6520 Project

Minjia Jia and Joia Zhang

Fall 2023

```
set.seed(6520)
```

Simulate data for regression and classification

```
# simulate data: regression
n = 100 # sample size
p = 200 # number of predictors

# beta
k = round(0.05*p, 0) # number of nonzero coefficients
sd_beta = 0.01
nonzero_indexes = sample.int(n=p, size=k)
beta = rep(0, p)
beta[nonzero_indexes] = rnorm(n=k, mean=100, sd=sd_beta)
sum(which(beta !=0) != sort(nonzero_indexes)) # test that we made the right indexes nonzero
```

```
## [1] 0
```

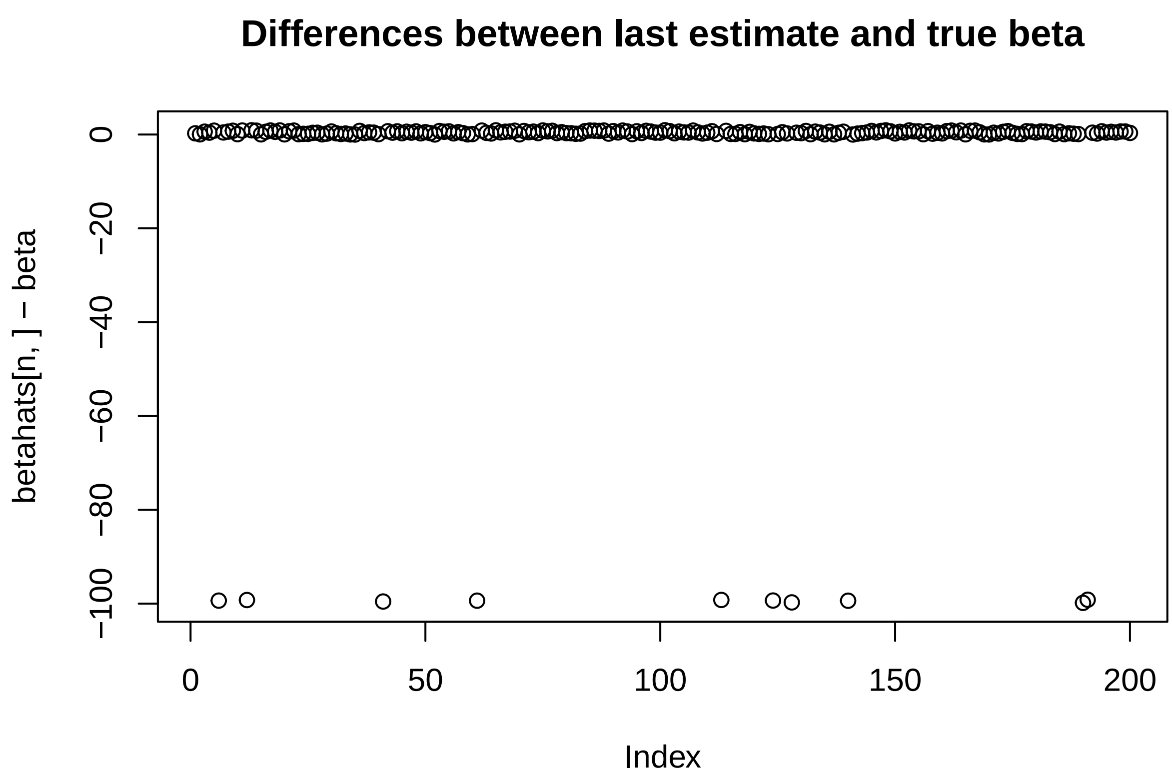
```
beta = as.matrix(beta)

# x
X = matrix(rnorm(n=n*p, mean=0, sd=5), nrow=n)

# epsilon
E = matrix(rnorm(n=n, mean=0, sd=1), nrow=n)

# y

Y = X%*%beta + E

# note that in the online setting, each t^th row of X and Y is for time t
```

```
# simulate data: classification
# X, beta same as above
probs = 1/(1+exp(-X%*%beta))
Y = rbinom(n=n, size=1, prob = probs) # Bernoulli
```

OGD

```r
# Online gradient descent for regression
my_OGD = function(X, Y, lr, beta_0) {
  n = nrow(X)
  p = ncol(X)
  betahats = matrix(nrow=n, ncol=p)
  betahats[1, ] = beta_0
  for (t in 1:(n-1)) {
    x_t = as.matrix(X[t, ])
    beta_t = as.matrix(betahats[t, ])
    y_t_hat = t(beta_t)%*%x_t
    Y_t = Y[t]
    # betahats[t+1, ] = beta_t - lr*as.numeric(y_t_hat-Y_t)*x_t # least means squares
    d_loss = 2*beta_t%*%t(x_t)%*%x_t - 2*x_t%*%Y_t
    betahats[t+1, ] = beta_t - lr*d_loss
  }
  return(betahats)
}
```

```r
# testing function
beta_0 = runif(p)
# beta_0 = rep(0, p)
betahats = my_OGD(X=X, Y=Y, lr=0.0000001, beta_0=beta_0)
plot(betahats[n, ] - beta, main="Differences between last estimate and true beta")
```



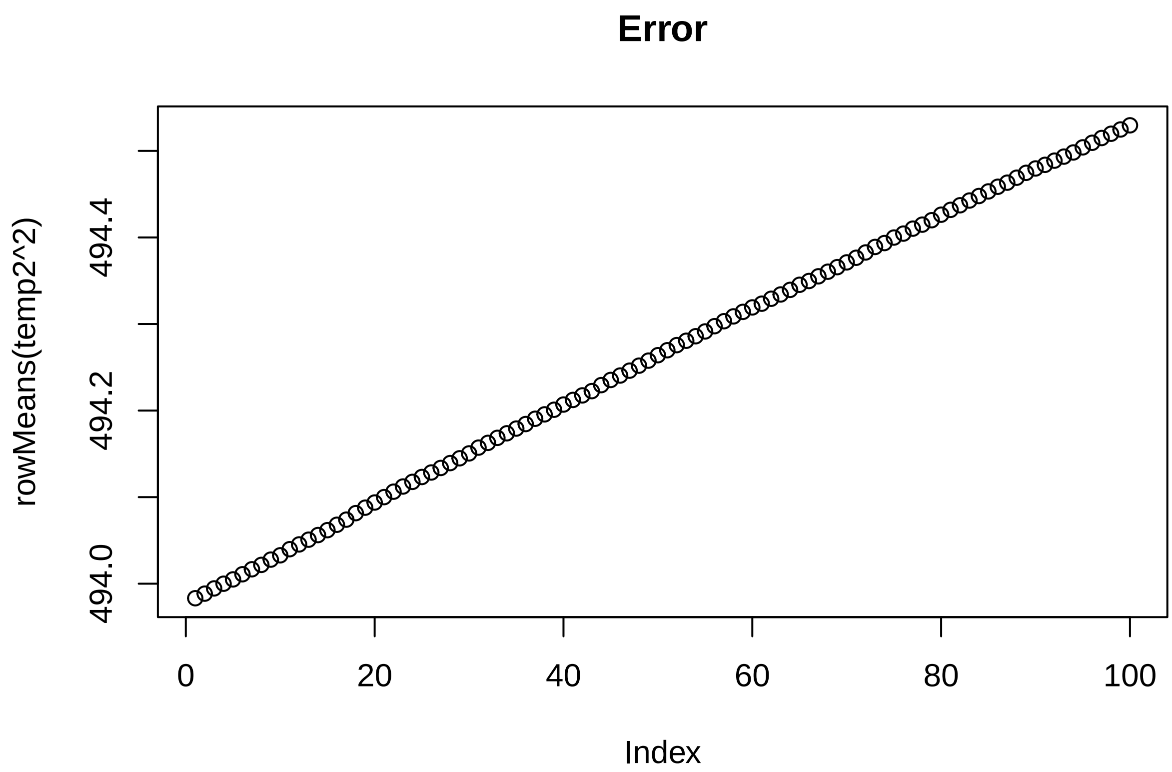
```r
sum(abs(betahats[n, ] - beta) > 0.5*mean(beta[nonzero_indexes])) == k
```

```
## [1] TRUE
```

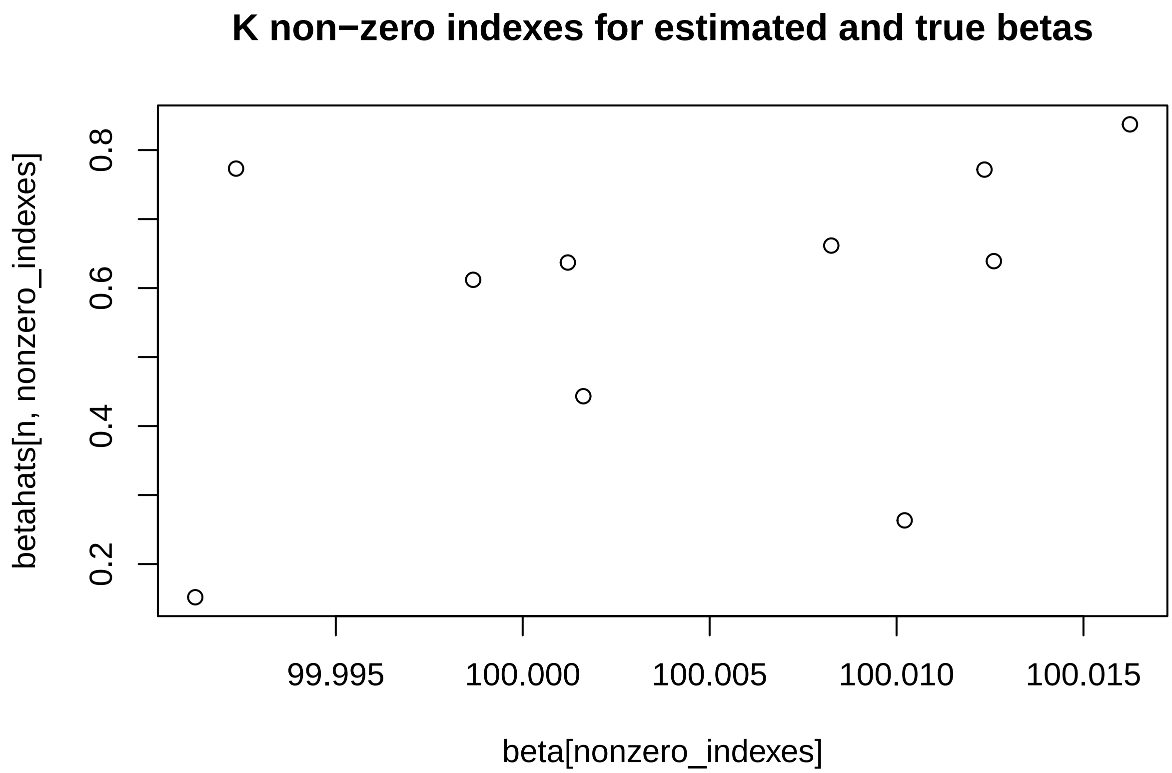
```r
mean(betahats[n, nonzero_indexes]) - mean(betahats[n, -nonzero_indexes])
```

```
## [1] 0.1326939
```

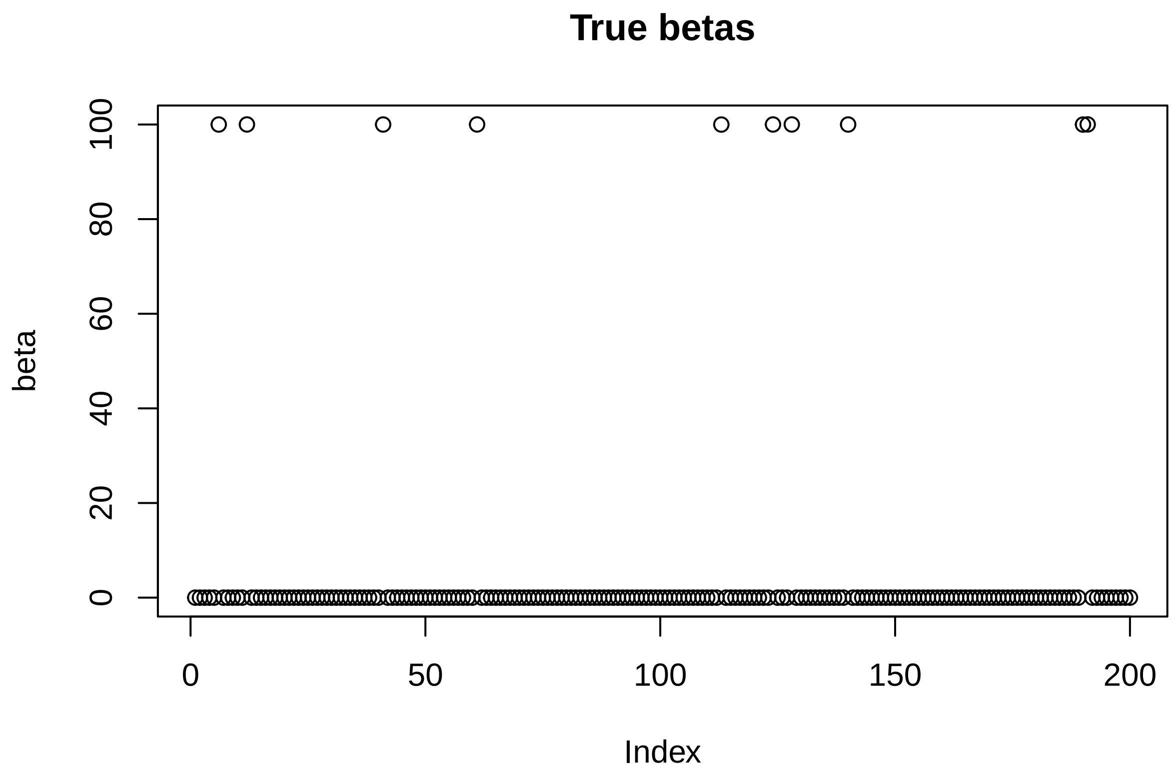
```r
temp = matrix(rep(beta, n), nrow=n, ncol=p, byrow=T)
temp2 = betahats - temp
plot(rowMeans(temp2^2), main="Error")
```



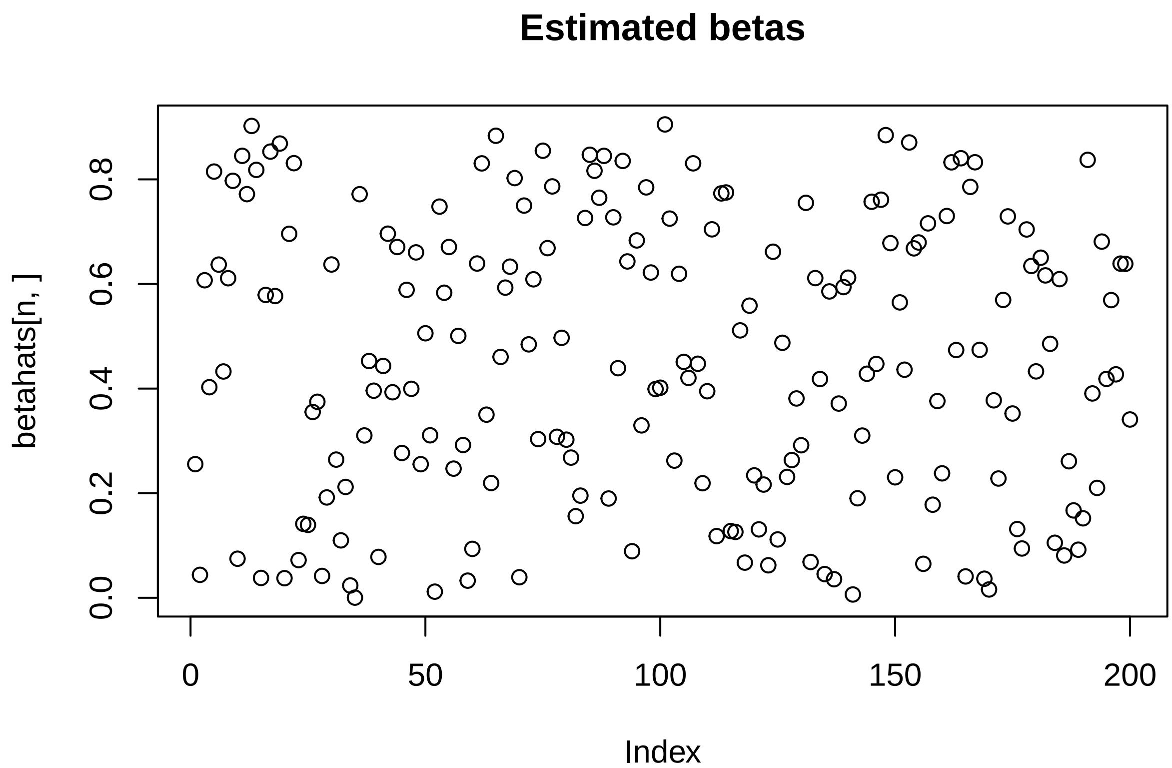

```r
plot(beta[nonzero_indexes], betahats[n, nonzero_indexes], main="K non-zero indexes for estimated and tr
```




```r
plot(beta, main="True betas")
```




```r
plot(betahats[n, ], main="Estimated betas")
```



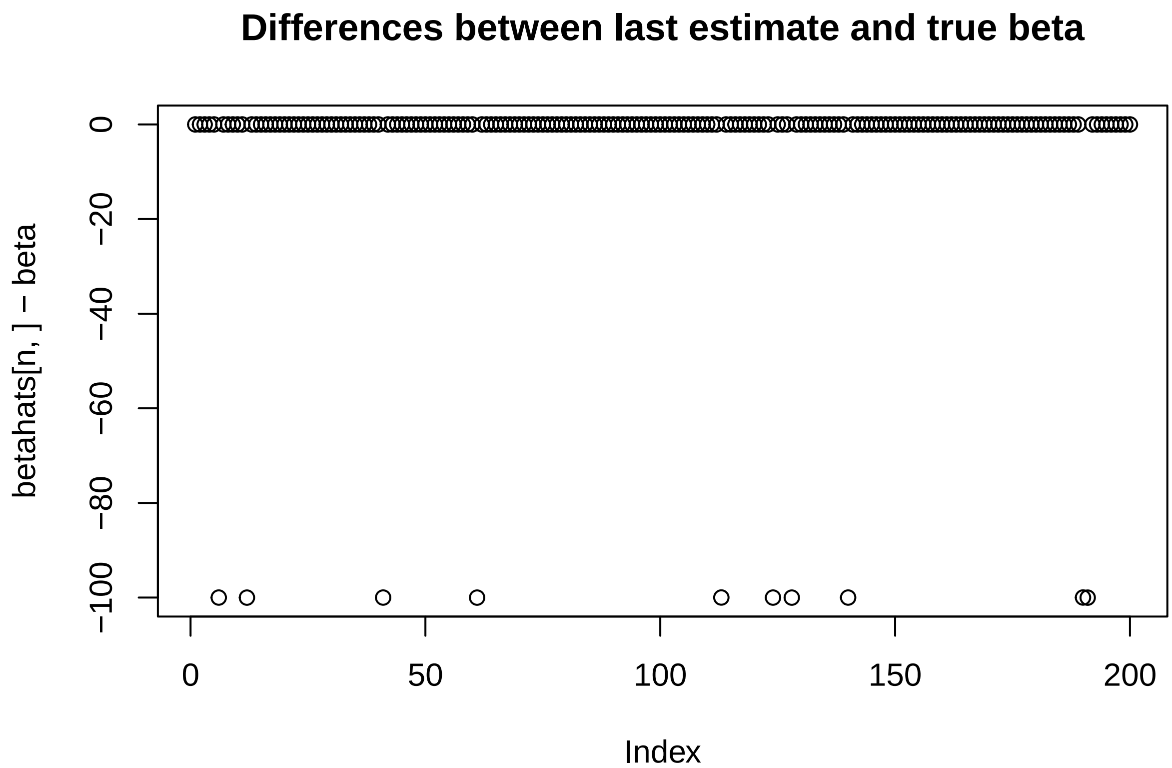

```
# function for online gradient descent (OGD)
# type: "classification" or "regresssion"
# X: rows are observations, columns are predictors
# Y: response variable
# learning rate (constant)
# beta_0: initialization for the estimate
# N: number of iterations for each data point (each row of X, Y)
# returns betahats for each data point and its N interations as a 3D array
my_OGD = function(type, X, Y, lr, beta_0, N) {
  if (type!='classification'&&type!='regression') {
    stop("Argument 'type' must be 'classification' or 'regression'")
  }

  n = nrow(X)
  betahats = array(data=rep(NA, n*p*N), dim = c(N, p, n)) # n arrays that are each Nxp arrays
  betahats[1, , ] = beta_0 # initialize
  if (type=='classification') {

  } else {
    # type is regression
    for (k in 1:n) { # for each data point (each row of X, Y)
      for (i in 1:(N-1)) { # for each iteration
        d_loss = 2*as.matrix(betahats[i, , k])%*%as.matrix(t(X[k, ]))%*%as.matrix(X[k, ])-2*as.matrix(X[k
        betahats[i+1, , k] = betahats[i, , k] - lr*d_loss
      } # end for i
    } # end for k
```

```
    return(betahats)
  } # end else for regression
}
```

```
# use my_OGD on generated data
betahats = my_OGD(type="regression", X=X, Y=Y, lr=0.00001, beta_0=rep(0, p), N=100)
```

```
# function for adaptive gradient descent (Adagrad)
# X: rows are observations, columns are predictors
# Y: response variable
# lr: global learning rate
# epsilon: noise for nonzero/invertibility
# beta_0: weight initialization
# full: boolean, uses full matrix for G if true, otherwise uses diagonal elements of G
my_adagrad = function(X, Y, lr, beta_0, full) {
  n = nrow(X)
  p = ncol(X)
  betahats = matrix(nrow=n, ncol=p)
  betahats[1, ] = beta_0
  g_vec = matrix(nrow=n, ncol=p) # save matrix for the gradients where each gradient g_t is the t^{th}
  G_t = matrix(data=rep(0, p^2), nrow=p, ncol=p) # matrix that is a cumulative sum

  for (t in 1:(n-1)) {
    x_t = as.matrix(X[t, ])
    beta_t = as.matrix(betahats[t, ])
    y_t_hat = t(beta_t)%*%x_t
    Y_t = Y[t]
    g_vec[t, ] = 2*beta_t%*%t(x_t)%*%x_t - 2*x_t%*%Y_t
    g_t = as.matrix(g_vec[t, ])
    G_t = G_t + g_t%*%t(g_t)
    diag_G_t = diag(diag(G_t), nrow=p, ncol=p)

    if (full) {
      # full
      betahats[t+1, ] = beta_t # TODO, maybe use built in functions for negative squre root
    } else {
      # diagonal
      betahats[t+1, ] = beta_t - lr*as.matrix(diag(diag(diag_G_t^(-1/2)), nrow=p, ncol=p))%*%g_t
    }
  } # end for
  return(betahats)
}
```

```
beta_0 = rep(0, nrow(X))
betahats = my_adagrad(X=X, Y=Y, lr=0.0000001, beta_0=beta_0, full=F)

# plotting
plot(betahats[n, ] - beta, main="Differences between last estimate and true beta")
```



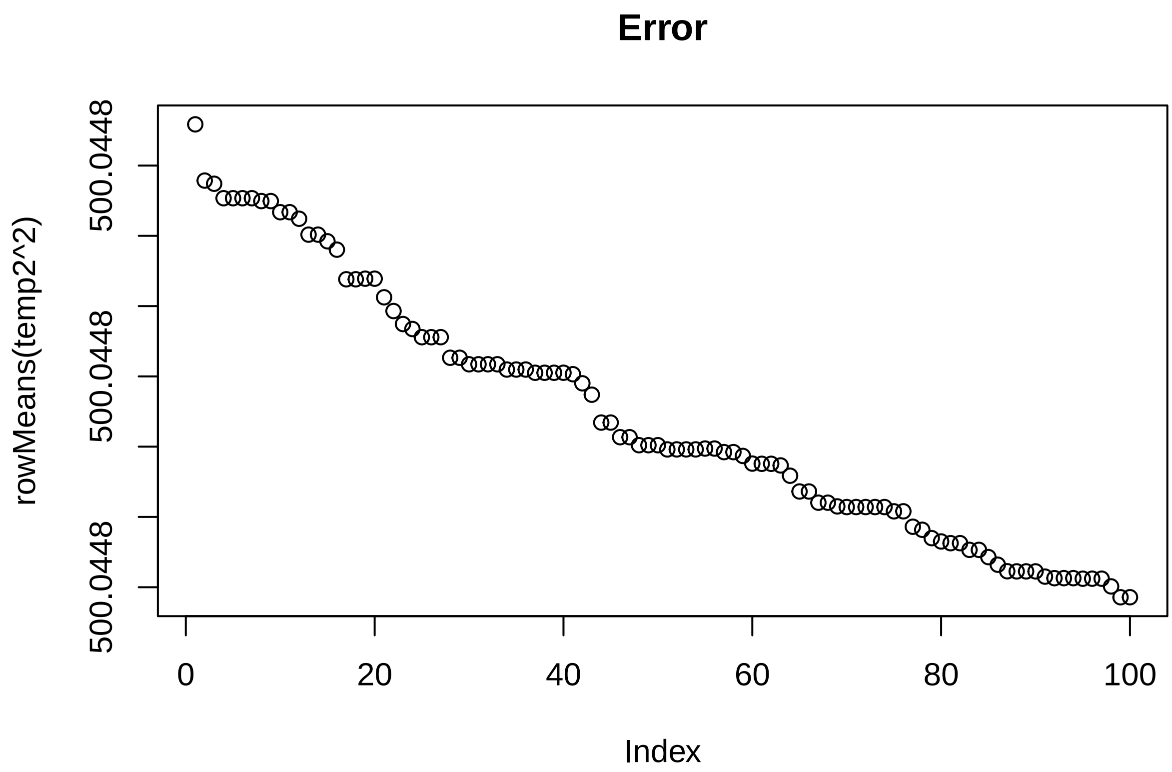

```
sum(abs(betahats[n, ] - beta) > 0.5*mean(beta[nonzero_indexes])) == k
```

```
## [1] TRUE
```

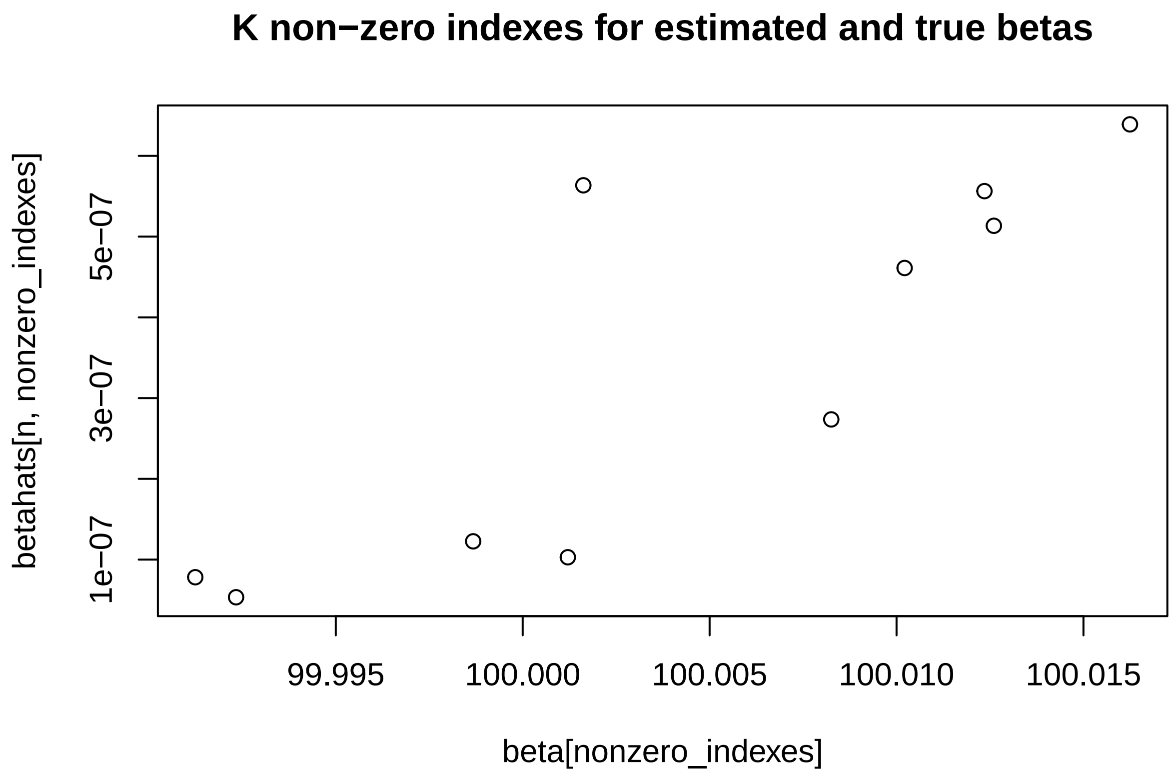
```
mean(betahats[n, nonzero_indexes]) - mean(betahats[n, -nonzero_indexes])
```

```
## [1] 3.370597e-07
```

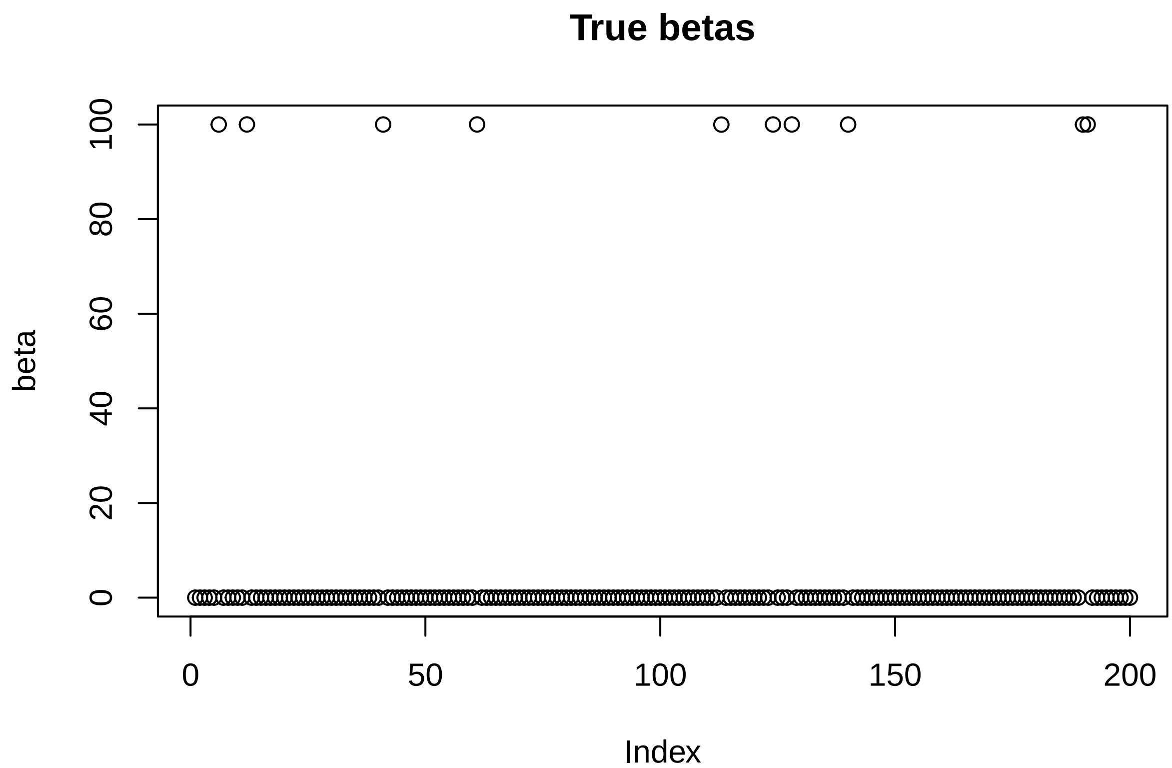
```
temp = matrix(rep(beta, n), nrow=n, ncol=p, byrow=T)
temp2 = betahats - temp
plot(rowMeans(temp2^2), main="Error")
```



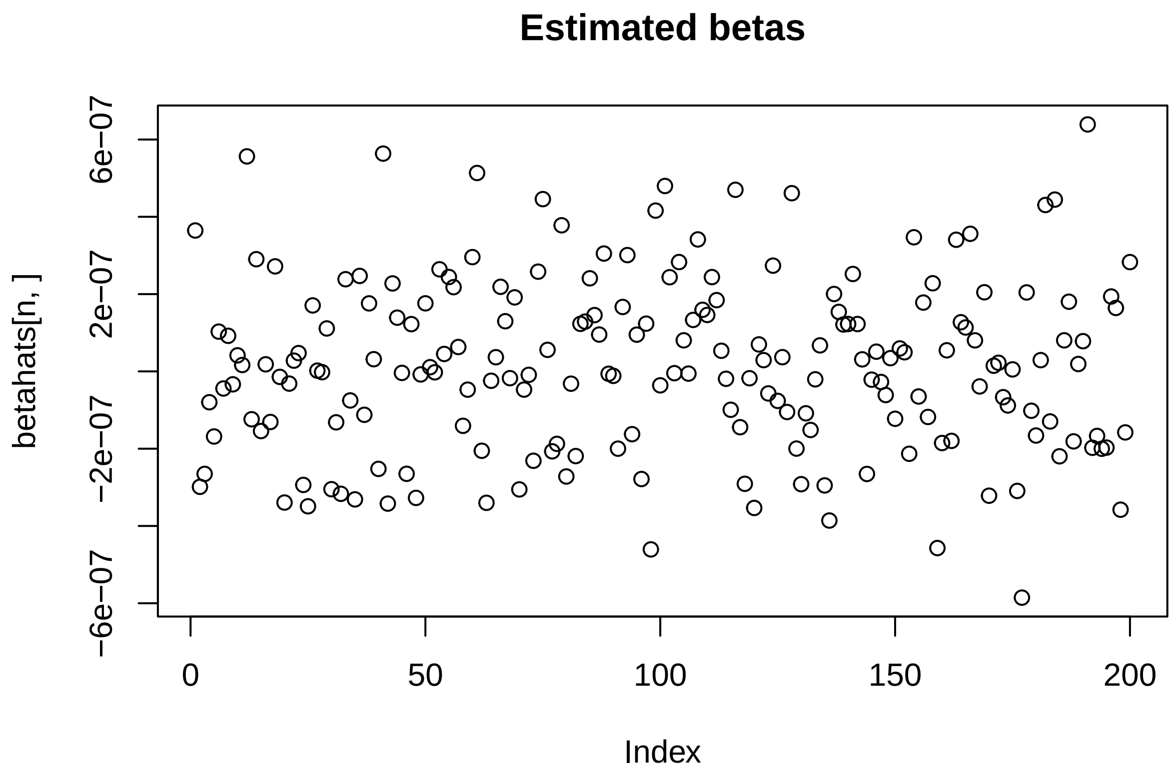

```
plot(beta[nonzero_indexes], betahats[n, nonzero_indexes], main="K non-zero indexes for estimated and tru
```




```
plot(beta, main="True betas")
```




```r
plot(betahats[n, ], main="Estimated betas")
```



Analysis of $\hat{\beta}$'s

Plots - Prediction error vs iterations - Estimation error vs iterations - Betahats for each dimension, nonzero vs zero indexes - Comparison of different learning rates - Run time of full vs diagonal Adagrad - Run time of OGD, Adagrad, etc - Variance of betahats across iterations?