# portfolio_5

2023-03-02

## Ridge Regression, LASSO and Smoothing

### Task 1

For this task I will be using the Communities and Crime Dataset, let's first load it in and then split it into a training and test set:

```
library(mogavs)
data(crimeData)

n.test <- round(0.15 * nrow(crimeData))
split <- sample(1:nrow(crimeData), n.test)
CC.train <- crimeData[-split,]
CC.test <- crimeData[split,]
y.index <- ncol(crimeData)
```

First we would like to fit a GLM using LASSO regression. To get our model we would also like to make sure we are using a suitable value for $\lambda$, to select $\lambda$ we will use cross validation. We will measure the performance of each value of $\lambda$ using the mean cross validation error for 10 fold cross-validation and pick the value of $\lambda$ that minimizes the error . Both this and consequently fitting a GLM using LASSO are carried out by **glmnet()**:

```
library(glmnet)
```

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

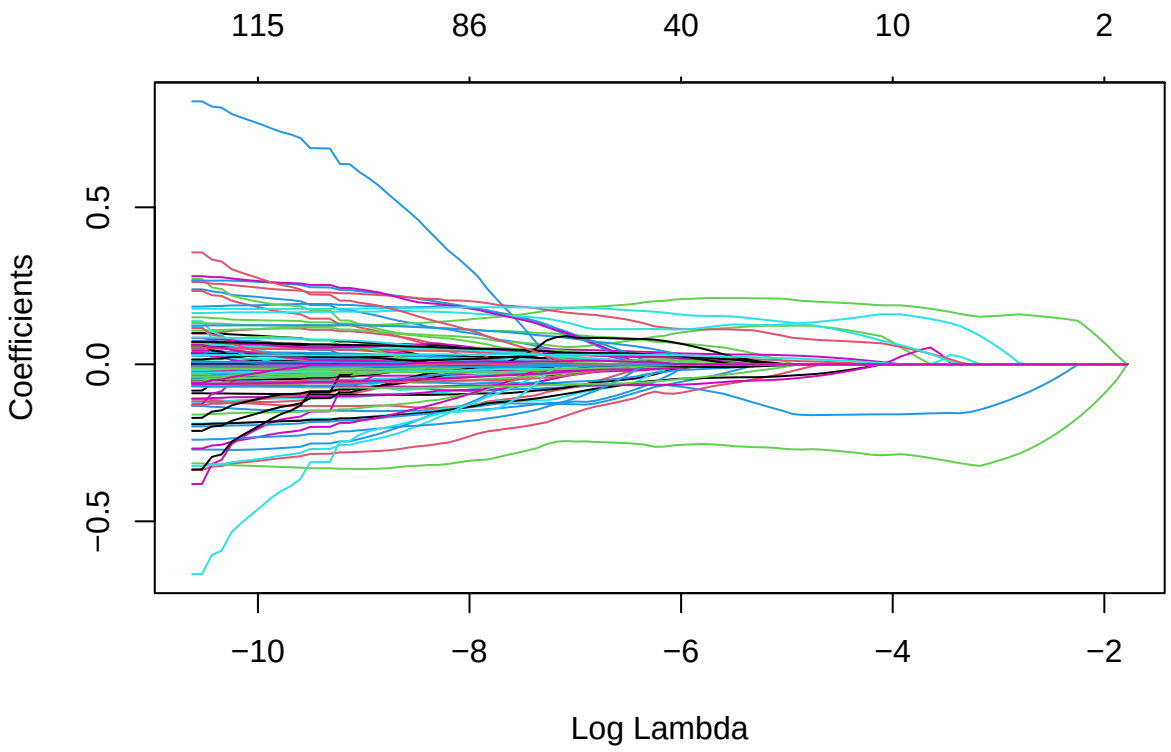
```
## Loaded glmnet 4.1-6
```

```
fit.lasso <- glmnet(CC.train[,-y.index], CC.train[,y.index], family = "gaussian", standardize=TRUE, alp
```

One thing to note about the above is that we set *standardize* = *TRUE* as we would like all the data to be on the same scale and to be centered. By setting *standardize* = *TRUE* behind the scenes glmnet will standardize and unstandardize automatically. For example when we later use *predict()* with new data it will automatically standardize the data and when we use *coefficient()* to get the coefficients it will automatically unstandardize the coefficients. So as long as we are working with glmnet functions we do not need to worry about standardization from here.

Now let's plot the LASSO path:

```
plot(fit.lasso, xvar="lambda")
```

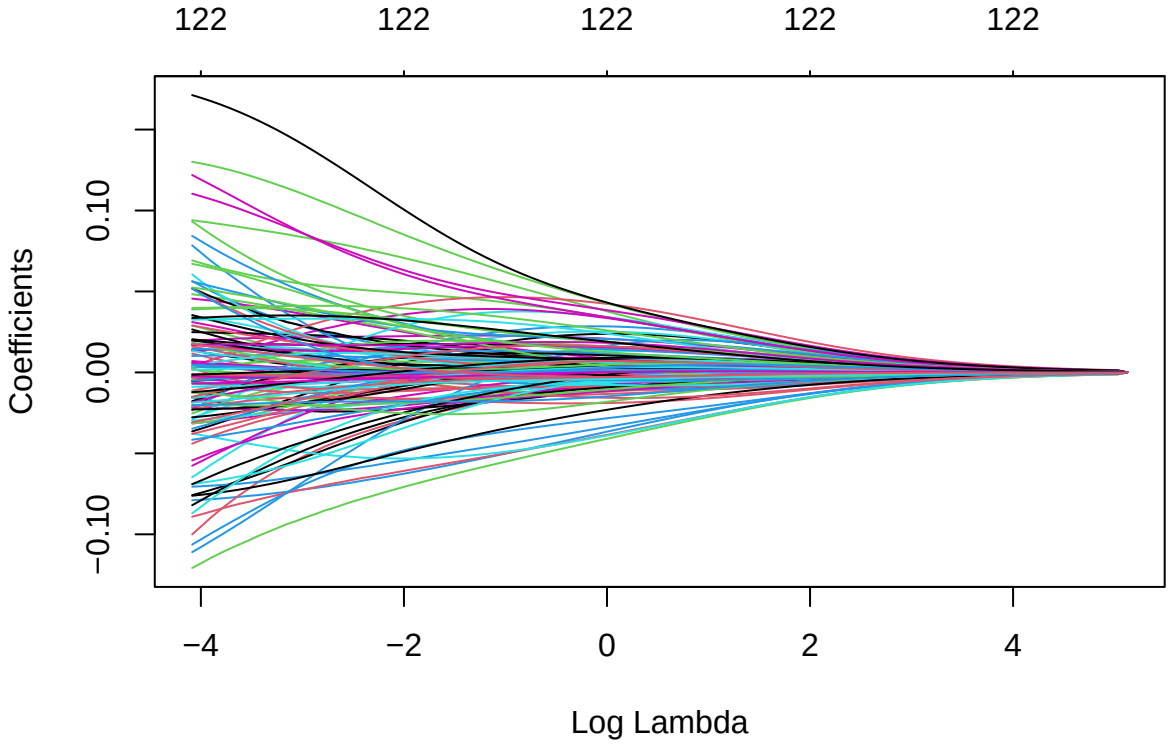



Let's now carry out ridge regression:

```
fit.ridge <- glmnet(CC.train[,-y.index], CC.train[,y.index], family = "gaussian", standardize=TRUE, alp
```

We can now plot the ridge path:

```
plot(fit.ridge, xvar="lambda")
```

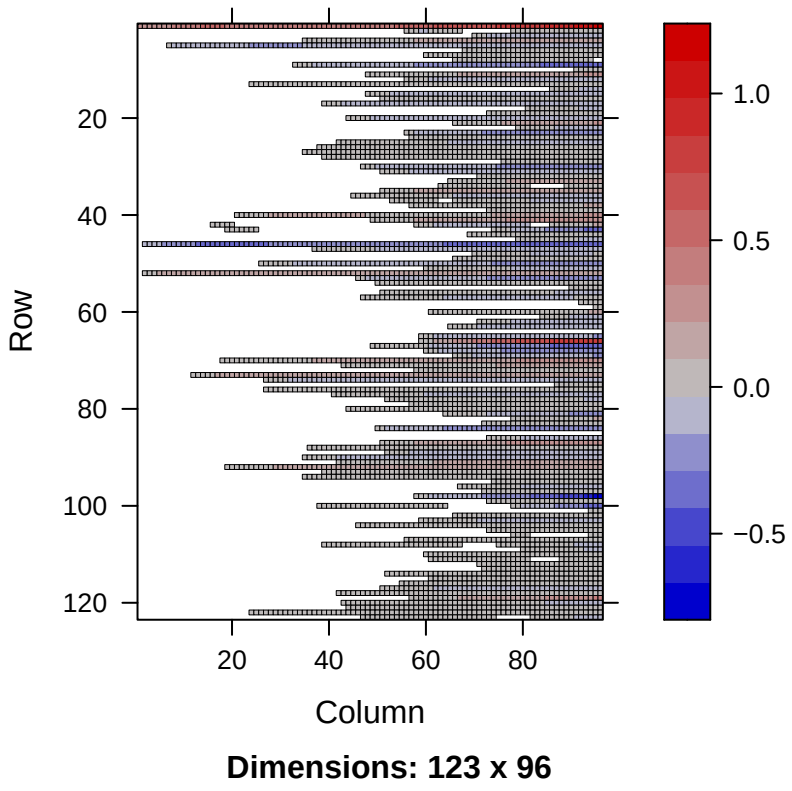



Observing the ridge path we see all the coefficients start close to zero and as $\lambda$ increases in general the magnitude

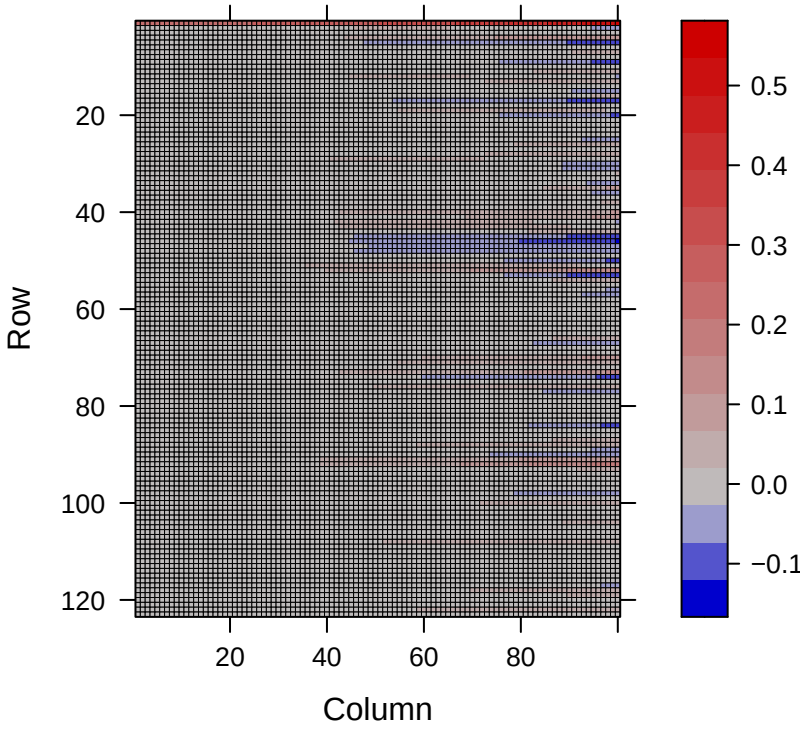
of the coefficients increases, in comparison for ridge regression we have some more complex behaviour. Some coefficients being by increasing in magnitude, then decrease, then the value of other coefficients begins increasing.

Now let's compare the coefficients, estimates of $\beta$, obtained with LASSO and with ridge regression for all the different values of lambda we used:

```
image(coefficients(fit.lasso))
```



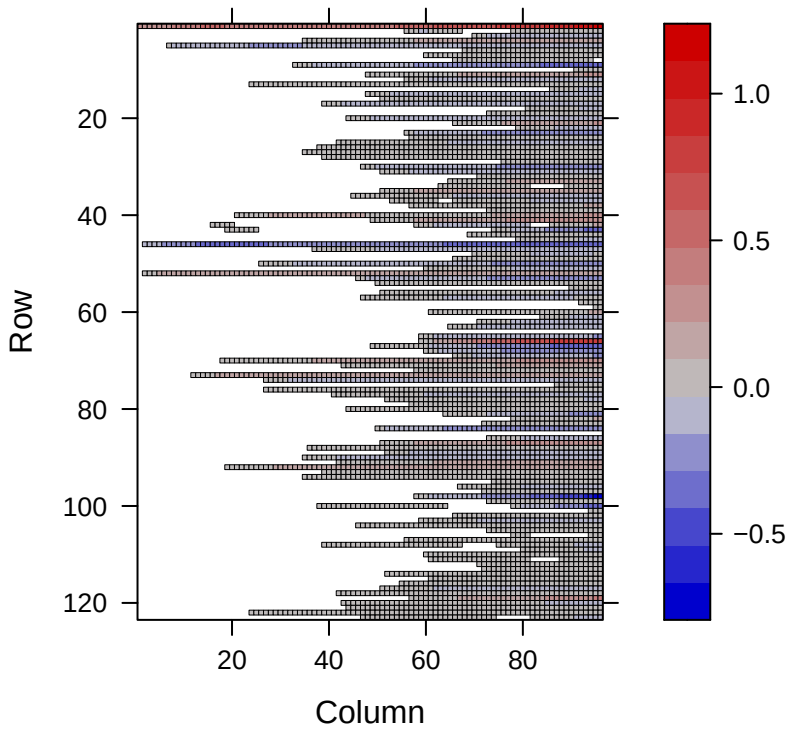

```
image(coefficients(fit.ridge))
```




**Dimensions: 123 x 100**

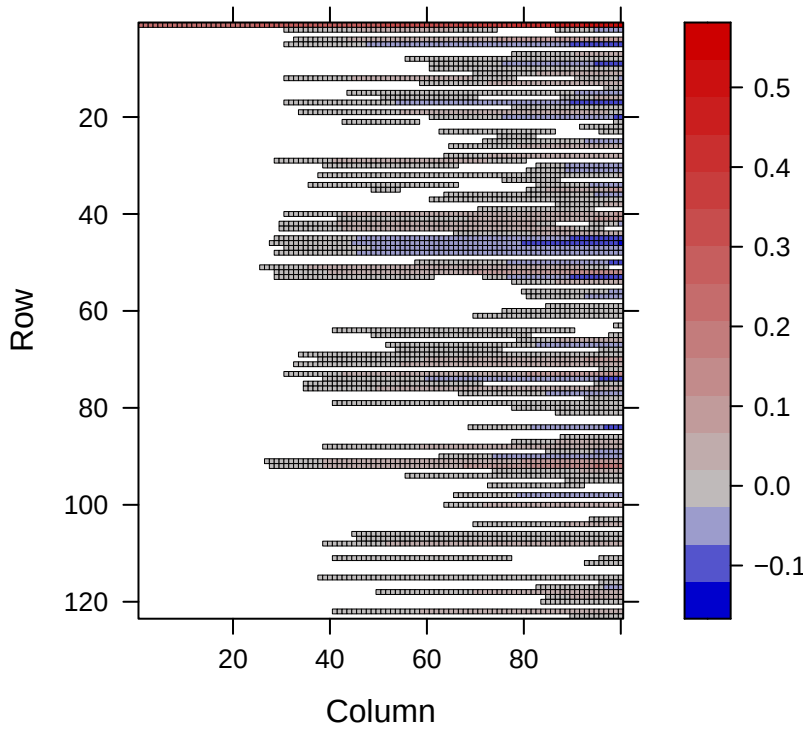
The first thing we notice is the amount of the plot that is white in LASSO compared to with ridge, this is because many of the coefficients obtained with LASSO are equal to zero, whereas with ridge although alot have values with very small magnitudes they aren't exactly equal to zero. If we produce the same plots but map values to zero if they are below some small threshold value then we get the following plots:

```
image(coefficients(fit.lasso))
```




**Dimensions: 123 x 96**

```
image(drop0(coefficients(fit.ridge), 0.01))
```

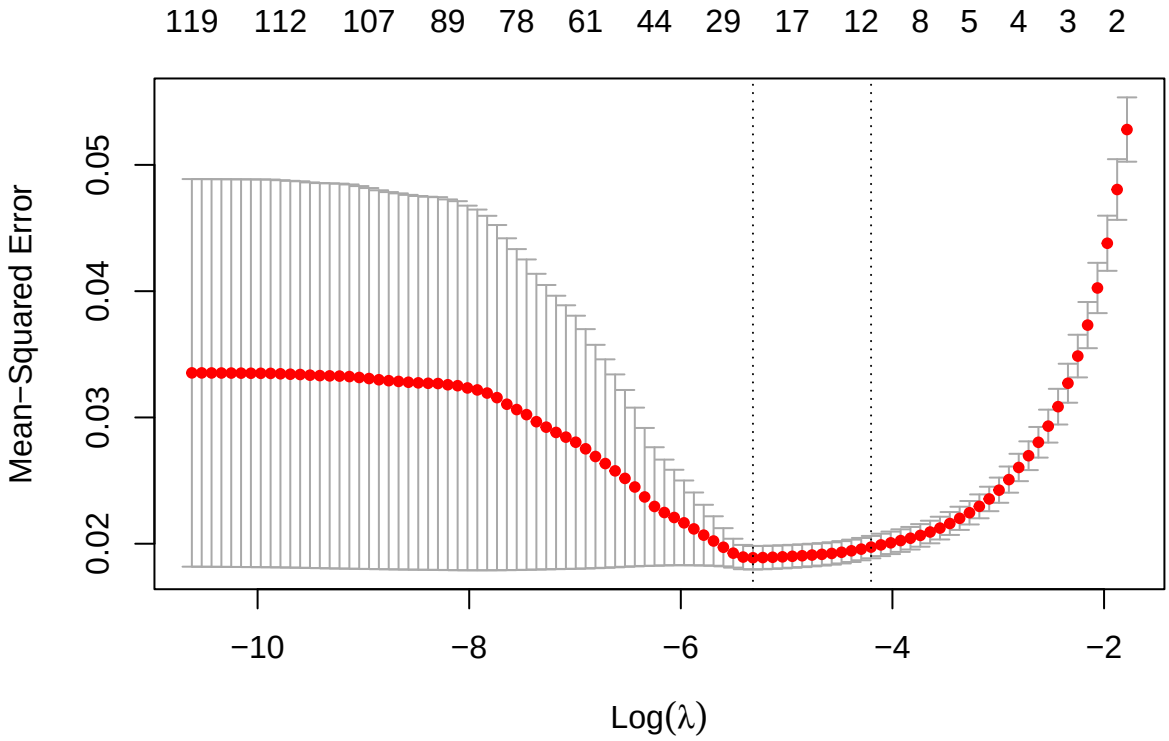



**Dimensions: 123 x 100**

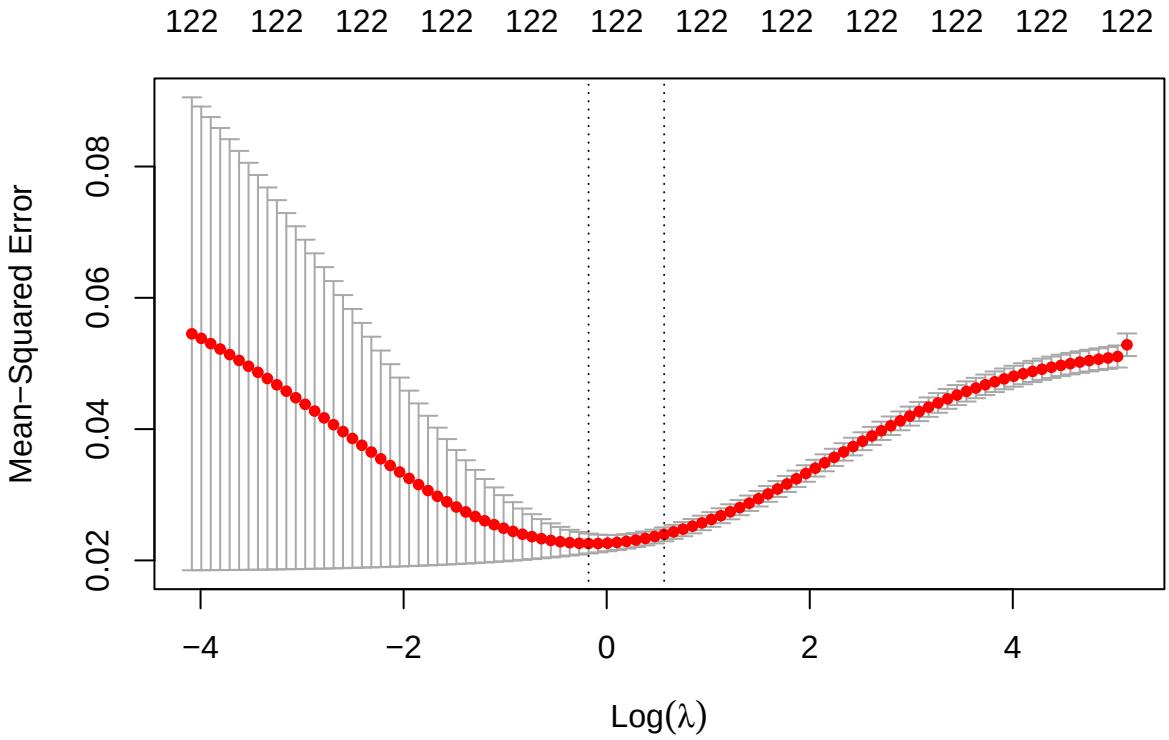
We see here that if we disregard values close to zero the estimate of $\beta$ obtained with ridge regression looks on the whole pretty similar to that we obtain with LASSO.

Let's first pick our values of lambda that we will use, we will carry out cross validation (10 folds) again and produce some plots and then we will use the values of lambda that give the minimum mean cross validation error:

```
lambda.lasso <- cv.glmnet(as.matrix(CC.train[,-y.index]), as.matrix(CC.train[,y.index]), alpha=1)
plot(lambda.lasso)
```




```
lambda.ridge <- cv.glmnet(as.matrix(CC.train[,-y.index]), as.matrix(CC.train[,y.index]), alpha=0)
plot(lambda.ridge)
```




```
lambda.lasso.val <- lambda.lasso$lambda.min
lambda.ridge.val <- lambda.ridge$lambda.min
```

Let's now use the test set to compare the out-of-sample prediction error of two models with the values of lambda we found above, first we will calculate our predictions and then the accuracy of the predictions:

```
preds.lasso <- predict(fit.lasso, newx= as.matrix(CC.test[,-y.index]), s=lambda.lasso.val)
preds.ridge <- predict(fit.ridge, newx= as.matrix(CC.test[,-y.index]), s=lambda.ridge.val)

diff <- as.vector(CC.test[,y.index]) - preds.lasso
mse.lasso <- mean((diff)^2)

diff <- as.vector(CC.test[,y.index]) - preds.ridge
mse.ridge <- mean((diff)^2)

mse.lasso
```

```
## [1] 0.02034117
```

```
mse.ridge
```

```
## [1] 0.02478031
```

We see that the value for the MSE obtained by LASSO is a little bit less than that obtained by ridge, this combined with the fact that the LASSO model is "more compact" as it essentially has a lower-dimensional $\beta$ (as many of the coefficients are exactly zero) are evidence for LASSO regression giving the better model in this scenario.

## Task 2

For this task I will use the Bone Mineral Density dataset and try to reproduce Figure 5.6 of "The Elements of Statistical Learning". To this aim we would like to fit a separate smoothing spline for males and for females. The functions we are trying to estimate is the non-linear relationship between age and relative change in

spinal BMD.

```
BMD <- read.csv("portfolio_5/spnbmd.csv")
head(BMD)
```

```
##   idnum ethnic  age sex spnbmd
## 1     1  White 11.2 mal  0.719
## 2     1  White 12.2 mal  0.732
## 3     1  White 13.2 mal  0.776
## 4     1  White 14.3 mal  0.781
## 5     2  White 12.7 mal  0.620
## 6     2  White 13.8 mal  0.627
```

The first thing to do is calculate the rate of change:

```
BMD <- BMD[order(BMD$id, BMD$age),]
BMD$sex <- as.factor(BMD$sex)
BMD$rc <- NA

for (id in as.numeric(BMD$idnum)) {
  BMD.dash <- BMD[BMD$idnum == id,]
  if (nrow(BMD.dash) > 1) {
    spnbmd_diff <- diff(BMD.dash$spnbmd)
    rc <- c(NA, spnbmd_diff / BMD.dash$spnbmd[1:(nrow(BMD.dash)-1)])
    BMD[BMD$idnum == id, "rc"] <- rc
  }
}

BMD <- BMD[complete.cases(BMD),]
```

Now we have added the Rate of Change (rc) to our data frame we want to estimate our smoothed function for both males and females which we will do now using the help of the "mgcv" package and the "gam" function. Note that the gam function automatically carries out GCV and selects lambda according to the results.

```
library(mgcv)
```

```
## Loading required package: nlme
```

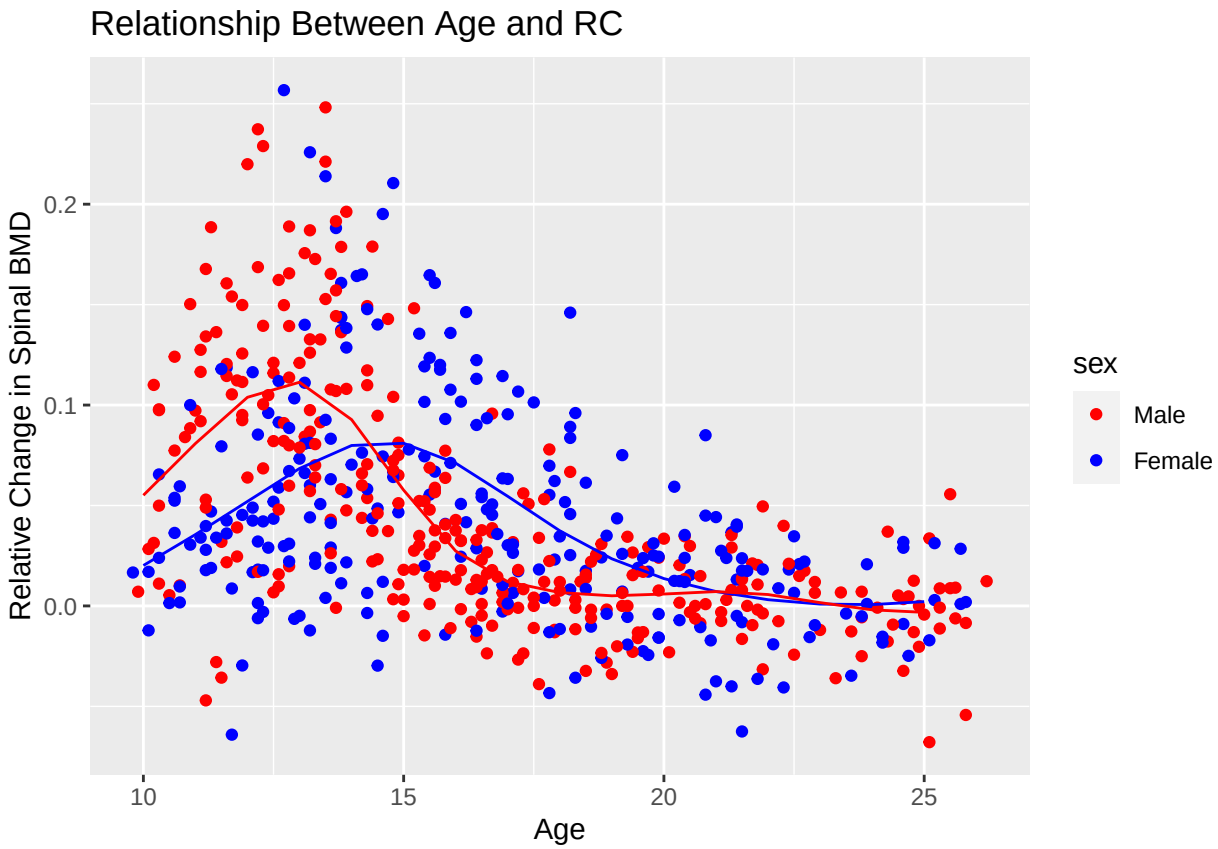
```
## This is mgcv 1.8-41. For overview type 'help("mgcv-package")'.
```

```
xs <- seq(10,25)

# Estimate function for males
fit.male <- gam(rc~s(age), data=BMD[BMD$sex == "mal",], method="GCV.Cp")
rc.fit.male <- predict(fit.male, data.frame(age=xs))

# Estimate function for females
fit.female <- gam(rc~s(age), data=BMD[BMD$sex == "fem",], method="GCV.Cp")
rc.fit.female <- predict(fit.female, data.frame(age=xs))
```

Now we will plot our fits:

```
library(ggplot2)
ggplot(data = BMD, aes(x = age, y = rc, color = sex)) +
  geom_point() +
  scale_color_manual(values = c("red", "blue"),
                     labels = c("Male", "Female")) +
  geom_line(data = data.frame(age=xs, rc=rc.fit.male), color = "blue") +
```

```
  geom_line(data = data.frame(age=xs, rc=rc.fit.female), color = "red") +
  labs(title = "Relationship Between Age and RC",
       x = "Age",
       y = "Relative Change in Spinal BMD")
```




Comparing the above to the plot in "The Elements of Statistical Learning", we notice that the two plots are very similar looking and that our estimated functions appear to fit the data pretty well. However, they do differ a bit in appearance to the figure in the book, for example the curves look more "wiggly". The reason for this is that the values used for $\lambda$ are different, in the book they use $\lambda \approx 0.00022$ and here we use:

```
fit.male$gcv.ubre
```

```
##      GCV.Cp
## 0.002175492
```

```
fit.female$gcv.ubre
```

```
##      GCV.Cp
## 0.001777194
```

Notice these values of lambda are much larger than the value used in the book, this explains why in the book it is more "wiggly" (there's less smoothing going on). The reason for different values of lambda being obtained include randomness during GCV, a small sample size (only 580 observations) or could be down to different initial values being used in our optimization algorithms.