HW 1

Bryan Mui - UID 506021334 - 14 April 2025

Loaded packages: ggplot2, tidyverse (include = false for this chunk)

Problem 1

Part a

Find the theoretical min for the function:

$$f(x) = x^4 + 2x^2 + 1$$

Solution: find f'(x) and f' '(x), set f'(x) to 0 and solve, and f' '(x) needs to be > 0 to be a min

Step 1: find f'(x) and f' '(x)

$$f(x) = x^4 + 2x^2 + 1 \tag{1}$$
$$f'(x) = 4x^3 + 4x \tag{2}$$
$$f''(x) = 12x^2 + 4 \tag{3}$$
$$\tag{4}$$

Step 2: set f'(x) to 0 and solve

$$f'(x) = 4x^3 + 4x \tag{5}$$
$$0 = 4x^3 + 4x \tag{6}$$
$$0 = 4x(x^2 + 4) \tag{7}$$

We get

$$x = 0$$

and

$$0 = x^2 + 4$$

which has no real solution

Step 3: check that f' '(x) needs to be > 0 to be a min

Our critical point is x = 0,

$$\begin{aligned} f''(0) &= 12(0)^2 + 4 && (8) \\ &= 4 && (9) \end{aligned}$$

Since f'(x) = 0 at 0 and f' '(x) > 0 at that point, **we have a min at x = 0, and plugging into f(0) we get the minimum point**

(0, 1)

Part b

0)

Use the gradient descent algorithm with **constant step size** and with **back-tracking line search** to calculate x_{min}

Constant step size descent is implemented as follows:

1. Select a random starting point x_0
2. While stopping criteria < tolerance, do:

- Select η_k(as a constant)
- Calculate $x_{(k+1)} = x_k - \eta_k * \nabla(f(x_k))$
- Calculate the value of stopping criterion

Stopping criteria: Stop if $| |\nabla(f(x_k)||_2 \leq \epsilon$

```r
# Gradient descent algorithm that uses constant step to minimize an objective
 ↪  function

gradient_descent_constant_step <- function(tol = 1e-6, max_iter = 10000,
 ↪  step_size = 0.01) {
  # Step 1: Initialize and select a random stopping point
  # Initialize
  set.seed(777) # example seeding
  last_iter <- 0 # the last iteration ran
  eta <- step_size # step size that is decided manually
  max_iter <- max_iter # max iterations before terminating if mininum isn't
 ↪  found
  tolerance <- tol # tolerance for the stoppign criteria
  obj_values <- numeric(max_iter) # Stores the value of f(x)
  eta_values <- numeric(max_iter)  # To store eta values used each iteration
  eta_values[1] <- step_size
  betas <- numeric(max_iter) # Stores the value of x guesses
  x0 <- runif(1, min=-10, max=10) # our first guess is somewhere between
 ↪  -10-10

  # Set the objective function to the function to be minimized
  # Objective function: f(x)
  obj_function <- function(x) {
    return(x^4 + 2*(x^2) + 1)
  }

  # Gradient function: d/dx of f(x)
  gradient <- function(x) {
    return(4*x^3 + 4*x)
  }

  # Append the first guess to the obj_values and betas vector
  betas[1] <- x0
  obj_values[1] <- obj_function(x0)

  # Step 2: While stopping criteria < tolerance, do:
  for (iter in 1:max_iter) { # the iteration goes n = 1, 2, 3, 4, but the
   ↪  arrays of our output starts at iter = 0 and guess x0
    # Select eta(step size), which is constant
    # There's nothing to do for this step

    # Calculate the next guess of x_k+1, calculate f(x_k+1), set eta(x_k+1)
```

```r
    betas[iter + 1] <- betas[iter] - (eta * gradient(betas[iter]))
    obj_values[iter + 1] <- obj_function(betas[iter + 1])
    eta_values[iter + 1] <- eta

    # Calculate the value of the stopping criterion
    stop_criteria <- abs(gradient(betas[iter + 1]))

    # If stopping criteria less than tolerance, break
    if(is.na(stop_criteria) || stop_criteria <= tolerance) {
      last_iter <- iter + 1
      break
    }

    # if we never set last iter, then we hit the max number of iterations and
     ↪  need to set
    if(last_iter == 0) { last_iter <- max_iter }

    # end algorithm
  }

  return(list(betas = betas, obj_values = obj_values, eta_values =
   ↪  eta_values, last_iter = last_iter)) # in this case, beta(predictors)
   ↪  are the x values, obj_values are f(x), eta is the step size, last iter
   ↪  is the value in the vector of the final iteration before stopping
}
```

Running the gradient descent algorithm with fixed step size:

```r
minimize_constant_step <- gradient_descent_constant_step(tol = 1e-6, max_iter
 ↪  = 10000, step_size = 0.03)
print(minimize_constant_step)
```

```
$betas
 [1]  3.757148134 -3.058091092  0.740763042  0.603094019  0.504399680
 [6]  0.428472253  0.367616075  0.317540520  0.275593469  0.240010435
[11]  0.209550087  0.183299884  0.160564858  0.140800331  0.123569332
[16]  0.108514593  0.095339505  0.083794772  0.073668795  0.064780563
[21]  0.056974273  0.050115167  0.044086243  0.038785612  0.034124337
[26]  0.030024648  0.026418442  0.023246017  0.020454987  0.017999362
[31]  0.015838739  0.013937613  0.012264775  0.010792780  0.009497496
[36]  0.008357693  0.007354700  0.006472088  0.005695405  0.005011934
```

```
[41]  0.004410487  0.003881218  0.003415465  0.003005605  0.002644929
[46]  0.002327535  0.002048229  0.001802441  0.001586147  0.001395809
 [ reached 'max' / getOption("max.print") -- omitted 9950 entries ]

$obj_values
 [1] 228.498357 107.162271   2.398564   1.859739   1.573567   1.400882
 [7]   1.288546   1.211831   1.157672   1.118528   1.089751   1.068327
[13]   1.052227   1.040042   1.030772   1.023689   1.018262   1.014092
[19]   1.010884   1.008411   1.006503   1.005029   1.003891   1.003011
[25]   1.002330   1.001804   1.001396   1.001081   1.000837   1.000648
[31]   1.000502   1.000389   1.000301   1.000233   1.000180   1.000140
[37]   1.000108   1.000084   1.000065   1.000050   1.000039   1.000030
[43]   1.000023   1.000018   1.000014   1.000011   1.000008   1.000006
[49]   1.000005   1.000004
 [ reached 'max' / getOption("max.print") -- omitted 9950 entries ]

$eta_values
 [1] 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03
[16] 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03
[31] 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03
[46] 0.03 0.03 0.03 0.03 0.03
 [ reached 'max' / getOption("max.print") -- omitted 9950 entries ]

$last_iter
[1] 118
```

```r
cat("The functions stopped after", minimize_constant_step$last_iter - 1,
 ↪  "iterations \n")
```

```
The functions stopped after 117 iterations
```

```r
cat("The function's point of minimization is", "(",
 ↪  minimize_constant_step$betas[minimize_constant_step$last_iter], "," ,
 ↪  minimize_constant_step$obj_values[minimize_constant_step$last_iter], ")
 ↪  \n")
```

```
The function's point of minimization is ( 2.342325e-07 , 1 )
```

Backtracking Line Search is implemented as follows:

1. Select a random starting point x_0

2. While stopping criteria < tolerance, do:

- Select η_k using backtracking line search
- Calculate $x_{(k+1)} = x_k - \eta_k * \nabla(f(x_k))$

- Calculate the value of stopping criterion

Backtracking Line Search:

- Set $\eta^0 > 0$(usually a large value), $\epsilon \in (0,1)$ and $\tau \in (0,1)$
- Set $\eta_1 = \eta^0$
- At iteration k, set $\eta_k < -\eta_{k-1}$

  1. Check whether the Armijo Condition holds:

  $$h(\eta_k) \leq h(0) + \epsilon \eta_k h'(0)$$

     where $h(\eta_k) = f(x_k) - \eta_k \nabla f(x_k)$
  2.
     - If yes(condition holds), terminate and keep η_k
     - If no, set $\eta_k = \tau \eta_k$ and go to Step 1

Stopping criteria: Stop if $||\nabla(f(x_k))||_2 \leq \epsilon$

Other note: Since we need h'(0) for the Armijo condition calculation, that is given by:

$$h'(0) = -[\nabla f(x_k)]^\top \nabla f(x_k)$$

Since we are minimizing x, we have a one dimensional beta, we can simplify to

$$h'(0) = -||\nabla f(x_k)||^2$$

To summarize, backtracking line search chooses the step size by ensuring the Armijo condition always holds. If the Armijo condition doesn't hold, we are probably overshooting, hence the step size gets updated iteratively

```
gradient_descent_backtracking <- function(tol = 1e-6, max_iter = 10000,
↪   epsilon = 0.5, tau = 0.5, init_step_size = 1) {
  # Step 1: Initialize and select a random stopping point
  # Initialize
  set.seed(777) # example seeding
  last_iter <- 0 # the last iteration ran
```

```r
max_iter <- max_iter # max iterations before terminating if minimum isn't
↪  found
tolerance <- tol # tolerance for the stopping criteria
epsilon <- epsilon # Epsilon used in the step size criteria calculation
tau <- tau # tau used in the step size criteria calculation
obj_values <- numeric(max_iter) # Stores the value of f(x)
eta_values <- numeric(max_iter)  # To store eta values used each iteration
eta_values[1] <- init_step_size
betas <- numeric(max_iter) # Stores the value of x guesses
x0 <- runif(1, min=-10, max=10) # our first guess is somewhere between -10
↪  to 10
eta <- init_step_size # our initial step size

# Set the objective function to the function to be minimized
# Objective function: f(x)
obj_function <- function(x) {
  return(x^4 + 2*(x^2) + 1)
}

# Gradient function: d/dx of f(x)
gradient <- function(x) {
  return(4*x^3 + 4*x)
}

# Armijo condition function
# returns TRUE or FALSE whether the condition is satisfied or not
armijo_stepsize <- function(beta, eta, grad, f, epsilon, tau, max_iter) {
  subiter <- 1 # set a hard limit of iterations
  # calc armijo
  beta_new <- beta - (eta)*grad(beta)
  armijo <- f(beta_new) > (f(beta) - epsilon*eta*sum(grad(beta)^2))
  while (armijo && (iter <= max_iter)) {
    #update eta
    eta <- tau * eta

    #recalculate armijo
    beta_new <- beta - (eta)*grad(beta)
    armijo <- f(beta_new) > (f(beta) - epsilon*eta*sum(grad(beta)^2))

    subiter <- subiter + 1
  }
  return(eta)
}
```

```
# Append the first guess to the obj_values and betas vector
betas[1] <- x0
obj_values[1] <- obj_function(x0)

# Step 2: While stopping criteria < tolerance, do:
for (iter in 1:max_iter) { # the iteration goes n = 1, 2, 3, 4, but the
 ↪  arrays of our output starts at iter = 0 and guess x0
  beta <- betas[iter]

  # use BLS to calculate eta
  eta <- armijo_stepsize(beta = beta, eta = eta, grad = gradient, f =
↪  obj_function, epsilon = epsilon, tau = tau, max_iter = max_iter)
  eta_values[iter + 1] <- eta

  # Calculate the next guess of x_k+1
  beta_new <- beta - (eta * gradient(beta))
  betas[iter + 1] <- beta_new

  # calculate f(x_k+1), to keep track obj values
  obj_values[iter + 1] <- obj_function(beta_new)

  # Calculate the value of the stopping criterion
  stop_criteria <- abs(gradient(beta_new))

  # If stopping criteria less than tolerance, break
  if(is.na(stop_criteria) || stop_criteria <= tolerance) {
    last_iter <- iter + 1
    break
  }

  # if we never set last iter, then we hit the max number of iterations and
   ↪  need to set
  if(last_iter == 0) { last_iter <- max_iter }

  # end algorithm
}

return(list(betas = betas, obj_values = obj_values, eta_values =
 ↪  eta_values, last_iter = last_iter)) # in this case, beta(predictors)
 ↪  are the x values, obj_values are f(x), eta is the step size, last iter
 ↪  is the value in the vector of the final iteration before stopping
```

```
}
```

Running the gradient descent algorithm with backtracking:

```
minimize_backtrack <- gradient_descent_backtracking(tol = 1e-6, max_iter =
 ↪  10000, epsilon = 0.5, tau = 0.8, init_step_size = 1)
print(minimize_backtrack)
```

```
$betas
 [1] 3.7571481 2.0808951 1.7535339 1.5426377 1.3887564 1.2687146 1.1709946
 [8] 1.0890410 1.0187765 0.9574987 0.9033291 0.8549116 0.8112373 0.7715364
[15] 0.7352094 0.7017805 0.6708666 0.6421547 0.6153861 0.5903448 0.5668485
[22] 0.5447423 0.5238933 0.5041868 0.4855230 0.4678148 0.4509856 0.4349676
[29] 0.4197007 0.4051313 0.3912114 0.3778977 0.3651513 0.3529370 0.3412225
[36] 0.3299788 0.3191791 0.3087989 0.2988156 0.2892087 0.2799588 0.2710482
[43] 0.2624606 0.2541805 0.2461937 0.2384869 0.2310477 0.2238643 0.2169259
[50] 0.2102221
 [ reached 'max' / getOption("max.print") -- omitted 9950 entries ]

$obj_values
 [1] 228.498357  28.410229  16.604656  11.422582   8.576958   6.810204
 [7]   5.622724   4.778641   4.153059   3.674137   3.297869   2.995924
[13]   2.749315   2.544881   2.373241   2.227544   2.102680   1.994768
[19]   1.900814   1.818471   1.745879   1.681546   1.624259   1.573028
[25]   1.527035   1.485597   1.448143   1.414189   1.383326   1.355202
[31]   1.329516   1.306007   1.284449   1.264645   1.246422   1.229628
[37]   1.214129   1.199806   1.186554   1.174279   1.162897   1.152332
[43]   1.142516   1.133390   1.124896   1.116987   1.109616   1.102742
[49]   1.096328   1.090340
 [ reached 'max' / getOption("max.print") -- omitted 9950 entries ]

$eta_values
 [1] 1.000000000 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
 [7] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[13] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[19] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[25] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[31] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[37] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[43] 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698 0.007378698
[49] 0.007378698 0.007378698
```

```
 [ reached 'max' / getOption("max.print") -- omitted 9950 entries ]

$last_iter
[1] 505
```

```r
cat("The functions stopped after", minimize_backtrack$last_iter - 1,
 ↪  "iterations \n")
```

```
The functions stopped after 504 iterations
```

```r
cat("The function's point of minimization is", "(",
 ↪  minimize_backtrack$betas[minimize_backtrack$last_iter], ",",
 ↪  minimize_backtrack$obj_values[minimize_backtrack$last_iter], ") \n")
```

```
The function's point of minimization is ( 2.470678e-07 , 1 )
```

1) For the constant step size version of gradient descent, discuss how you selected the step size used in your code

Theoretical Analysis proves that for functions with a unique global minimum, the step size should be within 0 to 1/L to converge to the unique global minimum, where L is the Lipchitz constant, given by:

$$||\nabla f(x) - \nabla f(y)||_2 \leq L||x - y||_2$$

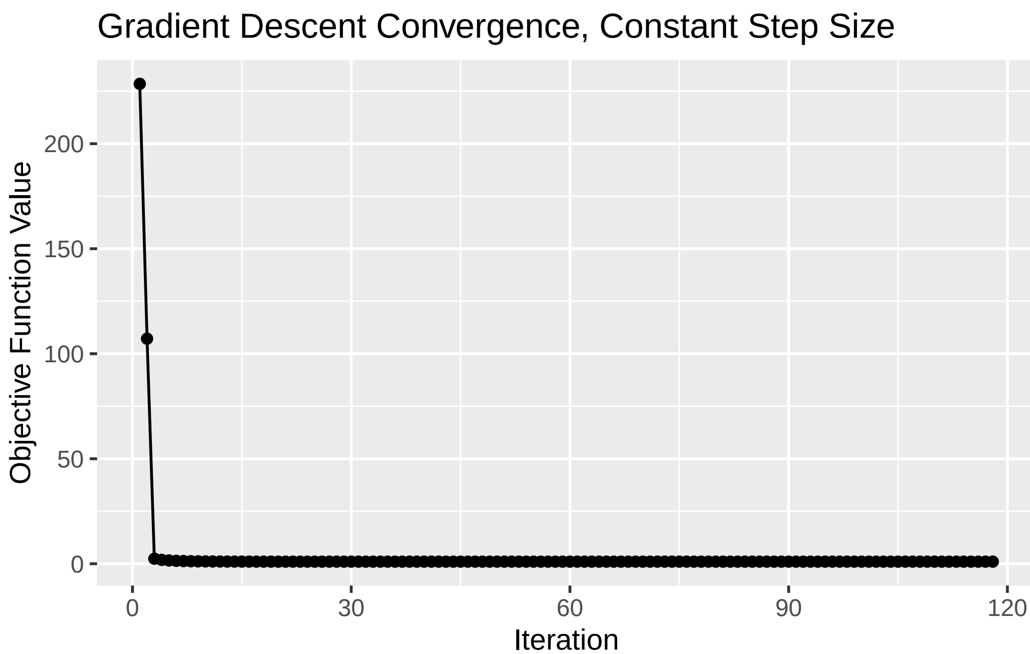
Since this cannot be calculated in practice, usually a small step size of 0.01 is what to begin with. From there, manually fine tuning to try 0.02 and 0.03 is a good idea to see if there's any better iterations. Starting from a big step size is usually unsafe do the algorithm overshooting and diverging instead of converging. Ultimately, the step size of 0.02 seemed best to reduce the number of iterations.

2) For both versions of the gradient descent algorithm, plot the value of $f(x_k)$ as a function of k the number of iterations

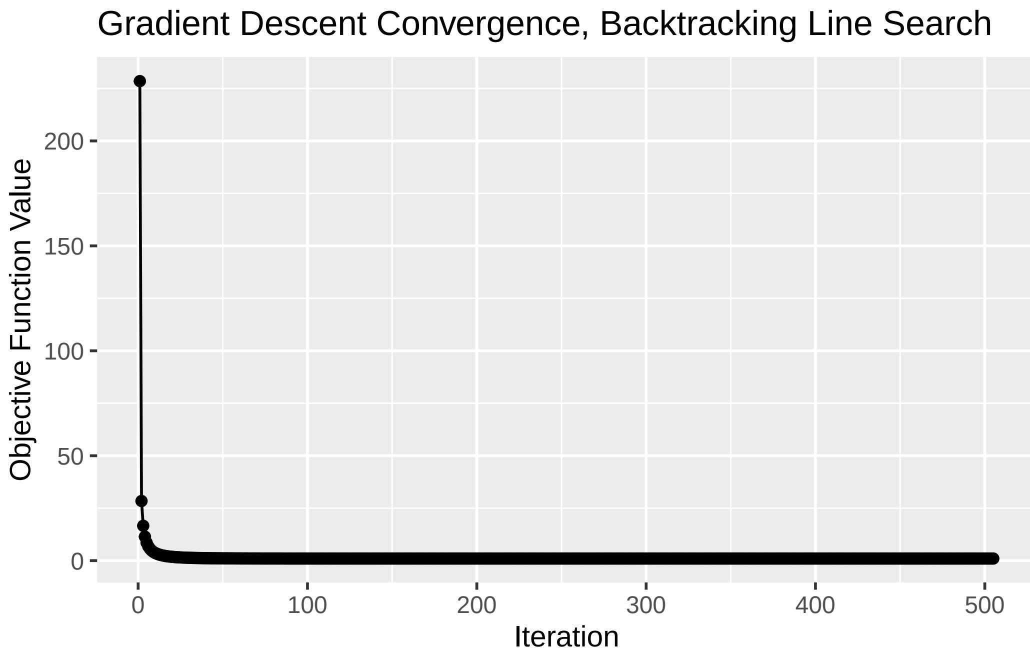
```r
# constant step size
iterations <- 1:minimize_constant_step$last_iter
obj_values <- (minimize_constant_step$obj_values)[iterations]
f_k_constant <- cbind(obj_values, iterations)
```

```r
iterations <- 1:minimize_backtrack$last_iter
obj_values <- (minimize_backtrack$obj_values)[iterations]
f_k_backtrack <- cbind(obj_values, iterations)

ggplot(f_k_constant, aes(x=iterations, y=obj_values)) +
  geom_point() +
  geom_line() +
  ggtitle("Gradient Descent Convergence, Constant Step Size") +
  xlab("Iteration") + ylab("Objective Function Value")
```



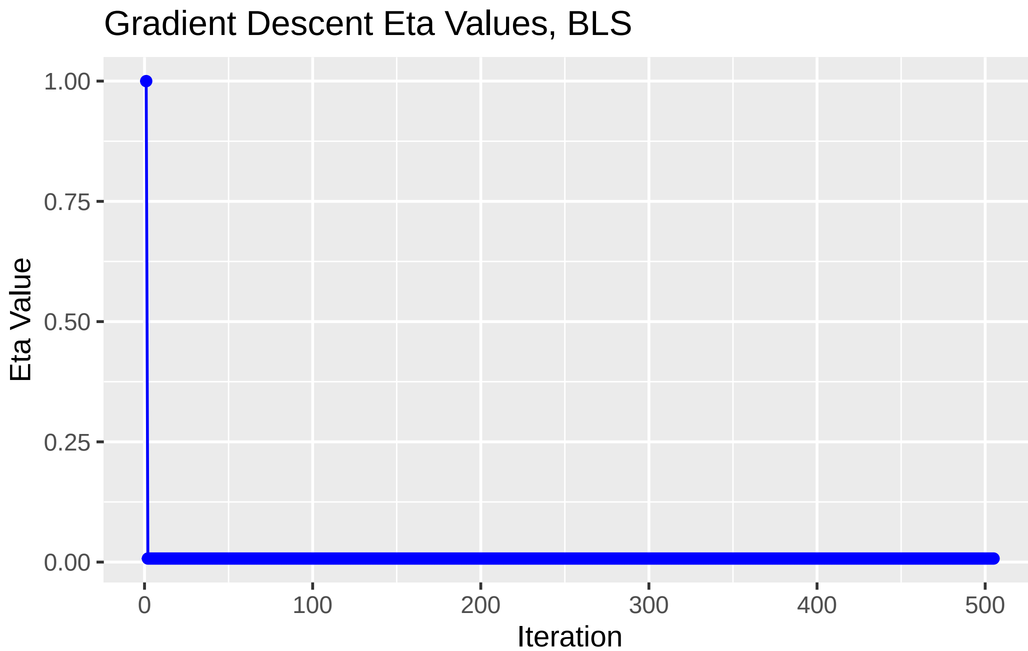

```r
ggplot(f_k_backtrack, aes(x=iterations, y=obj_values)) +
  geom_point() +
  geom_line() +
  ggtitle("Gradient Descent Convergence, Backtracking Line Search") +
  xlab("Iteration") + ylab("Objective Function Value")
```



3) For the the gradient descent method with backtracking line search, plot the step size η_k selected at step k as a function of k. Comment on the result

```
iterations <- 1:minimize_backtrack$last_iter
eta_values <- minimize_backtrack$eta_values[iterations]
eta_backtrack <- cbind(eta_values, iterations)

ggplot(eta_backtrack, aes(x=iterations, y=eta_values)) +
  geom_point(color = "blue") +
  geom_line(color = "blue") +
  ggtitle("Gradient Descent Eta Values, BLS") +
  xlab("Iteration") + ylab("Eta Value")
```



We can see that the step size was initially 0.02, but at the very first few iterations the Armijo condition immediately reduced the step size to a very small number < 0.005 in order to prevent overshooting. This condition held for the rest of the iterations until the algorithm converged eventually, after around 443 iterations. Compared to the constant step size gradient descent, the step size was much smaller for all the iterations, meaning that it converged with > 300 more iterations than the constant step size gradient descent. Although using a large step size like 0.02 would be much faster, it seems like the step sizes were chosen to be a safer bound so that the algorithm would not overshoot

Problem 2

To understand the sensitivity of the gradient descent algorithm and its variants to the "shape" of the function, the two data sets provided (dataset1.csv, dataset2.csv) will be used

They contain 100 observations for a response y and 20 predictors $x_j, j = 1, \cdots, 20$

Part a

Using the gradient descent code provided (both in R and Python) obtain the estimates of the regression coefficient, using both a constant step size and backtracking line search.

Read in the data and the gradient descent function:

```r
dataset1 <- read_csv("dataset1.csv")
```

```
Rows: 100 Columns: 21
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (21): Y, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X1...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
dataset2 <- read_csv("dataset2.csv")
```

```
Rows: 100 Columns: 21
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (21): Y, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X1...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# gradient descent given in class
gradient_descent_class <- function(X, y, eta = NULL, tol = 1e-6, max_iter =
↪  10000, backtracking = TRUE, epsilon = 0.5, tau = 0.8) {
  # Initialize
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  obj_values <- numeric(max_iter)
  eta_values <- numeric(max_iter)  # To store eta values used each iteration
  eta_bt <- 1  # Initial step size for backtracking

  # Objective function: Mean Squared Error (MSE)
  obj_function <- function(beta) {
    sum((X %*% beta - y)^2) / (2 * n)
  }

  # Gradient function
  gradient <- function(beta) {
```

```r
    t(X) %*% (X %*% beta - y) / n
  }

  for (iter in 1:max_iter) {
    grad <- gradient(beta)

    if (backtracking) {
      if (iter == 1) eta_bt <- 1  # Reset only in the first iteration
      beta_new <- beta - eta_bt * grad

      while (obj_function(beta_new) > obj_function(beta) - epsilon * eta_bt *
       ↪  sum(grad^2)) {
        eta_bt <- tau * eta_bt
        beta_new <- beta - eta_bt * grad
      }
      eta_used <- eta_bt
    } else {
      if (is.null(eta)) stop("When backtracking is FALSE, a fixed eta must be
       ↪  provided.")
      beta_new <- beta - eta * grad
      eta_used <- eta
    }

    eta_values[iter] <- eta_used

    obj_values[iter] <- obj_function(beta_new)

    if (sqrt(sum((beta_new - beta)^2)) < tol) {
      obj_values <- obj_values[1:iter]
      eta_values <- eta_values[1:iter]
      break
    }

    beta <- beta_new
  }

  return(list(beta = beta, obj_values = obj_values, eta_values = eta_values))
}
```

```r
X_dataset_1 <- dataset1 %>%
  select(-Y) %>%
  as.matrix()
```

```r
y_dataset_1 <- dataset1 %>%
  select(Y) %>%
  as.matrix()
X_dataset_2 <- dataset2 %>%
  select(-Y) %>%
  as.matrix()
y_dataset_2 <- dataset2 %>%
  select(Y) %>%
  as.matrix()

reg_const_dataset_1 <- gradient_descent_class(X_dataset_1, y_dataset_1,
↪  backtracking = FALSE, eta = 5)

cat("Constant Step Size: dataset1 \n")
```

```
Constant Step Size: dataset1
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(reg_const_dataset_1$beta)
```

```
               Y
X1   0.200284535
X2   0.071172300
X3   1.136549069
X4   0.143783956
X5  -0.007077920
X6  -0.012022200
X7   0.467646270
X8   0.038900577
X9   0.375594201
X10  1.193204076
X11  0.234086463
X12  1.001887238
X13  0.008026558
X14  0.647194983
X15 -0.075084887
```

```
X16  0.804818808
X17  1.021098020
X18  0.129799690
X19 -0.273712644
X20  0.401570242
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(reg_const_dataset_1$obj_values)
```

```
 [1] 0.4321102 0.3895214 0.3732853 0.3653793 0.3609190 0.3581722 0.3563922
 [8] 0.3552028 0.3543925 0.3538331 0.3534435 0.3531702 0.3529775 0.3528412
[15] 0.3527443 0.3526753 0.3526261 0.3525909 0.3525657 0.3525476 0.3525346
[22] 0.3525253 0.3525185 0.3525137 0.3525102 0.3525077 0.3525059 0.3525045
[29] 0.3525036 0.3525029 0.3525024 0.3525020 0.3525018 0.3525016 0.3525015
[36] 0.3525014 0.3525013 0.3525012 0.3525012 0.3525012 0.3525011 0.3525011
[43] 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011
[50] 0.3525011
 [ reached 'max' / getOption("max.print") -- omitted 28 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(reg_const_dataset_1$eta_values)
```

```
 [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[39] 5 5 5 5 5 5 5 5 5 5 5 5
 [ reached 'max' / getOption("max.print") -- omitted 28 entries ]
```

```r
cat("The functions stopped after",
↪  max(which(!is.na(reg_const_dataset_1$eta_values))), "iterations \n \n")
```

```
The functions stopped after 78 iterations
```

```r
reg_const_dataset_2 <- gradient_descent_class(X_dataset_2, y_dataset_2,
  backtracking = FALSE, eta = 0.02)

cat("Constant Step Size: dataset2 \n")
```

```
Constant Step Size: dataset2
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(reg_const_dataset_2$beta)
```

```
             Y
X1   0.2404371
X2   0.2959701
X3   0.5315576
X4   0.2003784
X5   0.4659575
X6   0.1774883
X7   0.3343755
X8  -0.0174628
X9   0.1653269
X10  0.7304544
X11  0.2279370
X12  0.3597918
X13  0.1830929
X14  0.2943043
X15 -0.1367969
X16  0.3184976
X17  0.4370496
X18  0.4642748
X19  0.2362962
X20  0.2229727
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(reg_const_dataset_2$obj_values)
```

```
 [1] 4.7108920 2.0434136 1.3308690 0.9980908 0.8045051 0.6817554 0.6001823
 [8] 0.5440527 0.5042603 0.4752878 0.4536824 0.4372228 0.4244423 0.4143483
[15] 0.4062534 0.3996715 0.3942518 0.3897372 0.3859359 0.3827035 0.3799294
[22] 0.3775285 0.3754344 0.3735948 0.3719685 0.3705223 0.3692296 0.3680689
[29] 0.3670223 0.3660753 0.3652158 0.3644335 0.3637198 0.3630673 0.3624698
[36] 0.3619217 0.3614183 0.3609555 0.3605295 0.3601371 0.3597753 0.3594416
[43] 0.3591335 0.3588490 0.3585862 0.3583432 0.3581185 0.3579107 0.3577184
[50] 0.3575404
 [ reached 'max' / getOption("max.print") -- omitted 8109 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(reg_const_dataset_2$eta_values)
```

```
 [1] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[16] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[31] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[46] 0.02 0.02 0.02 0.02 0.02
 [ reached 'max' / getOption("max.print") -- omitted 8109 entries ]
```

```r
cat("The functions stopped after",
 ↪  max(which(!is.na(reg_const_dataset_2$eta_values))), "iterations \n \n")
```

```
The functions stopped after 8159 iterations
```

```r
reg_bls_data1 <- gradient_descent_class(X_dataset_1, y_dataset_1, eta = NULL,
 ↪  tol = 1e-6, max_iter = 10000, backtracking = TRUE, epsilon = 0.5, tau =
 ↪  0.8)

cat("BLS: dataset1 \n")
```

```
BLS: dataset1
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(reg_bls_data1$beta)
```

```
               Y
X1   0.200288160
X2   0.071168997
X3   1.136538577
X4   0.143781908
X5  -0.007076641
X6  -0.012017800
X7   0.467649231
X8   0.038900395
X9   0.375589135
X10  1.193194040
X11  0.234088137
X12  1.001888404
X13  0.008031824
X14  0.647189722
X15 -0.075073783
X16  0.804823660
X17  1.021092124
X18  0.129789683
X19 -0.273703280
X20  0.401566025
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(reg_bls_data1$obj_values)
```

```
 [1] 0.5908489 0.5345036 0.4950110 0.4663575 0.4449979 0.4287255 0.4161031
 [8] 0.4061594 0.3982185 0.3917990 0.3865514 0.3822180 0.3786058 0.3755688
[15] 0.3729947 0.3707970 0.3689075 0.3672727 0.3658498 0.3646045 0.3635090
[22] 0.3625407 0.3616809 0.3609143 0.3602282 0.3596118 0.3590562 0.3585539
```

```
[29] 0.3580984 0.3576843 0.3573068 0.3569619 0.3566461 0.3563564 0.3560902
[36] 0.3558451 0.3556190 0.3554103 0.3552173 0.3550386 0.3548729 0.3547192
[43] 0.3545764 0.3544437 0.3543201 0.3542051 0.3540979 0.3539980 0.3539047
[50] 0.3538176
 [ reached 'max' / getOption("max.print") -- omitted 305 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(reg_bls_data1$eta_values)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1
 [ reached 'max' / getOption("max.print") -- omitted 305 entries ]
```

```r
cat("The functions stopped after",
↪  max(which(!is.na(reg_bls_data1$eta_values))), "iterations \n \n")
```

```
The functions stopped after 355 iterations
```

```r
reg_bls_data2 <- gradient_descent_class(X_dataset_2, y_dataset_2, eta = NULL,
↪  tol = 1e-6, max_iter = 10000, backtracking = TRUE, epsilon = 0.5, tau =
↪  0.8)

cat("BLS: dataset2 \n")
```

```
BLS: dataset2
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(reg_bls_data2$beta)
```

```
              Y
X1   0.24043923
X2   0.29598085
X3   0.53158006
X4   0.20040437
X5   0.46595899
X6   0.17748224
X7   0.33432770
X8  -0.01747586
X9   0.16533621
X10  0.73052099
X11  0.22793299
X12  0.35975400
X13  0.18307123
X14  0.29433152
X15 -0.13688470
X16  0.31846923
X17  0.43704902
X18  0.46434151
X19  0.23625194
X20  0.22297328
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(reg_bls_data2$obj_values)
```

```
 [1] 3.9233028 1.7852238 1.1730966 0.8802603 0.7137618 0.6115344 0.5454485
 [8] 0.5008908 0.4697418 0.4472740 0.4306231 0.4179897 0.4082051 0.4004876
[15] 0.3943000 0.3892646 0.3851109 0.3816415 0.3787103 0.3762080 0.3740515
[22] 0.3721774 0.3705363 0.3690897 0.3678071 0.3666643 0.3656416 0.3647229
[29] 0.3638952 0.3631472 0.3624700 0.3618554 0.3612969 0.3607887 0.3603255
[36] 0.3599031 0.3595175 0.3591653 0.3588432 0.3585486 0.3582791 0.3580323
[43] 0.3578062 0.3575990 0.3574091 0.3572350 0.3570753 0.3569287 0.3567942
[50] 0.3566707
 [ reached 'max' / getOption("max.print") -- omitted 7348 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(reg_bls_data2$eta_values)
```

```
 [1] 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518
 [9] 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518
[17] 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518
[25] 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518
[33] 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518
[41] 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518 0.022518
[49] 0.022518 0.022518
 [ reached 'max' / getOption("max.print") -- omitted 7348 entries ]
```

```r
cat("The functions stopped after",
↪  max(which(!is.na(reg_bls_data2$eta_values))), "iterations \n \n")
```

```
The functions stopped after 7398 iterations
```

1) Discuss how you selected the constant step size. Also, discuss which convergence criterion you used and the tolerance parameter used

Because we cannot find the Lipchitz constant, constant step size is tuned manually. For data set 1, the step size initially was set small = 0.01, but it did not converge, so I increased the step size manually until it converged at step size = 1. From there, I pushed the step size until the function would not converge anymore. The max step size I could use without 5. For data set 2, I started with 0.01 step size, and eventually a step size of 0.02 was chosen in order to make the algorithm converge in 8159 iterations. For the tolerance parameter, a tolerance of $1e^{-}6$ was used, which is a standard tolerance parameter and is very close approximation to the real solution. For the convergence criterion, the stop criteria given in the code was to terminate when the difference of beta between iterations is less than the tolerance, indicating that the function has converged.

2) Compare the results with those obtained from the lm command in R or from the class LinearRegression from the sklearn.linear model in Python.

Specifically, calculate $\parallel \widehat{\beta_{GD}} - \hat{\beta} \parallel_2$, where $\widehat{\beta_{GD}}$ is the estimate of the regression coefficient obtained from the gradient descent algorithm (both with constant step size and backtracking line search) and $\hat{\beta}$ obtained from the least squares solution implemented in R or Python

Use R's regression class:

```
m1 <- lm(data = dataset1, Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 +
 ↪  X10 +
           X11 + X12 + X13 + X14 + X15 + X16 + X17 + X18 + X19 + X20)
summary(m1)
```

```

Call:
lm(formula = Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 +
    X10 + X11 + X12 + X13 + X14 + X15 + X16 + X17 + X18 + X19 +
    X20, data = dataset1)

Residuals:
     Min       1Q   Median       3Q      Max
-2.27711 -0.57530 -0.03033  0.58303  1.73622

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.120492   0.105930   1.137 0.258783
X1           0.117292   0.306769   0.382 0.703231
X2           0.092145   0.225586   0.408 0.684034
X3           1.139352   0.338447   3.366 0.001178 **
X4           0.153116   0.263039   0.582 0.562154
X5           0.058437   0.286019   0.204 0.838636
X6          -0.105235   0.311715  -0.338 0.736561
X7           0.474469   0.357023   1.329 0.187687
X8          -0.008262   0.249096  -0.033 0.973625
X9           0.391447   0.323946   1.208 0.230509
 [ reached 'max' / getOption("max.print") -- omitted 11 rows ]
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.937 on 79 degrees of freedom
Multiple R-squared:  0.4812,    Adjusted R-squared:  0.3498
F-statistic: 3.663 on 20 and 79 DF,  p-value: 1.85e-05
```

```
m2 <- lm(data = dataset2, Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 +
 ↪  X10 +
           X11 + X12 + X13 + X14 + X15 + X16 + X17 + X18 + X19 + X20)
summary(m2)
```

```
Call:
lm(formula = Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 +
    X10 + X11 + X12 + X13 + X14 + X15 + X16 + X17 + X18 + X19 +
    X20, data = dataset2)

Residuals:
     Min       1Q   Median       3Q      Max
-2.27711 -0.57530 -0.03033  0.58303  1.73622

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.12049    0.10593   1.137 0.258783
X1           0.23737    0.02469   9.616 6.14e-15 ***
X2           0.30042    0.03972   7.564 6.14e-11 ***
X3           0.54373    0.08117   6.699 2.80e-09 ***
X4           0.21022    0.08549   2.459 0.016113 *
X5           0.47340    0.02502  18.917  < 2e-16 ***
X6           0.17036    0.03822   4.458 2.70e-05 ***
X7           0.32586    0.15408   2.115 0.037590 *
X8          -0.02699    0.04551  -0.593 0.554827
X9           0.16835    0.04326   3.892 0.000206 ***
 [ reached 'max' / getOption("max.print") -- omitted 11 rows ]
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.937 on 79 degrees of freedom
Multiple R-squared:  0.9879,    Adjusted R-squared:  0.9848
F-statistic: 321.2 on 20 and 79 DF,  p-value: < 2.2e-16
```

Plotting a table comparing β to β_{GD}:

```r
# transform dataset 1 coefs
beta_gd_d1 <- reg_const_dataset_1$beta
beta_1 <- m1$coefficients

names <- attributes(beta_gd_d1)$dimnames[[1]]
b_gd_1 <- as.vector(beta_gd_d1)
lm_1 <- as.vector(beta_1[-1])

d1 <- data.frame(Coef = names, Beta_GD = b_gd_1, Beta = lm_1)
```

```r
# transform dataset 2 coefs
beta_gd_d2 <- reg_const_dataset_2$beta
beta_2 <- m2$coefficients

names <- attributes(beta_gd_d2)$dimnames[[1]]
b_gd_2 <- as.vector(beta_gd_d2)
lm_2 <- as.vector(beta_2[-1])

d2 <- data.frame(Coef = names, Beta_GD = b_gd_2, Beta = lm_2)

# show the difference between Beta_GD and Beta
d1 <- d1 %>%
  mutate(`Beta_GD - Beta` = Beta_GD - Beta)

d2 <- d2 %>%
  mutate(`Beta_GD - Beta` = Beta_GD - Beta)

print(d1)
```

```
   Coef     Beta_GD         Beta Beta_GD - Beta
1    X1  0.20028453  0.117292007    0.082992528
2    X2  0.07117230  0.092144880   -0.020972579
3    X3  1.13654907  1.139351598   -0.002802530
4    X4  0.14378396  0.153116480   -0.009332524
5    X5 -0.00707792  0.058436652   -0.065514572
6    X6 -0.01202220 -0.105234557    0.093212357
7    X7  0.46764627  0.474468590   -0.006822320
8    X8  0.03890058 -0.008261791    0.047162368
9    X9  0.37559420  0.391447179   -0.015852978
10  X10  1.19320408  1.220257571   -0.027053495
11  X11  0.23408646  0.225458882    0.008627581
12  X12  1.00188724  1.061147318   -0.059260080
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

```r
print(d2)
```

```
   Coef    Beta_GD      Beta Beta_GD - Beta
1    X1 0.2404371 0.2373680    0.003069086
2    X2 0.2959701 0.3004188   -0.004448689
3    X3 0.5315576 0.5437310   -0.012173377
4    X4 0.2003784 0.2102227   -0.009844279
```

```
5     X5  0.4659575  0.4734042   -0.007446745
6     X6  0.1774883  0.1703577    0.007130587
7     X7  0.3343755  0.3258614    0.008514185
8     X8 -0.0174628 -0.0269920    0.009529200
9     X9  0.1653269  0.1683506   -0.003023745
10   X10  0.7304544  0.7513985   -0.020944139
11   X11  0.2279370  0.2244565    0.003480567
12   X12  0.3597918  0.3547595    0.005032282
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

Calculating the norm of difference in Betas:

The formula is given as:

$$||\hat{\beta}_{GD} - \hat{\beta}||_2 = \sqrt{\sum_{i=1}^{20}(\hat{\beta}_{GD,i} - \hat{\beta}_i)^2}$$

Calculating the norm of the beta difference:

```r
d1_norm <- sqrt(sum(d1$`Beta_GD - Beta`^2))
d2_norm <- sqrt(sum(d2$`Beta_GD - Beta`^2))
cat("Dataset 1 Norm:", d1_norm, "\n")
```

```
Dataset 1 Norm: 0.1956172
```

```r
cat("Dataset 2 Norm:", d2_norm, "\n")
```

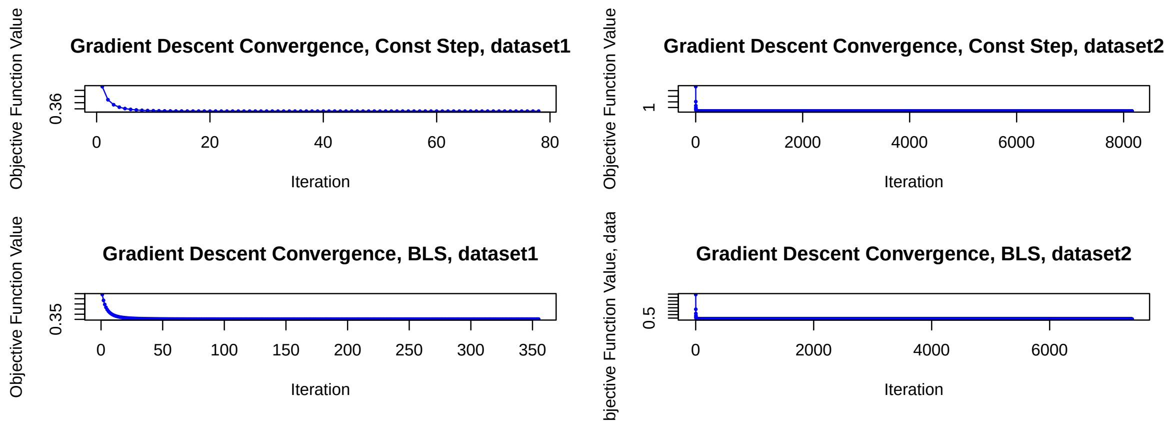
```
Dataset 2 Norm: 0.05606839
```

3) Plot the value of the objective function as a function of the number of iterations required

```r
par(mfrow=c(2,2))

plot(reg_const_dataset_1$obj_values, type = "o", col = "blue", pch = 16, cex
 ↪  = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Gradient Descent Convergence, Const Step, dataset1")
plot(reg_const_dataset_2$obj_values, type = "o", col = "blue", pch = 16, cex
 ↪  = 0.5,
```

```
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Gradient Descent Convergence, Const Step, dataset2")
plot(reg_bls_data1$obj_values, type = "o", col = "blue", pch = 16, cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Gradient Descent Convergence, BLS, dataset1")
plot(reg_bls_data2$obj_values, type = "o", col = "blue", pch = 16, cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value, dataset2",
     main = "Gradient Descent Convergence, BLS, dataset2")
```



Part b

Implement the Polyak and Nesterov momentum methods and obtain the estimates of the regression coefficients, using both a constant step size and backtracking line search

Polyak momentum with constant step size:

```
# polyak momentum constant step
polyak_constant_step <- function(X, y, eta = NULL, tol = 1e-6, max_iter =
↪  10000, xi = 0.5) {
  # Initialize
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  obj_values <- numeric(max_iter)
  eta_values <- numeric(max_iter)  # To store eta values used each iteration
  beta_values <- list() # To store beta values used each iteration
  eta_bt <- 1  # Initial step size for backtracking
  backtracking <- FALSE
```

```r
# Objective function: Mean Squared Error (MSE)
obj_function <- function(beta) {
  sum((X %*% beta - y)^2) / (2 * n)
}

# Gradient function
gradient <- function(beta) {
  t(X) %*% (X %*% beta - y) / n
}



for (iter in 1:max_iter) {
  grad <- gradient(beta)
  beta_values[[iter]] <- beta

  if (backtracking) {
    if (iter == 1) eta_bt <- 1  # Reset only in the first iteration
    beta_new <- beta - eta_bt * grad

    while (obj_function(beta_new) > obj_function(beta) - epsilon * eta_bt *
     ↪  sum(grad^2)) {
      eta_bt <- tau * eta_bt
      beta_new <- beta - eta_bt * grad
    }
    eta_used <- eta_bt
  } else {
    if (is.null(eta)) stop("When backtracking is FALSE, a fixed eta must be
     ↪  provided.")
    if(iter == 1) {
      y_k <- beta
    } else {
      beta_prev <- beta_values[[iter - 1]]
      y_k <- beta + xi * (beta - beta_prev)
    }
    beta_new <- y_k - eta * grad
    beta_values[[iter+1]] <- beta_new
    eta_used <- eta
  }

  eta_values[iter] <- eta_used
```

```r
    obj_values[iter] <- obj_function(beta_new)

    if (sqrt(sum((beta_new - beta)^2)) < tol) {
      obj_values <- obj_values[1:iter]
      eta_values <- eta_values[1:iter]
      break
    }

    beta <- beta_new
  }

  return(list(beta = beta, obj_values = obj_values, eta_values = eta_values,
   ↪  beta_values = beta_values))
}

polyak_reg_constant_1 <- polyak_constant_step(X_dataset_1, y_dataset_1, eta =
 ↪  5, tol = 1e-6, max_iter = 10000, xi = 0.5)

cat("Polyak Constant Step Size: dataset1 \n")
```

```
Polyak Constant Step Size: dataset1
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(polyak_reg_constant_1$beta)
```

```
               Y
X1    0.200283811
X2    0.071173054
X3    1.136551334
X4    0.143784177
X5   -0.007078068
X6   -0.012022932
X7    0.467645816
X8    0.038900742
X9    0.375595445
X10   1.193206312
```

```
X11  0.234085961
X12  1.001887132
X13  0.008025497
X14  0.647196312
X15 -0.075087258
X16  0.804817840
X17  1.021099150
X18  0.129801780
X19 -0.273714583
X20  0.401571366
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(polyak_reg_constant_1$obj_values)
```

```
 [1] 0.4321102 0.3946719 0.3821221 0.3632621 0.3598611 0.3547737 0.3537785
 [8] 0.3533852 0.3527468 0.3526611 0.3526151 0.3525518 0.3525253 0.3525148
[15] 0.3525065 0.3525034 0.3525027 0.3525017 0.3525015 0.3525012 0.3525012
[22] 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011
[29] 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011
[36] 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011 0.3525011
[43] 0.3525011
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(polyak_reg_constant_1$eta_values)
```

```
 [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[39] 5 5 5 5 5
```

```r
cat("The functions stopped after",
↪  max(which(!is.na(polyak_reg_constant_1$eta_values))), "iterations \n \n")
```

```
The functions stopped after 43 iterations
```

```r
polyak_reg_constant_2 <- polyak_constant_step(X_dataset_2, y_dataset_2, eta =
 ↪  0.02, tol = 1e-6, max_iter = 10000, xi = 0.5)

cat("Polyak Constant Step Size: dataset2 \n")
```

```
Polyak Constant Step Size: dataset2
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(polyak_reg_constant_2$beta)
```

```
              Y
X1   0.24044664
X2   0.29601821
X3   0.53165797
X4   0.20049437
X5   0.46596432
X6   0.17746121
X7   0.33416170
X8  -0.01752115
X9   0.16536857
X10  0.73075205
X11  0.22791901
X12  0.35962283
X13  0.18299591
X14  0.29442593
X15 -0.13718941
X16  0.31837082
X17  0.43704688
X18  0.46457311
X19  0.23609841
X20  0.22297533
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(polyak_reg_constant_2$obj_values)
```

```
 [1] 4.7108920 4.0114293 3.3391611 1.2443903 0.7436955 0.7480855 0.5237822
 [8] 0.4361249 0.4216429 0.3906799 0.3813611 0.3761214 0.3705711 0.3678586
[15] 0.3655436 0.3637019 0.3622549 0.3611302 0.3601803 0.3593992 0.3587529
[22] 0.3582082 0.3577529 0.3573694 0.3570472 0.3567758 0.3565466 0.3563531
[29] 0.3561895 0.3560509 0.3559333 0.3558333 0.3557482 0.3556754 0.3556131
[36] 0.3555596 0.3555134 0.3554734 0.3554386 0.3554081 0.3553813 0.3553576
[43] 0.3553365 0.3553175 0.3553004 0.3552848 0.3552705 0.3552573 0.3552450
[50] 0.3552335
 [ reached 'max' / getOption("max.print") -- omitted 4522 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(polyak_reg_constant_2$eta_values)
```

```
 [1] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[16] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[31] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[46] 0.02 0.02 0.02 0.02 0.02
 [ reached 'max' / getOption("max.print") -- omitted 4522 entries ]
```

```r
cat("The functions stopped after",
 ↪  max(which(!is.na(polyak_reg_constant_2$eta_values))), "iterations \n \n")
```

```
The functions stopped after 4572 iterations
```

Nesterov Constant Step:

```r
# nesterov momentum constant step
nesterov_constant_step <- function(X, y, eta = NULL, tol = 1e-6, max_iter =
 ↪  10000) {
  # Initialize
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  obj_values <- numeric(max_iter)
  eta_values <- numeric(max_iter)  # To store eta values used each iteration
  beta_values <- list() # To store beta values used each iteration
  eta_bt <- 1  # Initial step size for backtracking
  backtracking <- FALSE

  # Objective function: Mean Squared Error (MSE)
  obj_function <- function(beta) {
    sum((X %*% beta - y)^2) / (2 * n)
  }

  # Gradient function
  gradient <- function(beta) {
    t(X) %*% (X %*% beta - y) / n
  }



  for (iter in 1:max_iter) {
    grad <- gradient(beta)
    beta_values[[iter]] <- beta

    if (backtracking) {
      if (iter == 1) eta_bt <- 1  # Reset only in the first iteration
      beta_new <- beta - eta_bt * grad

      while (obj_function(beta_new) > obj_function(beta) - epsilon * eta_bt *
       ↪  sum(grad^2)) {
        eta_bt <- tau * eta_bt
        beta_new <- beta - eta_bt * grad
      }
      eta_used <- eta_bt
    } else {
      if (is.null(eta)) stop("When backtracking is FALSE, a fixed eta must be
       ↪  provided.")
      if(iter == 1) {
```

```r
        y_k <- beta
      } else {
        beta_prev <- beta_values[[iter - 1]]
        xi <- (iter - 1) / (iter + 2)
        y_k <- beta + xi * (beta - beta_prev)
      }
      beta_new <- y_k - eta * grad
      beta_values[[iter+1]] <- beta_new
      eta_used <- eta
    }

    eta_values[iter] <- eta_used

    obj_values[iter] <- obj_function(beta_new)

    if (sqrt(sum((beta_new - beta)^2)) < tol) {
      obj_values <- obj_values[1:iter]
      eta_values <- eta_values[1:iter]
      break
    }

    beta <- beta_new
  }

  return(list(beta = beta, obj_values = obj_values, eta_values = eta_values,
   ↪  beta_values = beta_values))
}

nesterov_reg_constant_1 <- nesterov_constant_step(X_dataset_1, y_dataset_1,
 ↪  eta = 5, tol = 1e-6, max_iter = 100000)

cat("Nesterov Constant Step Size: dataset1 \n")
```

```
Nesterov Constant Step Size: dataset1
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(nesterov_reg_constant_1$beta)
```

```
                Y
X1   0.200282647
X2   0.071172727
X3   1.136552000
X4   0.143784961
X5  -0.007078179
X6  -0.012023914
X7   0.467644858
X8   0.038900556
X9   0.375594538
X10  1.193208086
X11  0.234086367
X12  1.001886829
X13  0.008025014
X14  0.647196533
X15 -0.075089128
X16  0.804817210
X17  1.021101458
X18  0.129803283
X19 -0.273716587
X20  0.401571313
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(nesterov_reg_constant_1$obj_values)
```

```
 [1] 0.4321102 0.3782396 0.3658423 0.3590872 0.3564348 0.3542454 0.3534965
 [8] 0.3536559 0.3532581 0.3531900 0.3531580 0.3530627 0.3527671 0.3527248
[15] 0.3526208 0.3526519 0.3526621 0.3526990 0.3526422 0.3526128 0.3525603
[22] 0.3525603 0.3525685 0.3525585 0.3525438 0.3525547 0.3525592 0.3525498
[29] 0.3525354 0.3525249 0.3525185 0.3525233 0.3525261 0.3525301 0.3525301
[36] 0.3525221 0.3525155 0.3525120 0.3525144 0.3525112 0.3525163 0.3525161
[43] 0.3525160 0.3525105 0.3525113 0.3525103 0.3525086 0.3525049 0.3525076
[50] 0.3525117
 [ reached 'max' / getOption("max.print") -- omitted 14467 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(nesterov_reg_constant_1$eta_values)
```

```
 [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[39] 5 5 5 5 5 5 5 5 5 5 5 5
 [ reached 'max' / getOption("max.print") -- omitted 14467 entries ]
```

```r
cat("The functions stopped after",
↪  max(which(!is.na(nesterov_reg_constant_1$eta_values))), "iterations \n
↪  \n")
```

```
The functions stopped after 14517 iterations
```

```r
nesterov_reg_constant_2 <- nesterov_constant_step(X_dataset_2, y_dataset_2,
↪  eta = 0.01, tol = 1e-6, max_iter = 100000)

cat("Nesterov Constant Step Size: dataset2 \n")
```

```
Nesterov Constant Step Size: dataset2
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(nesterov_reg_constant_2$beta)
```

```
             Y
X1  0.24045810
X2  0.29607939
X3  0.53177889
X4  0.20063911
X5  0.46597238
```

```
X6   0.17742939
X7   0.33389131
X8  -0.01759342
X9   0.16542136
X10  0.73112261
X11  0.22789784
X12  0.35941098
X13  0.18287303
X14  0.29457632
X15 -0.13767622
X16  0.31821281
X17  0.43704273
X18  0.46494420
X19  0.23585233
X20  0.22297746
```

```
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```
print(nesterov_reg_constant_2$obj_values)
```

```
 [1] 12.7261939  4.9538959  2.2199145  1.5501199  1.3087877  1.0157432
 [7]  0.7246570  0.5566468  0.5135953  0.5114170  0.4889637  0.4509100
[13]  0.4247826  0.4150412  0.4077391  0.3968021  0.3884065  0.3862455
[19]  0.3853858  0.3808638  0.3742562  0.3694329  0.3668043  0.3644834
[25]  0.3620628  0.3606785  0.3608835  0.3621189  0.3634197  0.3638922
[31]  0.3632525  0.3621456  0.3612985  0.3605845  0.3596021  0.3586588
[37]  0.3583332  0.3585248  0.3587091  0.3587133  0.3585982  0.3582155
[43]  0.3574913  0.3568067  0.3565781  0.3566828  0.3566768  0.3564065
[49]  0.3561066  0.3560015
 [ reached 'max' / getOption("max.print") -- omitted 12699 entries ]
```

```
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```
print(nesterov_reg_constant_2$eta_values)
```

```
 [1] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
[16] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
[31] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
[46] 0.01 0.01 0.01 0.01 0.01
 [ reached 'max' / getOption("max.print") -- omitted 12699 entries ]
```

```
cat("The functions stopped after",
↪  max(which(!is.na(nesterov_reg_constant_2$eta_values))), "iterations \n
↪  \n")
```

```
The functions stopped after 12749 iterations
```

BLS with Polyak:

```
# polyak momentum constant step
polyak_bls <- function(X, y, eta = NULL, tol = 1e-6, max_iter = 10000, xi =
↪  0.5, epsilon = 0.5, tau = 0.5, backtracking=TRUE) {
  # Initialize
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  obj_values <- numeric(max_iter)
  eta_values <- numeric(max_iter)  # To store eta values used each iteration
  beta_values <- list() # To store beta values used each iteration
  eta_bt <- 1  # Initial step size for backtracking

  # Objective function: Mean Squared Error (MSE)
  obj_function <- function(beta) {
    sum((X %*% beta - y)^2) / (2 * n)
  }

  # Gradient function
  gradient <- function(beta) {
    t(X) %*% (X %*% beta - y) / n
  }
```

```r
for (iter in 1:max_iter) {
  grad <- gradient(beta)
  beta_values[[iter]] <- beta

  if (backtracking) {
    if (iter == 1) {
      eta_bt <- 1 # Reset only in the first iteration
      y_k <- beta
    }
    else {
      beta_prev <- beta_values[[iter - 1]]
      y_k <- beta + xi * (beta - beta_prev)
    }
    beta_new <- y_k - eta_bt * grad

    while (obj_function(beta_new) > obj_function(beta) - epsilon * eta_bt *
     ↪  sum(grad^2)) {
      eta_bt <- tau * eta_bt
      beta_new <- beta - eta_bt * grad
    }
    eta_used <- eta_bt
  } else {
    if (is.null(eta)) stop("When backtracking is FALSE, a fixed eta must be
     ↪  provided.")
    if(iter == 1) {
      y_k <- beta
    } else {
      beta_prev <- beta_values[[iter - 1]]
      y_k <- beta + xi * (beta - beta_prev)
    }
    beta_new <- y_k - eta * grad
    beta_values[[iter+1]] <- beta_new
    eta_used <- eta
  }

  eta_values[iter] <- eta_used

  obj_values[iter] <- obj_function(beta_new)

  if (sqrt(sum((beta_new - beta)^2)) < tol) {
    obj_values <- obj_values[1:iter]
    eta_values <- eta_values[1:iter]
```

```r
      break
    }

    beta <- beta_new
  }

  return(list(beta = beta, obj_values = obj_values, eta_values = eta_values,
   ↪  beta_values = beta_values))
}

polyak_bls_1 <- polyak_bls(X_dataset_1, y_dataset_1, eta = NULL, tol = 1e-6,
 ↪  max_iter = 10000, xi = 0.5, backtracking=TRUE, epsilon = 0.5, tau = 0.5)

cat("Polyak BLS Step Size: dataset1 \n")
```

```
Polyak BLS Step Size: dataset1
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(polyak_bls_1$beta)
```

```
               Y
X1   0.200285797
X2   0.071171129
X3   1.136545392
X4   0.143783222
X5  -0.007077460
X6  -0.012020668
X7   0.467647339
X8   0.038900514
X9   0.375592418
X10  1.193200546
X11  0.234087023
X12  1.001887660
X13  0.008028429
X14  0.647193125
X15 -0.075080976
```

```
X16  0.804820494
X17  1.021095940
X18  0.129796148
X19 -0.273709345
X20  0.401568770
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(polyak_bls_1$obj_values)
```

```
 [1] 0.5908489 0.5059182 0.4486420 0.4149280 0.3950842 0.3829025 0.3751543
 [8] 0.3700089 0.3663906 0.3637245 0.3617119 0.3601682 0.3589589 0.3579873
[15] 0.3571901 0.3565265 0.3559689 0.3554973 0.3550960 0.3547530 0.3544587
[22] 0.3542053 0.3539864 0.3537970 0.3536327 0.3534899 0.3533657 0.3532575
[29] 0.3531632 0.3530808 0.3530088 0.3529459 0.3528909 0.3528428 0.3528007
[36] 0.3527638 0.3527315 0.3527031 0.3526783 0.3526566 0.3526375 0.3526208
[43] 0.3526061 0.3525932 0.3525819 0.3525720 0.3525634 0.3525557 0.3525491
[50] 0.3525432
 [ reached 'max' / getOption("max.print") -- omitted 126 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(polyak_bls_1$eta_values)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1
 [ reached 'max' / getOption("max.print") -- omitted 126 entries ]
```

```r
cat("The functions stopped after", max(which(!is.na(polyak_bls_1
 ↪  $eta_values))), "iterations \n \n")
```

```
The functions stopped after 176 iterations
```

```
polyak_bls_2 <- polyak_bls(X_dataset_2, y_dataset_2, eta = NULL, tol = 1e-6,
↪  max_iter = 10000, xi = 0.5, backtracking=TRUE, epsilon = 0.5, tau = 0.5)

cat("Polyak BLS Step Size: dataset2 \n")
```

```
Polyak BLS Step Size: dataset2
```

```
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```
print(polyak_bls_2$beta)
```

```
              Y
X1   0.24043177
X2   0.29594329
X3   0.53150172
X4   0.20031389
X5   0.46595362
X6   0.17750338
X7   0.33449461
X8  -0.01743032
X9   0.16530367
X10  0.73028868
X11  0.22794705
X12  0.35988589
X13  0.18314696
X14  0.29423658
X15 -0.13657832
X16  0.31856818
X17  0.43705118
X18  0.46410865
X19  0.23640631
X20  0.22297121
```

```
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(polyak_bls_2$obj_values)
```

```
 [1] 7.2567507 2.7227794 1.8892170 1.3368668 1.0994139 0.9276946 0.7739216
 [8] 0.6651733 0.5954111 0.5464733 0.5100746 0.4832346 0.4631262 0.4474229
[15] 0.4348435 0.4246651 0.4163332 0.4094118 0.4035910 0.3986519 0.3944279
[22] 0.3907866 0.3876234 0.3848555 0.3824175 0.3802566 0.3783298 0.3766022
[29] 0.3750454 0.3736358 0.3723540 0.3711840 0.3701122 0.3691275 0.3682201
[36] 0.3673820 0.3666062 0.3658867 0.3652183 0.3645965 0.3640172 0.3634771
[43] 0.3629728 0.3625018 0.3620613 0.3616492 0.3612634 0.3609021 0.3605636
[50] 0.3602462
 [ reached 'max' / getOption("max.print") -- omitted 9937 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(polyak_bls_2$eta_values)
```

```
 [1] 0.0156250 0.0156250 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
 [8] 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
[15] 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
[22] 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
[29] 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
[36] 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
[43] 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125 0.0078125
[50] 0.0078125
 [ reached 'max' / getOption("max.print") -- omitted 9937 entries ]
```

```r
cat("The functions stopped after",
↪  max(which(!is.na(polyak_bls_2$eta_values))), "iterations \n \n")
```

```
The functions stopped after 9987 iterations
```

BLS with Nesterov:

```
# nesterov momentum constant step
nesterov_bls <- function(X, y, eta = NULL, tol = 1e-6, max_iter = 10000,
↪  epsilon = 0.5, tau = 0.5, backtracking=TRUE) {
  # Initialize
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  obj_values <- numeric(max_iter)
  eta_values <- numeric(max_iter)  # To store eta values used each iteration
  beta_values <- list() # To store beta values used each iteration
  eta_bt <- 1  # Initial step size for backtracking

  # Objective function: Mean Squared Error (MSE)
  obj_function <- function(beta) {
    sum((X %*% beta - y)^2) / (2 * n)
  }

  # Gradient function
  gradient <- function(beta) {
    t(X) %*% (X %*% beta - y) / n
  }



  for (iter in 1:max_iter) {
    grad <- gradient(beta)
    beta_values[[iter]] <- beta

    if (backtracking) {
      if (iter == 1) {
        eta_bt <- 1 # Reset only in the first iteration
        y_k <- beta
      }
      else {
        beta_prev <- beta_values[[iter - 1]]
        xi <- (iter - 1) / (iter + 2)
        y_k <- beta + xi * (beta - beta_prev)
      }
      beta_new <- y_k - eta_bt * grad

      while (obj_function(beta_new) > obj_function(beta) - epsilon * eta_bt *
       ↪  sum(grad^2)) {
        eta_bt <- tau * eta_bt
```

```r
        beta_new <- beta - eta_bt * grad
      }
      eta_used <- eta_bt
    } else {
      if (is.null(eta)) stop("When backtracking is FALSE, a fixed eta must be
      ↪  provided.")
      if(iter == 1) {
        y_k <- beta
      } else {
        beta_prev <- beta_values[[iter - 1]]
        y_k <- beta + xi * (beta - beta_prev)
      }
      beta_new <- y_k - eta * grad
      beta_values[[iter+1]] <- beta_new
      eta_used <- eta
    }

    eta_values[iter] <- eta_used

    obj_values[iter] <- obj_function(beta_new)

    if (sqrt(sum((beta_new - beta)^2)) < tol) {
      obj_values <- obj_values[1:iter]
      eta_values <- eta_values[1:iter]
      break
    }

    beta <- beta_new
  }

  return(list(beta = beta, obj_values = obj_values, eta_values = eta_values,
  ↪  beta_values = beta_values))
}

nesterov_bls_1 <- nesterov_bls(X_dataset_1, y_dataset_1, eta = NULL, tol =
↪  1e-6, max_iter = 10000, backtracking=TRUE, epsilon = 0.5, tau = 0.5)

cat("Nesterov BLS Step Size: dataset1 \n")
```

```
Nesterov BLS Step Size: dataset1
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(nesterov_bls_1$beta)
```

```
               Y
X1   0.200038965
X2   0.071133296
X3   1.136588639
X4   0.143860750
X5  -0.007093613
X6  -0.012072718
X7   0.467367137
X8   0.038908399
X9   0.375570507
X10  1.193617024
X11  0.234163484
X12  1.001783923
X13  0.007931389
X14  0.647031173
X15 -0.075491434
X16  0.804560392
X17  1.021205650
X18  0.130072384
X19 -0.274075560
X20  0.401602523
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(nesterov_bls_1$obj_values)
```

```
 [1] 0.5908489 0.5196567 0.4652743 0.4268089 0.4010798 0.3845780 0.3743313
 [8] 0.3680466 0.3640549 0.3612832 0.3591891 0.3575811 0.3563884 0.3555130
[15] 0.3548249 0.3542327 0.3537274 0.3533532 0.3531446 0.3530852 0.3530352
[22] 0.3529523 0.3528559 0.3527626 0.3526831 0.3526222 0.3525804 0.3525556
```

```
[29] 0.3525445 0.3525421 0.3525379 0.3525326 0.3525269 0.3525216 0.3525171
[36] 0.3525135 0.3525110 0.3525093 0.3525083 0.3525078 0.3525075 0.3525073
[43] 0.3525069 0.3525064 0.3525058 0.3525052 0.3525046 0.3525041 0.3525037
[50] 0.3525033
 [ reached 'max' / getOption("max.print") -- omitted 70 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(nesterov_bls_1$eta_values)
```

```
 [1] 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
[13] 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 0.500 0.500 0.500 0.500
[25] 0.500 0.500 0.500 0.500 0.500 0.250 0.250 0.250 0.250 0.250 0.250 0.250
[37] 0.250 0.250 0.250 0.250 0.250 0.125 0.125 0.125 0.125 0.125 0.125 0.125
[49] 0.125 0.125
 [ reached 'max' / getOption("max.print") -- omitted 70 entries ]
```

```r
cat("The functions stopped after", max(which(!is.na(nesterov_bls_1
↪  $eta_values))), "iterations \n \n")
```

```
The functions stopped after 120 iterations
```

```r
nesterov_bls_2 <- nesterov_bls(X_dataset_2, y_dataset_2, eta = NULL, tol =
↪  1e-6, max_iter = 10000, backtracking=TRUE, epsilon = 0.5, tau = 0.5)

cat("Nesterov BLS Step Size: dataset2 \n")
```

```
Nesterov BLS Step Size: dataset2
```

```r
print("Beta Values:")
```

```
[1] "Beta Values:"
```

```r
print(nesterov_bls_2$beta)
```

```
              Y
X1   0.24045357
X2   0.29604289
X3   0.53173674
X4   0.20051983
X5   0.46598103
X6   0.17742248
X7   0.33411178
X8  -0.01754793
X9   0.16537421
X10  0.73086234
X11  0.22788954
X12  0.35957750
X13  0.18297917
X14  0.29445766
X15 -0.13735573
X16  0.31831937
X17  0.43704120
X18  0.46467066
X19  0.23602866
X20  0.22298351
```

```r
print("Obj Function Values:")
```

```
[1] "Obj Function Values:"
```

```r
print(nesterov_bls_2$obj_values)
```

```
 [1] 7.2567507 2.3148022 1.6473834 1.2427905 0.7728966 0.5575479 0.5185890
 [8] 0.4798250 0.4568271 0.4431131 0.4296629 0.4148278 0.4013438 0.3910625
[15] 0.3838092 0.3787196 0.3750492 0.3721992 0.3697963 0.3677688 0.3660956
[22] 0.3645643 0.3629496 0.3613093 0.3598838 0.3587377 0.3577078 0.3567351
[29] 0.3560556 0.3559290 0.3558670 0.3557736 0.3556848 0.3556240 0.3555950
[36] 0.3555861 0.3555770 0.3555615 0.3555429 0.3555246 0.3555085 0.3554954
[43] 0.3554846 0.3554748 0.3554646 0.3554530 0.3554397 0.3554249 0.3554094
[50] 0.3553938
 [ reached 'max' / getOption("max.print") -- omitted 439 entries ]
```

```r
print("Eta Values:")
```

```
[1] "Eta Values:"
```

```r
print(nesterov_bls_2$eta_values)
```

```
 [1] 0.015625000 0.015625000 0.015625000 0.015625000 0.015625000 0.015625000
 [7] 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500
[13] 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500
[19] 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500
[25] 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500 0.007812500
[31] 0.003906250 0.003906250 0.003906250 0.003906250 0.003906250 0.003906250
[37] 0.001953125 0.001953125 0.001953125 0.001953125 0.001953125 0.001953125
[43] 0.001953125 0.001953125 0.001953125 0.001953125 0.001953125 0.001953125
[49] 0.001953125 0.001953125
 [ reached 'max' / getOption("max.print") -- omitted 439 entries ]
```

```r
cat("The functions stopped after",
 ↪  max(which(!is.na(nesterov_bls_2$eta_values))), "iterations \n \n")
```

```
The functions stopped after 489 iterations
```

1) Compare again the estimates obtained from the two momentum methods with the least-squares solution by calculating ˆ GD — ˆ 2

Calculate the difference of Betas:

```r
# transform dataset 1 coefs
beta_gd_d1_p_const <- as.vector(polyak_reg_constant_1$beta)
p1 <- data.frame(Coef_data1_polyak_constant = names, Beta_GD =
 ↪  beta_gd_d1_p_const, Beta = lm_1)

beta_gd_d1_n_const <- as.vector(nesterov_reg_constant_1$beta)
n1 <- data.frame(Coef_data1_nesterov_constant = names, Beta_GD =
 ↪  beta_gd_d1_n_const, Beta = lm_1)

# transform dataset 2 coefs
```

```r
beta_gd_d2_p_const <- as.vector(polyak_reg_constant_2$beta)
p2 <- data.frame(Coef_data2_polyak_constant = names, Beta_GD =
↪  beta_gd_d2_p_const, Beta = lm_2)

beta_gd_d2_n_const <- as.vector(nesterov_reg_constant_2$beta)
n2 <- data.frame(Coef_data1_nesterov_constant = names, Beta_GD =
↪  beta_gd_d2_n_const, Beta = lm_2)

# show the difference between Beta_GD and Beta
p1 <- p1 %>%
  mutate(`Beta_GD - Beta` = Beta_GD - Beta)

p2 <- p2 %>%
  mutate(`Beta_GD - Beta` = Beta_GD - Beta)

n1 <- n1 %>%
  mutate(`Beta_GD - Beta` = Beta_GD - Beta)

n2 <- n2 %>%
  mutate(`Beta_GD - Beta` = Beta_GD - Beta)


print(p1)
```

```
   Coef_data1_polyak_constant      Beta_GD         Beta Beta_GD - Beta
1                          X1  0.200283811  0.117292007    0.082991805
2                          X2  0.071173054  0.092144880   -0.020971826
3                          X3  1.136551334  1.139351598   -0.002800265
4                          X4  0.143784177  0.153116480   -0.009332304
5                          X5 -0.007078068  0.058436652   -0.065514721
6                          X6 -0.012022932 -0.105234557    0.093211625
7                          X7  0.467645816  0.474468590   -0.006822774
8                          X8  0.038900742 -0.008261791    0.047162534
9                          X9  0.375595445  0.391447179   -0.015851734
10                        X10  1.193206312  1.220257571   -0.027051259
11                        X11  0.234085961  0.225458882    0.008627079
12                        X12  1.001887132  1.061147318   -0.059260186
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

```r
print(p2)
```

```
   Coef_data2_polyak_constant     Beta_GD       Beta Beta_GD - Beta
1                          X1  0.24044664  0.2373680    0.003078636
2                          X2  0.29601821  0.3004188   -0.004400568
3                          X3  0.53165797  0.5437310   -0.012073007
4                          X4  0.20049437  0.2102227   -0.009728349
5                          X5  0.46596432  0.4734042   -0.007439873
6                          X6  0.17746121  0.1703577    0.007103499
7                          X7  0.33416170  0.3258614    0.008300347
8                          X8 -0.01752115 -0.0269920    0.009470855
9                          X9  0.16536857  0.1683506   -0.002982059
10                        X10  0.73075205  0.7513985   -0.020646494
11                        X11  0.22791901  0.2244565    0.003462553
12                        X12  0.35962283  0.3547595    0.004863311
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

```r
print(n1)
```

```
   Coef_data1_nesterov_constant      Beta_GD         Beta Beta_GD - Beta
1                            X1  0.200282647  0.117292007    0.082990640
2                            X2  0.071172727  0.092144880   -0.020972152
3                            X3  1.136552000  1.139351598   -0.002799598
4                            X4  0.143784961  0.153116480   -0.009331520
5                            X5 -0.007078179  0.058436652   -0.065514831
6                            X6 -0.012023914 -0.105234557    0.093210643
7                            X7  0.467644858  0.474468590   -0.006823732
8                            X8  0.038900556 -0.008261791    0.047162347
9                            X9  0.375594538  0.391447179   -0.015852640
10                          X10  1.193208086  1.220257571   -0.027049485
11                          X11  0.234086367  0.225458882    0.008627485
12                          X12  1.001886829  1.061147318   -0.059260489
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

```r
print(n2)
```

```
   Coef_data1_nesterov_constant     Beta_GD       Beta Beta_GD - Beta
1                            X1  0.24045810  0.2373680    0.003090099
2                            X2  0.29607939  0.3004188   -0.004339386
3                            X3  0.53177889  0.5437310   -0.011952086
4                            X4  0.20063911  0.2102227   -0.009583605
5                            X5  0.46597238  0.4734042   -0.007431820
```

```
6                           X6  0.17742939  0.1703577    0.007071682
7                           X7  0.33389131  0.3258614    0.008029950
8                           X8 -0.01759342 -0.0269920    0.009398588
9                           X9  0.16542136  0.1683506   -0.002929265
10                         X10  0.73112261  0.7513985   -0.020275931
11                         X11  0.22789784  0.2244565    0.003441388
12                         X12  0.35941098  0.3547595    0.004651461
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

Calculating the norms:

```r
p1_norm <- sqrt(sum(p1$`Beta_GD - Beta`^2))
p2_norm <- sqrt(sum(p2$`Beta_GD - Beta`^2))
n1_norm <- sqrt(sum(n1$`Beta_GD - Beta`^2))
n2_norm <- sqrt(sum(n2$`Beta_GD - Beta`^2))
cat("Dataset 1 Normm Polyak:", p1_norm, "\n")
```

```
Dataset 1 Normm Polyak: 0.195615
```

```r
cat("Dataset 2 Norm Polyak:", p2_norm, "\n")
```

```
Dataset 2 Norm Polyak: 0.05539779
```

```r
cat("Dataset 1 Norm Nesterov:", n1_norm, "\n")
```

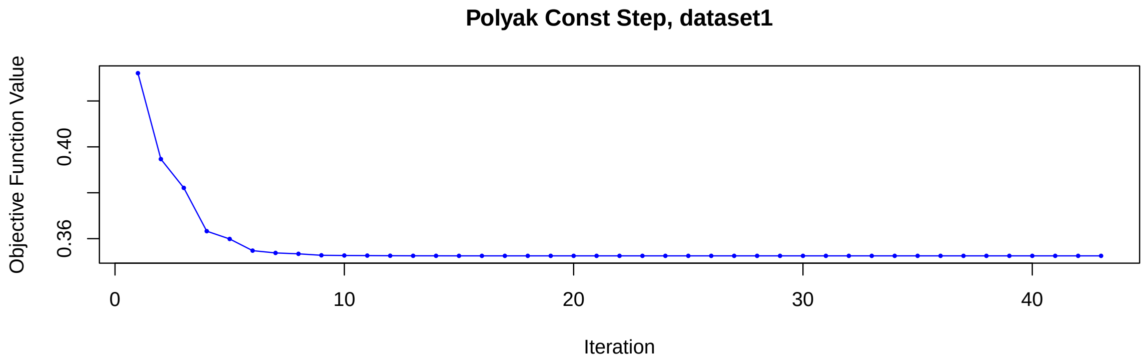
```
Dataset 1 Norm Nesterov: 0.1956129
```

```r
cat("Dataset 2 Norm Nesterov:", n2_norm, "\n")
```

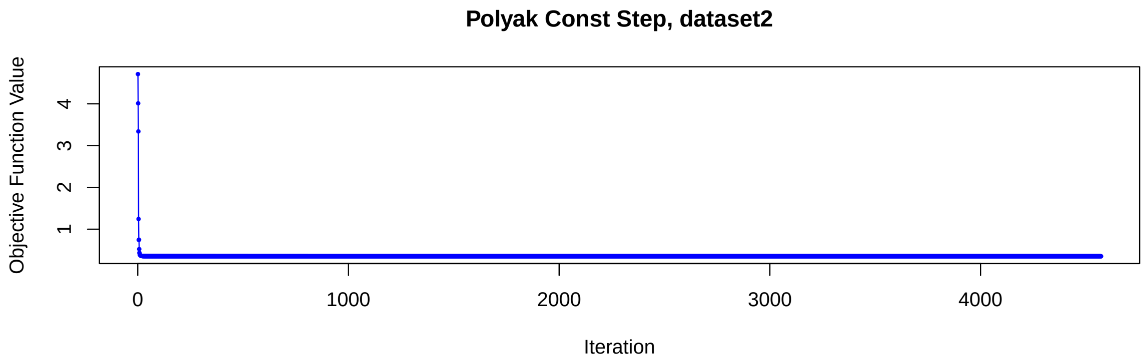
```
Dataset 2 Norm Nesterov: 0.05456602
```

2) Plot the value of the objective function as a function of the number of iterations required, and comment whether the momentum methods reduce the number of iterations requires to obtain the regression coeffcients (using the same tolerance)

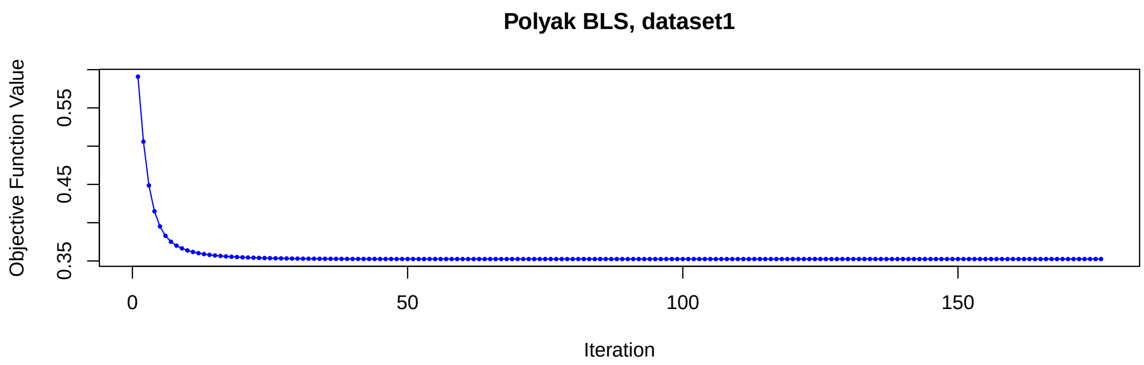
```r
plot(polyak_reg_constant_1$obj_values, type = "o", col = "blue", pch = 16,
↪  cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Polyak Const Step, dataset1")
```



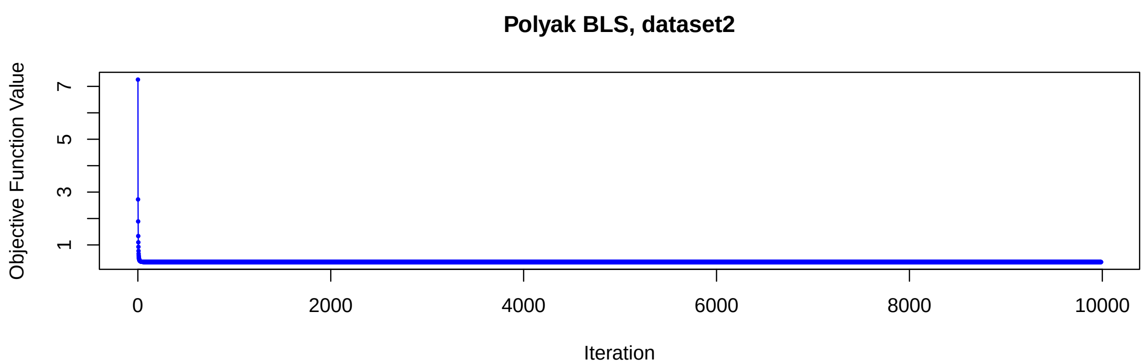

```r
plot(polyak_reg_constant_2$obj_values, type = "o", col = "blue", pch = 16,
↪  cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Polyak Const Step, dataset2")
```



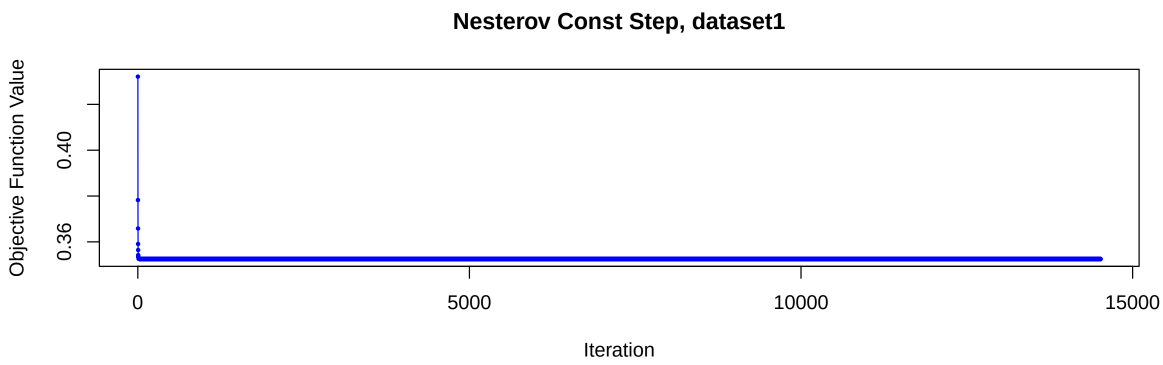

```r
plot(polyak_bls_1$obj_values, type = "o", col = "blue", pch = 16, cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Polyak BLS, dataset1")
```



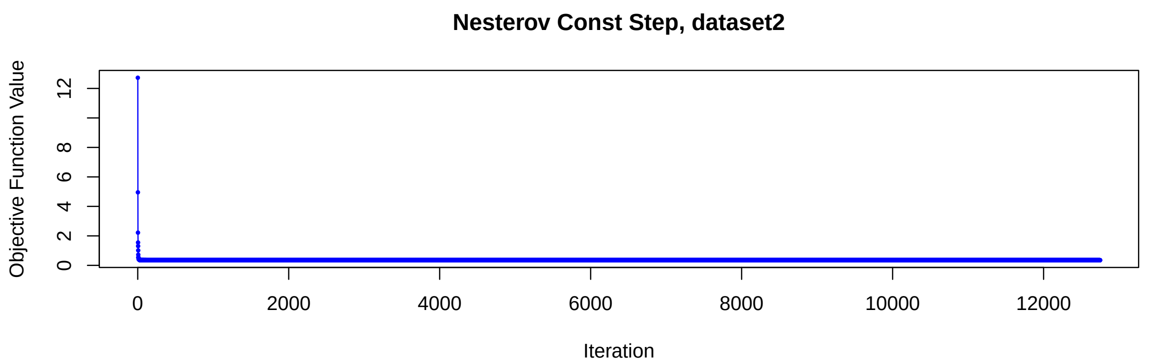

```r
plot(polyak_bls_2$obj_values, type = "o", col = "blue", pch = 16, cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Polyak BLS, dataset2")
```



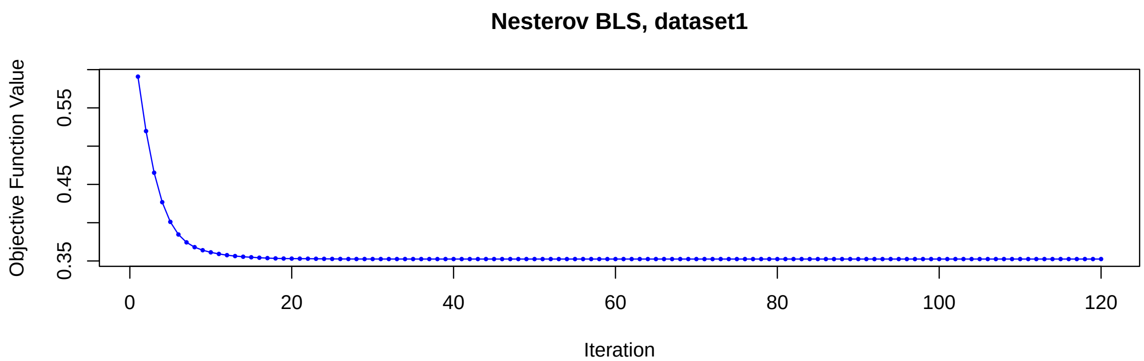

```r
plot(nesterov_reg_constant_1$obj_values, type = "o", col = "blue", pch = 16,
↪  cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Nesterov Const Step, dataset1")
```



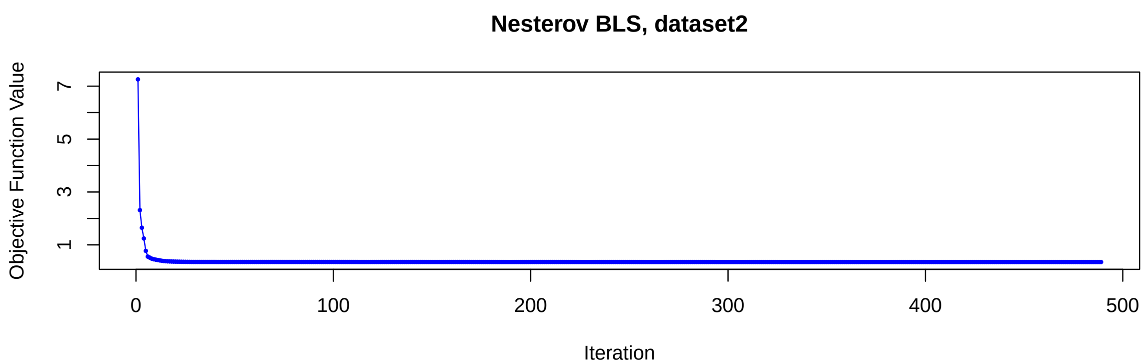

```
plot(nesterov_reg_constant_2$obj_values, type = "o", col = "blue", pch = 16,
↪  cex = 0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Nesterov Const Step, dataset2")
```




```
plot(nesterov_bls_1$obj_values, type = "o", col = "blue", pch = 16, cex =
↪  0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Nesterov BLS, dataset1")
```




```r
plot(nesterov_bls_2$obj_values, type = "o", col = "blue", pch = 16, cex =
↪  0.5,
     xlab = "Iteration", ylab = "Objective Function Value",
     main = "Nesterov BLS, dataset2")
```



We can see for Polyak and Nesterov, both of the BLS methods did not necessarily converge slower. This was likely because we were able to manually select a more agressive step size that what is calculated with BLS, or in other case BLS chose a better step size with the momentum shift. We see that Nesterov took way more iterations with constant step than polyak constant step. We see that Nesterov BLS is much better than Nesterov at a constant step size. Finally, we see that Polyak BLS took more iterations, likely because the step size was safer tha with constant step.

3) Comment on the results; namely, the difference in the accuracy of the solution and the standard gradient descent algorithm

a Looking at the accuacy of the solution, we see that the accuracy norms for the standard gradient descent is 0.195615 and 0.06. For polyak we got .195615 and 0.05539779, and for Nesterov we got .1956129 and 0.05456602. This might indicate that the norms are very close together, and since we used the same tolerance we expect all gradient descent algorithms to be within that solerance. Therefore the algorithms converged to the same answer.