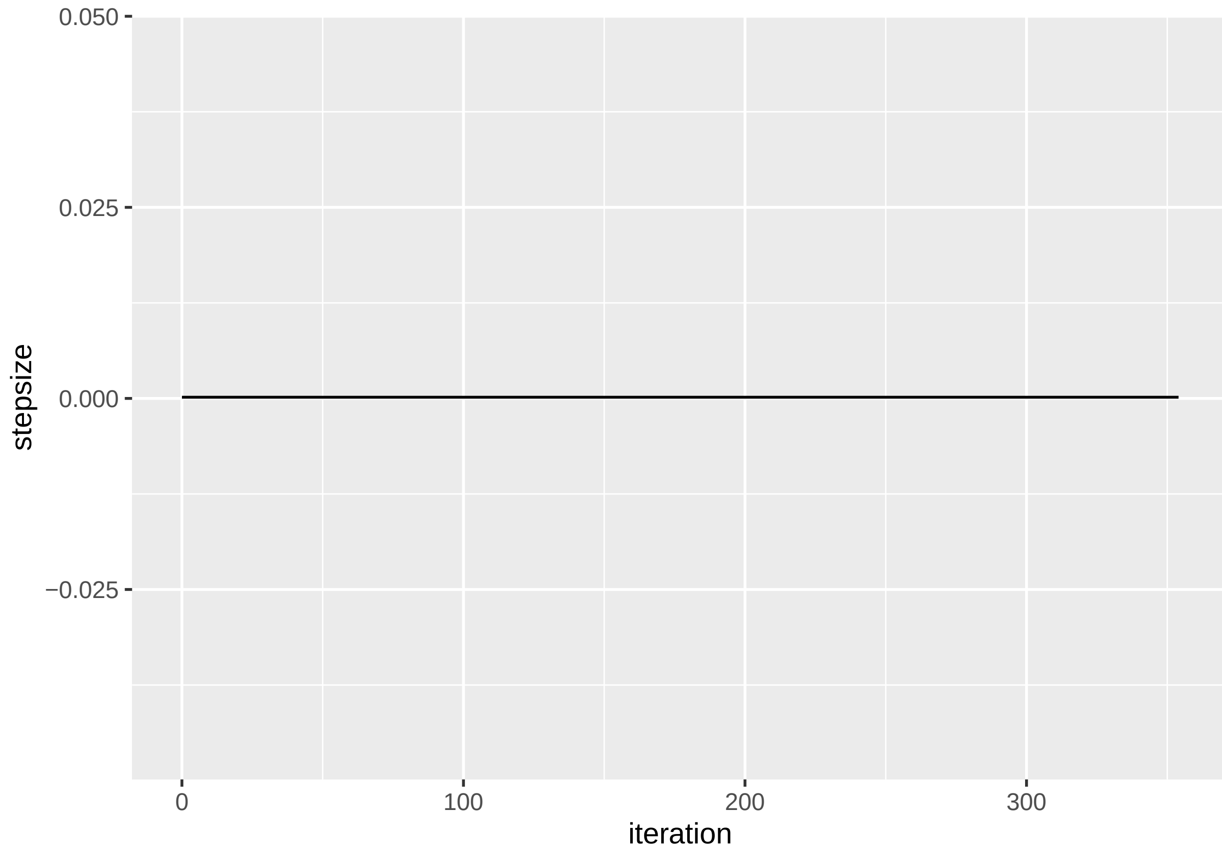
```
print(results$num_iterations)
```

```
## [1] 355
```

```
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()
```



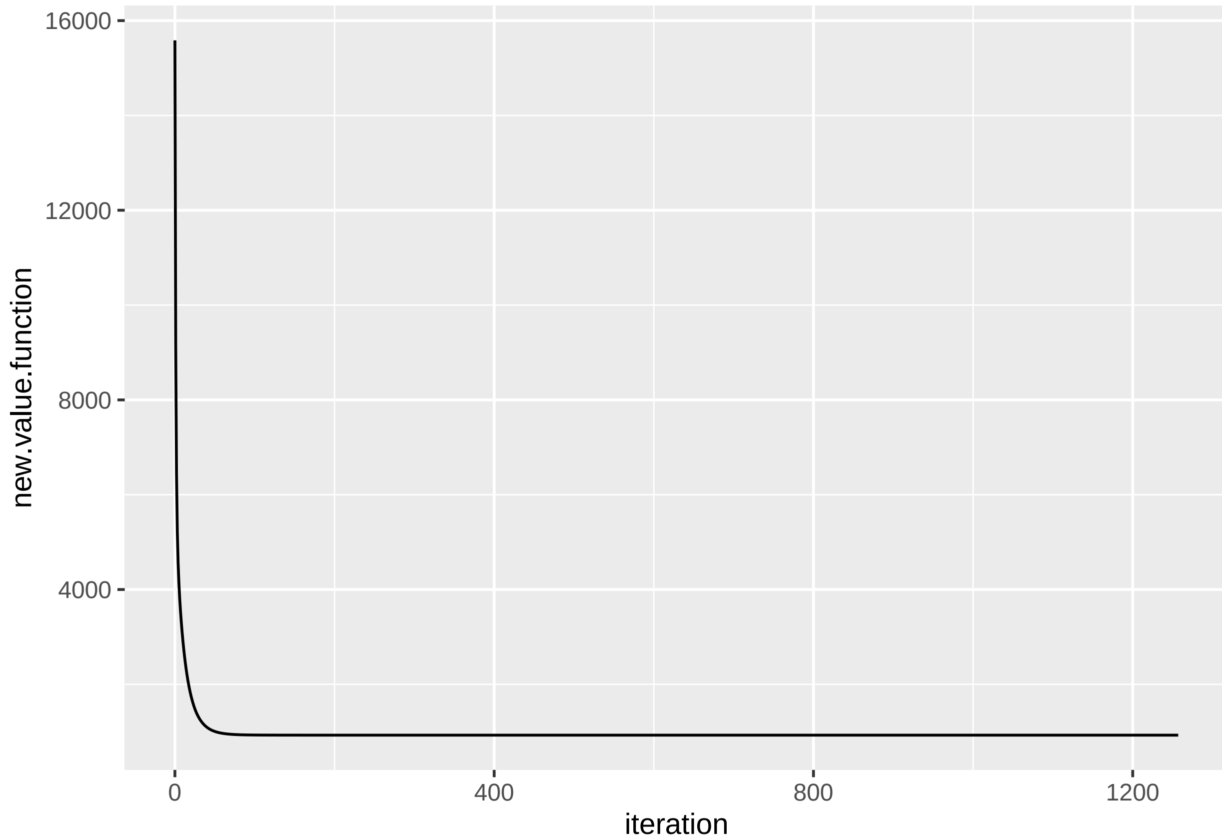

```
start = Sys.time()
results = SGD_BLS(response, design, mystartpoint, mystepsize, mytol,
                       myepsilon, mytau, mymb, 30000,3000)
print(Sys.time()-start)
```

```
## Time difference of 6.828795 secs
```

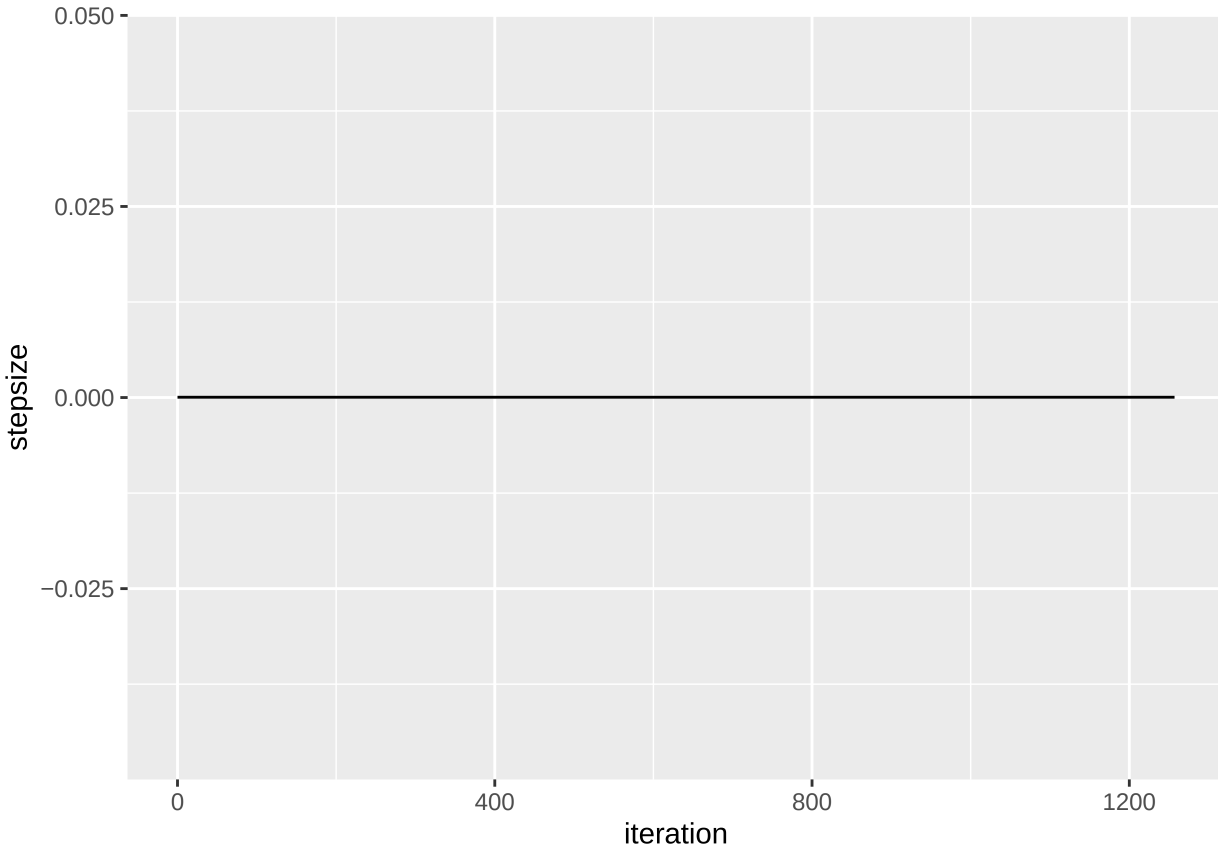
```
print(results$num_iterations)
```

```
## [1] 1258
```

```
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()
```


0.050
0.025
0.000
−0.025
stepsize
0
400
800
1200
iteration