Chapter 6 Conceptual:

1. **We perform best subset, forward stepwise, and backward stepwise selection on a single data set. For each approach, we obtain p + 1 models, containing 0, 1, 2..., p predictors. Explain your answers:**

   a. **Which of the three models with k predictors has the smallest training RSS?**

      The other two techniques determine models with a path dependency on the predictors they choose initially when they iterate to the k'th model, which results in best subset selection having the shortest training RSS.

   b. **Which of the three models with k predictors has the smallest test RSS?**

      Because it considers more models than the other techniques, best subset selection could have the smallest test RSS. The other models might have a higher chance of choosing a model that more closely matches the test data.

   c. i) True, ii) True, iii) False, iv) False, iv) False.

2. **For parts (a) through (c), indicate which of i. through iv. is correct. Justify your answer.**

   a. iii. Less flexible and better predictions because of less variance, more bias
   b. iii. Same as Lasso
   c. ii. More flexible, less bias, more variance

R Notebook

8. In this exercise, we will generate simulated data, and will then use this data to perform best subset selection.

(a) Use the rnorm() function to generate a predictor X of length n = 100, as well as a noise vector ϵ of length n = 100.

```
set.seed(1)
X = rnorm(100)
eps = rnorm(100)
```

b. Generate a response vector Y of length n = 100 according to the model Y = β0 + β1X + β2X2 + β3X3 + ϵ, where β0, β1, β2, and β3 are constants of your choice.

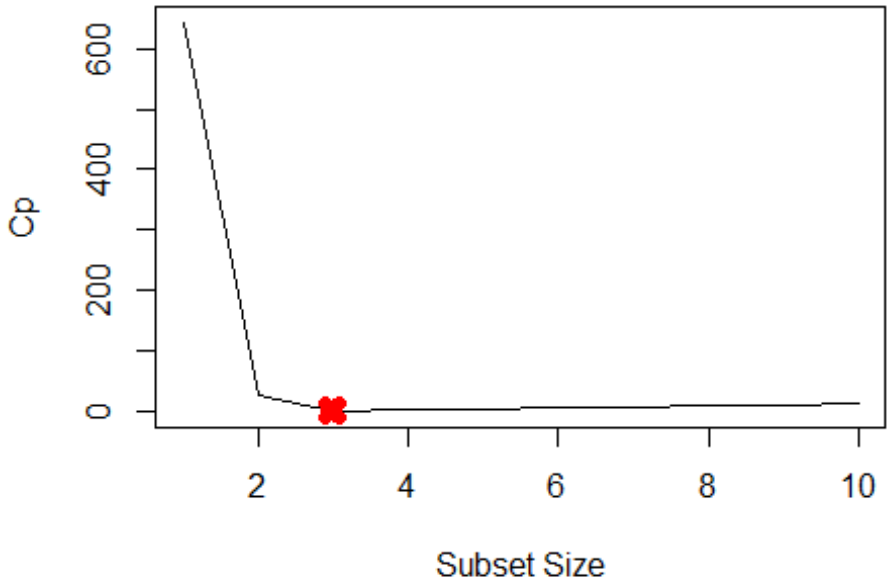
By selecting β0 = 3, β1 = 2, β2 = -3, and β3 = 0.3

```
beta0 = 3
beta1 = 2
beta2 = -3
beta3 = 0.3
Y = beta0 + beta1 * X + beta2 * X^2 + beta3 * X^3 + eps
```

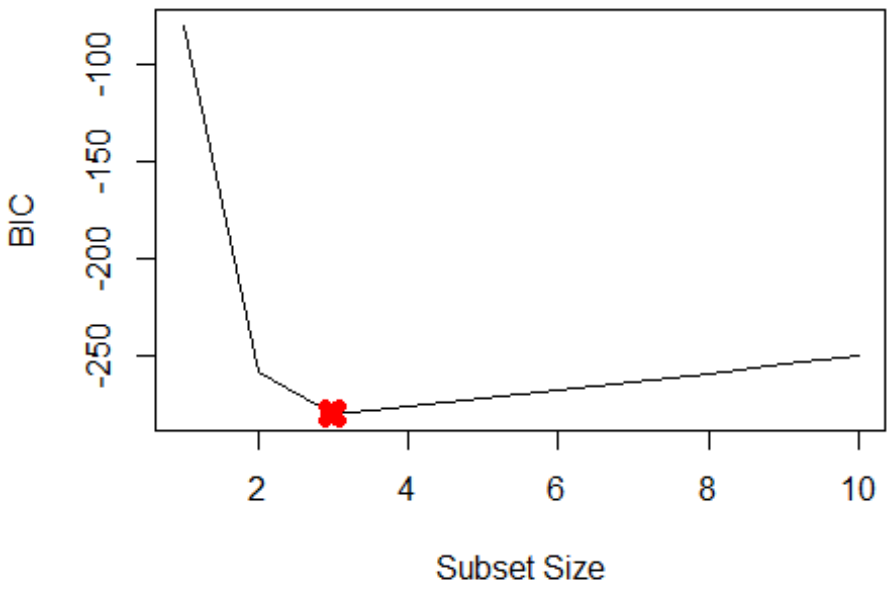
c. Use the reg subsets() function to perform best subset selection in order to choose the best model containing the predictors X, X2,...,X10. What is the best model obtained according to Cp, BIC, and adjusted R2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained. Note you will need to use the data.frame() function to create a single data set containing both X and Y .

```
install.Packages("leaps")
library(leaps)

## Warning: package 'leaps' was built under R version 4.2.1

data.full = data.frame(y = Y, x = X)
mod.full = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10)
mod.summary = summary(mod.full)

# Find the model size for best cp, BIC and adjr2
which.min(mod.summary$cp)

## [1] 3

which.min(mod.summary$bic)

## [1] 3

which.max(mod.summary$adjr2)

## [1] 3
```

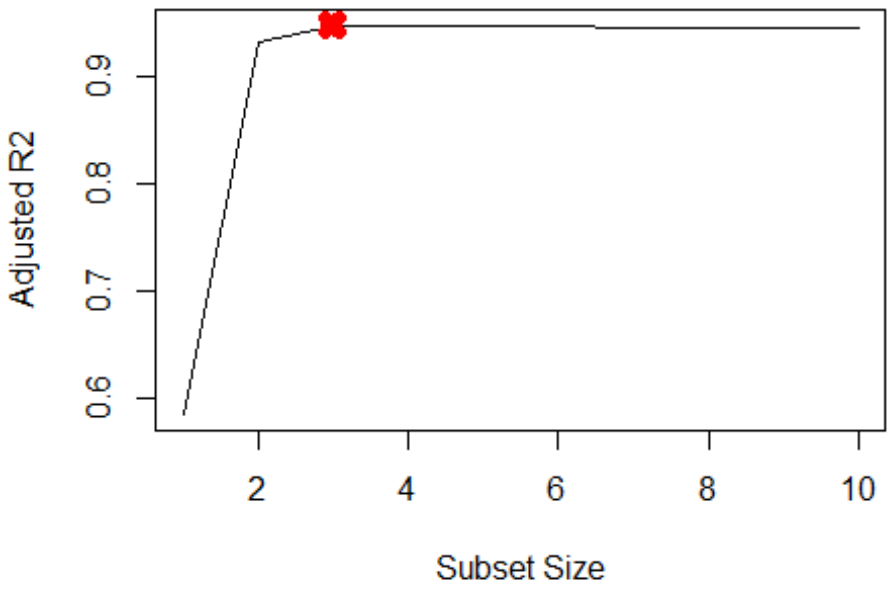
```r
# Plot cp, BIC and adjr2
plot(mod.summary$cp, xlab = "Subset Size", ylab = "Cp", pch = 20, type = "l")
points(3, mod.summary$cp[3], pch = 4, col = "red", lwd = 7)
```




```r
plot(mod.summary$bic, xlab = "Subset Size", ylab = "BIC", pch = 20, type =
"l")
points(3, mod.summary$bic[3], pch = 4, col = "red", lwd = 7)
```




```
plot(mod.summary$adjr2, xlab = "Subset Size", ylab = "Adjusted R2", pch = 20,
    type = "l")
points(3, mod.summary$adjr2[3], pch = 4, col = "red", lwd = 7)
```



We find that with Cp, BIC and Adjusted R2 criteria, 3, 3, and 3 variable models are respectively picked.

```
coefficients(mod.full, id = 3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##            3.07627412           2.35623596          -3.16514887
## poly(x, 10, raw = T)7
##            0.01046843
```

All statistics pick X7 over X3. The remaining coefficients are quite close to β s.

d. Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?

```
mod.fwd = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10,
    method = "forward")
mod.bwd = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10,
    method = "backward")
fwd.summary = summary(mod.fwd)
bwd.summary = summary(mod.bwd)
which.min(fwd.summary$cp)
```

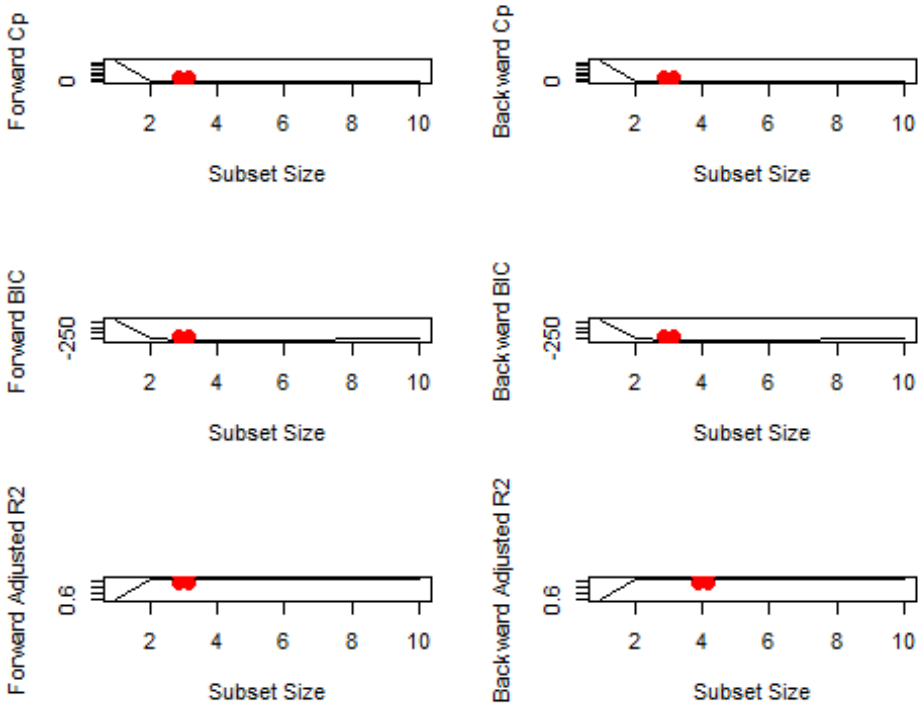
```
## [1] 3
```

```
which.min(bwd.summary$cp)
```

```
## [1] 3
```

```
which.min(fwd.summary$bic)
```

```
## [1] 3

which.min(bwd.summary$bic)

## [1] 3

which.max(fwd.summary$adjr2)

## [1] 3

which.max(bwd.summary$adjr2)

## [1] 3

# Plot the statistics
par(mfrow = c(3, 2))
plot(fwd.summary$cp, xlab = "Subset Size", ylab = "Forward Cp", pch = 20,
type = "l")
points(3, fwd.summary$cp[3], pch = 4, col = "red", lwd = 7)
plot(bwd.summary$cp, xlab = "Subset Size", ylab = "Backward Cp", pch = 20,
type = "l")
points(3, bwd.summary$cp[3], pch = 4, col = "red", lwd = 7)
plot(fwd.summary$bic, xlab = "Subset Size", ylab = "Forward BIC", pch = 20,
    type = "l")
points(3, fwd.summary$bic[3], pch = 4, col = "red", lwd = 7)
plot(bwd.summary$bic, xlab = "Subset Size", ylab = "Backward BIC", pch = 20,
    type = "l")
points(3, bwd.summary$bic[3], pch = 4, col = "red", lwd = 7)
plot(fwd.summary$adjr2, xlab = "Subset Size", ylab = "Forward Adjusted R2",
    pch = 20, type = "l")
points(3, fwd.summary$adjr2[3], pch = 4, col = "red", lwd = 7)
plot(bwd.summary$adjr2, xlab = "Subset Size", ylab = "Backward Adjusted R2",
    pch = 20, type = "l")
points(4, bwd.summary$adjr2[4], pch = 4, col = "red", lwd = 7)
```



all statistics pick 3 variable models except backward stepwise with adjusted R2. Here are the coefficients

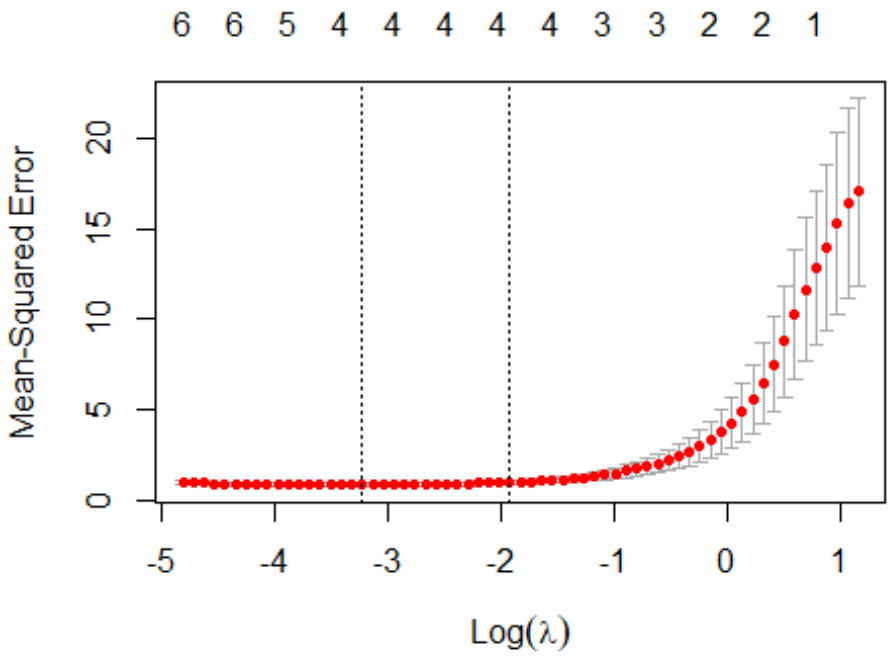
```
coefficients(mod.fwd, id = 3)

##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##            3.07627412           2.35623596          -3.16514887
## poly(x, 10, raw = T)7
##            0.01046843

coefficients(mod.bwd, id = 3)

##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.078881355          2.419817953          -3.177235617
## poly(x, 10, raw = T)9
##           0.001870457

coefficients(mod.fwd, id = 4)

##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.112358625          2.369858879          -3.275726574
## poly(x, 10, raw = T)4 poly(x, 10, raw = T)7
##           0.027673638          0.009997134
```

Here, X7 is chosen over X3 by forward stepwise. While backward stepwise with four variables selects X4 and X7, backward stepwise with three variables selects X9. Near s, all other coefficients are.

e. Now fit a lasso model to the simulated data, again using X, X2, …,X10 as predictors. Use cross-validation to select the optimal value of λ. Create plots of the cross-validation error as a function of λ. Report the resulting coefficient estimates, and discuss the results obtained.

Training Lasso on the data

```
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.2.1

## Loading required package: Matrix

## Loaded glmnet 4.1-4

xmat = model.matrix(y ~ poly(x, 10, raw = T), data = data.full)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha = 1)
best.lambda = mod.lasso$lambda.min
best.lambda

## [1] 0.03991416

plot(mod.lasso)
```




```
# Next fit the model on entire data using best lambda
best.model = glmnet(xmat, Y, alpha = 1)
predict(best.model, s = best.lambda, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                      s1
## (Intercept)             3.0398151056
## poly(x, 10, raw = T)1   2.2303371338
## poly(x, 10, raw = T)2  -3.1033192679
## poly(x, 10, raw = T)3   .
## poly(x, 10, raw = T)4   .
## poly(x, 10, raw = T)5   0.0498410763
## poly(x, 10, raw = T)6   .
## poly(x, 10, raw = T)7   0.0008068431
## poly(x, 10, raw = T)8   .
## poly(x, 10, raw = T)9   .
## poly(x, 10, raw = T)10  .
```

Lasso also picks X5 over X3. It also picks X7 with negligible coefficient.

f. Now generate a response vector Y according to the model Y = β0 + β7X7 + ϵ, and perform best subset selection and the lasso. Discuss the results obtained.

Create new Y with different β7=7.

```
beta7 = 7
Y = beta0 + beta7 * X^7 + eps
# Predict using regsubsets
data.full = data.frame(y = Y, x = X)
mod.full = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10)
mod.summary = summary(mod.full)

# Find the model size for best cp, BIC and adjr2
which.min(mod.summary$cp)

## [1] 2

which.min(mod.summary$bic)

## [1] 1

which.max(mod.summary$adjr2)

## [1] 4

coefficients(mod.full, id = 1)

##           (Intercept) poly(x, 10, raw = T)7
##               2.95894               7.00077

coefficients(mod.full, id = 2)

##           (Intercept) poly(x, 10, raw = T)2 poly(x, 10, raw = T)7
##             3.0704904            -0.1417084             7.0015552

coefficients(mod.full, id = 4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##             3.0762524             0.2914016            -0.1617671
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)7
##            -0.2526527             7.0091338
```

```
xmat = model.matrix(y ~ poly(x, 10, raw = T), data = data.full)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha = 1)
best.lambda = mod.lasso$lambda.min
best.lambda
```

```
## [1] 12.36884
```

```
best.model = glmnet(xmat, Y, alpha = 1)
predict(best.model, s = best.lambda, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)            3.820215
## poly(x, 10, raw = T)1  .
## poly(x, 10, raw = T)2  .
## poly(x, 10, raw = T)3  .
## poly(x, 10, raw = T)4  .
## poly(x, 10, raw = T)5  .
## poly(x, 10, raw = T)6  .
## poly(x, 10, raw = T)7  6.796694
## poly(x, 10, raw = T)8  .
## poly(x, 10, raw = T)9  .
## poly(x, 10, raw = T)10 .
```

Lasso also picks the best 1-variable model but intercet is quite off (3.8 vs 3).

9. In this exercise, we will predict the number of applications received using the other variables in the College data set.

a. Split the data set into a training set and a test set.

```
library(ISLR)
set.seed(11)
sum(is.na(College))
```

```
## [1] 0
```

```
train.size = dim(College)[1] / 2
train = sample(1:dim(College)[1], train.size)
test = -train
College.train = College[train, ]
College.test = College[test, ]
```

b. Fit a linear model using least squares on the training set, and report the test error obtained.

```
lm.fit = lm(Apps~., data=College.train)
lm.pred = predict(lm.fit, College.test)
mean((College.test[, "Apps"] - lm.pred)^2)
```

```
## [1] 1026096
```

Test RSS is 1026096.

c. Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```
library(glmnet)
```

```
train.mat = model.matrix(Apps~., data=College.train)
test.mat = model.matrix(Apps~., data=College.test)
grid = 10 ^ seq(4, -2, length=100)
mod.ridge = cv.glmnet(train.mat, College.train[, "Apps"], alpha=0,
lambda=grid, thresh=1e-12)
lambda.best = mod.ridge$lambda.min
lambda.best
```

```
## [1] 0.01
```

```
ridge.pred = predict(mod.ridge, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - ridge.pred)^2)
```

```
## [1] 1026069
```

Test RSS is almost equal to OLS, 1026069.

d. Fit a lasso model on the training set, with λ chosen by crossvalidation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
mod.lasso = cv.glmnet(train.mat, College.train[, "Apps"], alpha=1,
lambda=grid, thresh=1e-12)
lambda.best = mod.lasso$lambda.min
lambda.best

## [1] 0.01

lasso.pred = predict(mod.lasso, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - lasso.pred)^2)

## [1] 1026036
```

Test RSS is almost equal to OLS, 1026036.

```
mod.lasso = glmnet(model.matrix(Apps~., data=College), College[, "Apps"],
alpha=1)
predict(mod.lasso, s=lambda.best, type="coefficients")

## 19 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -471.39372052
## (Intercept)    .
## PrivateYes  -491.04485137
## Accept         1.57033288
## Enroll        -0.75961467
## Top10perc     48.14698892
## Top25perc    -12.84690695
## F.Undergrad    0.04149116
## P.Undergrad    0.04438973
## Outstate      -0.08328388
## Room.Board     0.14943472
## Books          0.01532293
## Personal       0.02909954
## PhD           -8.39597537
## Terminal      -3.26800340
## S.F.Ratio     14.59298267
## perc.alumni   -0.04404771
## Expend         0.07712632
## Grad.Rate      8.28950241
```

Chapter 7 Conceptual:

4. Suppose we fit a curve with basis functions b1(X) = I (0 ≤ X ≤ 2) − (X −1)I(1 ≤ X ≤ 2), b2(X)=(X −3)I(3 ≤ X ≤ 4) +I(4 < X ≤ 5). We fit the linear regression model.

$$Y = \beta 0 + \beta 1 b1(X) + \beta 2 b2(X) + \epsilon,$$

and obtain coefficient estimates βˆ0 = 1, βˆ1 = 1, βˆ2 = 3. Sketch the estimated curve between X = −2 and X = 6. Note the intercepts, slopes, and other relevant information.

We obtain the fitted model by plugging in these basis functions and coefficient estimates:

$$\hat{f}(X)=1+I(0\le X\le 2)-(X-1)I(1\le X\le 2)+3(X-3)I(3\le X\le 4)+3I(4<X\le 5)$$

We can simplify f^(X) throughout this range since we are only interested in the function over the range [2,2]:

$$\hat{f}(X)=1+I(0\le X\le 2)-(X-1)I(1\le X\le 2)\hat{f}(X)=1+I(0\le X\le 2)-(X-1)I(1\le X\le 2)$$

```r
install.packages("ggplot2")
library(ggplot2)

X = seq(-2, 2, 0.01)
Y = 1 + (X >= 0 & X <= 2) - (X - 1)*(X >= 1 & X <= 2) + 3*(X - 3)*(X >= 3 & X <= 4) + 3*(X > 4 & X <= 5)
df <- data.frame(X, Y)

ggplot(df, aes(x = X, y = Y)) +
  geom_vline(xintercept = 0) +
  geom_hline(yintercept = 0) +
  geom_line(size = 1.5)
```

For X<0X<0 none of the indicator variables are true, so f^(X)=1f^(X)=1 with a slope of 00.

For 0≤X<10≤X<1, the first indicator variable is true, so f^(X)=1+1=2f^(X)=1+1=2, so this is the intercept with a slope of 00.

For 1≤X≤21≤X≤2, both indicator variables are true, so f^(X)=1+1−(X−1)·1=3−Xf^(X)=1+1−(X−1)·1=3−X, so the slope is −1−1 here.

5.

a. Given that g^2 will be a higher order polynomial due to the order of the derivative penalty function, we would anticipate that g2 will have the smaller training RSS.

b. Because g^2 might be overfit with the additional degree of freedom, we would anticipate that g^1 would have the smaller test RSS.

c. when $\lambda=0$, g^1 = g^2.

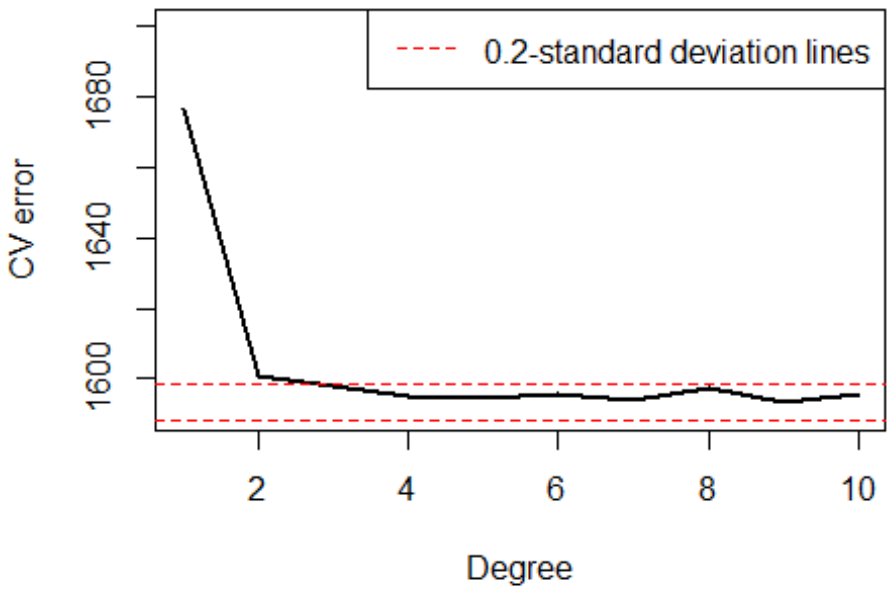
R Notebook

6. In this exercise, you will further analyze the Wage data set considered throughout this chapter.

a. Perform polynomial regression to predict wage using age. Use cross-validation to select the optimal degree d for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data.

Load Wage dataset. Keep an array of all cross-validation errors. We are performing K-fold cross validation with K=10.

```
set.seed(1)
library(ISLR)
library(boot)
all.deltas = rep(NA, 10)
for (i in 1:10) {
  glm.fit = glm(wage~poly(age, i), data=Wage)
  all.deltas[i] = cv.glm(Wage, glm.fit, K=10)$delta[2]
}
plot(1:10, all.deltas, xlab="Degree", ylab="CV error", type="l", pch=20,
lwd=2, ylim=c(1590, 1700))
min.point = min(all.deltas)
sd.points = sd(all.deltas)
abline(h=min.point + 0.2 * sd.points, col="red", lty="dashed")
abline(h=min.point - 0.2 * sd.points, col="red", lty="dashed")
legend("topright", "0.2-standard deviation lines", lty="dashed", col="red")
```



he cv-plot with standard deviation lines show that d=3 is the smallest degree giving reasonably small cross-validation error.

We now find best degree using Anova.

```
fit.1 = lm(wage~poly(age, 1), data=Wage)
fit.2 = lm(wage~poly(age, 2), data=Wage)
fit.3 = lm(wage~poly(age, 3), data=Wage)
fit.4 = lm(wage~poly(age, 4), data=Wage)
fit.5 = lm(wage~poly(age, 5), data=Wage)
fit.6 = lm(wage~poly(age, 6), data=Wage)
fit.7 = lm(wage~poly(age, 7), data=Wage)
fit.8 = lm(wage~poly(age, 8), data=Wage)
fit.9 = lm(wage~poly(age, 9), data=Wage)
fit.10 = lm(wage~poly(age, 10), data=Wage)
anova(fit.1, fit.2, fit.3, fit.4, fit.5, fit.6, fit.7, fit.8, fit.9, fit.10)
```

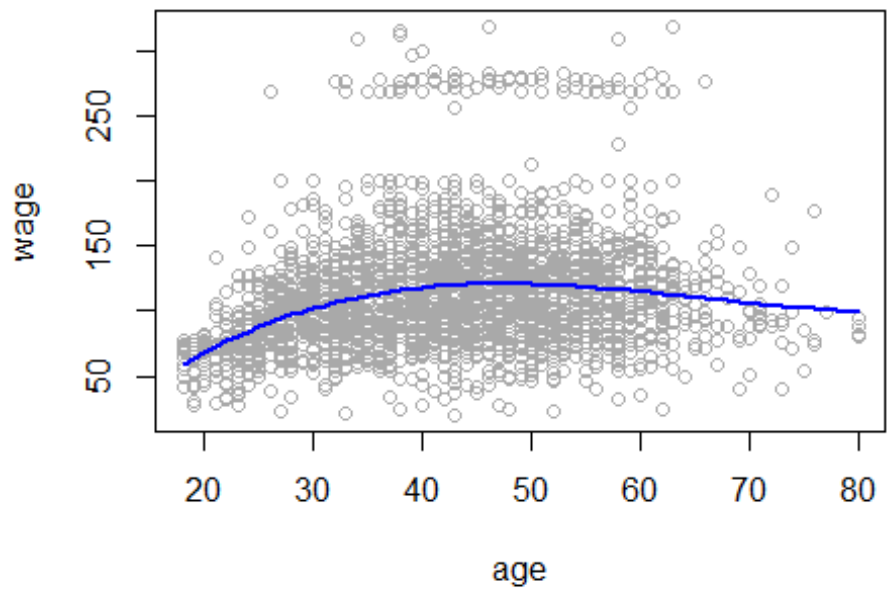
```
## Analysis of Variance Table
##
## Model  1: wage ~ poly(age, 1)
## Model  2: wage ~ poly(age, 2)
## Model  3: wage ~ poly(age, 3)
## Model  4: wage ~ poly(age, 4)
## Model  5: wage ~ poly(age, 5)
## Model  6: wage ~ poly(age, 6)
## Model  7: wage ~ poly(age, 7)
## Model  8: wage ~ poly(age, 8)
## Model  9: wage ~ poly(age, 9)
```

```
## Model 10: wage ~ poly(age, 10)
##    Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1    2998 5022216
## 2    2997 4793430  1    228786 143.7638 < 2.2e-16 ***
## 3    2996 4777674  1     15756   9.9005  0.001669 **
## 4    2995 4771604  1      6070   3.8143  0.050909 .
## 5    2994 4770322  1      1283   0.8059  0.369398
## 6    2993 4766389  1      3932   2.4709  0.116074
## 7    2992 4763834  1      2555   1.6057  0.205199
## 8    2991 4763707  1       127   0.0796  0.777865
## 9    2990 4756703  1      7004   4.4014  0.035994 *
## 10   2989 4756701  1         3   0.0017  0.967529
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Anova shows that all polynomials above degree 3 are insignificant at 1 significance level.

We now plot the polynomial prediction on the data

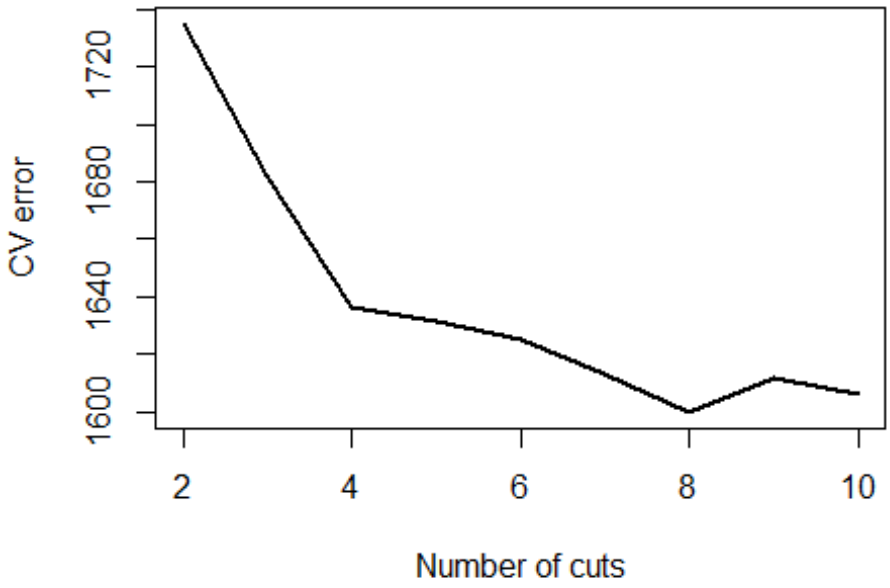
```
plot(wage~age, data=Wage, col="darkgrey")
agelims = range(Wage$age)
age.grid = seq(from=agelims[1], to=agelims[2])
lm.fit = lm(wage~poly(age, 3), data=Wage)
lm.pred = predict(lm.fit, data.frame(age=age.grid))
lines(age.grid, lm.pred, col="blue", lwd=2)
```



b. Fit a step function to predict wage using age, and perform crossvalidation to choose the optimal number of cuts. Make a plot of the fit obtained.

We use cut points of up to 10.

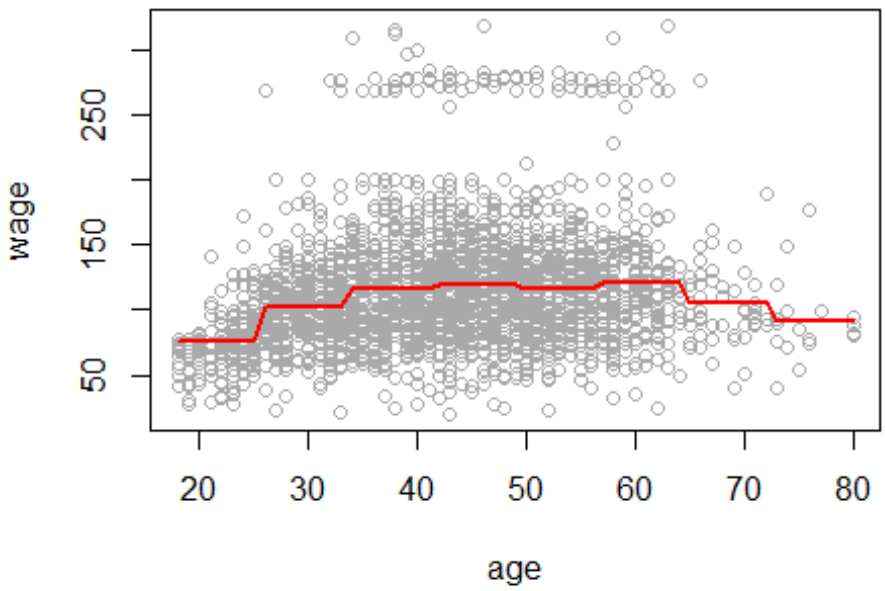
```
all.cvs = rep(NA, 10)
for (i in 2:10) {
  Wage$age.cut = cut(Wage$age, i)
  lm.fit = glm(wage~age.cut, data=Wage)
  all.cvs[i] = cv.glm(Wage, lm.fit, K=10)$delta[2]
}
plot(2:10, all.cvs[-1], xlab="Number of cuts", ylab="CV error", type="l",
pch=20, lwd=2)
```



The cross validation shows that test error is minimum for k=8 cuts.

We now train the entire data with step function using 8 cuts and plot it.

```
lm.fit = glm(wage~cut(age, 8), data=Wage)
agelims = range(Wage$age)
age.grid = seq(from=agelims[1], to=agelims[2])
lm.pred = predict(lm.fit, data.frame(age=age.grid))
plot(wage~age, data=Wage, col="darkgrey")
lines(age.grid, lm.pred, col="red", lwd=2)
```

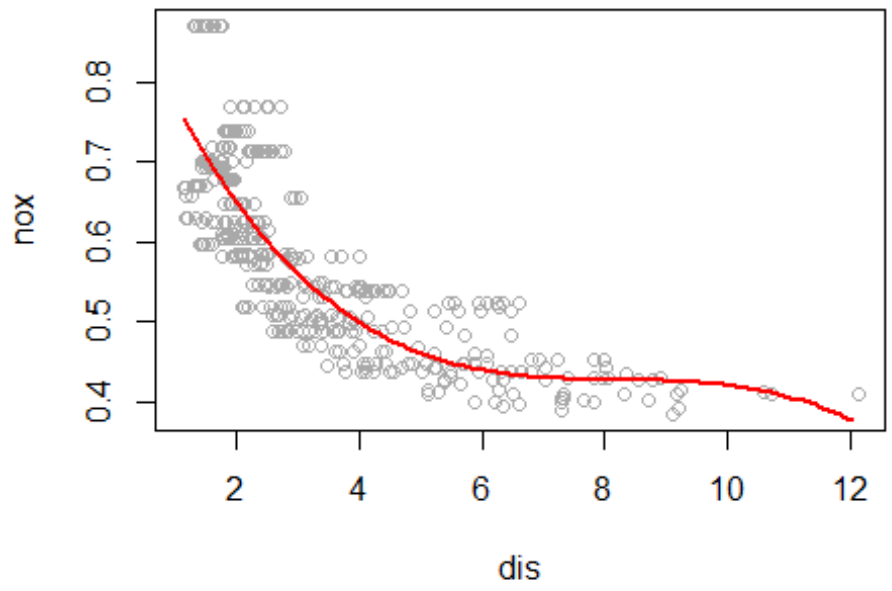

wage
50
150
250
20
30
40
50
60
70
80
age

9. This question uses the variables dis (the weighted mean of distances to five Boston employment centers) and nox (nitrogen oxides concentration in parts per 10 million) from the Boston data. We will treat dis as the predictor and nox as the response.

```
set.seed(1)
library(MASS)
attach(Boston)
```

a. Use the poly() function to fit a cubic polynomial regression to predict nox using dis. Report the regression output, and plot the resulting data and polynomial fits.

```
lm.fit = lm(nox ~ poly(dis, 3), data = Boston)
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

dislim = range(dis)
dis.grid = seq(from = dislim[1], to = dislim[2], by = 0.1)
lm.pred = predict(lm.fit, list(dis = dis.grid))
plot(nox ~ dis, data = Boston, col = "darkgrey")
lines(dis.grid, lm.pred, col = "red", lwd = 2)
```



Summary demonstrates that when forecasting nox with dis, all polynomial terms are meaningful. The plot displays a smooth curve that reasonably fits the data.

b. Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

We plot polynomials of degrees 1 to 10 and save train RSS.

```
all.rss = rep(NA, 10)
for (i in 1:10) {
    lm.fit = lm(nox ~ poly(dis, i), data = Boston)
    all.rss[i] = sum(lm.fit$residuals^2)
}
all.rss
```

```
##  [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484
1.835630
##  [9] 1.833331 1.832171
```

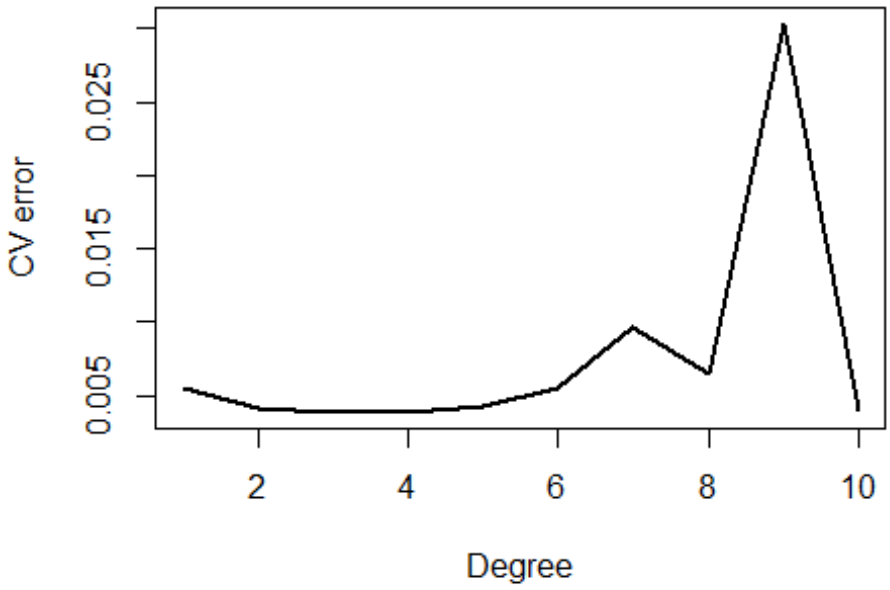
As expected, train RSS monotonically decreases with degree of polynomial.

c. Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

We use a 10-fold cross validation to pick the best polynomial degree

```
library(boot)
all.deltas = rep(NA, 10)
```

```
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(1:10, all.deltas, xlab = "Degree", ylab = "CV error", type = "l", pch =
20,
    lwd = 2)
```



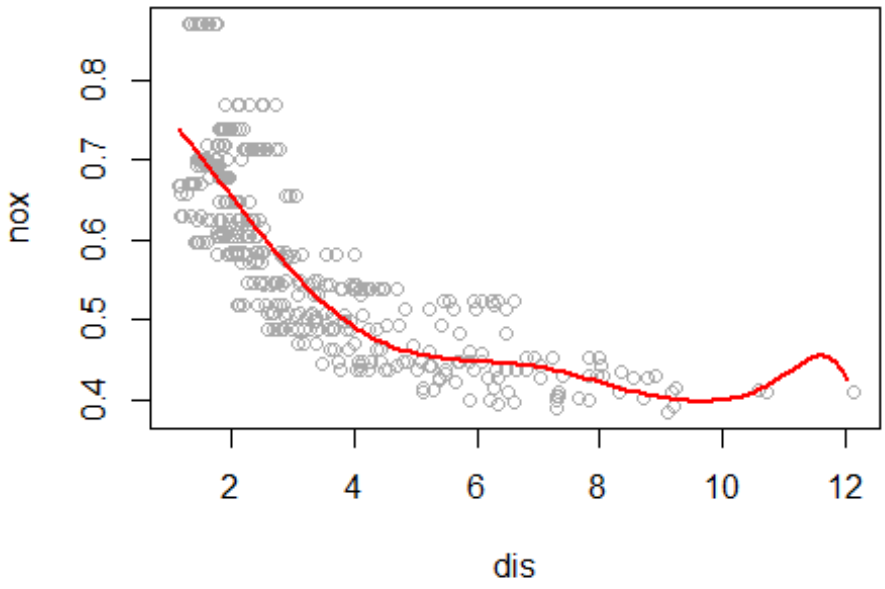
A 10-fold CV reveals that the CV error decreases from degree 1 to degree 3, remains about constant until degree 5, and then increases for higher degrees. The best polynomial degree, in our opinion, is 4.

d. Use the bs() function to fit a regression spline to predict nox using dis. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

We can see that dis has roughly 1 and 13 limitations, respectively. We divide this range into four relatively equal portions, and we create knots at [4,7,11]. Recall that the R bs function requires either a df or knots argument. Knots are disregarded if both are specified.

```
library(splines)
sp.fit = lm(nox ~ bs(dis, df = 4, knots = c(4, 7, 11)), data = Boston)
summary(sp.fit)

##
## Call:
## lm(formula = nox ~ bs(dis, df = 4, knots = c(4, 7, 11)), data = Boston)
##
```

```
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124567 -0.040355 -0.008702  0.024740  0.192920
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                           0.73926    0.01331  55.537  < 2e-16
***
## bs(dis, df = 4, knots = c(4, 7, 11))1 -0.08861    0.02504  -3.539  0.00044
***
## bs(dis, df = 4, knots = c(4, 7, 11))2 -0.31341    0.01680 -18.658  < 2e-16
***
## bs(dis, df = 4, knots = c(4, 7, 11))3 -0.26618    0.03147  -8.459 3.00e-16
***
## bs(dis, df = 4, knots = c(4, 7, 11))4 -0.39802    0.04647  -8.565  < 2e-16
***
## bs(dis, df = 4, knots = c(4, 7, 11))5 -0.25681    0.09001  -2.853  0.00451
**
## bs(dis, df = 4, knots = c(4, 7, 11))6 -0.32926    0.06327  -5.204 2.85e-07
***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06185 on 499 degrees of freedom
## Multiple R-squared:  0.7185, Adjusted R-squared:  0.7151
## F-statistic: 212.3 on 6 and 499 DF,  p-value: < 2.2e-16

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



e. Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

We fit regression splines with dfs between 3 and 16.

```
all.cv = rep(NA, 16)
for (i in 3:16) {
    lm.fit = lm(nox ~ bs(dis, df = i), data = Boston)
    all.cv[i] = sum(lm.fit$residuals^2)
}
all.cv[-c(1, 2)]
```

```
##  [1] 1.934107 1.922775 1.840173 1.833966 1.829884 1.816995 1.825653
1.792535
##  [9] 1.796992 1.788999 1.782350 1.781838 1.782798 1.783546
```

Train RSS monotonically decreases till df=14 and then slightly increases for df=15 and df=16.

f. Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.

Finally, we use a 10-fold cross validation to find best df. We try all integer values of df between 3 and 16.

```
all.cv = rep(NA, 16)
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]
}
```

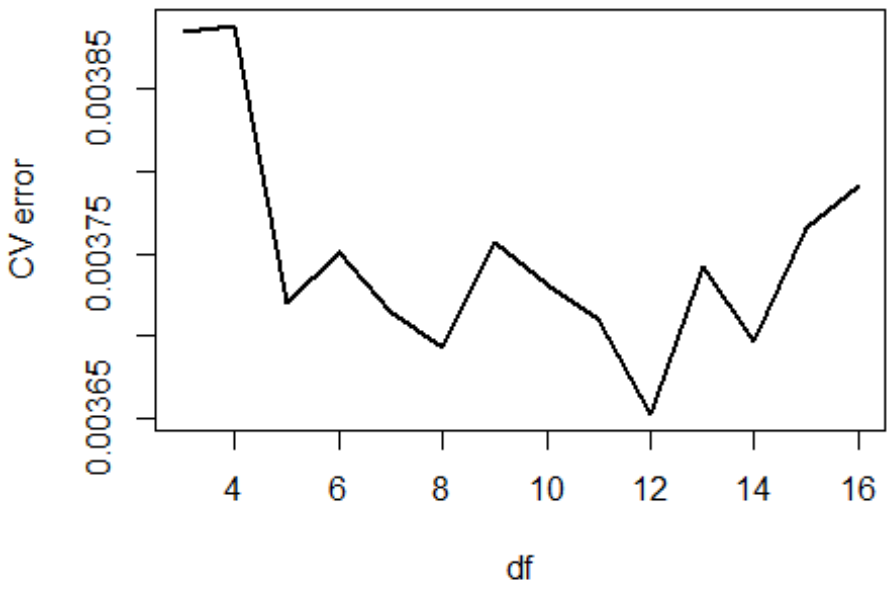
```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
c(1.1296, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
c(1.1296, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.0992), Boundary.knots
=
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.0992), Boundary.knots
=
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.2157), Boundary.knots
=
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-
conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.2157), Boundary.knots
=
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-
conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.354, `66.66667%`
=
## 4.2474: some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.354, `66.66667%`
=
## 4.2474: some 'x' values beyond boundary knots may cause ill-conditioned
bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4212,
`66.66667%`
## = 4.38856666666667: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4212,
`66.66667%`
## = 4.38856666666667: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1105, `50%` = 3.2721,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1105, `50%` = 3.2721,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1, `50%` = 3.1323,
`75%` =
## 5.118: some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1, `50%` = 3.1323,
`75%` =
## 5.118: some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.92938, `40%` =
2.55946, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.92938, `40%` =
2.55946, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.93736, `40%` =
2.59666, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.93736, `40%` =
2.59666, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.86156666666667,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.86156666666667,
```

```
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79777142857143,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79777142857143,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.7936,
`28.57143%`
## = 2.16771428571429, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.7936,
`28.57143%`
## = 2.16771428571429, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.734325, `25%` =
2.0941, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.734325, `25%` =
2.0941, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.751575, `25%` =
2.1084, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.751575, `25%` =
2.1084, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71552222222222,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71552222222222,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.66286666666667,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.66286666666667,
```

```
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.62008, `20%` =
1.92938, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.62008, `20%` =
1.92938, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.6283, `20%` = 1.9512,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.6283, `20%` = 1.9512,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61225454545455,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61225454545455,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61066363636364,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61066363636364,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.60476666666667,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.60476666666667,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.5881,
`16.66667%`
## = 1.82231666666667, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.5881,
`16.66667%`
```

```
## = 1.82231666666667, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.58949230769231,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.58949230769231,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.5741,
`15.38462%` =
## 1.8209, : some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.5741,
`15.38462%` =
## 1.8209, : some 'x' values beyond boundary knots may cause ill-conditioned
bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.54201428571429,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.54201428571429,
: some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.5768,
`14.28571%`
## = 1.81652857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.5768,
`14.28571%`
## = 1.81652857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

plot(3:16, all.cv[-c(1, 2)], lwd = 2, type = "l", xlab = "df", ylab = "CV
error")
```



CV error is jumpier in this case, but attains minimum at df=10. We pick 10 as the optimal degrees of freedom.