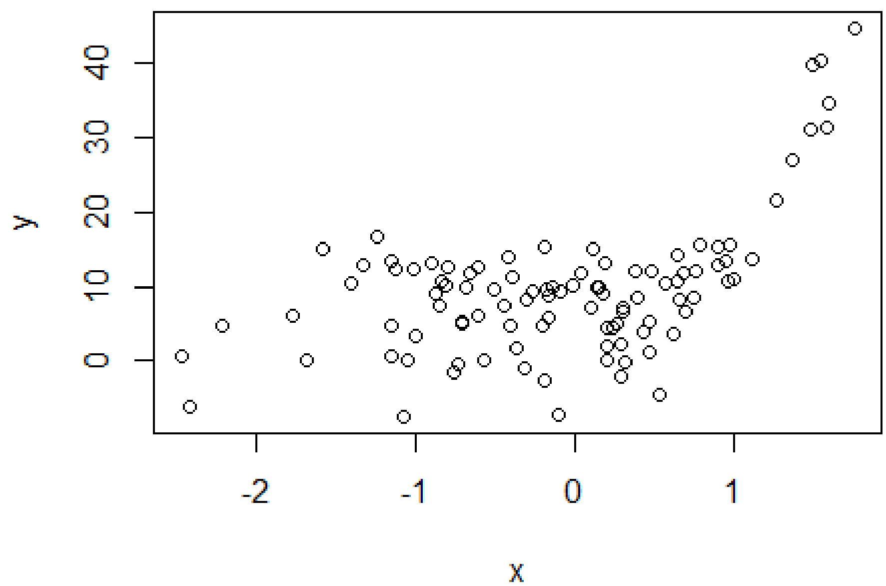
HW6

Anish Mohan

November 9, 2015

Q1

```
x=rnorm(100)
eps=rnorm(100)
y=5+2*x+7*x*x+3*x*x*x+5*eps
plot(x,y)
```

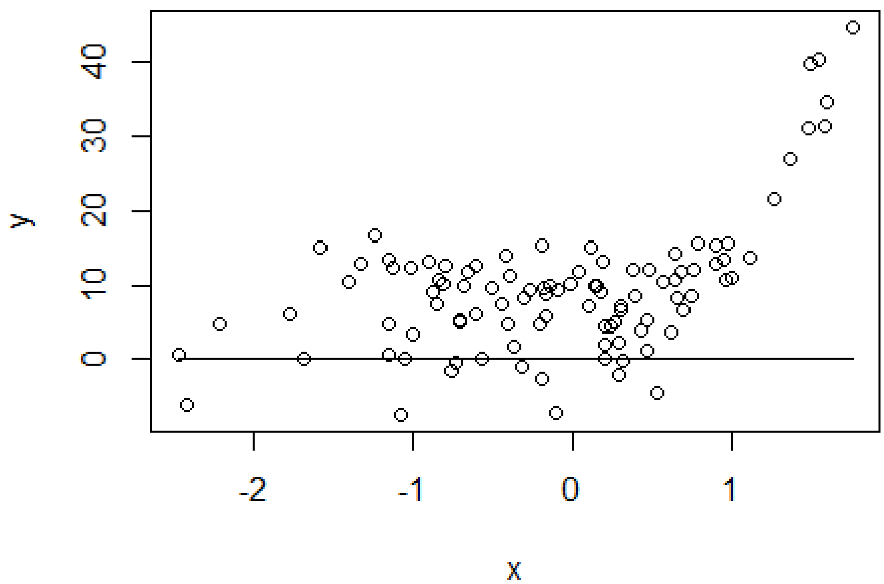



```
u=lm(y~-1)
u0=lm(y~1)
u1=lm(y~poly(x,1))
u2=lm(y~poly(x,2))
u3=lm(y~poly(x,3))
u4=lm(y~poly(x,4))
u5=lm(y~poly(x,5))
xlim=range(x)
xgrid=seq(from=xlim[1],to=xlim[2], length.out = 100)
```

- 1a. $\lambda = \infty$, m=0

  Since $\lambda = \infty$, to minimize the equation, we need $g^{(0)}$ be zero or tend towards zero. This is possible with a function g(x)=0

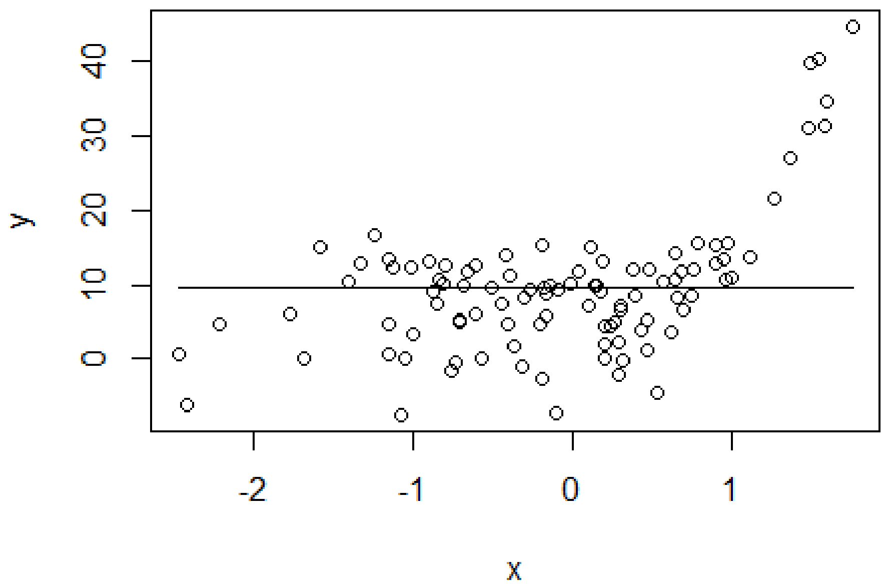
```
  plot(x,y)
tempval=rep(0,100)
lines(xgrid,tempval)
```



- 1b. $\lambda = \infty$, m=1

  Since $\lambda = \infty$, to minimize the equation, we need $g^{(1)}$ to be lowest or tend towards zero. This is only possible if g(x) is a constant function i.e g(x)= β_0

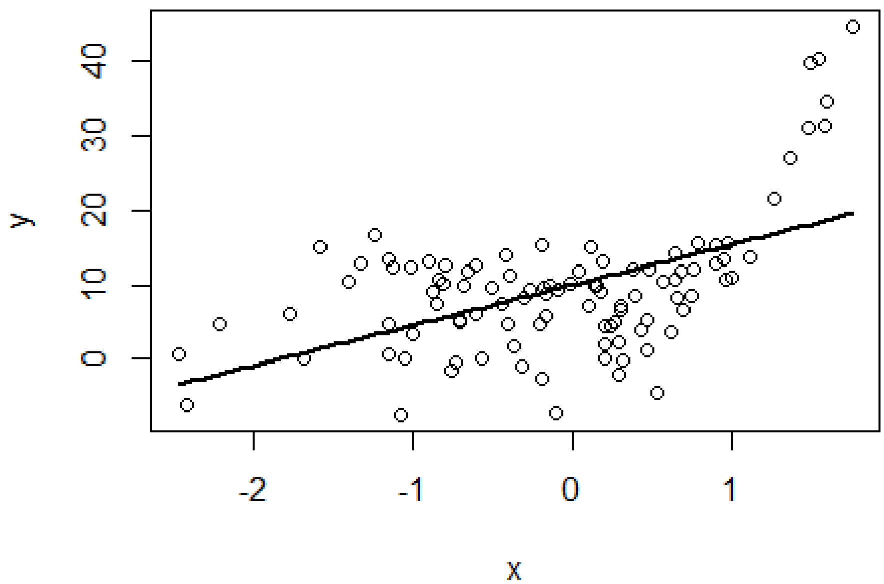
```
plot(x,y)
val=u0$coefficients[1]
tempval=rep(val,100)
lines(xgrid,tempval)
```



- 1c. $\lambda = \infty$, m=2

  Since $\lambda = \infty$, to minimize the equation, we need $g^{(2)}$ to be lowest or tend towards zero. This is possible if $g^{(1)}$ is constant i.e g(x) is a linear function with constant slope. Hence the function would be g(x)=$\beta_0 + \beta_1 * x$.

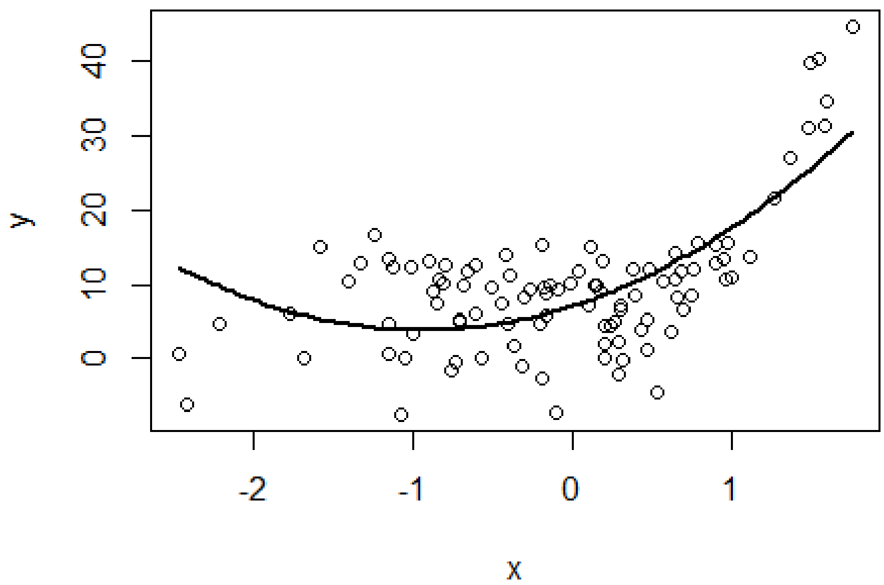
```
plot(x,y)
xlim=range(x)
xgrid=seq(from=xlim[1],to=xlim[2], length.out = 100)
pred=predict(u1,newdata=list(x=xgrid),se=T)
lines(xgrid, pred$fit, lwd=2)
```



- 1d. $\lambda = \infty$, m=3

  Since $\lambda = \infty$, to minimize the equation, we need $g^{(3)}$ to be lowest or tend towards zero. This is possible if $g^{(2)}$ is constant. This implied $g^{(1)}$ is uniformly increasing or has constant slope i.e the function g(x) would be incresing at a constant rate. This is possible with a function $g(x)=\beta_0 + \beta_1 * x + \beta_2 * x^2$

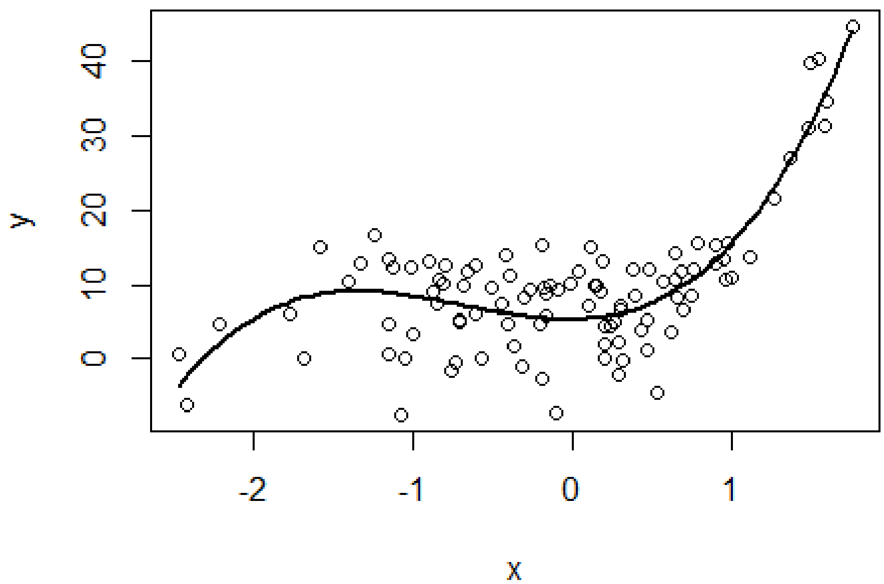
```
plot(x,y)
xlim=range(x)
xgrid=seq(from=xlim[1],to=xlim[2], length.out = 100)
pred=predict(u2,newdata=list(x=xgrid),se=T)
lines(xgrid, pred$fit, lwd=2)
```



- 1e. $\lambda = 0$, m=3

Since $\lambda = 0$, to minimize the equation, we need residual erro to be lowest. This is possible with a function that minimizes the RSS which in this case is $g(x)=\beta_0 + \beta_1 * x + \beta_2 * x^2 + \beta_3 * x^3$ a shown by low p value from the summary results.

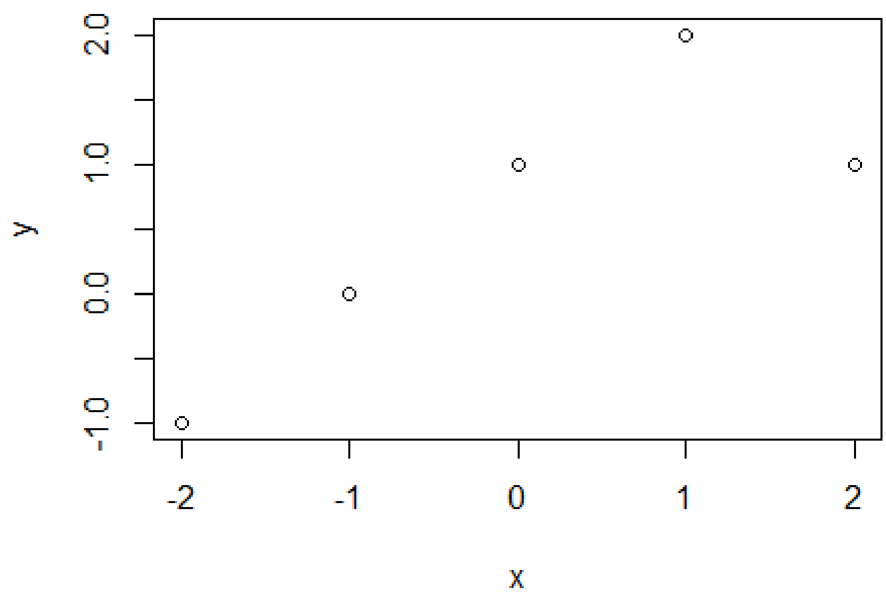
```
plot(x,y)
xlim=range(x)
xgrid=seq(from=xlim[1],to=xlim[2], length.out = 100)
pred=predict(u3,newdata=list(x=xgrid),se=T)
lines(xgrid, pred$fit, lwd=2)
```




```
summary(u4)

##
## Call:
## lm(formula = y ~ poly(x, 4))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.6386  -3.1319   0.6367   3.8068   9.1043
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)     9.492      0.517  18.362  < 2e-16 ***
## poly(x, 4)1    48.803      5.170   9.440 2.60e-15 ***
## poly(x, 4)2    39.694      5.170   7.678 1.42e-11 ***
## poly(x, 4)3    43.296      5.170   8.375 4.85e-13 ***
## poly(x, 4)4     6.683      5.170   1.293    0.199
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.17 on 95 degrees of freedom
## Multiple R-squared:  0.6983, Adjusted R-squared:  0.6856
## F-statistic: 54.97 on 4 and 95 DF,  p-value: < 2.2e-16
```

Q2

```
x=-2:2
beta0=1
beta1=1
beta2=-2
y=rep(NA,5)

for(i in 1:5 ) {
  if(x[i]<1){
    y[i]=beta0+x[i]*beta1
  } else {
    y[i]=beta0+beta1*x[i]+beta2*(x[i]-1)^2
  }
}
plot(x,y)
```



Q3

- 3a. Functions $\hat{g}_2$ will have a smaller training error as it is higher order polynomial and has higher order penalty function. It will fit the training data well
- 3b. Functions $\hat{g}_1$ could have a smaller test error as the higher order function $\hat{g}_2$ will tend to overfit the noise.
- 3c. Functions $\hat{g}_1$ and $\hat{g}_2$ are same when $\lambda = 0$, hence they will have same training and test error.

Q4

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.2.2
```

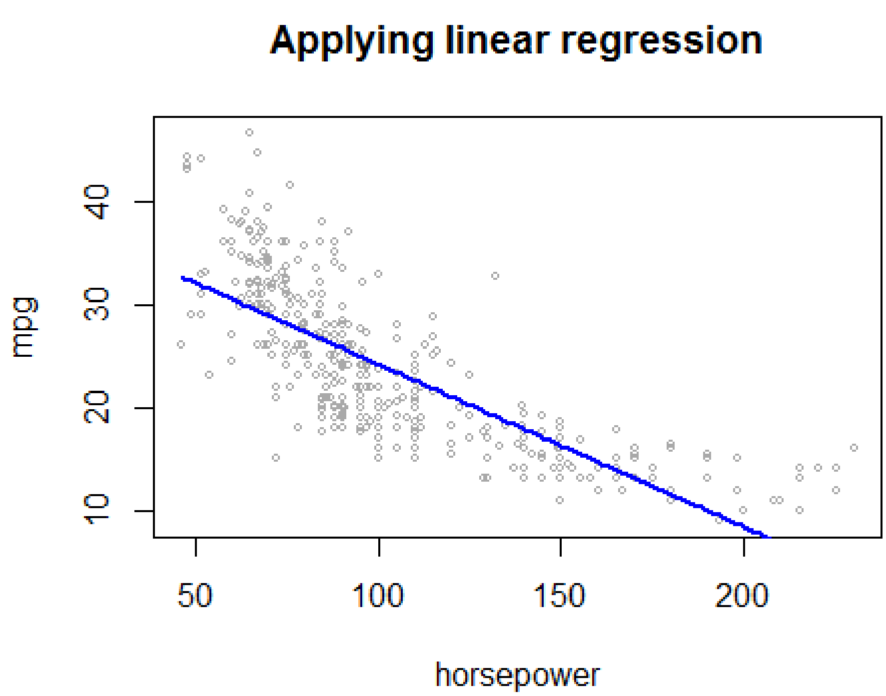
```r
library(boot)
```

```
## Warning: package 'boot' was built under R version 3.2.2
```

```r
attach(Auto)

set.seed(1)
#10-Fold validation comparing a linear and cubic model

fit.lm=glm(mpg~horsepower, data=Auto)
cv.error=cv.glm(Auto,fit.lm,K=10)
cv.error$delta
```

```
## [1] 24.10716 24.09865
```

```r
hplims=range(Auto$horsepower)
hp.grid=seq(from=hplims[1],to=hplims[2])

plot(horsepower, mpg, xlim=hplims, cex=0.5, col="darkgrey")
preds=predict(fit.lm,newdata=list(horsepower=hp.grid), se=T)
lines(hp.grid, preds$fit, lwd=2, col="blue")
title("Applying linear regression")
```



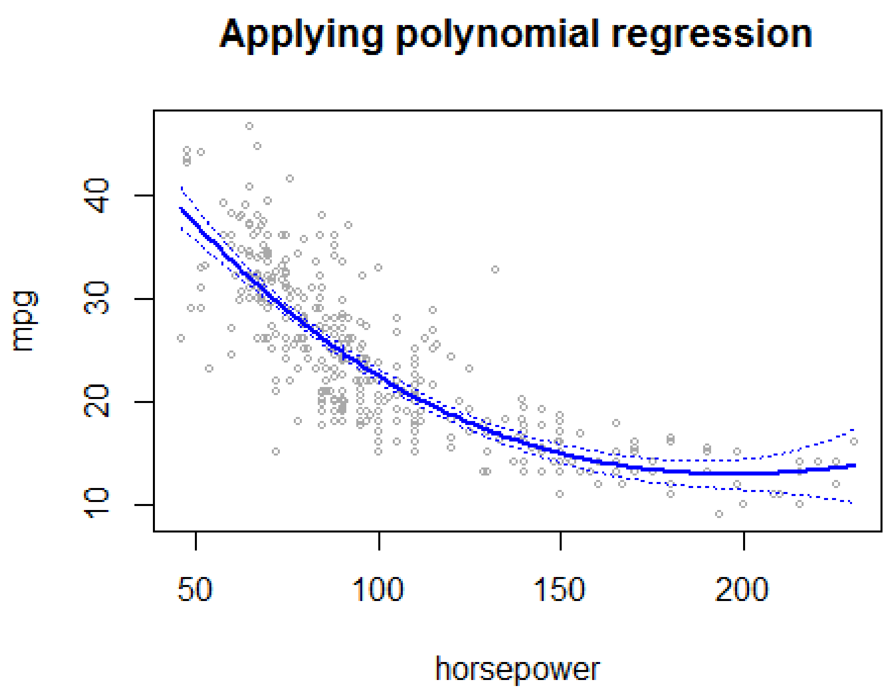
```r
fit.poly=glm(mpg~poly(horsepower,3),data=Auto)
cv.error=cv.glm(Auto,fit.poly,K=10)
cv.error$delta
```

```
## [1] 19.37347 19.35072
```

```r
print(paste0("Error from polynomial regression is smaller than linear regression" ))
```

```
## [1] "Error from polynomial regression is smaller than linear regression"
```

```r
#Using Polynomial regression

preds=predict(fit.poly,newdata=list(horsepower=hp.grid), se=T)
se.bands=cbind(preds$fit+2*preds$se,preds$fit-2*preds$se.fit)

plot(horsepower, mpg, xlim=hplims, cex=0.5, col="darkgrey")
title("Applying polynomial regression")
lines(hp.grid, preds$fit, lwd=2, col="blue")
matlines(hp.grid, se.bands, lwd=1, col="blue", lty=3)
```



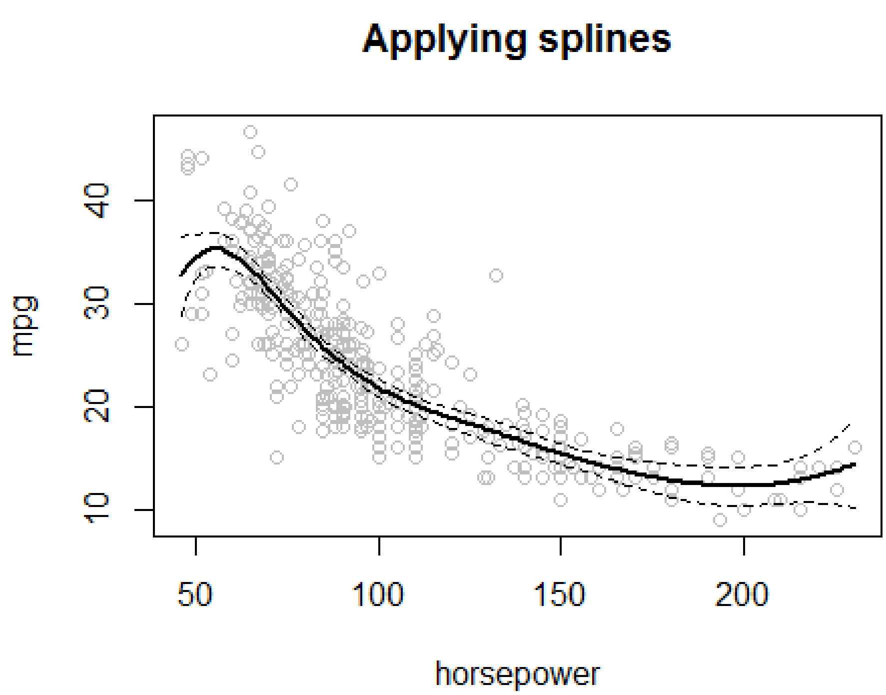

```r
#using  ANOVA to analyze variance.
fit.1=lm(mpg~horsepower, data=Auto)
fit.2=lm(mpg~poly(horsepower,2), data=Auto)
fit.3=lm(mpg~poly(horsepower,3), data=Auto)
fit.4=lm(mpg~poly(horsepower,4), data=Auto)

anova(fit.1,fit.2,fit.3,fit.4)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, 2)
## Model 3: mpg ~ poly(horsepower, 3)
## Model 4: mpg ~ poly(horsepower, 4)
##   Res.Df    RSS Df Sum of Sq        F Pr(>F)
## 1    390 9385.9
## 2    389 7442.0  1   1943.89 101.6666 <2e-16 ***
## 3    388 7426.4  1     15.59   0.8155 0.3670
## 4    387 7399.5  1     26.91   1.4076 0.2362
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

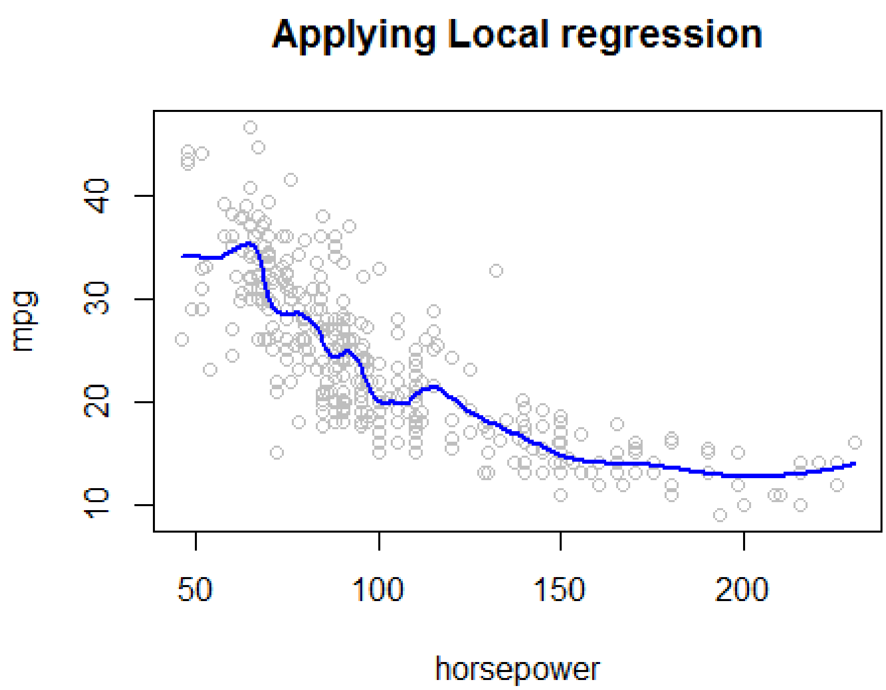
```
print(paste0("Quadratic model seems to be a good fit"))
```

```
## [1] "Quadratic model seems to be a good fit"
```

```
#Applying spline model
library(splines)
fit=lm(mpg~bs(horsepower, df=6),data=Auto)
pred=predict(fit,newdata=list(horsepower=hp.grid),se=T)
plot(horsepower,mpg, col="gray")
title('Applying splines')
lines(hp.grid, pred$fit,lwd=2)
lines(hp.grid, pred$fit+2*pred$se,lty="dashed")
lines(hp.grid, pred$fit-2*pred$se,lty="dashed")
```



```r
#Applying local regression
fit=loess(mpg~horsepower, span=0.2, data=Auto)
pred=predict(fit, newdata=data.frame(horsepower=hp.grid))
plot(horsepower,mpg, col="gray")
title('Applying Local regression')
lines(hp.grid, pred,col="blue", lwd=2)
```



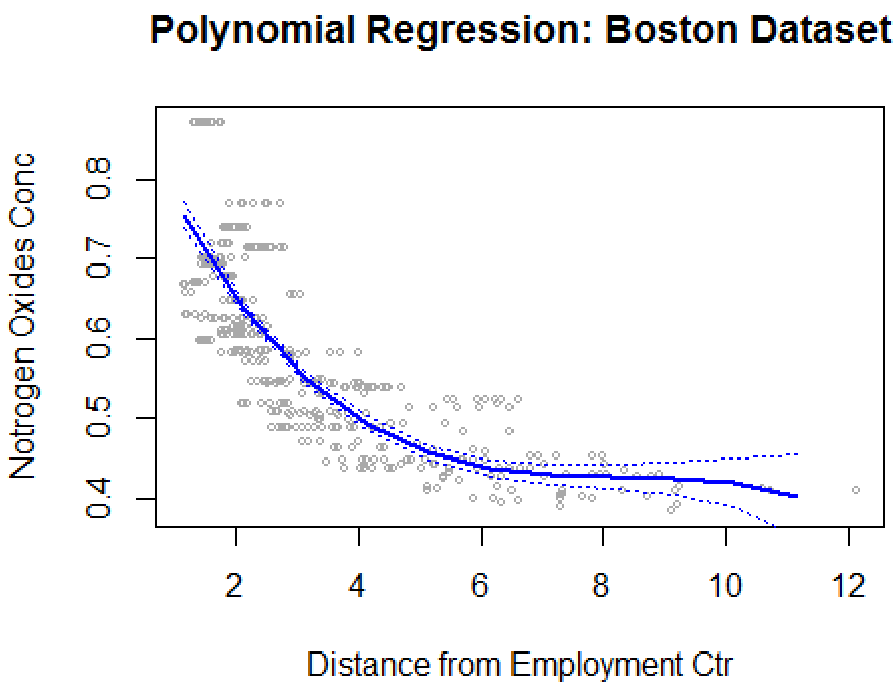
Q5

- 5a.

```r
library(MASS)
attach(Boston)

fit.poly=glm(nox~poly(dis,3))
print(paste0("Coefficient of the polynomial eqn"))

## [1] "Coefficient of the polynomial eqn"

coef(fit.poly)

##   (Intercept) poly(dis, 3)1 poly(dis, 3)2 poly(dis, 3)3
##     0.5546951    -2.0030959     0.8563300    -0.3180490

summary(fit.poly)
```

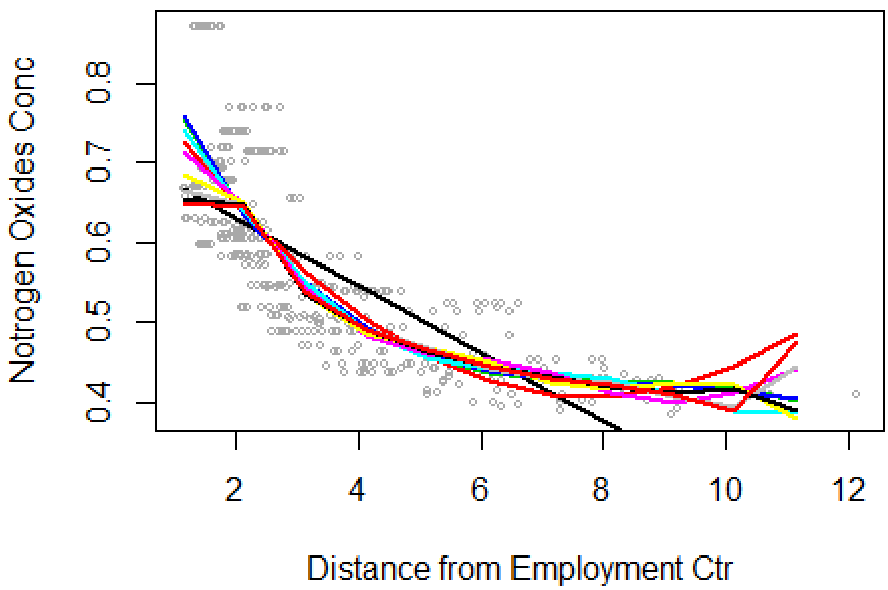
```
##
## Call:
## glm(formula = nox ~ poly(dis, 3))
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.121130  -0.040619  -0.009738   0.023385   0.194904
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.554695   0.002759 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
##
## (Dispersion parameter for gaussian family taken to be 0.003852802)
##
##     Null deviance: 6.7810  on 505  degrees of freedom
## Residual deviance: 1.9341  on 502  degrees of freedom
## AIC: -1370.9
##
## Number of Fisher Scoring iterations: 2
```

```
  dislims=range(Boston$dis)
  dis.grid=seq(from=dislims[1],to=dislims[2])
  preds=predict(fit.poly,newdata=list(dis=dis.grid), se=T)
  se.bands=cbind(preds$fit+2*preds$se,preds$fit-2*preds$se.fit)

  plot(dis,nox, xlim=dislims,xlab="Distance from Employment Ctr",
ylab="Notrogen Oxides Conc", cex=0.5, col="darkgrey")
  title("Polynomial Regression: Boston Dataset")
  lines(dis.grid, preds$fit, lwd=2, col="blue")
  matlines(dis.grid, se.bands, lwd=1, col="blue", lty=3)
```



- 5b.

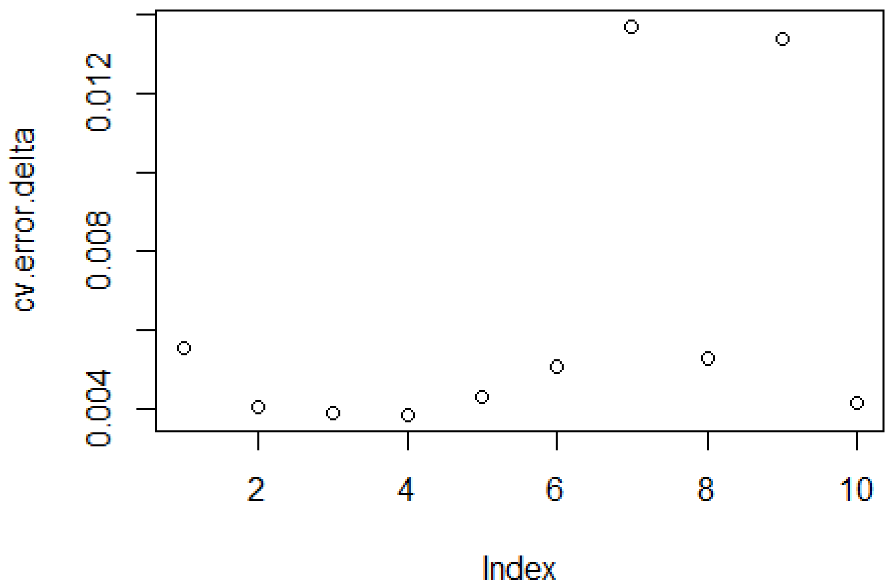
```
  fit=matrix(0,nrow=10,ncol=1)
plot(dis,nox, xlim=dislims,xlab="Distance from Employment Ctr",
ylab="Notrogen Oxides Conc", cex=0.5, col="darkgrey")
for (i in 1:10){
  lm.fit=glm(nox~poly(dis,i),data=Boston)
  fit[i]=sum(lm.fit$residuals^2)
  preds=predict(lm.fit,newdata=list(dis=dis.grid), se=T)
  lines(dis.grid, preds$fit, lwd=2, col=i)
}
```




```
fit

##             [,1]
##  [1,] 2.768563
##  [2,] 2.035262
##  [3,] 1.934107
##  [4,] 1.932981
##  [5,] 1.915290
##  [6,] 1.878257
##  [7,] 1.849484
##  [8,] 1.835630
##  [9,] 1.833331
## [10,] 1.832171
```

- 5c.

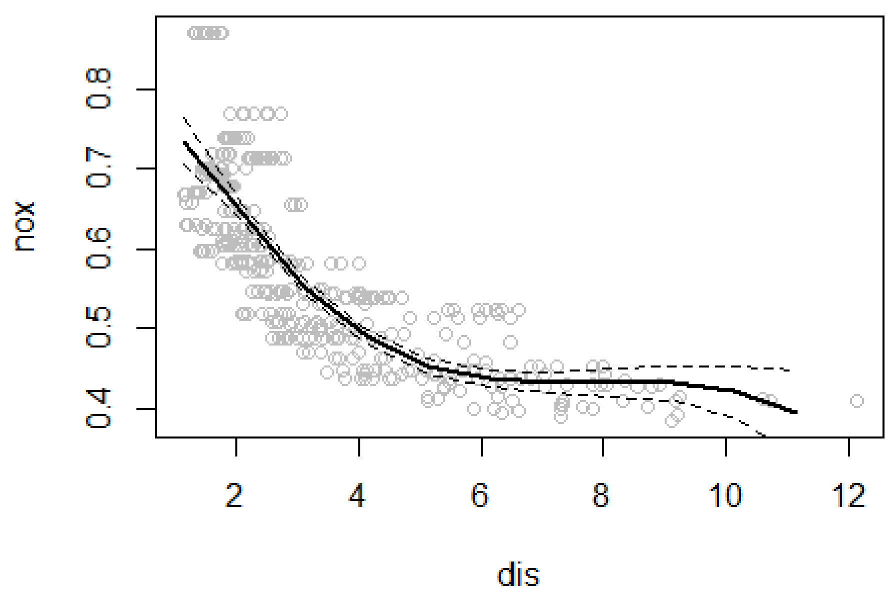
```
set.seed(1)
cv.error.delta=rep(NA,10)
for(i in 1:10){
  fit.poly=glm(nox~poly(dis,i),data=Boston)
  cv.error=cv.glm(Boston,fit.poly,K=10)
  cv.error.delta[i]=cv.error$delta[1]
}
plot(cv.error.delta)
```



- Degree 4 polynomial is the simplest model with lowest Cross-Validation error.
- A liner model does not fit the data, hence it has higher error. The quadratic and cubic models are comparable as the difference in error between them on a 10 fold CV set is small. Degree 4 polynomial shows the smallest error.
- For polynomials>4, the residual errors are high implying that the model fits to the training data but gives poor results on the CV test data.
- 5d.

```
  library(splines)
fit=lm(nox~bs(dis,df=4),data=Boston)
dislims=range(Boston$dis)
dis.grid=seq(from=dislims[1],to=dislims[2])
summary(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|)
```

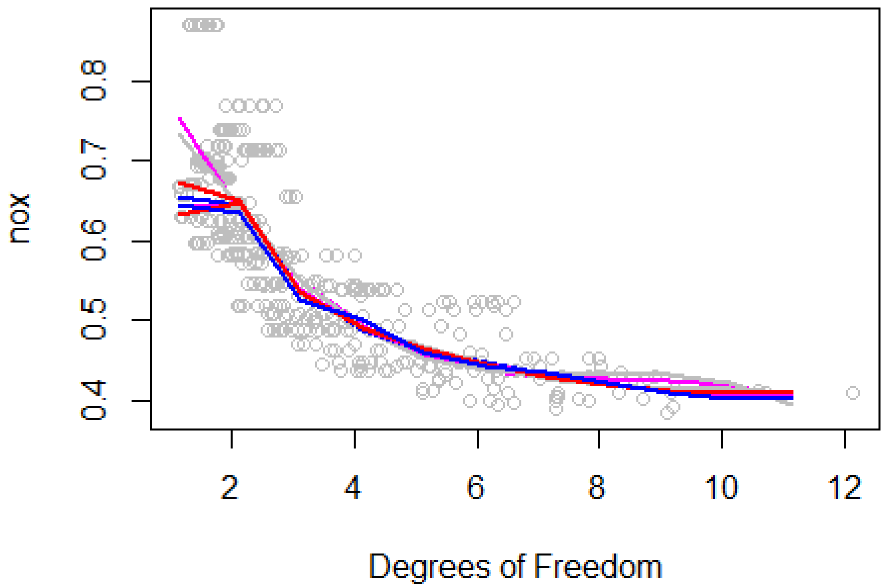
```
## (Intercept)       0.73447    0.01460  50.306  < 2e-16 ***
## bs(dis, df = 4)1 -0.05810    0.02186  -2.658  0.00812 **
## bs(dis, df = 4)2 -0.46356    0.02366 -19.596  < 2e-16 ***
## bs(dis, df = 4)3 -0.19979    0.04311  -4.634 4.58e-06 ***
## bs(dis, df = 4)4 -0.38881    0.04551  -8.544  < 2e-16 ***
## ---
## 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

pred=predict(fit,newdata=list(dis=dis.grid), se=T)
plot(dis,nox, col="gray")
lines(dis.grid, pred$fit, lwd=2)
lines(dis.grid, pred$fit+2*pred$se, lty="dashed")
lines(dis.grid, pred$fit-2*pred$se, lty="dashed")
```



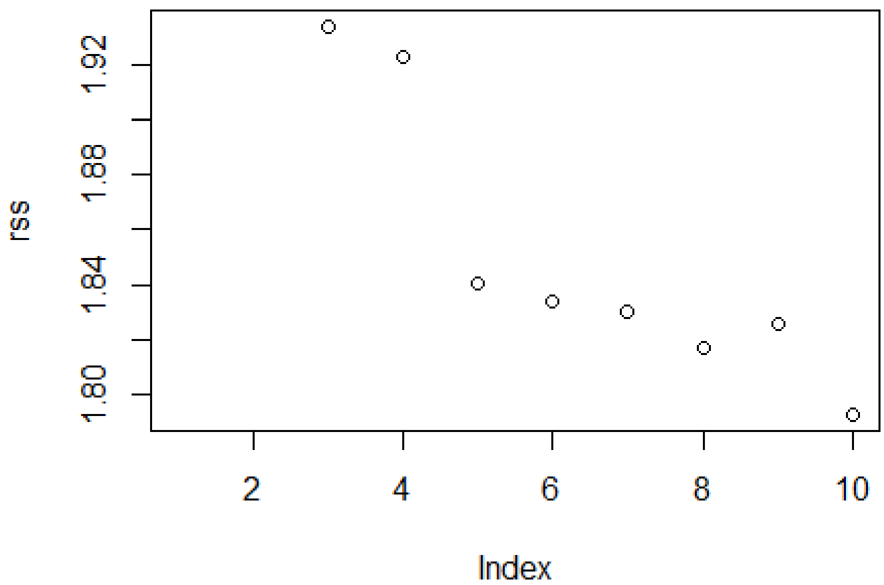
- Knots were chosen by specifying the df attribute that chose the knots at uniform quantile of data. In this case, the knots were at 25th, 50th and 75th quantiles
- 5e.

```
plot(dis,nox, col="gray", xlab="Degrees of Freedom")
rss=rep(NA,10)
for(i in 3:10){
  lm.fit=lm(nox~bs(dis,df=i),data=Boston)
```

```
  pred=predict(lm.fit,newdata=list(dis=dis.grid), se=T)
  lines(dis.grid, pred$fit, lwd=2, col=i*10)
  rss[i]=sum(lm.fit$residuals^2)
}
```

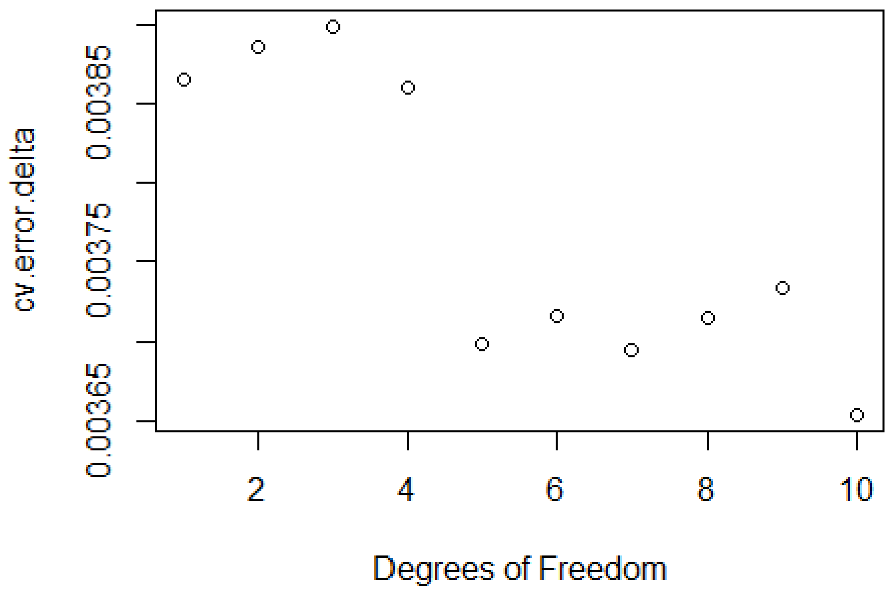



```
plot(rss)
```



- RSS decreases monotonically as we increase degrees of freedom.Lowest RSS is with df=10
- 5f

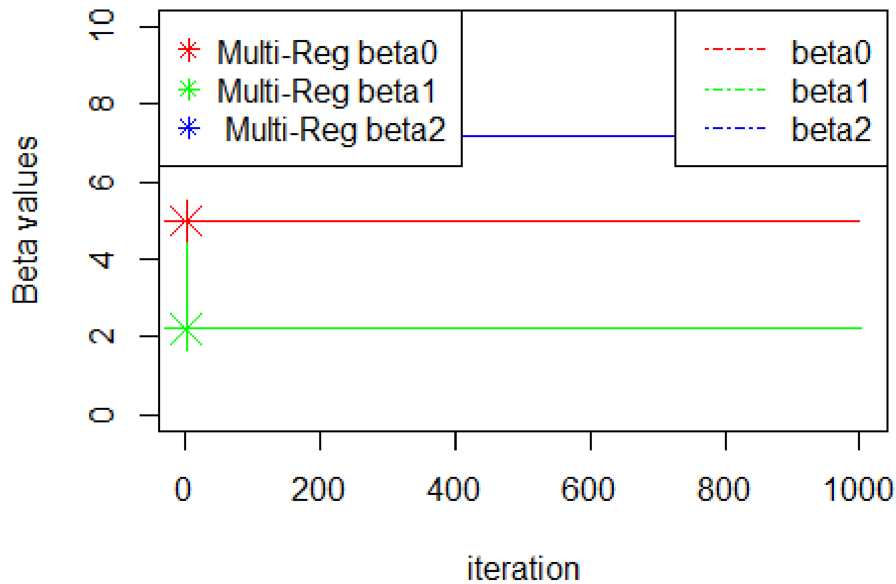
```
set.seed(1)
cv.error.delta=rep(NA,10)
for(i in 1:10){
  fit.spline=glm(nox~bs(dis,df=i),data=Boston)
  cv.error=cv.glm(Boston,fit.spline,K=10)
  cv.error.delta[i]=cv.error$delta[1]
}
plot(cv.error.delta, xlab="Degrees of Freedom")
```



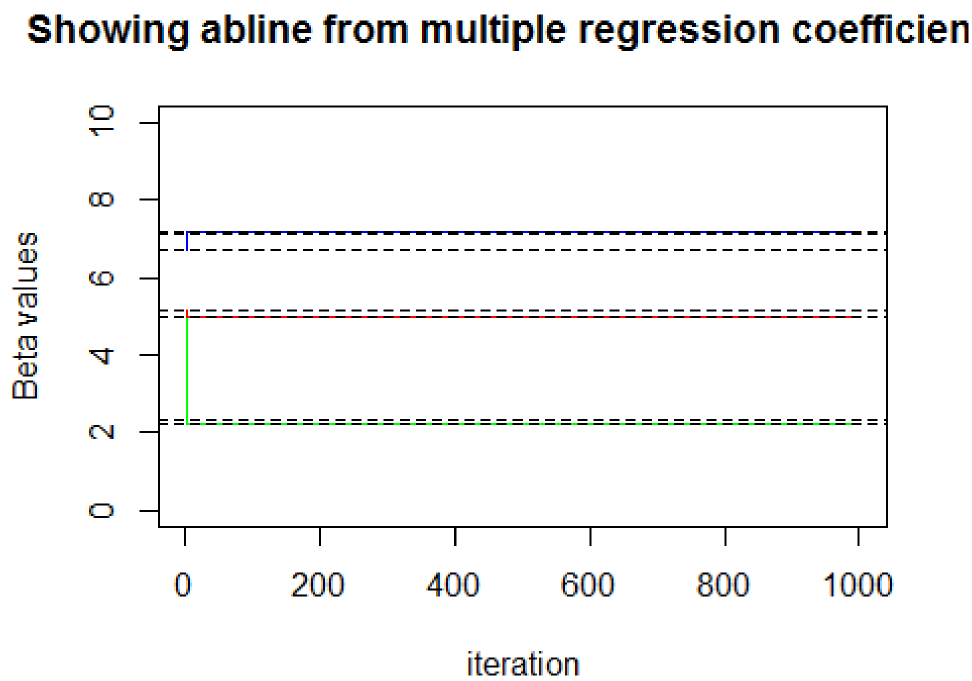
- Lowest error obtained through cross validation is for df=10.
- As we increase the degrees of freedome, initially, till we reach df=3, there is increase in the RSS. However, error starts decreasing after that till df=5. Errors fluctuate a bit, before getting the lowest error at df=10.

Q6

```
            x1=rnorm(100)
            x2=rnorm(100)
            eps=rnorm(100)
            y=5+2*x1+7*x2+eps

            beta0=rep(NA,1000)
            beta1=rep(NA,1000)
            beta2=rep(NA,1000)

            beta1[1]=5

          for(i in 1:1000){
            a=y-beta1[i]*x1
            beta2[i]=lm(a~x2)$coef[2]
            beta0[i]=lm(a~x2)$coef[1]

           a=y-beta2[i]*x2
            beta1[i+1]=lm(a~x1)$coef[2]

          }

          plot(1:1001,beta1, col="green", type='l', ylim=c(0,10),
xlab="iteration", ylab="Beta values")
          lines(1:1000,beta0, col="red")
          lines(1:1000,beta2, col="blue")


          u=lm(y~x1+x2)
          points(coef(u)[1],col="red", pch=8, cex=2)
          points(coef(u)[2],col="green", pch=8, cex=2)
          points(coef(u)[3],col="blue", pch=8, cex=2)
         legend("topright",c("beta0","beta1","beta2"), lty=4,col =
c("red","green","blue"))
          legend("topleft",c("Multi-Reg beta0","Multi-Reg beta1"," Multi-Reg
beta2"), pch=8,col = c("red","green","blue"))
```




```
        plot(1:1001,beta1, col="green", type='l', ylim=c(0,10),
xlab="iteration", ylab="Beta values")
        title("Showing abline from multiple regression coefficients.")
        lines(1:1000,beta0, col="red")
        lines(1:1000,beta2, col="blue")
        abline(h=beta0, lty="dashed")
        abline(h=beta1, lty="dashed")
        abline(h=beta2, lty="dashed")
```




```
          print(paste0("Within 2nd approximations the backfitting answers
were good estimates"))

## [1] "Within 2nd approximations the backfitting answers were good
estimates"

          beta0[1:5]

## [1] 5.149512 5.014518 5.009577 5.009396 5.009389

          beta1[1:5]

## [1] 5.000000 2.319657 2.221546 2.217955 2.217824

          beta2[1:5]

## [1] 6.693492 7.145315 7.161853 7.162459 7.162481
```