QR_Assign

Statistics and Data Science

2015313693 통계학과

권남택

```
library(tidyverse)

## -- Attaching packages ---------------------------------------------------
 tidyverse 1.3.0 --

## √ ggplot2 3.2.1     √ purrr   0.3.3
## √ tibble  2.1.3     √ dplyr   0.8.3
## √ tidyr   1.0.0     √ stringr 1.4.0
## √ readr   1.3.1     √ forcats 0.4.0

## -- Conflicts ------------------------------------------------------------tidyv
erse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(quantreg)

## Loading required package: SparseM

##
## Attaching package: 'SparseM'

## The following object is masked from 'package:base':
##
##     backsolve

library(glmnet)

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 3.0-1

library(mvtnorm)
```

우리의 데이터는 이러하다.

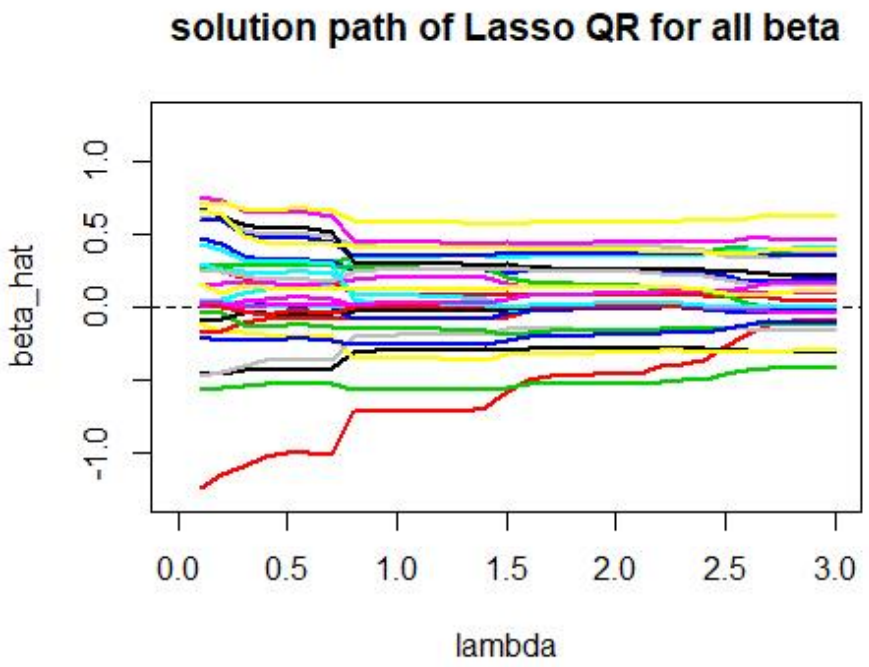
```r
set.seed(2019)
n = 60; p = 30
x = matrix(rnorm(n * p), ncol = p)
x = cbind(rep(1,n), x)

tau = 0.5
sd = 1
beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
obs_err = rnorm(n, sd = sd)
x[,2] = runif(n)
y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
beta_true = beta
beta_true[2] = beta[2] + tau * qnorm(tau)
beta_true[1] = beta[1] + qnorm(tau, sd = sd)
```

Q1. Create a solution path of Lasso QR with tau=0.5 using lambda = seq(0.1, 3, 0.1) for the first seven components.

```r
lambda = seq(0.1, 3, 0.1)

simul_beta <- matrix(0, nrow = length(lambda), ncol = length(beta))
for (i in seq_along(lambda)) {
  result <- rq(y ~ x[,-1], tau = tau, method = 'lasso', lambda = lambda[i])
  simul_beta[i, ] <- result$coefficients
}
```

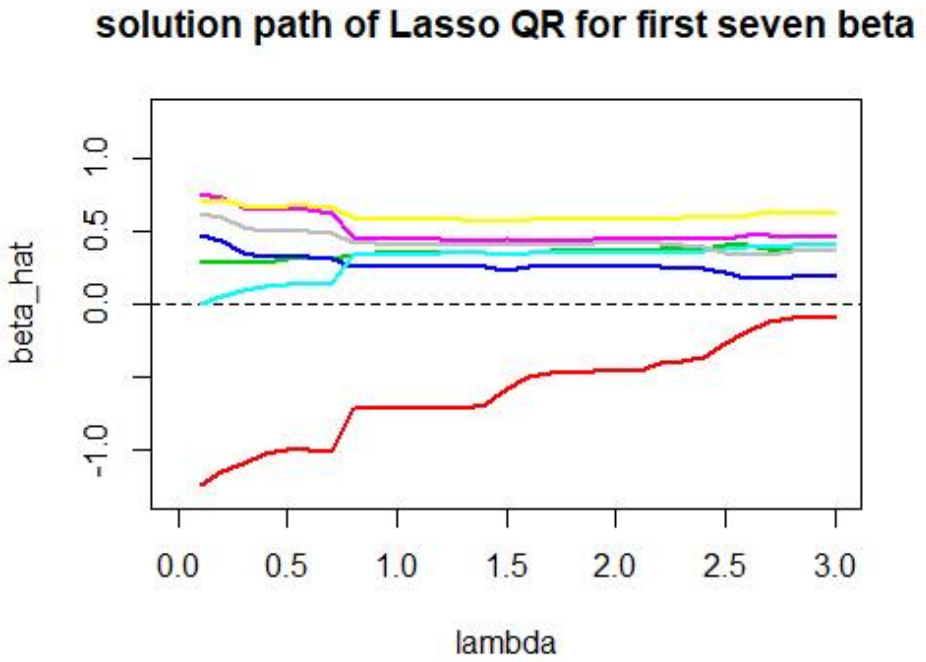
람다 0.1 부터 3 까지 0.1 단위로 놓았다. 행렬에 행별로 람다에 따른 베타 계수를 넣는다.

```r
plot(c(0, 3), c(-1.3, 1.3), type = 'n', main = 'solution path of Lasso QR for all bet
a',
     xlab = 'lambda', ylab = 'beta_hat')
abline(h = 0, lty = 2)
for (i in 1:(length(beta)-1)) {
  lines(x = lambda, y = simul_beta[, i+1], col = (i+1), lwd = 2)
}
```



모든 베타에 대해서 람다에 따른 solution path 를 그렸다. 너무 많은 베타가 있기 때문에 정확히 보이지는 않지만, 수렴하는 베타가 많아보이지는 않는다.

```
plot(c(0, 3), c(-1.3, 1.3), type = 'n', main = 'solution path of Lasso QR for first s
even beta',
     xlab = 'lambda', ylab = 'beta_hat')
abline(h = 0, lty = 2)
for (i in 1:7) {
  lines(x = lambda, y = simul_beta[, i+1], col = (i+1), lwd = 2)
}
```



원래 7 개의 non-zero 베타에 대해서 solution path 를 그렸다. 0 으로 수렴하는 베타는 없어보인다. 아무래도 람다가 3 이 최대인 것은 좁다고 판단할 수 있다. 따라서 3 번부터는 람다의 범위를 늘리고 간격은 좁혀서 보겠다.

Q2. Write your code to perform 10 fold cross-validation to determine penalty parameters (regularization parameters) for Lasso and apply it to the simulation model. Does the determined lambda gives a descent result compared to other regularization parameter values?

```
lambda = seq(0.1, 3, 0.1)
some_lasso <- glmnet(x, y, family = 'gaussian', alpha = 0, lambda = lambda) # lambda 가 큰거부터 정렬

some_lasso_coef <- matrix(0, nrow = 31, ncol = 30)
for (i in seq_along(lambda)) {
  some_lasso_coef[1, i] <- some_lasso$a0[i]  # beta zero
  some_lasso_coef[-1, i] <- some_lasso$beta[-1, i]  # each slope
}   # 각 열에 30 개의 람다에 대한 lasso fit 의 베타계수를 저장해둠
head(some_lasso_coef)
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
## [1,]  0.20928569  0.21525166  0.22154840  0.22820824  0.23526394  0.24275238
## [2,] -0.05591151 -0.05931434 -0.06296943 -0.06690824 -0.07116086 -0.07576159
## [3,]  0.20816216  0.21205003  0.21607794  0.22025608  0.22459228  0.22909474
## [4,]  0.17297571  0.17644318  0.18005370  0.18382025  0.18775362  0.19186564
## [5,]  0.12781645  0.12997633  0.13220866  0.13451184  0.13688837  0.13934072
## [6,]  0.11482752  0.11766041  0.12063976  0.12378045  0.12709649  0.13060364
```

```
##             [,7]        [,8]        [,9]       [,10]      [,11]      [,12]
## [1,]  0.25071533  0.25920017  0.26826103  0.27795996  0.2883685  0.2995697
## [2,] -0.08074969 -0.08617029 -0.09207551 -0.09852588 -0.1055920 -0.1133569
## [3,]  0.23377214  0.23863351  0.24368829  0.24894621  0.2544173  0.2601116
## [4,]  0.19616936  0.20067922  0.20541125  0.21038331  0.2156154  0.2211300
## [5,]  0.14187126  0.14448216  0.14717534  0.14995230  0.1528139  0.1557603
## [6,]  0.13431967  0.13826473  0.14246173  0.14693692  0.1517205  0.1568474
##            [,13]      [,14]      [,15]      [,16]      [,17]      [,18]
## [1,]  0.3116604  0.3247390  0.3389698  0.3545002  0.3715251  0.3902818
## [2,] -0.1219184 -0.1313799 -0.1419091 -0.1536624 -0.1668470 -0.1817191
## [3,]  0.2660390  0.2722006  0.2786185  0.2852938  0.2922307  0.2994298
## [4,]  0.2269525  0.2331182  0.2396486  0.2465875  0.2539793  0.2618767
## [5,]  0.1587904  0.1618989  0.1650854  0.1683398  0.1716498  0.1749971
## [6,]  0.1623582  0.1682910  0.1747194  0.1817029  0.1893223  0.1976762
##            [,19]      [,20]      [,21]      [,22]      [,23]      [,24]
## [1,]  0.4110632  0.4342353  0.4602627  0.4897229  0.5234525  0.5624994
## [2,] -0.1985984 -0.2178894 -0.2401114 -0.2659464 -0.2963111 -0.3324353
## [3,]  0.3068857  0.3145842  0.3224982  0.3305501  0.3387050  0.3468202
## [4,]  0.2703424  0.2794523  0.2893002  0.3000195  0.3117394  0.3246779
## [5,]  0.1783552  0.1816852  0.1849311  0.1880016  0.1907919  0.1931154
## [6,]  0.2068862  0.2171039  0.2285219  0.2413648  0.2560012  0.2728571
##            [,25]      [,26]      [,27]      [,28]      [,29]      [,30]
## [1,]  0.6083445  0.6630982  0.7299807  0.8139589  0.9234108  1.0734519
## [2,] -0.3760477 -0.4297225 -0.4971563 -0.5844719 -0.7016565 -0.8670155
## [3,]  0.3547011  0.3619670  0.3682304  0.3725713  0.3737939  0.3699468
## [4,]  0.3391252  0.3555516  0.3745822  0.3975088  0.4266601  0.4676741
## [5,]  0.1946983  0.1950873  0.1936158  0.1890321  0.1791428  0.1594218
## [6,]  0.2925509  0.3159343  0.3444244  0.3801052  0.4267512  0.4915877
```

```r
# 이제 각각 람다에 대한 rss 를 저장해서 비교한다.
some_lasso_rss <- NULL
for (i in seq_along(lambda)) {
  some_lasso_rss[i] <- sum((y - x %*% some_lasso_coef[, i])^2)
}

some_lasso_rss
```

```
##  [1] 84.53499 83.49894 82.42920 81.32375 80.18074 78.99820 77.77402 76.50589
##  [9] 75.19138 73.82780 72.41228 70.94168 69.41258 67.82183 66.16426 64.43577
## [17] 62.63130 60.74516 58.77092 56.70124 54.52769 52.24145 49.82915 47.27802
## [25] 44.57227 41.69485 38.62808 35.36795 31.96254 28.67087
```

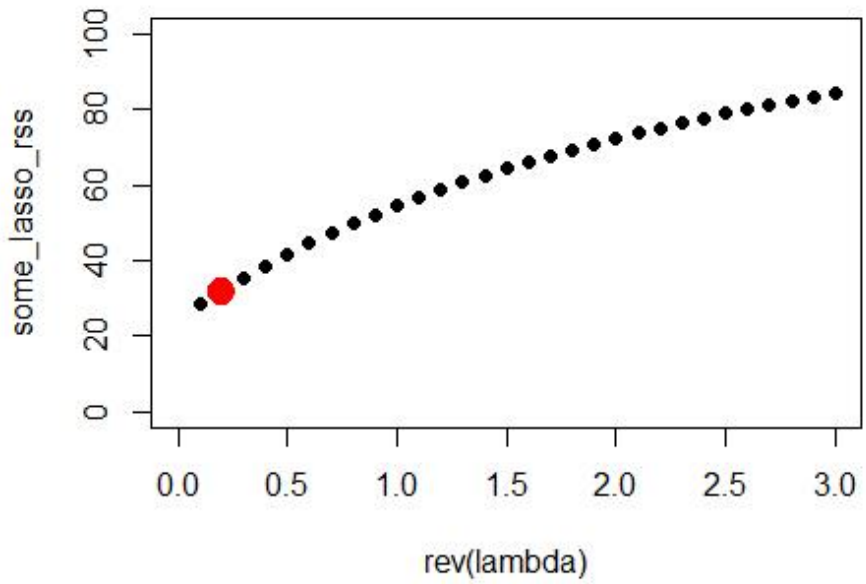
위의 코드는 람다 시퀀스에 대해 전체데이터에 대한 RSS 를 구한 값들을 저장했다. 이제 cv 를 통해 최적의 람다를 구하고, plot 을 통해 비교하려 한다.

```r
lambda = seq(0.1, 3, 0.1)
cv_lasso <- cv.glmnet(x, y, family = "gaussian", alpha = 0, lambda = lambda)
opt_lasso_lambda <- cv_lasso$lambda.min  # 최적의 람다는 0.2
opt_lasso <- glmnet(x, y, family = 'gaussian', alpha = 0, lambda = opt_lasso_lambda)

opt_lasso_coef <- rep(NA, length(beta))
opt_lasso_coef[1] <- opt_lasso$a0
```

```r
opt_lasso_coef[-1] <- opt_lasso$beta[-1]

opt_lasso_rss <- sum((y - x %*% opt_lasso_coef)^2)
opt_lasso_rss # 31.9607

## [1] 31.9607
```

최적의 람다는 0.2 고, 그에따른 전체 데이터의 rss 값은 31.96 이다. 이제 plot 을 그린다.

```r
# Lasso 에서 lambda 가 큰순서대로 정렬되어있기 때문에 rev 로 뒤집어 줘야!
plot(x = rev(lambda), y = some_lasso_rss, pch = 19, type = 'b',
     xlim = c(0, 3), ylim = c(0, 100))

points(x = opt_lasso_lambda, y = opt_lasso_rss, col = 'red', pch = 19, cex = 2)
```



우리의 데이터 셋에서 가장 작은 에러를 만드는 람다는 0.1 이지만, 선택된 람다는 0.2 이다. 물론 cv 로 0.1 을 뽑아내진 못했지만, 매우 가깝다. 데이터 자체의 에러를 최소화하는 람다는 0.1 이지만, 다소 과적합의 위험이 있는 값이다. 하지만 람다 0.2 는 이런 과적합의 위험을 조정하기 위해 cv 를 통해 선택된 값이다. 따라서 0.2 가 더 적절한 값이다.

Q3. As done in linear regression case, we can consider adaptive lasso estimator for quantile regression. Write your code for this.

```r
set.seed(1)
n = 60
tau = 0.5
```

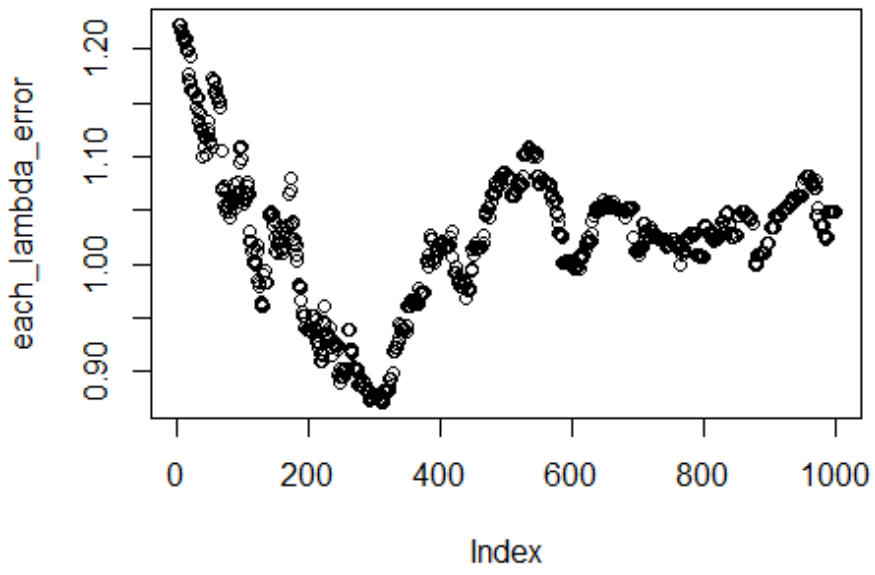
```r
k = 10
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

each_error <- matrix(0, nrow = length(lambda), ncol = k)
# error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], metho
d = 'lasso')
    each_beta <- each_lasso_rq$coefficients
    each_predicted_y <- x[ind_for, ] %*% each_beta
    each_error[j, i] <- quantile(abs(valid_y - each_predicted_y), 0.5) # 에러값 기록
  }
  print(ind_for)
  print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
  # print 는 이번 한번만 보이고, 이후부터는 보이지 않겠다.
}
```

```
## [1] 57  4 39  1 34 23
##  [1]  2  3  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27 28
## [26] 29 30 31 32 33 35 36 37 38 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
## [51] 56 58 59 60
## [1] 48 16 20 56 37 24
##  [1]  2  3  5  6  7  8  9 10 11 12 13 14 15 17 18 19 21 22 25 26 27 28 29 30 31
## [26] 32 33 35 36 38 40 41 42 43 44 45 46 47 49 50 51 52 53 54 55 58 59 60
## [1] 27 52 58 12  9 11
##  [1]  2  3  5  6  7  8 10 13 14 15 17 18 19 21 22 25 26 28 29 30 31 32 33 35 36
## [26] 38 40 41 42 43 44 45 46 47 49 50 51 53 54 55 59 60
## [1] 22 31 51 36 55 53
##  [1]  2  3  5  6  7  8 10 13 14 15 17 18 19 21 25 26 28 29 30 32 33 35 38 40 41
## [26] 42 43 44 45 46 47 49 50 54 59 60
## [1] 54 41 25 50 32  5
##  [1]  2  3  6  7  8 10 13 14 15 17 18 19 21 26 28 29 30 33 35 38 40 42 43 44 45
## [26] 46 47 49 59 60
## [1] 10 17 59 60 28 38
##  [1]  2  3  6  7  8 13 14 15 18 19 21 26 29 30 33 35 40 42 43 44 45 46 47 49
## [1] 44 26 13 15 47 46
##  [1]  2  3  6  7  8 14 18 19 21 29 30 33 35 40 42 43 45 49
## [1] 18 29 14 19 40  3
```

```
##  [1]  2  6  7  8 21 30 33 35 42 43 45 49
## [1]  6 30 45  2  7 35
## [1]  8 21 33 42 43 49
## [1] 49 42 21 33  8 43
## integer(0)
```

10 fold CV 를 통해, lasso qr 의 최적의 람다값을 얻어낸다. 그러기 위해 CV 를 생성하는 함수를 만들고, 거기서 train set 에 피팅한 후 cv error 을 행렬에 값으로 저장한다.

```
each_lambda_error <- apply(each_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_lambda_error)  # cv 를 통한 rss 의 형태가 이런식으로 나타남!
```



람다별로 cv error 을 담은 벡터를 만들고, 그 벡터에 대해 플롯을 만들었다.

```
min_lambda_error <- min(each_lambda_error)
min_lambda_index <- which(each_lambda_error == min_lambda_error)
min_lambda_index

## [1] 312

min_lambda_lasso <- lambda[min_lambda_index]
min_lambda_lasso

## [1] 3.12
```

최소의 cv error 을 지니는 람다의 인덱스를 뽑았다.

```
# optimum lasso rq fit
opt_lasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_lambda_lasso, method = 'la
sso')
beta_lasso_rq <- opt_lasso_rq_fit$coefficients
sum((beta_true[-1] - beta_lasso_rq[-1])^2)  # 1.43
```

```
## [1] 1.437106
```

최적의 람다에 대해 lasso quantile regression 을 시행하고, 그 결과의 베타 값을 저장한다. 이를 바탕으로 우리 데이터에 대한 추정오차의 제곱합을 기록한다.

```
beta_lasso_rq
```

```
##   (Intercept)      x[, -1]1      x[, -1]2      x[, -1]3      x[, -1]4
##  6.225019e-01 -9.253417e-02  3.677471e-01  1.823717e-01  4.002282e-01
##      x[, -1]5      x[, -1]6      x[, -1]7      x[, -1]8      x[, -1]9
##  4.553042e-01  6.241247e-01  3.581758e-01 -3.136110e-01  4.424390e-02
##     x[, -1]10     x[, -1]11     x[, -1]12     x[, -1]13     x[, -1]14
##  9.457327e-10  3.499312e-01  5.080420e-10  9.190926e-02 -3.041975e-01
##     x[, -1]15     x[, -1]16     x[, -1]17     x[, -1]18     x[, -1]19
## -1.576527e-01  2.189765e-01 -5.073253e-10 -1.217027e-01 -1.678637e-02
##     x[, -1]20     x[, -1]21     x[, -1]22     x[, -1]23     x[, -1]24
##  2.777598e-10 -4.202159e-02  3.939004e-01  1.453734e-01  1.241482e-10
##     x[, -1]25     x[, -1]26     x[, -1]27     x[, -1]28     x[, -1]29
##  2.105868e-10 -4.156140e-01 -1.049362e-01  3.325914e-10  1.653644e-01
##     x[, -1]30
##  1.057816e-01
```

원래 라쏘에서는 정확하게 0 으로 찍어주지만, 현재 다른 베타들과 다르게 너무 작은 값들이 존재한다. 아마 quantile regression 에서 계산하는 것의 한계인 것 같다. 소수점 8 자리 밑으로 나오는 값들은 모두 0 으로 봐도 무방하지만, adaptive 라쏘를 적용할때 좀더 편하게 페널티를 고려하기 위해, 이 상태로 놓아 두는 것이 편할 것이다.

그러면 이제 adaptive lasso 를 위한 페널티가 될 베타틸데를 구했다. lasso 베타의 역수가 베타틸데가 될 것이다. 이제 다시 adpative lasso 의 최적의 람다 값을 찾아야 하는데, 시드를 다르게 해서 람다값을 찾을 것이다. adpative lasso 는 계층적인 알고리즘을 가지고 있기 때문에, cv 도 계층적으로 짜야하겠지만, 우리 관측수가 많지 않으므로, seed 를 다르게 하겠다. 따라서 overfitting 의 가능성을 배제할 수 없다는 점은 고려하자.

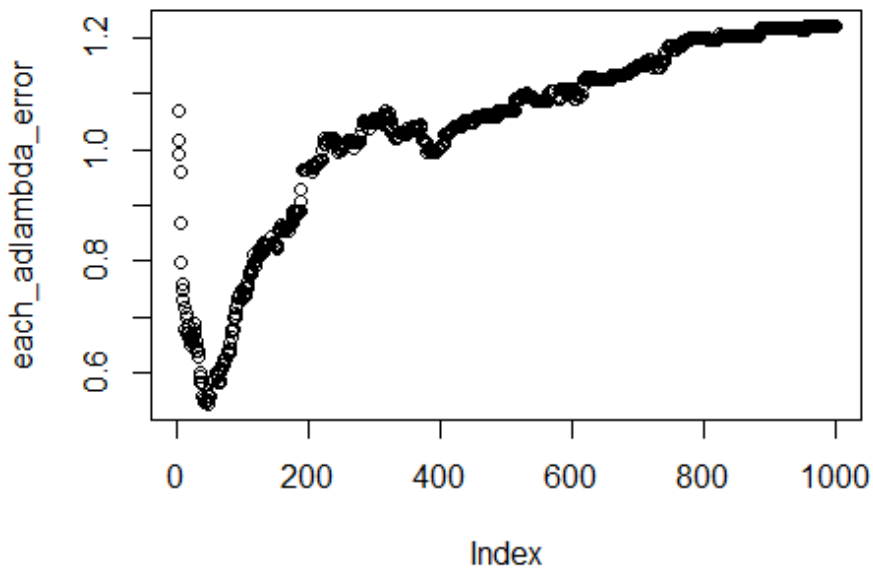
전체적인 과정은 이전과 거의 동일하다.

```
set.seed(2)  # seed 를 다르게 설정
n = 60
tau = 0.5
k = 10
```

```r
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

weight_beta <- beta_lasso_rq %>% abs  # 베타 절대값의 역수만큼 weight 을 줘야하니 바꿔준다.
weight_beta <- weight_beta[2:length(weight_beta)]  # 베타0 의 가중치는 0 이 되어야하니 제외

each_ad_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    penalty <-c(0, lambda[j] / weight_beta)  # 베타 0 에 대한 lambda_0 를 0 으로 만들어준다.
    # 각각 람다에 대해 lasso beta 로 나눈 값을 고려하고 있다.
    each_adlasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = penalty, method = 'lasso')
    each_ad_beta <- each_adlasso_rq$coefficients
    each_ad_predicted_y <- x[ind_for, ] %*% each_ad_beta
    each_ad_error[j, i] <-  quantile(abs(valid_y - each_ad_predicted_y), 0.5)
  }
  #print(ind_for)
  #print(ind)
}
```

전체적인 과정은 비슷하다. adpative lasso 의 weight 를 베타틸데의 역수를 넣는다. 이때 베타 제로의 가중치는 0 이라는 점을 고려하자.

```r
each_adlambda_error <- apply(each_ad_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_adlambda_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```



똑같이 cv 오차제곱합을 구한다.

```
min_adlambda_error <- min(each_adlambda_error)
min_adlambda_index <- which(each_adlambda_error == min_adlambda_error)
min_adlambda_index

## [1] 46

min_adlambda_lasso <- lambda[min_adlambda_index]
min_adlambda_lasso

## [1] 0.46

# optimum adaptive lasso rq fit
opt_adlasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_adlambda_lasso, method =
 'lasso')
beta_adlasso_rq <- opt_adlasso_rq_fit$coefficients

length(which(abs(beta_adlasso_rq) > 10^(-5)))

## [1] 30

# error for Adaptive Lasso Quantile Regression
sum((beta_true - beta_adlasso_rq)^2)  # 4.82

## [1] 4.824572
```

우리 데이터에 대한 adaptive laaso 의 추정오차제곱합은 4.82 이다.

Q4. We can also consider weighted lasso estimator for quantile regression by using SCAD derivative function. Write your code for this, and compare this estimator with the adaptive lasso estimator. For comparisons, you can choose lambda using cross-validation.

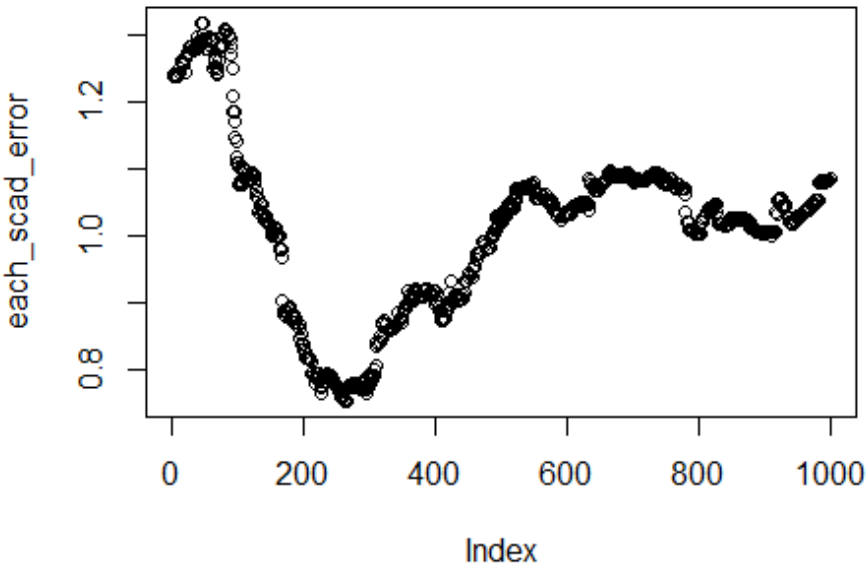
```
scad_deriv = function(x,lam,gam=3.7){
  (abs(x) <= lam)*lam +
    (abs(x)>lam)*(abs(x)<gam*lam)*(gam*lam - abs(x))/(gam-1)
}
```

테일러 근사를 통한 스캐드 미분함수를 정의한다. 이때 x = beta 값은 3 번에서 구한 lasso qr 의 베타값을 넣은 예정이다. 그리고 람다 시퀀스에 대한 우리의 최적의 람다를 10 fold-CV 를 통해 구하겠다.

```
set.seed(2)
ind <- 1:n
ind_all <- 1:n
lambda = seq(0.01, 10, by = 0.01)

each_scad_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    weight = c(0, scad_deriv(abs(beta_lasso_rq[-1]), lam = lambda[j]))
    # 베타 0 에 대한 lambda_0 를 0 으로 만들어준다.
    # 각각 람다에 대해 스캐드 미분함수를 고려한다.
    each_scad_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = weight, method = 'lasso')
    each_scad_beta <- each_scad_rq$coefficients
    each_scad_predicted_y <- x[ind_for, ] %*% each_scad_beta
    each_scad_error[j, i] <-  quantile(abs(valid_y - each_scad_predicted_y), 0.5)
  }
  #print(ind_for)
  #print(ind)
}
```

각각 베타값에 대한 스캐드 미분함수를 람다값들에 대응시킨다. 이를 weight 로 넣어준다. 이를 바탕으로 cv error 을 계산한다.

```r
each_scad_error <- apply(each_scad_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_scad_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```




```r
min_scad_error <- min(each_scad_error)
min_scad_index <- which(each_scad_error == min_scad_error)
min_scad_index

## [1] 265

min_scad_lasso <- lambda[min_scad_index]
min_scad_lasso

## [1] 2.65
```

이전과 같은 방식으로 최소의 에러를 만들어내는 람다값을 뽑는다.

```r
# optimum weighted lasso rq fit with scad derivative funtion
opt_weight = c(0, scad_deriv(abs(beta_lasso_rq[-1]), lam = min_scad_lasso))
opt_scad_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = opt_weight, method = 'lasso')

beta_scad_rq <- opt_scad_rq_fit$coefficients
length(which(abs(beta_scad_rq) > 10^(-5)))

## [1] 25
```

이제 구한 람다값을 스캐드 미분함수에 넣는다. 그리고 lasso rq 의 베타값을 넣어서, 이를 우리가 계산하려는 scad rq 에 넣어준다.

```
# error for weighted lasso rq fit with scad derivative funtion
sum((beta_true - beta_scad_rq)^2) # 1.59
```

```
## [1] 1.59129
```

그리고 이때의 추정오차의 제곱합은 1.59 이다.

```
# error for Adaptive Lasso Quantile Regression
sum((beta_true - beta_adlasso_rq)^2)  # 4.82
```

```
## [1] 4.824572
```

```
# error for weighted lasso rq fit with scad derivative funtion
sum((beta_true - beta_scad_rq)^2) # 1.59
```

```
## [1] 1.59129
```

현재 우리의 결과로는 scad 가 추정한 베타값이 더 정확해보인다.

Q5. Consider the following quantile regresion model with tau = 0.25 and tau = 0.75, respectively. Obatin estimation errors for the selected penalty parameters via 10 fold cross-validation for Lasso, adaptive Lasso, and weighte Lasso (using SCAD derivative). And compare them.

tau = 0.25

먼저 tau = 0.25 인 경우에 대해 시행하겠다. 데이터 다시 고려하겠다.

```
set.seed(2019)
n = 60; p = 30
x = matrix(rnorm(n * p), ncol = p)
x = cbind(rep(1,n), x)

tau = 0.25
sd = 1
beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
obs_err = rnorm(n, sd = sd)
x[,2] = runif(n)
y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
beta_true = beta
beta_true[2] = beta[2] + tau * qnorm(tau)
beta_true[1] = beta[1] + qnorm(tau, sd = sd)
```

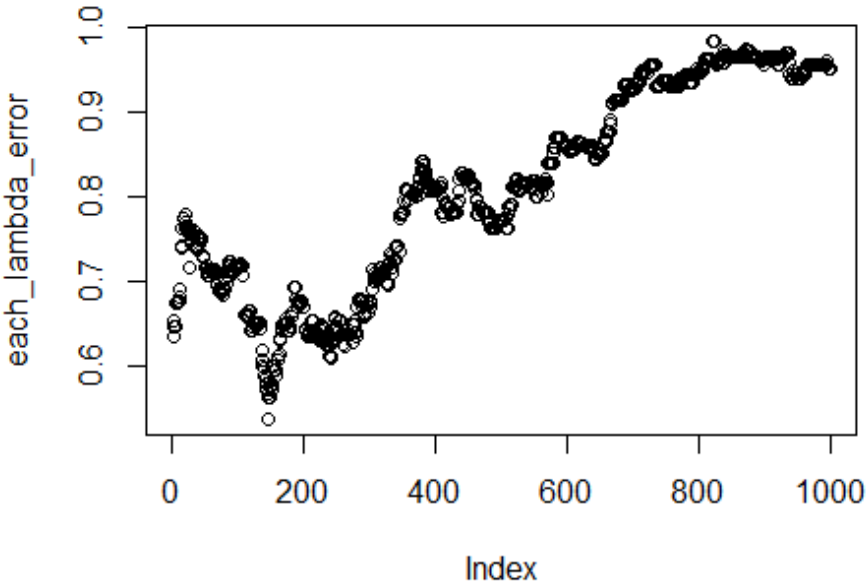
heteroscadicity 를 고려하면, tau 에 따라 바꿔줘야하기 때문이다. 이제 모형별로 비교하려고 한다. 이때 동일한 cv 안에서 비교해야하기 때문에, 같은 층위에서 비교할 모델끼리는 `set.seed(2)`로 지정하겠다. adaptive lasso 와 scad 에서 고려될 베타틸데에 대해서는 `set.seed(1)`로 고려하겠다.

lasso 에 대해서

```
# lasso 에 대해서
set.seed(2)
k = 10
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

each_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], metho
d = 'lasso')
    each_beta <- each_lasso_rq$coefficients
    each_predicted_y <- x[ind_for, ] %*% each_beta
    each_error[j, i] <- quantile(abs(valid_y - each_predicted_y), 0.25)
  }
  #print(ind_for)
  #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}
```

이전과 동일한 방식으로 cv 를 나누고, 이를 통해 람다별 에러를 기록한다.

```
each_lambda_error <- apply(each_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_lambda_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```




```
min_lambda_error <- min(each_lambda_error)
min_lambda_index <- which(each_lambda_error == min_lambda_error)
min_lambda_index  #

## [1] 147

min_lambda_lasso <- lambda[min_lambda_index]
min_lambda_lasso

## [1] 1.47

# optimum lasso rq fit

opt_lasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_lambda_lasso, method = 'la
sso')
beta_optlasso_rq <- opt_lasso_rq_fit$coefficients
length(which(abs(beta_optlasso_rq) > 10^(-5)))

## [1] 30

sum((beta_true - beta_optlasso_rq)^2) #2.615134

## [1] 2.615134
```

lasso 의 추정오차 제곱합은 2.61 이다.

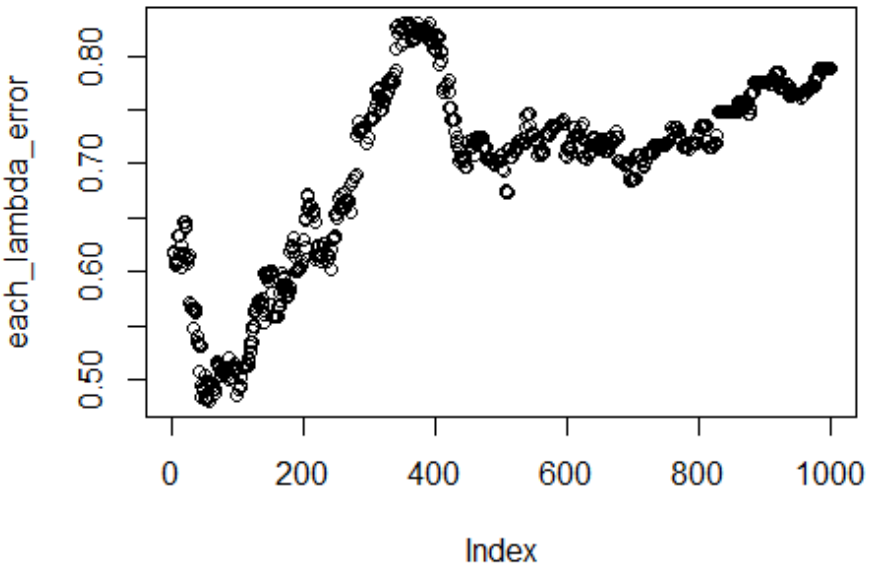
adaptive lasso 에 대해서

adaptive lasso rq 와 scad rq 에 쓰일 베타틸데(lasso qr)을 구하기 위해, lasso qr 을 피팅한다. 이때 시드는 1 로 잡아서 하자.

```
set.seed(1)
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

each_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], metho
d = 'lasso')
    each_beta <- each_lasso_rq$coefficients
    each_predicted_y <- x[ind_for, ] %*% each_beta
    each_error[j, i] <- quantile(abs(valid_y - each_predicted_y),0.25)
  }
  print(ind_for)
  print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}
```

```
## [1] 57  4 39  1 34 23
##  [1]  2  3  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27 28
## [26] 29 30 31 32 33 35 36 37 38 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
## [51] 56 58 59 60
## [1] 48 16 20 56 37 24
##  [1]  2  3  5  6  7  8  9 10 11 12 13 14 15 17 18 19 21 22 25 26 27 28 29 30 31
## [26] 32 33 35 36 38 40 41 42 43 44 45 46 47 49 50 51 52 53 54 55 58 59 60
## [1] 27 52 58 12  9 11
##  [1]  2  3  5  6  7  8 10 13 14 15 17 18 19 21 22 25 26 28 29 30 31 32 33 35 36
## [26] 38 40 41 42 43 44 45 46 47 49 50 51 53 54 55 59 60
## [1] 22 31 51 36 55 53
##  [1]  2  3  5  6  7  8 10 13 14 15 17 18 19 21 25 26 28 29 30 32 33 35 38 40 41
## [26] 42 43 44 45 46 47 49 50 54 59 60
## [1] 54 41 25 50 32  5
##  [1]  2  3  6  7  8 10 13 14 15 17 18 19 21 26 28 29 30 33 35 38 40 42 43 44 45
## [26] 46 47 49 59 60
## [1] 10 17 59 60 28 38
##  [1]  2  3  6  7  8 13 14 15 18 19 21 26 29 30 33 35 40 42 43 44 45 46 47 49
## [1] 44 26 13 15 47 46
```

```
##  [1]  2  3  6  7  8 14 18 19 21 29 30 33 35 40 42 43 45 49
## [1] 18 29 14 19 40  3
##  [1]  2  6  7  8 21 30 33 35 42 43 45 49
## [1]  6 30 45  2  7 35
## [1]  8 21 33 42 43 49
## [1] 49 42 21 33  8 43
## integer(0)

each_lambda_error <- apply(each_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error
 기록
plot(each_lambda_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```



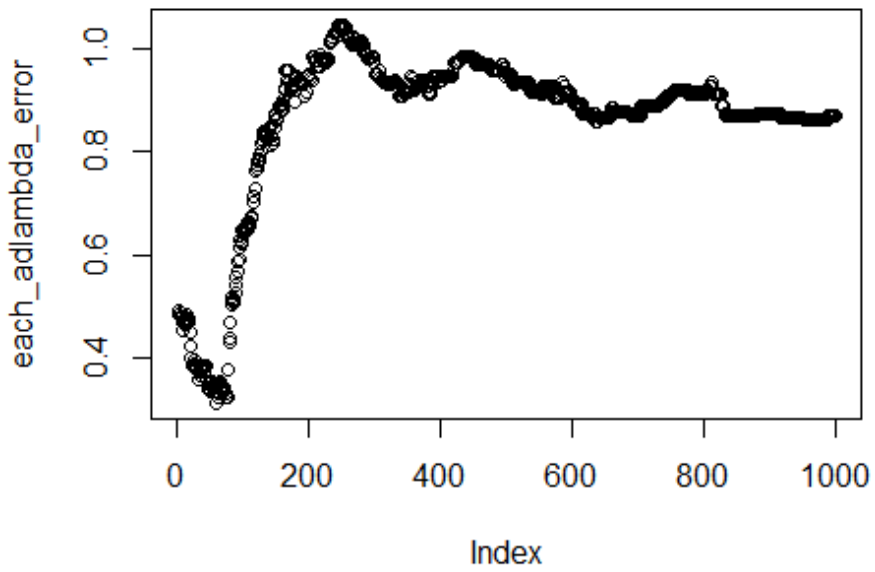

```
min_lambda_error <- min(each_lambda_error)
min_lambda_index <- which(each_lambda_error == min_lambda_error)
min_lambda_index

## [1] 56

min_lambda_lasso <- lambda[min_lambda_index]
min_lambda_lasso

## [1] 0.56

lasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_lambda_lasso, method = 'lasso')
beta_lasso_rq <- lasso_rq_fit$coefficients
beta_lasso_rq
```

```
##   (Intercept)      x[, -1]1      x[, -1]2      x[, -1]3      x[, -1]4
##  5.891263e-01 -5.122619e-01  3.669854e-01  3.503652e-01  1.604138e-01
##      x[, -1]5      x[, -1]6      x[, -1]7      x[, -1]8      x[, -1]9
##  4.953386e-01  6.238053e-01  5.315280e-01 -1.657933e-01  2.337306e-01
##     x[, -1]10     x[, -1]11     x[, -1]12     x[, -1]13     x[, -1]14
##  3.522469e-01  4.112600e-01  2.227408e-02  2.649583e-01 -1.661987e-01
##     x[, -1]15     x[, -1]16     x[, -1]17     x[, -1]18     x[, -1]19
## -3.068837e-01  3.645739e-01 -1.513321e-09 -2.773808e-02 -2.870316e-02
##     x[, -1]20     x[, -1]21     x[, -1]22     x[, -1]23     x[, -1]24
##  1.117825e-01 -1.712996e-01  4.691012e-01  2.249812e-01  2.458484e-03
##     x[, -1]25     x[, -1]26     x[, -1]27     x[, -1]28     x[, -1]29
## -9.522469e-10 -6.388581e-01 -4.175792e-01  2.242382e-01 -9.638425e-03
##     x[, -1]30
##  1.602171e-01
```

여기의 람다를 이용해서 라쏘모델을 피팅하고, 이때의 베타값들을 얻어낸다. 이는 이후 사용될 것이다. 이제 adaptive lasso 를 하자.

```
set.seed(2)  # seed 를 다르게 설정
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

weight_beta <- beta_lasso_rq %>% abs  # 베타 절대값의 역수만큼 weight 을 줘야하니 바꿔준다.
weight_beta <- weight_beta[2:length(weight_beta)]  # 베타0 의 가중치는 0 이 되어야하니 제외

each_ad_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    penalty <-c(0, lambda[j] / weight_beta)  # 베타 0 에 대한 lambda_0 를 0 으로 만들어준다.
    # 각각 람다에 대해 lasso beta 로 나눈 값을 고려하고 있다.
    each_adlasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = penalty, method = 'lasso')
```

```
    each_ad_beta <- each_adlasso_rq$coefficients
    each_ad_predicted_y <- x[ind_for, ] %*% each_ad_beta
    each_ad_error[j, i] <- quantile(abs(valid_y - each_ad_predicted_y),0.25)
  }
  #print(ind_for)
  #print(ind)  # print를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}

each_adlambda_error <- apply(each_ad_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold eror 기록
plot(each_adlambda_error)  # cv를 통한 error의 형태가 이런식으로 나타남!
```




```
min_adlambda_error <- min(each_adlambda_error)
min_adlambda_index <- which(each_adlambda_error == min_adlambda_error)
min_adlambda_index
```

```
## [1] 60
```

```
min_adlambda_lasso <- lambda[min_adlambda_index]
min_adlambda_lasso
```

```
## [1] 0.6
```

```
# optimum adaptive lasso rq fit
```

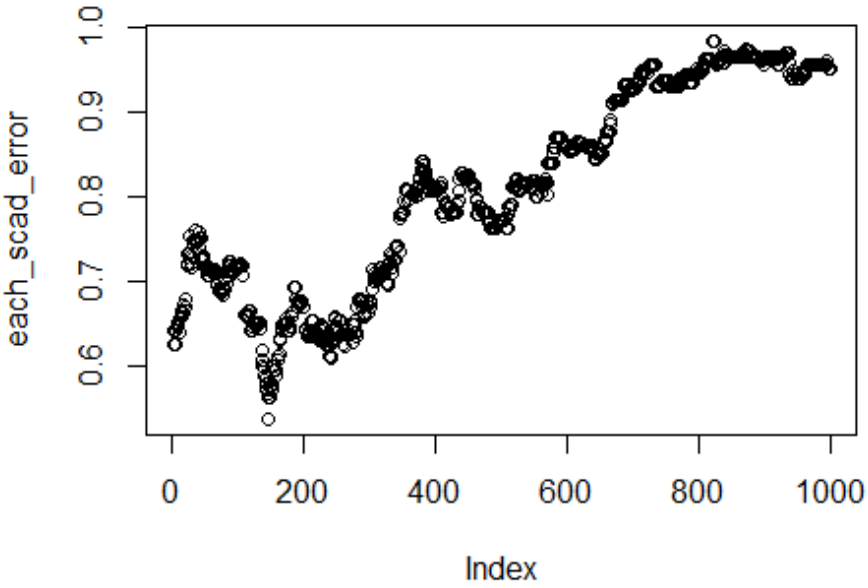
```r
opt_adlasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_adlambda_lasso, method =
 'lasso')
beta_adlasso_rq <- opt_adlasso_rq_fit$coefficients

length(which(abs(beta_adlasso_rq) > 10^(-5)))

## [1] 30

# error for Adaptive Lasso Quantile Regression
sum((beta_true - beta_adlasso_rq)^2) # 3.398

## [1] 3.398382
```

adaptive lasso 의 추정오차 제곱합은 3.398 이다.

SCAD 의 경우

```r
# SCAD tau = 0.25
set.seed(2)
ind <- 1:n
ind_all <- 1:n
lambda = seq(0.01, 10, by = 0.01)

each_scad_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에
대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    weight = c(0, scad_deriv(abs(beta_lasso_rq[-1]), lam = lambda[j]))
    # 베타 0 에 대한 lambda_0 를 0 으로 만들어준다.
    # 각각 람다에 대해 스캐드 미분함수를 고려한다.
    each_scad_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = weight, method =
'lasso')
    each_scad_beta <- each_scad_rq$coefficients
    each_scad_predicted_y <- x[ind_for, ] %*% each_scad_beta
    each_scad_error[j, i] <- quantile(abs(valid_y - each_scad_predicted_y), 0.25)
  }
  print(ind_for)
  print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}
```

```
## [1] 21 15  6 58 32  8
##  [1]  1  2  3  4  5  7  9 10 11 12 13 14 16 17 18 19 20 22 23 24 25 26 27 28 29
## [26] 30 31 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
## [51] 56 57 59 60
## [1] 20 34 60 14 46 13
##  [1]  1  2  3  4  5  7  9 10 11 12 16 17 18 19 22 23 24 25 26 27 28 29 30 31 33
## [26] 35 36 37 38 39 40 41 42 43 44 45 47 48 49 50 51 52 53 54 55 56 57 59
## [1]  1  3 23 41 10 49
##  [1]  2  4  5  7  9 11 12 16 17 18 19 22 24 25 26 27 28 29 30 31 33 35 36 37 38
## [26] 39 40 42 43 44 45 47 48 50 51 52 53 54 55 56 57 59
## [1] 48 57 52 54 27 55
##  [1]  2  4  5  7  9 11 12 16 17 18 19 22 24 25 26 28 29 30 31 33 35 36 37 38 39
## [26] 40 42 43 44 45 47 50 51 53 56 59
## [1] 11 31 17 16 24 44
##  [1]  2  4  5  7  9 12 18 19 22 25 26 28 29 30 33 35 36 37 38 39 40 42 43 45 47
## [26] 50 51 53 56 59
## [1] 12  4 43 37  5 39
##  [1]  2  7  9 18 19 22 25 26 28 29 30 33 35 36 38 40 42 45 47 50 51 53 56 59
## [1]  9  2 40 28 35 38
##  [1]  7 18 19 22 25 26 29 30 33 36 42 45 47 50 51 53 56 59
## [1]  7 30 22 47 50 25
##  [1] 18 19 26 29 33 36 42 45 51 53 56 59
## [1] 59 19 33 36 53 42
## [1] 18 26 29 45 51 56
## [1] 26 45 51 18 56 29
## integer(0)

each_scad_error <- apply(each_scad_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold er
ror 기록
plot(each_scad_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```




```
min_scad_error <- min(each_scad_error)
min_scad_index <- which(each_scad_error == min_scad_error)
min_scad_index

## [1] 147

min_scad_lasso <- lambda[min_scad_index]
min_scad_lasso

## [1] 1.47

# optimum weighted lasso rq fit with scad derivative funtion

opt_weight = c(0, scad_deriv(abs(beta_lasso_rq[-1]), lam = min_scad_lasso))
opt_scad_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = opt_weight, method = 'lasso')

beta_scad_rq <- opt_scad_rq_fit$coefficients
length(which(abs(beta_scad_rq) > 10^(-5)))

## [1] 30

# RSS for weighted lasso rq fit with scad derivative funtion
sum((beta_true - beta_scad_rq)^2) # 2.615

## [1] 2.615134
```

scad 의 추정오차제곱합은 2.615 이다.

세 경우를 비교하자.

```
sum((beta_true - beta_optlasso_rq)^2) # 2.615134
```

```
## [1] 2.615134
```

```
sum((beta_true - beta_adlasso_rq)^2) # 3.421521
```

```
## [1] 3.398382
```

```
sum((beta_true - beta_scad_rq)^2) # 2.615134
```

```
## [1] 2.615134
```

추정오차 제곱합은 다음과 같다. 그런데 라쏘와 스캐드의 값이 같은 것처럼 보이는데, 이후에도 동일한 현상이 발생한다. 이는 우리 코드의 문제가 아니다. 처음 주어진 베타값은 0.5 로 상당히 작은 편이다. 그러나 scad 에서는 1.47 가 최적의 람다로 선택된다. 결국 베타값이 머무는 범위보다 람다값이 훨씬 크기때문에, 이경우 라쏘와 스캐드는 동일한 결과를 산출한다. 이는 스캐드 함수를 보면 이해할 수 있다. 그리고 이런 경향은 tau = 0.75 일때도 동일하게 발생한다.

tau = 0.75

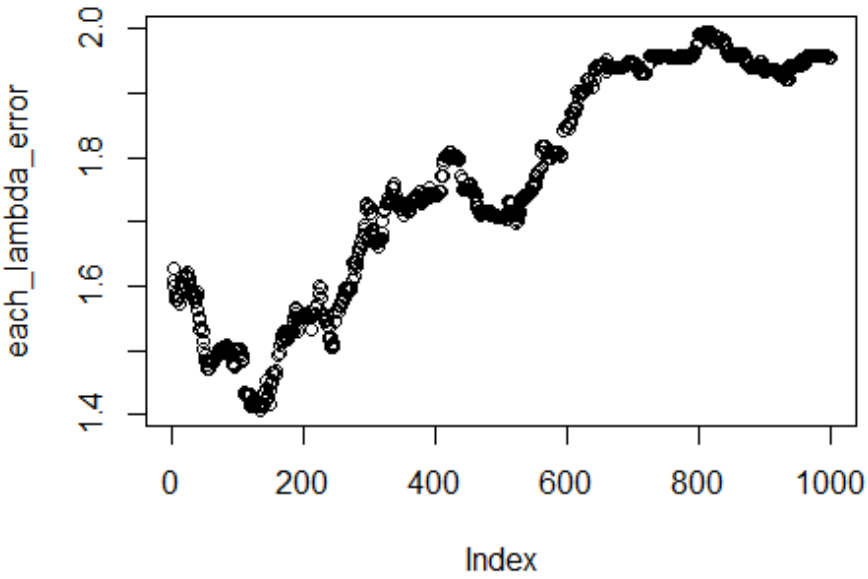
tau = 0.75 에 대한 데이터를 다시 고려하겠다.

```
set.seed(2019)
n = 60; p = 30
x = matrix(rnorm(n * p), ncol = p)
x = cbind(rep(1,n), x)

tau = 0.25
sd = 1
beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
obs_err = rnorm(n, sd = sd)
x[,2] = runif(n)
y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
beta_true = beta
beta_true[2] = beta[2] + tau * qnorm(tau)
beta_true[1] = beta[1] + qnorm(tau, sd = sd)
```

lasso 에 대해서

```
# Lasso 에 대해서
set.seed(2)
k = 10
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

each_error <- matrix(0, nrow = length(lambda), ncol = k)  # rss 를 각각 람다에 대해 저
```

```
장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], metho
d = 'lasso')
    each_beta <- each_lasso_rq$coefficients
    each_predicted_y <- x[ind_for, ] %*% each_beta
    each_error[j, i] <-  quantile(abs(valid_y - each_predicted_y), 0.75)
  }
  #print(ind_for)
  #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없
다.
}

each_lambda_error <- apply(each_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error
 기록
plot(each_lambda_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```

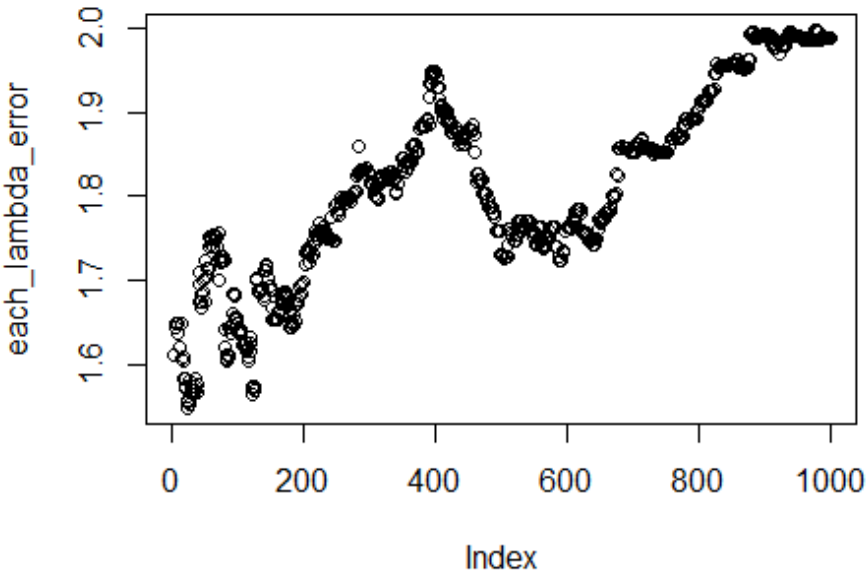



```
min_lambda_error <- min(each_lambda_error)
min_lambda_index <- which(each_lambda_error == min_lambda_error)
min_lambda_index

## [1] 134

min_lambda_lasso <- lambda[min_lambda_index]  #

# optimum lasso rq fit

opt_lasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_lambda_lasso, method = 'la
sso')
beta_optlasso_rq <- opt_lasso_rq_fit$coefficients
length(which(abs(beta_optlasso_rq) > 10^(-5)))

## [1] 30

sum((beta_true - beta_optlasso_rq)^2) # 4.68

## [1] 2.622981
```

lasso 의 추정오차제곱합은 4.68 이다.

```
## for adaptive lasso qr and scad qr
set.seed(1)
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n
```

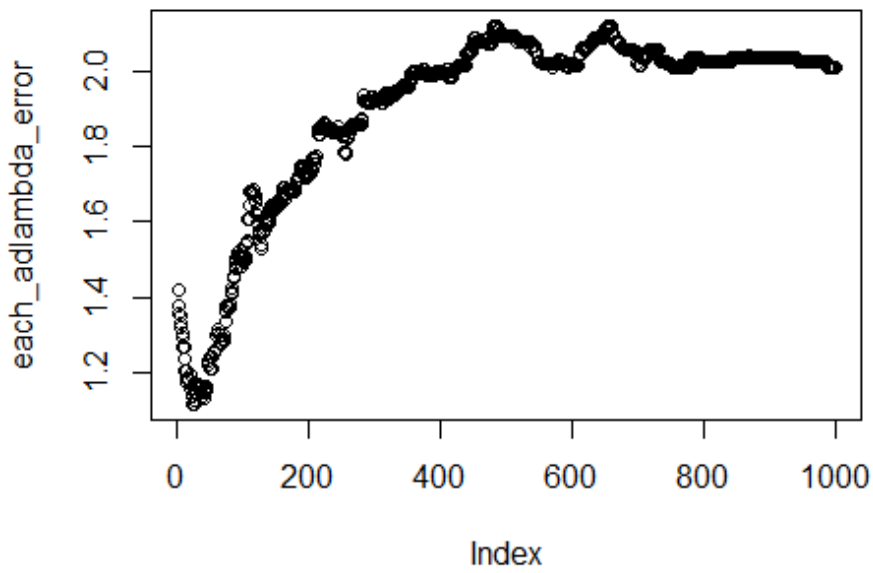
```r
each_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], metho
d = 'lasso')
    each_beta <- each_lasso_rq$coefficients
    each_predicted_y <- x[ind_for, ] %*% each_beta
    each_error[j, i] <- quantile(abs(valid_y - each_predicted_y), 0.75)
  }
  #print(ind_for)
  #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}

each_lambda_error <- apply(each_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_lambda_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```




```
min_lambda_error <- min(each_lambda_error)
min_lambda_index <- which(each_lambda_error == min_lambda_error)
min_lambda_index

## [1] 24

min_lambda_lasso <- lambda[min_lambda_index]
min_lambda_lasso

## [1] 0.24

lasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_lambda_lasso, method = 'lasso')
beta_lasso_rq <- lasso_rq_fit$coefficients
beta_lasso_rq

##  (Intercept)     x[, -1]1     x[, -1]2     x[, -1]3     x[, -1]4     x[, -1]5
##  1.066789997 -1.227614454  0.516543585  0.441934492  0.260446285  0.479899609
##     x[, -1]6     x[, -1]7     x[, -1]8     x[, -1]9    x[, -1]10    x[, -1]11
##  0.579591765  0.576787477 -0.176984821  0.227699300  0.378009572  0.436098144
##    x[, -1]12    x[, -1]13    x[, -1]14    x[, -1]15    x[, -1]16    x[, -1]17
##  0.004509873  0.402207608 -0.224553281 -0.380369236  0.356322825 -0.041439334
##    x[, -1]18    x[, -1]19    x[, -1]20    x[, -1]21    x[, -1]22    x[, -1]23
##  0.105078373 -0.088756063  0.222966331 -0.372646997  0.515050737  0.286227914
##    x[, -1]24    x[, -1]25    x[, -1]26    x[, -1]27    x[, -1]28    x[, -1]29
##  0.100343807  0.065648226 -0.560880275 -0.409789009  0.112886457 -0.055977870
##    x[, -1]30
##  0.021723767
```

adaptive lasso qr with tau = 0.75

```r
set.seed(2)  # seed 를 다르게 설정
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1:n

weight_beta <- beta_lasso_rq %>% abs  # 베타 절대값의 역수만큼 weight 을 줘야하니 바꿔준다.
weight_beta <- weight_beta[2:length(weight_beta)]  # 베타0 의 가중치는 0 이 되어야하니 제외

each_ad_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    penalty <-c(0, lambda[j] / weight_beta)  # 베타 0 에 대한 lambda_0 를 0 으로 만들어준다.
    # 각각 람다에 대해 lasso beta 로 나눈 값을 고려하고 있다.
    each_adlasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = penalty, method = 'lasso')
    each_ad_beta <- each_adlasso_rq$coefficients
    each_ad_predicted_y <- x[ind_for, ] %*% each_ad_beta
    each_ad_error[j, i] <- quantile(abs(valid_y - each_ad_predicted_y), 0.75)
  }
  #print(ind_for)
  #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}

each_adlambda_error <- apply(each_ad_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_adlambda_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```

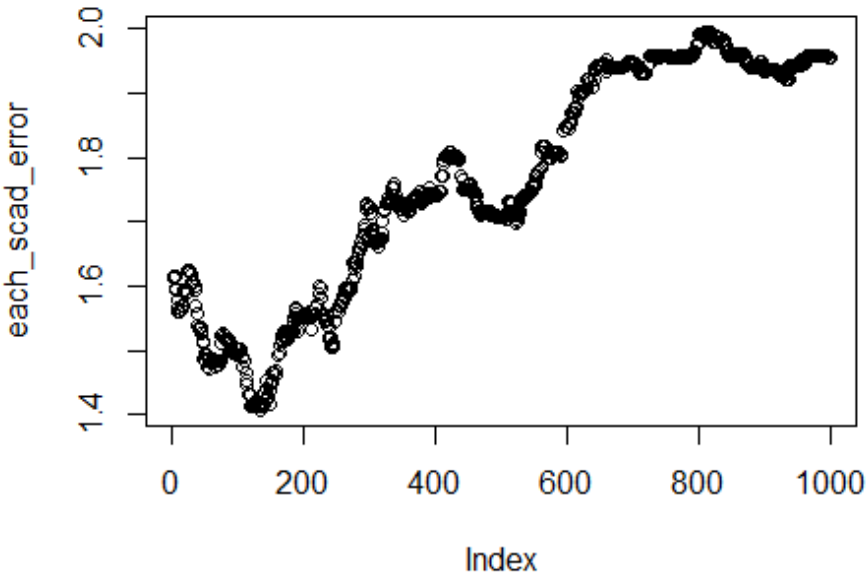



```
min_adlambda_error <- min(each_adlambda_error)
min_adlambda_index <- which(each_adlambda_error == min_adlambda_error)
min_adlambda_index

## [1] 25

min_adlambda_lasso <- lambda[min_adlambda_index]
min_adlambda_lasso

## [1] 0.25

# optimum adaptive lasso rq fit

opt_adlasso_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = min_adlambda_lasso, method =
 'lasso')
beta_adlasso_rq <- opt_adlasso_rq_fit$coefficients

length(which(abs(beta_adlasso_rq) > 10^(-5)))

## [1] 31

# error for Adaptive Lasso Quantile Regression
sum((beta_true - beta_adlasso_rq)^2) # 7.48

## [1] 6.341452
```

adaptive lasso 의 추정오차제곱합은 7.48 이다.

SCAD

```r
# SCAD tau = 0.75
set.seed(2)
ind <- 1:n
ind_all <- 1:n
lambda = seq(0.01, 10, by = 0.01)

each_scad_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (i in 1:k) {
  ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
  ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
  ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
  ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
  train_x <- x[ind_each, ]
  train_y <- y[ind_each]
  valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
  valid_y <- y[ind_for]
  for (j in seq_along(lambda)) {
    weight = c(0, scad_deriv(abs(beta_lasso_rq[-1]), lam = lambda[j]))
    # 베타 0 에 대한 lambda_0 를 0 으로 만들어준다.
    # 각각 람다에 대해 스캐드 미분함수를 고려한다.
    each_scad_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = weight, method = 'lasso')
    each_scad_beta <- each_scad_rq$coefficients
    each_scad_predicted_y <- x[ind_for, ] %*% each_scad_beta
    each_scad_error[j, i] <- quantile(abs(valid_y - each_scad_predicted_y), 0.75)
  }
  #print(ind_for)
  #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
}


each_scad_error <- apply(each_scad_error, 1, mean) # 각각 람다 시퀀스에 대한 10fold error 기록
plot(each_scad_error)  # cv 를 통한 error 의 형태가 이런식으로 나타남!
```




```
min_scad_error <- min(each_scad_error)
min_scad_index <- which(each_scad_error == min_scad_error)
min_scad_index

## [1] 134

min_scad_lasso <- lambda[min_scad_index]
min_scad_lasso

## [1] 1.34

# optimum weighted lasso rq fit with scad derivative funtion

opt_weight = c(0, scad_deriv(abs(beta_lasso_rq[-1]), lam = min_scad_lasso))
opt_scad_rq_fit <- rq(y ~ x[,-1], tau = tau, lambda = opt_weight, method = 'lasso')

beta_scad_rq <- opt_scad_rq_fit$coefficients
length(which(abs(beta_scad_rq) > 10^(-5)))

## [1] 30

# error for weighted lasso rq fit with scad derivative funtion
sum((beta_true - beta_scad_rq)^2) # 4.686

## [1] 2.622981
```

scad 의 추정오차 제곱합은 4.68 이다.

tau = 0.75 일때의 결과를 요약하자면,

```
sum((beta_true - beta_optlasso_rq)^2) # 4.68

## [1] 2.622981

sum((beta_true - beta_adlasso_rq)^2) # 7.48

## [1] 6.341452

sum((beta_true - beta_scad_rq)^2) # 4.68

## [1] 2.622981
```

전체적으로 tau = 0.75 일때 추정오차 제곱합이 상대적으로 크다. 그중에서 adaptive lasso 의 추정오차가 가장 크다.그리고 여기에서도 스캐드와 라쏘의 결과는 동일하게 나타난다.

tau = 0.5 일때와 이외에 0.75, 0.25 일때의 결과를 비교하면, 베타 추정의 측면에서 0.5 일때의 에러가 더 적다. 결국 추정을 하는데에 있어서 제일 좋은 tau 는 0.5 라고 말할 수 있다.

Q6. Suppose that design matrix X is generated from AR(1)-covariance structure with a correlation parameter sigma between 0 and 1. By varying a simga value (use seq(0.1,0.9,0.1)), compare the performances of Lasso estimators in terms of estimation error. Consider the quantile level tau = 0.25, 0.5, 0.75, respectively.

```
sigma <- seq(0.1, 0.9, 0.1)
corr_list <- list(NULL)
corr_mat <- matrix(0, nrow = p, ncol = p)
for (i in seq_along(sigma)) {
  rho <- sigma[i]
  for (j in 1:p) {
    for (k in 1:p) {
      difference <- abs(j - k)
      if (difference == 0) {
        corr_mat[j,k] <- 1
      } else {
        corr_mat[j,k] <- rho^difference
      }
    }
  }
  corr_list[[i]] <- corr_mat %>% round(6)
}
```

AR1 covariance structure 에 기반한 상관계수행렬을 만들었다. 주대각성분은 1 이고, 변수간 거리가 늘어날 수록 상관관계가 약해지도록 만들었다.

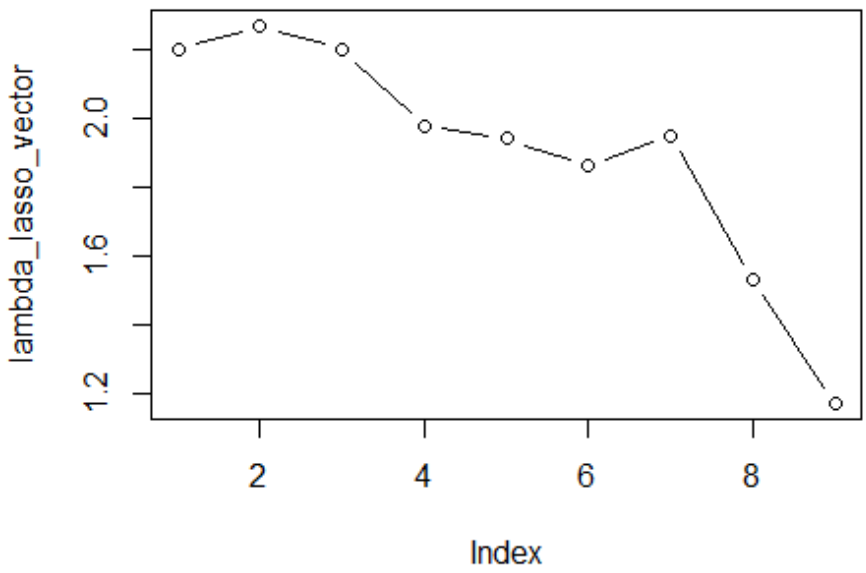
먼저 tau = 0.5 에 대해 시행하겠다.

```
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1
```

```r
k = 10
x_list <- list(NULL)
y_list <- list(NULL)
each_error_list <- list(NULL)
each_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (l in seq_along(sigma)) {
  set.seed(2019)  # 같은 시드에 대해서 상관관계 행렬만 다르게 해서 샘플링한다.
  ind <- 1:n
  ind_all <- 1:n
  n = 60; p = 30
  x <- rmvnorm(n = n, mean = rep(0, p), sigma = corr_list[[l]])
  x = cbind(rep(1,n), x)
  tau = 0.5
  sd = 1
  beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
  obs_err = rnorm(n, sd = sd)
  x[,2] = runif(n)
  y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
  beta_true = beta
  beta_true[2] = beta[2] + tau * qnorm(tau)
  beta_true[1] = beta[1] + qnorm(tau, sd = sd)
  x_list[[l]] <- x # 상관관계 행렬마다 바뀌는 우리의 데이터를 리스트로 저장한다.
  y_list[[l]] <- y
  for (i in 1:k) {
    ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
    ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
    ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
    ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
    train_x <- x[ind_each, ]
    train_y <- y[ind_each]
    valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
    valid_y <- y[ind_for]
    for (j in seq_along(lambda)) {
      each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], met
hod = 'lasso')
      each_beta <- each_lasso_rq$coefficients
      each_predicted_y <- x[ind_for, ] %*% each_beta
      each_error[j, i] <- quantile(abs(valid_y - each_predicted_y), 0.5)
    }
    #print(ind_for)
    #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
  }
  each_error_list[[l]] <- each_error
  #print(l)# l 번 잘 작동하는지 확인하기 위한 작업
}
```

시그마 별로 리스트를 만들었다. 그 리스트에 이전과 동일한 방식으로 람다와 fold 별로 기록된 error matrix 를 저장한다. 이때 시그마 별로 데이터가 다르게 생성되도록 만들어준다.

```
each_lambda_error_list <- list(NULL)
min_lambda_error_list <- list(NULL)
min_lambda_index_list <- list(NULL)
min_lambda_lasso_list <- list(NULL)
for (i in seq_along(sigma)) {
  each_lambda_error_list[[i]] <- apply(each_error_list[[i]], 1, mean)
  min_lambda_error_list[[i]] <- min(each_lambda_error_list[[i]])
  min_lambda_index_list[[i]] <- which(each_lambda_error_list[[i]] == min_lambda_error
_list[[i]])
  min_lambda_lasso_list[[i]] <- lambda[min_lambda_index_list[[i]]]
}

lambda_lasso_vector <- min_lambda_lasso_list %>% unlist
lambda_lasso_vector

## [1] 2.20 2.27 2.20 1.98 1.94 1.86 1.95 1.53 1.17

# 2.20 2.27 2.20 1.98 1.94 1.86 1.95 1.53 1.17
plot(lambda_lasso_vector, type = 'b')
```



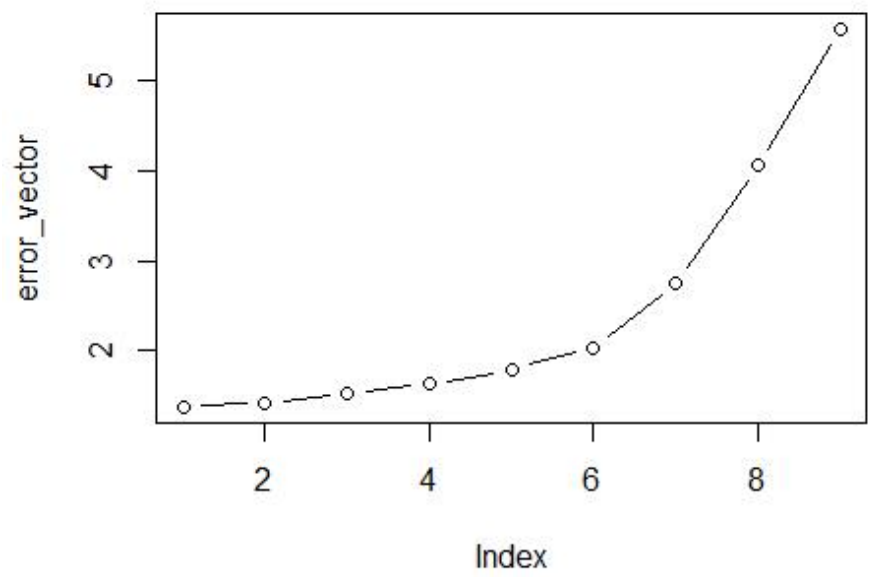
상관관계가 매우 클경우, 람다가 작아지는 경향이 나타난다.

```
# optimum lasso rq fit
sigma_lasso_fit <- NULL
```

```r
error_vector <- rep(0, length(lambda_lasso_vector))
for (i in seq_along(lambda_lasso_vector)) {
  x_data <- x_list[[i]]
  y_data <- y_list[[i]]
  sigma_lasso_fit <- rq(y_data ~ x_data[, -1], tau = tau, lambda_lasso_vector[i], met
hod = 'lasso')
  beta_sigma_lasso_rq <- sigma_lasso_fit$coefficients
  error_vector[i] <- sum((beta_true - beta_sigma_lasso_rq)^2)
  #print(i)
}
names(error_vector) <- c('0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9
')
error_vector
```

```
##       0.1       0.2       0.3       0.4       0.5       0.6       0.7       0.8
## 1.377606 1.419574 1.529862 1.631743 1.791691 2.039142 2.756524 4.064099
##       0.9
## 5.572387
```

```r
#   0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8      0.9
# 1.377606 1.419574 1.529862 1.631743 1.791691 2.039142 2.756524 4.064099 5.572387

plot(error_vector, type = 'b')
```

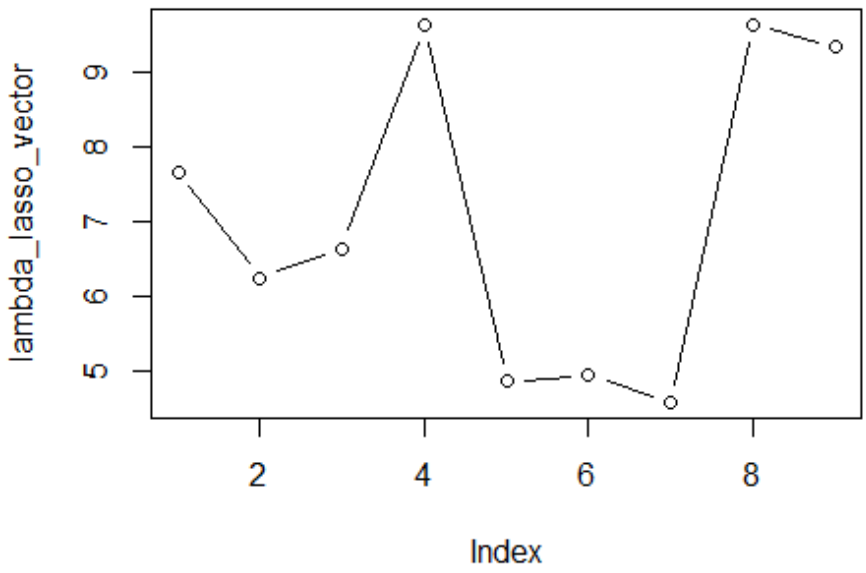


변수간 상관관계가 클수록 추정오차가 점점 증가하는 것을 보여준다!

다음은 tau = 0.25 일 때이다.

```r
## tau = 0.25
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1
k = 10
x_list <- list(NULL)
y_list <- list(NULL)
each_error_list <- list(NULL)
each_error <- matrix(0, nrow = length(lambda), ncol = k)  # error 를 각각 람다에 대해 저장하는 매트릭스
for (l in seq_along(sigma)) {
  set.seed(2019)  # 같은 시드에 대해서 상관관계 행렬만 다르게 해서 샘플링한다.
  ind <- 1:n
  ind_all <- 1:n
  n = 60; p = 30
  x <- rmvnorm(n = n, mean = rep(0, p), sigma = corr_list[[l]])
  x = cbind(rep(1,n), x)
  tau = 0.25  #### tau = 0.25
  sd = 1
  beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
  obs_err = rnorm(n, sd = sd)
  x[,2] = runif(n)
  y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
  beta_true = beta
  beta_true[2] = beta[2] + tau * qnorm(tau)
  beta_true[1] = beta[1] + qnorm(tau, sd = sd)
  x_list[[l]] <- x # 상관관계 행렬마다 바뀌는 우리의 데이터를 리스트로 저장한다.
  y_list[[l]] <- y
  for (i in 1:k) {
    ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
    ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
    ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
    ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
    train_x <- x[ind_each, ]
    train_y <- y[ind_each]
    valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
    valid_y <- y[ind_for]
    for (j in seq_along(lambda)) {
      each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], met
hod = 'lasso')
      each_beta <- each_lasso_rq$coefficients
      each_predicted_y <- x[ind_for, ] %*% each_beta
      each_error[j, i] <- quantile(abs(valid_y - each_predicted_y), 0.25)
    }
    #print(ind_for)
    #print(ind)  # print 를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가 없다.
  }
```

```
  each_error_list[[l]] <- each_error
  #print(l) # l 번 잘 작동하는지 확인하기 위한 작업
}

each_lambda_error_list <- list(NULL)
min_lambda_error_list <- list(NULL)
min_lambda_index_list <- list(NULL)
min_lambda_lasso_list <- list(NULL)
for (i in seq_along(sigma)) {
  each_lambda_error_list[[i]] <- apply(each_error_list[[i]], 1, mean)
  min_lambda_error_list[[i]] <- min(each_lambda_error_list[[i]])
  min_lambda_index_list[[i]] <- which(each_lambda_error_list[[i]] == min_lambda_error
_list[[i]])
  min_lambda_lasso_list[[i]] <- lambda[min_lambda_index_list[[i]]]
}

lambda_lasso_vector <- min_lambda_lasso_list %>% unlist
lambda_lasso_vector

## [1] 7.67 6.24 6.64 9.64 4.85 4.94 4.57 9.64 9.34

#7.67 6.24 6.64 9.64 4.85 4.94 4.57 9.64 9.34
plot(lambda_lasso_vector, type = 'b')
```

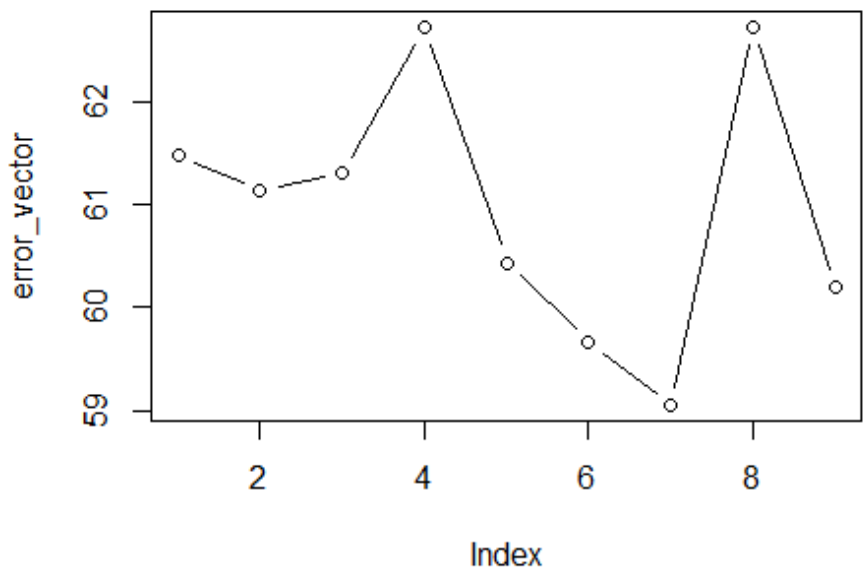


상관관계가 클수록 람다가 커지는 경향이 tau = 0.25 에서도 나타난다. 물론 중간에 갑자기 튀어오른 값이 있지만, 이상현상이라고 생각할 수 있다.

```r
# optimum lasso rq fit
sigma_lasso_fit <- NULL
error_vector <- rep(0, length(lambda_lasso_vector))
for (i in seq_along(lambda_lasso_vector)) {
  x_data <- x_list[[i]]
  y_data <- y_list[[i]]
  sigma_lasso_fit <- rq(y_data ~ x_data[, -1], tau = tau, lambda_lasso_vector[i], met
hod = 'lasso')
  beta_sigma_lasso_rq <- sigma_lasso_fit$coefficients
  error_vector[i] <- sum((y_data - x_data %*% beta_sigma_lasso_rq)^2)
  #print(i)
}
# Error in eval(predvars, data, env) : not that many frames on the stack (에러발생)
names(error_vector) <- c('0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9
')
error_vector
```

```
##      0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8
## 61.46712 61.13422 61.30572 62.70868 60.43609 59.66641 59.05596 62.71706
##      0.9
## 60.19180
```

```r
#   0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8      0.9
#1.361156 1.391118 1.467933 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
# 모델이 작동하지 않는 경우, 그냥 0 으로 기록했다.

plot(error_vector, type = 'b')
```

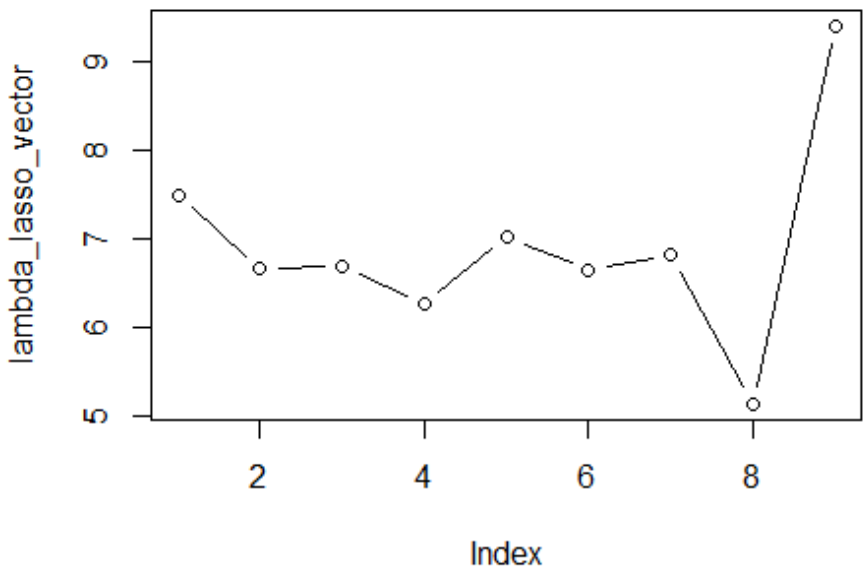


시그마가 커지면 모델이 계산이 안되서, 정확히 파악하기는 어렵지만 비슷한 경향이 나타날 것이다.

다음은 tau = 0.75 일때의 경우다.

```
### tau = 0.75

lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1
k = 10
x_list <- list(NULL)
y_list <- list(NULL)
each_error_list <- list(NULL)
each_error <- matrix(0, nrow = length(lambda), ncol = k)  # rss 를 각각 람다에 대해 저장하는 매트릭스
for (l in seq_along(sigma)) {
  set.seed(2019)  # 같은 시드에 대해서 상관관계 행렬만 다르게 해서 샘플링한다.
  ind <- 1:n
  ind_all <- 1:n
  n = 60; p = 30
  x <- rmvnorm(n = n, mean = rep(0, p), sigma = corr_list[[l]])
  x = cbind(rep(1,n), x)
  tau = 0.75  #### tau = 0.75
  sd = 1
  beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
  obs_err = rnorm(n, sd = sd)
  x[,2] = runif(n)
  y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
  beta_true = beta
  beta_true[2] = beta[2] + tau * qnorm(tau)
  beta_true[1] = beta[1] + qnorm(tau, sd = sd)
  x_list[[l]] <- x # 상관관계 행렬마다 바뀌는 우리의 데이터를 리스트로 저장한다.
  y_list[[l]] <- y
  for (i in 1:k) {
    ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
    ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
    ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
    ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
    train_x <- x[ind_each, ]
    train_y <- y[ind_each]
    valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
    valid_y <- y[ind_for]
    for (j in seq_along(lambda)) {
      each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], met
hod = 'lasso')
      each_beta <- each_lasso_rq$coefficients
      each_predicted_y <- x[ind_for, ] %*% each_beta
```

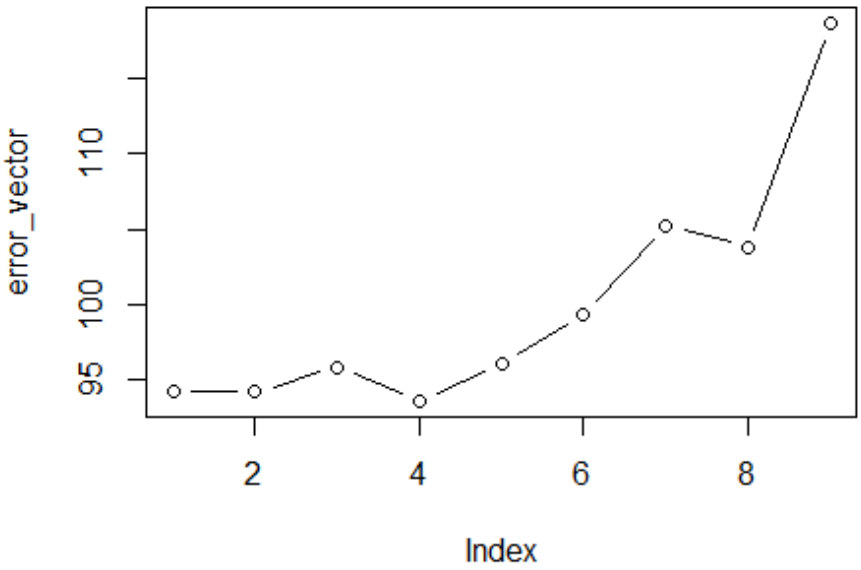
```
      each_error[j, i] <- quantile(abs(valid_y - each_predicted_y),0.75)
    }
    #print(ind_for)
    #print(ind)  # print를 통해 직접 짠 cv_fold 함수가 잘 작동하는지를 확인한다. 문제가
없다.
  }
  each_error_list[[l]] <- each_error
  #print(l) # l 번 잘 작동하는지 확인하기 위한 작업
}

each_lambda_error_list <- list(NULL)
min_lambda_error_list <- list(NULL)
min_lambda_index_list <- list(NULL)
min_lambda_lasso_list <- list(NULL)
for (i in seq_along(sigma)) {
  each_lambda_error_list[[i]] <- apply(each_error_list[[i]], 1, mean)
  min_lambda_error_list[[i]] <- min(each_lambda_error_list[[i]])
  min_lambda_index_list[[i]] <- which(each_lambda_error_list[[i]] == min_lambda_error
_list[[i]])
  min_lambda_lasso_list[[i]] <- lambda[min_lambda_index_list[[i]]]
}

lambda_lasso_vector <- min_lambda_lasso_list %>% unlist
lambda_lasso_vector

## [1] 7.50 6.67 6.69 6.27 7.03 6.65 6.82 5.14 9.41

#7.50 6.67 6.69 6.27 7.03 6.65 6.82 5.14 9.41
plot(lambda_lasso_vector, type = 'b')
```




```r
# optimum lasso rq fit
sigma_lasso_fit <- NULL
error_vector <- rep(0, length(lambda_lasso_vector))
for (i in seq_along(lambda_lasso_vector)) {
  x_data <- x_list[[i]]
  y_data <- y_list[[i]]
  sigma_lasso_fit <- rq(y_data ~ x_data[, -1], tau = tau, lambda_lasso_vector[i], met
hod = 'lasso')
  beta_sigma_lasso_rq <- sigma_lasso_fit$coefficients
  error_vector[i] <- sum((y_data - x_data %*% beta_sigma_lasso_rq)^2)
  #print(i)
}
# Error in eval(predvars, data, env) : not that many frames on thestack
names(error_vector) <- c('0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9
')
error_vector
```

```
##       0.1       0.2       0.3       0.4       0.5       0.6       0.7       0.8
##  94.27665  94.16850  95.83518  93.57593  96.04698  99.29643 105.26271 103.80955
##       0.9
## 118.70251
```

```r
#   0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8      0.9
# 2.910968 3.043536 3.076815 2.912499 3.123237 3.488799 3.850182 4.828573 0.000000
# 모델이 작동하지 않는 경우, 그냥 0 으로 기록했다.

plot(error_vector, type = 'b')
```



이번에도 비슷한 경향이 발생했다. 상관관계가 0.9 일때 모델이 작동하지 않는다. 뿐만아니라 전체적으로 추정오차가 증가하는 경향을 보인다.

6 번의 결론

변수간 상관관계가 클수록 lasso quantile regression 은 잘 작동하지 않고, 이는 tau 가 0.5 가 아닐때 더 큰 문제로 나타난다. 또한 전체적으로 상관관계가 클수록 추정오차가 증가하는 경향을 보인다.

Q7. Consider the model used in Q5 but errors follow from normal mixture distribution. Specifically, errors follow 0.9 N(0,1) + 0.1 N(0, variance), where variance = 2,4,9,16, respectively. For mixture distribution refer to https://www.rdocumentation.org/packages/EnvStats/versions/2.3.1/topics/NormalMix Consider tau = 0.5 and obtain estimation errors for the quantile regression estimator using Lasso, Adaptive Lasso, and weighted Lasso (using SCAD derivative), respectively. Since this quantile regression model is homoscedasticity, underlying "beta_true" corresponds to underlying coefficients for the regular linear regression. Apply linear regression using Lasso, adaptive Lasso, and weighte Lasso (using SCAD derivative), respectively, and obtain estimation errrors. Between quantile regression and linear regression estimators, which one is better, i.e. yields a smaller estimation error?

```
stdv <- c(sqrt(2), 2, 3, 4)
```

변하는 에러를 잡아내기 위해 설정했다.

lasso 에 대하여

```
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1
k = 10
x_list <- list(NULL)
y_list <- list(NULL)
each_rq_error_list <- list(NULL)
each_linear_error_list <- list(NULL)
true <- list(NULL)
each_error_rq <- matrix(0, nrow = length(lambda), ncol = k)  # rss 를 각각 람다에 대해 저장하는 매트릭스
each_error_linear <- matrix(0, nrow = length(lambda), ncol = k)
for (l in seq_along(stdv)) {
  ind <- 1:n
  ind_all <- 1:n
  set.seed(2019) # seed 지정
  n = 60; p = 30
  x = matrix(rnorm(n * p), ncol = p)
  x = cbind(rep(1,n), x)
  tau = 0.5
  beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
  obs_err = 9/10 * rnorm(n, sd = 1) + 1/10 * rnorm(n, mean = 0, sd = stdv[l])
  x[,2] = runif(n)
  y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
  beta_true = beta
  beta_true[2] = beta[2] + tau * qnorm(tau)
  beta_true[1] = beta[1] + qnorm(tau, sd = sd)
  true[[l]] <- beta_true
  x_list[[l]] <- x #
  y_list[[l]] <- y # 에러가 바뀔때마다 내용을 저장
  for (i in 1:k) {
    ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
    ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
    ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
    ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
    train_x <- x[ind_each, ]
    train_y <- y[ind_each]
    valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
    valid_y <- y[ind_for]
    for (j in seq_along(lambda)) {
      each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = lambda[j], met
hod = 'lasso')
      each_beta_rq <- each_lasso_rq$coefficients
      each_predicted_y <- x[ind_for, ] %*% each_beta_rq
      each_error_rq[j, i] <-  quantile(abs(valid_y - each_predicted_y),0.5)
```

```
      each_lasso_linear <- glmnet(train_x[, -1], train_y, lambda = lambda[j], alpha =
 1)
      each_beta_linear <- rbind(each_lasso_linear$a0, each_lasso_linear$beta)
      each_linear_predicted_y <- x[ind_for, ] %*% each_beta_linear
      each_error_linear[j,i] <- sum((valid_y - each_linear_predicted_y)^2)
    }
  }
  each_rq_error_list[[l]] <- each_error_rq
  each_linear_error_list[[l]] <- each_error_linear
  #print(l)
}
```

3 중 for 문 안에서 lasso quantile regression 과 lasso linear regression 을 동시에 진행했다. 믹스쳐 모델을 따르는 데이터의 형태도 계속 바꿔주었다.

```
each_lam_rq_error_list <- list(NULL)
each_lam_linear_error_list <- list(NULL)
min_rq_error_list <- list(NULL)
min_linear_error_list <- list(NULL)
min_rq_index_list <- list(NULL)
min_linear_index_list <- list(NULL)
min_rq_lasso_list <- list(NULL)
min_linear_lasso_list <- list(NULL)
for (i in seq_along(stdv)) {
  each_lam_rq_error_list[[i]] <- apply(each_rq_error_list[[i]], 1, mean)
  each_lam_linear_error_list[[i]] <- apply(each_linear_error_list[[i]], 1, mean)
  min_rq_error_list[[i]] <- min(each_lam_rq_error_list[[i]])
  min_linear_error_list[[i]] <- min(each_lam_linear_error_list[[i]])
  min_rq_index_list[[i]] <- which(each_lam_rq_error_list[[i]] == min_rq_error_list
[[i]])
  min_linear_index_list[[i]] <- which(each_lam_linear_error_list[[i]] == min_linear_e
rror_list[[i]])
  min_rq_lasso_list[[i]] <- lambda[min_rq_index_list[[i]]]
  min_linear_lasso_list[[i]] <- lambda[min_linear_index_list[[i]]]
}

lambda_rq_vector <- min_rq_lasso_list %>% unlist
lambda_linear_vector <- min_linear_lasso_list %>% unlist
lambda_rq_vector # 4.23 3.11 3.06 2.97

## [1] 4.23 3.11 3.06 2.97

lambda_linear_vector # 0.10 0.10 0.11 0.11

## [1] 0.10 0.10 0.11 0.11
```

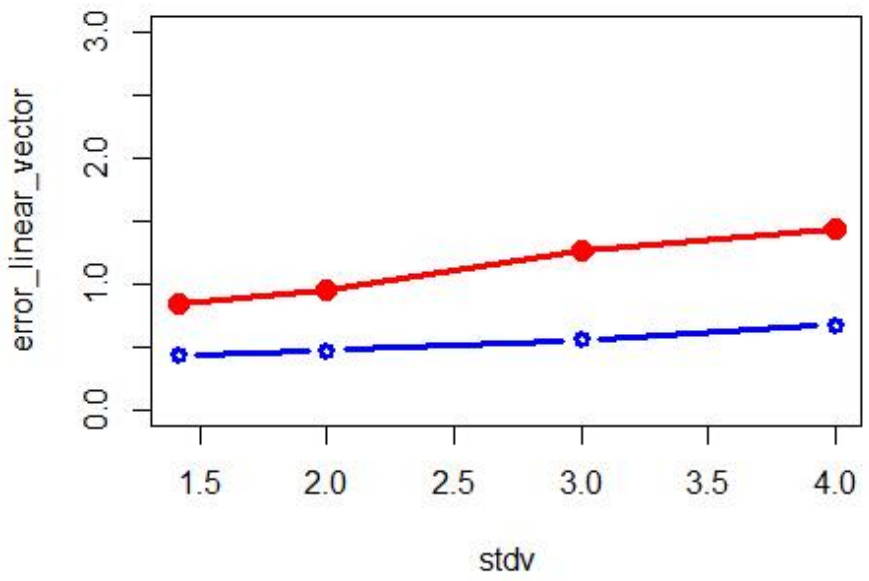
cv 를 통해 얻어진 값들에 대해서, 최소의 error 를 만들어내는 람다를 선택하는 과정이다. 그리고 선택된 람다들이다.

```
variance_rq_laaso_fit <- NULL
variance_linear_lasso_fit <- NULL
```

```
beta_variance_lasso_rq <- list(NULL)
beta_variance_linear <- list(NULL)
error_rq_vector <- rep(0, length(lambda_rq_vector))
error_linear_vector <- rep(0, length(lambda_linear_vector))
for (i in seq_along(lambda_rq_vector)) {
  x_data <- x_list[[i]]
  y_data <- y_list[[i]]
  variance_rq_lasso_fit <- rq(y_data ~ x_data[, -1], tau = tau, lambda = lambda_rq_ve
ctor[i], method = 'lasso')
  beta_variance_lasso_rq[[i]] <- variance_rq_lasso_fit$coefficients
  error_rq_vector[i] <- sum((true[[i]] - beta_variance_lasso_rq[[i]])^2)

  variance_linear_lasso_fit <- glmnet(x_data[, -1], y_data, lambda = lambda_linear_ve
ctor[i], alpha = 1)
  beta_variance_linear[[i]] <- rbind(variance_linear_lasso_fit$a0, variance_linear_la
sso_fit$beta)
  error_linear_vector[i] <- sum((true[[i]] - beta_variance_linear[[i]])^2)
}

error_rq_vector

## [1] 0.8364276 0.9519339 1.2584946 1.4379704

#  0.8364276 0.9519339 1.2584946 1.4379704
error_linear_vector

## [1] 0.4308296 0.4702891 0.5572955 0.6777991

# 0.4308296 0.4702891 0.5572955 0.6777991
```

이제 이를 전체 데이터에 넣어서 error 를 계산했다.

```
plot(y = error_linear_vector, x = stdv, type = 'b', lwd = 3, col = 'blue',
     ylim = c(0, 3))
lines(error_rq_vector, x = stdv, lwd = 3, col = 'red')
points(error_rq_vector, x = stdv, cex = 1.5, col = 'red', pch =16)
```




```
# quantile lasso의 추정오차가 더 크다.
```

추저오차를 시각화했다. 전체적으로 분산이 커질수록 추정오차가 커진다. 또한 이 경우엔 linear lasso 의 추정오차가 quantile lasso 보다 작다.

Adaptive Lasso 에 대하여

`beta_variance_lasso_rq`와 `beta_variance_linear`, 이 두 베타계수 벡터를 adaptive lasso 의 베타틸데로 사용하겠다.

```
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1
k = 10
x_list <- list(NULL)
y_list <- list(NULL)
each_rq_error_list <- list(NULL)
each_linear_error_list <- list(NULL)
true <- list(NULL)
each_error_rq <- matrix(0, nrow = length(lambda), ncol = k)  # rss를 각각 람다에 대해 저장하는 매트릭스
each_error_linear <- matrix(0, nrow = length(lambda), ncol = k)
for (l in seq_along(stdv)) {
  ind <- 1:n
```

```r
  ind_all <- 1:n
  set.seed(2019) # seed 지정
  n = 60; p = 30
  x = matrix(rnorm(n * p), ncol = p)
  x = cbind(rep(1,n), x)
  tau = 0.5
  beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
  obs_err = 9/10 * rnorm(n, sd = 1) + 1/10 * rnorm(n, mean = 0, sd = stdv[l])
  x[,2] = runif(n)
  y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
  beta_true = beta
  beta_true[2] = beta[2] + tau * qnorm(tau)
  beta_true[1] = beta[1] + qnorm(tau, sd = sd)
  true[[l]] <- beta_true
  x_list[[l]] <- x #
  y_list[[l]] <- y # 에러가 바뀔때마다 내용을 저장
  for (i in 1:k) {
    ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
    ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
    ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
    ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
    train_x <- x[ind_each, ]
    train_y <- y[ind_each]
    valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
    valid_y <- y[ind_for]
    for (j in seq_along(lambda)) {
      rq_beta_tilde <- beta_variance_lasso_rq[[l]] %>% abs
      rq_penalty <- c(0, lambda[j] / rq_beta_tilde[-1])
      each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = rq_penalty, me
thod = 'lasso')
      each_beta_rq <- each_lasso_rq$coefficients
      each_predicted_y <- x[ind_for, ] %*% each_beta_rq
      each_error_rq[j, i] <- quantile(abs(valid_y - each_predicted_y), 0.5)

      linear_beta_tilde <- beta_variance_linear[[l]] %>% abs
      linear_beta_tilde <- ifelse(linear_beta_tilde == 0, 1e-08, linear_beta_tilde)
      # 베타가 0 일 경우, 가중치가 무한으로 잡히는걸 방지해주기 위해
      linear_penalty <- c(0, lambda[j] / linear_beta_tilde[-1])
      each_lasso_linear <- glmnet(train_x[, -1], train_y,
                                  lambda = linear_penalty, alpha = 1)
      each_beta_linear <- rbind(each_lasso_linear$a0, each_lasso_linear$beta)
      each_linear_predicted_y <- x[ind_for, ] %*% each_beta_linear
      each_error_linear[j,i] <- sum((valid_y - each_linear_predicted_y)^2)
    }
  }
  each_rq_error_list[[l]] <- each_error_rq
  each_linear_error_list[[l]] <- each_error_linear
  #print(l)
}
```

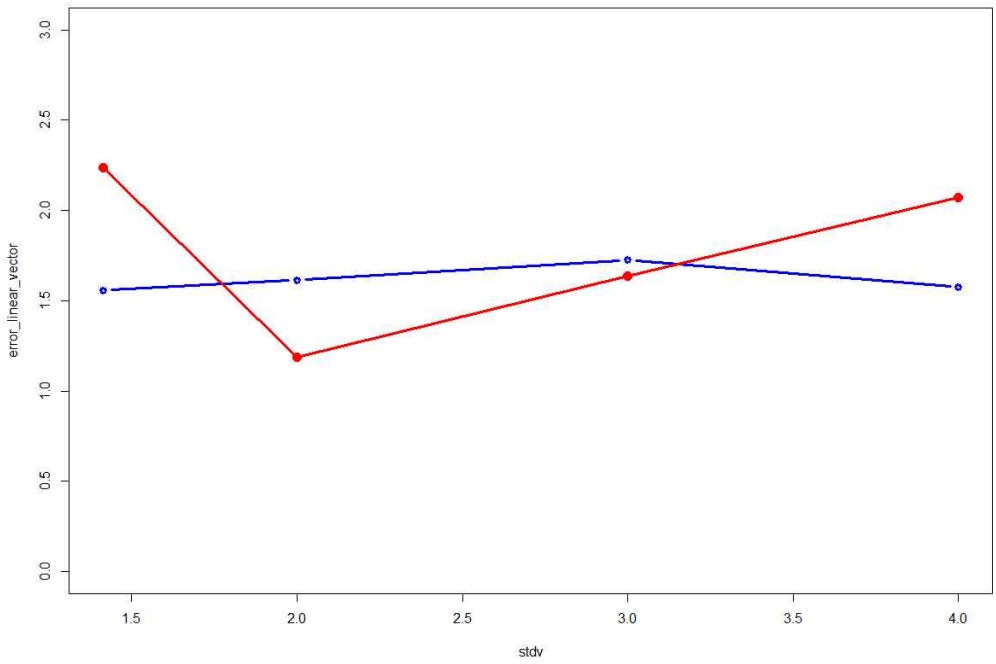
이전과정과 전체적으로 동일하지만, adaptive laaso 의 페널티를 고려하고, 그 페널티가 `Inf` 로 설정되지 않도록 고려했다.

```
each_lam_rq_error_list <- list(NULL)
each_lam_linear_error_list <- list(NULL)
min_rq_error_list <- list(NULL)
min_linear_error_list <- list(NULL)
min_rq_index_list <- list(NULL)
min_linear_index_list <- list(NULL)
min_rq_lasso_list <- list(NULL)
min_linear_lasso_list <- list(NULL)
for (i in seq_along(stdv)) {
  each_lam_rq_error_list[[i]] <- apply(each_rq_error_list[[i]], 1, mean)
  each_lam_linear_error_list[[i]] <- apply(each_linear_error_list[[i]], 1, mean)
  min_rq_error_list[[i]] <- min(each_lam_rq_error_list[[i]])
  min_linear_error_list[[i]] <- min(each_lam_linear_error_list[[i]])
  min_rq_index_list[[i]] <- which(each_lam_rq_error_list[[i]] == min_rq_error_list
[[i]])
  min_linear_index_list[[i]] <- which(each_lam_linear_error_list[[i]] == min_linear_e
rror_list[[i]])
  min_rq_lasso_list[[i]] <- lambda[min_rq_index_list[[i]]]
  min_linear_lasso_list[[i]] <- lambda[min_linear_index_list[[i]]]
}

lambda_rq_vector <- min_rq_lasso_list %>% unlist
lambda_linear_vector <- min_linear_lasso_list %>% unlist
lambda_rq_vector # 0.64 1.10 1.07 0.99

## [1] 0.64 1.10 1.07 0.99

lambda_linear_vector # 0.01 0.01 0.01 0.02

## [1] 0.01 0.01 0.01 0.02

variance_rq_laaso_fit <- NULL
variance_linear_lasso_fit <- NULL
beta_variance_lasso_rq <- list(NULL)
beta_variance_linear <- list(NULL)
error_rq_vector <- rep(0, length(lambda_rq_vector))
error_linear_vector <- rep(0, length(lambda_linear_vector))
for (i in seq_along(lambda_rq_vector)) {
  x_data <- x_list[[i]]
  y_data <- y_list[[i]]
  variance_rq_lasso_fit <- rq(y_data ~ x_data[, -1], tau = tau, lambda = lambda_rq_ve
ctor[i], method = 'lasso')
  beta_variance_lasso_rq[[i]] <- variance_rq_lasso_fit$coefficients
  error_rq_vector[i] <- sum((true[[i]] - beta_variance_lasso_rq[[i]])^2)

  variance_linear_lasso_fit <- glmnet(x_data[, -1], y_data, lambda = lambda_linear_ve
ctor[i], alpha = 1)
  beta_variance_linear[[i]] <- rbind(variance_linear_lasso_fit$a0, variance_linear_la
sso_fit$beta)
  error_linear_vector[i] <- sum((true[[i]] - beta_variance_linear[[i]])^2)
```

```
}

error_rq_vector

## [1] 2.235412 1.187215 1.636302 2.072584

# 2.235412 1.187215 1.636302 2.072584
error_linear_vector

## [1] 1.557832 1.613845 1.726111 1.575780

# 1.557832 1.613845 1.726111 1.575780
```

이번 adaptive lasso 의 경우는 quantile regression 보다 linear regression 의 에러가 더 적다. 명확한 이유는 모르겠다.

```
plot(y = error_linear_vector, x = stdv, type = 'b', lwd = 3, col = 'blue',
     ylim = c(20, 70), xlim = c(1, 4))
lines(error_rq_vector, x = stdv, lwd = 3, col = 'red')
points(error_rq_vector, x = stdv, cex = 1.5, col = 'red', pch =16)
```




```
# quantile regression 에서는 첫 값을 제외하곤 분산이 커질 수록 추정오차가 증가하는 경향이 있다.
```

SCAD 에 대하여

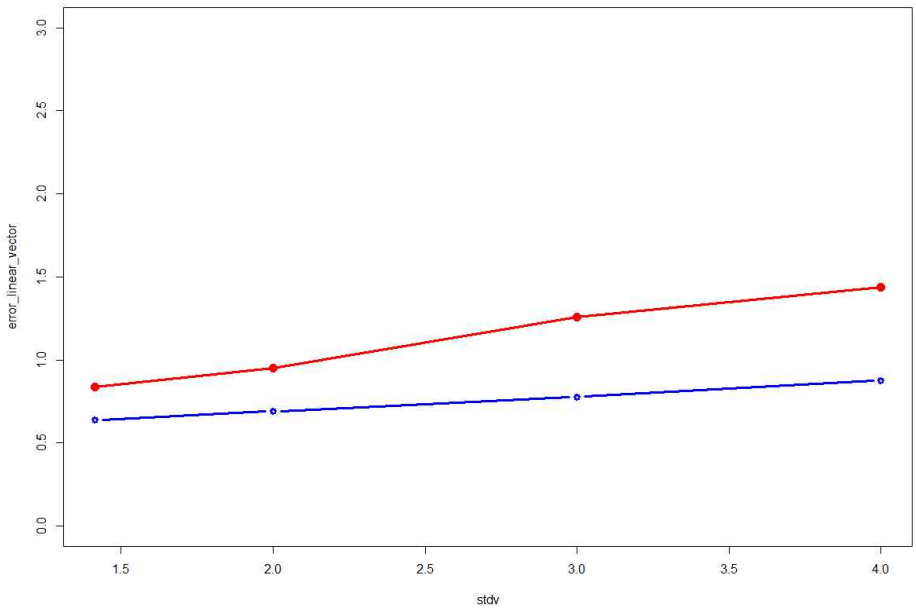
```r
lambda = seq(0.01, 10, by = 0.01)
ind <- 1:n
ind_all <- 1
k = 10
x_list <- list(NULL)
y_list <- list(NULL)
each_rq_error_list <- list(NULL)
each_linear_error_list <- list(NULL)
true <- list(NULL)
each_error_rq <- matrix(0, nrow = length(lambda), ncol = k)  # rss 를 각각 람다에 대해 저장하는 매트릭스
each_error_linear <- matrix(0, nrow = length(lambda), ncol = k)
for (l in seq_along(stdv)) {
  ind <- 1:n
  ind_all <- 1:n
  set.seed(2019) # seed 지정
  n = 60; p = 30
  x = matrix(rnorm(n * p), ncol = p)
  x = cbind(rep(1,n), x)
  tau = 0.5
  beta = matrix(c(rep(0.5, 7), rep(0, p - 6)))
  obs_err = 9/10 * rnorm(n, sd = 1) + 1/10 * rnorm(n, mean = 0, sd = stdv[l])
  x[,2] = runif(n)
  y = x %*% beta + matrix(tau * x[,2] * obs_err) + obs_err
  beta_true = beta
  beta_true[2] = beta[2] + tau * qnorm(tau)
  beta_true[1] = beta[1] + qnorm(tau, sd = sd)
  true[[l]] <- beta_true
  x_list[[l]] <- x #
  y_list[[l]] <- y # 에러가 바뀔때마다 내용을 저장
  for (i in 1:k) {
    ind_all <- 1:n  # 1 부터  n 까지 인덱스가 있다.
    ind_for <- sample(ind, n/k, replace = F)  # 이중에 n/k = 6 개를 비복원추출한다.
    ind <- setdiff(ind, ind_for)  # 한번 뽑혔으면 다음에 뽑히면 안된다.
    ind_each <- ind_all[-ind_for]  # 뽑힌 친구들을 제외한 train set 의 인덱스
    train_x <- x[ind_each, ]
    train_y <- y[ind_each]
    valid_x <- x[ind_for, ]  # 뽑힌 친구들이 validation set 의 인덱스로 활용
    valid_y <- y[ind_for]
    for (j in seq_along(lambda)) {
      rq_beta_tilde <- beta_variance_lasso_rq[[l]] %>% abs
      rq_weight = c(0, scad_deriv(rq_beta_tilde[-1], lam = lambda[j])) # 스캐드 미분함수의 가중치
      each_lasso_rq <- rq(train_y ~ train_x[, -1], tau = tau, lambda = rq_weight, method = 'lasso')
      each_beta_rq <- each_lasso_rq$coefficients
      each_predicted_y <- x[ind_for, ] %*% each_beta_rq
      each_error_rq[j, i] <- quantile(abs(valid_y - each_predicted_y),0.5)
```

```
      linear_beta_tilde <- beta_variance_linear[[l]] %>% abs
      linear_weight = c(0, scad_deriv(linear_beta_tilde[-1], lam = lambda[j]))
      each_lasso_linear <- glmnet(train_x[, -1], train_y,
                                  lambda = linear_weight, alpha = 1)
      each_beta_linear <- rbind(each_lasso_linear$a0, each_lasso_linear$beta)
      each_linear_predicted_y <- x[ind_for, ] %*% each_beta_linear
      each_error_linear[j,i] <- sum((valid_y - each_linear_predicted_y)^2)
    }
  }
  each_rq_error_list[[l]] <- each_error_rq
  each_linear_error_list[[l]] <- each_error_linear
  #print(l)
}
```

이전과 전체적인 과정은 유사하나, 스캐드 미분함수를 사용한 점이 다르다.

```
each_lam_rq_error_list <- list(NULL)
each_lam_linear_error_list <- list(NULL)
min_rq_error_list <- list(NULL)
min_linear_error_list <- list(NULL)
min_rq_index_list <- list(NULL)
min_linear_index_list <- list(NULL)
min_rq_lasso_list <- list(NULL)
min_linear_lasso_list <- list(NULL)
for (i in seq_along(stdv)) {
  each_lam_rq_error_list[[i]] <- apply(each_rq_error_list[[i]], 1, mean)
  each_lam_linear_error_list[[i]] <- apply(each_linear_error_list[[i]], 1, mean)
  min_rq_error_list[[i]] <- min(each_lam_rq_error_list[[i]])
  min_linear_error_list[[i]] <- min(each_lam_linear_error_list[[i]])
  min_rq_index_list[[i]] <- which(each_lam_rq_error_list[[i]] == min_rq_error_list
[[i]])
  min_linear_index_list[[i]] <- which(each_lam_linear_error_list[[i]] == min_linear_e
rror_list[[i]])
  min_rq_lasso_list[[i]] <- lambda[min_rq_index_list[[i]]]
  min_linear_lasso_list[[i]] <- lambda[min_linear_index_list[[i]]]
}

lambda_rq_vector <- min_rq_lasso_list %>% unlist
lambda_linear_vector <- min_linear_lasso_list %>% unlist
lambda_rq_vector# 4.23 3.11 3.06 2.97

## [1] 4.23 3.11 3.06 2.97

lambda_linear_vector # 0.18 0.19 0.20 0.21

## [1] 0.18 0.19 0.20 0.21

variance_rq_laaso_fit <- NULL
variance_linear_lasso_fit <- NULL
beta_variance_lasso_rq <- list(NULL)
beta_variance_linear <- list(NULL)
error_rq_vector <- rep(0, length(lambda_rq_vector))
error_linear_vector <- rep(0, length(lambda_linear_vector))
```

```r
for (i in seq_along(lambda_rq_vector)) {
  x_data <- x_list[[i]]
  y_data <- y_list[[i]]
  variance_rq_lasso_fit <- rq(y_data ~ x_data[, -1], tau = tau, lambda = lambda_rq_ve
ctor[i], method = 'lasso')
  beta_variance_lasso_rq[[i]] <- variance_rq_lasso_fit$coefficients
  error_rq_vector[i] <-  sum((true[[i]] - beta_variance_lasso_rq[[i]])^2)

  variance_linear_lasso_fit <- glmnet(x_data[, -1], y_data, lambda = lambda_linear_ve
ctor[i], alpha = 1)
  beta_variance_linear[[i]] <- rbind(variance_linear_lasso_fit$a0, variance_linear_la
sso_fit$beta)
  error_linear_vector[i] <- sum((true[[l]] - beta_variance_linear[[i]])^2)
}

error_rq_vector

## [1] 0.8364276 0.9519339 1.2584946 1.4379704

# 0.8364276 0.9519339 1.2584946 1.4379704
error_linear_vector

## [1] 0.6378238 0.6912100 0.7783083 0.8776945

# 0.6378238 0.6912100 0.7783083 0.8776945

plot(y = error_linear_vector, x = stdv, type = 'b', lwd = 3, col = 'blue',
     ylim = c(30, 100), xlim = c(0,4))
lines(error_rq_vector, x = stdv, lwd = 3, col = 'red')
points(error_rq_vector, x = stdv, cex = 1.5, col = 'red', pch =16)
```



```
# SCAD 에서는  quantile regression 의 성능이  linear regression 의 성능보다  더 좋다.
```

SCAD 에서는 quantile regression 의 추정오차가 linear regression 의 추정오차보다 더 적다. 또한 전체적으로 분산이 증가할수록 추정오차가 증가한다.

7 번의 결론 전체적으로 분산이 증가할때에 추정오차가 증가한다. 또한 처음 가정에는 등분산이 가정되어있지 않지만, tau = 0.5 이기 때문에 큰 영향이 없어 linear regressiong model 에 오차가 크게 증가하지 않는 것으로 보인다.