# 6520 Project

Minjia Jia and Joia Zhang

Fall 2023

```
set.seed(6520)

library(expm)
```

```
## Loading required package: Matrix

##
## Attaching package: 'expm'

## The following object is masked from 'package:Matrix':
##
##     expm
```

## 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)
```

```r
# y

Y = X%*%beta + E

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

```r
# 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
# X: rows are observations, columns are predictors
# Y: response variable
# lr: global learning rate
# beta_0: weight initialization
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]
    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)
}
```

# Adagrad

```r
# 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 - lr*as.matrix(solve(sqrtm(G_t)))%*%g_t
    } 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)
}
```

# Adam

```
# function for adaptive moment estimation (Adam)
# X: rows are observations, columns are predictors
# Y: response variable
# lr: global learning rate (typical choice 0.001)
# beta_0: weight initialization
# epsilon: positive noise for nonzero/invertibility (typical choice 10^{-8})
# rho_1: 1st moment decay rate (typical choice 0.9)
# rho_2: 2nd moment decay rate (typical choice 0.999)
my_adam = function(X, Y, lr, beta_0, rho_1, rho_2, epsilon) {
  n = nrow(X)
  p = ncol(X)
  betahats = matrix(nrow=n, ncol=p)
  Ms = matrix(nrow=n, ncol=p) # 1st moment estimate
  Rs = matrix(nrow=n, ncol=p) # 2nd moment estimate
  Mhats = matrix(nrow=n, ncol=p) # 1st moment bias correction
  Rhats = matrix(nrow=n, ncol=p) # 2nd moment bias correction

  # initialize
  betahats[1, ] = beta_0
  Ms[1, ] = rep(0, p)
  Rs[1, ] = rep(0, p)

  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]
    d_loss = 2*beta_t%*%t(x_t)%*%x_t - 2*x_t%*%Y_t
    Ms[t+1, ] = rho_1*as.matrix(Ms[t, ]) + (1-rho_1)*d_loss
    Rs[t+1, ] = rho_2*as.matrix(Rs[t, ]) + (1-rho_2)*d_loss^2
    Mhats[t+1, ] = Ms[t, ] / (1-rho_1^t)
    Rhats[t+1, ] = Rs[t, ] / (1-rho_2^t)

    betahats[t+1, ] = beta_t - lr*(Mhats[t+1, ]/(sqrt(Rhats[t+1, ]+epsilon))) # update
  } # end for
  return(betahats)
}
```

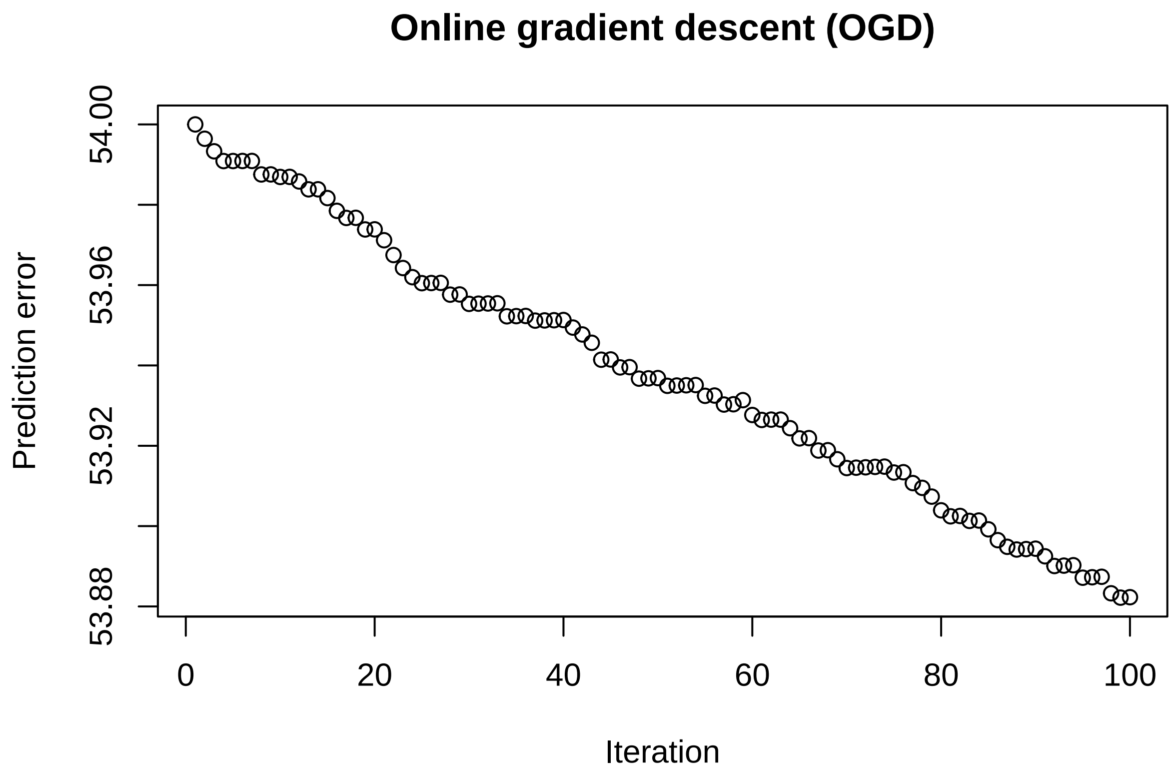
## Analysis of $\hat{\beta}$'s

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?

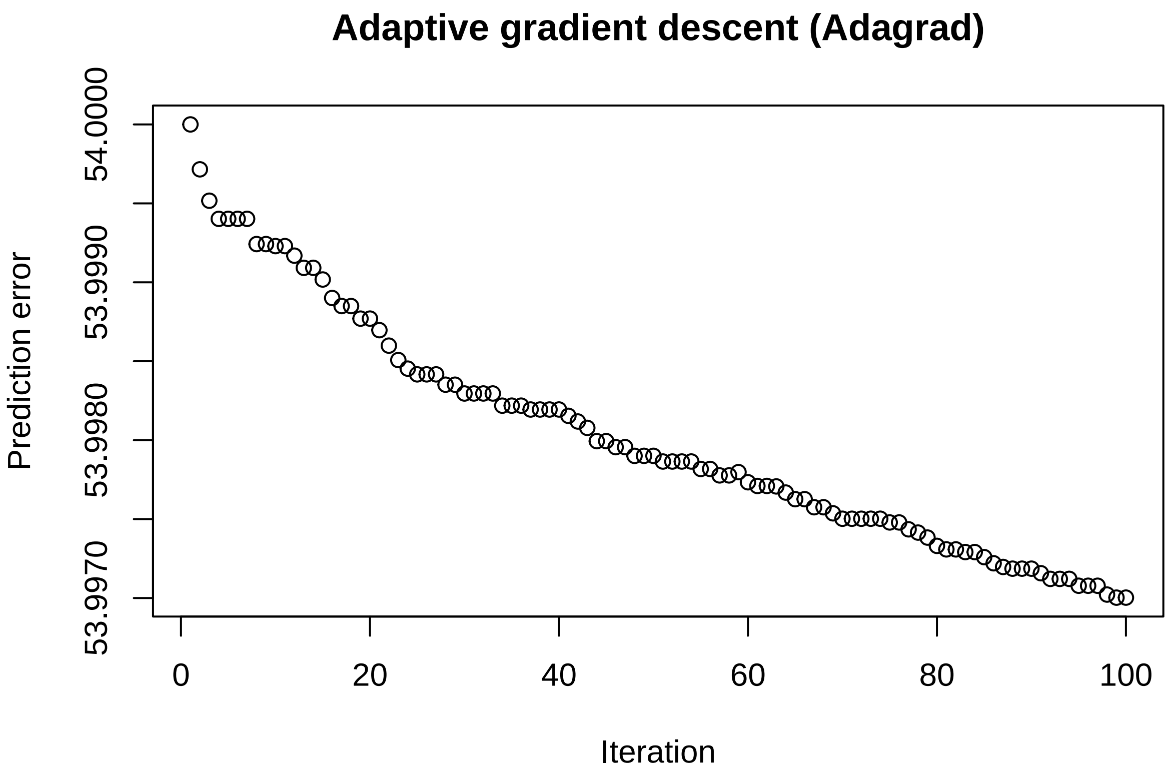
## Plots

```
# plot prediction error
# X: rows are observations, columns are predictors
# Y: response variable
# betahats: n x p matrix where each ith row is the coefficients for the ith iteration and the columns a
# title: string for the title of the plot
plot_prediction_error = function(betahats, X, Y, title) {
  n = nrow(X)
  p = ncol(X)
  pred_err = colSums((X%*%t(betahats) - matrix(rep(Y, n), nrow=n, ncol=n, byrow=F))^2) # row of the ins
  plot(pred_err, xlab="Iteration", ylab="Prediction error", main=title)
}
```

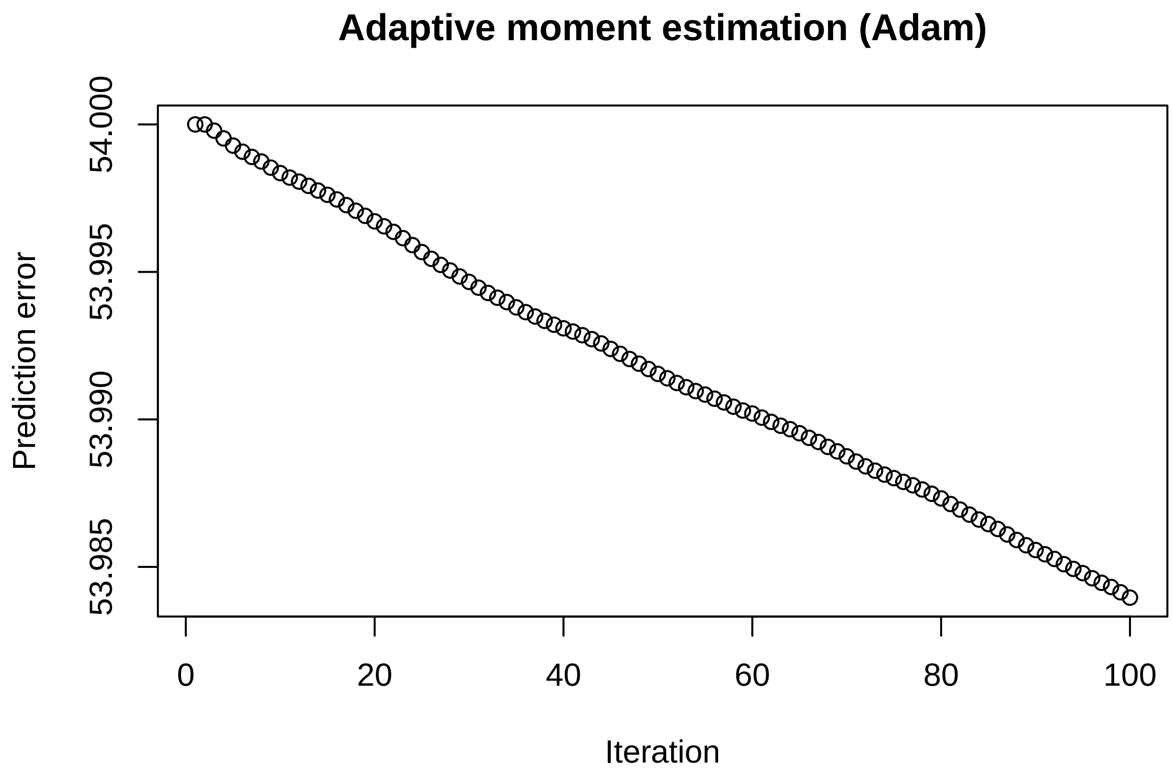
```
# OGD
betahats = my_OGD(X=X, Y=Y, lr=0.0000001, beta_0=rep(0, p))
plot_prediction_error(betahats, X, Y, "Online gradient descent (OGD)")
```




```r
# Adagrad
betahats = my_adagrad(X=X, Y=Y, lr=0.0000001, beta_0=rep(0, p), full=F)
plot_prediction_error(betahats, X, Y, "Adaptive gradient descent (Adagrad)")
```




```r
# Adam
betahats = my_adam(X=X, Y=Y, lr=0.0000001, beta_0=rep(0, p), rho_1=0.9, rho_2=0.999, epsilon=1e-8)
plot_prediction_error(betahats, X, Y, "Adaptive moment estimation (Adam)")
```



Adaptive moment estimation (Adam)
Prediction error
54.000
53.995
53.990
53.985
0
20
40
60
80
100
Iteration