Exercise set 2

your candidate number

Exercise 1

Background reading. Delete this section before submitting Wikipedia: Singular value decomposition. Read the introduction, Example, and Pseudoinverse sections.

Wikipedia: Pseudoinverse. Read the introduction and sections on Projectors, Examples, Linearly independent rows, and Applications.

Delete from here to the previous line about deletion.

```
set.seed(1) # change this to some other number
# Generate a matrix of predictors X
# with 3 rows, 10 columns, and i.i.d. normal entries
p <- 10
n <- 3
X <- matrix(rnorm(n * p), nrow = n)
# Create a sparse coefficient vector beta
beta <- rep(0, p)
beta[1] <- 1
# Compute outcome y from the noise-free linear model
y <- X %*% beta
```

Generate data

Compute pseudoinverse Use output from the `svd` function to compute the pseudoinverse of `X`. (Note: if you use a source aside from the Wikipedia articles above to figure out how to do this you should cite your source and include a link if it's a website)

```
S <- svd(X)
X_pseudoinv <- S$v %*% diag(1/S$d) %*% t(S$u)
```

```
X %*% X_pseudoinv
```

Verify right-inverse property

```
##               [,1]          [,2]          [,3]
## [1,]  1.000000e+00 -1.662692e-16 -1.289275e-16
## [2,] -3.247361e-16  1.000000e+00  1.401532e-16
## [3,] -1.886456e-16  3.672968e-16  1.000000e+00
```

Explanation: Expected to be the identity matrix, and the diagonal entries are numerically close to 1 and off diagonal close to zero, i.e. the matrix is numerically close to the identity matrix.

Compare estimated beta to true beta and find the *minimum L2-norm solution* to the under-determined linear equation `y = X %*% beta`. Is this solution sparse like the original `beta`?

```
beta_hat <- X_pseudoinv %*% y
cbind(beta, beta_hat)
```

```
##       beta
##  [1,]    1  0.21523253
##  [2,]    0 -0.11954345
##  [3,]    0 -0.11594383
##  [4,]    0  0.06647867
##  [5,]    0 -0.15885567
##  [6,]    0 -0.11901813
##  [7,]    0 -0.21929896
##  [8,]    0  0.14767483
##  [9,]    0 -0.07675998
## [10,]    0  0.14684021
```

Explanation: Expected they are not equal, because we do not have enough data to exactly solve for or estimate beta

```
y_hat <- X %*% beta_hat
mean((y - y_hat)^2)
```

Compute MSE for predicting y

```
## [1] 1.848893e-31
```

Explanation: Expected to be numerically close to zero, because perfect prediction is possible when `p > n`.

```
X_test <- matrix(rnorm(n * p), nrow = n)
y_test <- X_test %*% beta
y_test_hat <- X_test %*% X_pseudoinv %*% y
mean((y_test_hat - y_test)^2)
```

Generate a new sample of test data and compute the test MSE

```
## [1] 0.7142776
```

Explanation: Expected that this would be larger than zero, because test error is generally higher than training error.

Use gradient descent to estimate beta If you use any sources (including from this course) cite them here with a link.

```
neglogL <- function(X, Y, beta) {
  Xbeta <- X %*% beta
  -sum((Y-Xbeta)^2)
}
numeric_grad <- function(X, Y, beta) {
  y_hat <- X %*% beta
  -2 * t(X) %*% (y - y_hat)
}
Y <- y
max_steps <- 50 # try ~3 for comparison
beta_prev2 <- rnorm(ncol(X)) # random start
```

```r
#beta_prev2 <- c(mean(Y), cor(X[,-1], Y)) # "warm" start
grad_prev2 <- numeric_grad(X, Y, beta_prev2)
beta_prev1 <- beta_prev2 + 0.1 * grad_prev2 / sqrt(sum(grad_prev2^2))
grad_prev1 <- numeric_grad(X, Y, beta_prev1)
previous_loss <- neglogL(X, Y, beta_prev2)
next_loss <- neglogL(X, Y, beta_prev1)
steps <- 1

tol <- 1e-5 # (error) tolerance
while (abs(previous_loss - next_loss) > tol) {
  # Compute BB step size
  grad_diff <- grad_prev1 - grad_prev2
  step_BB <- sum((beta_prev1 - beta_prev2) * grad_diff) / sum(grad_diff^2)
  beta_prev2 <- beta_prev1

  # Update step
  beta_prev1 <- beta_prev1 - abs(step_BB) * grad_prev1

  # Shift previous steps
  grad_prev2 <- grad_prev1
  grad_prev1 <- numeric_grad(X, Y, beta_prev1)
  previous_loss <- next_loss
  next_loss <- neglogL(X, Y, beta_prev1)

  steps <- steps + 1
  if (steps > max_steps) break
}
```

```r
neglogL(X, Y, beta_prev1)
```

```
## [1] -7.129231e-10
```

```r
beta_prev1 - beta_hat
```

```
##              [,1]
##  [1,]  0.9430165
##  [2,]  0.7322280
##  [3,]  1.5522293
##  [4,] -0.2642195
##  [5,]  0.1902087
##  [6,]  1.0303714
##  [7,] -0.2500780
##  [8,]  0.3784304
##  [9,]  0.6479153
## [10,]  1.1847005
```

```r
cbind(beta, beta_hat, beta_prev1)
```

```
##       beta
##  [1,]    1  0.21523253  1.15824899
##  [2,]    0 -0.11954345  0.61268453
##  [3,]    0 -0.11594383  1.43628550
##  [4,]    0  0.06647867 -0.19774080
##  [5,]    0 -0.15885567  0.03135301
##  [6,]    0 -0.11901813  0.91135323
##  [7,]    0 -0.21929896 -0.46937694
```

```
##  [8,]    0  0.14767483  0.52610518
##  [9,]    0 -0.07675998  0.57115535
## [10,]    0  0.14684021  1.33154069
```