STAT 598Z HW 6

Linna Henry

April 7, 2016

1) a) Write a function gen data to generate a training dataset (X, Y). Your function should take in 4 arguments, n, p, sparsity and level. n is the number of observations, and p is their dimensionality, and generate X as an n × p matrix of mean-0, variance-1 Gaussian elements. The weight vector w is a p-dimensional vector, all of whose elements are 0 except the first sparsity elements, which all take value level. Generate the output vector Y as Y(i) = X(i)*w + *epsilon*(i) where X(i) is the ith input, and *epsilon*(i) is Gaussian noise. Do not use for loops.

```r
gen_data <- function(n,p, sparsity, level){
  xvec <- rnorm(n*p)
  x <- matrix(xvec, n, p)
  w <- rep(0,p)
  w[1:sparsity] <- level
  w <- t(w)
  w <- t(w)
  y <- x%*%w
  y <- y + rnorm(length(y))
  ret <- list(x,y)
  names(ret) <- c("x", "y")
  return(ret)
}
```

b) Write a function lasso loss that takes two inputs w and lambda and returns the values of the LASSO loss function for (X, Y). You can treat (X, Y) as additional inputs, or as global variables.

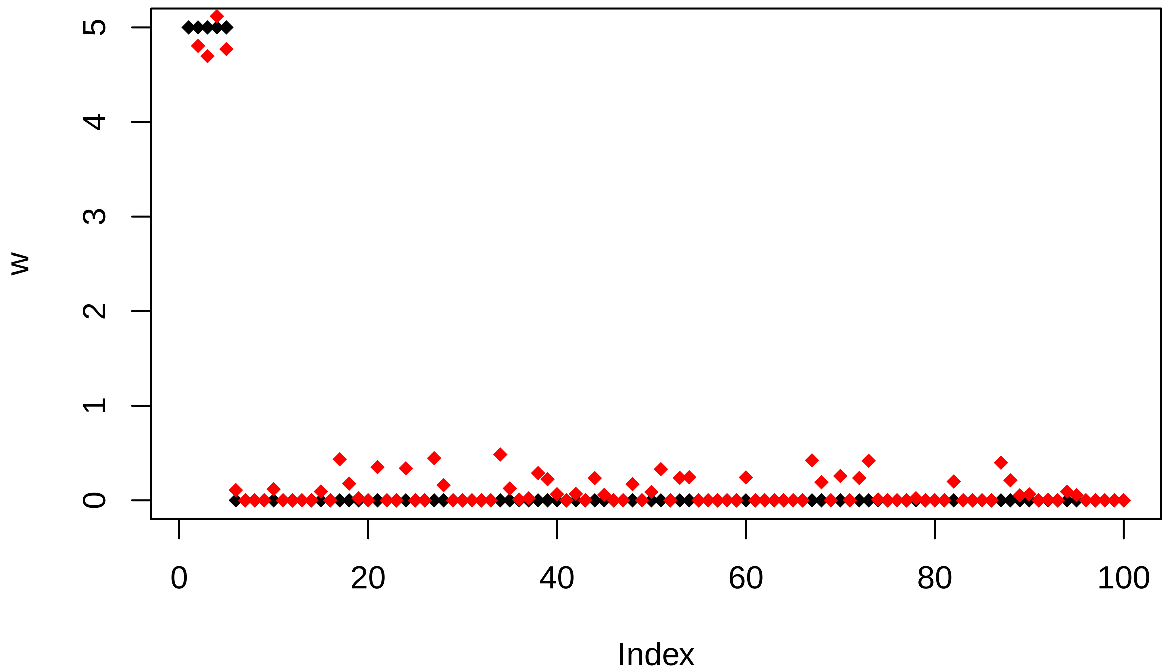
```r
lasso_loss <- function(w, lambda, x, y){
  w <- t(w)
  w <- t(w)
  lw <- sum((y - x%*%w)^2)
  omw <- sum(abs(w))
  lamomw <- lambda*omw
  return(lw + lamomw)
}
```

c) Generate a dataset with n=50, p = 100, sparsity=5, level=5.

```r
data <- gen_data(50, 100, 5, 5)
x <- data$x
y <- data$y
```

d) Use the optim function to find the best-values of w for the dataset above on the LASSO loss function. Set lambda=1. Plot the true w and the returned w.

```r
op<-optim(1:100, lasso_loss, lambda = 1, x=data$x, y=data$y,method = "L-BFGS-B", control=list(maxit=100
plot(c(rep(5,5),rep(0,95)), pch=18, ylab ="w")
points(op$par, col = 'red', pch = 18)
```

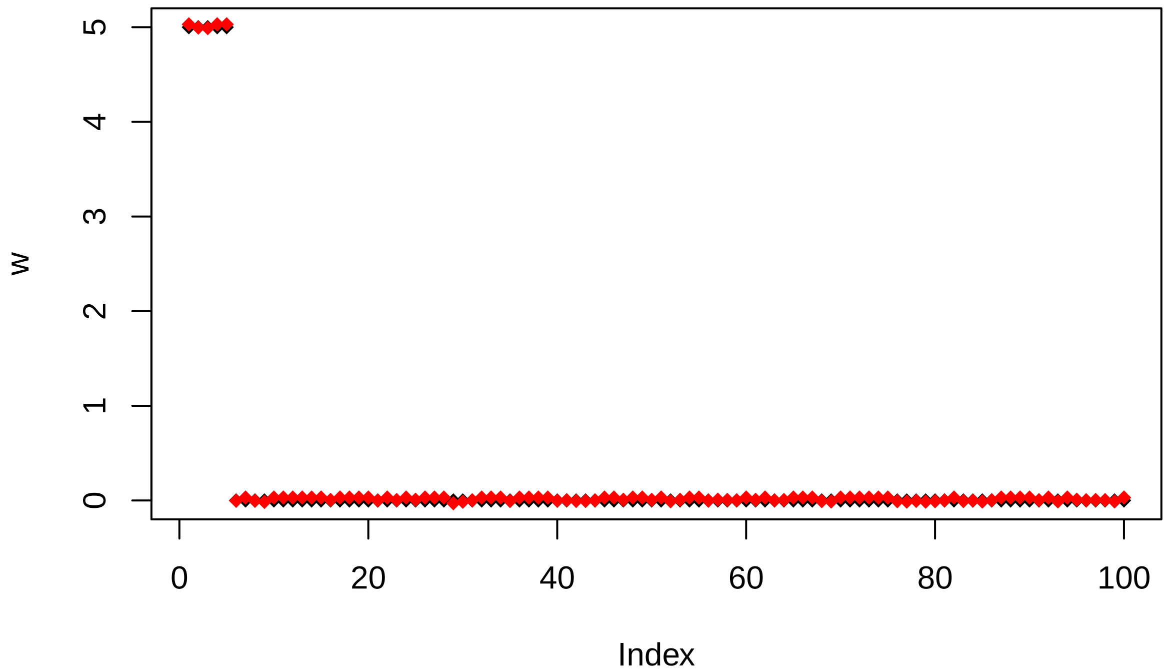


The black points are the true w and the red points are the w returned from optim.

e) Use the optim function to find the best-values of w and lambda for the dataset above on the LASSO loss function. Plot the true w and the returned w.

```
lasso_loss2 <- function(wlam, x, y){
  w <- wlam[1:100]
  lambda <- wlam[101]
  w <- t(w)
  w <- t(w)
  lw <- sum((y - x%*%w)^2)
  omw <- sum(abs(w))
  lamomw <- lambda*omw
  return(lw + lamomw)
}

wlament <- c(rep(5,5), rep(0,95))
wlament[101] <- 2
op2 <- optim(wlament, lasso_loss2, x=data$x, y=data$y, control=list(maxit=1000))
plot(c(rep(5,5),rep(0,95)), pch=18, ylab = "w")
points(op2$par[1:100], col = 'red', pch = 18)
```




```
op2$par[101]
```

```
## [1] 1.015432
```

The black points are the true w and the red points are the w returned by optim. This plot looks better in the sense that the red points are a lot closer to the black points.

2) a) First we'll solve the 1-d case. Write a function lasso1d that takes three inputs, length-n inputs x, y and lambda, and returns a scalar weight w by first calculating the OLS solution (correlation coefficient) and then soft-thresholding it. See the slides.

```
lasso1d <- function(lambda, xx, y){ #xx is 50x1
  wnum <- (t(xx)%*%y)
  wden <- (t(xx)%*%xx)
  w <- wnum/wden
  wlass <- sign(w)*(w - (lambda/wden))
  return(wlass)
}
```

b) Given a p-dimensional weight vector, write a function get residual to calculate the residual for some dimension dim. This function should take two inputs w and dim (and X,Y unless they are global), and return the residual error from trying to predict Y using all dimensions of X except dim. The simplest way to do this is to set w[dim] <- 0, and then calculate Y pred = X $\cdot$ w.

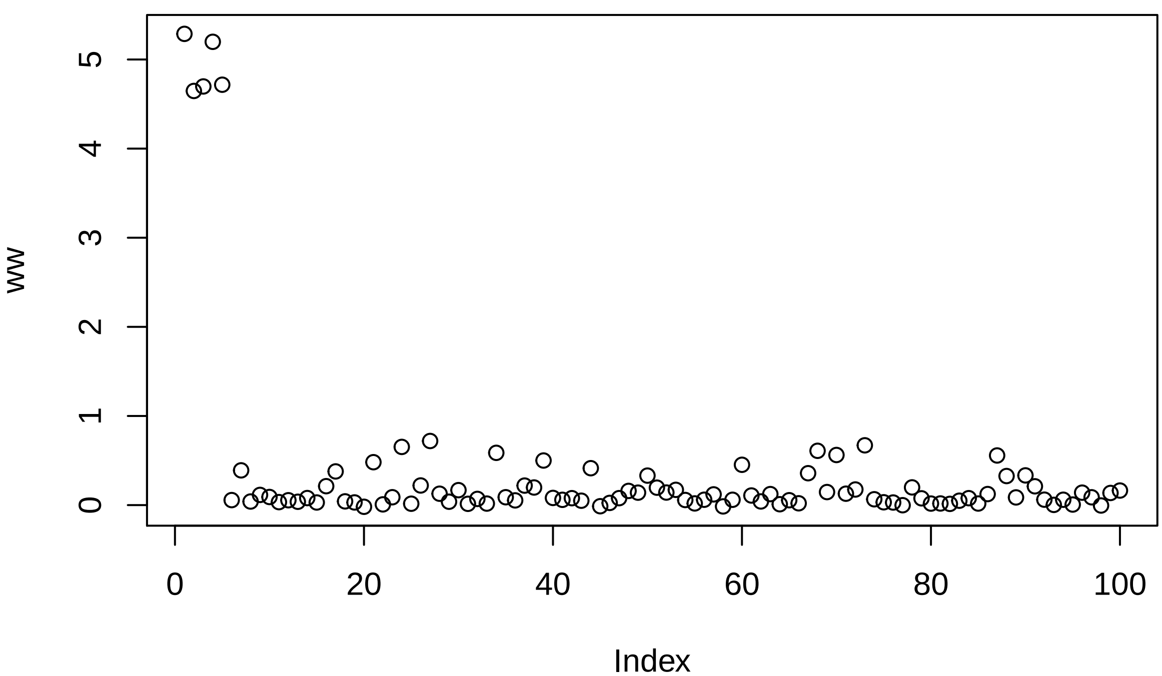
```
get_residual <- function(w, dim, x, y){
  w[dim]<-0
  y_pred = x%*%w
  return(y-y_pred)
}
```

c) Now we will solve for the p-dimensions w vector by coordinate descent. Initialize w to some value. Cycle through each dimension, first calculating its residual, and then updating the corresponding component of w. Repeat this until the change in w aftern an entire sweep is less than some threshold.

```
ww <- 1:100 #initial w
wwcurr <- rep(0,100)
threshold <- 0.05
iter <- 1
#while((sum(ww-wwcurr > threshold) > 0)){ #takes too long to converge
while(iter < 10000){ #5000
  #print(iter)
  wwcurr <- ww
  for(i in 1:100){
     res <- get_residual(ww, i, data$x, data$y)
     ww[i] <- lasso1d(1, data$x[,i], res)
  }
  iter <- iter + 1
}
#plot(c(rep(5,5),rep(0,95)), pch=18, ylab = "w")
#points(ww, pch = 18, color = 'red')
```

d)

```
plot(ww)
```

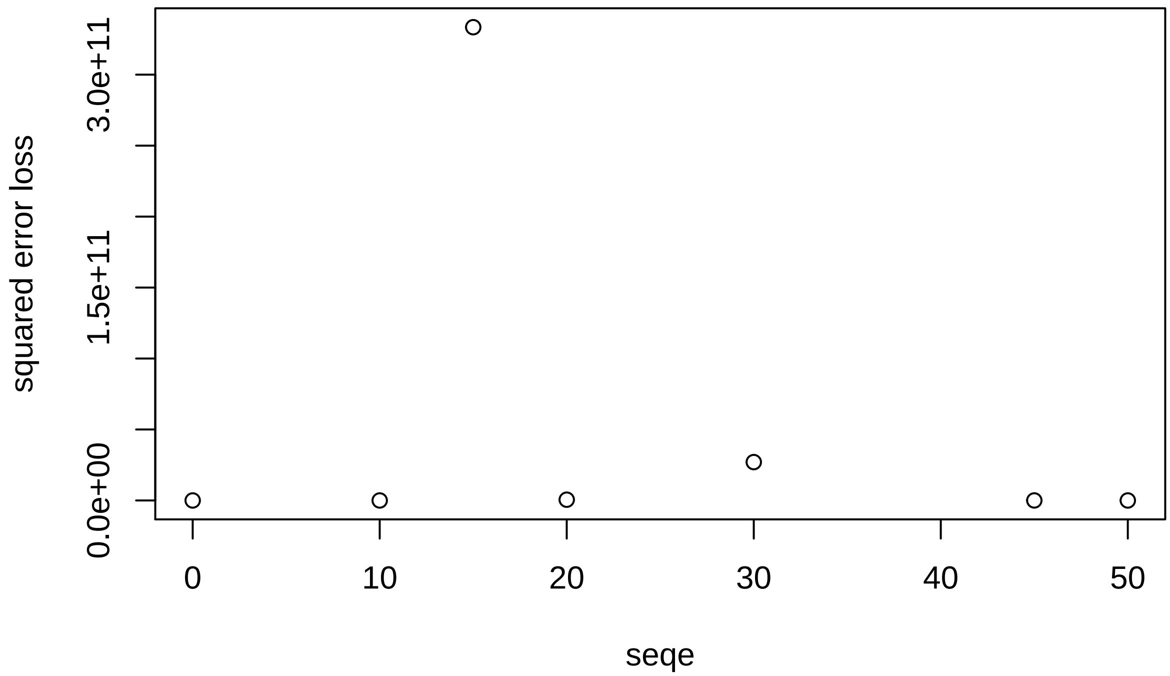


When I iterate for longer, the points become closer and closer to what they should be. I stopped at 10000 iterations, and I think the returned w got closer and closer to the real w each time. I was surprised at how close this algorithm got to returning the correct w. I think this way and optim return similar results.

e) Rerun your algorithm from the first n elements of X, where n varies from 0 to 50 in steps of 5. Plot the L2 error between the resuling w and the true w.

```r
seqe <- seq(5,50,5)
wmat <- matrix(NA, 100,10)
for(j in 1:length(seqe)){
  #print(j)
  datx <- data$x[1:seqe[j],]
  daty <- data$y[1:seqe[j]]
  ww <- 1:100 #initial w
  wwcurr <- rep(0,100)
  threshold <- 0.2
  iter <- 1
  #while(sum(ww-wwcurr > threshold) > 0 | (iter < 10000)){ #not working? idk
  while(iter < 5000){
    #print(iter)
    wwcurr <- ww
    for(i in 1:100){
      res <- get_residual(ww, i, datx, daty)
      ww[i] <- lasso1d(1, datx[,i], res)
    }
    iter <- iter + 1
  }
  wmat[,j] <- ww
}

#plot the l2 error between returned w and true w
truw <- c(rep(5,5), rep(0,95))
l2mat <- rep(NA, 10)
for(j in 1:10){
  l2mat[j] <- sum((truw-wmat[,j])^2)
}
seqe <- c(0, seqe)
l2mat <- c(sum(truw-rep(0,100)^2), l2mat)
plot(seqe, l2mat, ylab = "squared error loss")
```



For whatever reason, there is a huge error loss when n = 15. I am not sure why. The graph also seems to show some kind of trend around n=15 because n=10 and n=20 have higher L2 error loss than all the other n's.