STAT 542: Homework 3

Shashi Roshan (sroshan2)

Contents

Question 1 [70 Points] Lasso and Coordinate Descent

We will write our own lasso algorithm. You should only use functions in the R base package to write your lasso code. However, you can use other packages to generate data and check your answers. Perform the following steps.

a. [15 points]

Let's start with the objective function

$$\ell(\boldsymbol{\beta}) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda|\boldsymbol{\beta}|_1$$

Since we will utilize coordinate descent, derive the solution of β_j while holding all other parameters fixed. In particular, you should write the solution as a "soft-thresholding" function, which is in a similar format as Page 36 in the lecture note `Penalized`. Finally, write an R function for soft-thresholding in the form of `soft_th <- function(b, lambda)`. The function outputs

- $b - \lambda$ if $b > \lambda$

- $b + \lambda$ if $b < -\lambda$
- 0 otherwise

Que. 1. a.

$$\ell(\beta) = \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda |\beta|_1$$

Goal is to find $\underset{\beta_j}{\operatorname{argmin}} \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda|\beta|_1$

$$= \underset{\beta_j}{\operatorname{argmin}} \frac{1}{2n} \| y - X_j\beta_j - X_{(-j)}\beta_{(-j)} \|^2 + \lambda|\beta_j| + \lambda|\beta_{-j}|$$

[Let $r = y - X_{(-j)}\beta_{(-j)}$]

$$= \underset{\beta_j}{\operatorname{argmin}} \frac{1}{2n} \|r - X_j\beta_j\|^2 + \lambda|\beta_j| + \lambda|\beta_{-j}|$$

Differentiating wrt β_j, and setting it equal to 0:

[case 1) $\beta_j > 0$]

$$\Rightarrow -\frac{2}{2n} X_j^T (r - X_j\beta_j) + 0 + \lambda = 0$$

$$-\frac{X_j^T}{n} (r - X_j\beta_j) = -\lambda$$

$$\beta_j = \frac{X_j^T r - n\lambda}{X_j^T X_j}, \text{ for } X_j^T r - n\lambda > 0$$

[case 2) $\beta_j < 0$]

$$\Rightarrow \beta_j = \frac{X_j^T r + n\lambda}{X_j^T X_j}, \text{ for } X_j^T r + n\lambda < 0$$

finally:

$$\hat{\beta}_j = \begin{cases} \frac{X_j^T r - n\lambda}{X_j^T X_j}, & \text{if } X_j^T r - n\lambda > 0 \\ \frac{X_j^T r + n\lambda}{X_j^T X_j}, & \text{if } X_j^T r + n\lambda < 0 \\ 0, & \text{otherwise} \end{cases}$$

```
soft_th <- function(b, lambda) {
  if (b > lambda) {
    return (b - lambda)
  }
  else if (b < (-lambda)) {
    return (b + lambda)
  }
  else
    return(0)
}
```

b. [15 points]

Let's write one iteration of the coordinate descent algorithm that goes through variables 1 to p and update their parameter values. First, we will generate the following data:

For this question, you should pre-process (center and scale) the data such that $\sum_i X_{ij} = 0$ and $X_j^{\mathrm{T}} X_j = n$ for each j, and center (but not scale) the outcome, so that the intercept term is not needed. You should record the mean and sd of these transformations such that the parameter estimates on the original scale can be recovered (like HW2). Please be aware that the `scale()` function in R gives you $X_j^{\mathrm{T}} X_j = n - 1$ instead of n. After you transform the data, proceed with one iteration of the coordinate descent algorithm that updates each parameter once. You should consider two things (both were mentioned during our lectures) that may improve the efficiency of your algorithm:

- After standardizing the X variables, how your updates can be simplified?
- When calculating the partial residuals (you should have this term in your formula), inherit the residuals from the previous update to save calculations.

Pick $\lambda = 0.4$. Use the initial value of $\boldsymbol{\beta}$ being all zeros. However, your code should work for any initial values. Report the first ten entries of your $\boldsymbol{\beta}$ after this one iteration of updates. You should use the soft-thresholding `soft_th` during the update.

```
library(MASS)
set.seed(1)
n = 200
p = 500

# generate data
V = matrix(0.2, p, p)
diag(V) = 1
X_org = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V)) # 200*500
true_b = c(runif(10, -1, 1), rep(0, p-10))
y_org = X_org %*% true_b + rnorm(n) # 200*1
```

```
x_mean = colMeans(X_org)
x_sd = apply(X_org, 2, sd) * sqrt((n-1)/n)
y_mean = mean(y_org)

X = X_org
X = scale(X, center=TRUE, scale=FALSE)
X = apply(X, 2, function(y) y * (sqrt(200/199)/ sd(y))) #D
y = scale(y_org, center=TRUE, scale=FALSE)

lambda = 0.4

# initialize beta values
beta = c(rep(0, p))

for (j in 1:ncol(X))
{
  if(j==1){
    r = y - X[,-j] %*% beta[-j]
  }
  else{
```

```
    r = r - (X[, j-1] * beta[j-1]) + (X[, j] * beta[j])
  }

  beta[j] = t(r) %*% X[, j] / n
  beta[j] = soft_th(beta[j], lambda)
}

beta[1:10]
```

```
##  [1] 0.18330176 0.08744669 0.17079989 0.24813098 0.00000000 0.17953579 0.00000000
##  [8] 0.10104141 0.00000000 0.19245049
```

c. [15 points]

Now, let's finish this coordinate descent algorithm. Write a function `myLasso(X, y, lambda, tol, maxitr)`. Set the tolerance level `tol` = 1e-5, and `maxitr` = 100 as the default values. Use the "one loop" code that you just wrote part b), and integrate that into a grand for-loop code that will repeatedly updating the parameters up to `maxitr` runs. Check your parameter updates after each loop and stop the algorithm once the ℓ_1 distance between the parameter values before and after each iteration is less than the tolerance level. Again, use $\lambda = 0.4$, and initial value zero. You function should output the estimated parameter values.

```
myLasso <- function(X, y, beta = rep(0, ncol(X)), epsilon = 1e-5, maxitr = 100)
{

  for(itr in 1:maxitr){

    beta_initial = beta

    for (j in 1:ncol(X))
    {
      if(j==1){
        r = y - X[,-j] %*% beta[-j]
      }
      else{
        r = r - (X[, j-1] * beta[j-1]) + (X[, j] * beta[j])
      }

      beta[j] = t(r) %*% X[, j] / n
      beta[j] = soft_th(beta[j], lambda)
    }

    if (max(abs(beta_initial - beta)) < epsilon)
          break;

  }

  if (itr == maxitr) {
    cat("maximum iteration reached\n")
  }

  return(beta)
```

```
}

beta = myLasso(X, y)

beta[1:10]
```

```
##  [1] 0.06928720 0.00000000 0.09019752 0.18168886 0.00000000 0.16057251 0.00000000
##  [8] 0.07634888 0.00000000 0.24045213
```

d. [10 points]

Recover the parameter estimates on the original scale of the data. Report the nonzero parameter estimates. Check your results with the `glmnet` package on the first 11 parameters (intercept and the ten nonzero ones).

- If your result matches the `glmnet` package, report the ℓ_1 distance between your solution and their solution
- If your result does not match the `glmnet` package, discuss potential issues and fixes of your code

```
beta_unscaled = beta / x_sd
intercept = y_mean - sum((beta*x_mean) / x_sd)

unscaled_estimates = c(intercept, beta_unscaled)
unscaled_nonzero_estimates = unscaled_estimates[which(unscaled_estimates!=0)]
```

```
library(glmnet)
lasso_model = glmnet(X_org, y_org, alpha = 1, lambda = lambda)
lasso_model_estimates = lasso_model$beta
c(lasso_model$a0, lasso_model_estimates[1:10])
```

```
##          s0
## 0.05530008 0.07147648 0.00000000 0.09305485 0.17397865 0.00000000 0.16406666 0.00000000
##
## 0.07067729 0.00000000 0.22916558
```

```
l1dist = sum(
  abs(unscaled_estimates[2:length(unscaled_estimates)] - lasso_model_estimates),
  abs(unscaled_estimates[1] - lasso_model$a0)
)
l1dist # 0.0001024846
```

```
## [1] 0.0001024846
```

```
c(lasso_model$a0, lasso_model_estimates[which(lasso_model_estimates!=0)])
```

```
##          s0
## 0.05530008 0.07147648 0.09305485 0.17397865 0.16406666 0.07067729 0.22916558 0.06592934
##
## 0.02469750
```

The recovered unscaled beta estimates are similar to the beta estimates provided by the glmnet package. The l1 distance between them is 1.0248464×10^{-4}.

e. [15 points]

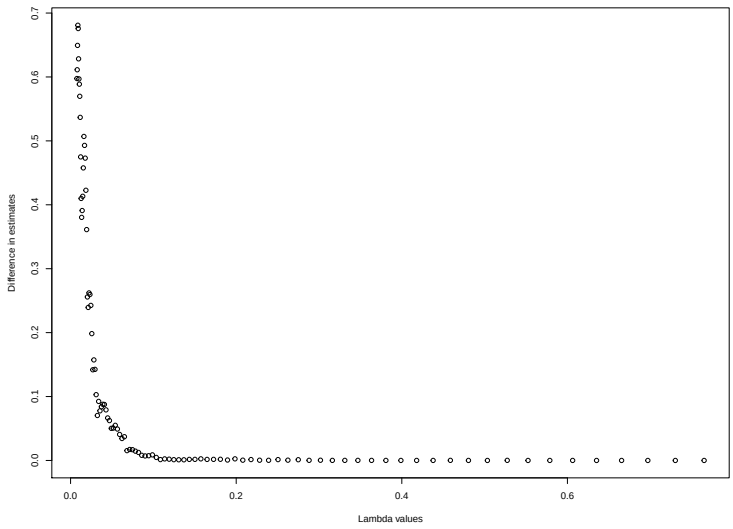
Let's modify our Lasso code to perform path-wise coordinate descent. The idea is simple: we will solve the solution on a grid of λ values, starting from the largest one. After obtaining the optimal $\boldsymbol{\beta}$ for a given λ, we simply use this solution as the initial value (instead of all zero) for the next (smaller) λ. This is referred to as a warm start in optimization problems. We will consider the following grid of λ:

```
# lmax = max(t(X) %*% y) / n
# where X is standardized and y is centered from the original data

lmax = 0.765074054
lambda_all = exp(seq(log(lmax), log(lmax*0.01), length.out = 100))

myLasso <- function(X, y, beta = rep(0, ncol(X)), epsilon = 1e-5, maxitr = 100, lambda_all)
{
  estimates_grid = matrix(0, nrow=p, ncol=length(lambda_all))

  for(lambda_idx in 1:length(lambda_all)) {

    for (itr in 1:maxitr) {
      lambda = lambda_all[lambda_idx]
      beta_initial = beta

      for (j in 1:ncol(X))
      {
        if (j == 1) {
          r = y - X[,-j] %*% beta[-j]
        }
        else{
          r = r - (X[, j - 1] * beta[j - 1]) + (X[, j] * beta[j])
        }

        beta[j] = t(r) %*% X[, j] / n
        beta[j] = soft_th(beta[j], lambda)
      }

      if (max(abs(beta_initial - beta)) < epsilon)
        break
    }

    estimates_grid[, lambda_idx] = beta

  }

  return(estimates_grid)

}

estimates_grid = myLasso(X, y, lambda_all = lambda_all)

# unscaled estimates
estimates_grid_unscaled = estimates_grid / x_sd

# unscaled intercept
```

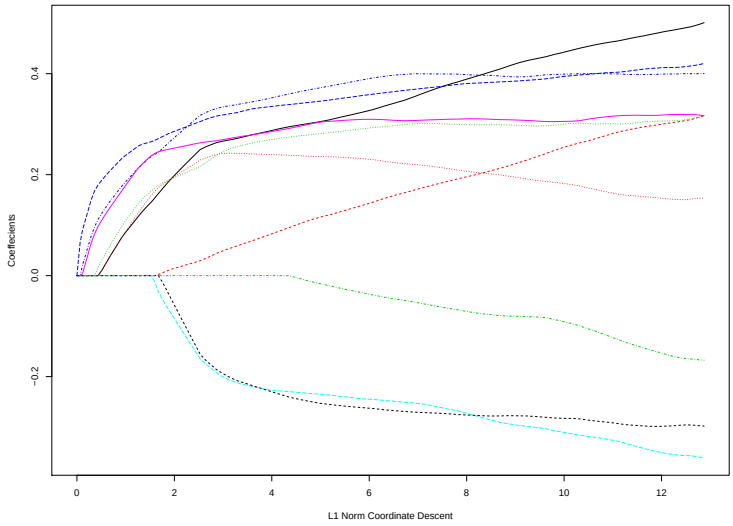
```
intercept = y_mean - colSums((estimates_grid*x_mean) / x_sd)

# glmnet
lasso_model = glmnet(X_org, y_org, alpha = 1, lambda = lambda_all)

# compare glmnet solution with own solution (unscaled data)
unscaled_estimates = rbind(intercept, estimates_grid_unscaled)
lasso_model_estimates = rbind(lasso_model$a0, lasso_model$beta)

diff = abs(unscaled_estimates - lasso_model_estimates)
max(colSums(diff))
```

```
## [1] 0.6808973
```

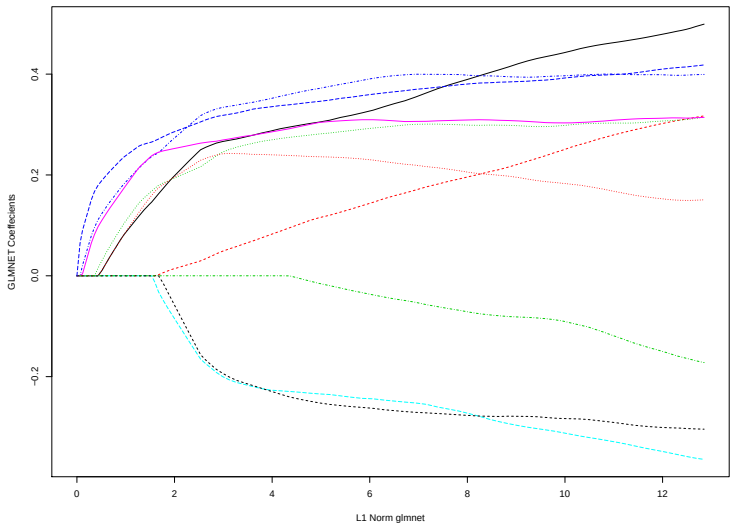
```
# plot difference vs lambda values
# output 1/2
plot(x = lambda_all , y = colSums(diff), xlab = 'Lambda values', ylab = 'Difference in estimates')
```




```
l1_norm_of_coefficients = colSums(abs(estimates_grid_unscaled))

l1norm = colSums(abs(estimates_grid_unscaled))

l1norm_glmnet = colSums(abs(lasso_model$beta))

# output 2/2
matplot(l1norm,
        t(estimates_grid_unscaled[1:10,]),
        type = "l",
        xlab = "L1 Norm Coordinate Descent",
        ylab = "Coeffecients")
```




```
matplot(l1norm_glmnet,
        t(lasso_model$beta[1:10,]),
        type = "l",
        xlab = "L1 Norm glmnet",
        ylab = "GLMNET Coeffecients")
```



After finishing your algorithm, you should have a matrix that records all the fitted parameters on your λ grid. Provide a plot similar to page 41 (right panel) in the lecture note "Penalized" using the first 10 (nonzero) parameters. Use parameter values on the original scale of the data.

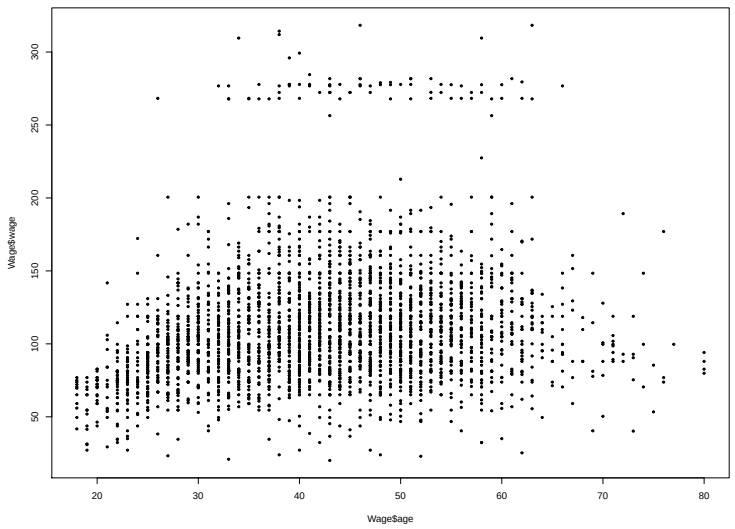
- Comment on the similarity between your solution and the `glmnet` solution.
  The estimates provided by myLasso() function is close to the estimates provided by glmnet() package. However, they are not exactly the same. The maximum l1 norm difference between them is 0.68.

- Does that vary across different λ values?
  As the lambda value increases, the difference in the estimates of myLasso() and glmnet decreases.
- Do you notice anything unusual about the plot? What could be the cause of such a phenomenon?
  Both the plots look almost same.
  This is because the way glmnet estimates beta is by first scaling and centering the X, and centering the Y, then it calculates the beta estimates, and then unscales them back to their original scale. This is similar to what we have done in myLasso() function. Hence, both the plots look almost same.

Question 2 [30 Points] Splines

We will fit and compare different spline models to the `Wage` dataset form the `ISLR` package.

```
library(ISLR)
data(Wage)
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
```



a. [20 points]

Let's consider several different spline methods to model `Wage` using `age`. For each method (except the smoothing spline), report the degrees of freedom. To test your model, use the first 300 observations of the data as the training data and the rest as testing data. Use the mean squared error as the metric.

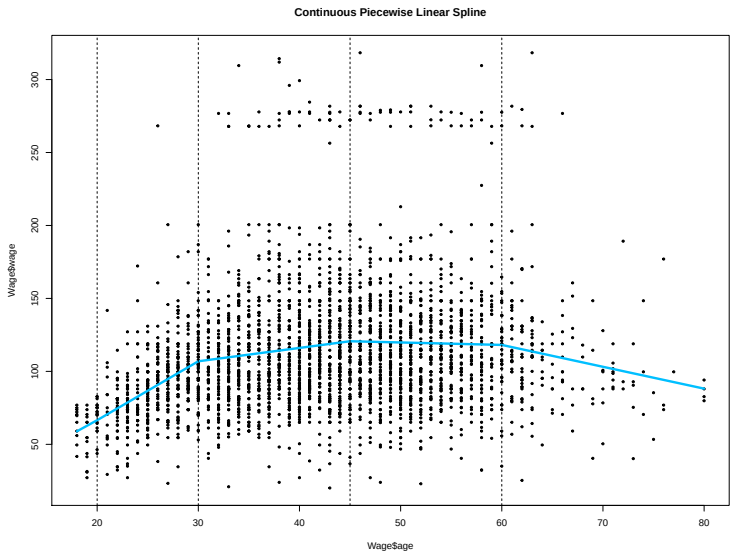
- Write your own code (you cannot use `bs()` or similar functions) to implement a continuous piecewise linear spline fitting. Pick 4 knots using your own judgment.

```
train_data = Wage[1:300,]

test_data = Wage[301:nrow(Wage),]

calculate_mse <- function(actual, predicted){
  return(mean((actual-predicted)^2))
```

```
}

myknots = c(20,30,45,60)
pos <- function(x) {x*(x>0)}
pos2 <- function(x) I(x^2)*(x>0)
```

```
# sort data for spline fit and plots (on entire data)
Wage = Wage[order(Wage$age),]
par(mar = c(2,2,2,0))
```

```
# plot using entire data
mybasis = cbind("int" = 1,
                "x_1" = Wage$age,
                "x_2" = pos(Wage$age - myknots[1]),
                "x_3" = pos(Wage$age - myknots[2]),
                "x_4" = pos(Wage$age - myknots[3]),
                "x_5" = pos(Wage$age - myknots[4]))

lmfit <- lm(Wage$wage ~ . -1, data = data.frame(mybasis))
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
lines(Wage$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
abline(v = myknots, lty = 2)
title("Continuous Piecewise Linear Spline")
```



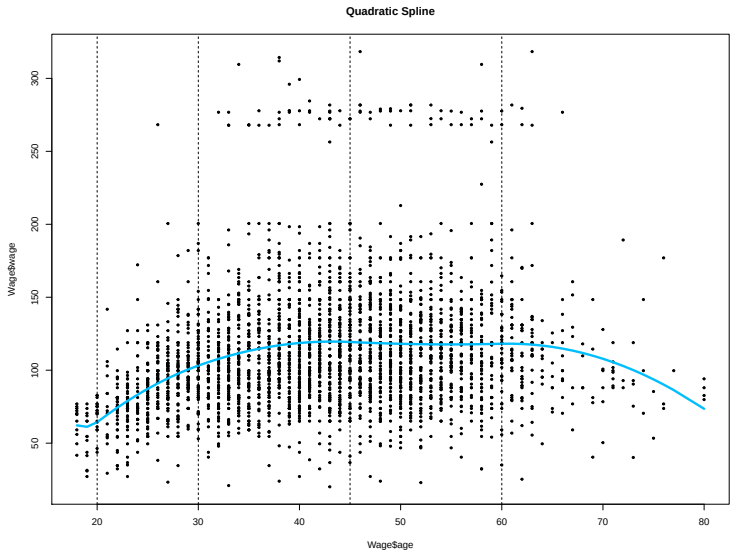

```
# model fitting
mybasis_train = cbind("int" = 1,
                "x_1" = train_data$age,
                "x_2" = pos(train_data$age - myknots[1]),
                "x_3" = pos(train_data$age - myknots[2]),
                "x_4" = pos(train_data$age - myknots[3]),
                "x_5" = pos(train_data$age - myknots[4]))

mybasis_test = cbind("int" = 1,
```

```
                "x_1" = test_data$age,
                "x_2" = pos(test_data$age - myknots[1]),
                "x_3" = pos(test_data$age - myknots[2]),
                "x_4" = pos(test_data$age - myknots[3]),
                "x_5" = pos(test_data$age - myknots[4]))

lmfit2 = lm(train_data$wage ~ . -1, data = data.frame(mybasis_train))
mse_1 = calculate_mse(test_data$wage, predict(lmfit2, newdata = data.frame(mybasis_test)))
mse_1
```

```
## [1] 1599.823
```

- Write your own code to implement a quadratic spline fitting. Your spline should have a continuous first derivative. Pick 4 knots using your own judgment.

```
# plot using entire data
mybasis = cbind("int" = 1,
                "x_1" = Wage$age,
                "x_2" = I(Wage$age^2),
                "x_3" = pos2(Wage$age - myknots[1]),
                "x_4" = pos2(Wage$age - myknots[2]),
                "x_5" = pos2(Wage$age - myknots[3]),
                "x_6" = pos2(Wage$age - myknots[4]))

lmfit <- lm(Wage$wage ~ . -1, data = data.frame(mybasis))
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
lines(Wage$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
abline(v = myknots, lty = 2)
title("Quadratic Spline")
```



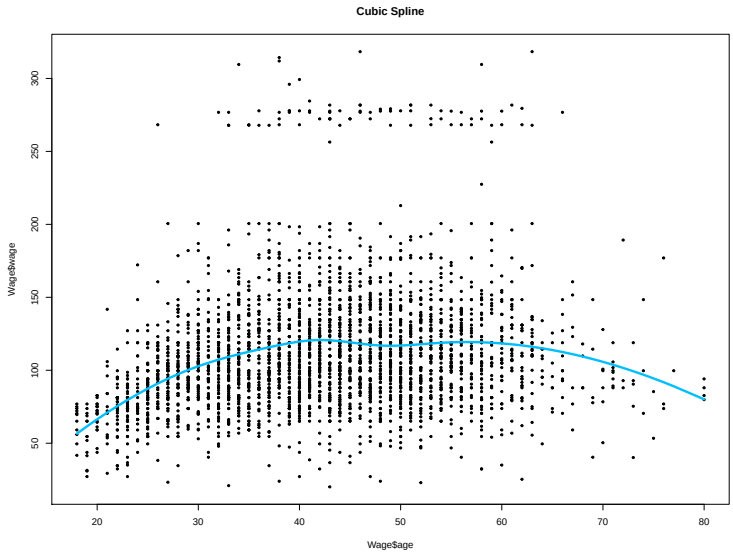

```
# model fitting
mybasis_train = cbind("int" = 1,
                "x_1" = train_data$age,
```

```
                "x_2" = I(train_data$age^2),
                "x_3" = pos2(train_data$age - myknots[1]),
                "x_4" = pos2(train_data$age - myknots[2]),
                "x_5" = pos2(train_data$age - myknots[3]),
                "x_6" = pos2(train_data$age - myknots[4]))


mybasis_test = cbind("int" = 1,
                "x_1" = test_data$age,
                "x_2" = I(test_data$age^2),
                "x_3" = pos2(test_data$age - myknots[1]),
                "x_4" = pos2(test_data$age - myknots[2]),
                "x_5" = pos2(test_data$age - myknots[3]),
                "x_6" = pos2(test_data$age - myknots[4]))

lmfit2 = lm(train_data$wage ~ . -1, data = data.frame(mybasis_train))
mse_2 = calculate_mse(test_data$wage, predict(lmfit2, newdata = data.frame(mybasis_test)))
mse_2
```

```
## [1] 1602.116
```

- Use existing functions (`bs()` or similar) to implement a cubic spline with 4 knots. Use their default knots.

```
# plot using entire data
lmfit = lm(Wage$wage ~ splines::bs(age, degree = 3, df=8), data = data.frame(Wage))
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
lines(Wage$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
title("Cubic Spline")
```

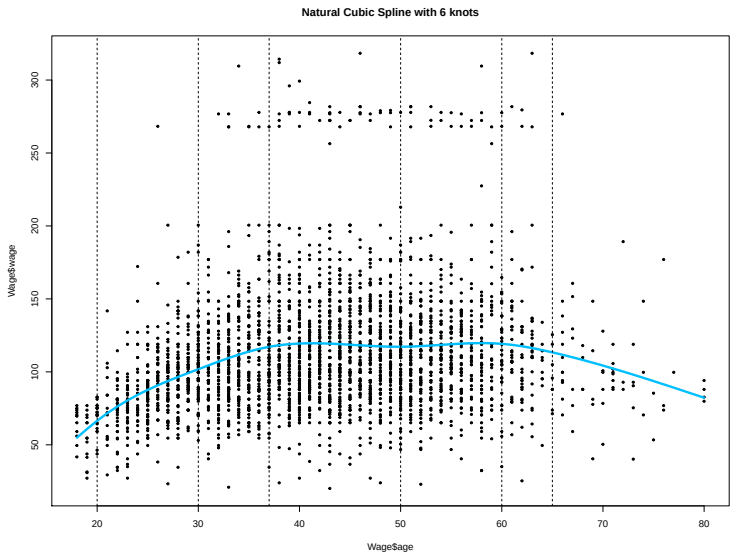



```
# model fitting
lmfit2 = lm(train_data$wage ~ splines::bs(age, degree = 3, df=8), data = data.frame(train_data))
mse_3 = calculate_mse(test_data$wage, predict(lmfit2, newdata = test_data))
mse_3
```

```
## [1] 1597.686
```

- Use existing functions to implement a natural cubic spline with 6 knots. Choose your own knots.

```
# plot using entire data
myknots_2 = c(20,30,37,50,60,65)
lmfit = lm(Wage$wage ~ splines::ns(age, df = 10, knots = myknots_2), data = data.frame(Wage))
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
lines(Wage$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
abline(v = myknots_2, lty = 2)
title("Natural Cubic Spline with 6 knots")
```



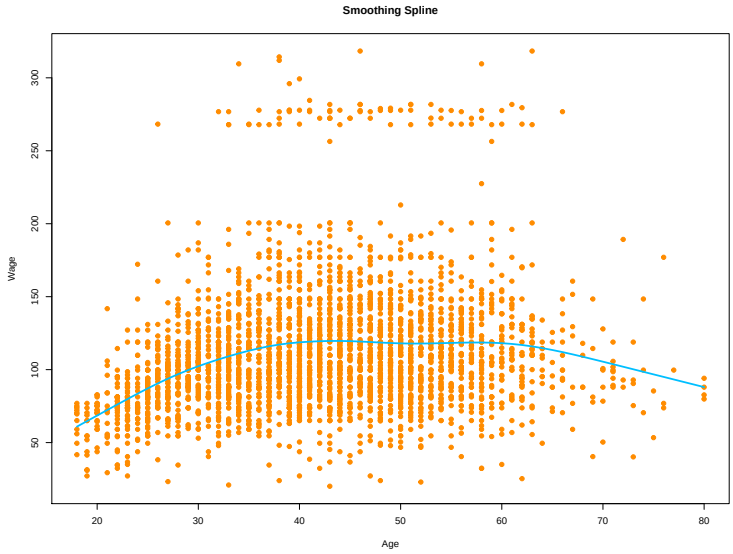

```
# model fitting
lmfit2 = lm(train_data$wage ~ splines::ns(age, df = 6, knots = myknots_2),
            data = data.frame(train_data))
mse_4 = calculate_mse(test_data$wage, predict(lmfit2, newdata = test_data))
mse_4
```

```
## [1] 1596.218
```

- Use existing functions to implement a smoothing spline. Use the built-in generalized cross-validation method to select the best tuning parameter.

```
# plot using entire data
fit = smooth.spline(x = Wage$age, y = Wage$wage)
plot(
  x = Wage$age,
  y = Wage$wage,
  pch = 19,
  xlab = "Age",
  ylab = "Wage",
  col = "darkorange"
)
```

```r
lines(
  seq(min(Wage$age), max(Wage$age)),
  predict(fit, seq(min(Wage$age), max(Wage$age)))$y,
  col = "deepskyblue",
  lty = 1,
  lwd = 3
)
title("Smoothing Spline")
```




```r
# model fitting
library(splines)
lmfit2 = smooth.spline(x = train_data$age, y = train_data$wage, cv=TRUE)
predictions = predict(lmfit2, test_data$age)
mse_5 = calculate_mse(test_data$wage, predictions$y)
mse_5
```

```
## [1] 1589.56
```

```r
library(kableExtra)
library(tibble)
tibble(
  "Spline Model" = c(
    "Continuous Piecewise Linear Spline",
    "Quadratic Spline",
    "Cubic Spline",
    "Natural Cubic Spline",
    "Smoothing Spline"
  ),
  "Mean Squared Error" = c(mse_1, mse_2, mse_3, mse_4, mse_5),
  "Degree of freddom" = c(
    '6 (#knots + 2)',
    '7 (#knots + 3)',
    '8 (#knots 4)',
```

```
    '6 (#knots)',
    round(lmfit2$df, 2)
  )
) %>%
  kable(digits = 2) %>%
  kable_styling("striped", full_width = FALSE)
```

Spline Model	Mean Squared Error	Degree of freddom
Continuous Piecewise Linear Spline	1599.82	6 (#knots + 2)
Quadratic Spline	1602.12	7 (#knots + 3)
Cubic Spline	1597.69	8 (#knots 4)
Natural Cubic Spline	1596.22	6 (#knots)
Smoothing Spline	1589.56	4.61

b. [10 points]

After writing these models, evaluate their performances by repeatedly doing a train-test split of the data. Randomly sample 300 observations as the training data and the rest as testing data. Repeat this process 200 times. Use the mean squared error as the metric.

- Record and report the mean, median, and standard deviation of the errors for each method. Also, provide an informative boxplot that displays the error distribution for all models side-by-side.
- For each method, provide a discussion of their (theoretical) advantages and disadvantages. Based on the empirical results, what method would you prefer?

```
library(dplyr)
library(splines)
```

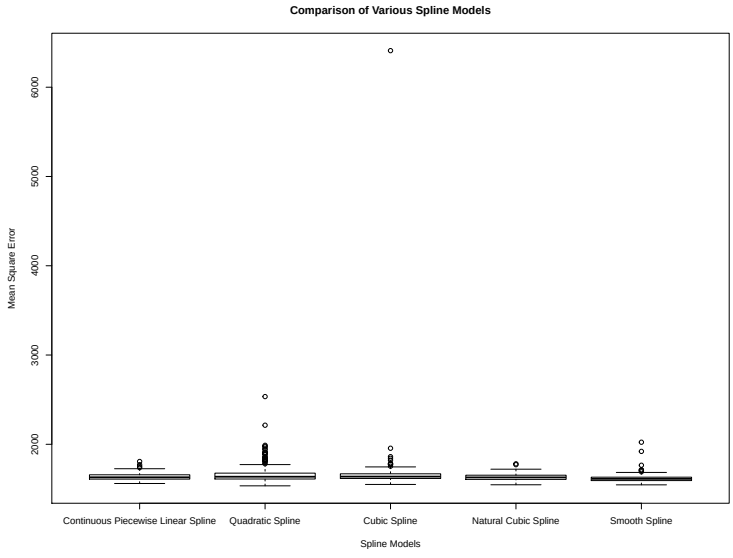
```
lm_mse = c()
quadratic_model_mse = c()
cubic_mse = c()
natural_mse = c()
smoothing_mse = c()

for(iteration in 1:200){

  set.seed(iteration)
  train_ind = sample(seq_len(nrow(Wage)), size = 300)
  train_data = Wage[train_ind, ]
  test_data = Wage[-train_ind, ]

  myknots_train = seq(min(train_data$age), max(train_data$age), length=4+2)[2:5]
  myknots_test = seq(min(test_data$age), max(test_data$age), length=4+2)[2:5]

  myknots_2_train = seq(min(train_data$age), max(train_data$age), length=6+2)[2:7]

  # 1 Continuous Piecewise Linear Spline
  mybasis_train = cbind("int" = 1,
                  "x_1" = train_data$age,
                  "x_2" = pos(train_data$age - myknots_train[1]),
                  "x_3" = pos(train_data$age - myknots_train[2]),
```

```
                  "x_4" = pos(train_data$age - myknots_train[3]),
                  "x_5" = pos(train_data$age - myknots_train[4]))

  mybasis_test = cbind("int" = 1,
                  "x_1" = test_data$age,
                  "x_2" = pos(test_data$age - myknots_test[1]),
                  "x_3" = pos(test_data$age - myknots_test[2]),
                  "x_4" = pos(test_data$age - myknots_test[3]),
                  "x_5" = pos(test_data$age - myknots_test[4]))

  lmfit2 = lm(train_data$wage ~ . - 1, data = data.frame(mybasis_train))
  lm_mse = c(lm_mse, calculate_mse(test_data$wage, predict(lmfit2, newdata = data.frame(mybasis_test))))

  # 2 Quadratic spline
  mybasis_train = cbind("int" = 1,
                "x_1" = train_data$age,
                "x_2" = I(train_data$age^2),
                "x_3" = pos2(train_data$age - myknots_train[1]),
                "x_4" = pos2(train_data$age - myknots_train[2]),
                "x_5" = pos2(train_data$age - myknots_train[3]),
                "x_6" = pos2(train_data$age - myknots_train[4]))

  mybasis_test = cbind("int" = 1,
                "x_1" = test_data$age,
                "x_2" = I(test_data$age^2),
                "x_3" = pos2(test_data$age - myknots_test[1]),
                "x_4" = pos2(test_data$age - myknots_test[2]),
                "x_5" = pos2(test_data$age - myknots_test[3]),
                "x_6" = pos2(test_data$age - myknots_test[4]))

  lmfit2 = lm(train_data$wage ~ . -1, data = data.frame(mybasis_train))
  quadratic_model_mse = c(quadratic_model_mse, calculate_mse(test_data$wage, predict(lmfit2, newdata = d

  # 3 Cubic spline
  lmfit2 = lm(train_data$wage ~ splines::bs(age, degree = 3, df=8), data = data.frame(train_data))
  cubic_mse = c(cubic_mse, calculate_mse(test_data$wage, predict(lmfit2, newdata = test_data)))

  # 4 Natural Cubic Spline with 6 knots
  lmfit2 = lm(train_data$wage ~ splines::ns(age, df = 6, knots = myknots_2_train), data = data.frame(tra
  natural_mse = c(natural_mse, calculate_mse(test_data$wage, predict(lmfit2, newdata = test_data)))

  # 5 Smoothing spline
  lmfit2 = smooth.spline(x = train_data$age, y = train_data$wage, cv=TRUE)
  predictions = predict(lmfit2, test_data$age)
  smoothing_mse = c(smoothing_mse, calculate_mse(test_data$wage, predictions$y))

}
```

```
tibble(
  "Spline Model" = c(
    "Continuous Piecewise Linear Spline",
    "Quadratic Spline",
    "Cubic Spline",
```

```
    "Natural Cubic Spline with 6 knots",
    "Smoothing Spline"
  ),
  "Mean" = c(
    mean(lm_mse),
    mean(quadratic_model_mse),
    mean(cubic_mse),
    mean(natural_mse),
    mean(smoothing_mse)
  ),
  "Median" = c(
    median(lm_mse),
    median(quadratic_model_mse),
    median(cubic_mse),
    median(natural_mse),
    median(smoothing_mse)
  ),
  "Standard Deviation" = c(
    sd(lm_mse),
    sd(quadratic_model_mse),
    sd(cubic_mse),
    sd(natural_mse),
    sd(smoothing_mse)
  )
) %>%
  kable(digits = 2) %>%
  kable_styling("striped", full_width = FALSE)
```

Spline Model	Mean	Median	Standard Deviation
Continuous Piecewise Linear Spline	1636.15	1630.35	39.20
Quadratic Spline	1672.34	1636.82	115.43
Cubic Spline	1672.57	1638.27	340.76
Natural Cubic Spline with 6 knots	1633.09	1629.09	36.09
Smoothing Spline	1618.95	1612.98	48.25



Linear Spline:

Advantage:

- Since they are linear, they will not overshoot at the boundaries.

Diadvantages:

- At the knots, the estimates are not smooth. This leads to not so smooth estimates around the knots.
- The selection of knots, and the number of knots - both are decided by the user. Poor selection of knots will give bad results.

Cubic and Quadratic Spline :

Advantages:

- The function is smooth around the knots, and provide better estimates compared to linear splines.

Disadvantages:

- They perform very badly in case of extrapolation (extrapolation should be avoided). These tend to overshoot at the boundaries as they are not linear beyond the boundary points.
- The selection of knots, and the number of knots - both are decided by the user. Poor selection of knots will give bad results.

Natural Cubic spline:

Advantages:

- Beyond the boundary points, they are linear. Hence they don't overshoot, and can handle extrapolation to some extent.
- More stable estimates at boundaries, with narrower confidence interval.

Disadvantages:

- The selection of knots, and the number of knots - both are decided by the user. Poor selection of knots will give bad results.

Smoothing Splines:

Advantages:

- Every data point is a knot. User does not have to select the knots position.

Disadvantage:

- You have to estimate as many parameters as you have data and yet the effect of many of those parameters will in general be low because of the penalty against overly complex (wiggly) fits.