Solution to Series 9

1. **a)** Since the matrix $\mathbf{X}$ is the identity matrix, with the OLS we are minimizing the loss

$$\sum_{i=1}^{n}(y_i - \beta_i)^2,$$

this decouples into n independent optimizations of the loss

$$(y_i - \beta_i)^2,$$

for $i = 1, \ldots, n$. The derivative with respect to β_i is $-2(y_i - \beta_i)$. Setting this equal to zero yields $\hat{\beta}_i^{OLS} = y_i$ for all i.

b) Since the matrix $\mathbf{X}$ is the identity matrix, with the ridge we are minimizing the penalized loss

$$\sum_{i=1}^{n}(y_i - \beta_i)^2 + \lambda \sum_{i=1}^{n}\beta_i^2,$$

this decouples once again into n independent optimizations of the loss

$$(y_i - \beta_i)^2 + \lambda\beta_i^2,$$

for $i = 1, \ldots, n$. The derivative with respect to β_i is $-2(y_i - \beta_i) + 2\lambda\beta_i$. Setting this equal to zero yields $\hat{\beta}_i^R = y_i/(1+\lambda)$ for all i.

c) Since the matrix $\mathbf{X}$ is the identity matrix with the lasso we are minimizing the penalized loss

$$\sum_{i=1}^{n}(y_i - \beta_i)^2 + \lambda \sum_{i=1}^{n}|\beta_i|,$$

once again this decouples in n independent optmizations of the loss

$$(y_i - \beta_i)^2 + \lambda|\beta_i|,$$

for $i = 1, \ldots, n$.
We derive the solution to this minimization problem by using subdifferentials (the definition and some useful properites of subdifferential can be found in Wikipedia). If we take the subdifferential of the loss function with respect to β_i we obtain

$$-2(y_i - \beta_i) + \partial\left(\lambda|\beta_i|\right).$$

Let $\hat{\beta}_i^L$ be the global minimun of the above loss function, then 0 must be contained in the subdefferential of the loss function at point $\hat{\beta}_i^L$, that is,

$$0 \in -2(y_i - \hat{\beta}_i^L) + \partial(\lambda|\hat{\beta}_i^L|),$$

which is equivalent to

$$\hat{\beta}_i^L - y_i \in -\frac{1}{2}\partial(\lambda|\hat{\beta}_i^L|).$$

The subdifferential of the function $f(x) = \lambda|x|$ is given by

$$\partial f(x) = \begin{cases} \lambda, & \text{if } x > 0 \\ -\lambda, & \text{if } x < 0 \\ [-\lambda, \lambda], & \text{if } x = 0 \end{cases}.$$

Therefore, if $\hat{\beta}_i^L$ is the global minimun, we should have

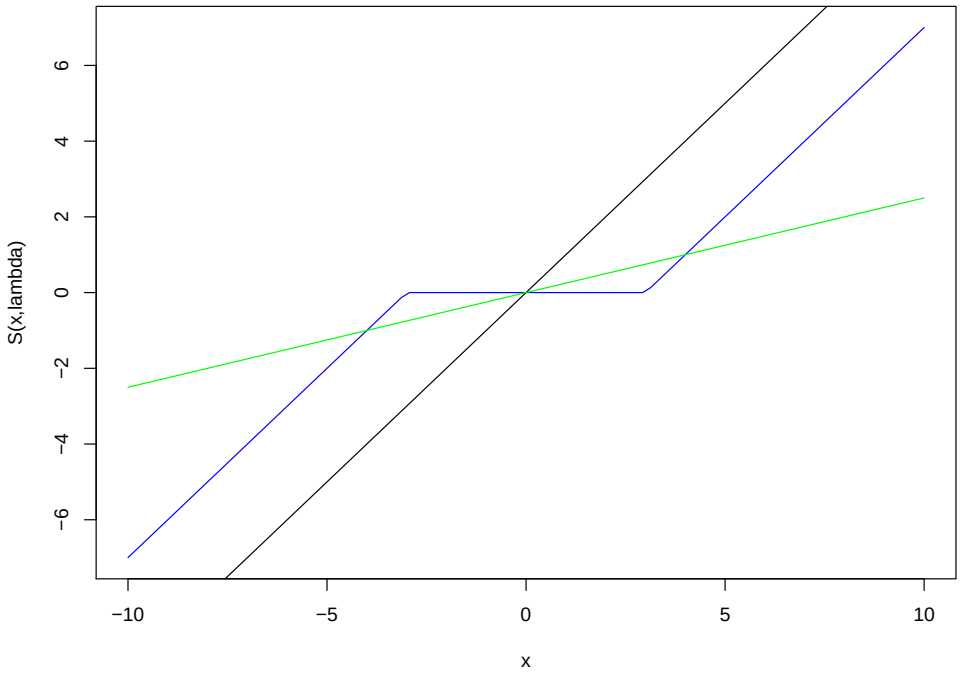
$$\hat{\beta}_i^L - y_i \in \begin{cases} \{-\lambda/2\}, & \text{if } \hat{\beta}_i^L > 0 \\ \{\lambda/2\}, & \text{if } \hat{\beta}_i^L < 0 \\ [-\lambda/2, \lambda/2], & \text{if } \hat{\beta}_i^L = 0 \end{cases} \qquad (*)$$

Now we discuss three situations: $y_i > \lambda/2$, $y_i < -\lambda/2$ and $|y_i| \leq \lambda/2$.
When $y_i > \lambda/2$, first assume that the global minimum $\hat{\beta}_i^L < 0$, then from $(*)$, we have that $\hat{\beta}_i^L = y_i + \lambda/2 < 0$, which implies $y_i < -\lambda/2 < \lambda/2$. This contradicts to $y_i > \lambda/2$, hence the global minimum $\hat{\beta}_i^L \geq 0$. Now assume that the global minimum $\hat{\beta}_i^L = 0$, then from $(*)$, we have $|y_i| \leq \lambda/2$, which contradict to $y_i > \lambda/2$, Hence the global minimum $\hat{\beta}_i^L > 0$. And by $(*)$, we have $\hat{\beta}_i^L = y_i - \lambda/2$.
Similarly, we have that when $y_i < -\lambda/2$, $\hat{\beta}_i^L = y_i + \lambda/2$. When $|y_i| \leq \lambda/2$, $\hat{\beta}_i^L = 0$.
In summary,

$$\hat{\beta}_i^L = \begin{cases} y_i - \lambda/2, & \text{if } y_i > \lambda/2 \\ y_i + \lambda/2, & \text{if } y_i < -\lambda/2 \\ 0, & \text{if } |y_i| \leq \lambda/2 \end{cases}.$$

d) The following plot shows the shrinkage properties of the lasso (blue), ridge (green) and OLS (black). We remark that in comparison to the ridge, the lasso allows some coefficients to be shrunk entirely to zero (and therefore allows for variable selection). Lasso provides more shrinkage than ridge for values of x that are close to zero, while ridge provides more shrinkage than lasso for values of x that are far away from zero.

```
> x <- seq(-10,10,length=100)
> lambda <- 3
> st <- numeric(length=length(x))
> st[which(x > lambda)] <- x[which(x > lambda)] - lambda
> st[which(x < -lambda)] <- x[which(x < -lambda)] + lambda
> sr <- numeric(length=length(x))
> sr <- x/(1+lambda)
> par(mfrow=c(1,1))
> plot(x,st,type="l",col="blue",xlab = c("x"), ylab = c("S(x,lambda)"))
> lines(x,x,type="l")
> lines(x,sr,type="l",col="green")
```



2. **a)** Let T_1 be the test statistic for the t-test associated to β_1. Note that T_1 is a random variable. Let F_0 be the CDF of T_1 under H_0. The p-value is computed as $p_1 = P(|\tilde{T}_1| > |T_1|)$ where $\tilde{T}_1 \sim F_0$ is a random variable independent of T_1. Equivalently,

$$p_1 = P(\tilde{T}_1^2 \geq T_1^2) = 1 - F_1(T_1^2), \qquad (\star)$$

where F_1 is the cumulative distribution function of T_1^2 under H_0 (so F_1 is also the CDF of $\tilde{T}_1^2$ under H_0 because $\tilde{T}_1$ and T_1 are identically distributed). From the last relation, you can clearly observe that the p-value p_1 is a random variable depending on T_1. We need to show that under H_0, p_1 has a uniform distribution on $[0, 1]$, i.e.,

$$P(p_1 \leq \alpha \mid H_0) = \alpha,$$

or equivalently

$$P(p_1 \geq 1 - \alpha \mid H_0) = \alpha.$$

The latter is shown as follows. For $\alpha \in [0, 1]$, we have

$$\begin{aligned} P(p_1 \geq 1 - \alpha \mid H_0) &\overset{(a)}{=} P(1 - F_1(T_1^2) \geq 1 - \alpha \mid H_0) \\ &= P(F_1(T_1^2) \leq \alpha \mid H_0) \\ &\overset{(b)}{=} P(F_1^{-1}(F_1(T_1^2)) \leq F_1^{-1}(\alpha) \mid H_0) \\ &= P(T_1^2 \leq F_1^{-1}(\alpha) \mid H_0) \\ &\overset{(c)}{=} F_1(F_1^{-1}(\alpha)) \\ &= \alpha, \end{aligned}$$

where F_1^{-1} denotes the inverse of F_1. Here (a) follows from $(\star)$, (b) follows from the monotonicity of F_1^{-1}, and (c) follows since T_1^2 has CDF F_1 under H_0.

b) We include the whole procedure in the following loop:

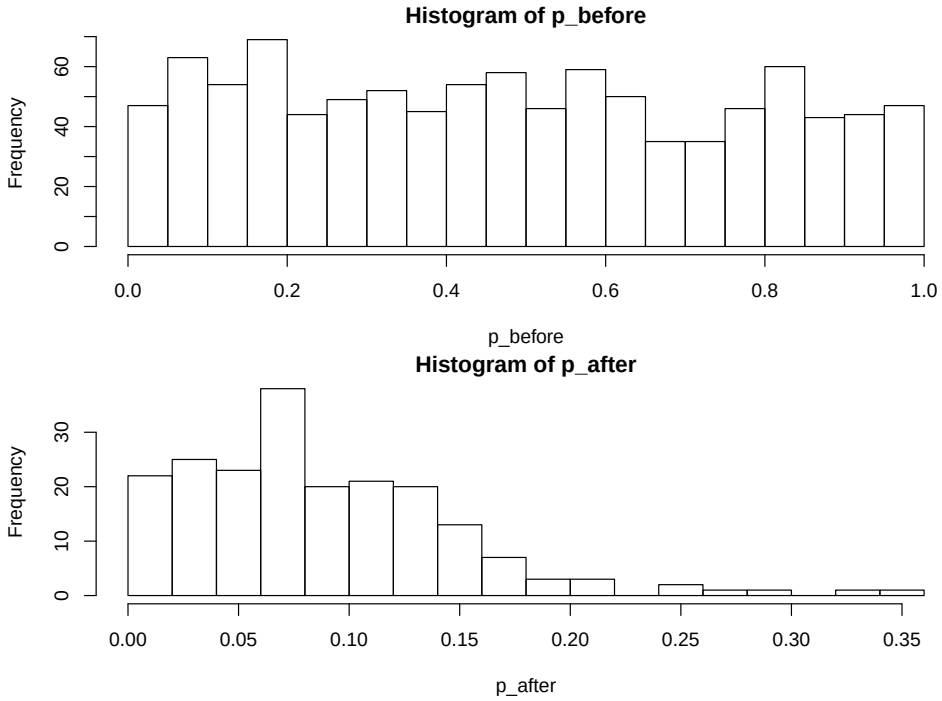
```
> # libs for best subset
> require(leaps)
> # reproducibility
> set.seed(0)
> # store the p-values vectors
> p_after <- c()
> p_before <- c()
> # fix the covariates
> x <- matrix(rnorm(200*10),ncol=10)
> colnames(x) <- paste("X", 1:10, sep = "")
> # loop over replicates
> for (i in 1:1000) {

  # generate the responses
  y <- 2+rnorm(200)
  train <- data.frame(y = y, x)

  # fit the global linear model
  lm_g <- lm(y~., data = train)
  lm_g_s <- summary(lm_g)
  p_before <- c(p_before, lm_g_s$coefficients["X1",4])

  # do model selection with Mallow^s Cp
  m <- leaps::regsubsets(y~., data = train, nvmax = 10)
  best.model <- which.min(summary(m)$cp)

  # retrieve the p-values after model selection for X1 if X1 is
  # included in the model
  form <- as.formula(paste("y~",
```

```
                                paste(names(coef(m, best.model))[-1],
                                      collapse ="+"), sep = ""))

  l <- lm(form, data = train)
  sl <- summary(l)

  # add the p-value if X1 is included in the variable select by the currency best model
  if ("X1" %in% names(coef(m, best.model))[-1]) {
    p_after <-c(p_after, sl$coefficients["X1",4])
  }
 }
```

You have now at hand two samples of p-values, what can you say about their expected behaviour and what do you observe? Discuss your findings.

We can look at summaries and histograma of the two samples of p-values:

```
> # summary of the p-values
> summary(p_before)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000877 0.221750 0.470606 0.479680 0.736989 0.999976
> summary(p_after)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0006598 0.0428515 0.0768162 0.0875600 0.1210299 0.3501947
> # histograms
> par(mfrow=c(2,1))
> hist(p_before, breaks = 20)
> hist(p_after, breaks = 20)
```



Under the global null hypothesis the p-value for the t-test associated to variable x_1 should be uniformly distributed. Indeed, the first histogram (p-values from the general linear model) supports this fact. However, after model selection, this property is no more valid since a prescreening of correlated variables with the response (on the given dataset) has been applied (through the model selection procedure). p-values are then systematically lower in the second case.

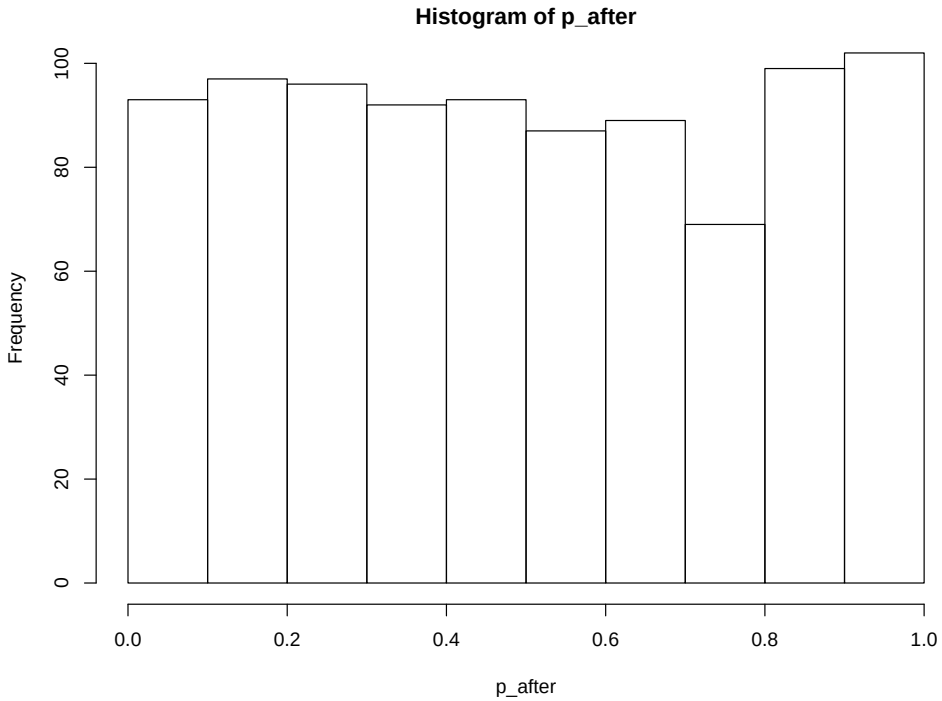
We run again the same analysis, but with sample splitting this time

```
> # reproducibility
> set.seed(0)
> # store the p-values
> p_after <- c()
```

```
> p_before <- c()
> # fix the covariates
> x <- matrix(rnorm(200*10),ncol=10)
> colnames(x) <- paste("X", 1:10, sep = "")
> # loop over replicates
> for (i in 1:5000) {

  # generate the data
  y <- 2 + rnorm(200)

  train <- data.frame(y = y, x)

  # fit the global linear model
  lm_g <- lm(y~., data = train)
  lm_g_s <- summary(lm_g)
  p_before <- c(p_before, lm_g_s$coefficients["X1",4])

  # do model selection with Mallow^s Cp
  m <- leaps::regsubsets(y~., data = train[1:100,], nvmax = 10)
  best.model <- which.min(summary(m)$cp)

  # retrieve the p-values after model selection for X1 if X1 is
  # included in the model
  form <- as.formula(paste("y~",
                           paste(names(coef(m, best.model))[-1],
                                 collapse ="+"), sep = ""))

  l <- lm(form, data = train[101:200,])
  sl <- summary(l)

  # store the p-value of X1 if included in the best model
  if ("X1" %in% names(coef(m, best.model))[-1]) {
    p_after <-c(p_after, sl$coefficients["X1",4])
  }
 }
```

We can look again at summaries and histograms of of p-values:

```
> # summary of the p-values
> summary(p_after)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.001071 0.236061 0.488596 0.495818 0.763253 0.999599
> # histogram
> par(mfrow=c(1,1))
> hist(p_after, breaks = 10)
```



This time you do not observe a difference in p-value distribution, this is due to the fact that when you do best-subset selection, you naturally select variables that show correlation with the response. The p-value computed on the same data used for model selection are then invalid. The sample splitting strategy solves this issue by assessing the p-value on an independent part of the dataset, this is then a direct and correct way to obtain p-values.

3. a) This solution is adapted from unofficial solutions available online at `http://blog.princehonest.com/stat-learning/ch7/9.html`. In the first case that $x \leq \xi$, then $f_1(x)$ should have coefficients

$$\begin{aligned} a_1 &= \beta_0 \\ b_1 &= \beta_1 \\ c_1 &= \beta_2 \\ d_1 &= \beta_3. \end{aligned}$$

b) In the other case $x > \xi$, we can develop the coefficients of $f_1(x)$ to get

$$\begin{aligned} &\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)^3 \\ &= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi^2) x + (\beta_2 - 3\beta_4 \xi) x^2 + (\beta_3 + \beta_4) x^3, \end{aligned}$$

giving us:

$$\begin{aligned} a_2 &= \beta_0 - \beta_4 \xi^3 \\ b_2 &= \beta_1 + 3\beta_4 \xi^2 \\ c_2 &= \beta_2 - \beta_4 \xi \\ d_2 &= \beta_3 + \beta_4. \end{aligned}$$

c) For the continuity of the functions f, it is clear that both f_1 and f_2 are continous on their respective domains. It remains to show that they coincide in ξ. We find by evaluation of f_1 and f_2 at ξ

$$\begin{aligned} f_1(\xi) &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3, \\ f_2(\xi) &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3. \end{aligned}$$

d) Once again it is clear that the derivatives of f_1 and f_2 on their respective domains are defined and continuous. We compute the left and right derivatives of f_1 and f_2 at ξ

$$\begin{aligned} f_1'(\xi) &= \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2, \\ f_2'(\xi) &= \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2, \end{aligned}$$

These values coincide, so that the derivative of f at ξ is continuous.

e) Similarly, for the left and right second derivative of the functions f at ξ we find

$$\begin{aligned} f_1''(\xi) &= 2\beta_2 + 6\beta_3\xi, \\ f_2''(\xi) &= 2\beta_2 + 6\beta_3\xi, \end{aligned}$$

So that the second derivative of f at ξ is also continuous.

4. This solution is adapted from the unofficial solutions available online `http://blog.princehonest.com/stat-learning/ch7/9.html`. First we load the dataset and the needed libraries into memory:

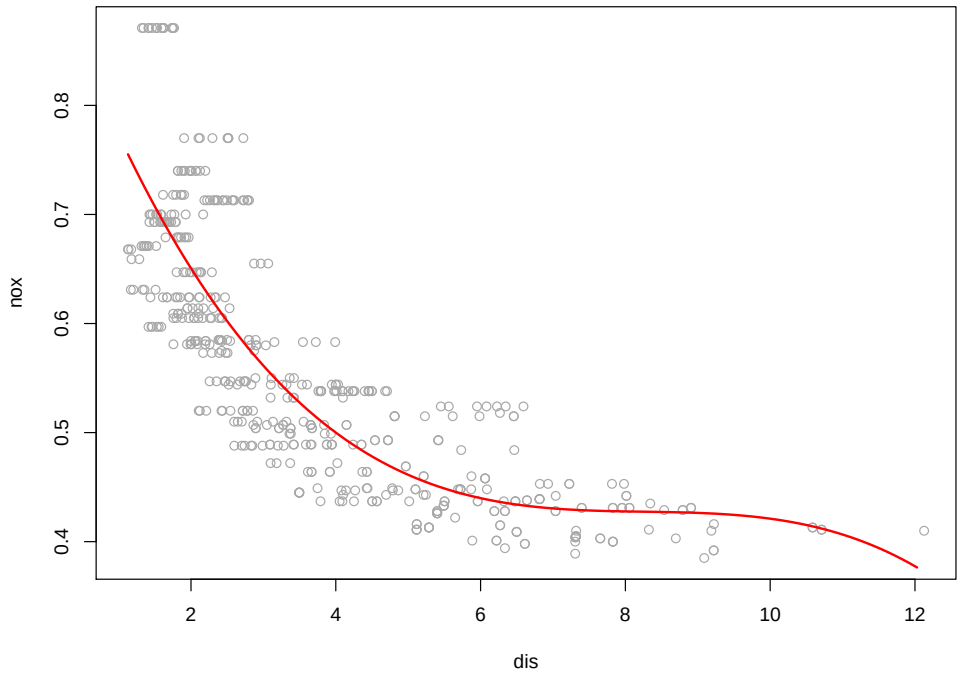
```
> set.seed(1)
> library(MASS)
> attach(Boston)
> library(boot)
> library(splines)
```

a) here we fit a polynomial linear model of degree 3 to the data

```
> # fit the linear model with a polynomial of degree three in dis
> lm.fit = lm(nox ~ poly(dis, 3), data = Boston)
> # summary of the model
> summary(lm.fit)

Call:
lm(formula = nox ~ poly(dis, 3), data = Boston)

Residuals:
      Min        1Q    Median        3Q       Max
-0.121130 -0.040619 -0.009738  0.023385  0.194904

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06207 on 502 degrees of freedom
Multiple R-squared:  0.7148,        Adjusted R-squared:  0.7131
F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

and we plot the fitted curves

```
> # we plot on a specific range for dis
> dislim = range(dis)
> dis.grid = seq(from = dislim[1], to = dislim[2], by = 0.1)
> # get the predictions
> lm.pred = predict(lm.fit, list(dis = dis.grid))
> # plot the observation and fitted line
> plot(nox ~ dis, data = Boston, col = "darkgrey")
> lines(dis.grid, lm.pred, col = "red", lwd = 2)
```



b) We fit now polynomial models with increasing degrees

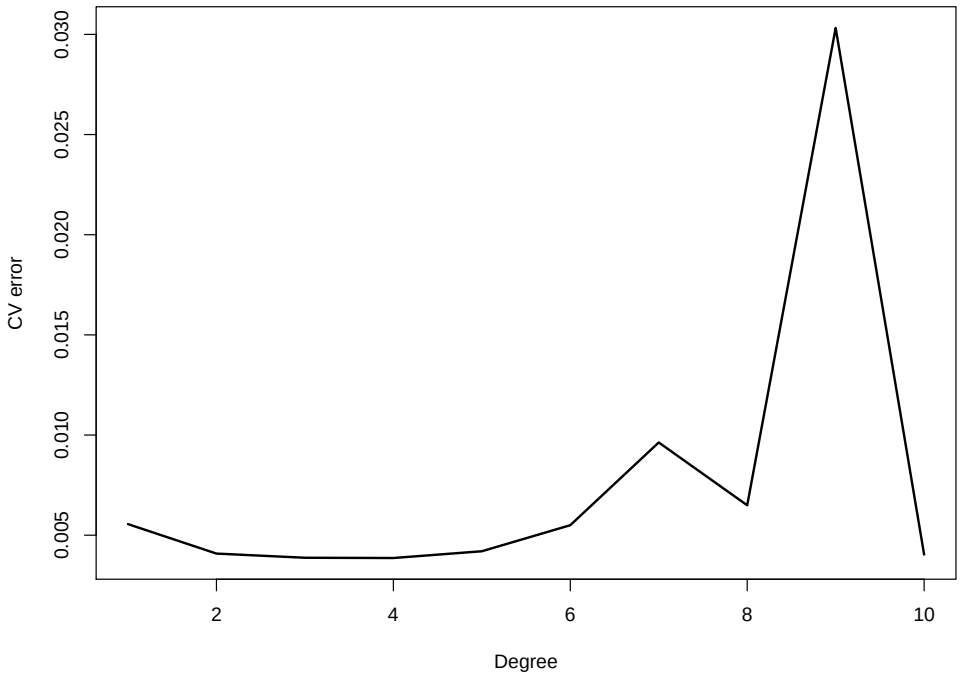
```
> # store the residual sum of squared
> all.rss = rep(NA, 10)
> # loop over degree
> for (i in 1:10) {
      lm.fit = lm(nox ~ poly(dis, i), data = Boston)
      all.rss[i] = sum(lm.fit$residuals^2)
  }
> # print the results
> print(all.rss)
```

```
 [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257
 [7] 1.849484 1.835630 1.833331 1.832171
```

You can observe that the training error decreeases when the degree of the polynomial increases, which is envident since this increases the number of parameters and we have seen that the training error can only go down for multiple linear regression in this setting.

c) We can select the best polynomial fit (out of the ones fitted in the last point) by cross-validation using the `boot` R-package (allowing to perform cross-validation). Please note that we use the function `glm(...)` to fit a linear model here due to the existence of the function `cv.glm(...)` which allows for automatic cross-validation

```
> # load boot package
> library(boot)
> # store the cross-validation errors
> all.deltas = rep(NA, 10)
> # loop over degree
> for (i in 1:10) {
      glm.fit = glm(nox ~ poly(dis, i), data = Boston)
      all.deltas[i] = cv.glm(Boston, glm.fit, K = 10)$delta[2]
  }
> # plot the resulting cross validation error in function of the degree
> plot(1:10, all.deltas, xlab = "Degree", ylab = "CV error", type = "l", pch = 20,
      lwd = 2)
```



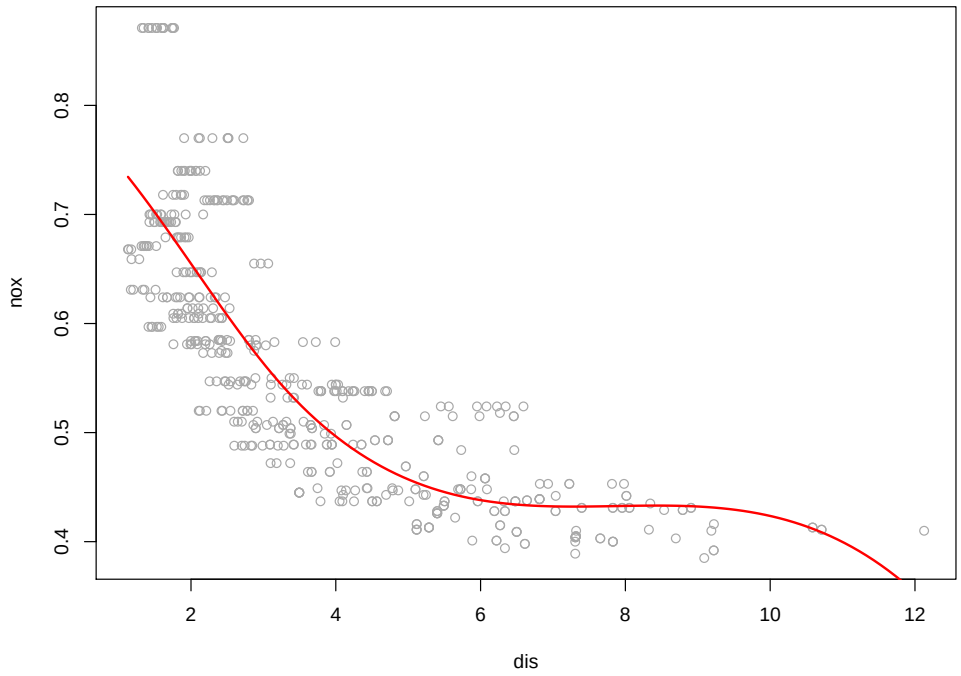
The degree 4 is picked as the best polynomial degree according to the cross-validation scores.

d) In the function `bs()`, knots are automatically placed at empirical quantiles of `nox` according to the chosen degree of freedom

```
> sp.fit = lm(nox ~ bs(dis, df = 4), data = Boston)
> summary(sp.fit)

Call:
lm(formula = nox ~ bs(dis, df = 4), data = Boston)

Residuals:
      Min        1Q    Median        3Q       Max
-0.124622 -0.039259 -0.008514  0.020850  0.193891

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       0.73447    0.01460  50.306  < 2e-16 ***
bs(dis, df = 4)1 -0.05810    0.02186  -2.658  0.00812 **
bs(dis, df = 4)2 -0.46356    0.02366 -19.596  < 2e-16 ***
bs(dis, df = 4)3 -0.19979    0.04311  -4.634 4.58e-06 ***
bs(dis, df = 4)4 -0.38881    0.04551  -8.544  < 2e-16 ***
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.06195 on 501 degrees of freedom
Multiple R-squared:  0.7164,        Adjusted R-squared:  0.7142
F-statistic: 316.5 on 4 and 501 DF,  p-value: < 2.2e-16
```

we can plot the fitted spline

```
> # predict
> sp.pred = predict(sp.fit, list(dis = dis.grid))
> # plot the data and fitted spline
> plot(nox ~ dis, data = Boston, col = "darkgrey")
> lines(dis.grid, sp.pred, col = "red", lwd = 2)
```



e) We fit regression splines with degrees of freedom between 3 and 16 and look at the training MSE.

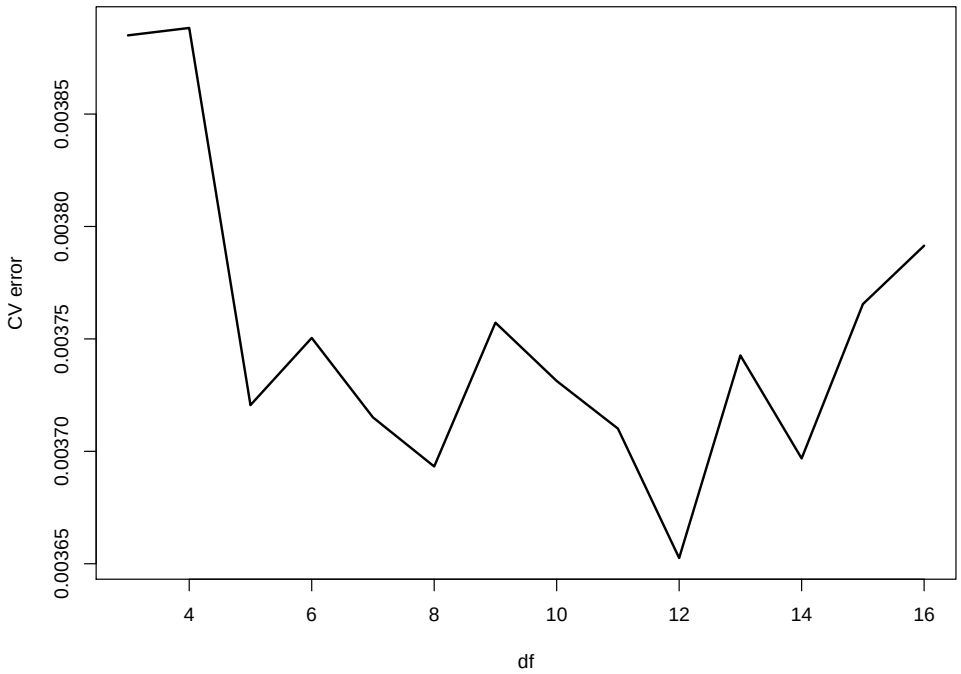
```
> # store the RSS
> all.rss = rep(NA, 16)
> # loop over degree of freedom
> for (i in 3:16) {
      bs.fit = lm(nox ~ bs(dis, df = i), data = Boston)
      all.rss[i] = sum(bs.fit$residuals^2)
 }
> # RSS values
> all.rss[-c(1, 2)]
```

```
 [1] 1.934107 1.922775 1.840173 1.833966 1.829884 1.816995
 [7] 1.825653 1.792535 1.796992 1.788999 1.782350 1.781838
[13] 1.782798 1.783546
```

This time you observe that the training error is not monotonically decreasing. Indeed it is not always the case that a spline with higher degree of freedom will make systematically the training error lower.

f) We finally perform 10-fold cross-validation to find the best degree of freedom for a spline in `dis`, we try all values between 3 and 16 and plot the cross-validation error

```
> # store the cross-validation errors
> all.cv = rep(NA, 16)
> # loop of degree of freedom
> for (i in 3:16) {
      lm.fit = glm(nox ~ bs(dis, df = i), data = Boston)
      all.cv[i] = cv.glm(Boston, lm.fit, K = 10)$delta[2]
 }
> # plot the resulting cross-validation scores
> plot(3:16, all.cv[-c(1, 2)], lwd = 2, type = "l", xlab = "df", ylab = "CV error")
```



According to the cross-validation scores, we should pick $df = 10$ as the best degreee of freedom.