Third assignment$_4$ Course

Mathematical and Computational Statistics

Achladianakis Minas

A report presented for the Course Mathematical and Computational Statistics

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Department of Applied Mathematics
University of Crete

Greece, November 24, 2023

Abstract

This assignment is part of my evaluation for the course: Mathematical and Computational Statistics.

List of Figures

Assignment 3

3.1 Reprocess

To use the M5 model from assignment 2 for our airquality dataset there are some steps we have to make before we proceed to the requested adaptive LAD LASSO implementations.

3.1.1 The airquality dataset

The original dataset had a fair amount of NA values, due to our small sample size I chose to fill these NA's with their column mean (even though there are better ways for filling null values)

```
#The data
data("airquality")
```

```
# Function to replace missing values with the mean of the column
replace_na_with_mean <- function(x) {
  if (is.numeric(x)) {
    x[is.na(x)] <- mean(x, na.rm = TRUE)
  }
  return(x)
}

# Function to clean each column of the airquality dataset
airquality_clean <- apply(airquality, 2, replace_na_with_mean)

# Convert the result back to a data frame
airquality_clean <- as.data.frame(airquality_clean)

# The form of the data and the initial columns
> head(airquality_clean)
     Ozone  Solar.R Wind Temp Month Day        Z1        Z2       Z3
1 41.00000 190.0000  7.4   67     5   1 122.42025  3.788493 42.16097
2 36.00000 118.0000  8.0   72     5   2  82.25212  6.147501 58.80387
3 12.00000 149.0000 12.6   74     5   3 201.46264  9.487300 66.22328
4 18.00000 313.0000 11.5   62     5   4 254.12116 10.390048 61.17171
5 42.12931 185.9315 14.3   56     5   5 155.56852 10.652739 52.18959
6 28.00000 185.9315 14.9   66     5   6 239.25975 13.985127 57.09785
```

To use Model M5 is to use its predictors which were 3 noise predictors on (M3), correlated with rho approximately equal to 0.8 with SolarR, Wind and Temp, the

quadratic and linear terms of Solar$\dot{R}$, Wind and Temp as well as their two-way interactions Solar.R Wind, Solar.R Temp and Temp Wind

3.1.2 Noise Predictors

```r
set.seed(123) # For reproducibility
rho <- 0.8 # Desired correlation coefficient

# Create Z1 correlated with Solar.R
Z1 <- rho * airquality_clean$Solar.R + sqrt(1 - rho^2) *
 ↪  rnorm(nrow(airquality_clean), sd = sd(airquality_clean$Solar.R))

# Create Z2 correlated with Wind
Z2 <- rho * airquality_clean$Wind + sqrt(1 - rho^2) *
 ↪  rnorm(nrow(airquality_clean), sd = sd(airquality_clean$Wind))

# Create Z3 correlated with Temp
Z3 <- rho * airquality_clean$Temp + sqrt(1 - rho^2) *
 ↪  rnorm(nrow(airquality_clean), sd = sd(airquality_clean$Temp))

# Add these to your dataset
airquality_clean$Z1 <- Z1
airquality_clean$Z2 <- Z2
airquality_clean$Z3 <- Z3
```

Before we move on to the M5 fit we should fit the M4 and use its predicted values to create the ln_Q target variable (as seen in Tofallis's paper[1])

3.1.3 The target variable

```r
#The M4 model
M4 <- lm(Ozone ~ poly(Solar.R, 2) + poly(Wind, 2) + poly(Temp, 2)
         + Solar.R:Wind + Solar.R:Temp + Wind:Temp + Z1 + Z2 + Z3,
         data = airquality_clean)

# Predicted values from M4
predicted_M4 <- predict(M4)

# Compute the quotient Q
Q <- predicted_M4 / log(airquality_clean$Ozone +1e-6)

# Apply the logarithm to Q (avoiding log(0))
airquality_clean$ln_Q <- log(Q + 1e-6)
```

[1]For more information on Tofalis paper and my previous work, see assignment 2.

3.1.4 M5 fit

```
#Our M5
M5 <- lm(ln_Q ~ poly(Solar.R, 2) + poly(Wind, 2) + poly(Temp, 2)
         + Solar.R:Wind + Solar.R:Temp + Wind:Temp + Z1 + Z2 + Z3,
         data = airquality_clean)
```

3.1.5 Some metrics

```
### Some metrics functions
#RMSE function
calculate_rmse <- function(predicted, actual) {
  if(length(predicted) != length(actual)) {
    stop("Lengths of predicted and actual values do not match.")
  }

  mse <- mean((predicted - actual) ^ 2)
  rmse <- sqrt(mse)
  return(rmse)
}
#MAE function
calculate_mae <- function(predicted, actual) {
  if(length(predicted) != length(actual)) {
    stop("Lengths of predicted and actual values do not match.")
  }

  mae <- mean(abs(predicted - actual))
  return(mae)
}
#MAPE function
calculate_mape <- function(predicted, actual) {
  if(length(predicted) != length(actual)) {
    stop("Lengths of predicted and actual values do not match.")
  }

  mape <- mean(abs(predicted - actual)/abs(actual))*100
  return(mape)
}
# sum log change function
calculate_lch <- function(predicted, actual) {
  if(length(predicted) != length(actual)) {
    stop("Lengths of predicted and actual values do not match.")
  }

  lch <- sum(log((predicted +1e-6)/(actual +1e-6)))
  return(lch)
}
```

where the last LCH is the one mentioned in Tofallis's paper as the log change metric.

3.1.6 Compairing Models

```
M4_rmse=calculate_rmse(predicted_M4,airquality_clean$Ozone)
M4_mae=calculate_mae(predicted_M4,airquality_clean$Ozone)
M4_mape=calculate_mape(predicted_M4,airquality_clean$Ozone)
M4_lch=calculate_lch(predicted_M4,airquality_clean$Ozone)

M5_rmse=calculate_rmse(predict(M5),airquality_clean$ln_Q)
M5_mae=calculate_mae(predict(M5),airquality_clean$ln_Q)
M5_mape=calculate_mape(predict(M5),airquality_clean$ln_Q)
M5_lch=calculate_lch(predict(M5),airquality_clean$ln_Q)
```

Metric Results				
model	RMSE	MAE	MAPE	LCH
M4	17.43341	13.05916	48.07146	13.54007
M5	0.4913037	0.3744274	166.0625	3.436372

This model was taught based on Tofallis instructions to provide better results regarding LCH though it seems it does better regarding other metrics as well, the only metric that did better under M4 was MAPE which could inflate due to singular extreme values.

3.2 Adaptive LAD LASSO

This assignment aims to compare our M5 estimations using 2 adaptive LAD LASSO implementations. One uses rqpen R package and the other uses the hqreg.

3.2.1 Data preparetion

```
# Prepare your data
x_matrix <- model.matrix(~ poly(Solar.R, 2) + poly(Wind, 2) + poly(Temp,
↪  2)
                         + Solar.R:Wind + Solar.R:Temp + Wind:Temp + Z1 +
                         ↪  Z2 + Z3,
                         data = airquality_clean)[, -1]  # Exclude
                         ↪  intercept
y_vector <- airquality_clean$ln_Q


coef_M5 <- abs(coef(M5)[-1])
adaptive_weights <- 1 / coef_M5  # Inverse of absolute coefficients
```

with the weights being:

```
> adaptive_weights
poly(Solar.R, 2)1 poly(Solar.R, 2)2    poly(Wind, 2)1    poly(Wind, 2)2
↪  poly(Temp, 2)1    poly(Temp, 2)2
     3.822621e-02      1.249847e+00      3.393103e-02      5.850143e-01
     ↪  6.854823e-02      3.539151e-01
              Z1                Z2                Z3      Solar.R:Wind
               ↪  Solar.R:Temp          Wind:Temp
     7.437291e+02      1.122742e+01      1.429780e+02      1.071852e+05
     ↪  2.994795e+03      1.493332e+02
```

3.2.2 The rqpen

```
library(rqPen)
```

CV for optimal lambda value

```
cv_model_rqPen <- rq.pen.cv(x = x_matrix, y = y_vector, tau = 0.5,
                            penalty = "aLASSO", nfolds = 10,
                            lambda = NULL, penalty.factor =
                            ↪  adaptive_weights, set.seed(123))

best_lambda_rqPen <- cv_model_rqPen$btr$lambda[1]
```

The best lambda procured has value **0.1496446** with estimated coefficients:

```
> coefficients_df
            Predictor Coefficient
1           intercept   2.3937688
2  poly(Solar.R, 2)1   0.9255582
3  poly(Solar.R, 2)2   0.0000000
4     poly(Wind, 2)1  -2.3860965
5     poly(Wind, 2)2   0.0000000
6     poly(Temp, 2)1   2.0173224
7     poly(Temp, 2)2   0.0000000
8                 Z1   0.0000000
9                 Z2   0.0000000
10                Z3   0.0000000
11  Solar.R_log:Wind   0.0000000
12  Solar.R_log:Temp   0.0000000
13     Wind_log:Temp   0.0000000
```

generated by running the R code:

```
# For the coefficients

# Extract the index of the best lambda
best_lambda_index <- cv_model_rqPen$btr$lambdaIndex

# Extract the coefficients corresponding to the best lambda
best_coefficients <- cv_model_rqPen$fit$models$tau0.5a1$coefficients[,
↪  best_lambda_index]

# View the best coefficients
best_coefficients

best_coefficients <- c(2.3937688 , 0.9255582, 0.000000, -2.3860965,
↪  0.000000,
                       2.0173224, 0.000000, 0.000000, 0.000000, 0.000000,
                       0.000000, 0.000000, 0.000000)

predictors <- c("intercept", "poly(Solar.R, 2)1", "poly(Solar.R, 2)2",
                "poly(Wind, 2)1", "poly(Wind, 2)2", "poly(Temp, 2)1",
                "poly(Temp, 2)2", "Z1", "Z2", "Z3",
```

```
                "Solar.R:Wind", "Solar.R:Temp", "Wind:Temp")

coefficients_df <- data.frame(Predictor = predictors, Coefficient =
↪  best_coefficients)
coefficients_df
```

rqpen Solution stability

Below I will also submit the coefficients for rqpen adaptive LAD-LASSO implementation near the best value provided to present the stability of these values through small changes in the parameter λ which aims to provide evidence of continuous dependence on initial conditions and prediction stability

```
> coef(model_M5_rqPen)
                    tau0.5a1 L1 tau0.5a1 L2 tau0.5a1 L3 tau0.5a1 L4 tau0.5a1
                    ↪  L5 tau0.5a1 L6 tau0.5a1 L7
intercept             2.3933144    2.393819   2.3937774   2.3938064
↪  2.3939467   2.3940874   2.3941379
poly(Solar.R, 2)1   0.9111265    0.921361   0.9226808   0.9222245
↪  0.9186264   0.9148948   0.9132678
poly(Solar.R, 2)2   0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
poly(Wind, 2)1     -2.3677614   -2.387893  -2.3875075  -2.3873036
↪  -2.3872875  -2.3874316  -2.3876774
poly(Wind, 2)2      0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
poly(Temp, 2)1      2.0499651    2.022131   2.0193506   2.0176824
↪  2.0178652   2.0181148   2.0169336
poly(Temp, 2)2      0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
Z1                  0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
Z2                  0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
Z3                  0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
Solar.R:Wind        0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
Solar.R:Temp        0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
Wind:Temp           0.0000000    0.000000   0.0000000   0.0000000
↪  0.0000000   0.0000000   0.0000000
```

generated using the code

```
# Generate a sequence of of best lambda
lambda_sequence <- seq(best_lambda_rqPen * 0.98, best_lambda_rqPen * 1.03,
↪  length.out = 7)

# Fit the adaptive LAD-LASSO model using rq.pen with a sequence of lambda
↪  values
model_M5_rqPen <- rq.pen(x = x_matrix, y = y_vector, tau = 0.5,
```

```
                                lambda = lambda_sequence, penalty = "aLASSO",
                                penalty.factor = adaptive_weights)

# View the coefficients of the fitted model
coef(model_M5_rqPen)
```

Coefficient Results Validity

As we can see it captured the useless noisy predictors Zi sending their coefficients to 0, though some could argue that it shrank to 0 useful predictors as well and they could also prove it through forward or backward selection schemes or VIF filtering etc. by keeping different predictors. I run such schemes in most cases you will see that the intercept and at least two of these predictors are kept, which means that those predictors provided via rqpen are in fact the most important ones according to rather variable elimination procedures as well.
A Result using backward elimination procedure The coefficients were calculated form the full model to the reduced one and in each step we through those we deemed as less important based on their p_values to reach the below "final" model.

```
Call:
lm(formula = ln_Q ~ poly(Solar.R, 1) + poly(Wind, 1) + poly(Temp,
    1) + Solar.R:Temp, data = airquality_clean)

Residuals:
    Min      1Q  Median      3Q     Max
-1.9975 -0.2114 -0.0268  0.1857 11.6259

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)      -3.827e+00  1.621e+00  -2.361 0.019551 *
poly(Solar.R, 1) -3.447e+01  8.906e+00  -3.870 0.000162 ***
poly(Wind, 1)    -3.307e+00  1.198e+00  -2.760 0.006518 **
poly(Temp, 1)    -9.252e+00  2.732e+00  -3.386 0.000908 ***
Solar.R:Temp      4.270e-04  1.102e-04   3.876 0.000159 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.059 on 148 degrees of freedom
Multiple R-squared:  0.1362,        Adjusted R-squared:  0.1129
F-statistic: 5.835 on 4 and 148 DF,  p-value: 0.000218
```

VIF

3.2.3 The hqreg

```
library(hqreg)
cv_model_hqreg <- cv.hqreg(x_matrix, y_vector, method = "quantile",tau =
↪  0.5,
                           penalty.factor = adaptive_weights,
                           ↪  type.measure='mae', set.seed(123))
```

```
# Extract the best lambda
best_lambda_hqreg <- cv_model_hqreg$lambda.min
```

which produced the result **0.4093123**, thought through the graphical illustration (A.3) we can observe that the curve provides insignificant fluctuations from 1 to 0. The coefficients of hqreg for the min λ are presented below L100 which was the last:

	L99	L100
(Intercept)	2.387676	2.386002
poly(Solar.R, 2)1	1.061181	1.085216
poly(Solar.R, 2)2	0.000000	0.000000
poly(Wind, 2)1	-2.433052	-2.437642
poly(Wind, 2)2	0.000000	0.000000
poly(Temp, 2)1	2.136130	2.146139
poly(Temp, 2)2	0.000000	0.000000
Z1	0.000000	0.000000
Z2	0.000000	0.000000
Z3	0.000000	0.000000

The insignificant fluctuations under smaller intervals

To further discuss and observe that phenomenon I will produce a revised algorithm with a custom λ interval from which values will be drawn, thought before that as we saw in the previous implementation a threshold for λ should be decided by us similar to the one presented in plot (A.1) for rqpen. We can do that more informatively by grating a focused graph in our area of question as:

```
cv_model_hqreg <- cv.hqreg(x_matrix, y_vector, method = "quantile",tau =
↪  0.5,
                           lambda = seq(0.05, 1.5, length.out = 100),
                           penalty.factor = adaptive_weights,
                           ↪  type.measure='mae', set.seed(123))
# Extract the best lambda
best_lambda_hqreg <- cv_model_hqreg$lambda.min
```

In this case, the min λ was **0.07929293** thought as we observe from the plot (A.4) the best lambda is greater than 0.07929293 swimmingly around 0.1, with minimal change in the coefficients.

	L98	L99	L100
(Intercept)	2.3986221	2.3987093	2.3987474
poly(Solar.R, 2)1	0.9268374	0.9242075	0.9220536
poly(Solar.R, 2)2	0.0000000	0.0000000	0.0000000
poly(Wind, 2)1	-2.4679167	-2.4680294	-2.4679692
poly(Wind, 2)2	0.0000000	0.0000000	0.0000000
poly(Temp, 2)1	1.8373741	1.8349395	1.8325477
poly(Temp, 2)2	0.0000000	0.0000000	0.0000000
Z1	0.0000000	0.0000000	0.0000000
Z2	0.0000000	0.0000000	0.0000000
Z3	0.0000000	0.0000000	0.0000000

rqpen vs hqreg

I felt that the rqpen had easier and more user-friendly implementation than hqreg and without further cv out of sample evidence and metrics, they provide the same important predictors with small fluctuation in their magnitude probably due to shareholding differently the λ variable.

3.2.4 The glmnet

Naive approach

```
library(glmnet)
cv_glmnet <- cv.glmnet(x_matrix, y_vector, alpha = 1, penalty.factor =
↪  adaptive_weights)
```

with the above implementation using our predetermined m5 coefficients the results are troubling:
The range of lambda does not include values less than 6.7 as presented below

```
[1] 67266.828336 61291.028701 55846.102637 50884.888797 46364.415521
↪  42245.528632 38492.552301 35072.980045
 [9] 31957.193163 29118.204200 26531.423192 24174.444679 22026.853642
 ↪  20070.048673 18287.080864 16662.507001
[17] 15182.255803 13833.506042 12604.575492 11484.819745 10464.540012
↪  9534.899119  8687.844960  7916.040759
[25]  7212.801515  6572.036108  5988.194533  5456.219835  4971.504303
↪  4529.849563  4127.430212  3760.760688
[33]  3426.665074  3122.249593  2844.877544  2592.146464  2361.867316
↪  2152.045534  1960.863740  1786.666009
[41]  1627.943525  1483.321509  1351.547315  1231.479576  1122.078325
↪  1022.395979   931.569138   848.811103
[49]   773.405064   704.697890   642.094472   585.052569   533.078111
↪  485.720921   442.570813   403.254042
[57]   367.430064   334.788590   305.046894   277.947369   253.255291
↪  230.756789   210.256991   191.578340
[65]   174.559048   159.051703   144.921988   132.047518   120.316781
↪  109.628170    99.889105    91.015232
[73]    82.929690    75.562445    68.849686    62.733269    57.160219
↪  52.082262    47.455418    43.239609
[81]    39.398321    35.898283    32.709179    29.803385    27.155735
↪  24.743294    22.545168    20.542317
[89]    18.717395    17.054593    15.539510    14.159023    12.901174
 ↪  11.755069    10.710781     9.759265
[97]     8.892279     8.102313     7.382526     6.726683
```

the best lambda comes as **38492.55** which is far to great for the penalty term. And the only predictors remaining are:

```
> coef(cv_glmnet,cv_glmnet$lambda.min)
13 x 1 sparse Matrix of class "dgCMatrix"
                           s1
(Intercept)         2.448233
poly(Solar.R, 2)1   .
```

```
poly(Solar.R, 2)2  .
poly(Wind, 2)1    -1.302890
poly(Wind, 2)2     .
poly(Temp, 2)1     .
poly(Temp, 2)2     .
Z1                 .
Z2                 .
Z3                 .
Solar.R:Wind       .
Solar.R:Temp       .
Wind:Temp          .
```

Lecture slides Part5d Implementation

```
>cvfit<-cv.glmnet(x_matrix,y_vector)
> best_lambda<-cvfit$lambda.min
> best_lambda
[1] 0.1720987
> best_coefs<-coef(cvfit, s=best_lambda)
> best_coefs
13 x 1 sparse Matrix of class "dgCMatrix"
                            s1
(Intercept)         2.4482332
poly(Solar.R, 2)1   .
poly(Solar.R, 2)2   .
poly(Wind, 2)1     -0.9031771
poly(Wind, 2)2      .
poly(Temp, 2)1      .
poly(Temp, 2)2      0.9464335
Z1                  .
Z2                  .
Z3                  .
Solar.R:Wind        .
Solar.R:Temp        .
Wind:Temp           .

> coef_4_weights<-best_coefs[-1]
> coef_4_weights
 [1]  0.0000000  0.0000000 -0.9031771  0.0000000  0.0000000  0.9464335
 ↪  0.0000000  0.0000000  0.0000000
[10]  0.0000000  0.0000000  0.0000000
> adaptive_weights <- 1 / (coef_4_weights +(1/length(coef_4_weights)))
> adaptive_weights
 [1] 12.0000000 12.0000000 -1.2197446 12.0000000 12.0000000  0.9710936
 ↪  12.0000000 12.0000000 12.0000000
[10] 12.0000000 12.0000000 12.0000000
```

To avoid errors as seen above we introduced a small constant in the denominator to ensure that we did not have problems with infinities and we excluded the intercept coefficient as we should.

```
cvfit2<-cv.glmnet(x_matrix,y_vector,alpha = 1, penalty.factor =
 ↪  adaptive_weights)
```

```
best_lambda<-cvfit2$lambda.min
best_coefs<-coef(cvfit2, s=best_lambda)
```

provides as with the result **0.1720201** for min lambda and the coefficients:

```
> best_coefs
13 x 1 sparse Matrix of class "dgCMatrix"
                            s1
(Intercept)          2.448233
poly(Solar.R, 2)1    .
poly(Solar.R, 2)2    .
poly(Wind, 2)1      -3.004119
poly(Wind, 2)2       .
poly(Temp, 2)1       .
poly(Temp, 2)2       2.839352
Z1                   .
Z2                   .
Z3                   .
Solar.R:Wind         .
Solar.R:Temp         .
Wind:Temp            .
```

As we expected the adaptive lasso has a greater shrinkage effect on the coefficient leaving only two predictors and the intercept term, based on which Solar$\dot{\text{R}}$ now is not an important predictor and in place of the Temp predictor our previous model had now is the quadratic term of Temp.

3.2.5 Bootstraping

Boot for glmnet

```
library(boot)


# Add predictors to my dataset
airquality_clean$Solar.R_2 <- airquality_clean$Solar.R^2
airquality_clean$Temp_2 <- airquality_clean$Temp^2
airquality_clean$Wind_2 <- airquality_clean$Wind^2
airquality_clean$Wind_Temp <- airquality_clean$Wind*airquality_clean$Temp
airquality_clean$Solar.R_Temp <-
↪  airquality_clean$Solar.R*airquality_clean$Temp
airquality_clean$Solar.R_Wind <-
↪  airquality_clean$Solar.R*airquality_clean$Wind

# Function to fit adaptive LASSO
fit_adaptive_lasso <- function(data, indices) {
  # Extract the training set
  training_set <- data[indices, ]

  x_matrix <- as.matrix(training_set[, !(names(training_set) %in%
  ↪  c("Ozone", "Day", "Month", "ln_Q"))])
  y_vector <- training_set$ln_Q
```

```
  # Fit the model
  adaptive_lasso_fit <- glmnet(x_matrix, y_vector, alpha = 1,
  ↪  penalty.factor = adaptive_weights, lambda = best_lambda)

  # Return coefficients
  return(coef(adaptive_lasso_fit, s = best_lambda)[-1])  # excluding
  ↪  intercept
}
```

```
# Bootstrap
set.seed(123)  # For reproducibility
glmnet_boot_results <- boot(data = airquality_clean, statistic =
↪  fit_adaptive_lasso, R = 100)

# Extracting coefficients
glmnet_boot_coefs <- boot_results$t

# Calculate confidence intervals (e.g., 95% CI)
glmnet_boot_ci <- apply(boot_coefs, 2, function(x) quantile(x, probs =
↪  c(0.025, 0.975)))
```

results:

```
> glmnet_boot_results

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = airquality_clean, statistic = fit_adaptive_lasso,
    R = 100)


Bootstrap Statistics :
          original        bias     std. error
t1*  -3.015691e-03 -3.135212e-03 1.297124e-02
t2*  -1.774499e-01 -2.278384e-02 1.088985e-01
t3*  -2.904314e-01  1.309911e-03 2.309164e-01
t4*  -1.916092e-03  1.014921e-05 1.547426e-03
t5*   8.274340e-02  3.806042e-03 9.331797e-02
t6*  -9.196533e-03 -9.696246e-04 8.595993e-03
t7*   1.266171e-05  6.001080e-06 2.228429e-05
t8*   1.976058e-03 -4.807233e-06 1.431229e-03
t9*   1.277927e-03  3.195788e-04 4.409535e-03
t10*  0.000000e+00  0.000000e+00 0.000000e+00
t11*  0.000000e+00  5.646613e-06 3.972604e-05
t12*  0.000000e+00  4.807929e-05 3.089280e-04
```

The original column presents estimates of the model parameters based on the full dataset, where the bias indicates the average bias (mean of bootstrap estimate - original estimate). A large bias might suggest that the model is sensitive to the

specific sample used and could be unstable. Finally, the standard error column provides an idea of the variability of the estimate. The zeros in t10*, t11*, and t12* could indicate that these predictors were not selected in the adaptive LASSO model (had all coefficients that were shrunk to zero).
For Confidence intervals:

```
> glmnet_boot_ci
           Solar.R       Wind       Temp         Z1          Z2       Z3
            ↪  Solar.R_2   Temp_2
2.5%  -0.034827119 -0.47993672 -0.74521022 -5.452405e-03 -0.02888882
 ↪  -0.030600523 -7.252551e-06 6.955257e-05
97.5%  0.007104201 -0.07189691  0.01626727  2.368815e-05  0.26027524
↪  0.002887144  6.804982e-05 4.821631e-03
       Wind_2   Wind_Temp  Solar.R_Temp   Solar.R_Wind
2.5%  -0.009661315     0     0 -0.0003273589
97.5%  0.007715593     0     0  0.0007548734
```

Boot for rqpen

```
rqpen_fit_adaptive_lad_lasso <- function(data, indices) {
  # Extract the training set
  training_set <- data[indices, ]

  x_matrix <- as.matrix(training_set[, !(names(training_set) %in%
  ↪  c("Ozone", "Day", "Month", "ln_Q"))])
  y_vector <- training_set$ln_Q

  # Fit the model
  model_M5_rqPen <- rq.pen(x_matrix, y_vector, tau = 0.5,
                           lambda = lambda_sequence, penalty = "aLASSO",
                           penalty.factor = adaptive_weights)
  #adaptive_lasso_fit <- glmnet(x_matrix, y_vector, alpha = 1,
  ↪  penalty.factor = adaptive_weights, lambda = best_lambda)

  # Return coefficients
  return(coef(model_M5_rqPen, s = best_lambda_rqPen)[-1])  # excluding
  ↪  intercept
}


set.seed(123)
rqpen_boot_results <-boot(data = airquality_clean, statistic =
↪  rqpen_fit_adaptive_lad_lasso, R = 100)


# Extracting coefficients
rqpen_boot_coefs <- rqpen_boot_results$t

# Calculate confidence intervals (e.g., 95% CI)
rqpen_boot_ci <- apply(boot_coefs, 2, function(x) quantile(x, probs =
↪  c(0.025, 0.975)))
```

with results :

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = airquality_clean, statistic = rqpen_fit_adaptive_lad_lasso,
    R = 100)


Bootstrap Statistics :
          original        bias     std. error
t1*   0.0009397290  1.944277e-05 0.0004519345
t2*   0.0000000000  0.000000e+00 0.0000000000
t3*   0.0252660925  6.910444e-04 0.0028237565
t4*   0.0000000000  0.000000e+00 0.0000000000
...          ....          ....          ....
t82*  0.0000000000  0.000000e+00 0.0000000000
t83* -0.0326898260  5.662952e-03 0.0144785518
t84*  0.0000000000  0.000000e+00 0.0000000000
```

The results are also plotted in (3.1) and for the coefficients see plot (3.2) as for the confidence intervals see the table below.

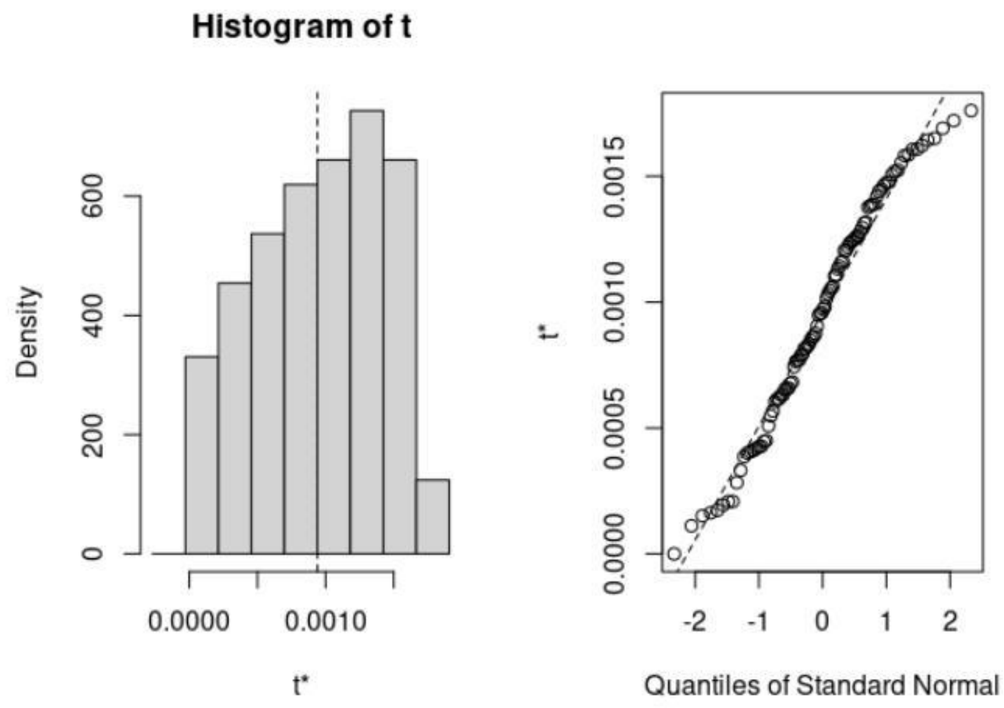

Figure 3.1: (II) Boot rqpen plot

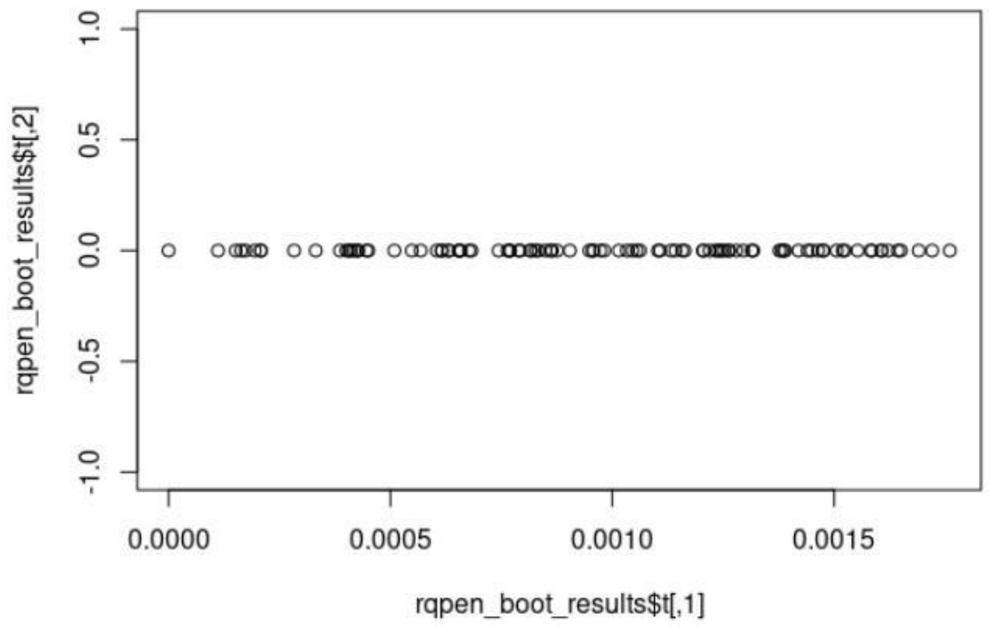

Figure 3.2: (II) Boot rqpen Coefficients plot

CI rqpen Results		
Subset	2.5%	97.5%
V1	2.347264	2.493112
V2	0	0
V3	0	0
V4	0	0
V5	0	0
V6	0	0
V7	0	0
V8	0	0
V9	0	0
V10	0	0
V11	0	0
V12	0	0
V13	2.269665	2.493112
V14	0	0
V15	-0.001640461	0.000000000
V16	0	0
V17	0	0
V18	0	0
V19	0	0
V20	0.000000e+00	1.822415e-05
V21	0	0
V22	0	0
V23	0	0
V24	2.112874	2.585276
V25	0	0
V26	-0.01610739	0.00000000
V27	0	0
V28	0	0
V29	-0.003155324	0.000000000
V30	0	0
V31	0.000000e+00	5.308067e-05
V32	0	0
V33	0	0
V34	1.967093	2.601963
V35	0	0
V36	-0.02766008	0.00000000

Boot for hqreg

```
hqreg_fit_adaptive_lad_lasso <- function(data, indices) {
  # Extract the training set
  training_set <- data[indices, ]

  x_matrix <- as.matrix(training_set[, !(names(training_set) %in%
  ↪  c("Ozone", "Day", "Month", "ln_Q"))])
  y_vector <- training_set$ln_Q

  # Fit the model
  model_M5_hqreg<- hqreg(x_matrix, y_vector, method = "quantile",tau =
  ↪  0.5,
                         lambda = seq(0.05, 1.5, length.out = 100),
                         penalty.factor = adaptive_weights)
  #adaptive_lasso_fit <- glmnet(x_matrix, y_vector, alpha = 1,
  ↪  penalty.factor = adaptive_weights, lambda = best_lambda)

  # Return coefficients
  return(coef(model_M5_hqreg, s = best_lambda_hqreg)[-1])  # excluding
  ↪  intercept
}

set.seed(123)
hqreg_boot_results <-boot(data = airquality_clean, statistic =
↪  hqreg_fit_adaptive_lad_lasso, R = 100)


# Extracting coefficients
hqreg_boot_coefs <- hqreg_boot_results$t

# Calculate confidence intervals (e.g., 95% CI)
hqreg_boot_ci <- apply(boot_coefs, 2, function(x) quantile(x, probs =
↪  c(0.025, 0.975)))
```

with results:

```
> hqreg_boot_results
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = airquality_clean, statistic = hqreg_fit_adaptive_lad_lasso,
    R = 100)

Bootstrap Statistics :
            original        bias     std. error
t1*     0.0013443180 -5.695227e-05 3.376738e-04
t2*    -0.0041192696 -4.246466e-03 1.084838e-02
t3*     0.0251756876  3.675446e-04 2.936446e-03
t4*     0.0000000000 -6.159969e-06 1.423829e-04
t5*    -0.0422832708  4.065240e-03 9.398864e-03
t6*    -0.0028099255 -5.140349e-04 2.309495e-03
```

```
t7*      0.0000000000  0.000000e+00 0.000000e+00
...      ...           ....           ....
t327*    0.0000000000  0.000000e+00 0.000000e+00
t328*    0.0237211583  5.467059e-05 2.332281e-03
t329*    0.0000000000  0.000000e+00 0.000000e+00
t330*   -0.0417343726  1.337498e-03 5.799320e-03
t331*    0.0000000000  0.000000e+00 0.000000e+00
t332*    0.0000000000  0.000000e+00 0.000000e+00
t333*    0.0000000000  0.000000e+00 0.000000e+00
```

The results are plotted in (3.3) and for the coefficients see plot (3.4) as for the confidence intervals see the table below

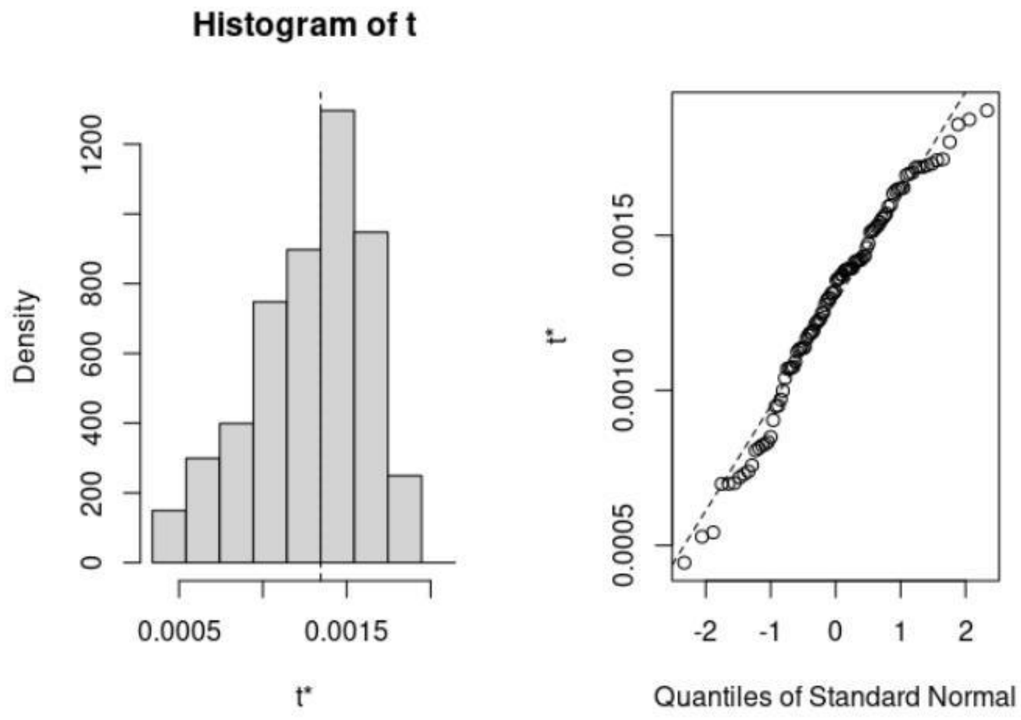

Figure 3.3: (II) Boot hqreg plot

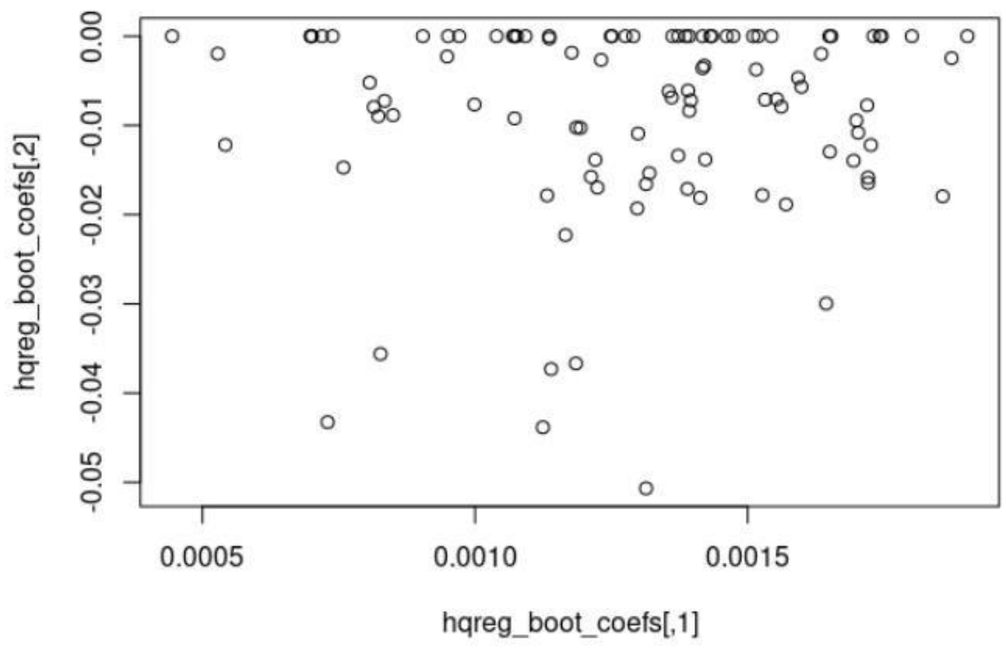

Figure 3.4: (II) Boot hqreg Coefficients plot

CI hqreg Results		
Subset	2.5%	97.5%
V1	0.0006163839	0.0018307814
V2	-0.0404335	0
V3	0.01960638	0.03113227
V4	-0.0003017126	0.0001584369
V5	-0.05345528	-0.01292548
V6	-0.007590109	0
V7	0	0
V8	0	0
V9	0	0
V10	0	0
V11	0	0
V12	0	0
V13	0.2507286	1.2059996
V14	0.0006038883	0.0019047229
V15	-0.03171616	0
V16	0.01972984	0.03145042
V17	-1.312530e-04	9.665806e-05
V18	-0.05484609	-0.01879310
V19	-0.007706812	0
V20	0	0
V21	0	0
V22	0	0
V23	0	0
V24	0	0
V25	0	0
V26	0.2396964	1.1308598
V27	0.0005885266	0.0018489806
V28	-0.0169221	0
V29	0.01981515	0.03140536
V30	-2.456186e-05	0.000000e+00
V31	-0.05560195	-0.02766871
V32	-0.007368275	0.000000000
V33	0	0
V34	0	0
V35	0	0
V36	0	0
V37	0	0
V38	0.2391763	1.1066720
V39	0.0005796293	0.0018500692
V40	-0.005021693	0.000000000
V41	0.01958890	0.03035412
V42	0	-0.03281203
V43	-0.05537145	0.000000000
V44	-0.006953005	0.000000000
V45	0...	0...

A visual aid confidence Interval Boxplot by the method (and by subset) can

provide more insight into our predetermined notion about these subsets as it solidifies our opinion about the fact that most of the resulted in 2.5% and 97.5% are mostly zero or near zero see (A.17) (A.18)

3.2.6 10-fold Cross Validation

I will present the code for estimating the MAE utilizing the capabilities of the caret R package which allows us to create folds based on indices and split our data accordingly inside the Cross Validation loop in which all our models are fitted and then used for predictions from which we calculate the MAE and then afterwards take the average:

```
library(caret)

set.seed(123)
folds <- createFolds(y_vector, k = 10, list = TRUE)

# To store the MAE for each fold for each model
mae_values_glmnet <- numeric(length(folds))
mae_values_rqpen <- numeric(length(folds))
mae_values_hqreg <- numeric(length(folds))

# 10-fold cross-validation loop
for(i in seq_along(folds)) {
# Splitting data into training and testing sets
train_indices <- unlist(folds[-i])
test_indices <- unlist(folds[i])

train_predictors <- x_matrix[train_indices, , drop = FALSE]
train_response <- y_vector[train_indices]

test_predictors <- x_matrix[test_indices, , drop = FALSE]
test_response <-y_vector[test_indices]

# Fit the glmnet model on the training set
glmnet_fit <- glmnet(train_predictors, train_response, alpha = 1,
                     penalty.factor = adaptive_weights, lambda =
                      ↪  best_lambda ) # adaptive LASSO
predictions_glmnet <- predict(glmnet_fit, newx = test_predictors, s =
↪  "lambda.min")

# Fit the rqpen model on the training set
rqpen_fit <- rq.pen(train_predictors, train_response, tau = 0.5,
                    lambda = lambda_sequence, penalty = "aLASSO",
                    nlambda = 100, penalty.factor = adaptive_weights)
                     ↪  #adaptive LAD-LASSO
predictions_rqpen <- predict(rqpen_fit, newx = test_predictors)

# Fit the hqreg model on the training set
hqreg_fit <- hqreg(train_predictors, train_response, method =
↪  "quantile",tau = 0.5,
```

```
                    lambda = seq(0.05, 1.5, length.out = 100),
                    penalty.factor = adaptive_weights)
  predictions_hqreg <- predict(hqreg_fit, X = test_predictors, s =
  ↪  hqreg_fit$lambda.min)

  # Calculate MAE for each model
  mae_values_glmnet[i] <- mean(abs(test_response - predictions_glmnet))
  mae_values_rqpen[i] <- mean(abs(test_response - predictions_rqpen))
  mae_values_hqreg[i] <- mean(abs(test_response - predictions_hqreg))
}

# Calculate the average MAE across all folds for each model
average_mae_glmnet <- mean(mae_values_glmnet)
average_mae_rqpen <- mean(mae_values_rqpen)
average_mae_hqreg <- mean(mae_values_hqreg)
```

the results are:

```
> print(average_mae_glmnet)
[1] 0.475286
> print(average_mae_rqpen)
[1] 0.2644732
> print(average_mae_hqreg)
[1] 0.2618617
```

As you can see there is a significant difference between glmnet and the other two which resulted in similar values and that is to be expected as the hqreg and rqpen used LAD Regression instead of ls.

3.3 Backward Step-wise Median Regression based on AIC

To create the backward step-wise median regression model we will use the function rq from the package quantreg and create a loop in which the model with the best AIC survives by fitting the model without one of the predictors for all the predictors one by one in each step.

```
library(quantreg)

# Define the full model formula excluding "Day", "Month", and "ln_Q"
full_formula <- reformulate(termlabels =
↪  names(airquality_clean)[!(names(airquality_clean) %in% c("Ozone",
↪  "Day", "Month", "ln_Q"))], response = "ln_Q")

# Fit a quantile regression model for the median (tau=0.5)
full_qr_model <- rq(full_formula, data = airquality_clean, tau = 0.5)

# Initialize the current best model as the full model
current_best_model <- full_qr_model
current_best_AIC <- AIC(full_qr_model)
```

```
# Get the names of the predictors from the full model
predictors <- all.vars(full_formula)[-1]  # Exclude the response variable

# Function to remove one predictor and refit the model
remove_predictor_and_refit <- function(predictor, current_predictors) {
  formula <- reformulate(termlabels = setdiff(current_predictors,
  ↪  predictor), response = "ln_Q")
  model <- rq(formula, data = airquality_clean, tau = 0.5)
  return(model)
}
```

```
# Perform backward selection
repeat {
  improvement <- FALSE
  # Copy the current predictors to a new object to avoid scoping issues
  current_predictors <- predictors

  for (predictor in current_predictors) {
    # Remove predictor and refit model
    temp_model <- remove_predictor_and_refit(predictor,
    ↪  current_predictors)
    temp_AIC <- AIC(temp_model)

    # Update the best model if improvement is found
    if (temp_AIC < current_best_AIC) {
      cat("Improved model found by removing", predictor, "\n")
      current_best_AIC <- temp_AIC
      current_best_model <- temp_model
      improvement <- TRUE
      # Update the predictors list with the one from the best model
      predictors <- all.vars(formula(current_best_model))[-1]
    }
  }

  # If no improvement in this iteration, stop the loop
  if (!improvement) {
    cat("No further improvement; stopping\n")
    break
  }
}
```

the result:

```
>current_best_model
Call:
rq(formula = formula, tau = 0.5, data = airquality_clean)

Coefficients:
  (Intercept)       Solar.R          Wind            Z2            Z3
  ↪  Solar.R_2        Temp_2
 1.664332e+00  7.391207e-03 -7.372591e-02 -2.158277e-02 -5.930823e-03
 ↪  -1.093846e-05  1.576454e-04
```

```
      Wind_2  Solar.R_Wind
 4.416858e-03 -2.159717e-04

Degrees of freedom: 153 total; 144 residual
> AIC(current_best_model)
[1] 73.87049
attr(,"edf")
[1] 9
```

As we can see this algorithmic approach seems to get stuck in that model as removing one of its predictors with not have significant improvement in AIC. I believe this is due to the ad-hoc way of removing the first significant difference first one out, instead a better algorithm could hold all of the AICs for any removal of a predictor and thoughts the predictor who in his absence the model had smaller AIC.

3.4 Monte Carlo experiment

As in the 5d slides Scenario 3 we will implement the Monte Carlo simulation by simulation results through the function gedData2, we will create our functions in which we will fit the model and return the coefficients, store the results, and then use them to create mae_results. The code should be like:

```
library(mvtnorm)

# Function to generate data
genData2 <- function(n, p, beta, rho, df) {
  Sigma <- toeplitz(rho ^ seq(0, p - 1))
  X <- rmvnorm(n, sigma = Sigma)
  e <- rt(n, df)  # t-distributed error terms
  y <- X %*% beta + e
  return(list(X = X, y = y))
}

# Backward selection function using AIC for quantile regression
backward_selection <- function(Data) {
  # Convert Data£X to a data frame and add the response variable
  predictors_df <- as.data.frame(Data$X)
  predictors_df$y <- Data$y

  # Define the full model formula excluding the response
  full_formula <- as.formula(paste("y ~",
  ↪  paste(names(predictors_df)[names(predictors_df) != "y"], collapse =
  ↪  " + ")))

  # Fit a quantile regression model for the median (tau=0.5)
  full_qr_model <- rq(full_formula, data = predictors_df, tau = 0.5)

  # Initialize the current best model as the full model
  current_best_model <- full_qr_model
  current_best_AIC <- AIC(full_qr_model)
```

```r
# Get the names of the predictors from the full model
predictors <- names(predictors_df)[names(predictors_df) != "y"]

# Function to remove one predictor and refit the model
remove_predictor_and_refit <- function(predictors_df,
 ↪  predictor_to_remove) {
  modified_df <- predictors_df[, !(names(predictors_df) %in%
   ↪  predictor_to_remove)]
  formula <- as.formula(paste("y ~",
   ↪  paste(names(modified_df)[names(modified_df) != "y"], collapse = "
   ↪  + ")))
  model <- rq(formula, data = modified_df, tau = 0.5)
  return(model)
}

# Perform backward selection
repeat {
  improvement <- FALSE

  for (predictor in predictors) {
    # Remove predictor and refit model
    temp_model <- remove_predictor_and_refit(predictors_df, predictor)
    temp_AIC <- AIC(temp_model)

    # Update the best model if improvement is found
    if (temp_AIC < current_best_AIC) {
      current_best_AIC <- temp_AIC
      current_best_model <- temp_model
      improvement <- TRUE
      predictors <- setdiff(predictors, predictor)  # Update the list of
       ↪  predictors
    }
  }

  # If no improvement in this iteration, stop the loop
  if (!improvement) {
    break
  }
}

return(current_best_model)
}


adaptive_lad_lasso_rqpen<-function(Data)
{
  x_matrix <- Data$X
  y_vector <- Data$y
# Perform cross-validation using rq.pen.cv with adaptive LASSO
cv_model_rqPen <- rq.pen.cv(x = x_matrix, y = y_vector, tau = 0.5,
                            penalty = "aLASSO", nfolds = 10,
```

```
                                lambda = NULL, penalty.factor =
                                ↪  adaptive_weights, set.seed(123))

best_lambda_rqPen <- cv_model_rqPen$btr$lambda[1]


# More For the coefficients

# Extract the index of the best lambda
best_lambda_index <- cv_model_rqPen$btr$lambdaIndex

# Extract the coefficients corresponding to the best lambda
best_coefficients <- cv_model_rqPen$fit$models$tau0.5a1$coefficients[,
↪  best_lambda_index]


# Generate a sequence of of best lambda
lambda_sequence <- seq(best_lambda_rqPen * 0.99, best_lambda_rqPen * 1.01,
↪  length.out = 2)

# Fit the adaptive LAD-LASSO model using rq.pen with a sequence of lambda
↪  values
model_M5_rqPen <- rq.pen(x = x_matrix, y = y_vector, tau = 0.5,
                         lambda = lambda_sequence, penalty = "aLASSO",
                         penalty.factor = adaptive_weights)

# View the coefficients of the fitted model
return(coef(model_M5_rqPen)[,1])
}

fill_coefficients <- function(model, predictors) {
  # Initialize a vector with zeros
  coefs <- setNames(rep(0, length(predictors)), predictors)

  # Replace with actual coefficients where available
  model_coefs <- coef(model)
  coefs[names(model_coefs)] <- model_coefs

  return(coefs)
}


N <- 100  # Number of replications
n <- 100  # Sample size
beta <- c(2, -2, 1, -1, 0.5, 0.2, -0.3, -0.15, rep(0, 12))  #
↪  Coefficients
rho <- 0.9
df <- 7  # Degrees of freedom for t-distribution

results <- array(NA, dim = c(N, length(beta), 2))  # Store results
```

```r
for (i in 1:N) {
  Data <- genData2(n, p, beta, rho)

  # Get coefficients from backward selection
  backward_model <- backward_selection(Data)
  backward_coefs <- fill_coefficients(backward_model, names(Data$X))

  # Get coefficients from adaptive LAD LASSO using hqreg
  adaptive_coefs <- adaptive_lad_lasso_rqpen(Data)

  # Store results
  results[i, , 1] <- backward_coefs
  results[i, , 2] <- adaptive_coefs
}


# Calculate MAE
mae_results <- apply(results, c(2, 3), function(coefs) mean(abs(coefs -
↪  beta)))

# Summarize the results
summary_results <- apply(mae_results, 2, function(x) c(MAE = mean(x), SD =
↪  sd(x)))
print(summary_results)
```

Alas, I failed to debag even after this all-night-er debating session...
or with something like:

```r
# Adaptive LAD LASSO function using rq.pen
adaptive_lad_lasso_rqpen <- function(Data) {
  x_matrix <- as.matrix(Data$X)
  y_vector <- Data$y
  adaptive_weights <- rep(1, ncol(x_matrix))

  cv_model_rqPen <- rq.pen.cv(x = x_matrix, y = y_vector, tau = 0.5,
  ↪  penalty = "aLASSO", nfolds = 10, lambda = NULL, penalty.factor =
  ↪  adaptive_weights)
  best_lambda_rqPen <- cv_model_rqPen$btr$lambda[1]

  # Generate a sequence of of best lambda
  lambda_sequence <- seq(best_lambda_rqPen * 0.99, best_lambda_rqPen *
  ↪  1.01, length.out = 2)

  model_M5_rqPen <- rq.pen(x = x_matrix, y = y_vector, tau = 0.5,
  ↪  lambda_sequence, penalty = "aLASSO", penalty.factor =
  ↪  adaptive_weights)
  return(coef(model_M5_rqPen)[,1])
}
```

```r
# Initialize the results array
results <- array(NA, dim = c(N, length(beta), 2))

# Monte Carlo simulation
set.seed(123)  # For reproducibility
for (i in 1:N) {
  Data <- genData2(n, p, beta, rho, df)

  # Backward selection model
  backward_model <- backward_selection(Data)
  backward_coefs <- coef(backward_model)

  # Adaptive LAD LASSO model
  adaptive_coefs <- adaptive_lad_lasso_rqpen(Data)

  # Store results
  results[i, , 1] <- backward_coefs
  results[i, , 2] <- adaptive_coefs
}
```

The problem is in the relationship of our imported values in results and the results from the Adaptive LAD LASSO function using rq.pen as the algorithms run correctly for as:

```r
set.seed(123)  # For reproducibility
for (i in 1:N) {
  Data <- genData2(n, p, beta, rho, df)

  # Backward selection model
  backward_model <- backward_selection(Data)
  backward_coefs <- fill_coefficients(coef(backward_model),
  ↪  all_predictors)

  # Adaptive LAD LASSO model using rqpen
  adaptive_coefs <- fill_coefficients(adaptive_lad_lasso_rqpen(Data),
  ↪  all_predictors)

  print(backward_coefs)
  print(adaptive_coefs)
  # Store results
  #results[i, , 1] <- backward_coefs
  #results[i, , 2] <- adaptive_coefs
}
```

Thus we just need to change our data import procedure and the fill_coefficients function as:

```r
fill_coefficients <- function(model_coefs, all_predictors) {
  # Initialize a vector with zeros for all predictors
```

```
filled_coefs <- setNames(rep(0, length(all_predictors)), all_predictors)

# Correct names if needed (V1, V2, etc. to x1, x2, etc.)
corrected_names <- names(model_coefs)
corrected_names <- sub("^V", "x", corrected_names)

# Match and fill the coefficients
for (idx in seq_along(corrected_names)) {
  coef_name <- corrected_names[idx]
  if (coef_name %in% all_predictors) {
    filled_coefs[coef_name] <- model_coefs[idx]
  }
}

  return(filled_coefs)
}



# Adjust the dimensions of the results array if necessary
results <- array(NA, dim = c(N, length(all_predictors), 2))  # N
↪  simulations, all_predictors, 2 methods

set.seed(123)  # For reproducibility
for (i in 1:N) {
Data <- genData2(n, p, beta, rho, df)

# Backward selection model
backward_model <- backward_selection(Data)
backward_coefs <- fill_coefficients(coef(backward_model),
↪  all_predictors)

# Adaptive LAD LASSO model using rqpen
adaptive_coefs <- fill_coefficients(adaptive_lad_lasso_rqpen(Data),
↪  all_predictors)

#print(backward_coefs)

#print(adaptive_coefs)
# Store results
results[i, , 1] <- backward_coefs
  results[i, , 2] <- adaptive_coefs
}
```

with results:

```
> mae_results <- apply(results, c(2, 3), function(coefs) mean(abs(coefs -
↪  beta)))
> mae_results
           [,1]      [,2]
 [1,] 0.3575000 0.4168276
 [2,] 1.9506448 1.7672786
```

```
 [3,] 2.0157843 1.6303776
 [4,] 1.2020714 0.8137942
 [5,] 1.2080205 0.8060557
 [6,] 0.7748221 0.5310210
 [7,] 0.6149202 0.4634584
 [8,] 0.6575658 0.4967174
 [9,] 0.6326604 0.4676251
[10,] 0.6550086 0.4314207
[11,] 0.6342049 0.4229826
[12,] 0.5674263 0.4339052
[13,] 0.5845101 0.4110731
[14,] 0.5625327 0.4005091
[15,] 0.6178478 0.4177252
[16,] 0.6036712 0.4025949
[17,] 0.5717213 0.4121746
[18,] 0.5915331 0.4174976
[19,] 0.6041187 0.3978151
[20,] 0.5680758 0.4254617
[21,] 0.5442009 0.4069155
```

and summary

```
> summary_results <- apply(mae_results, 2, function(x) c(MAE = mean(x), SD
↪  = sd(x)))
> summary_results
         [,1]      [,2]
MAE 0.7866115 0.5892015
SD  0.4436097 0.3877701
```

Appendix A

Appendix Title

Some useful Plots

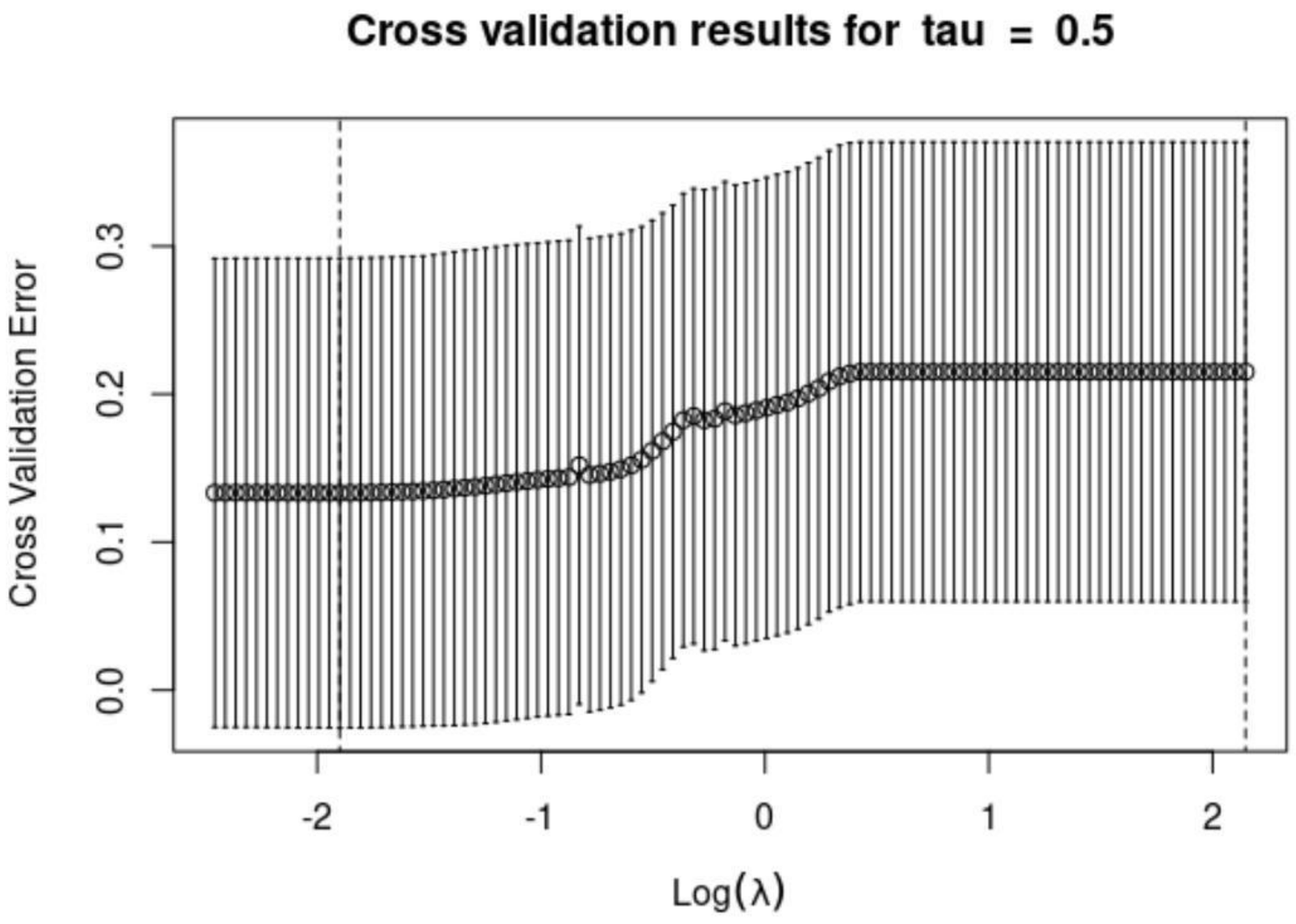

Figure A.1: rqPen Cross Validation Error

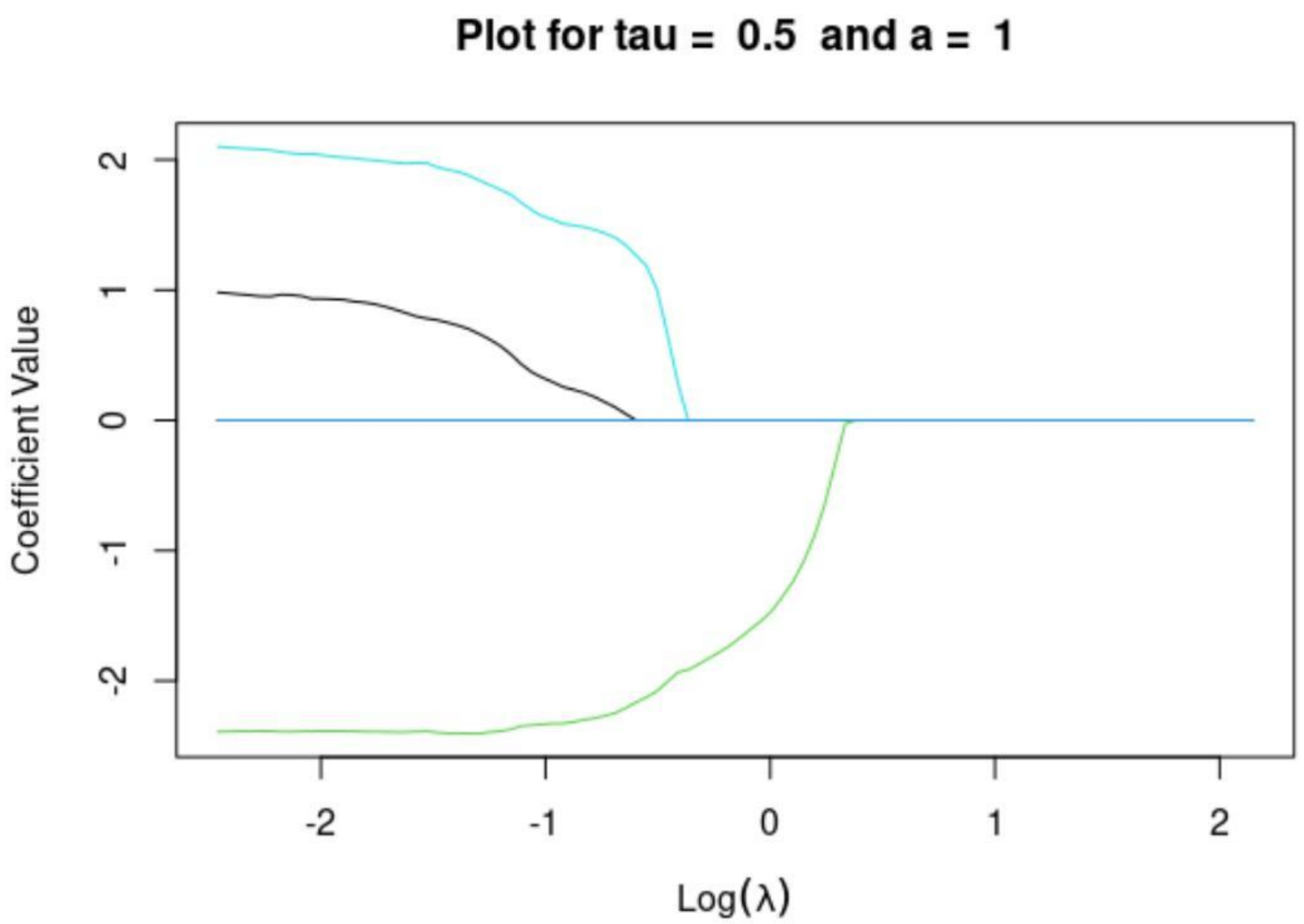

Figure A.2: rqPen Model fit

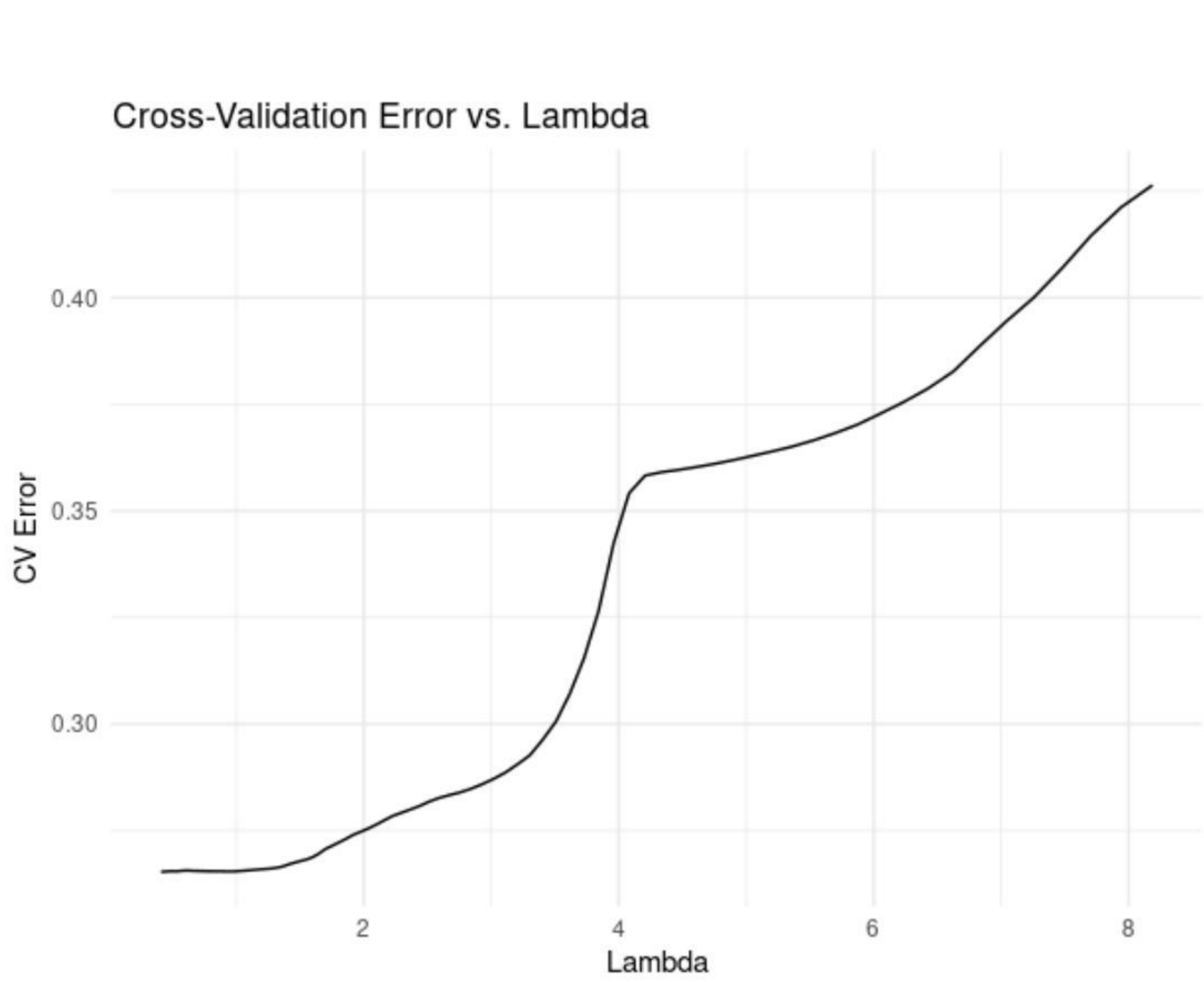

Figure A.3: hqreg Cross Validation Error

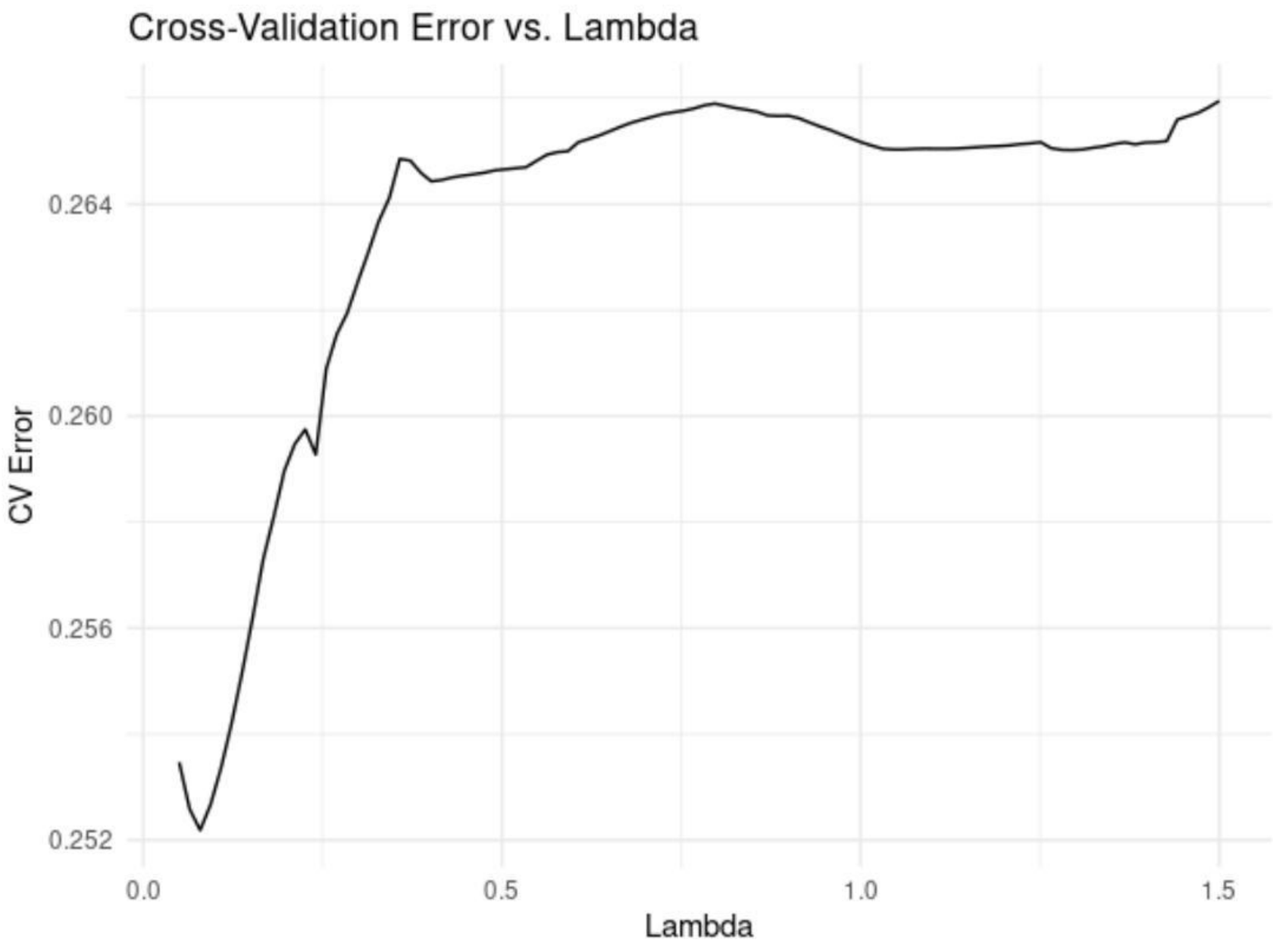

Figure A.4: hqreg Focused Cross Validation Error

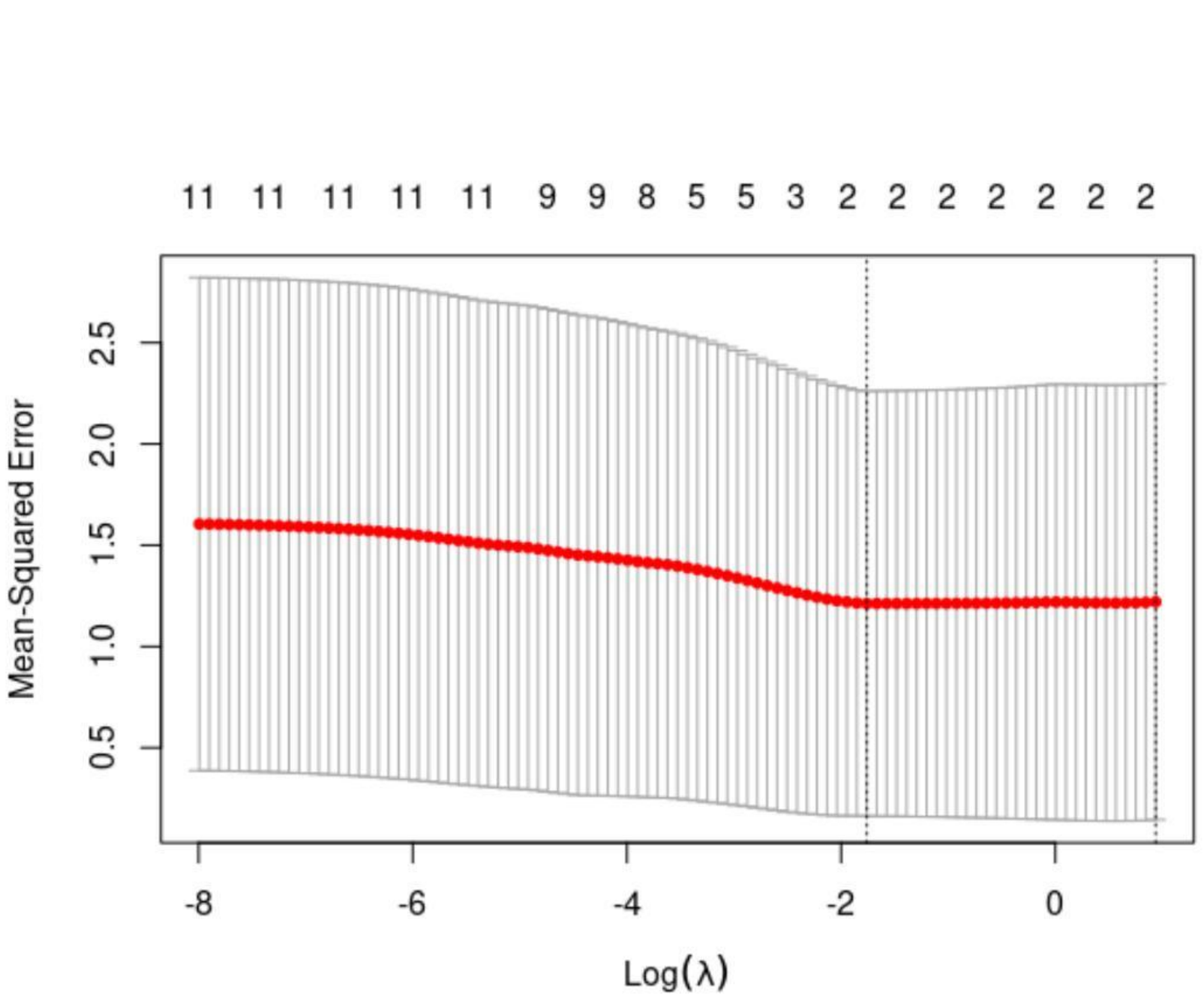

Figure A.5: glmnet Informative Implementation (II)

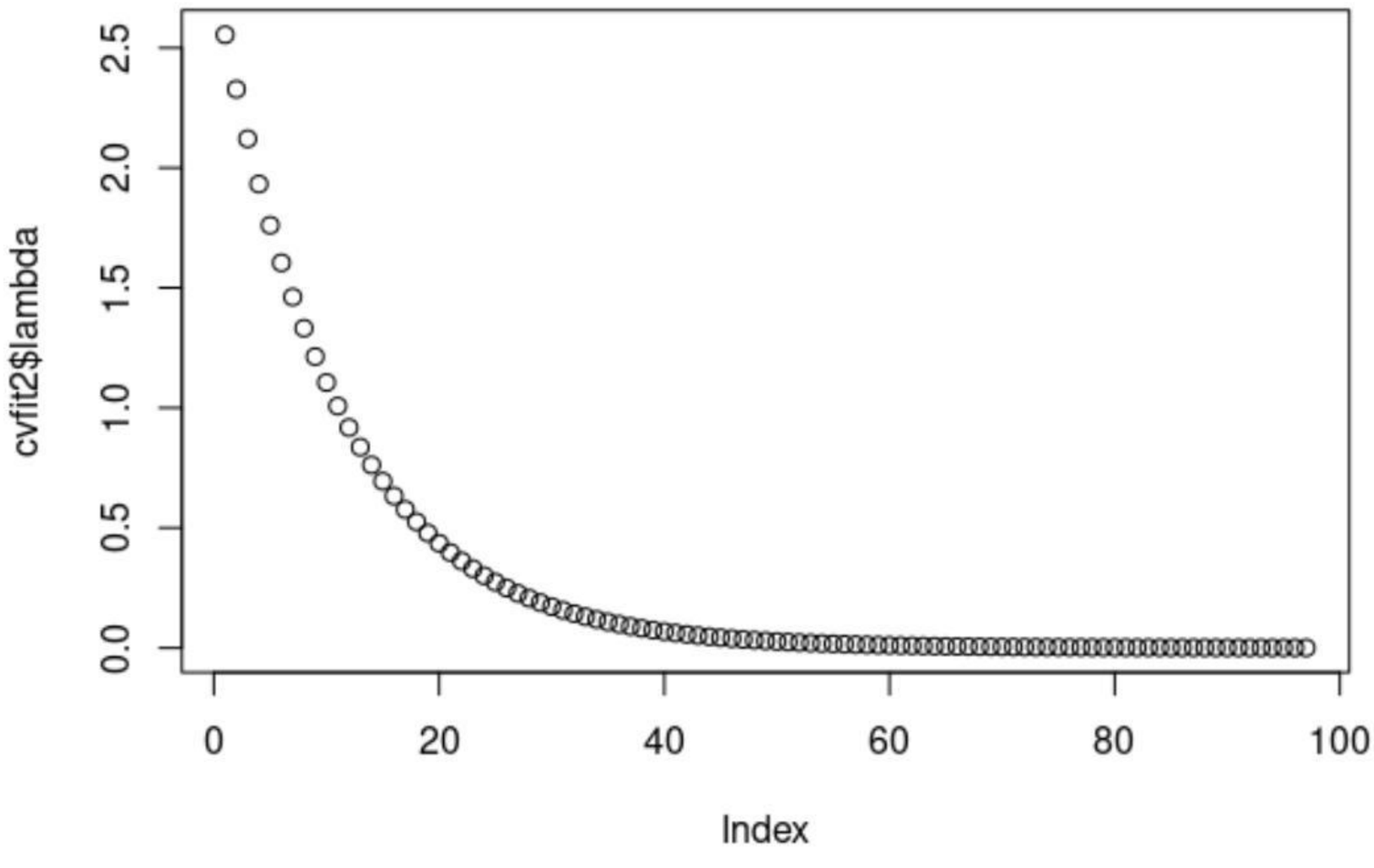

Figure A.6: (II) glmnet Cross Validation Error

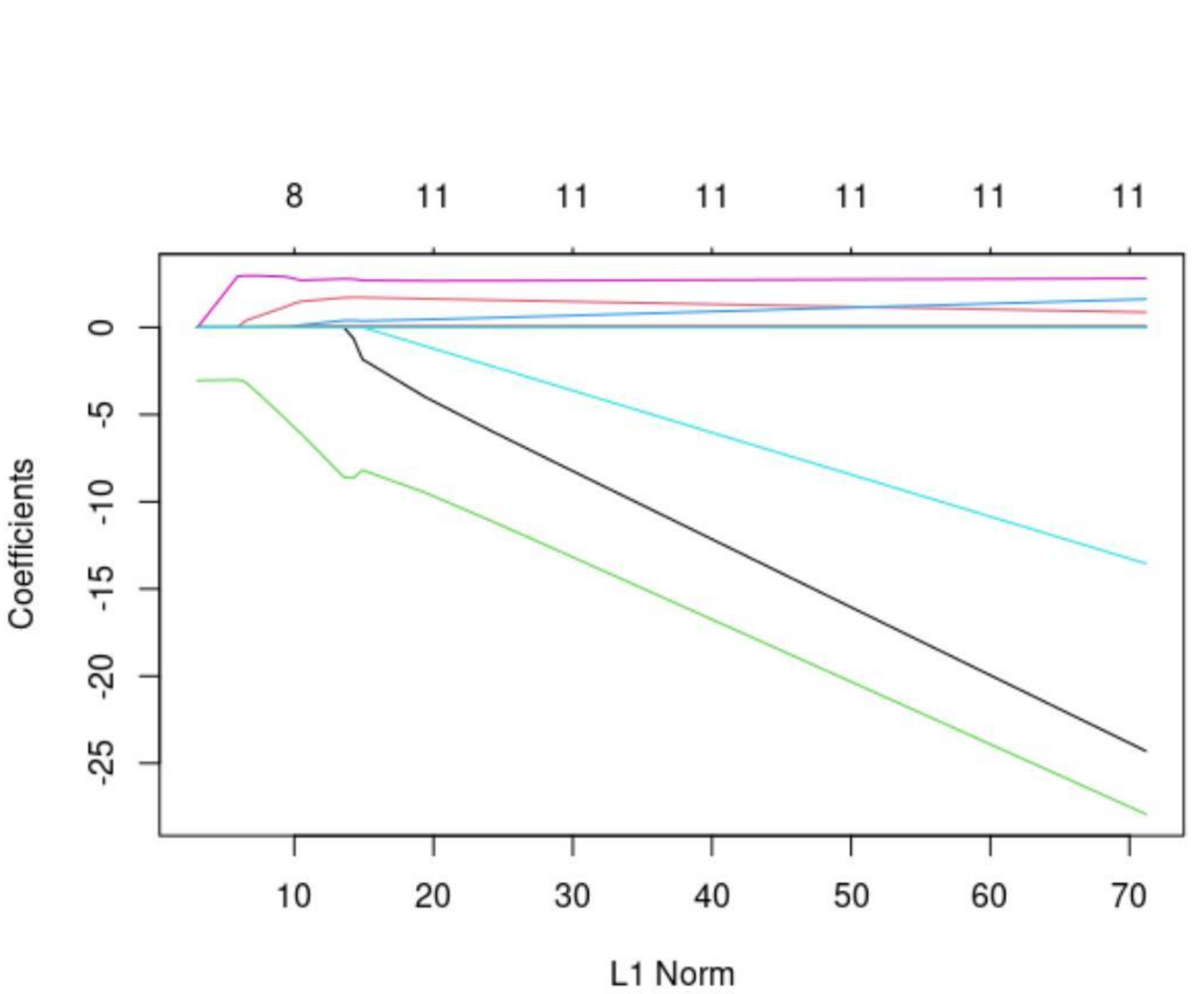

Figure A.7: (II) glmnet Coefficient fit

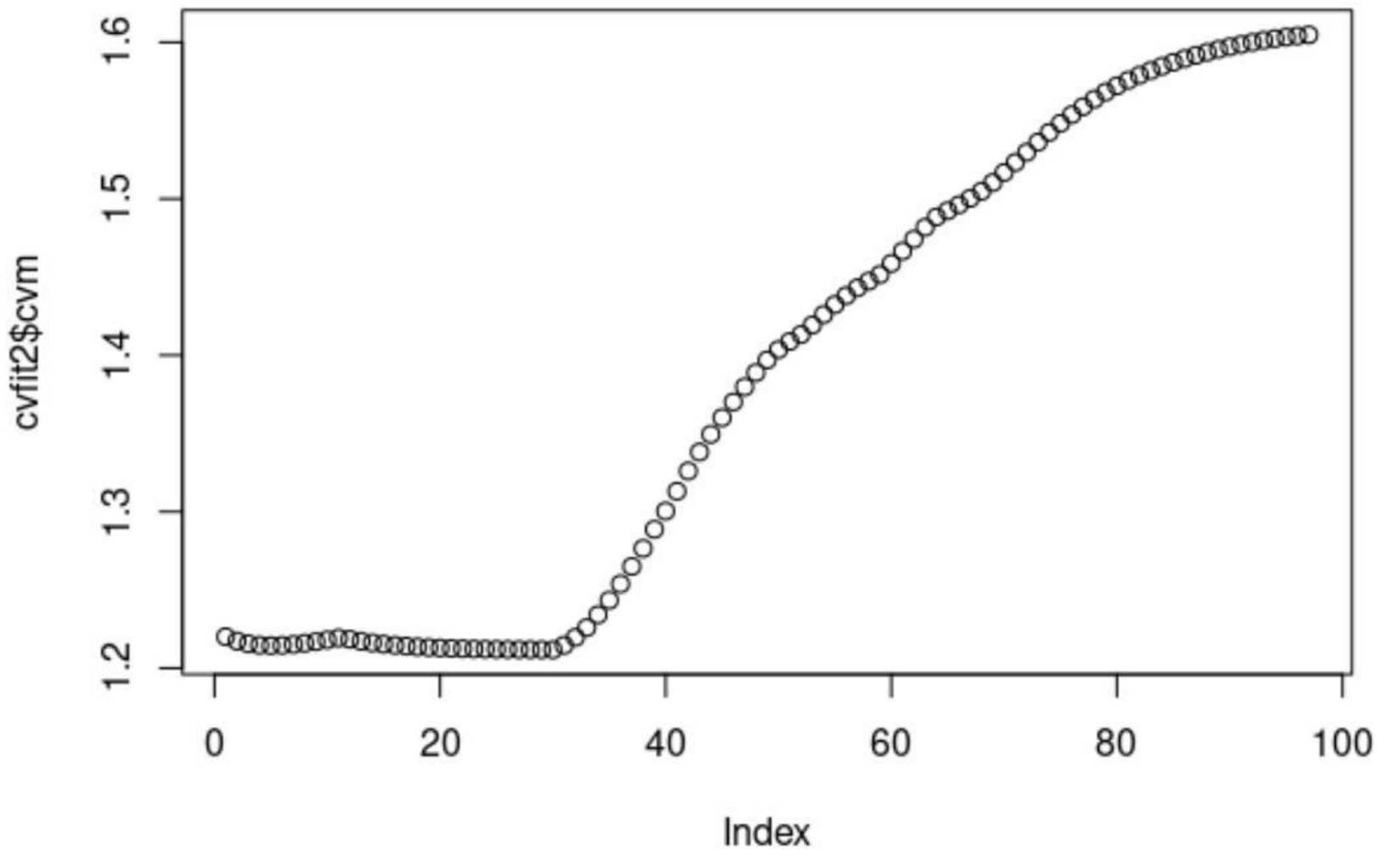

Figure A.8: (II) glmnet Cross Validation mean

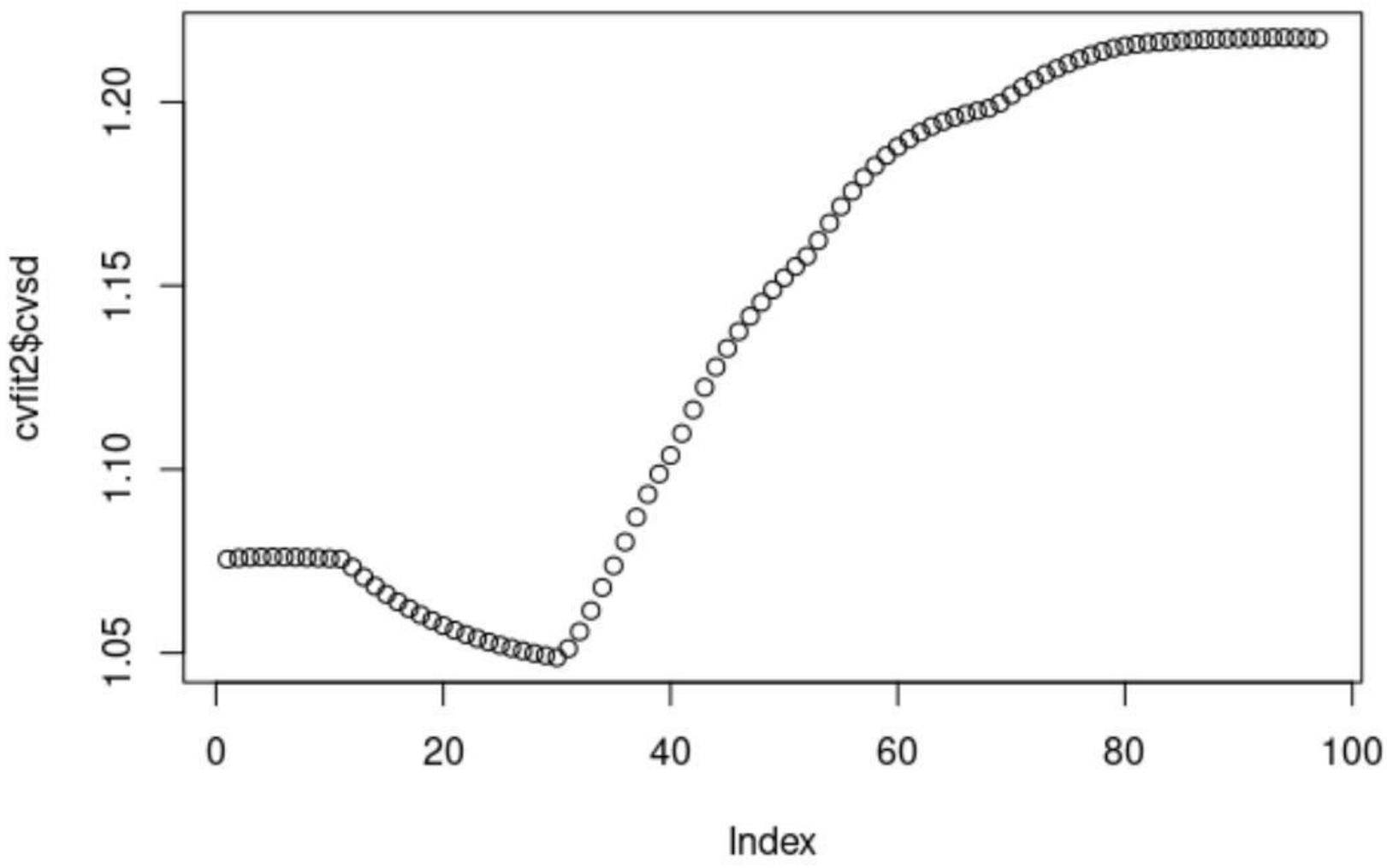

Figure A.9: (II) glmnet Cross Validation standard deviation

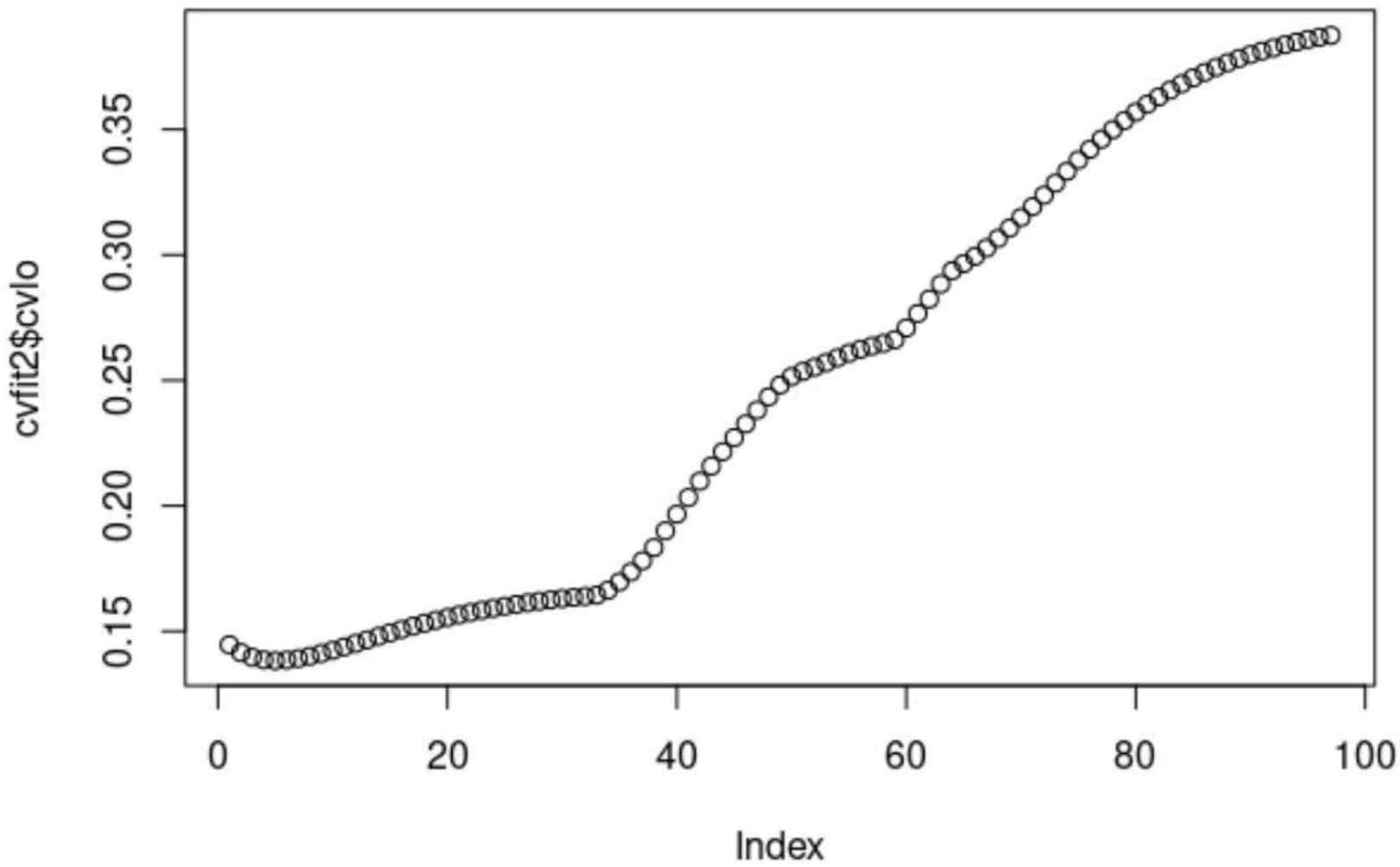

Figure A.10: (II) glmnet Cross Validation lower bound

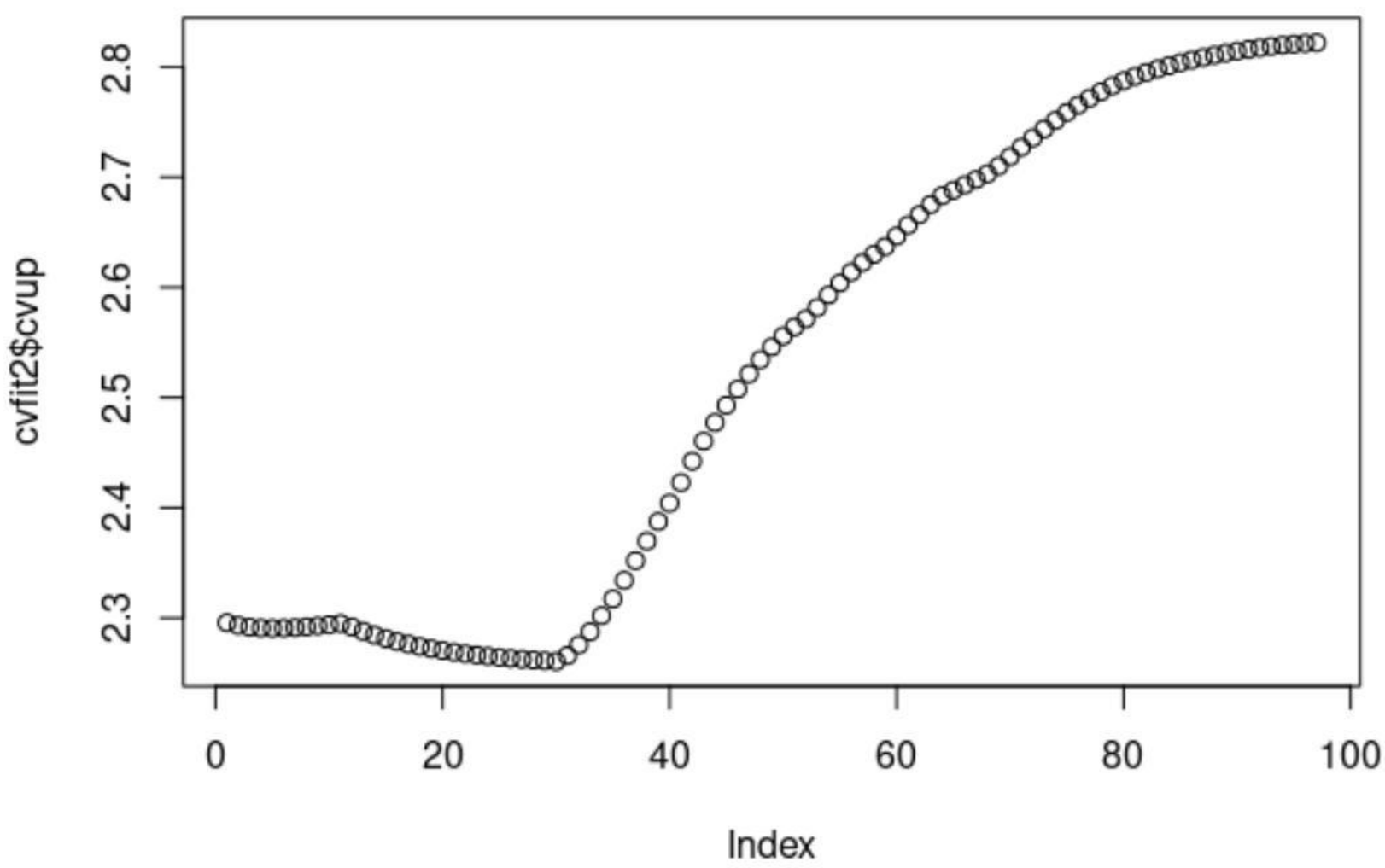

Figure A.11: (II) glmnet Cross Validation upper bound

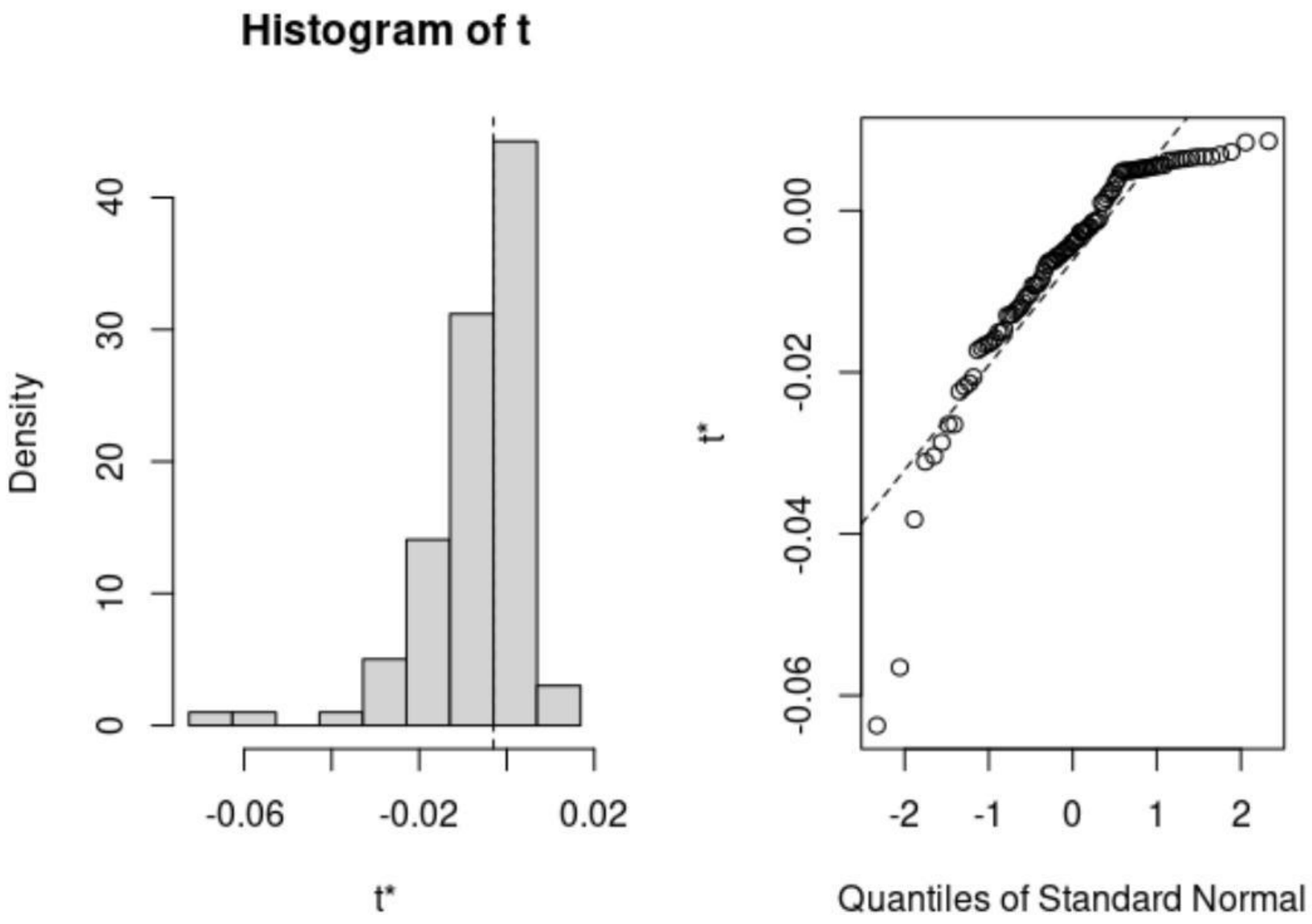

Figure A.12: (II) Boot glmnet plot

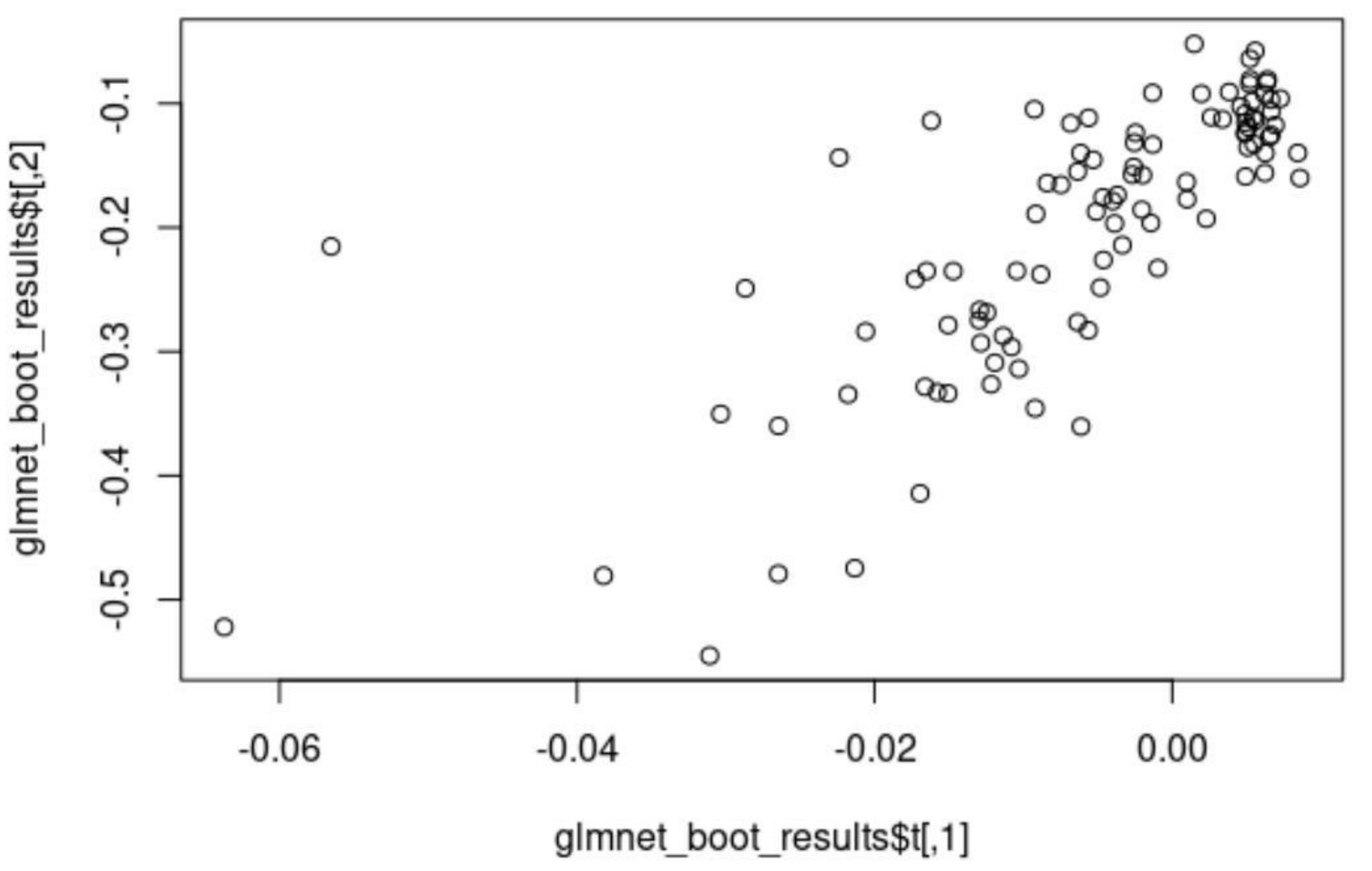

Figure A.13: (II) Boot glmnet Coefficients plot

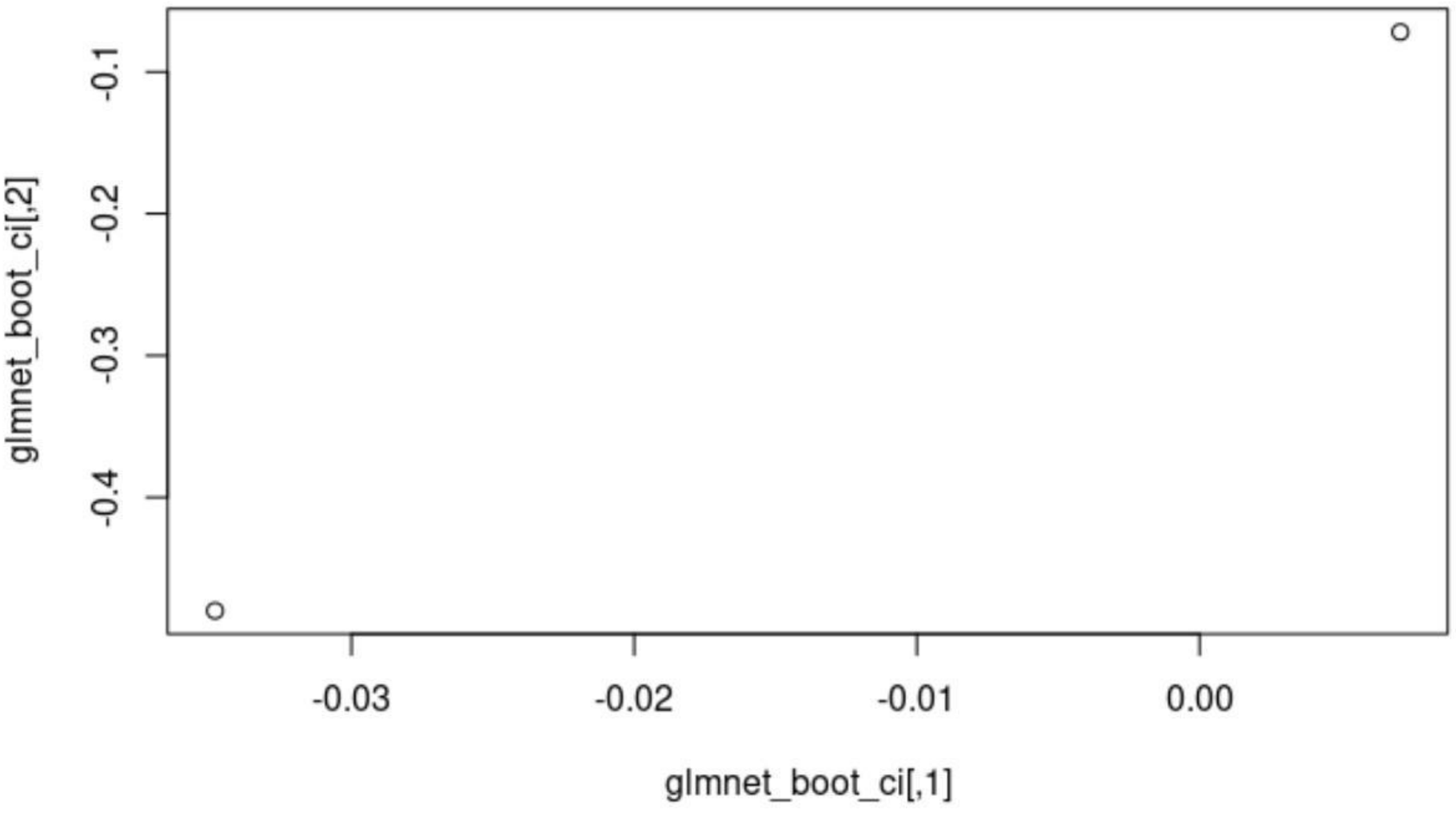

Figure A.14: (II) Boot glmnet CI plot

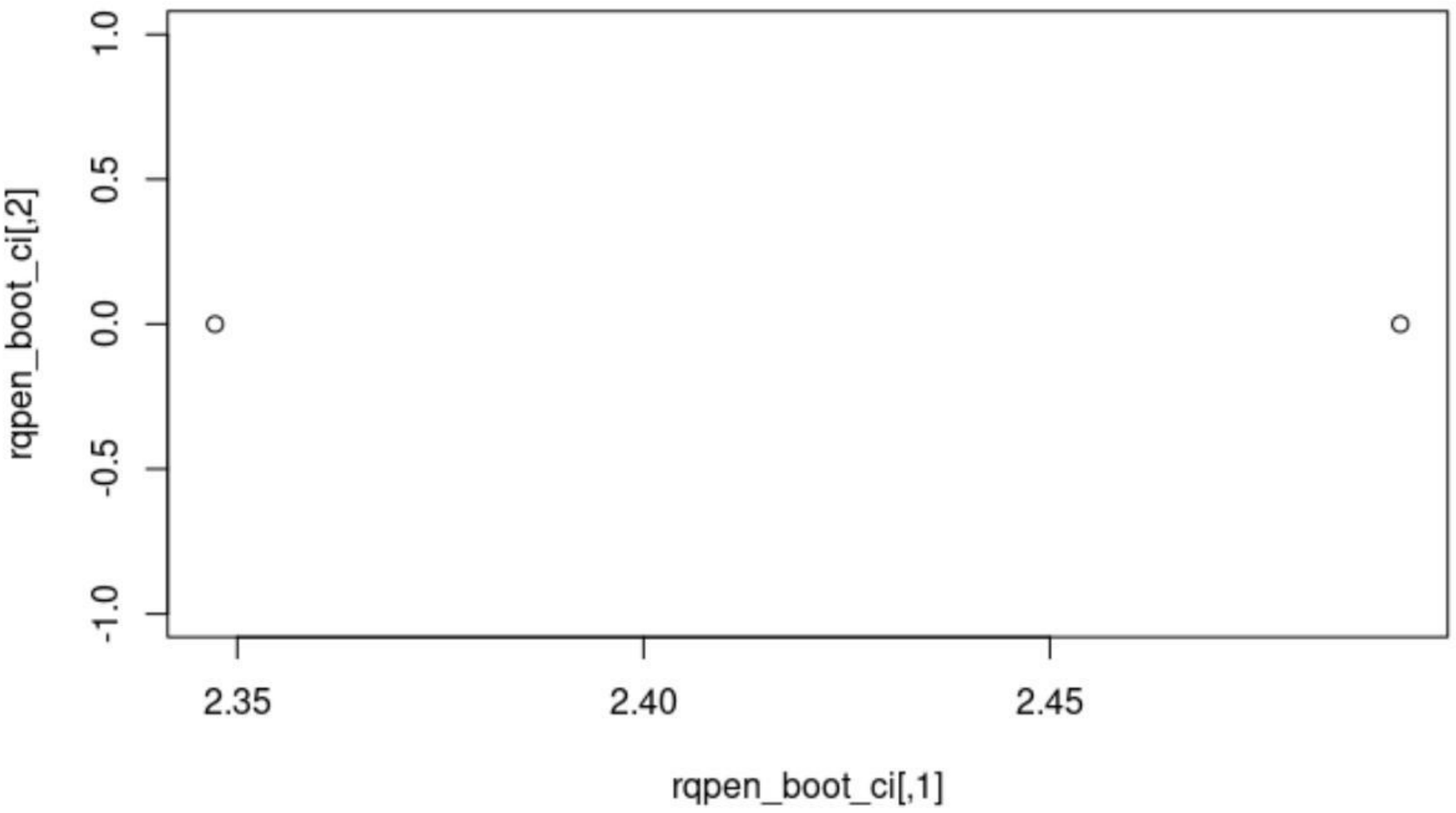

Figure A.15: (II) Boot rqpen CI plot

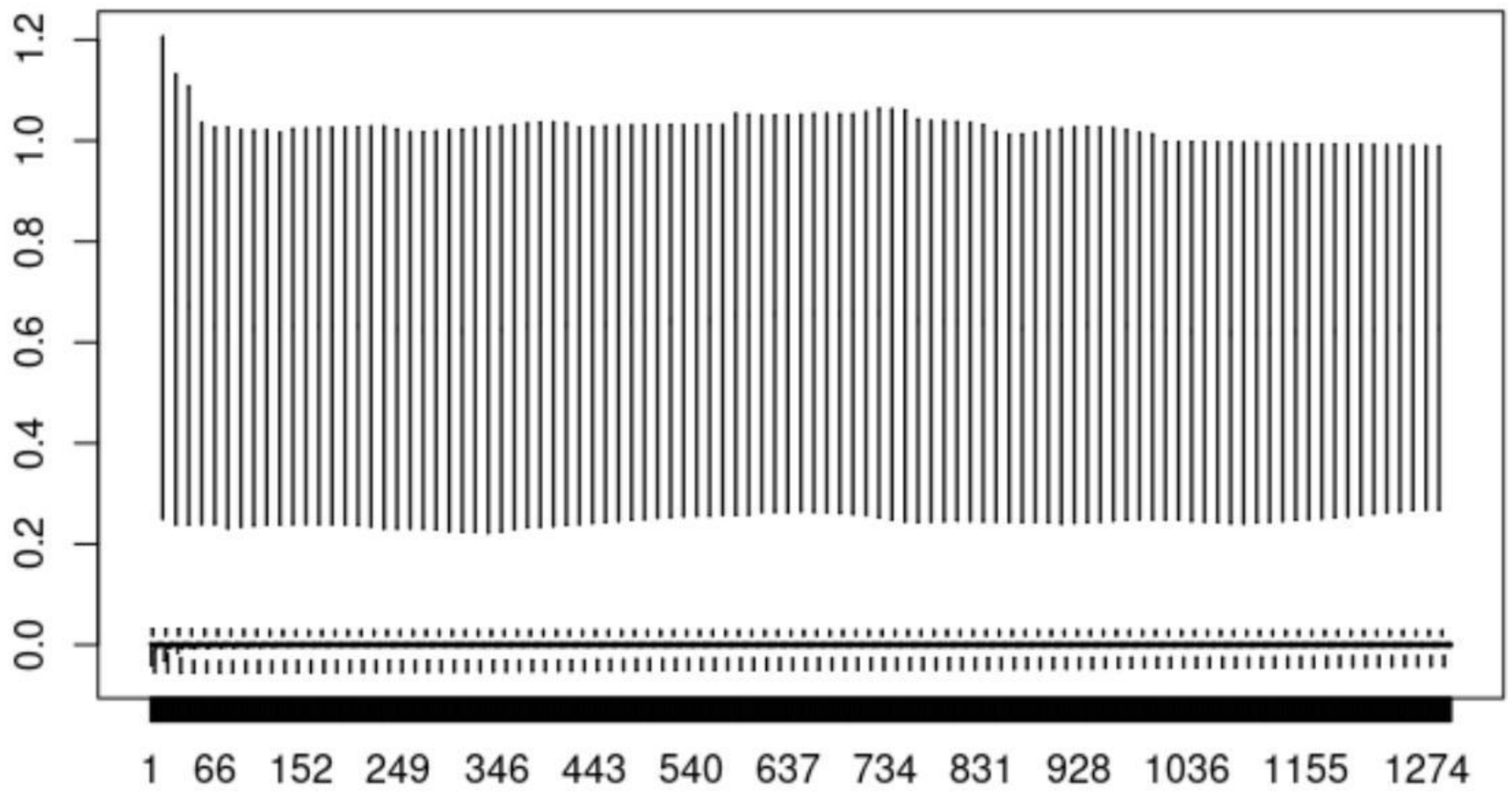

Figure A.16: (II) Boot hqreg CI plot

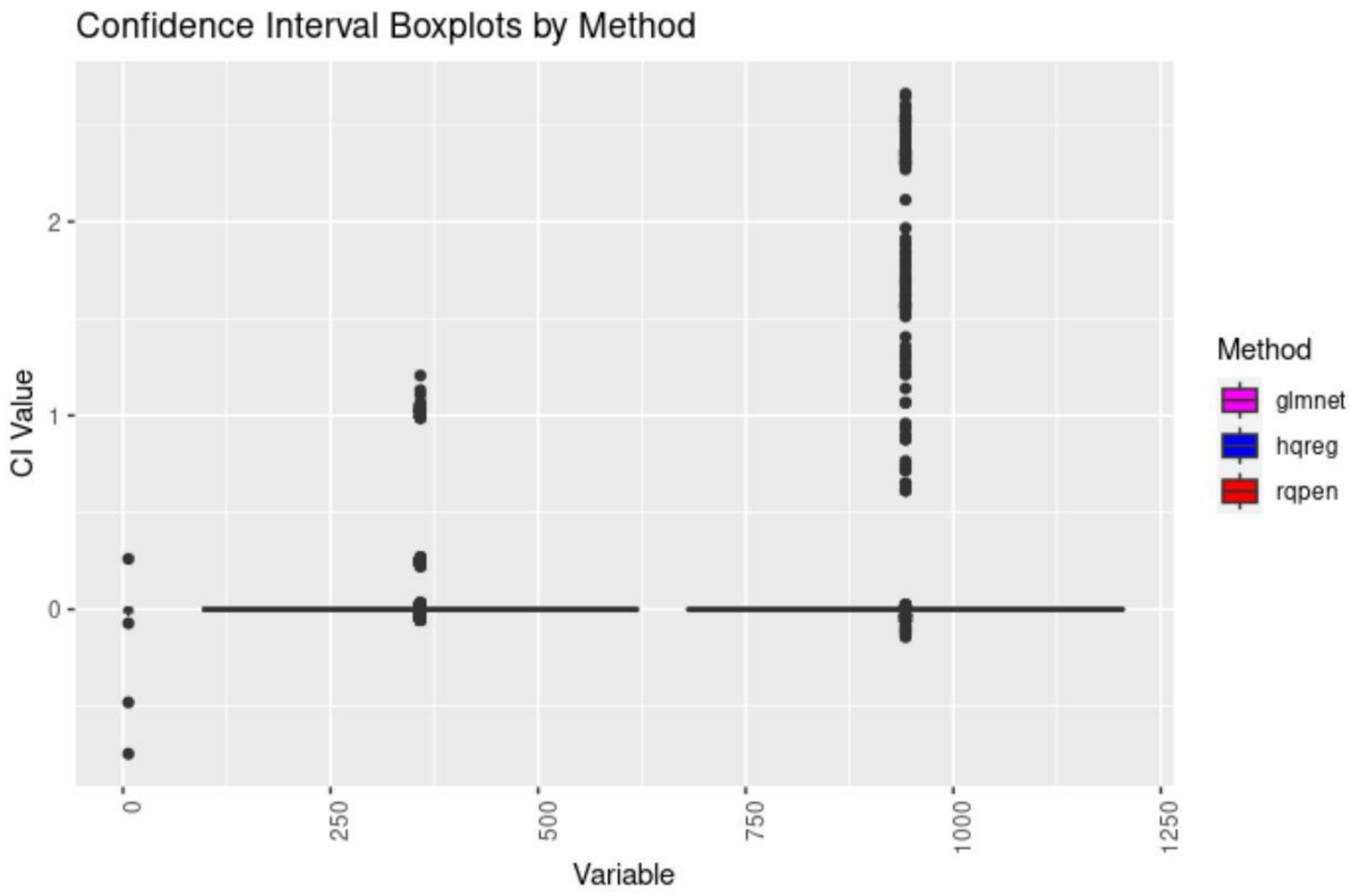

Figure A.17: CI Box plot by method

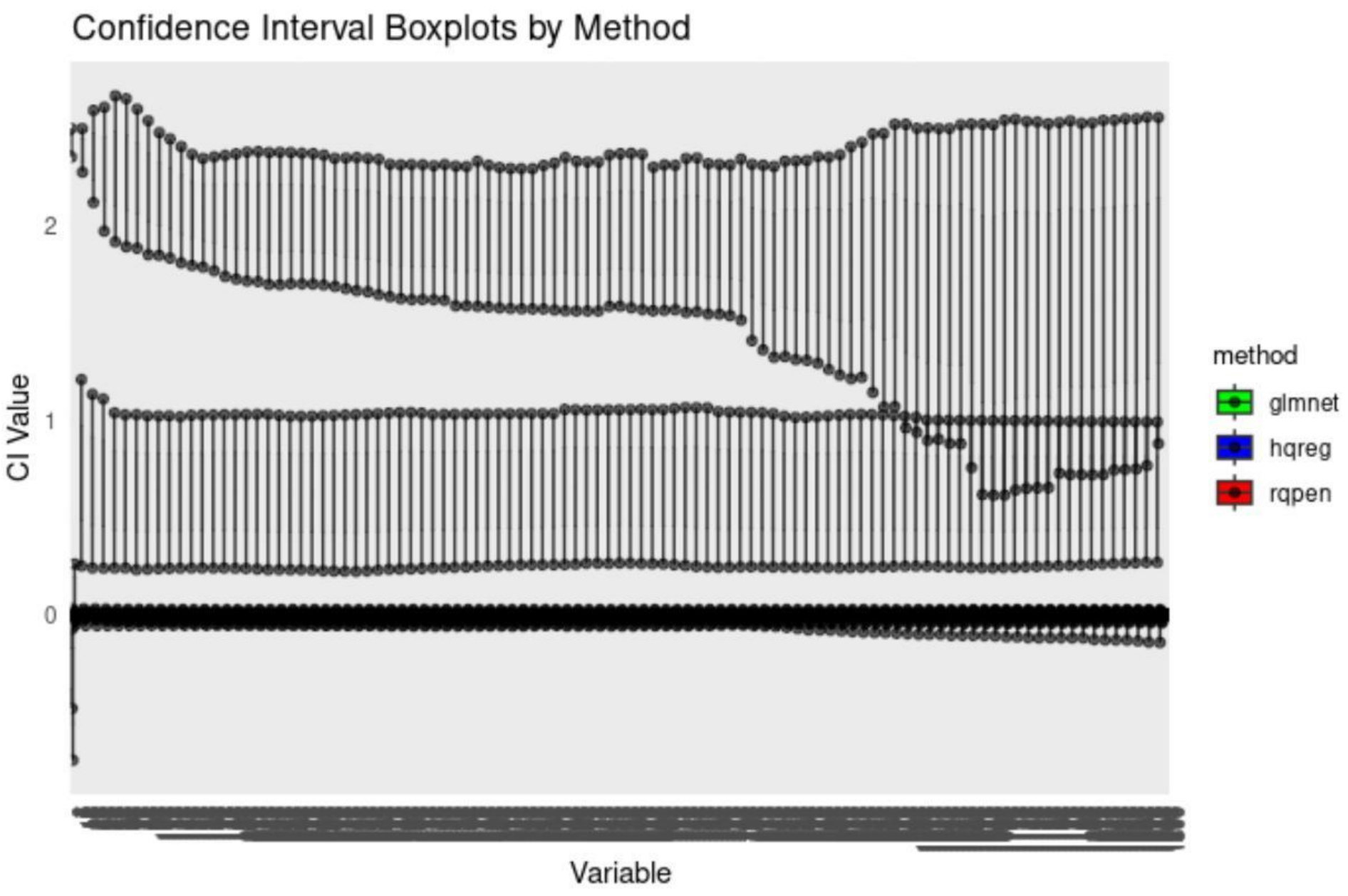

Figure A.18: Box plot by subset and method