Math642_HW5_FyonaSun

Fyona Sun

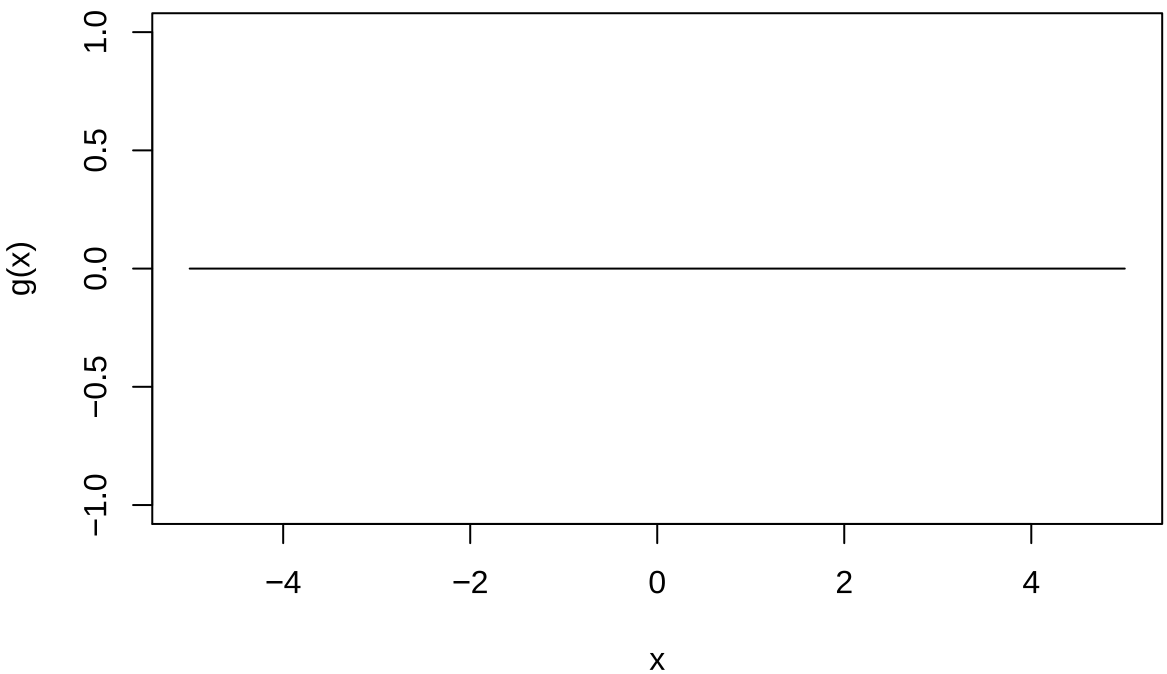
2/17/2020

7.2

a) $\lambda = \infty, m = 0$

To minimize $\hat{g}$ when $\lambda = \infty$, g(x) must equal to zero. $\hat{g}(x) = 0$

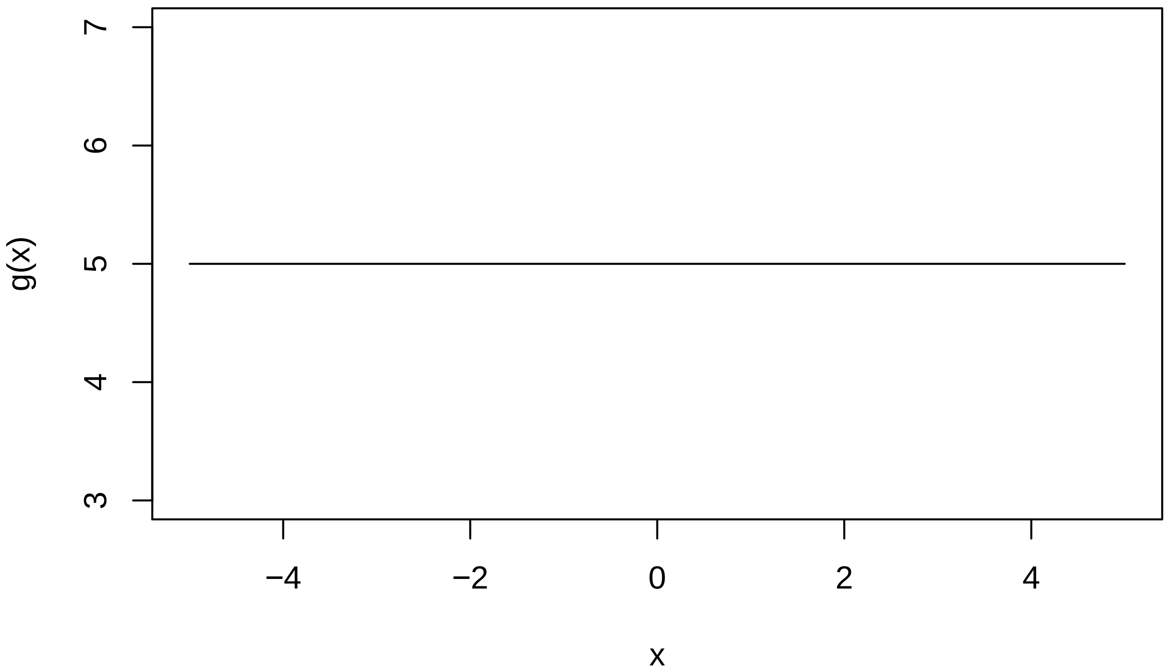
```
g<- function(x) {return(rep(0,length(x)))}
x<- -5:5
plot(x,g(x),type = 'l')
```



b) $\lambda = \infty, m = 1$

To minimize $\hat{g}$ when $\lambda = \infty$, g'(x) must equal to zero, that is g(x) must be a constant. $\hat{g}(x) = c$, c=5 in the example.

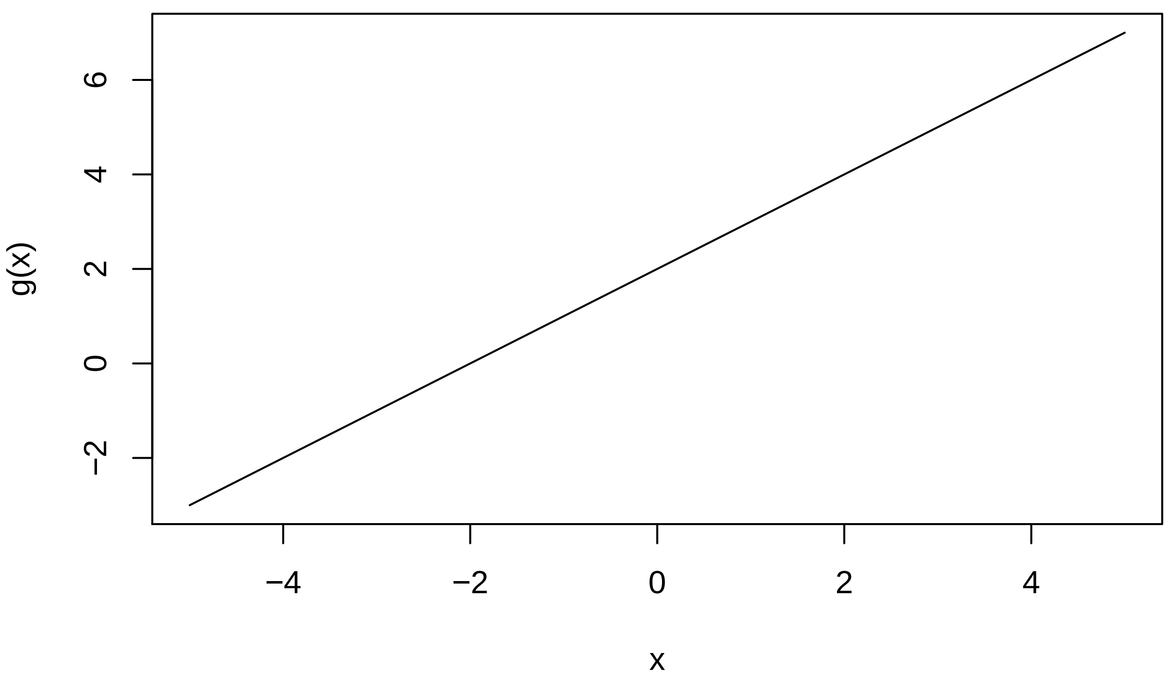
```
g<- function(x) {return(rep(5,length(x)))}
x<- -5:5
plot(x,g(x),type = 'l')
```



c)$\lambda = \infty, m = 2$

To minimize $\hat{g}$ when $\lambda = \infty$, g''(x) must equal to zero, that is g(x) must be a constant. $g(x) = ax + b$, a=1, b=2 in this example.

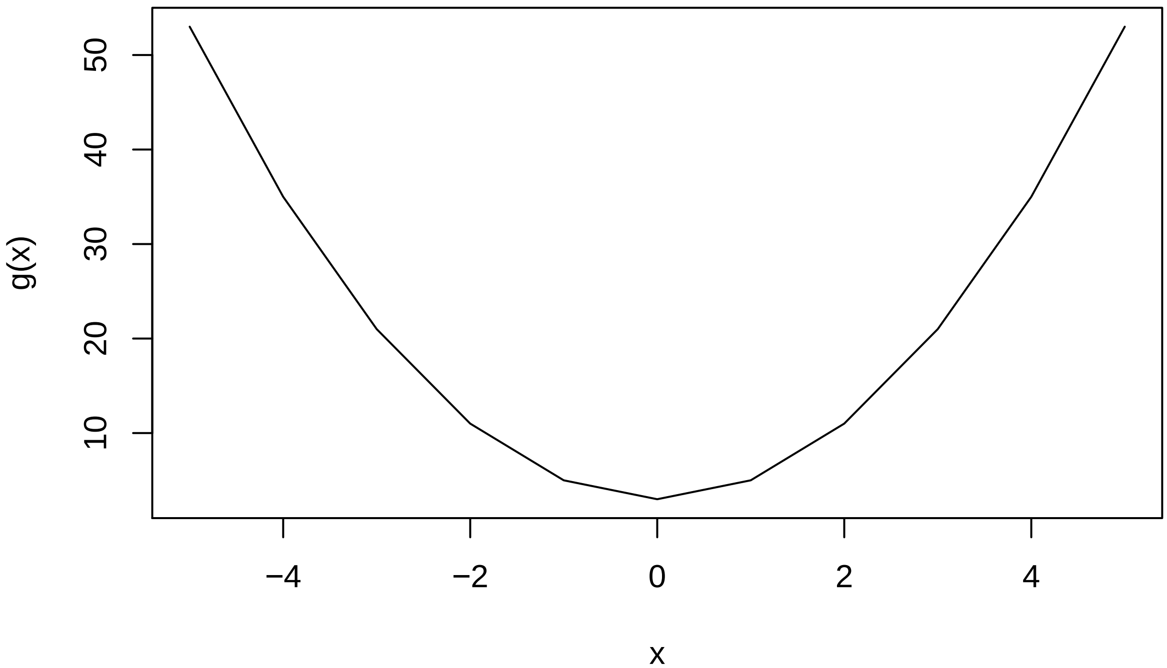
```
g<- function(x) {x+2}
x<- -5:5
plot(x,g(x),type = 'l')
```



d)$\lambda = \infty, m = 3$ To minimize $\hat{g}$ when $\lambda = \infty$, g'''(x) must equal to zero, that is g(x) must be a constant. $g(x) = ax^2 + b$, a=2, b=3 in this example.

```
g<- function(x) {2*x^2+3}
x<- -5:5
```

```r
plot(x,g(x),type = 'l')
```



e)$\lambda = 0, m = 3$ The penalty term doesn't have a function, so in this case g is the interpolating spline.

7.5

(a) Since $\hat{g}_2$ has a higher order of the penalty term, it has a higher order of polynomial. Compared with $\hat{g}_1$, $\hat{g}_2$ is more flexible and probably has a smaller training RSS.

(b) Since $\hat{g}_2$ is more flexible, it could have over-fitting problem which leads to a higher testing RSS. Thus $\hat{g}_1$ would possibaly have a smaller test RSS.

(c) When $\lambda = 0$, the penalty term disappears, that is $\hat{g}_1 = \hat{g}_2$. Therefore $\hat{g}_1$ and $\hat{g}_2$ would have the same training and test RSS.

7.9

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

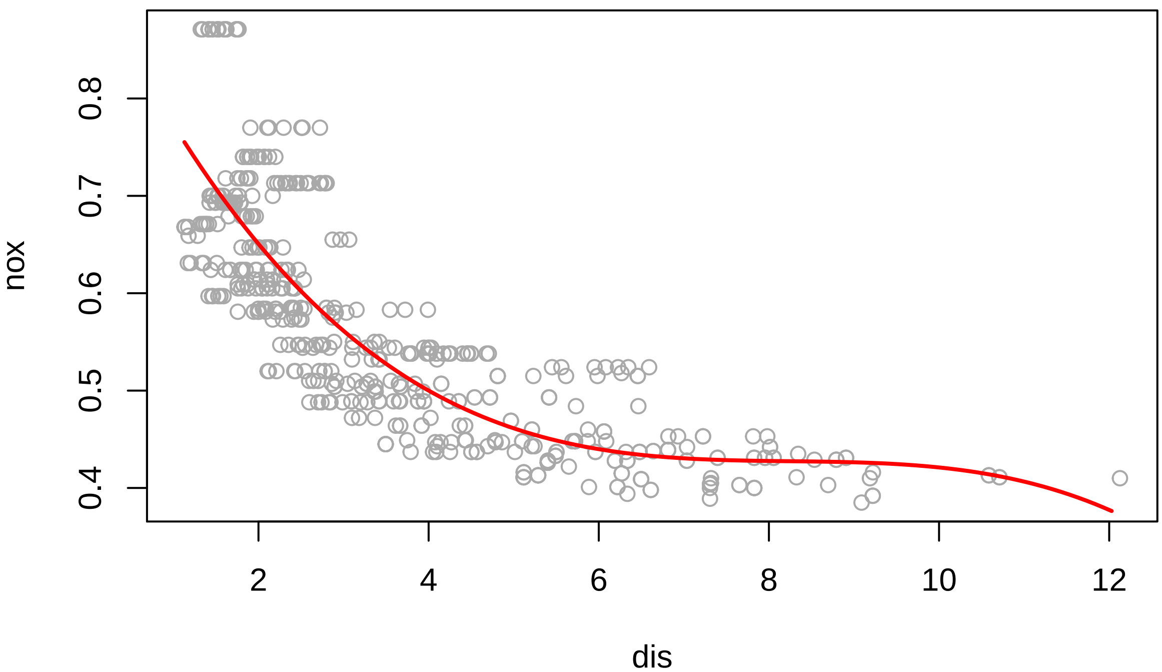
(a) Use the poly() function to fit a cubic polynomial regression to predict nox using dis. Report the regression output, and plot the resulting data and polynomial fits.

```r
library(MASS)
attach(Boston)
set.seed(1)
fit <- lm(nox ~ poly(dis, 3), data = Boston)
summary(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
```

```r
dis.new <- seq(min(Boston$dis), max(Boston$dis), by = 0.1)
pred <- predict(fit, list(dis = dis.new))
plot(nox ~ dis, data = Boston, col = "darkgrey")
lines(dis.new,pred, col = "red", lwd = 2)
```

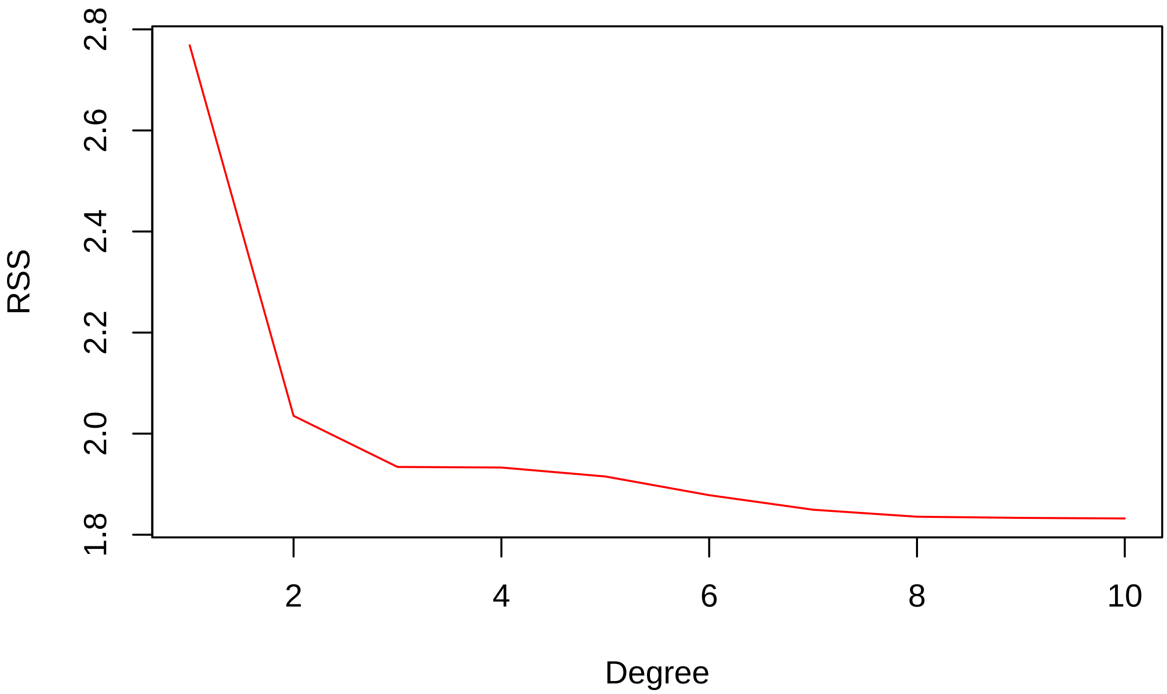


From the regression result, all the polynomial terms are significant. Thus $\hat{nox} = 0.554695 - 2.003096 * dis + 0.856330 * dis^2 - 0.318049 * dis^3$, which gives us the ploted line.

(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.

```r
rss <- rep(0, 10)
for (i in 1:10) {
    fit <- lm(nox ~ poly(dis, i), data = Boston)
    rss[i] <- sum(fit$residuals^2)
}
```

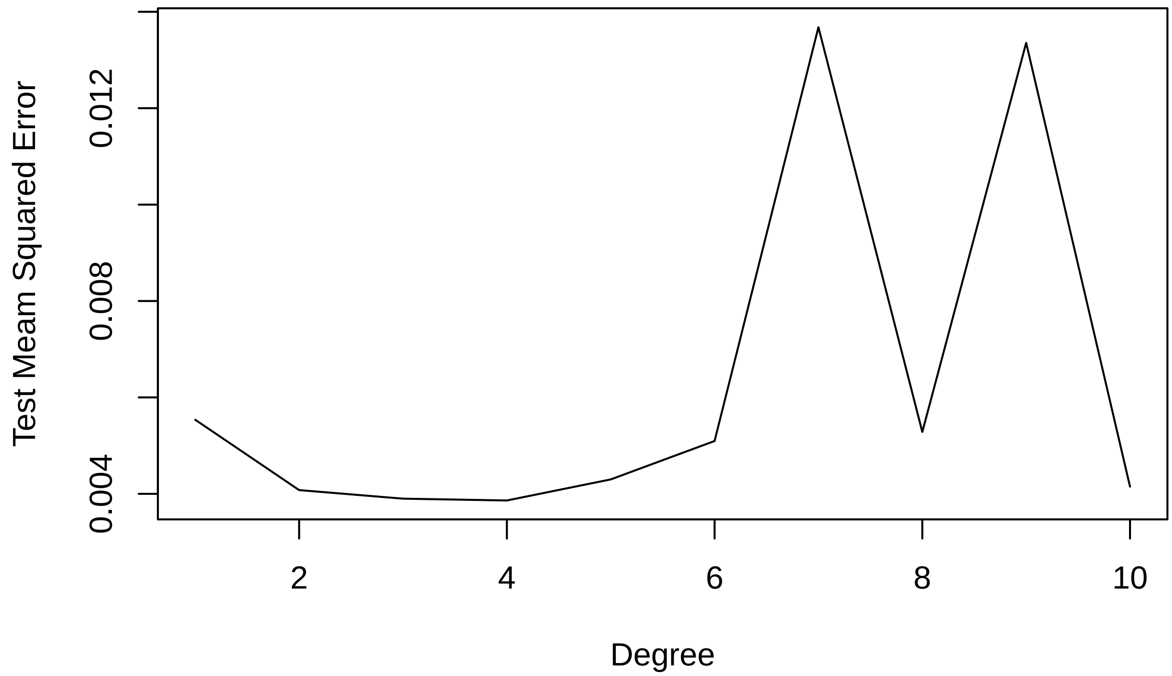
```r
plot(1:10, rss, xlab = "Degree", ylab = "RSS", type = "l", col='red')
```



The RSS decreases as the degree of the polynomial decreases. It achieves its minimum at degree 10.

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

```r
library(boot)
cv.error <- rep(0, 10)
for (i in 1:10) {
    fit <- glm(nox ~ poly(dis, i), data = Boston)
    cv.error[i] <- cv.glm(Boston, fit, K = 10)$delta[1]
}

plot(1:10, cv.error, xlab = "Degree", ylab = "Test Meam Squared Error", type = "l")
```




```
cv.error
```

```
##  [1] 0.005536329 0.004077147 0.003899587 0.003862127 0.004298590
##  [6] 0.005095283 0.013680327 0.005284520 0.013355413 0.004148996
```

At degree of 3, the test MSE is 0.003874862 and it's the smallest of the 10 degress.

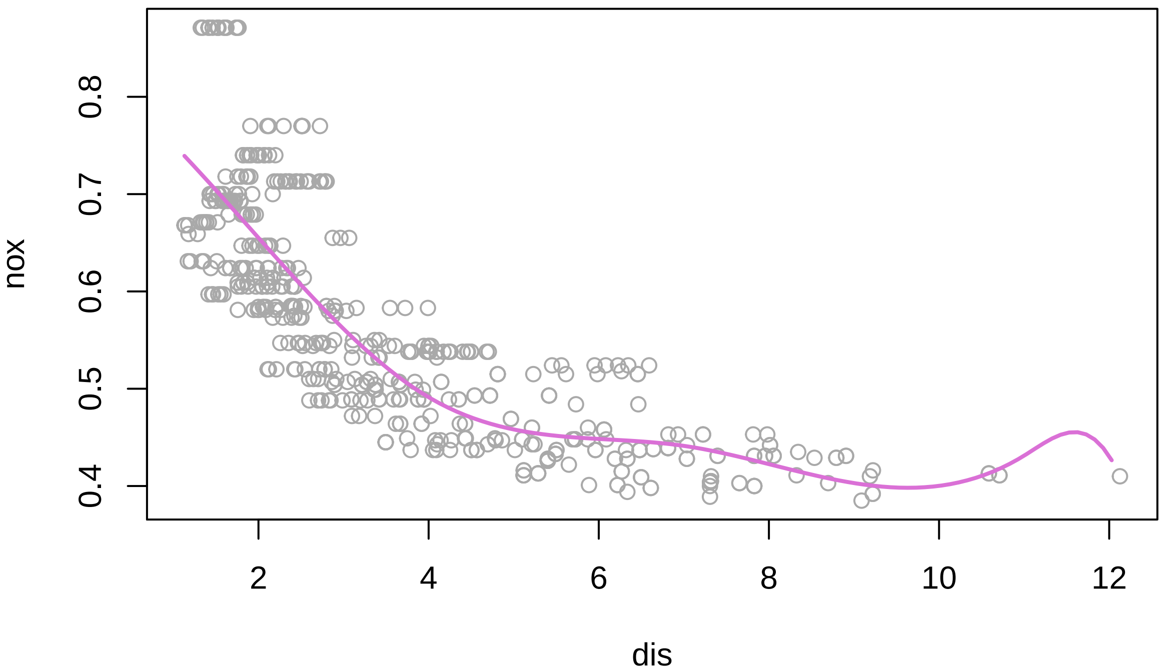
(d) Use the bs() function to fit a regression spline to predict nox using dis. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

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

```
##
## Call:
## lm(formula = nox ~ bs(dis, 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, knots = c(4, 7, 11))1 -0.08861    0.02504  -3.539  0.00044 ***
## bs(dis, knots = c(4, 7, 11))2 -0.31341    0.01680 -18.658  < 2e-16 ***
## bs(dis, knots = c(4, 7, 11))3 -0.26618    0.03147  -8.459 3.00e-16 ***
## bs(dis, knots = c(4, 7, 11))4 -0.39802    0.04647  -8.565  < 2e-16 ***
## bs(dis, knots = c(4, 7, 11))5 -0.25681    0.09001  -2.853  0.00451 **
## bs(dis, 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
```

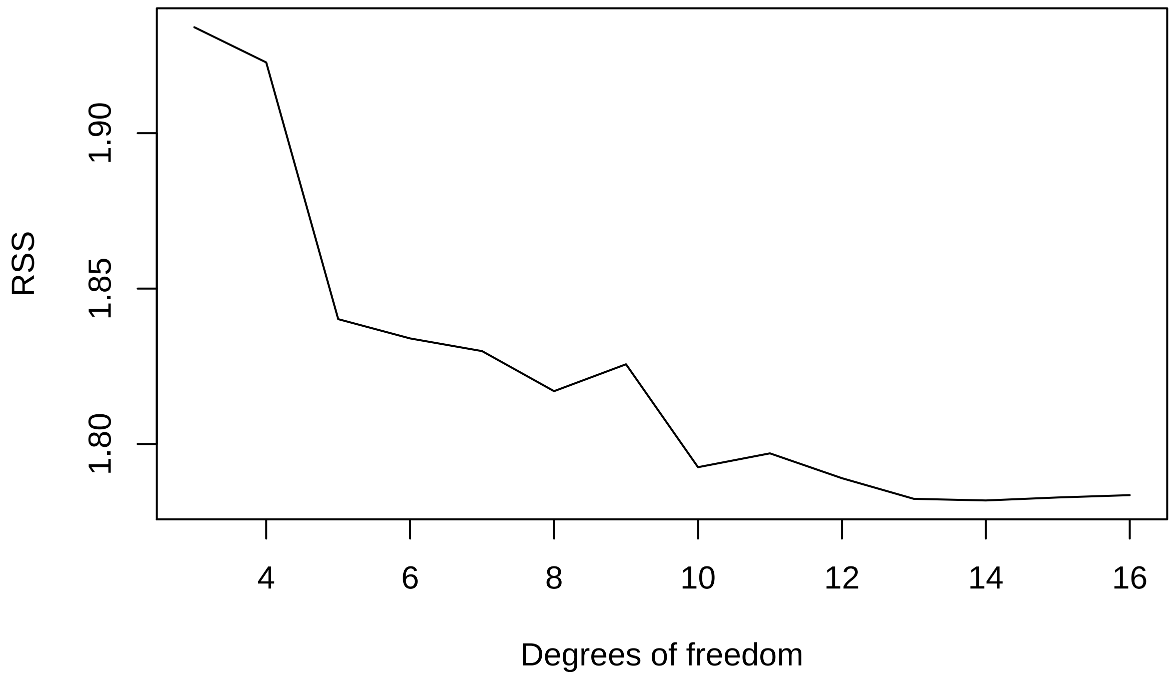
```
pred <- predict(fit, list(dis = dis.new))
plot(nox ~ dis, data = Boston, col = "darkgrey")
lines(dis.new, pred, col = "orchid", lwd = 2)
```



Here we have prespecified knots at dis 4, 7, and 11. This produces a spline with six basis functions. The regression indicates that all the spine terms are significant.

(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.

```
#Recall that a cubic spline with three knots has seven degrees of freedom
rss <- rep(0, 16)
for (i in 3:16) {
    fit <- lm(nox ~ bs(dis, df = i), data = Boston)
    rss[i] <- sum(fit$residuals^2)
}
plot(3:16, rss[-c(1, 2)], xlab = "Degrees of freedom", ylab = "RSS", type = "l")
```




```
rss[-c(1, 2)]
```

```
##  [1] 1.934107 1.922775 1.840173 1.833966 1.829884 1.816995 1.825653
##  [8] 1.792535 1.796992 1.788999 1.782350 1.781838 1.782798 1.783546
```

RSS decreases as the degrees of freedom increases at first, and RSS slightly increases after 14.

(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.

```
cv <- rep(0, 16)
for (i in 3:16) {
    fit <- glm(nox ~ bs(dis, df = i), data = Boston)
    cv[i] <- cv.glm(Boston, fit, K = 10)$delta[1]
}
```

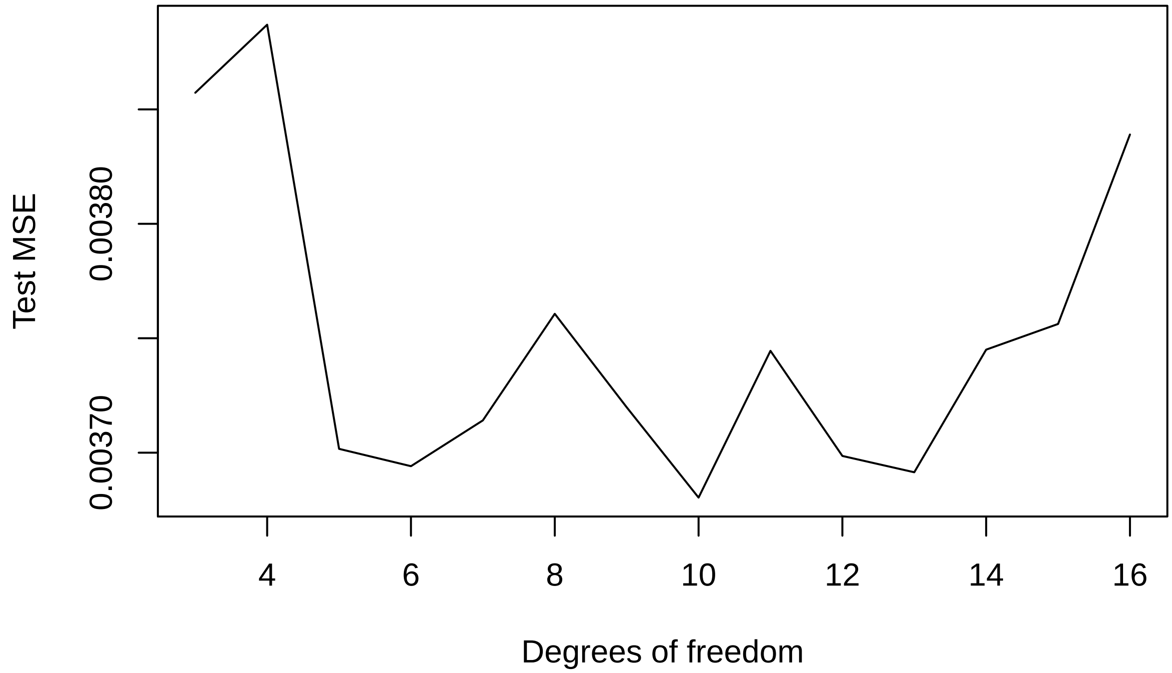
```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## 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%` = 3.0993), 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%` = 3.0993), 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%` = 3.1523), Boundary.knots =
## c(1.1691, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.1523), Boundary.knots =
## c(1.1691, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.38403333333333, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.38403333333333, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.38876666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.38876666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.102875, `50%` =
## 3.26745, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.102875, `50%` =
## 3.26745, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.0788, `50%` = 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.0788, `50%` = 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.94264, `40%` =
## 2.62334, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.94264, `40%` =
## 2.62334, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.97036, `40%` =
## 2.66262, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.97036, `40%` =
## 2.66262, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.87351666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.87351666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.84476666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.84476666666667, :
## 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%` = 1.79078571428571, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79078571428571, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.734325, `25%` =
## 2.0493, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.734325, `25%` =
## 2.0493, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.7275, `25%` =
## 2.0581, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.7275, `25%` =
## 2.0581, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.65344444444444, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.65344444444444, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71806666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71806666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.6424, `20%` =
## 1.96376, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.6424, `20%` =
## 1.96376, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.64186, `20%` =
```

```
## 1.95434, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.64186, `20%` =
## 1.95434, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.59590909090909, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.59590909090909, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61450909090909, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61450909090909, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.591425,
## `16.66667%` = 1.86565, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.591425,
## `16.66667%` = 1.86565, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.58948333333333, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.58948333333333, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.5888, `15.38462%`
## = 1.8172, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.5888, `15.38462%`
## = 1.8172, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.5311, `14.28571%`
## = 1.80062857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.5311, `14.28571%`
## = 1.80062857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.584, `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%` = 1.584, `14.28571%`
## = 1.81652857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```

```r
plot(3:16, cv[-c(1, 2)], xlab = "Degrees of freedom", ylab = "Test MSE", type = "l")
```



The test MSE minized at 8 degrees of freedom. ##7.11 (a) Generate a response Y and two predictors X1 and X2, with n = 100.

```r
set.seed(1)
Y <- rnorm(100)
X1 <- rnorm(100)
X2 <- rnorm(100)
```

(b) Initialize β_1 to take on a value of your choice. It does not matter what value you choose.

```r
beta1 <- 0.25
```

(c) Keeping β_1 fixed, fit the model $Y - \hat{\beta}_1 X_1 = \beta_0 + \beta_2 X_2 + \epsilon$

```r
a<- Y-beta1*X1
beta2<- lm(a~X2)$coef[2]

beta2
```

```
##         X2
## 0.02743328
```

(d) Keeping β_2 fixed, fit the model $Y - \hat{\beta}_2 X_2 = \beta_0 + \beta_1 X_1 + \epsilon$

```r
a<- Y-beta2*X2
beta1<- lm(a~X1)$coef[2]

beta1
```

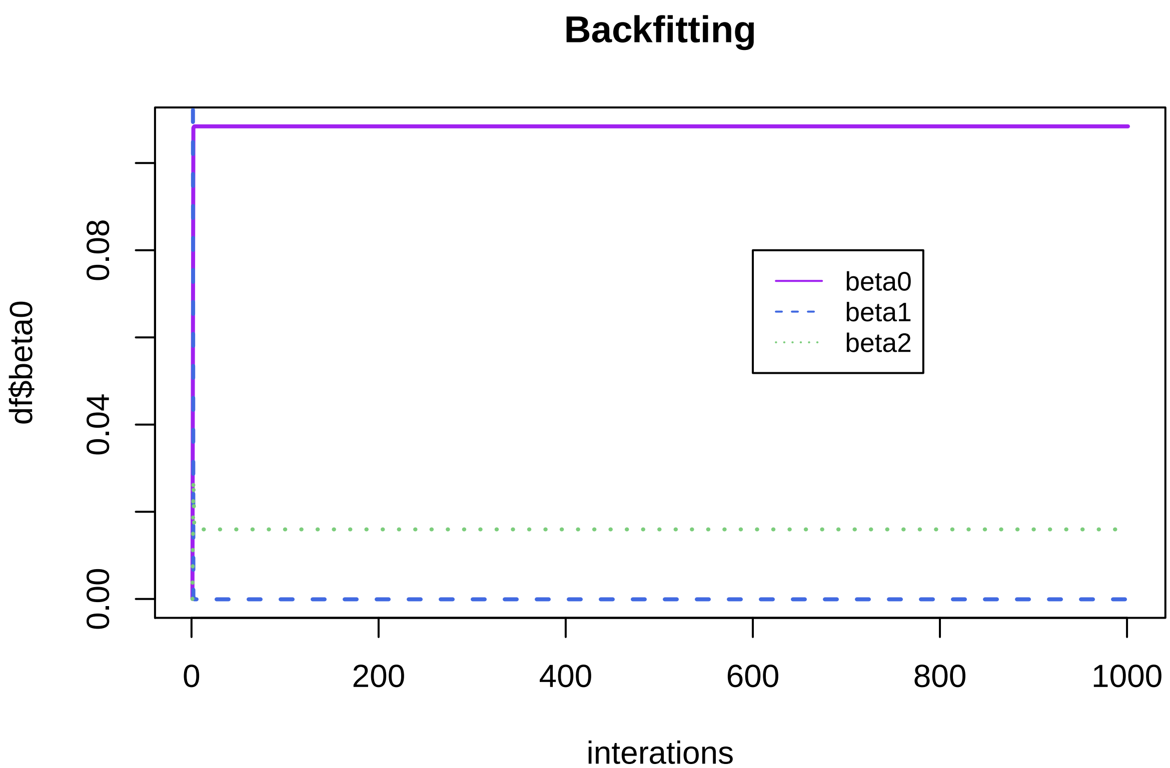
```
##           X1
## 0.0005349403
```

(e)Write a for loop to repeat (c) and (d) 1,000 times.

```r
iter<- 1000
df<- data.frame(0, 0.25, 0)
names(df)=c('beta0','beta1','beta2')
for (i in 1:iter) {
  beta1 <- df[nrow(df), 2]
  a <- Y - beta1 * X1
  beta2 <- lm(a ~ X2)$coef[2]
  a <- Y - beta2 * X2
  beta1 <- lm(a ~ X1)$coef[2]
  beta0 <- lm(a ~ X1)$coef[1]
  df[nrow(df) + 1,] <- list(beta0, beta1, beta2)
}

head(df)
```

```
##       beta0         beta1      beta2
## 1 0.0000000  2.500000e-01 0.00000000
## 2 0.1080935  5.349403e-04 0.02743328
## 3 0.1084100 -7.720619e-05 0.01598840
## 4 0.1084108 -7.870830e-05 0.01596032
## 5 0.1084108 -7.871198e-05 0.01596025
## 6 0.1084108 -7.871199e-05 0.01596025
```

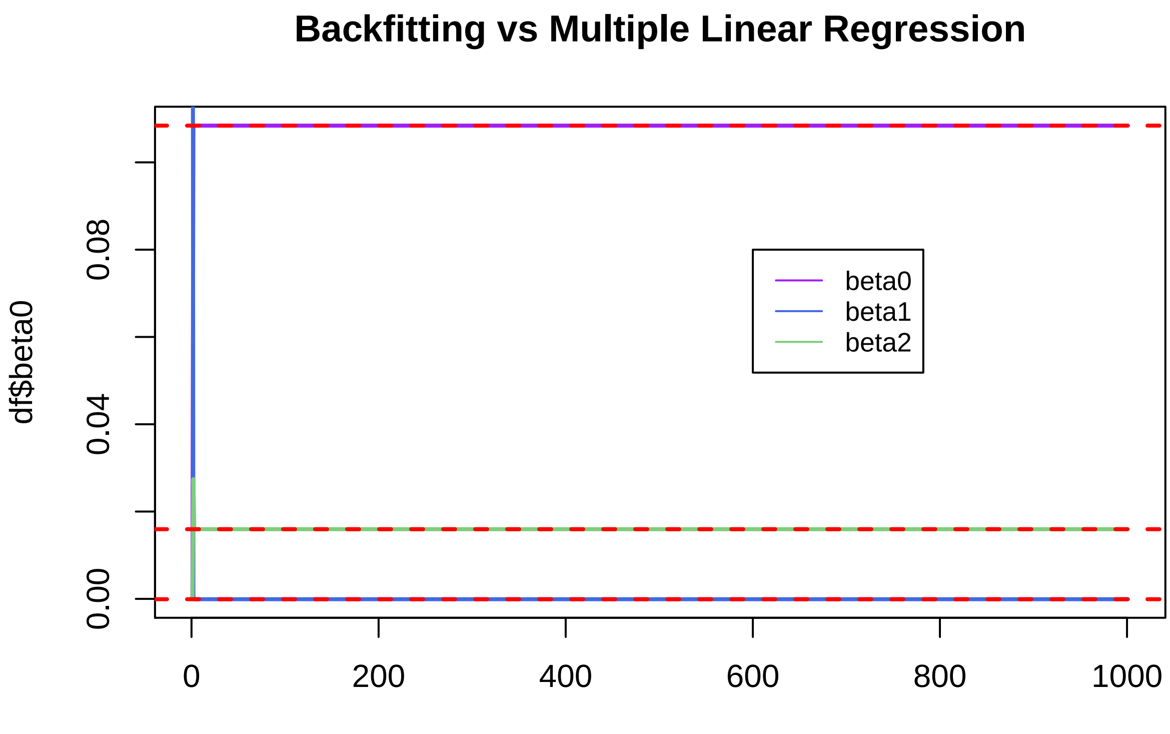
```r
plot(df$beta0, col = 'purple', type = 'l',lwd=2,xlab = 'interations', main='Backfitting')
lines(df$beta1, col = 'royalblue',lwd=2, lty=2)
lines(df$beta2, col = 'palegreen3',lwd=2,lty=3)
legend(600,0.08,legend=c('beta0', 'beta1', 'beta2'),
       col=c('purple', 'royalblue','palegreen3'),lty=1:3,cex=0.8)
```



beta0, beta1 and beta2 atained the least squared values very quick and remained unchange.

(f) Compare your answer in (e) to the results of simply performing multiple linear regression to predict Y using X1 and X2. Use the abline() function to overlay those multiple linear regression coefficient estimates on the plot obtained in (e).

```
lm.fit<-coef(lm(Y~X1+X2))
plot(df$beta0, col = 'purple', type='l',lwd=2,xlab = 'interations', main='Backfitting vs Multiple Linea
lines(df$beta1, col = 'royalblue',lwd=2)
lines(df$beta2, col = 'palegreen3',lwd=2)
legend(600,0.08,legend=c('beta0', 'beta1', 'beta2'),
       col=c('purple', 'royalblue','palegreen3'),lty=1,cex=0.8)
abline(h=lm.fit[1], col="red", lty=2, lwd=2)
abline(h=lm.fit[2], col="red", lty=2, lwd=2)
abline(h=lm.fit[3], col="red", lty=2, lwd=2)
```



The red line on this graph show that the coefficients given by multiple regression match with the coefficients obtained by the backfitting estimation.

(g) On this data set, how many backfitting iterations were required in order to obtain a "good" approximation to the multiple regression coefficient estimates?

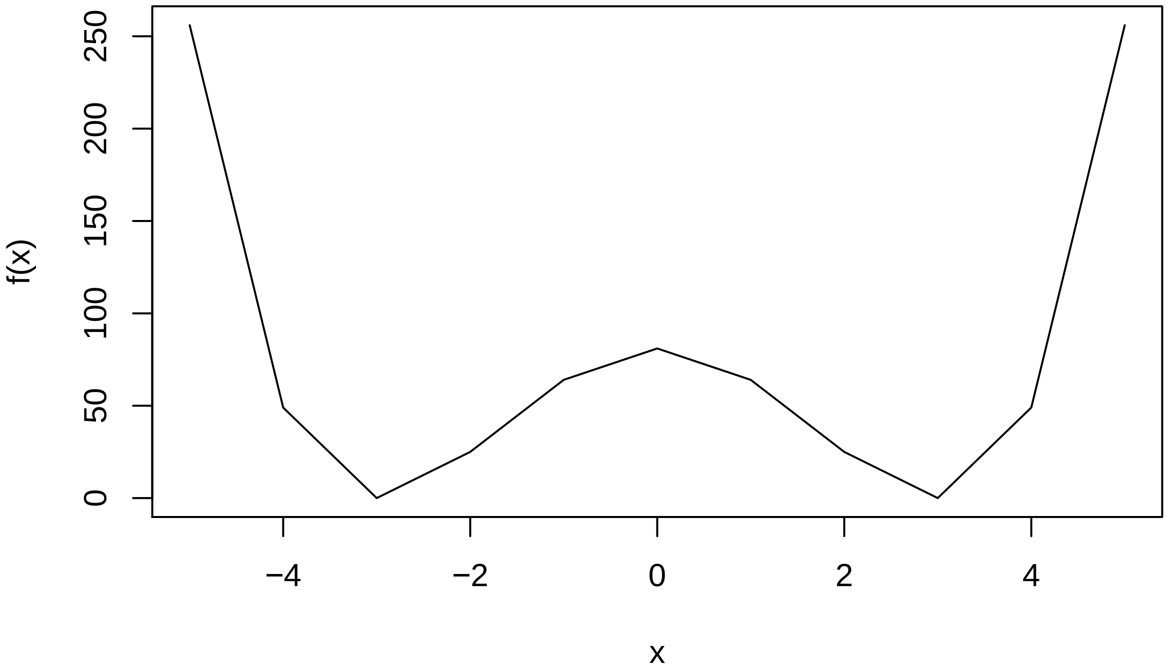
```
head(df)
```

```
##       beta0         beta1      beta2
## 1 0.0000000  2.500000e-01 0.00000000
## 2 0.1080935  5.349403e-04 0.02743328
## 3 0.1084100 -7.720619e-05 0.01598840
## 4 0.1084108 -7.870830e-05 0.01596032
## 5 0.1084108 -7.871198e-05 0.01596025
## 6 0.1084108 -7.871199e-05 0.01596025
```

From the iteration results, beta0 converges to constant at iteration 4, beta1 converges at iteration4 and beta2 converges at iteration 5. So for this data set at least 5 iterations are required to obtain a 'good' approximation to the multiple regression coefficient estimates.

Write a Gradient Descent program in R to find the minimum of the equation. $y_i = (x_i^2 - 9)^2$

```
#plot the function
f<- function(x) ((x^2) -9)^2
x<- -5:5
plot(x,f(x),type = 'l')
```




```
#using gradient descent to find the numerical solution

gd<- function(x,f,lr,iter){
x<- 0.001
xtrace<- x
ytrace<- f(x)
lr<- 0.01
for (i in 1:iter){
  delta<- (f(x+lr)-f(x))/lr
  x<- x - lr*delta
  xtrace<- c(xtrace,x)
  ytrace<- c(ytrace,f(x))
}
  print(x)
  print(xtrace)
  print(ytrace)
}

iter<- 100
gd(x,f,lr,iter)
```

```
## [1] 2.994996
##   [1] 0.001000000 0.003159985 0.006097550 0.010092603 0.015525787
##   [6] 0.022914704 0.032963099 0.046627588 0.065207964 0.090468918
##  [11] 0.124802828 0.171444235 0.234744257 0.320500755 0.436301219
##  [16] 0.591731538 0.798054724 1.066438122 1.402955208 1.798175958
##  [21] 2.212800507 2.574864061 2.816780641 2.933887813 2.976550286
```

```
##  [26] 2.989708809 2.993505399 2.994577709 2.994878695 2.994963029
##  [31] 2.994986648 2.994993261 2.994995113 2.994995632 2.994995777
##  [36] 2.994995818 2.994995829 2.994995832 2.994995833 2.994995833
##  [41] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [46] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [51] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [56] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [61] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [66] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [71] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [76] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [81] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [86] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [91] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
##  [96] 2.994995833 2.994995833 2.994995833 2.994995833 2.994995833
## [101] 2.994995833
##   [1] 8.099998e+01 8.099982e+01 8.099933e+01 8.099817e+01 8.099566e+01
##   [6] 8.099055e+01 8.098044e+01 8.096087e+01 8.092348e+01 8.085274e+01
##  [11] 8.071988e+01 8.047179e+01 8.001115e+01 7.916158e+01 7.760978e+01
##  [16] 7.481997e+01 6.994159e+01 6.182220e+01 4.944504e+01 3.325325e+01
##  [21] 1.683883e+01 5.617256e+00 1.135816e+00 1.539011e-01 1.964157e-02
##  [26] 3.799642e-03 1.515189e-03 1.056532e-03 9.425885e-04 9.118258e-04
##  [31] 9.033018e-04 9.009221e-04 9.002563e-04 9.000700e-04 9.000178e-04
##  [36] 9.000032e-04 8.999991e-04 8.999979e-04 8.999976e-04 8.999975e-04
##  [41] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [46] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [51] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [56] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [61] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [66] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [71] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [76] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [81] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [86] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [91] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [96] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
## [101] 8.999975e-04
```

The gradient descent with the starting x=0.01 and 100 iteration gives x=2.994996 at its minimum, which is close to one of the solution x=3.

```r
#using gradient descent to find the numerical solution

gd<- function(x,f,lr,iter){
x<- -5
xtrace<- x
ytrace<- f(x)
lr<- 0.01
for (i in 1:iter){
  delta<- (f(x+lr)-f(x))/lr
  x<- x - lr*delta
  xtrace<- c(xtrace,x)
  ytrace<- c(ytrace,f(x))
}
  print(x)
```

```r
  print(xtrace)
  print(ytrace)
}

iter<- 100
gd(x,f,lr,iter)
```

```
## [1] -3.004996
##   [1] -5.000000 -1.813180 -2.227648 -2.588589 -2.828867 -2.944730 -2.986823
##   [8] -2.999789 -3.003528 -3.004584 -3.004880 -3.004964 -3.004987 -3.004993
##  [15] -3.004995 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [22] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [29] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [36] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [43] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [50] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [57] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [64] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [71] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [78] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [85] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [92] -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996 -3.004996
##  [99] -3.004996 -3.004996 -3.004996
##   [1] 2.560000e+02 3.263127e+01 1.630208e+01 5.286345e+00 9.950248e-01
##   [6] 1.079534e-01 6.223686e-03 1.607869e-06 4.486260e-04 7.576521e-04
##  [11] 8.588826e-04 8.883865e-04 8.967386e-04 8.990844e-04 8.997418e-04
##  [16] 8.999259e-04 8.999775e-04 8.999919e-04 8.999959e-04 8.999971e-04
##  [21] 8.999974e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [26] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [31] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [36] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [41] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [46] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [51] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [56] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [61] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [66] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [71] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [76] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [81] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [86] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [91] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
##  [96] 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04 8.999975e-04
## [101] 8.999975e-04
```

The gradient descent with the starting x=-5 and 100 iteration gives x=-3.004996 at its minimum, which is close to the other solution x=-3.