MTH210: Lab 8 Solutions

Linear Regression, MLE, Ridge, and Newton-Raphson

1. **Load the `cars` dataset in R:**

```
data(cars)
```

Fit a linear regression model using maximum likelihood with response y being the distance and x being speed. Remember to include an intercept term in X by making the first column as a column of 1s. Do not use inbuilt functions in R to fit the model.

In this example, we recall that the linear regression MLE is

$$\hat{\beta}_{\text{MLE}} = (X^T X)^{-1} X^T y \,.$$

We just need to obtain X first, and then calculate the estimator.

```
X <- cars$speed
X <- cbind(1, X)  # add a column
y <- cars$dist

beta.mle <- solve(t(X) %*% X) %*% t(X) %*% y
beta.mle
```

```
        [,1]
  -17.579095
X   3.932409
```

2. **Load the `fuel2001` dataset in R:**

```
fuel2001 <- read.csv("https://dvats.github.io/assets/fuel2001.csv", row.names = 1)
```

   a. **Fit the linear regression model using maximum likelihood with response `FuelC`. Remember to include an intercept in X. What is your final estimate of β and σ^2?**

   We do the same thing again, but in this case we have to ensure that $(X^T X)$ is stable and invertible.

```r
X <- as.matrix(fuel2001[, -2])
X <- cbind(1, X)
y <- fuel2001$FuelC
n <- length(y)

XtX <- t(X) %*% X
XtX.inv <- qr.solve(XtX, tol = 1e-20)

beta.mle <- XtX.inv %*% t(X) %*% y
beta.mle
```

```
                 [,1]
        -4.902281e+05
Drivers  6.368120e-01
Income   7.690244e+00
Miles    5.850354e+00
MPC      4.561872e+01
Pop     -1.944886e-02
Tax     -2.086607e+04
```

```r
sig2 <- t(y - X%*%beta.mle) %*% (y - X%*%beta.mle)/n
sig2
```

```
             [,1]
[1,] 136947309151
```

b. **For $\lambda = 1$, what is the ridge regression estimator of β?**

We know that the ridge solution is $\hat{\beta}_{\text{Ridge}} = (X^T X + \lambda I_p)^{-1} X^T y$.

```r
lam <- 1
beta.ridge <- solve(t(X) %*% X  + diag(lam, dim(X)[2])) %*% t(X) %*% y
beta.ridge
```

```
                 [,1]
        -9.379124e+04
Drivers  6.336245e-01
Income   1.307651e+00
Miles    5.731114e+00
MPC      3.214120e+01
Pop     -1.673608e-02
Tax     -2.398454e+04
```

3. **Simulating data in R: Let $X \in \mathbb{R}^{n \times p}$ be the design matrix, where all entries in its first column equal one (to form an intercept). Let $x_{i,j}$ be the (i,j)th element of X. For the i^{th}**

case, $x_{i1} = 1$ and $x_{i2}, \dots, x_{ip}$ are the values of the $p-1$ predictors. Let y_i be the response for the ith case and define $y = (y_1, \dots, y_n)^T$. The model assumes that y is a realization of the random vector:

$$
Y \sim N_n(X\beta_*, \sigma_*^2 I_n)\,,
$$

where $\beta_* \in \mathbb{R}^p$ are unknown regression coefficients and $\sigma_*^2 > 0$ is the unknown variance. We would like to *generate data* that actually follows the following model. This is useful when building too methods and different estimation techniques for β.

a. **Study the code below and understand how the data is being generated according to the model:**

```
    set.seed(1)
    n <- 50
    p <- 5
    sigma2.star <- 1/2
    beta.star <- rnorm(p)

    X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))
    y <-  X %*% beta.star + rnorm(n, mean = 0, sd = sqrt(sigma2.star))
```

b. **Having generated the above `(y,X)`, obtain the MLE of β. Is the MLE roughly close to β^*? What happens when you increase $n = 500$?**

```
# MLE
beta.mle <- solve(t(X) %*% X) %*% t(X) %*% y
cbind(beta.mle, beta.star)
```

```
                beta.star
[1,] -0.6209133 -0.6264538
[2,]  0.2312506  0.1836433
[3,] -0.7667199 -0.8356286
[4,]  1.7272425  1.5952808
[5,]  0.2038577  0.3295078
```

The above is close but not too close. We can now increase the data to $n = 500$.

```
n <- 500
X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))
y <-  X %*% beta.star + rnorm(n, mean = 0, sd = sqrt(sigma2.star))

beta.mle <- solve(t(X) %*% X) %*% t(X) %*% y
cbind(beta.mle, beta.star)
```

```
           beta.star
[1,] -0.6037567 -0.6264538
[2,]  0.2313300  0.1836433
[3,] -0.8396466 -0.8356286
[4,]  1.6169639  1.5952808
[5,]  0.3444908  0.3295078
```

The estimator is much closer than before.

c. **Repeat the data generation process, but now change $p = 100$ and keep $n = 50$. Can you find the traditional MLE in this case?**

In this case, since $p > n$, $X^T X$ will not be invertible, so we cannot find the tradition MLE.

```r
set.seed(1)
n <- 50
p <- 100
sigma2.star <- 1/2
beta.star <- rnorm(p)

X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))
y <-  X %*% beta.star + rnorm(n, mean = 0, sd = sqrt(sigma2.star))
```

d. **For the above data find the ridge regression estimator of β for $\lambda = 0.01, 0.1, 1, 10$.**

Although we cannot find the MLE, we can find the ridge regression estimator when $p > n$. Doing that for different values of λ

```r
beta.01  <- solve(t(X) %*% X  + diag(.01, p)) %*% t(X) %*% y
beta.1   <- solve(t(X) %*% X  + diag(.1, p)) %*% t(X) %*% y
beta1    <- solve(t(X) %*% X  + diag(1, p)) %*% t(X) %*% y
beta10   <- solve(t(X) %*% X  + diag(10, p)) %*% t(X) %*% y

Allbetas <- cbind(beta.01,beta.1, beta1, beta10)
head(Allbetas, 10) # only looking at the first rows columns
```

```
            [,1]        [,2]        [,3]         [,4]
[1,]  0.27196872  0.27135049  0.26549662  0.226051532
[2,]  0.09680004  0.09577826  0.08575324  0.007474578
[3,] -0.94376703 -0.94231652 -0.92813991 -0.814301669
[4,]  0.74930506  0.74861633  0.74226009  0.703487016
[5,] -0.78515762 -0.78353942 -0.76755411 -0.635366192
[6,] -0.61452871 -0.61221345 -0.59066796 -0.461015082
[7,]  0.54167557  0.54293846  0.55423841  0.598769112
[8,]  1.21353371  1.20953415  1.17176116  0.924644637
[9,]  0.76317166  0.76431894  0.77470956  0.818672045
```

```
[10,] -0.60347190 -0.60174399 -0.58473630 -0.446821186
```

We see that for large value of λ, the β estimates are closer to 0.

4. **Write a Newton-Raphson code to find the MLE of α for Gamma $(\alpha, 1)$ distribution. That is, suppose $X_1, X_2, \ldots, X_n \stackrel{iid}{\sim}$ Gamma$(\alpha, 1)$, then write a function to obtain $\hat{\alpha}_{MLE}$.**

   In order to implement this, you will need data that is from $Gamma(\alpha, 1)$. You may use the following:

```
set.seed(100)
alpha <- 5 #true value of alpha
n <- 10 # actual data size is small first
dat <- rgamma(n, shape = alpha, rate = 1)
```

   The above generates $n = 10$ observations. Use `dat` to obtain the MLE of α. You will need the `pracma` library in R to calculate the derivatives of $\Gamma(\cdot)$ function.

```
library(pracma)  #for psi function
?psi
```

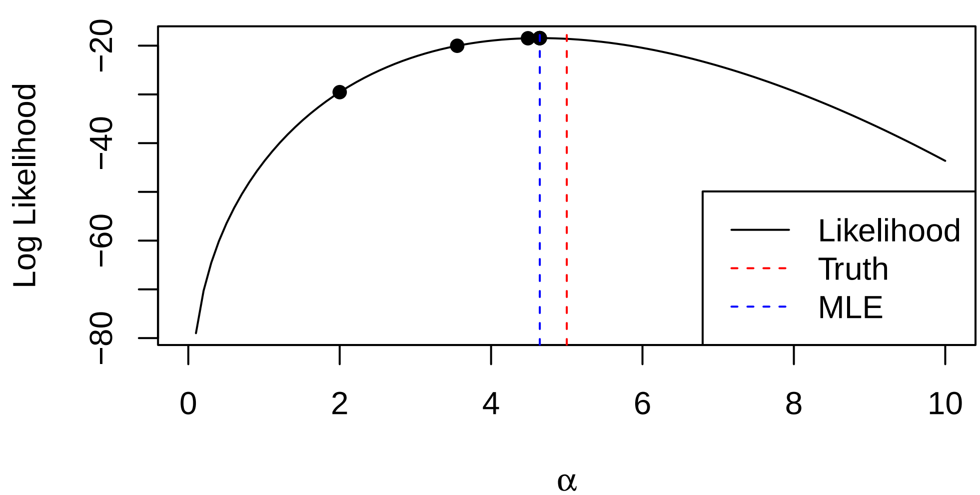
   Recall the theory for this does in class. We will now implement the algorithm.

```
####################################################
### MLE for Gamma(alpha, 1)
####################################################
set.seed(100)
library(pracma)  #for psi function

####################################################

# will save alpha_k sequences
alpha_newton <- numeric()
epsilon <- 1e-8  #some tolerance level preset
alpha_newton[1] <- 2  #alpha_0
count <- 1
tol <- 100 # large number
while(tol > epsilon)
{
  count <- count + 1

  #first derivative
  f.prime <- -n*psi(k = 0, alpha_newton[count - 1]) + sum(log(dat))

  #second derivative
```

```r
  f.dprime <- -n*psi(k = 1, alpha_newton[count - 1])
  alpha_newton[count] <- alpha_newton[count - 1] - f.prime/f.dprime
  tol <- abs(alpha_newton[count] - alpha_newton[count-1])
}
alpha_newton
```

```
[1] 2.000000 3.552357 4.487264 4.640581 4.643535 4.643536 4.643536
```

```r
# The NR methods estimates the MLE. Here the
# blue and red lines will not match because
# the data is not large enough for the consistency of
# the MLE to kick in.
#Plot the log.likelihood for different values of alpha
alpha.grid <- seq(0, 10, length = 100)
log.like <- numeric(length = 100)
for(i in 1:100)
{
  log.like[i] <- sum(dgamma(dat, shape = alpha.grid[i], log = TRUE))
}
plot(alpha.grid, log.like, type = 'l', xlab = expression(alpha), ylab = "Log Likelihood")
abline(v = alpha, col = "red", lty = 2)
for(t in 1:count)
{
  points(alpha_newton[t], sum(dgamma(dat, shape = alpha_newton[t], log = TRUE)), pch = 16)
}
abline(v = tail(alpha_newton[count]), col = "blue", lty = 2)
legend("bottomright", legend = c("Likelihood", "Truth", "MLE"), lty = c(1,2,2), col = c("black",
```



5. **Using the Newton-Raphson algorithm to maximize objective function**

$$f(x) = \cos(x) \quad x \in [-\pi, 3\pi].$$

Our task is to find:

$$x^* = \arg \max_{x \in [-\pi, 3\pi]} \cos(x)$$

In this simple problem, we actually already know that the maxima occurs at two points: $x = 0, 2\pi$, both. Thus our algorithms are expected to converge to either of those two points. In order to implement Newton-Raphson, we need the gradient and the second derivative of the objective function. Let's find that first.

$$f'(x) = -\sin(x) \qquad f''(x) = -\cos(x)$$

Note that since $\cos(x)$ takes both positive and negative values from $[-\pi, 3\pi]$, the objective function is not concave. This means that Newton-Raphson can converge to a local minima as well! Let us write functions to calculate the gradient and second derivative.

```
f.grad <- function(x)  -sin(x)
f.hessian <- function(x) -cos(x)
```

Since the function is simple, the gradients and derivatives are simple to code, as seen above. Now let us first implement Newton-Raphson. Since Newton-Raphson is not guaranteed to converge, we have to be smart about choosing the starting value.

```
tol <- 1e-10
compare <- 100
iter <- 1

xk <- c()  # will store sequence here
xk[1] <- .5 # starting value
while(compare > tol)
{
  iter <- iter + 1  # tracking iterations
  gradient <- f.grad(xk[iter - 1])
  hessian <- f.hessian(xk[iter - 1])
  xk[iter] <- xk[iter - 1] - gradient/hessian

  compare <- abs(gradient)
}
iter
```

```
[1] 5
```

```
xk[iter] # N-R last iterate.
```

```
[1] 0
```

Starting from $x_0 = .5$, the above NR converges to 0. If we change the starting value, to say $\pi/2$, then since $\pi/2$ is an inflection point, the second derivative is 0 and the NR algorithm becomes unstable! (try it). Also change the starting value to a little more than $\pi/2$, say, $\pi/2 + .1$, and you will notice that the algorithm converges to a minima.

We can plot this as well

```
foo <- seq(-pi, 3*pi, length = 1e3)
plot(foo, cos(foo), type = 'l')
# putting the iterates in light color
points(xk, cos(xk), pch = 16, col = adjustcolor("blue", alpha.f = .3))
points(xk[iter], cos(xk[iter]), pch = 16, col = "blue")  # putting the final value in dark
```