27765369 Assignment 2

Ian Tongs

18/09/2020

Import some libraries:

```
library(dplyr)
library(readr)
library(tidyverse)
library(ggplot2)
library(gridExtra)
library(grid)
library(cowplot)
```

I don't actually use these (I think) but they're here just in case. Results on the next pages.

Part A

Functions for Input Error and Gradient for Linear Regression and Logistic Regression

In - Sample Error Calculations:

```
#################################################################################################
# Compute In Sample Error - Linreg:
#################################################################################################

Ein_linreg <- function(X, y, w){
  # Calculates the Input Error for linear regression with a vector w of weights
  # and X and y inputs
  n <- nrow(y)
  Ein_raw <- 0

  # Main loop for calculating error:
  for (i in 1:n) {
    Ein_raw <- Ein_raw + (t(w) %*% X[i, ] - y[i])^2
  }
  Ein <- (1/n)*Ein_raw

  return(Ein)
}


#################################################################################################
# Compute In Sample Error - Logreg:
#################################################################################################

Ein_logreg <- function(X, y, w){
  # Calculates the Input Error for linear regression with a vector w of weights
  # and X and y inputs
  n <- nrow(y)
  Ein_raw <- 0

  # Main loop for calculating error:
  for (i in 1:n) {
    Ein_raw <- Ein_raw + log(1 + exp(- y[i]*(t(w) %*% X[i, ])))
  }
  Ein <- (1/n)*Ein_raw

  return(Ein)
}
```

Gradient Functions:

```
#################################################################################################
# Compute In Gradient - Linreg:
#################################################################################################
```

```r
gEin_linreg <- function(X, y, w){
  # Computes the (linear) gradient for a set of weights and an input X and y matrices
  n <- nrow(y)
  grad_raw <- 0

  # Main loop for calculating gradient:
  for (i in 1:n) {
    grad_raw <- grad_raw + (t(w) %*% X[i, ] - y[i])*X[i, ]
  }
  grad <- (2/n)*grad_raw

  return(grad)
}


################################################################################################
# Compute In Gradient - Logreg:
################################################################################################

gEin_logreg <- function(X, y, w){
  # Computes the (logistic) gradient for a set of weights and an input X and y matrices
  n <- nrow(y)
  grad_raw <- 0

  # Main loop for calculating gradient:
  for (i in 1:n) {
    grad_raw <- grad_raw + (y[i]*X[i, ])/(1 + exp(y[i]*(t(w) %*% X[i, ])))
  }
  grad <- (-1/n)*grad_raw

  return(grad)
}
```

Part B

Algorithm for Gradient Descent (GD)

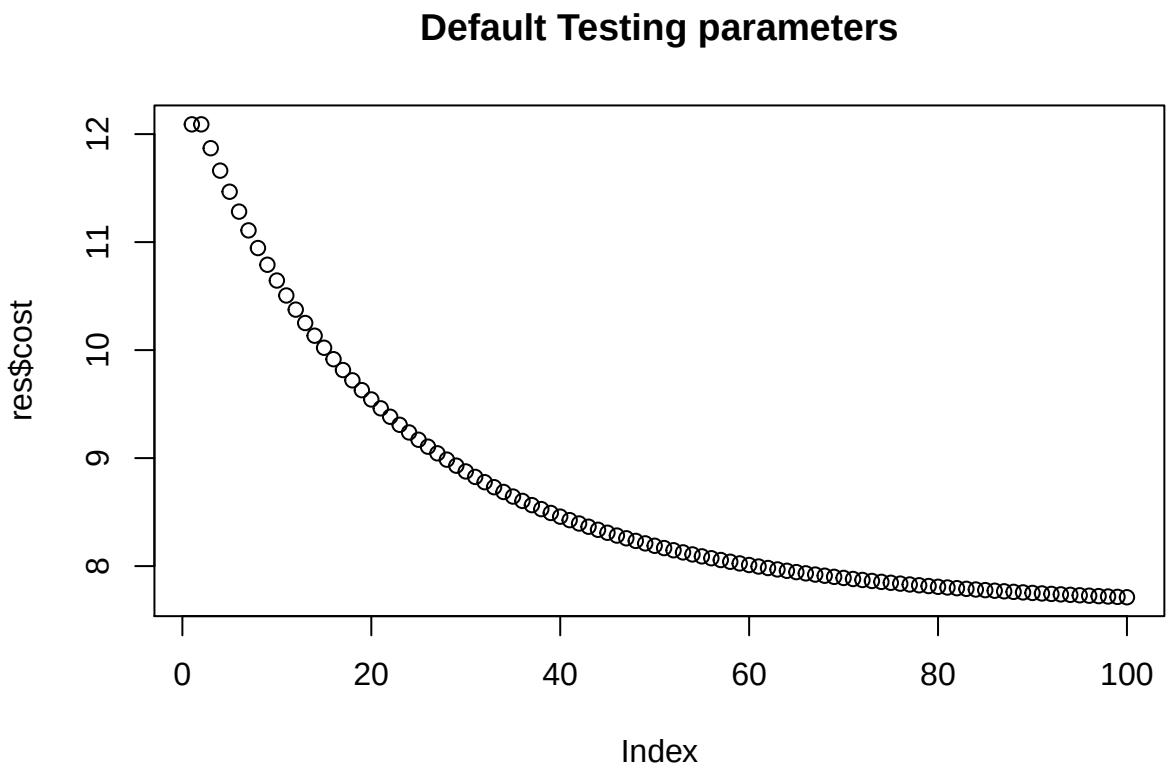
```
GD <- function(X, y, Ein, gEin, w0, eta, precision, nb_iters){
  # Initial values as defined by the question:
  allw <- vector("list", nb_iters)
  cost <- numeric(nb_iters)
  allw[[1]] <- w0
  cost[1] <- Ein(X, y, allw[[1]])

  # Termination Variables:
  i <- 2
  Ein_old <- 0
  Ein_cur <- cost[1]
  Ein_diff <- abs(Ein_cur- Ein_old)

  # Main Loop - while less than the number of iterations and while precision above target:
  while (Ein_diff >= precision && i <= nb_iters) {

    # Add new results to results lists
    allw[[i]] <- allw[[i-1]] - eta * gEin(X, y, allw[[i-1]])
    cost[i] <- Ein(X, y, allw[[i-1]])

    # Update termination conditions
    Ein_old <- Ein_cur
    Ein_cur <- Ein(X, y, allw[[i]])
    Ein_diff <- abs(Ein_cur- Ein_old)
    i <- i + 1
  }

  # For making sure 'bottom row' is correct if terminated before max iterations
  # Done because of the nature of the testing we are required to do
  # Technically defeats the point of an early termination condition though
  while (i <= nb_iters) {

    # Add new results to results lists
    allw[[i]] <- allw[[i-1]]
    cost[i] <- cost[i-1]

    # Update termination conditions
    i <- i + 1
  }

  # Return the required values as a list, as required:
  list(allw = allw, cost = cost)
}
```

Part C

Testing Linear Regression using Gradient Descent

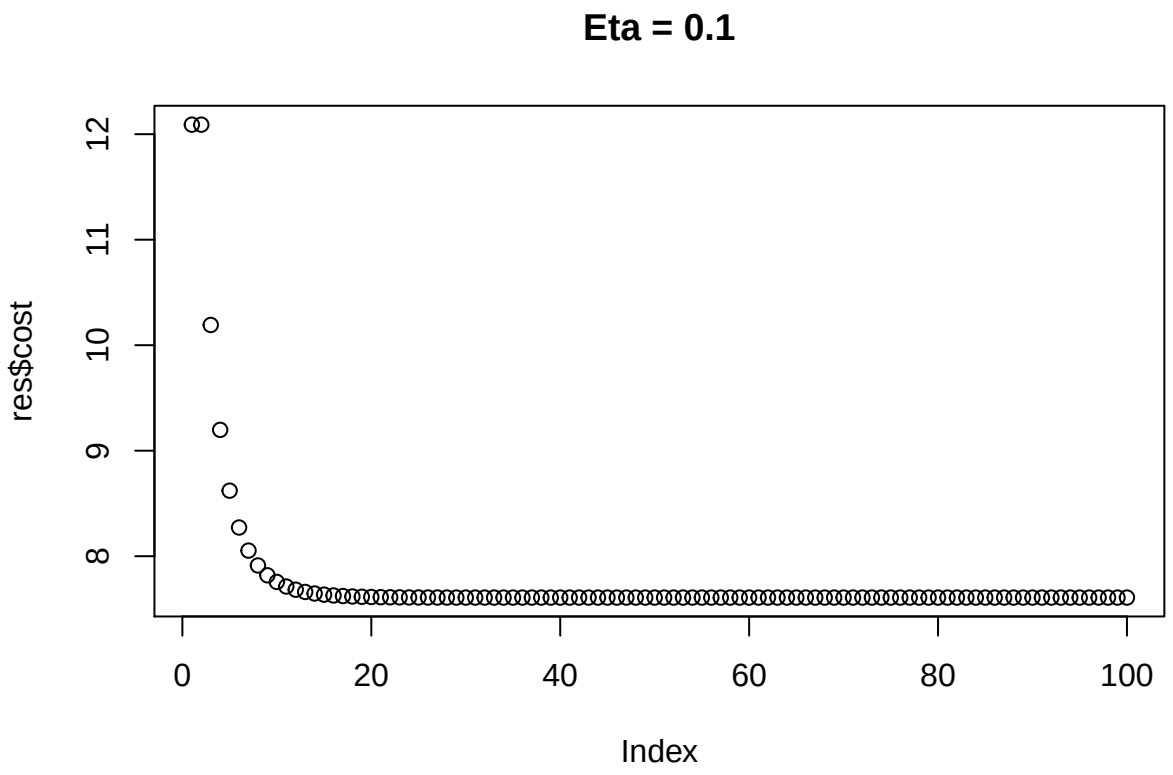
Using the default tesing code from the instructions, we get:

```
set.seed(1900)
# Function taken from Friedman et al.
genx <- function(n,p,rho){
  # generate x's multivariate normal with equal corr rho
  # Xi = b Z + Wi, and Z, Wi are independent normal.
  # Then Var(Xi) = b^2 + 1
  # Cov(Xi, Xj) = b^2 and so cor(Xi, Xj) = b^2 / (1+b^2) = rho
  z <- rnorm(n)
  if(abs(rho) < 1){
    beta <- sqrt(rho/(1-rho))
    x <- matrix(rnorm(n*p), ncol=p)
    A <- matrix(rnorm(n), nrow=n, ncol=p, byrow=F)
    x <- beta * A + x
  }
  if(abs(rho)==1){ x=matrix(rnorm(n),nrow=n,ncol=p,byrow=F)}
  return(x)
}


N <- 100
p <- 10
rho <- 0.2
X <- genx(N, p, rho)
w_true <- ((-1)^(1:p))*exp(-2*((1:p)-1)/20)
eps <- rnorm(N)
k <- 3
y <- X %*% w_true + k * eps


res <- GD(X, y, Ein_linreg, gEin_linreg, rep(0, p), 0.01, 0.0001, 100)
plot(res$cost, main="Default Testing parameters")
```



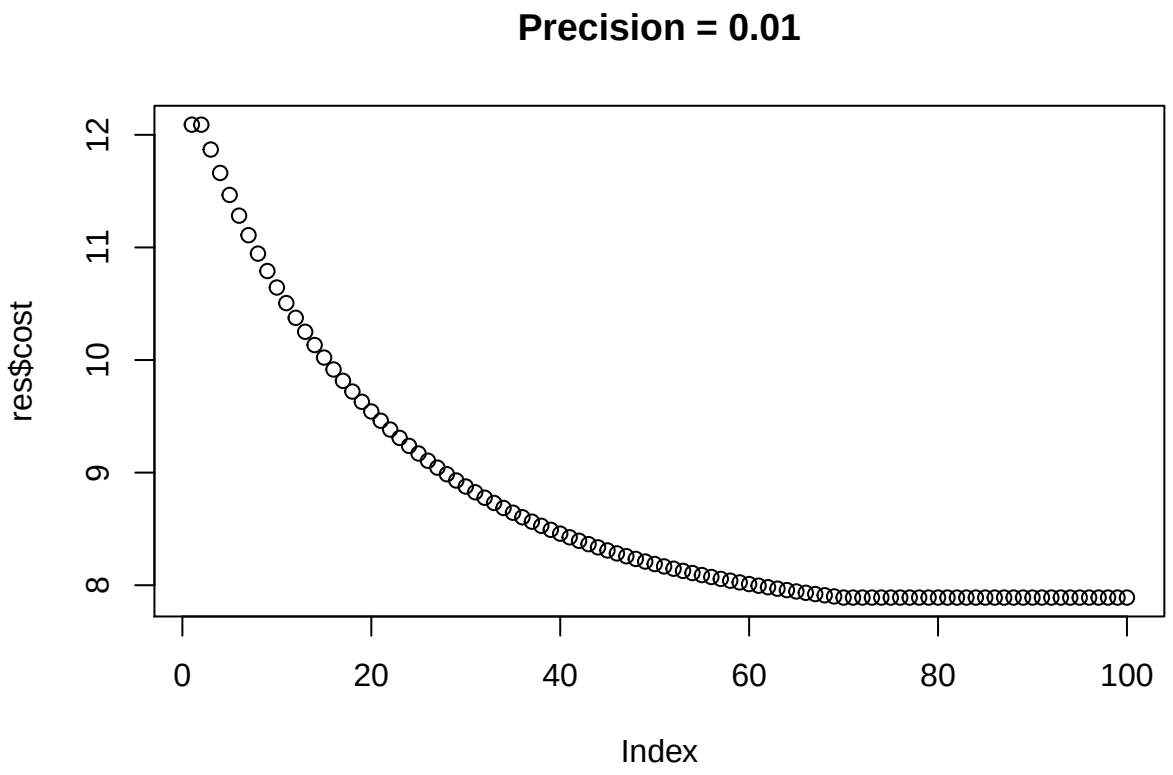
Varying the trial parameters, we can also find that (on the following pages):

Increased Eta:



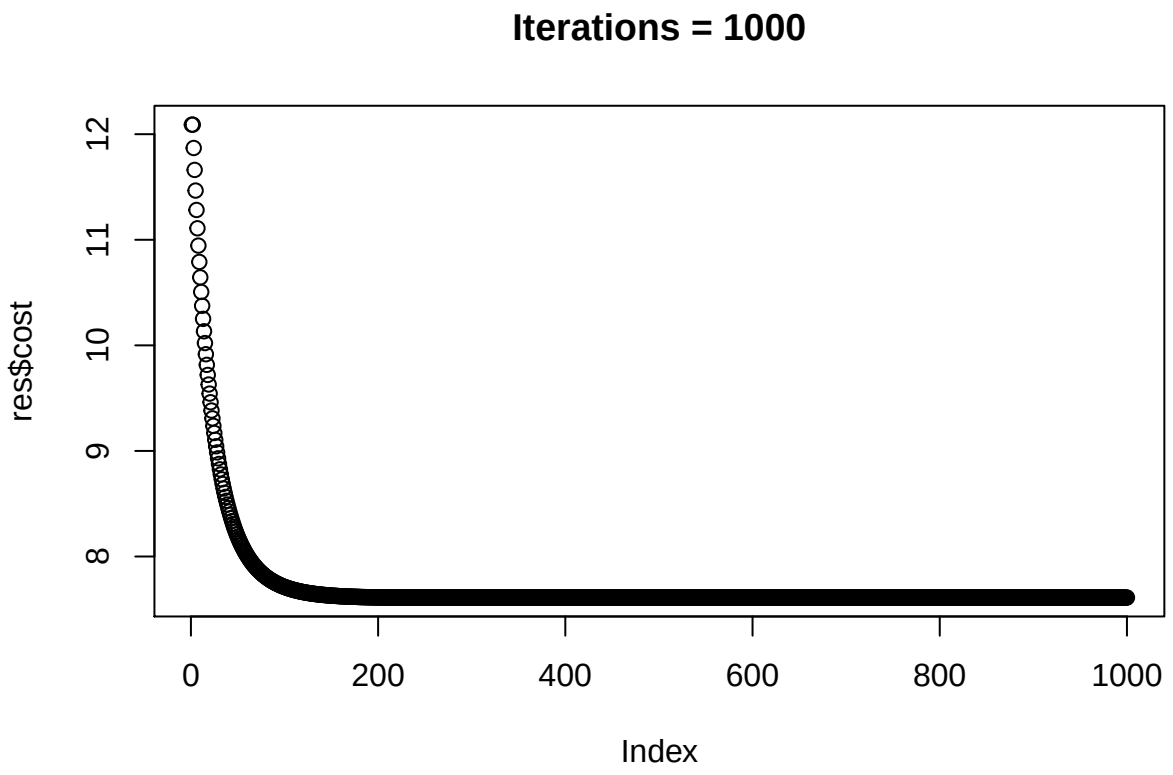
We may observe that the cost falls faster and reaches a lower final value before the function terminates.

Increased Precision:



We may observe that the cost falls more slowly to a higher cost value. Furthermore, the flatlining suggests that the function terminated early - which one would expect with a higher precision value.

Increased Iterations:



We may observe that the cost falls faster with a lower final cost value. There is fairly minimal change after about 200 iterations and it is likely that the algorithm terminated early.

Overall, the initial parameters appear to be quite good so we sill stick with them, especially with no easy way of optimising over so many parameters. Let us compare the closed-form solutions:

```
## [1] "True values of w:"
```

```
##  [1] -1.0000000  0.9048374 -0.8187308  0.7408182 -0.6703200  0.6065307
##  [7] -0.5488116  0.4965853 -0.4493290  0.4065697
```

```
## [1] "Gradient Descent value of w:"
```

```
##  [1] -0.8947185  0.7261167 -0.4558094  0.2108312 -0.6120182  0.3818576
##  [7] -0.5216388  0.6724877 -0.4411060  0.1635883
```

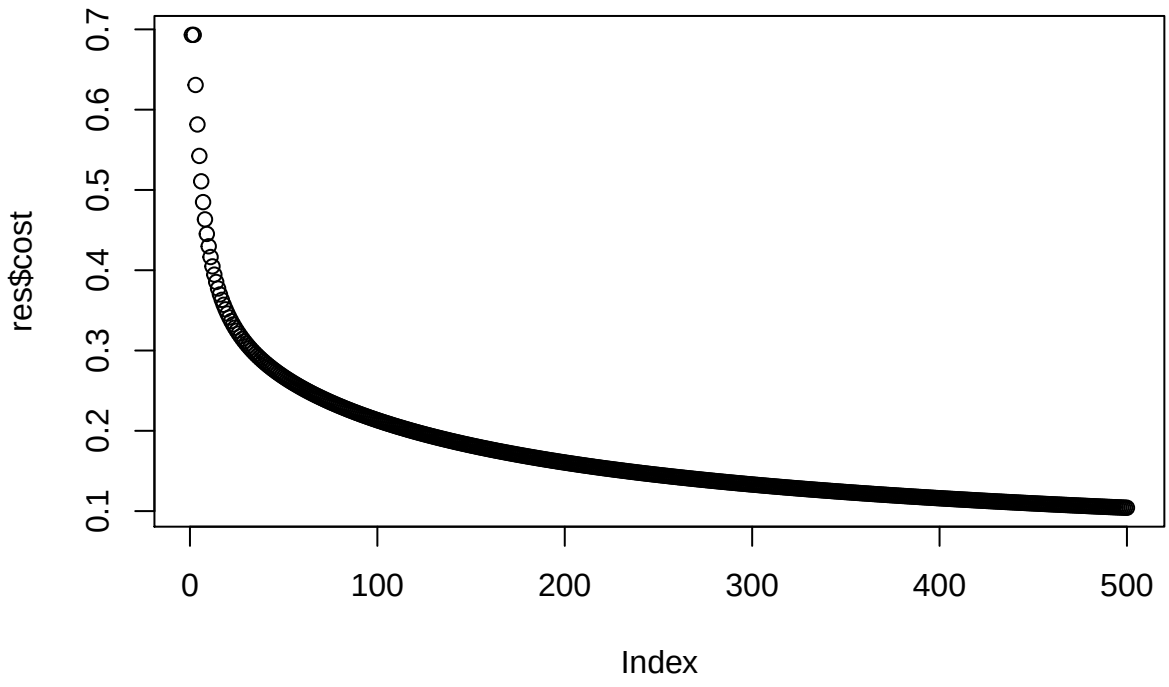
We may observe that the values for w found via gradient descent are generally quite close to the true values. Given our sample size is only of a size $n = 100$, some degree of inaccuracy is to be expected, especially across a 10 dimenaional space.

Part D

Testing Logistic Regression using Gradient Descent

We can run the code on as required in the assignment question:

```
set.seed(1900)
N <- 100
l <- -5; u <- 5
x <- seq(l, u, by = 0.1)
w_true <- matrix(c(-3, 1, 1), ncol = 1)
a <- -w_true[2]/w_true[3]
b <- -w_true[1]/w_true[3]

X0 <- matrix(runif(2 * N, l, u), ncol = 2)

X <- cbind(1, X0)
y <- sign(X %*% w_true)

res <- GD(X, y, Ein_logreg, gEin_logreg, rep(0, 3), 0.05, 0.0001, 500)
plot(res$cost)
```




```
print(w_true)
```

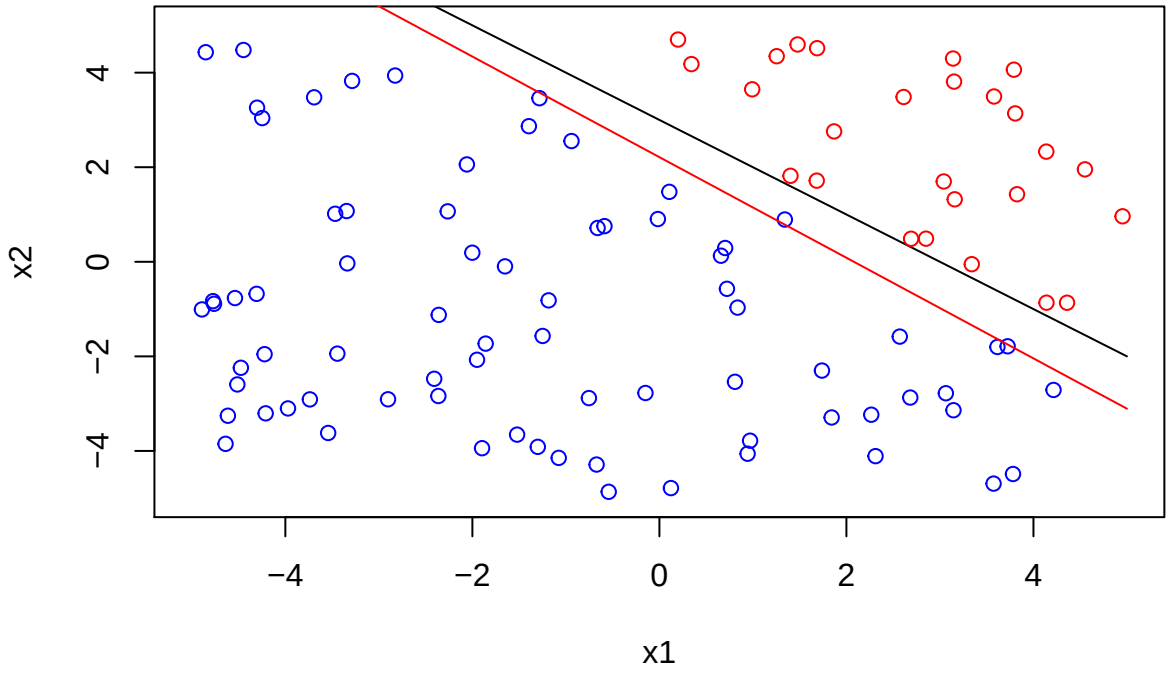
```
##      [,1]
## [1,]   -3
## [2,]    1
```

```
## [3,]    1
```

```
w_best <- unlist(tail(res$allw, 1))
print(w_best)
```

```
## [1] -2.0843183  1.0020933  0.9419666
```

```
plot(c(l, u), c(u, l), type = 'n', xlab = "x1", ylab = "x2")
lines(x, a*x +b)
points(X0, col = ifelse(y == 1, "red", "blue"))

a_best <- -w_best[2]/w_best[3]
b_best <- -w_best[1]/w_best[3]
lines(x, a_best*x + b_best, col = "red")
```



Testing completed as required. The gradient descent logistic regression identifies most values in the training data set correctly with the given parameters.