STAT 542: Homework 5

Spring 2022, by Kamila Makhambetova (kamilam3)

Due: Thursday, Feb 24, 11:59 PM CT

Instruction

Students are encouraged to work together on homework. However, sharing, copying, or providing any part of a homework solution or code is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to compass2g. No email or hardcopy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat542/homework.html), please refer to the course website.

- What is expected for the submission to **Gradescope**
  - You are required to submit one rendered **PDF** file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file generated by a `.Rmd` file. `.html` format cannot be accepted.
  - Please follow the instructions on Gradescope to select corresponding PDF pages for each question.
- Please note that your homework file is a **PDF** report instead of a messy collection of R codes. This report should **include**:
  - Your Name and NetID. (Replace `Ruoqing Zhu(rqzhu)` by your name and NetID if you are using this template).
  - Make all of your `R` code chunks visible for grading.
  - Relevant outputs from your `R` code chunks that support your answers.
  - Provide clear answers or conclusions for each question. For example, you could start with `Answer: I fit SVM with the following choice of tuning parameters ...`
  - Many assignments require your own implementation of algorithms. **Basic comments are strongly encouraged** to explain the logic to our graders. However, line-by-line code comments are unnecessary.
- Requirements regarding the `.Rmd` file.
  - You do **NOT** need to submit `Rmd` files. However, your PDF file should be rendered directly from it.
  - Make sure that you **set random seeds** for simulation or randomized algorithms so that the results are reproducible. If a specific seed number is not provided in the homework, you can consider using your NetID.
  - For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.

About HW5

We utilize the coordinate descent algorithm introduced in the class to implement the entire Lasso solution. For coordinate descent, you may also want to review HW4. This HW involves two steps: in the first step, we solve the solution for a fixed λ value, while in the second step, we consider a sequence of λ values and solve it using the path-wise coordinate descent.

Question 1 [50 Points] Lasso solution for fixed λ

For this question, you cannot use functions from any additional library, except the `MASS` package, which is used to generate multivariate normal data. Following HW4, we use the this version of the objective function:

$$\arg\min_{\beta} \frac{1}{n}\|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

The following data is used to fit this model. You can consider using similar functions in Python if needed. We use

```
library(MASS)
set.seed(10)
n = 100
p = 200

# generate data
V = matrix(0.3, p, p)
diag(V) = 1
X_org = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V))
true_b = c(1:3, -3:-1, rep(0, p-6))
y_org = X_org %*% true_b + rnorm(n)

# pre-scaling and centering X and y
X = scale(X_org)*sqrt(n/(n-1))
y = scale(y_org)*sqrt(n/(n-1))
lambda = 0.3
```

a. [10 pts] State the solution x of the following problem

$$\arg\min_x \ (x-b)^2 + \lambda|x|, \quad \lambda > 0$$

Then, implement a function in the form of `soft_th <- function(b, lambda)` to return the result of the above problem. Note in the coordinate descent discussed in the slides, where b is an OLS estimator, and λ is the penalty parameter. Report the function output for the following testing cases with $\lambda = 0.3$: 1) $b = 1$; 2) $b = -1$; 3) $b = -0.1$.

Solution

Know $x = \hat{B_j}^{lasso}$ and $b = \hat{B_j}^{ols}$

$g(x) = (x-b)^2 + \lambda x$

Take derivative of g(x) with respect to x and equalize to 0.

$g(x) = (x-b)^2 + \lambda x$

$\frac{\partial g(x)}{x} = 2(x-b) \pm \lambda$

$$\hat{B_j}^{lasso} = \begin{cases} b - \frac{\lambda}{2}, & \text{if } b > \frac{\lambda}{2} \\ 0, & \text{if } |b| \leq \frac{\lambda}{2} \\ b + \frac{\lambda}{2}, & \text{if } b < \frac{-\lambda}{2} \end{cases}$$

```
#soft treshold function
soft_th <- function(b, lambda){
  lambda_half=lambda/2
  if(b>lambda_half){
    res=b-lambda_half
  }
  else if(abs(b)<=lambda_half){
    res=0
  }
  else if(b<(-1)*lambda_half){
    res=b+lambda_half
  }
  res
}


#print results
print(paste0("For B_ols=1, B_lasso = ",soft_th(1, 0.3)))
```

```
## [1] "For B_ols=1, B_lasso = 0.85"
```

```
print(paste0("For B_ols=-1, B_lasso = ",soft_th(-1, 0.3)))
```

```
## [1] "For B_ols=-1, B_lasso = -0.85"
```

```
print(paste0("For B_ols=-0.1, B_lasso =",soft_th(-0.1, 0.3)))
```

```
## [1] "For B_ols=-0.1, B_lasso =0"
```

Comment

For $\beta_{ols} = 1, \beta_{lasso} = 0.85$

For $\beta_{ols} = -1, \beta_{lasso} = -0.85$

For $\beta_{ols} = -0.1, \beta_{lasso} = 0$

b. [40 pts] We will use the pre-scale and centered data `X` and `y` for this question, hence no intercept is needed. Write a Lasso algorithm function `myLasso(X, y, lambda, beta_init, tol, maxitr)` , which return two outputs (as a list with two components):

- a vector of β values **without** the intercept
- number of iterations

You need to consider the following while completing this question:

- Do not use functions from any additional library
- Start with a vector `beta_init` : $\boldsymbol{\beta} = \mathbf{0}_{p\times 1}$
- Use the soft-threshold function in the iteration when performing the coordinate-wise update.
- Use the efficient $\mathbf{r}$ updating appraoch (we discussed this in lecture and HW4) in the iteration
- Run your coordinate descent algorithm for a maximum of `maxitr` = 100 iterations. Each iteration should loop through all variables.
- You should implement the early stopping rule with `tol` . This means terminating the algorithm when the $\boldsymbol{\beta}$ value of the current iteration is sufficiently similar to the previous one, i.e., $\|\boldsymbol{\beta}^{(k)} - \boldsymbol{\beta}^{(k-1)}\|^2 \leq \text{tol}$.

Aftering completing your code, run it on the data we generated previously. Provide the following results:

- Print out the first 8 coefficients and the number of iterations.

- Check and compare your answer to the `glmnet` package using the following code. You should report their **first 8 coefficients** and the L_1 norm of the difference $\| \hat{\boldsymbol{\beta}}^{\text{glment}}_{[1:8]} - \hat{\boldsymbol{\beta}}^{\text{yours}}_{[1:8]} \|_1$.

```
  library(glmnet)

  # glmnetfit use a different loss function. Use lambda / 2 as the penalty
  glmnetfit = glmnet(X, y, lambda = lambda / 2, intercept = FALSE)

  #soft treshold function
  soft_th <- function(b, lambda){
  lambda_half=lambda/2
  if(b>lambda_half){
    res=b-lambda_half
  }
  else if(abs(b)<=lambda_half){
    res=0
  }
  else if(b<(-1)*lambda_half){
    res=b+lambda_half
  }
  res
  }

  #myLasso function implementation
  myLasso<-function(X, y, lambda, beta_init, tol, maxitr){
  beta_matrix=matrix(0,ncol(X),maxitr)
  i=1
  beta=beta_init
  beta_lasso=rep(0, ncol(X))

  # calculate k-th beta vectors
  while(i<=maxitr){
    for(j in (1:ncol(X))){
      X_j=X[ ,j]
      X_w_j= X[, -j]
      b_w_j=beta[-j]
      beta[j]=soft_th((t(X_j)%*%(y- X_w_j%*%b_w_j))/(t(X_j)%*%X_j), lambda)
      beta_matrix[j,i]=beta[j]
    }

    if(i>1){
      #stop loop when L1 of the difference  of beta previous and
      #current beta vectors < tolerance

      if(sum(abs(beta_matrix[, i]-beta_matrix[, i-1]))<tol){
        break
      }
    }

    if(i==maxitr){
      break
    }
    i=i+1
  }
  results <- list(beta=beta_matrix[, i], iter= i)
  return( results)
  }

beta_init=rep(0, ncol(X))
results= myLasso(X, y, lambda, beta_init,0.000001, 100)
```

```
L1_norm=sum(abs(glmnetfit$beta[1:8]-results$beta[1:8]))

#print results
print(paste0("From glmnet Lasso B_1 = ",glmnetfit$beta[1]))
```

```
## [1] "From glmnet Lasso B_1 = 0"
```

```
print(paste0("From glmnet Lasso B_2 = ",glmnetfit$beta[2]))
```

```
## [1] "From glmnet Lasso B_2 = 0.158233309004053"
```

```
print(paste0("From glmnet Lasso B_3 = ",glmnetfit$beta[3]))
```

```
## [1] "From glmnet Lasso B_3 = 0.429648258957437"
```

```
print(paste0("From glmnet Lasso B_4 = ",glmnetfit$beta[4]))
```

```
## [1] "From glmnet Lasso B_4 = -0.519431172781611"
```

```
print(paste0("From glmnet Lasso B_5 = ",glmnetfit$beta[5]))
```

```
## [1] "From glmnet Lasso B_5 = -0.171467908977004"
```

```
print(paste0("From glmnet Lasso B_6 = ",glmnetfit$beta[6]))
```

```
## [1] "From glmnet Lasso B_6 = -0.00665250716429136"
```

```
print(paste0("From glmnet Lasso B_7 = ",glmnetfit$beta[7]))
```

```
## [1] "From glmnet Lasso B_7 = 0"
```

```
print(paste0("From glmnet Lasso B_8 = ",glmnetfit$beta[8]))
```

```
## [1] "From glmnet Lasso B_8 = 0"
```

```
print(paste0("myLasso fucntion performed outer loop iterations = ", results$iter))
```

```
## [1] "myLasso fucntion performed outer loop iterations = 12"
```

```
print(paste0("From myLasso fucntion B_1 = ",results$beta[1]))
```

```
## [1] "From myLasso fucntion B_1 = 0"
```

```
print(paste0("From myLasso fucntion B_2 = ",results$beta[2]))
```

```
## [1] "From myLasso fucntion B_2 = 0.158234979128219"
```

```
print(paste0("From myLasso fucntion B_3 = ",results$beta[3]))
```

```
## [1] "From myLasso fucntion B_3 = 0.429647200748673"
```

```
print(paste0("From myLasso fucntion B_4 = ",results$beta[4]))
```

```
## [1] "From myLasso fucntion B_4 = -0.519445236442654"
```

```
print(paste0("From myLasso fucntion B_5 = ",results$beta[5]))
```

```
## [1] "From myLasso fucntion B_5 = -0.171476118889474"
```

```
print(paste0("From myLasso fucntion B_6 = ",results$beta[6]))
```

```
## [1] "From myLasso fucntion B_6 = -0.00664526303674387"
```

```
print(paste0("From myLasso fucntion B_7 = ",results$beta[7]))
```

```
## [1] "From myLasso fucntion B_7 = 0"
```

```
print(paste0("From myLasso fucntion B_8 = ",results$beta[8]))
```

```
## [1] "From myLasso fucntion B_8 = 0"
```

```
print(paste0("L1 norm of differences = ",L1_norm))
```

```
## [1] "L1 norm of differences = 3.22460339910458e-05"
```

Comment

First 8 parameters for lasso from glmnet are
$\beta^{glmnet} = [0.000000000, 0.158233309, 0.429648259, -0.519431173, -0.171467909, -0.006652507, 0.000000000, 0.000000000]$
First 8 parameters for my lasso are
$\beta^{my} = [0.000000000, 0.158234979, 0.42964720, -0.519445236, -0.171476119, -0.006645263, 0.000000000, 0.000000000]$
How we can see the β^{glmnet} and β^{my} have the same values, that's why L_1 norm of difference between them are really small, almost 0, i.e. $\| \hat{\boldsymbol{\beta}}^{\text{glment}}_{[1:8]} - \hat{\boldsymbol{\beta}}^{\text{yours}}_{[1:8]} \|_1 = 3.22460339910458 * 10^{-5}$. In total we need 12 outer loop iterations to estimate β_{lasso} using myLasso function.

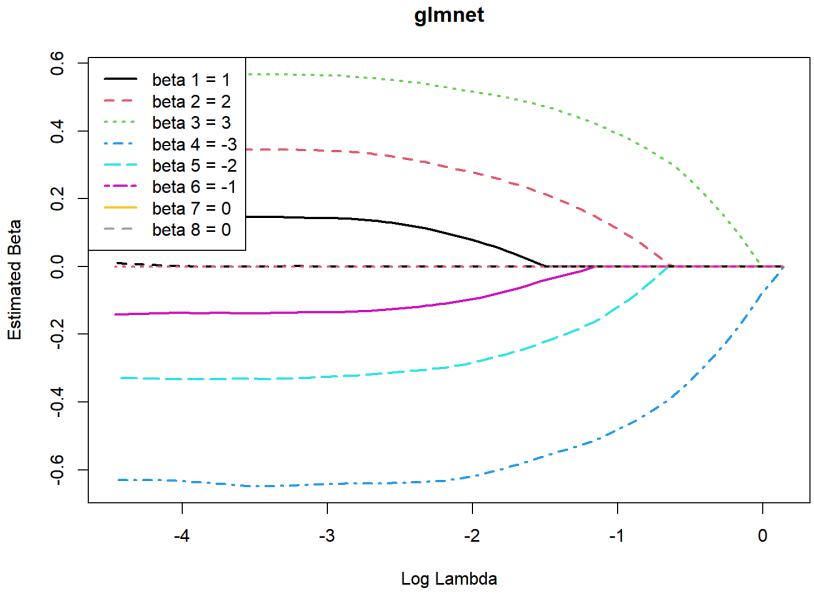
Question 2 [50 Points] Path-wise Coordinate Descent

Let's perform path-wise coordinate descent. The idea is simple: we will solve the solution on a sequence of λ values, starting from the largest one in the sequence. The first initial $\boldsymbol{\beta}$ are still all zero. After obtaining the optimal $\boldsymbol{\beta}$ for a given λ, we simply use this solution as the initial value for the next, smaller λ. This is referred to as a **warm start** in optimization problems. We will consider the following sequence of λ borrowed from `glmnet` . Note that this is a decreasing sequence from large to small values.

```
glmnetfit = glmnet(X, y, intercept = FALSE)

# Again, twice lambda is used for our function
lambda_all = glmnetfit$lambda * 2

# a matplot of the first 8 coefficients vs log scale of lambda
matplot(log(lambda_all), t(glmnetfit$beta[1:8, ]), type = "l", lwd = 2,
        xlab = "Log Lambda",ylab = "Estimated Beta", main = "glmnet")
legend("topleft", paste("beta", 1:8, "=", c(1:3, -3:-1, 0, 0)),
       col = 1:8, lty = 1:8, lwd = 2)
```



a. [20 pts] Write a function `myLasso_pw <- function(X, y, lambda_all, tol, maxitr)`, which output a $p \times N_\lambda$ matrix. N_λ is the number of unique λ values. Also follow the above instruction at the beginning of this question to include the **warm start** for path-wise solution. Your `myLasso_pw` should make use of your `myLasso` in Question 1.

```
#Lasso piece wise function
myLasso_pw <- function(X, y, lambda_all, tol, maxitr){
  beta_matrix=matrix(0, ncol(X), length(lambda_all))
  i=1
  beta_fit=rep(0, ncol(X))

  #warm start implementation
  while(i<=length(lambda_all)){
    res=myLasso(X, y, lambda_all[i], beta_fit, tol, maxitr)$beta
    beta_matrix[, i]=res
    beta_fit=res
    i=i+1

  }

  beta_matrix
}
```

b. [5 pts] Provide the same plot as the above `glmnet` solution plot of the first 8 parameter in your solution path. Make the two plots side-by-side (e.g. `par(mfrow = c(1, 2))` in `R`) with `glmnet` on the left and your solution path on the right.

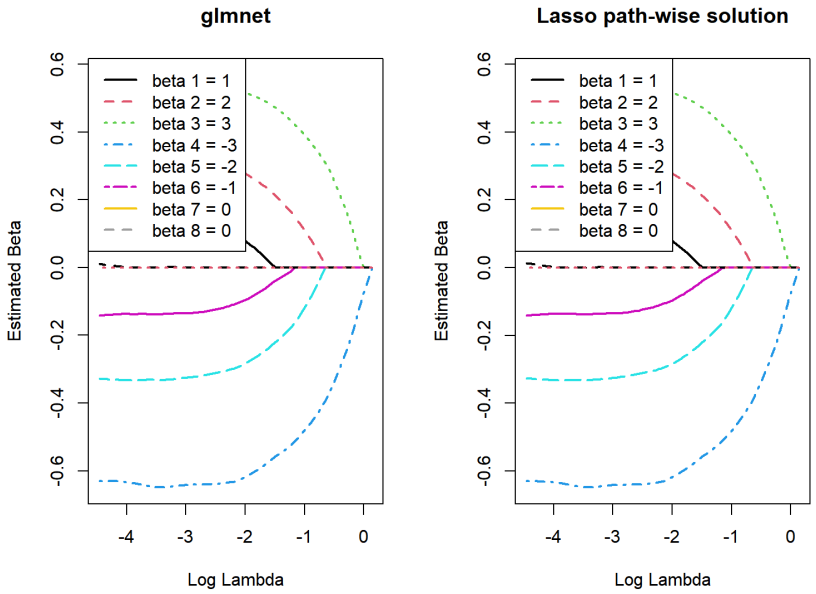
```
matrix_beta=myLasso_pw(X, y, lambda_all, 0.0000001, 100)


glmnetfit = glmnet(X, y, intercept = FALSE)

# a matplot of the first 8 coefficients vs log scale of lambda
par(mfrow = c(1, 2))

  matplot(log(lambda_all), t(glmnetfit$beta[1:8, ]), type = "l", lwd = 2,
          xlab = "Log Lambda",ylab = "Estimated Beta", main = "glmnet")
  legend("topleft", paste("beta", 1:8, "=", c(1:3, -3:-1, 0, 0)),
         col = 1:8, lty = 1:8, lwd = 2)

  matplot(log(lambda_all), t(matrix_beta[1:8, ]), type = "l", lwd = 2,
          xlab = "Log Lambda",ylab = "Estimated Beta", main = "Lasso path-wise solution")
  legend("topleft", paste("beta", 1:8, "=", c(1:3, -3:-1, 0, 0)),
         col = 1:8, lty = 1:8, lwd = 2)
```



c. [5 pts] Based on your plot, if we decrease λ from its maximum value, which two variables enter (start to have nonzero values) the model first? You may denote your covariates as $X_1, \ldots, X_8$.

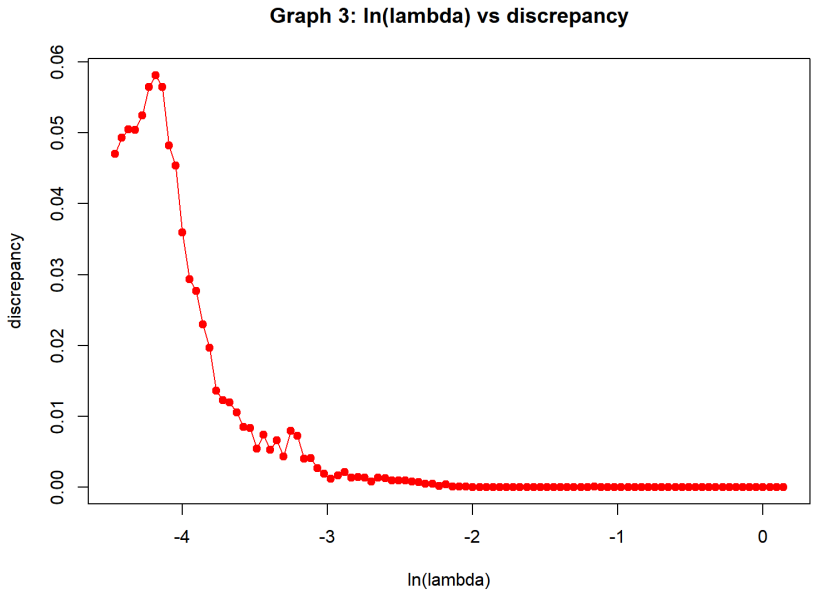
Comment

We know that the lambda_all[1]= 1.153 is maximum lambda, ln(1.153)= 0.1424. So as lambda decrease, decreases ln(lambda) decreases too. We need to look at graph from right side to left side. We can see that β_4 and β_3 intercept y=0 axis at ln(lambda) is approximately equal 0. So as lambda decreases the graphs of β_4 and β_3 begin first among all other betas graphs change. So based on my plot X_4 and X_3 enter the model first.

d. [5 pts] In Question 1, we calculated the L1 norm discrepancy between our solution and `glmnet` on the first 8 parameters. In this question, we will calculate the discrepancy on **all** coefficients, and over all λ parameters. After calculating the discrepancies, show a scatter plot of **log(λ) vs. discrepancy**. Comment on what you observe.

```
#calculate discrepancy
abs_difference=abs(glmnetfit$beta-matrix_beta)
i=1
discrepancy=rep(0,length(lambda_all))
while(i<=length(lambda_all)){
  discrepancy[i]=sum(abs_difference[,i])
  i=i+1
}

#draw ln(lambda) vs discrepancy graph
plot(log(lambda_all),discrepancy, type = "l", col="red", main="Graph 3: ln(lambda) vs discrepancy", xlab =
"ln(lambda)", ylab ="discrepancy")
points(log(lambda_all),discrepancy, pch=19, col="red")
```



Comment

So as $ln(\lambda)$ decreases, the discrepancy begins exponentially increase until it reaches $ln(\lambda) = -4$, the peak. Then after $ln(\lambda) = -4$ for other $ln(\lambda) < -4$ discrepancy start to decrease.

e. [15 pts] Based on the solution you obtained in the previous question, recover the unscaled coefficients using the formula in HW4. Then compare the first 9 coefficients (including the intercept term) with the following using a similar plot in b). Report the maximum value of discrepancy (see d) across all λ values.

```
  glmnetfit2 = glmnet(X_org, y_org, lambda = lambda_all/2*sd(y_org)*sqrt(n/(n-1)))
  lassobeta2 = coef(glmnetfit2)[1:9, ]

beta_matrix_unscaled=matrix(0, nrow(matrix_beta)+1, ncol(matrix_beta))

#calculate initial B0 using scaled Beta parameters
i=1
sigma_b0=0
while(i<=100){
  for (j in c(1:200)){
    sigma_b0= sigma_b0+(mean(X_org[ ,j])*sd(y_org)*matrix_beta[j,i])/sd(X_org[ ,j])
  }
  b0=mean(y_org)-sigma_b0
  beta_matrix_unscaled[1,i]=b0
  sigma_b0=0
  b0=0
  i=i+1
}

#calculate B1, B2, B3, ..., B199 using scaled Beta parameters
k=1
beta=0
while(k<=100){
  for (j in c(1:200)){
    beta= (sd(y_org)*matrix_beta[j,k])/sd(X_org[ ,j])
    beta_matrix_unscaled[j+1,k]=beta
  }
  k=k+1
}

#find discrepancy
abs_difference2=abs(coef(glmnetfit2)-beta_matrix_unscaled)
i=1
discrepancy2=rep(0,length(lambda_all))
while(i<=length(lambda_all)){
  discrepancy2[i]=sum(abs_difference2[,i])
  i=i+1
}

max_discrepancy=max(discrepancy2)
index=which(discrepancy2==max_discrepancy)

print(paste0("max discrepancy = ",max_discrepancy))
```

```
## [1] "max discrepancy = 0.312884013014213"
```

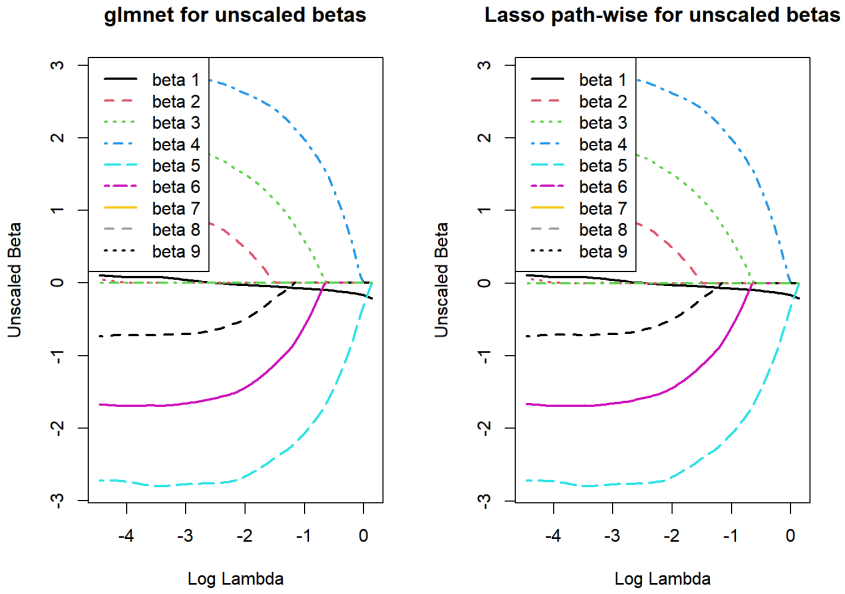
```
print(paste0("max discrepancy corresponds to lambda= ",lambda_all[index]))
```

```
## [1] "max discrepancy corresponds to lambda= 0.015241280598183"
```

```
# a matplot of the first 8 coefficients vs log scale of lambda
par(mfrow = c(1, 2))

  matplot(log(lambda_all), t(lassobeta2), type = "l", lwd = 2,
          xlab = "Log Lambda",ylab = "Unscaled Beta", main = "glmnet for unscaled betas")
  legend("topleft", paste("beta", 1:9), col = 1:9, lty = 1:9, lwd = 2)

  matplot(log(lambda_all), t(beta_matrix_unscaled[1:9, ]), type = "l", lwd = 2,
          xlab = "Log Lambda",ylab = "Unscaled Beta", main = "Lasso path-wise for unscaled betas")
  legend("topleft", paste("beta", 1:9), col = 1:9, lty = 1:9, lwd = 2)
```



Comment

Looking at $ln(\lambda)$ vs $\hat{\beta}_{unscaled}$ graphs for glmnet and Lasso path-wise, I can see that max discrepancy will be very small, as the graphs matches to each other. So calculated $discrepancy_{max} = 0.312884013014213$ and it corresponds to $\lambda_{94} = 0.015241280598183$.