MTH210: Lab 10

MM Algorithm

1. **Run the code in `classDemonstration.R` which contains the MM-algorithm implementation for the location Cauchy example. Change the seed at the top, and rerun the code for different starting values.**

   No solution required for this.

2. **Ridge Regression:** Show that the ridge regression estimator in biased for β but has lower variance than the MLE estimator. (No need to use R for this).

3. **Bridge Regression**: **In this problem you will write a function to calculate the Bridge Regression estimate using MM algorithm for any given data set.**

   Recall that the Bridge objective function to minimize is

$$Q_B(\beta) = \frac{(y - X\beta)^T(y - X\beta)}{2} + \frac{\lambda}{2\alpha}\sum_{j=1}^{p} |\beta_j|^\alpha$$

   We found the minorizing function to be

$$\tilde{f}(\beta|\beta_{(k)}) = \text{constants} + \frac{(y - X\beta)^T(y - X\beta)}{2} + \frac{\lambda}{2\alpha}\sum_{j=1}^{p} m_{j,(k)}\beta_j^2$$

   where $m_{j,(k)} = \alpha|\beta_{j,(k)}|^{\alpha-2}$. Here β_j and $\beta_{j,(k)}$ denoted the jth component of β and $\beta_{(k)}$ respectively. The maximizer of this minorizing function gives us the next iterate of the MM algorithm, which is

$$\beta_{(k+1)} = \left(X^T X + \frac{\lambda}{\alpha} M_{(k)}\right)^{-1} X^T y\,,$$

   where $M_{(k)}$ is the diagonal matrix of $m_{j,(k)}$s.

   Now, you will write a function that takes data y, X as input, along with values of λ and α, and returns the Bridge regression estimate. The function should look like:

```
# y = response
# X = covariate matrix
# lambda = penlaty term
# alpha = bridge term
# max.ter = maximum iterations for the MM algorithm. if max iteration has been reached, the func
# tol = tolerance level for when to stop MM
bridgeReg <- function(y, X, alpha, lambda, max.iter, tol)
{
  ...

  while(... < tol)
  {
    if(iter > max.iter)
    {
      print("Maximum iterations reached")
      stop
    }
    # MM steps
  }

  return(...)
}
```

Below is a version of the function. In the function below I have set the starting value to be the ridge solution as that is available tractably, can be easily calculated, and it expected to be close to the bridge solution.

```
# y = response
# X = covariate matrix
# lambda = penlaty term
# alpha = bridge term
# max.ter = maximum iterations for the MM algorithm. if max iteration has been reached, the func
# tol = tolerance level for when to stop MM
bridgeReg <- function(y, X, alpha, lambda, max.iter, tol)
{
  # a value larger than tol
  distance <- tol + 1
  iter <- 0
  p <- dim(X)[2]

  # starting from the ridge solution
  current <- matrix(1, ncol = 1, nrow = p)
    #(t(X) %*% X + lambda*diag(p)) %*% t(X) %*%y
  while(distance > tol)
  {
    iter <- iter + 1
    if(iter > max.iter)
    {
      print("Maximum iterations reached")
      stop
    }

    # MM steps
    previous <- current
    mjs <- alpha* abs(current)^(alpha - 2)

    ## using qr.solve since that is more stable than solve
    current <- qr.solve(t(X)%*%X + lambda/alpha * diag(mjs) ) %*% t(X) %*% y
    distance <- norm(previous - current, "2")
  }
  #returning the last iterate of the
  return(current)
}
```

4. **Simulate the following data:**

```
set.seed(1)
n <- 100
p <- 5
beta.star <- c(3, 1, .01, -2, -.003)

# X matrix with intercept
X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))

# generating response
y <- X %*% beta.star + rnorm(n)
```

Call your `bridgeReg()` function from the previous question on this dataset for the following values of α and λ, and the print the estimate. See if the estimate is close to the true value of β from which the data was simulated.

Below is the completed code

```
alpha.vec <- seq(1, 2, length = 5)
lambda <- c(.01, .1, 1, 10, 100)

for(i in 1:length(alpha.vec))
{
  for(j in 1:length(lambda))
  {
    bridge.est <- bridgeReg(y, X, alpha = alpha.vec[i], lambda = lambda[j], max.iter = 100, tol
    print(bridge.est)
  }
}
```

```
            [,1]
[1,]  2.94295600
[2,]  1.19323492
[3,]  0.17953850
[4,] -1.96632131
[5,]  0.03960107
            [,1]
[1,]  2.94233976
[2,]  1.19239256
[3,]  0.17862704
[4,] -1.96694066
[5,]  0.03887208
            [,1]
[1,]  2.93626765
[2,]  1.18409239
[3,]  0.16964596
[4,] -1.97304337
[5,]  0.03168906
            [,1]
[1,]  2.88354026
[2,]  1.11201757
[3,]  0.09165849
[4,] -2.02603647
[5,] -0.03068493
           [,1]
[1,]  2.6836096
[2,]  0.8387257
[3,] -0.2040529
[4,] -2.2269747
[5,] -0.2671934
            [,1]
```

```
[1,]  2.94293477
[2,]  1.19320589
[3,]  0.17950709
[4,] -1.96634265
[5,]  0.03957594
          [,1]
[1,]  2.9421281
[2,]  1.1921032
[3,]  0.1783140
[4,] -1.9671534
[5,]  0.0386217
           [,1]
[1,]  2.93422146
[2,]  1.18129539
[3,]  0.16661951
[4,] -1.97509987
[5,]  0.02926852
           [,1]
[1,]  2.86865134
[2,]  1.09166542
[3,]  0.06963675
[4,] -2.04100042
[5,] -0.04829781
          [,1]
[1,]  2.6619496
[2,]  0.8091180
[3,] -0.2360894
[4,] -2.2487438
[5,] -0.2928161
           [,1]
[1,]  2.94290695
[2,]  1.19316787
[3,]  0.17946595
[4,] -1.96637061
[5,]  0.03954304
           [,1]
[1,]  2.94185125
[2,]  1.19172480
[3,]  0.17790449
[4,] -1.96743163
[5,]  0.03829419
           [,1]
[1,]  2.93157660
```

```
[2,]  1.17768004
[3,]  0.16270758
[4,] -1.97775806
[5,]  0.02613978
            [,1]
[1,]  2.85120433
[2,]  1.06781652
[3,]  0.04383141
[4,] -2.05853535
[5,] -0.06893679
           [,1]
[1,]  2.6427868
[2,]  0.7829236
[3,] -0.2644326
[4,] -2.2680032
[5,] -0.3154848
            [,1]
[1,]  2.94287053
[2,]  1.19311809
[3,]  0.17941208
[4,] -1.96640721
[5,]  0.03949996
            [,1]
[1,]  2.94148929
[2,]  1.19123002
[3,]  0.17736913
[4,] -1.96779542
[5,]  0.03786601
            [,1]
[1,]  2.92817312
[2,]  1.17302770
[3,]  0.15767358
[4,] -1.98117869
[5,]  0.02211362
            [,1]
[1,]  2.83131566
[2,]  1.04063003
[3,]  0.01441468
[4,] -2.07852425
[5,] -0.09246415
           [,1]
[1,]  2.6261654
[2,]  0.7602033
```

```
[3,] -0.2890168
[4,] -2.2847083
[5,] -0.3351471
            [,1]
[1,]  2.94282284
[2,]  1.19305289
[3,]  0.17934154
[4,] -1.96645515
[5,]  0.03944354
            [,1]
[1,]  2.94101636
[2,]  1.19058356
[3,]  0.17666963
[4,] -1.96827073
[5,]  0.03730656
            [,1]
[1,]  2.92381895
[2,]  1.16707585
[3,]  0.15123347
[4,] -1.98555480
[5,]  0.01696285
            [,1]
[1,]  2.80932708
[2,]  1.01057312
[3,] -0.01810795
[4,] -2.10062363
[5,] -0.11847558
           [,1]
[1,]  2.6119778
[2,]  0.7408099
[3,] -0.3100012
[4,] -2.2989674
[5,] -0.3519304
```

When the above code runs, the estimate of β is printed for each value of α and λ. You can see that for larger values of λ, the larger estimates are pushed closer to 0.

5. **In the previous question, suppose we fix $\lambda = 10$ and for different values of α, we obtain a different estimation of bridge regression coefficient: $\hat{\beta}_{\alpha,\lambda}$. The distance of this estimate from the true β can be measured with $\|\hat{\beta}_{\alpha,\lambda} - \beta\|$.**

```
for(i in 1:length(alpha.vec))
{
    bridge.est <- bridgeReg(......)
```

```r
    norm(bridge.est - beta.star, "2")
  }
```

This is a random quantity (since $\hat{\beta}_{\alpha,\lambda}$ is random). Suppose I want to find the average of expected distance of $\hat{\beta}_{\alpha,\lambda}$ from β.

$$E\left[\|\hat{\beta}_{\alpha,\lambda} - \beta\|\right]$$

The expectation is with respect to the distribution of $\hat{\beta}_{\alpha,\lambda}$, which we don't know. But since this is a simulated data setup, we can simulate multiple such datasets (keeping the same value of β), and obtain different realizations of $\hat{\beta}_{\alpha,\lambda}$. Taking an average of all the resulting distances will give us an estimate of the above expectation. Implement this exercise and see which value of α gives the lowest expected distance. Below is code that might help you.

```r
alpha.vec <- seq(1, 2, length = 5)
reps <- 100
dist <- matrix(0, nrow = reps, ncol = length(alpha.vec))
for(r in 1:reps)
{
  # generate X and y again here
  # X matrix with intercept
    X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))

    # generating response
    y <- X %*% beta.star + rnorm(n)
  for(i in 1:length(alpha.vec))
  {
    bridge.est <- bridgeReg(y, X, alpha = alpha.vec[i], lambda = 10, max.iter = 100, tol = 1e-5)
    dist[r, i] <- norm(bridge.est - beta.star, "2")
  }
}
```

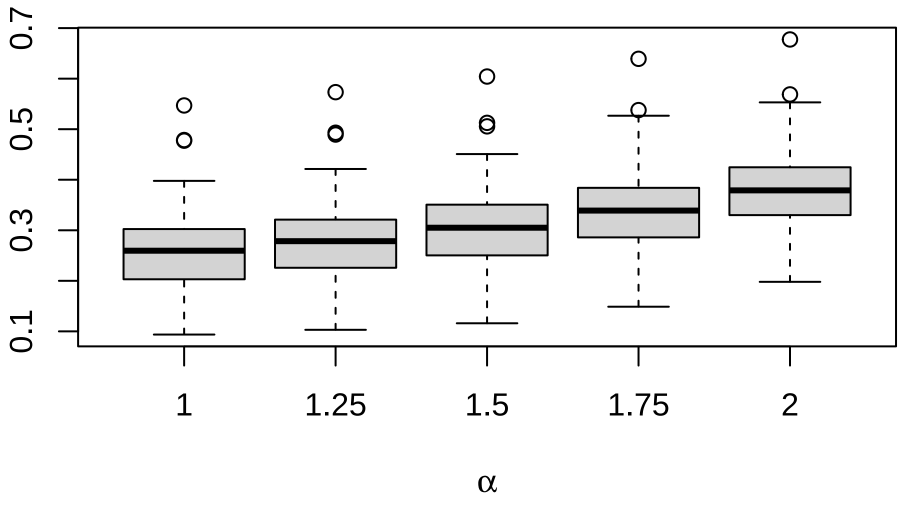
Now `dist` has the distance from the true β for every value of α in the each column, and replications are in rows. Thus to estimate the expectation, we can take column means:

```r
# naming the columns of dist
colnames(dist) <- alpha.vec
# expectation
colMeans(dist)
```

```
        1      1.25       1.5      1.75         2
0.2590982 0.2781903 0.3047150 0.3390168 0.3803636
```

We can also plot the column-wise boxplot:

```r
boxplot(dist, xlab = expression(alpha))
```



From the looks of it, $\alpha = 1$ (Lasso) has the least expected error.