6520 Project

Minjia Jia and Joia Zhang

Fall 2023

```
rm(list=ls())
set.seed(6520)
library(ggplot2)

library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'expm'
```

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

```
setwd("/Users/jwz34/Documents/Github/6520project/onlinegrad/R")
devtools::install()
```

```
## -- R CMD build -------------------------------------------------------------
##      checking for file '/Users/jwz34/Documents/Github/6520project/onlinegrad/DESCRIPTION' ...  v  ch
##   -  preparing 'onlinegrad':
##      checking DESCRIPTION meta-information ...  v  checking DESCRIPTION meta-information
##   -  checking for LF line-endings in source and make files and shell scripts
##   -  checking for empty or unneeded directories
##   -  building 'onlinegrad_0.0.0.9000.tar.gz'
##
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD INSTALL \
##   /var/folders/3l/795v58rx6110l72qtnz6rwmc0000gp/T//RtmpO8Jhcb/onlinegrad_0.0.0.9000.tar.gz \
##   --install-tests
## * installing to library '/Users/jwz34/Library/R/arm64/4.3/library'
## * installing *source* package 'onlinegrad' ...
## ** using staged installation
## ** R
## ** byte-compile and prepare package for lazy loading
## ** help
## *** installing help indices
## ** building package indices
## ** testing if installed package can be loaded from temporary location
## ** testing if installed package can be loaded from final location
## ** testing if installed package keeps a record of temporary installation path
## * DONE (onlinegrad)
```

```
library(onlinegrad)
```

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
```

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?

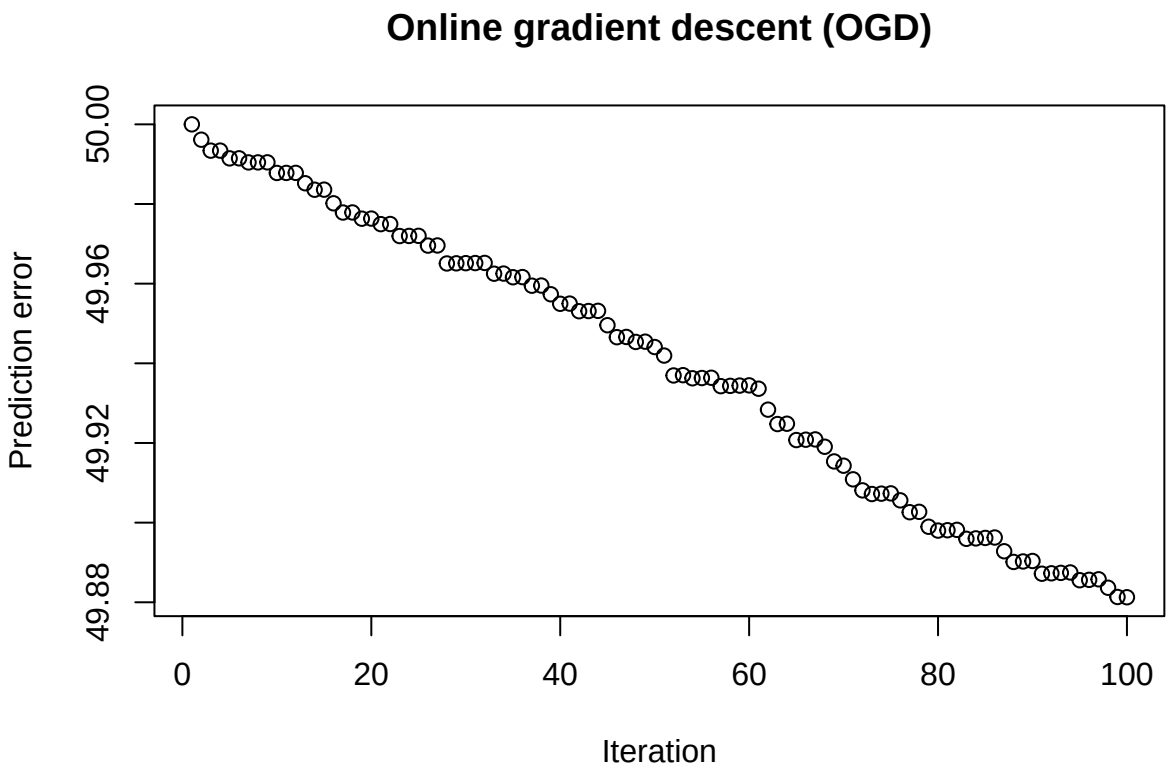
```
# plot prediction or estimation error
# X: rows are observations, columns are predictors
# Y: response variable
# betahats: n x p matrix where each ith row is the coefficients for the ith iteration and the columns a
# beta: true beta coefficient px1 vector
```

```r
# title: string for the title of the plot
# type: "prediction" or "estimation" for prediction error or estimation error
plot_prediction_error = function(betahats, beta, X, Y, title, type) {
  n = nrow(X)
  p = ncol(X)
  if ((type!="prediction") && (type!="estimation")) {
    stop("type parameter must be 'prediction' or 'estimation'")
  }
  if (type=="prediction") {
    # prediction error
    err = colSums((X%*%t(betahats) - matrix(rep(Y, n), nrow=n, ncol=n, byrow=F))^2) # row of the inside
    ylab = "Prediction error"
  } else {
    # estimation error
    beta = t(beta)
    beta = matrix(rep(beta, n),nrow=n, byrow=T) # row combine n number of t(beta)'s
    err = sqrt(rowSums((betahats - beta)^2))
    ylab = "Estimation error"
  }
  plot(as.matrix(err), xlab="Iteration", ylab=ylab, main=title)
}
```

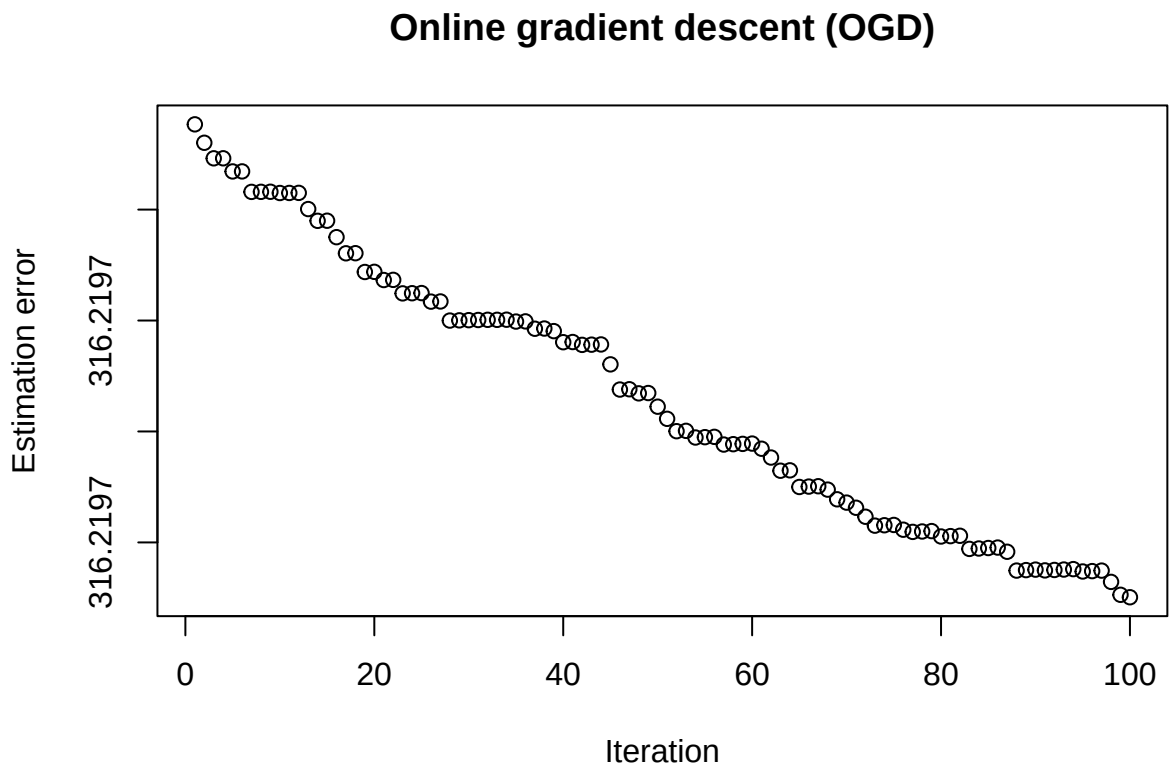
```r
# plot last iteration of betahat for nonzero vs zero indexes, true beta overlaid
# nonzero: boolean, if true plot only nonzero indexes (k indexes of the true k-sparse beta), otherwise
plot_betas = function(betahats, beta, nonzero_indexes, nonzero=T) {
  n = nrow(betahats)
  p = ncol(betahats)
  dat = data.frame("p"=1:p, "bethat_n"=betahats[n, ], "beta"=beta)
  if (nonzero) {
    dat = dat[nonzero_indexes, ]
    title = "Last iteration of bethat (orange) and true beta (black) at nonzero indexes"
  } else {
    dat = dat[-nonzero_indexes, ]
    title = "Last iteration of bethat (orange) and true beta (black) at zero indexes"
  }
  ggplot(dat) + geom_point(aes(x=p, y=bethat_n), color="orange") +
    geom_point(aes(x=p, y=beta), color="black", shape=4) +
    xlab("p") +
    ylab("") +
    ggtitle(title)

}
```

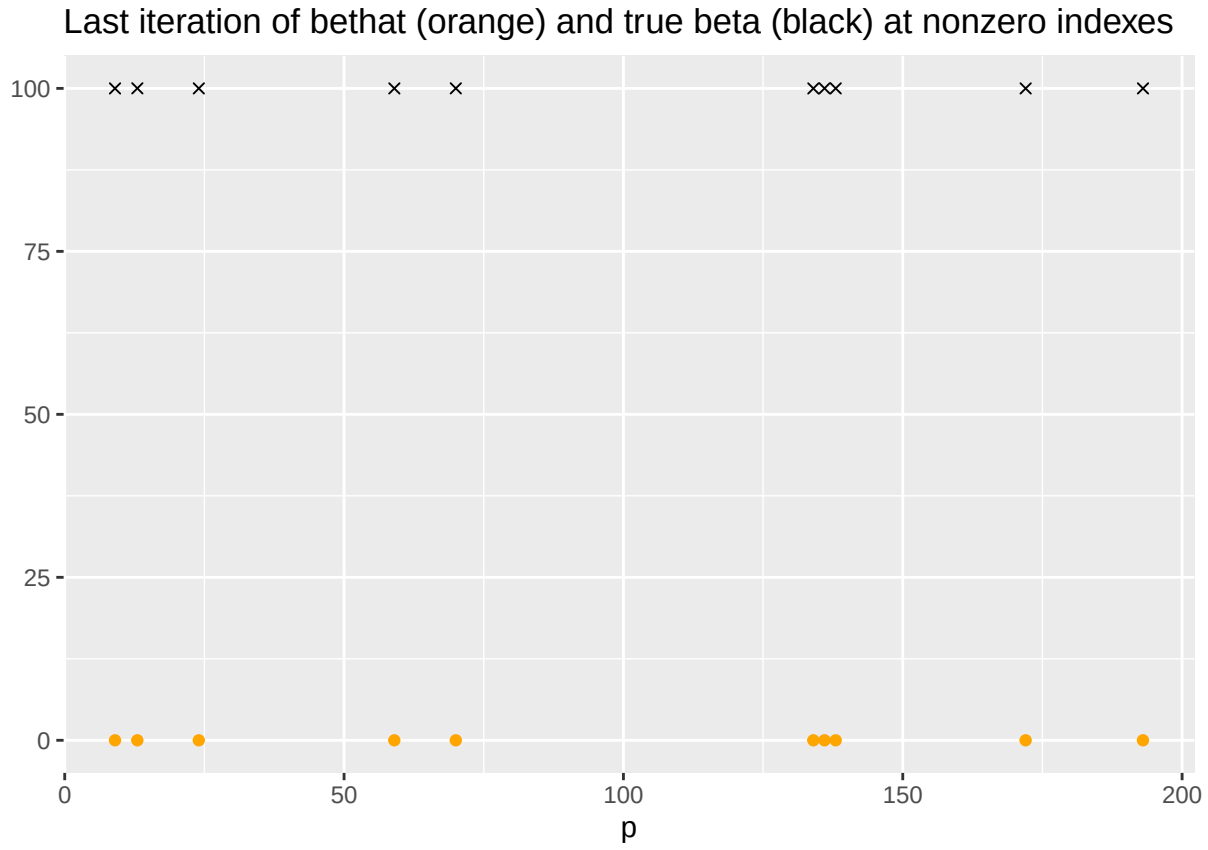
```r
# OGD
betahats = my_OGD(X=X, Y=Y, lr=0.0000001, beta_0=rep(0, p))
plot_prediction_error(betahats, beta, X, Y, title="Online gradient descent (OGD)", type="prediction") #
```



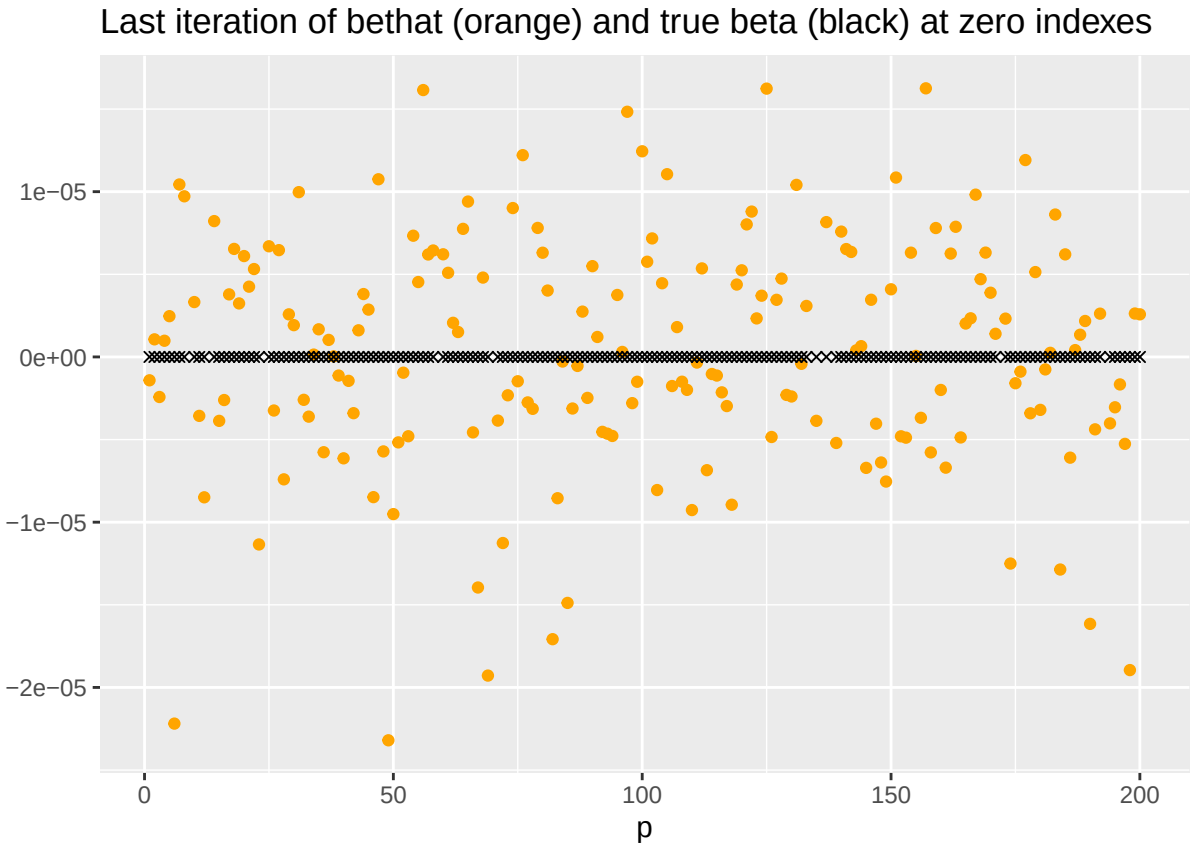

```
plot_prediction_error(betahats, beta, X, Y, title="Online gradient descent (OGD)", type="estimation") #
```



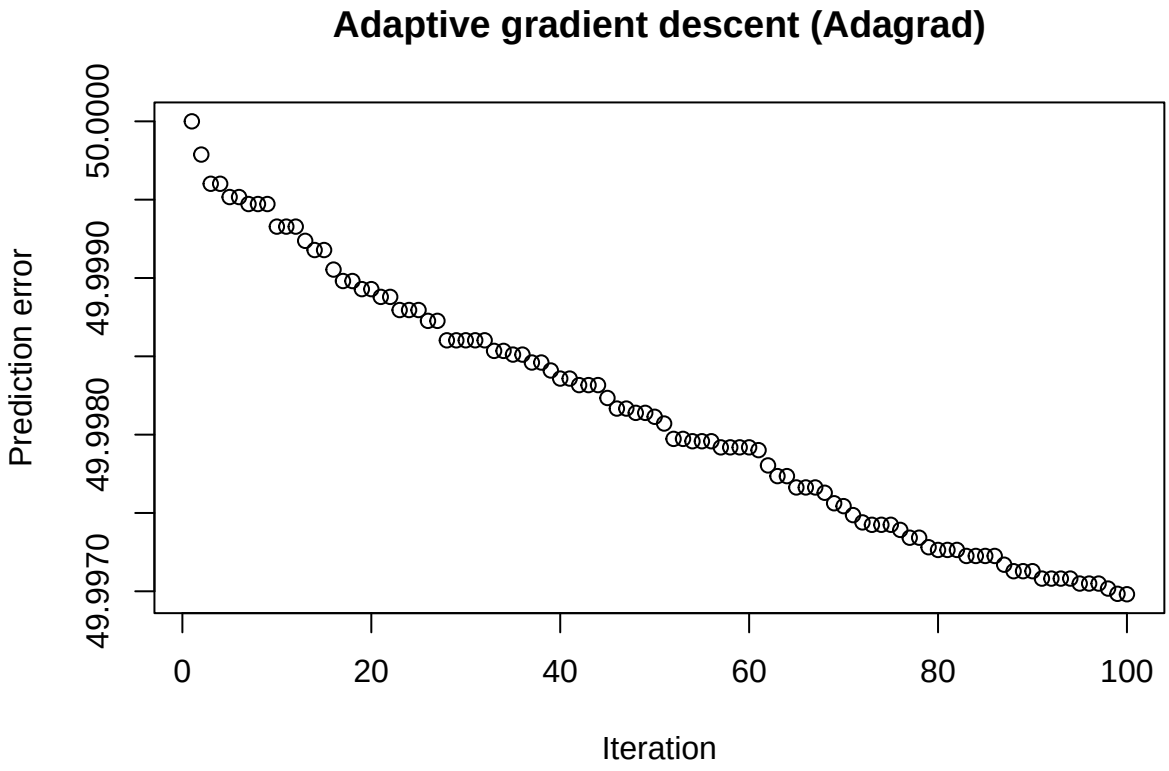

```
plot_betas(betahats, beta, nonzero_indexes, nonzero=T) # nonzero indexes
```



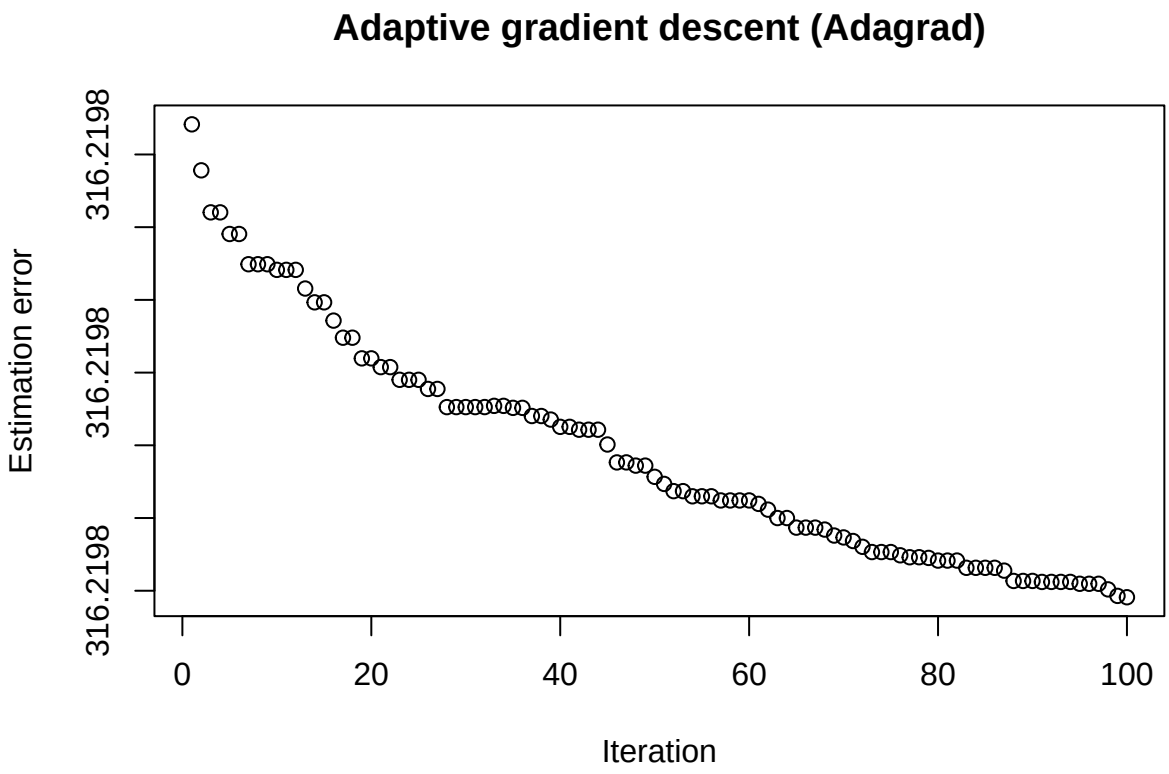

```
plot_betas(betahats, beta, nonzero_indexes, nonzero=F) # zero indexes
```



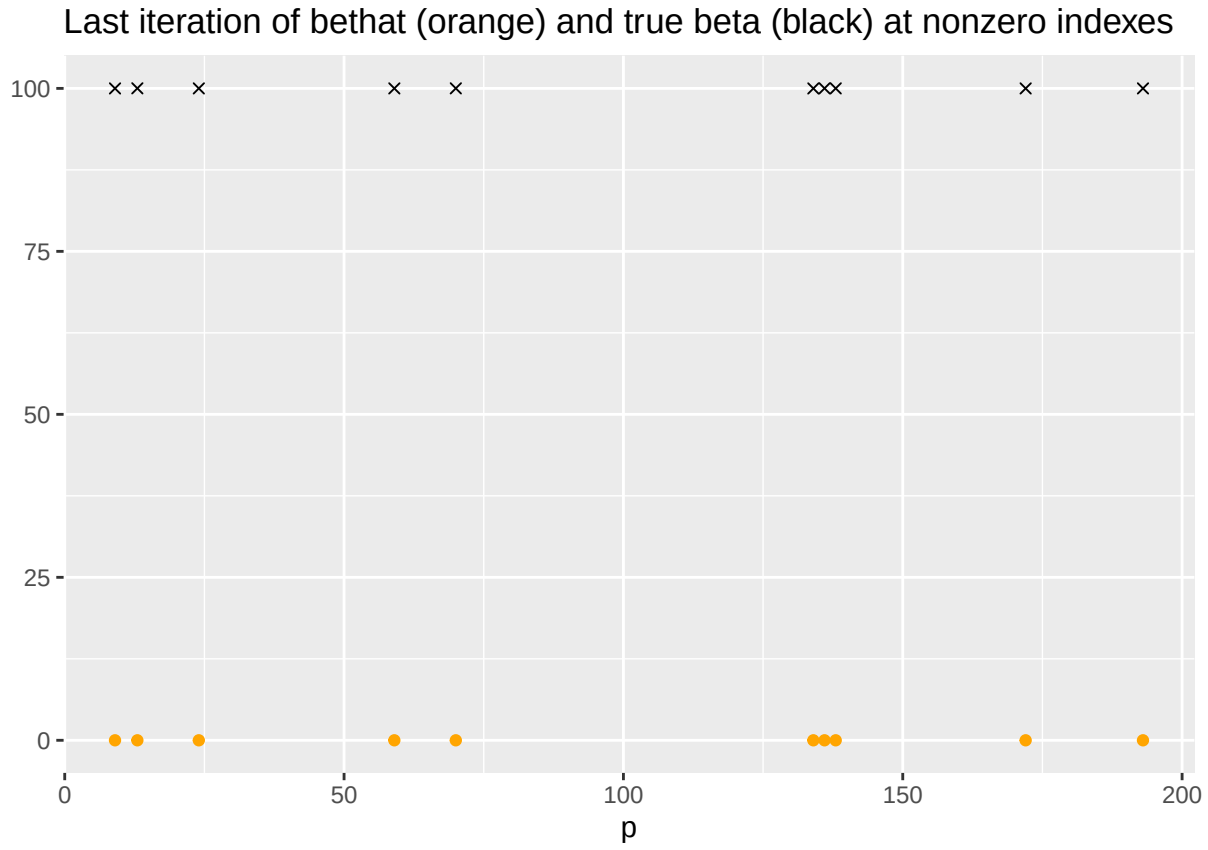

```
# Adagrad
betahats = my_adagrad(X=X, Y=Y, lr=0.0000001, beta_0=rep(0, p), full=F)
plot_prediction_error(betahats, beta, X, Y, title="Adaptive gradient descent (Adagrad)", type="predicti
```



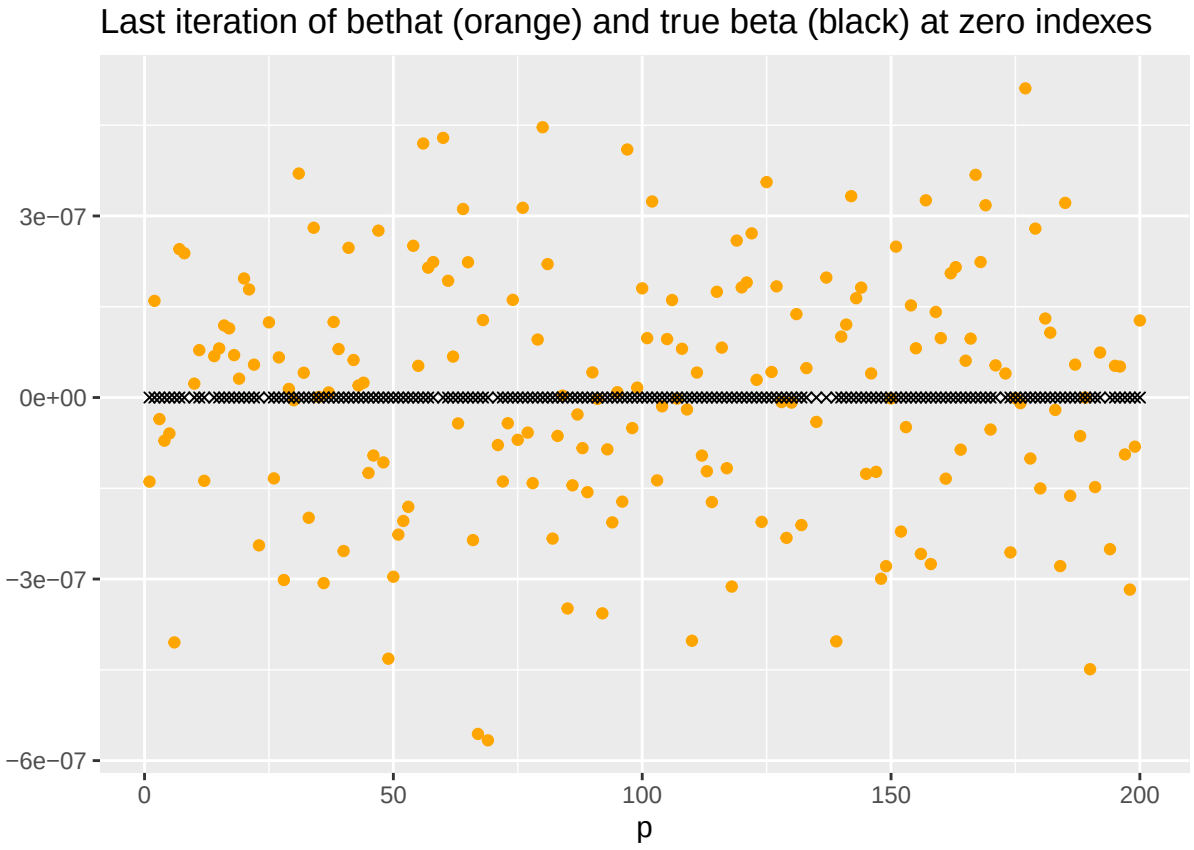

```
plot_prediction_error(betahats, beta, X, Y, title="Adaptive gradient descent (Adagrad)", type="estimati
```



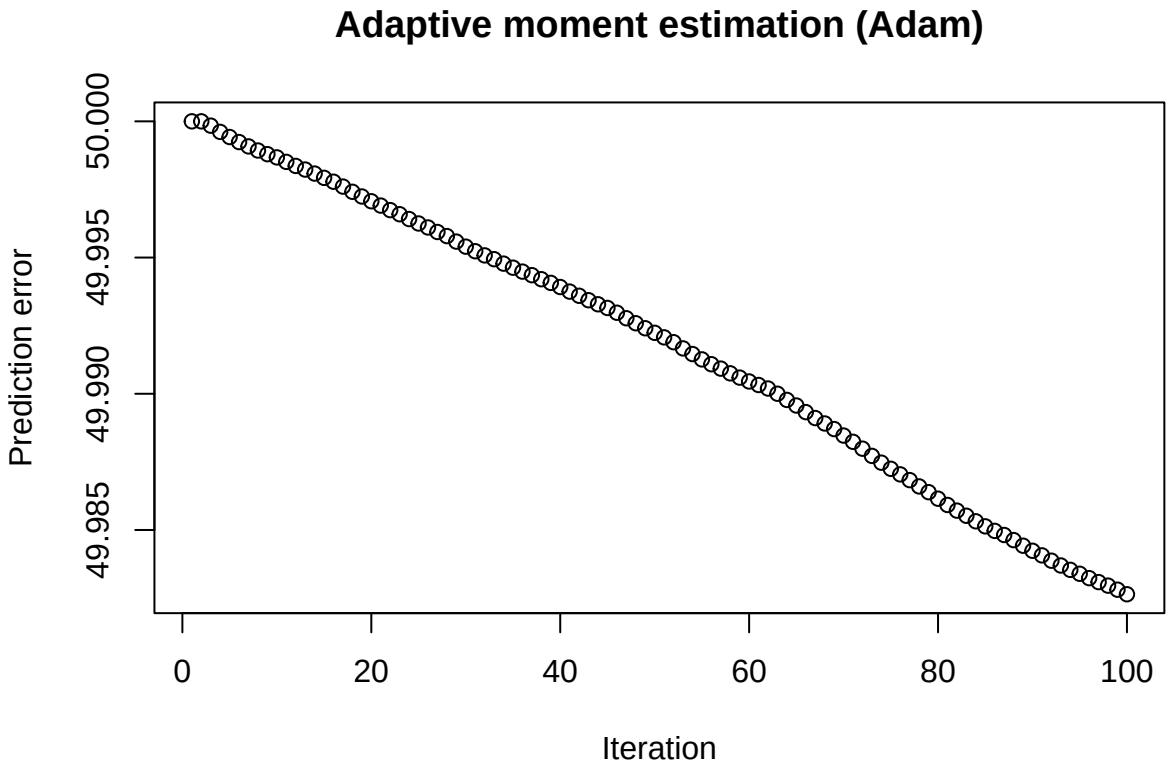

```
plot_betas(betahats, beta, nonzero_indexes, nonzero=T)
```



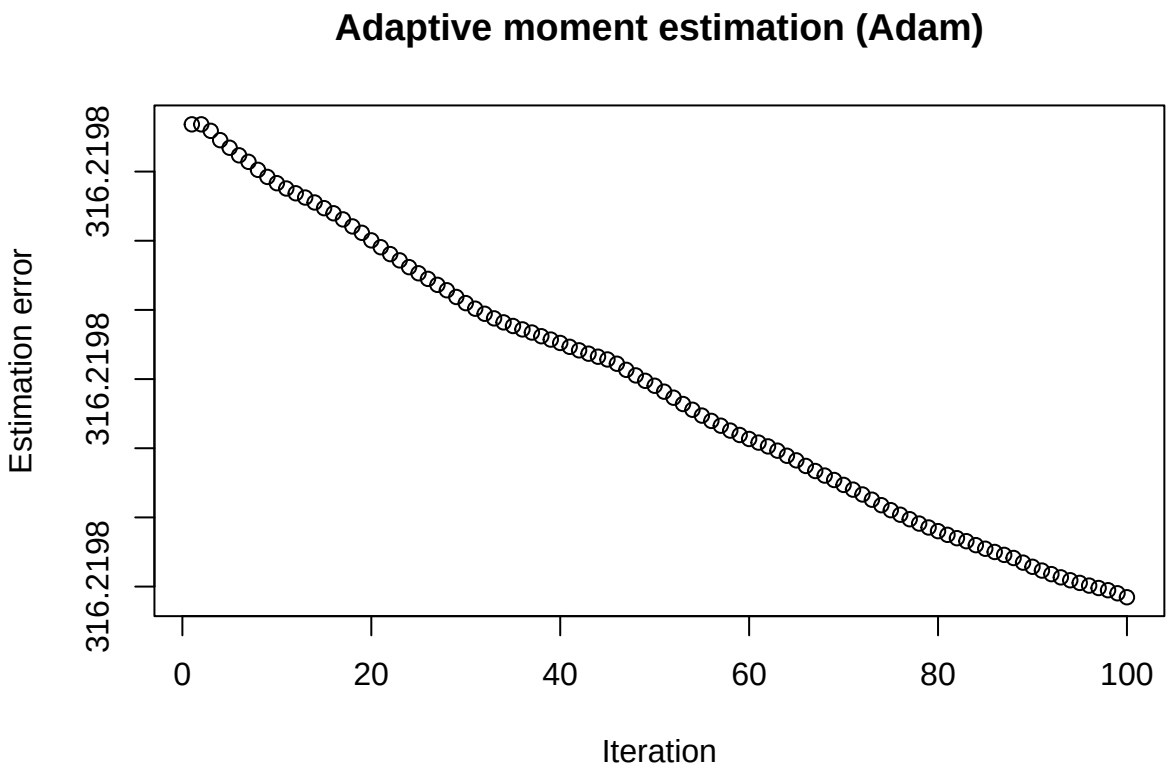

```
plot_betas(betahats, beta, nonzero_indexes, nonzero=F)
```



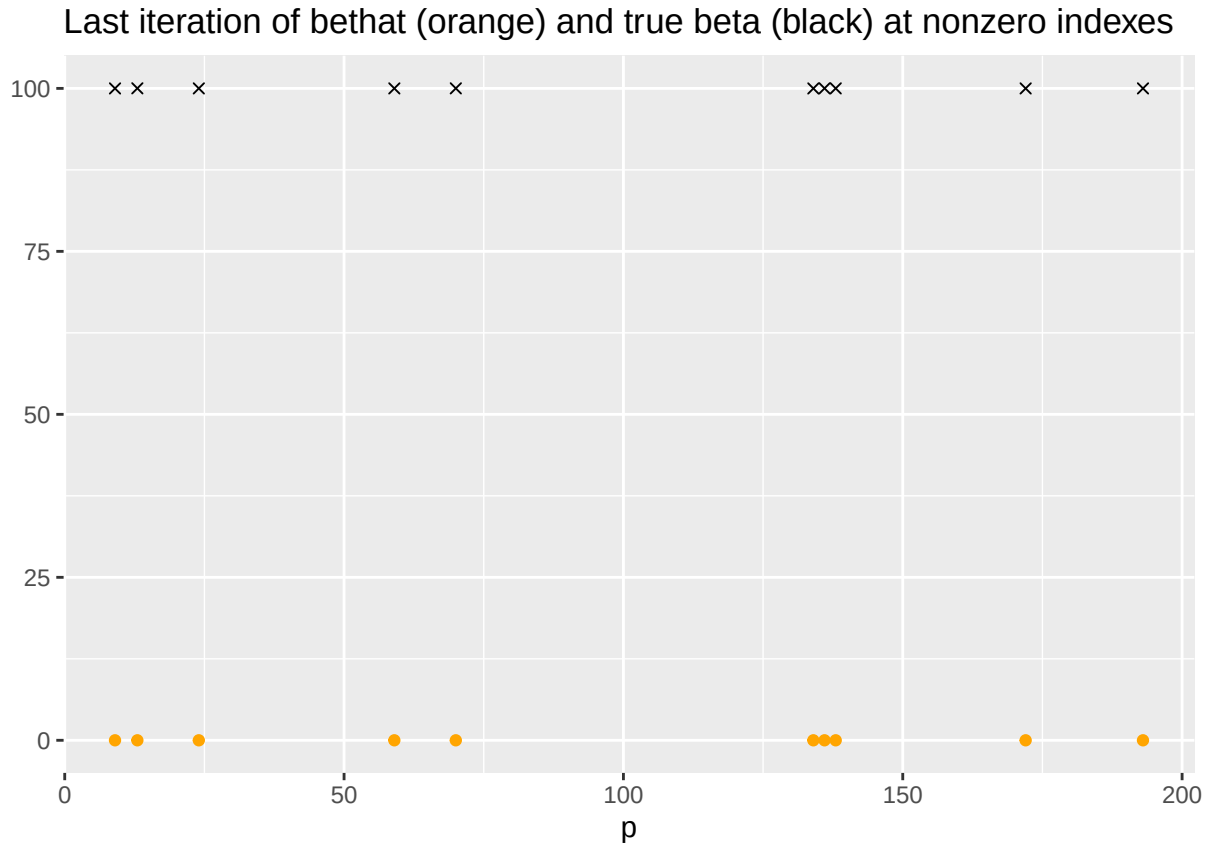

```
# Adam
betahats = my_adam(X=X, Y=Y, lr=0.0000001, beta_0=rep(0, p), rho_1=0.9, rho_2=0.999, epsilon=1e-8)
plot_prediction_error(betahats, beta, X, Y, title="Adaptive moment estimation (Adam)", type="prediction
```



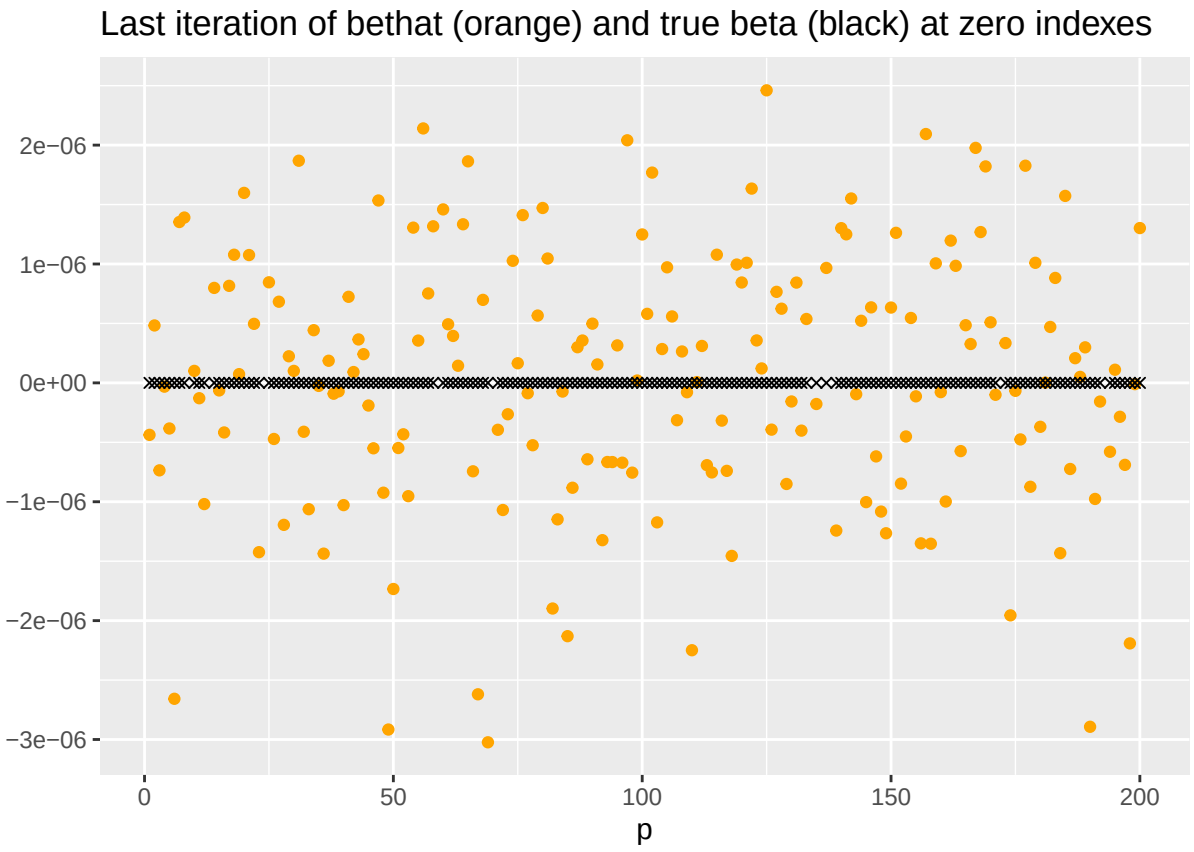

```
plot_prediction_error(betahats, beta, X, Y, title="Adaptive moment estimation (Adam)", type="estimation
```




```
plot_betas(betahats, beta, nonzero_indexes, nonzero=T)
```




```
plot_betas(betahats, beta, nonzero_indexes, nonzero=F)
```




```
# plot betahats for each dimension, nonzero vs zero indexes
```