Gov 2018 Quiz: Newton-Raphson Method & Post Double Selection

Sooahn Shin

February 15, 2022

Please upload answers to all questions (including computational questions, any related graphics and R code *with comments*), in .rmd *and* knitted pdf forms to Gradescope by Thursday 11:59 pm. For any problems that require calculations, please show your work. Finally, note that only clarification questions can be asked for the quiz and you cannot consult your answer with other students.

Part 1: Newton-Raphson Method

In this part, we seek to solve the optimization problem $\min_x F(x)$ where $F : \mathbb{R} \to \mathbb{R}$ is a twice differentiable function. We will use Newton-Raphson method to find a solution for $f(x) = 0$ (i.e., the root(s) of $f(x)$) where f is the derivative of F.

Question 1: Newton-Raphson method function (2 pts)

Write a function `newton.raphson` with the following characteristics:

- 4 arguments:
  - f: Function for which we are searching for a solution $f(x) = 0$.
  - x_0: Initial guess for a solution $f(x) = 0$.
  - Tolerance (e.g., 1×10^{-5}): If $x_{(i+1)} - x_{(i)} <$ tolerance, then stop the iterations. In other words, stop when the improvement drops below a threshold.
  - Maximum number of iterations (e.g., 10000)
- For i-th iteration, update your guess to be $x_{(i+1)} = x_{(i)} - \frac{f(x_{(i)})}{f'(x_{(i)})}$, where f' is a derivative of f

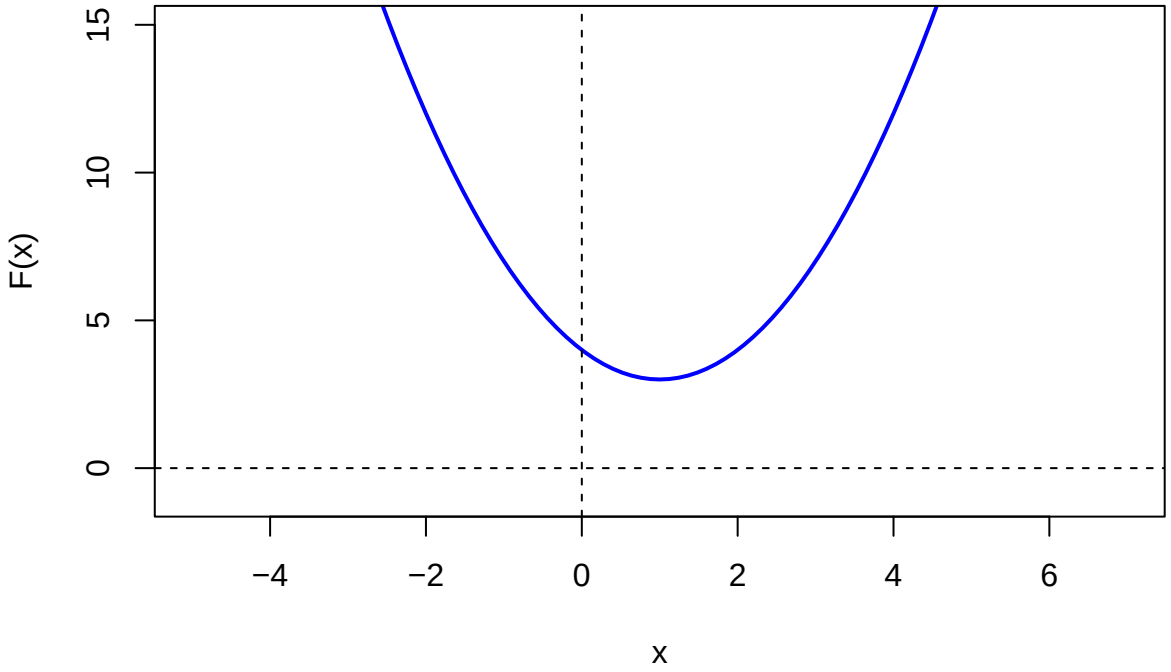
You may use `f.prime(x)` from Problem Set 1.

```
# Defining function
newton.raphson <- function(f, x, tol, max_iter) {
  # Counting number of loops
  n <- 0
  # Initially setting step to be greater than tolerance to enter the while loop, it will be properly de
  step <- tol + 10
  # Creating fprime function inside this function
  df <- function(x, d = 0.001) {
    (f(x+d) - f(x-d)) / (2*d)
  }
  # keep looping until step gets very small or n exceeds max iterations
  while(abs(step) > tol && n < max_iter) {
    # Increment loop
    n <- n + 1
    # Break the loop if we have already hit 0
    if(df(x) == 0) {
```

```
      break
    }
    # Create the step of the function
    step <- -(f(x)/df(x))
    # Apply to X
    x <- x + step

  }
  # Return x, f(x), and n
  return(list(x, f(x), n))
}
```

Question 2: Example with one root (1 pt)

Suppose we want to find a global minimum of $F_1(x) = x^2 - 2x + 4$ (see figure below). Create a function called `f1` that computes the first derivative of F_1 at given point x. Then using the function `newton.raphson` with the initial guess of $x_0 = -2$, find the root of f_1.



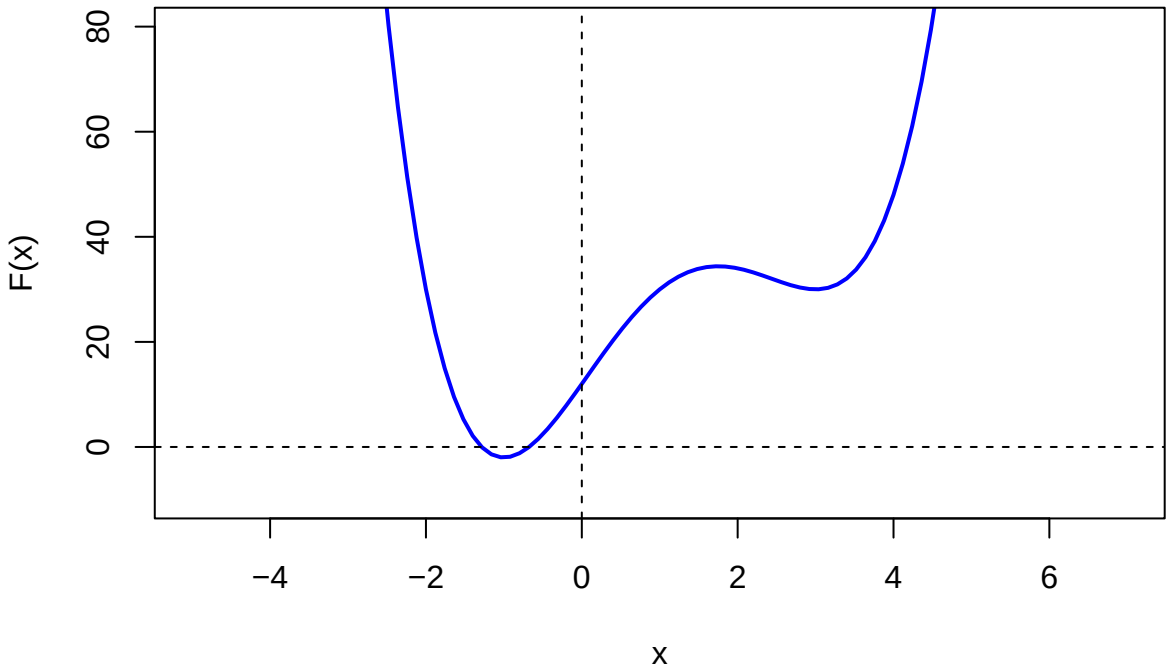

```
f1 <- function(x) {
  return((2 * x) - 2)
}

newton.raphson(f1, -2, 0.00001, 1000)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 0
##
## [[3]]
## [1] 2
```

At this level of tolerance, the function took 2 steps to find x = 1 as the global minimum, which is reflected in the plot above. f(x) is 0, confirming that this is the minimum.

Question 3: Example with two roots (1 pt)

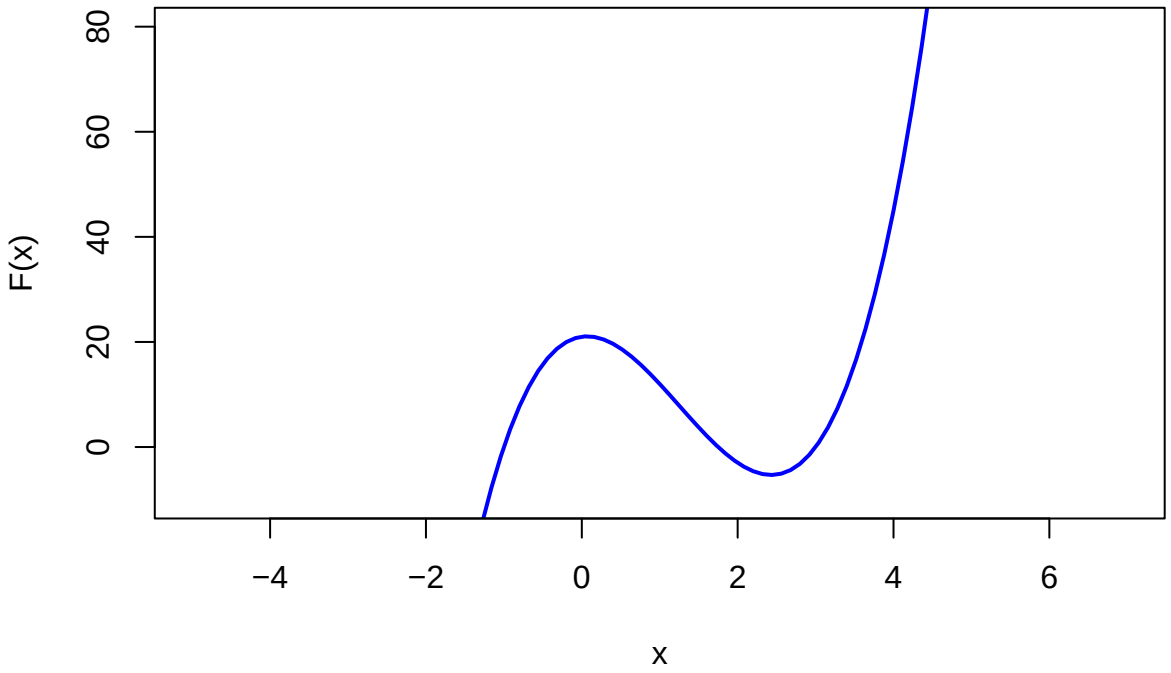
Now, suppose we want to find a global minimum of $F_2(x) = x^4 - 5x^3 + x^2 + 21x + 30$ (see figure below). Create a function called `f2` that computes the first derivative of F_2 at given point x. Then run the function `newton.raphson` with the initival guess of $x_0 = 4$. Do you think the approximated root of the function is the global minimum of F_2? If not, what can be done to find the global minimum?




```
f2 <- function(x) {
  4*x^3 - 15*x^2 + 2*x + 21
}

newton.raphson(f2, 4, 0.00001, 1000)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 3.552714e-15
##
## [[3]]
## [1] 6
```

```
curve(f2, col = 'blue', lwd = 2, xlim=c(-5,7), ylim=c(-10,80), ylab='F(x)')
```



At this level of tolerance, the function took 6 steps to find x = 3 as the minimum. As we can see in the F2 plot and the f2 plot, 3 is a minimum in this function. f(x) is essentially 0, confirming that this is a minimum. However, as we can see from the F2 plot, this is not the global minimum, as that is located around -1 in the F2 plot. To properly identify the global minimum and not just the local minimum of x = 3, we could take a similar approach to Pset1 and generate thousands of starting points for the newton.raphson function, and then generate a histogram of the minimums they find. This method would likely lead us to -1 being the global minimum, since the starting points will be randomly distributed across the window of this function. We could also forego simulations and pick the starting point more strategically based on what we see in the plot of the function (in this case, starting closer to -1 instead of 4)

Part 2: Post Double Selection

In this part, we will conduct simulation study of post double selection from Belloni, Chernozhukov, and Hansen (2014). Using a synthetic data, we will investigate the following (partially) linear model with high-dimensional covariates.

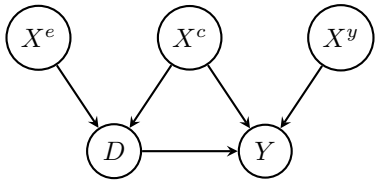
$$Y_i = D_i\alpha + X_i^c\beta + X_i^y\delta + \epsilon_{1i}$$
$$D_i = X_i^c\gamma + X_i^e\sigma + \epsilon_{2i}$$

where

- Y_i is the outcome variable (n by 1 vector);
- D_i is the policy (treatment) variable (n by 1 vector);
- X_i^c is a set of confounders we would like to control (X^c is n by $3p_c - 1$ matrix);
- X_i^y is a set of covariates that only has an impact on Y_i (X^y is n by $3p_y - 1$ matrix);
- X_i^e is a set of exogenous variables that only affects D_i (X^e is n by $3p_e - 1$ matrix);
- $\mathbb{E}[\epsilon_{1i} \mid D_i, X_i^c, X_i^y] = 0$ and $\mathbb{E}[\epsilon_{2i} \mid X_i^c, X_i^e] = 0$.

Under the assumption of (approximate) sparsity and some regularity conditions, we will estimate α, the impact of D_i on Y_i. We have a high-dimensional covariates $X_i = X_i^c \cup X_i^y \cup X_i^e \cup X_i^u$ from which we may select X_i^c, X_i^y, and X_i^e. Note that X_i^u does not affect D_i nor Y_i; we will use exact sparse regime instead of

approximate sparse regime in the simulation study for simplicity. We will compare two different estimation strategies, one with usual post model selection and the other with post double selection. Please check the lecture slides and the paper for more details.



A note on Causal Inference: Note that we would need additional assumptions to interpret the regression coefficient α as an unbiased estimator for an Average Treatment Effect (ATE). In this question, we will ignore the issues with identification of the causal effect and focus on investigating the post double selection method.

Question 1: Creating synthetic data (1 pt)

We will start by creating a function that generates synthetic data given true parameters. The data generating process (DGP) we will use is as follows:

1. Generate $Z^c_{i1}, \ldots, Z^c_{i,p_c}$, $Z^y_{i1}, \ldots, Z^y_{i,p_y}$, $Z^e_{i1}, \ldots, Z^e_{i,p_y}$, and $Z^u_{i1}, \ldots, Z^u_{i,p}$ as follows:
   - $Z^c_{ik} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, p_c$
   - $Z^y_{ik} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, p_y$
   - $Z^e_{ik} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, p_e$
   - $Z^u_{ik} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, p_u$
2. Let X^c_i, X^y_i, X^e_i, and X^u_i be a function of $(Z^c_{i1}, \ldots, Z^c_{i,p_c})$, $(Z^y_{i1}, \ldots, Z^y_{i,p_y})$, $(Z^e_{i1}, \ldots, Z^e_{i,p_e})$, and $(Z^u_{i1}, \ldots, Z^u_{i,p_u})$ respectively as follows:
   - $X^c_i = (Z^c_{i1}, Z^c_{i2}, \ldots, Z^c_{i,p_c}, {Z^c_{i1}}^2, {Z^c_{i2}}^2, \ldots, {Z^c_{i,p_c}}^2, Z^c_{i1} \cdot Z^c_{i2}, Z^c_{i2} \cdot Z^c_{i3}, \ldots, Z^c_{i,p_c-1} \cdot Z^c_{i,p_c})^\top$
   - $X^y_i = (Z^y_{i1}, Z^y_{i2}, \ldots, Z^y_{i,p_y}, {Z^y_{i1}}^2, {Z^y_{i2}}^2, \ldots, {Z^y_{i,p_y}}^2, Z^y_{i1} \cdot Z^y_{i2}, Z^y_{i2} \cdot Z^y_{i3}, \ldots, Z^y_{i,p_y-1} \cdot Z^y_{i,p_y})^\top$
   - $X^e_i = (Z^e_{i1}, Z^e_{i2}, \ldots, Z^e_{i,p_e}, {Z^e_{i1}}^2, {Z^e_{i2}}^2, \ldots, {Z^e_{i,p_e}}^2, Z^e_{i1} \cdot Z^e_{i2}, Z^e_{i2} \cdot Z^e_{i3}, \ldots, Z^e_{i,p_e-1} \cdot Z^e_{i,p_e})^\top$
   - $X^u_i = (Z^u_{i1}, Z^u_{i2}, \ldots, Z^u_{i,p_u}, {Z^u_{i1}}^2, {Z^u_{i2}}^2, \ldots, {Z^u_{i,p_u}}^2, Z^u_{i1} \cdot Z^u_{i2}, Z^u_{i2} \cdot Z^u_{i3}, \ldots, Z^u_{i,p_u-1} \cdot Z^u_{i,p_u})^\top$
3. Let $X_i = \begin{pmatrix} X^c_i & X^y_i & X^e_i & X^u_i \end{pmatrix}$
4. Generate regression coefficients as follows:
   - $\alpha = 0.5$
   - $\beta_j \sim \mathcal{N}(0.1, 0.01^2)$ for $j = 1, \ldots, (3p_c - 1)$
   - $\delta_j \sim \mathcal{N}(0, 0.01^2)$ for $j = 1, \ldots, (3p_y - 1)$
   - $\gamma_j \sim \mathcal{N}(0.8, 0.01^2)$ for $j = 1, \ldots, (3p_c - 1)$
   - $\sigma_j \sim \mathcal{N}(0, 0.01^2)$ for $j = 1, \ldots, (3p_e - 1)$
5. Generate random noise as follows:
   - $\epsilon_{1i} \sim \mathcal{N}(0,1)$ for $i = 1, \ldots, n$
   - $\epsilon_{2i} \sim \mathcal{N}(0,1)$ for $i = 1, \ldots, n$
6. Let $D_i = X^c_i \gamma + X^e_i \sigma + \epsilon_{2i}$ and $Y_i = D_i \alpha + X^c_i \beta + X^y_i \delta + \epsilon_{1i}$.

Using this DGP and the sample codes below, write a function `generate_synth_data` which generates the data and parameters.

```r
generate_synth_data <- function(n = 100, # number of observations
                                pc = 5, # dimensionality (size of dimensions) of Z_i^c
                                py = 2, # dimensionality of Z_i^y
                                pe = 2,  # dimensionality of Z_i^e
                                pu = 30,  # dimensionality of Z_i^u
                                alpha = 0.5,
                                beta = NULL,
                                delta = NULL,
                                gamma = NULL,
                                sigma = NULL,
                                seed.number = 2018
                                ) {
  set.seed(seed.number)

  #-- TODO 1: Generate Z^c, Z^y, Z^e, and Z^u (Hint: use rnorm())
  #Zc = matrix(NA, nrow = n, ncol = pc)
  #Zy = matrix(NA, nrow = n, ncol = py)
  #Ze = matrix(NA, nrow = n, ncol = pe)
  #Zu = matrix(NA, nrow = n, ncol = pu)

  # Your code here
  Zc <- matrix(rnorm(n * pc), nrow = n, ncol = pc)
  Zy <- matrix(rnorm(n * py), nrow = n, ncol = py)
  Ze <- matrix(rnorm(n * pe), nrow = n, ncol = pe)
  Zu <- matrix(rnorm(n * pu), nrow = n, ncol = pu)

  #-- TODO 2: Create X^c, X^y, X^e, and X^u
  #Xc = matrix(NA, nrow = n, ncol = pc*3-1)
  #Xy = matrix(NA, nrow = n, ncol = py*3-1)
  #Xe = matrix(NA, nrow = n, ncol = pe*3-1)
  #Xu = matrix(NA, nrow = n, ncol = pu*3-1)

  Xc = cbind(Zc, Zc^2, Zc[,1:4] * Zc[,2:5])
  Xy = cbind(Zy, Zy^2, Zy[,1] * Zy[,2])
  Xe = cbind(Ze, Ze^2, Ze[,1] * Ze[,2])
  Xu = cbind(Zu, Zu^2, Zu[,1:29] * Zu[,2:30])

  # Your codes here
  colnames(Xc) = paste0("Xc", 1:(pc*3-1))
  colnames(Xy) = paste0("Xy", 1:(py*3-1))
  colnames(Xe) = paste0("Xe", 1:(pe*3-1))
  colnames(Xu) = paste0("Xu", 1:(pu*3-1))

  #-- 3: Create X
  X = cbind(Xc, Xy, Xe, Xu)


  #-- TODO 4: Generate regression coefficients
  if (is.null(alpha)) {
    alpha = rnorm(1, mean = 0, sd = 2) # example
  }
  if (is.null(beta)) {
    # Your codes here
```

```
    beta = matrix(rnorm(pc*3-1, mean = 0.1, sd = 0.01^2), ncol = 1)
  }
  if (is.null(delta)) {
    # Your codes here
    delta = matrix(rnorm(py*3-1, mean = 0, sd = 0.01^2), ncol = 1)
  }
  if (is.null(gamma)) {
    # Your codes here
    gamma = matrix(rnorm(pc*3-1, mean = 0.8, sd = 0.01^2), ncol = 1)
  }
  if (is.null(sigma)) {
    # Your codes here
    sigma = matrix(rnorm(pe*3-1, mean = 0, sd = 0.01^2), ncol = 1)
  }

  #-- TODO 5: Generate random noise
  #epsilon1 = rep(NA, n)
  #epsilon2 = rep(NA, n)
  # Your codes here
  epsilon1 = matrix(rnorm(1, mean = 0, sd = 1), n)
  epsilon2 = matrix(rnorm(1, mean = 0, sd = 1), n)

  #-- TODO 6: Create D and Y
  D = rep(NA, n)
  Y = rep(NA, n)

  D = Xc %*% gamma + Xe %*% sigma + epsilon2
  Y = D * alpha + Xc %*% beta + Xy %*% delta + epsilon1

  colnames(D) = "D"
  colnames(Y) = "Y"

  # cat("## num of observations:",n,"\n## num of predictors:",ncol(X),"\n")

  res = list(X = X, D = D, Y = Y,
             alpha = alpha, beta = beta, delta = delta, gamma = gamma, sigma = sigma)
  return(res)
}
```

Question 2: Lasso methods (2 pts)

Generate synthetic data using the function `generate_synth_data` with following arguments:

```
n = 100; pc = 5; py = 2; pe = 2; pu = 30
alpha = 0.5
set.seed(2018)
beta = matrix(rnorm(pc*3-1, mean = 0.1, sd = 0.01), ncol = 1)
delta = matrix(rnorm(py*3-1, mean = 0, sd = 0.01), ncol = 1)
gamma = matrix(rnorm(pc*3-1, mean = 0.8, sd = 0.01), ncol = 1)
sigma = matrix(rnorm(pe*3-1, mean = 0, sd = 0.01), ncol = 1)

# Calling function with new fields
res <- generate_synth_data(beta = beta, delta = delta, gamma = gamma, sigma = sigma)
```

Estimate α using `glmnet` with two different methods: post single lasso and post double lasso. For post double lasso, use two different packages for the estimation: `glmnet` and `hdm` (the latter is written by the authors of the paper). Briefly discuss the results.

Hint: For `glmnet`, use `cv.glmnet` in place of `glmnet` — this is a cross-validated glmnet — and `coef(your-glmnet-object, s = "lambda.1se")` functions to select λ value. For `hdm`, the syntax is as follows: `hdm::rlassoEffect(x=your-X,d=your-D,y=your-Y,method="double selection")` Note that the estimates may differ between `glmnet` and `hdm` due to different default methods of selecting for λ.

```
set.seed(2018)
library(hdm)
library(glmnet)
```

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

```
## Loaded glmnet 4.1-3
```

```
# Pulling out X, Y, and D from the generate synth data function
X <- res$X
D <- res$D
Y <- res$Y

### POST SINGLE GLMNET
# Cbinding X and D for single selection since we are treating it as the same set of predictors
XD <- cbind(X, D)
# Post single selection with cv.glmnet, alpha = 1 for lasso
post_single <- cv.glmnet(XD, Y, alpha = 1)
# Printing ALPHA
tail(coef(post_single, s = "lambda.1se"),1)
```

```
## 1 x 1 sparse Matrix of class "dgCMatrix"
##          s1
## D 0.6129666
```

```
### POST DOUBLE GLMNET:
### Citation: referred to this article that I found on Google: https://insightr.wordpress.com/2017/05/1
# First, predict X on Y
model1 <- cv.glmnet(X, Y, alpha = 1)
# Selecting only the predictors of X on Y that are significant at the 0.01 level
model1_select <- which(coef(model1, s = "lambda.1se")[-1]<0.01)
# Second, redict X on D
model2 <- cv.glmnet(X, D, alpha = 1)
# Selecting only the predictors of X on D that are significant at the 0.01 level
model2_select <- which(coef(model2, s = "lambda.1se")[-1]<0.01)
# Selecting only the predictors that are present in both subsets of predictors
select_vars <- sort(union(model1_select,model2_select))
# Ordering and sorting the selected indices
post_double_select=sort(union(model1_select, model2_select))
# Filtering X based on the selected indices and binding with D
dXD <- cbind(X[,post_double_select], D)
# Make a glmnet with this dataset
post_double <- cv.glmnet(dXD, Y, alpha = 1)
# Printing ALPHA
tail(coef(post_double, s = "lambda.1se"),1)
```

```
## 1 x 1 sparse Matrix of class "dgCMatrix"
##          s1
```

```
## D 0.6129666
```

```
### POST DOUBLE HDM:
coef(rlassoEffect(d=D, x=X, y=Y, method="double selection"))
```

```
##         D
## 0.6105658
```

Using post single lasso, alpha was estimated to be 0.6113. Using post double lasso with the glmnet package, alpha was estimated to be 0.6130. Using post double lasso with the hdm package, alpha was estimated to be 0.6106. These values are all very similar, indicating that each different method yields a similar result for this scenario.

Question 3: Monte-Carlo simulation (3 pts)

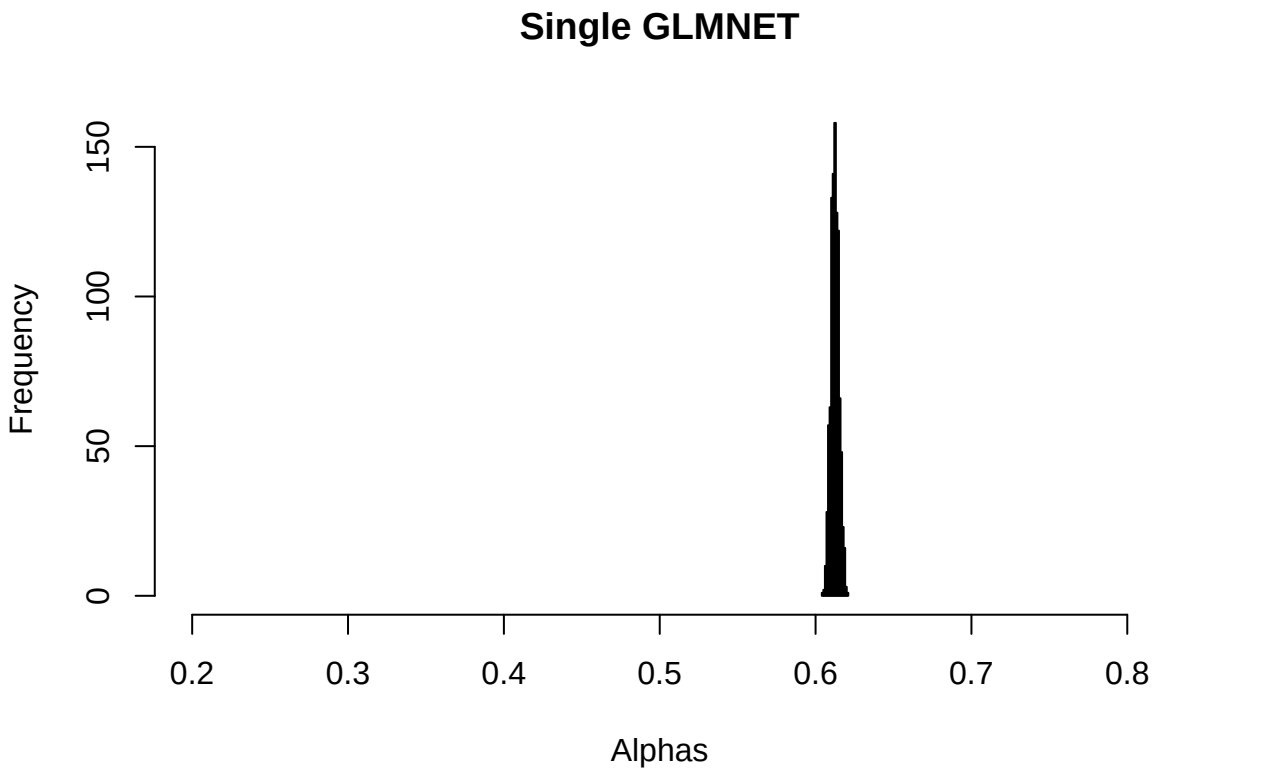
Note that the result from Question 2 is based on a single simulation. Now, we will conduct a Monte-Carlo simulation where we generate synthetic data multiple times (e.g., 1000 iterations) and compare the distribution of α for each method. Plot the histogram of three distributions of α (single LASSO, double LASSO with `glmnet`, and double LASSO with `hdm`). Briefly discuss the results and the difference between two methods.

Hint: Make sure to set different seed number for each iteration while using the same parameters given in the previous question.

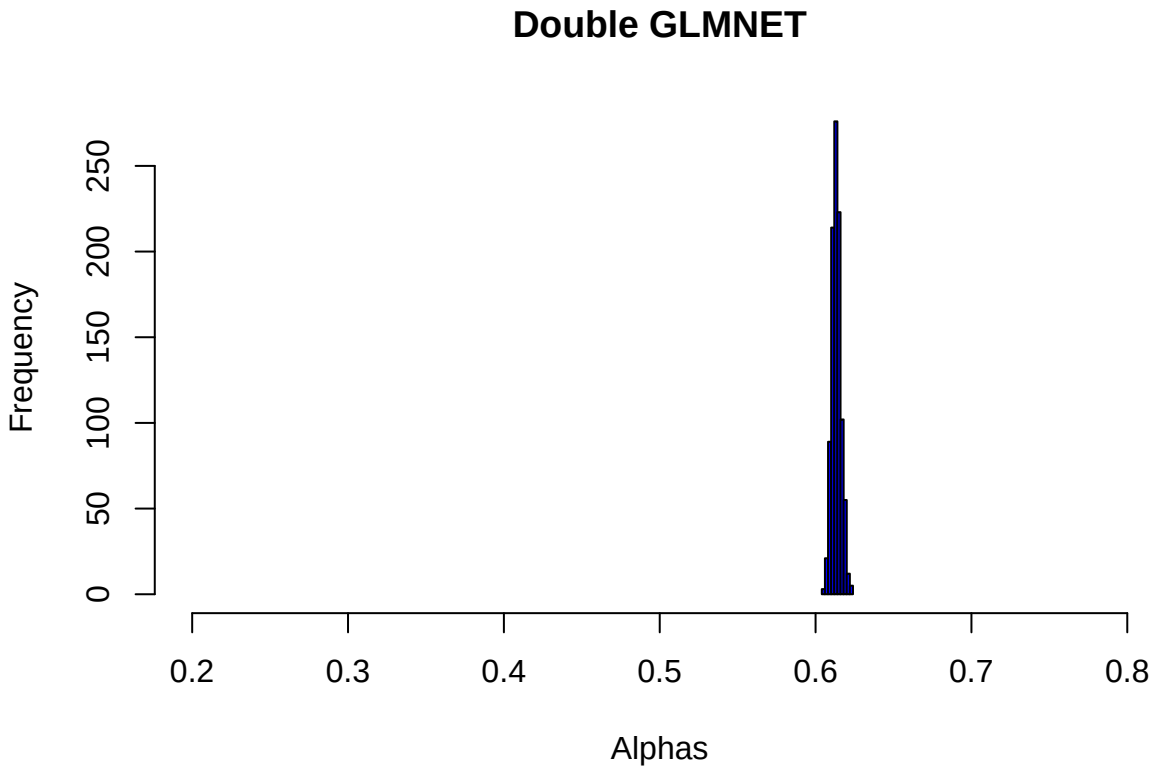
```
i <- 0
seed <- 1
glmnet1 <- c()
glmnet2 <- c()
hdm1 <- c()

while(i < 1000) {
  set.seed(seed)
  dat <- generate_synth_data(beta = beta,
                             delta = delta,
                             gamma = gamma,
                             sigma = sigma,
                             seed.number = seed)

  # Pulling out X, Y, and D from the generate synth data function
    X <- dat$X
    D <- dat$D
    Y <- dat$Y

    ### POST SINGLE GLMNET
    # Cbinding X and D for single selection since we are treating it as the same set of predictors
    XD <- cbind(X, D)
    # Post single selection with cv.glmnet, alpha = 1 for lasso
    post_single <- cv.glmnet(XD, Y, alpha = 1)
    # Adding alpha to list
    glmnet1<- c(glmnet1, tail(coef(post_single, s = "lambda.1se"),1)[1])


    ### POST DOUBLE GLMNET:
    # First, predict X on Y
    model1 <- cv.glmnet(X, Y, alpha = 1)
    # Selecting only the predictors of X on Y that are significant at the 0.01 level
    model1_select <- which(coef(model1, s = "lambda.1se")[-1]<0.01)
    # Second, redict X on D
```

```
    model2 <- cv.glmnet(X, D, alpha = 1)
    # Selecting only the predictors of X on D that are significant at the 0.01 level
    model2_select <- which(coef(model2, s = "lambda.1se")[-1]<0.01)
    # Selecting only the predictors that are present in both subsets of predictors
    select_vars <- sort(union(model1_select,model2_select))
    # Ordering and sorting the selected indices
    post_double_select=sort(union(model1_select, model2_select))
    # Filtering X based on the selected indices and binding with D
    dXD <- cbind(X[,post_double_select], D)
    # Make a glmnet with this dataset
    post_double <- cv.glmnet(dXD, Y, alpha = 1)
    # Adding ALPHA to list
    glmnet2 <- c(glmnet2, tail(coef(post_double, s = "lambda.1se"),1)[1])

    ### POST DOUBLE HDM:
    # Adding alpha to list
    hdm1 <- c(hdm1, coef(rlassoEffect(d=D, x=X, y=Y, method="double selection"))[[1]])


    i <- i + 1
    seed <- seed + runif(1, 0, 100)
}
```

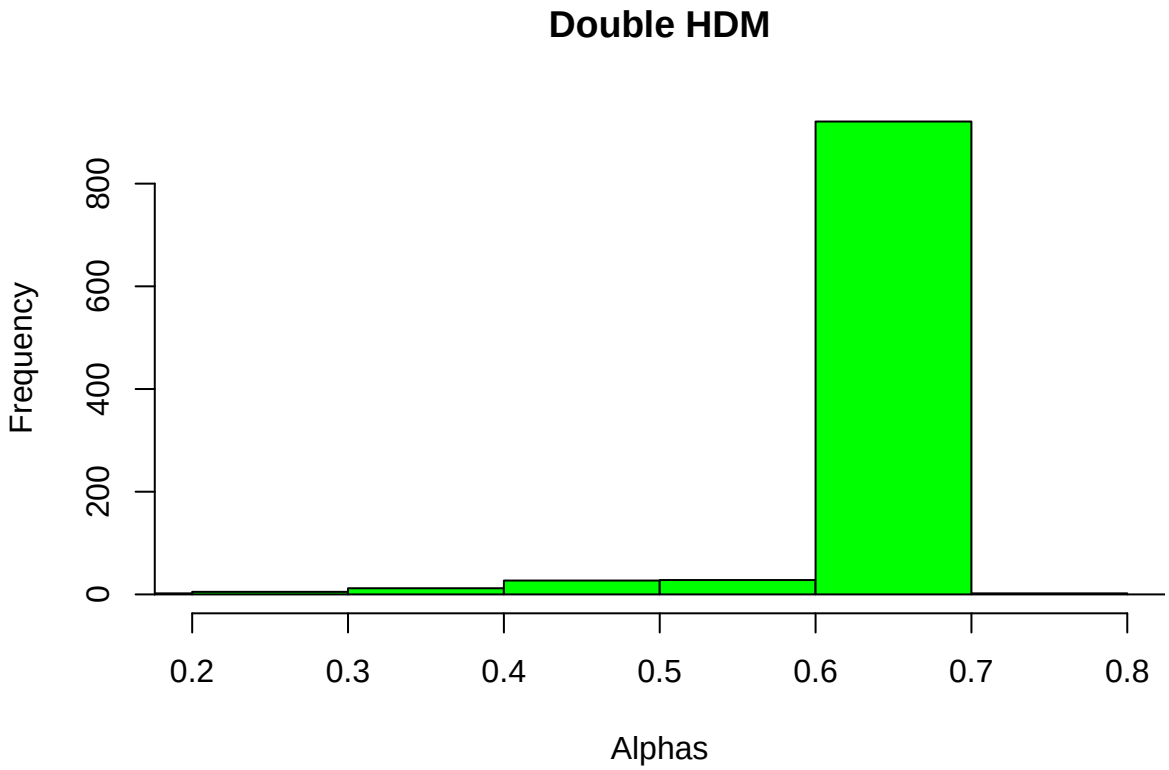
```
hist(glmnet1, col='red', xlim=c(0.2, 0.8), main='Single GLMNET', xlab='Alphas')
```




```
hist(glmnet2, col='blue', xlim=c(0.2, 0.8), main='Double GLMNET', xlab='Alphas')
```




```
hist(hdm1, col='green', xlim=c(0.2, 0.8), main='Double HDM', xlab='Alphas')
```



As we can see in the results, both single and double selection GLMNET spiked around the same area (roughly 0.61) across many different seeds. The HDM model also spiked at this value, with a (very) slightly wider range. The double selection method is different from the single selection method in that predictors in X are tossed out if they are not significant in predicting Y or D. Despite the difference in the methods, the results are very

similar in this scenario.