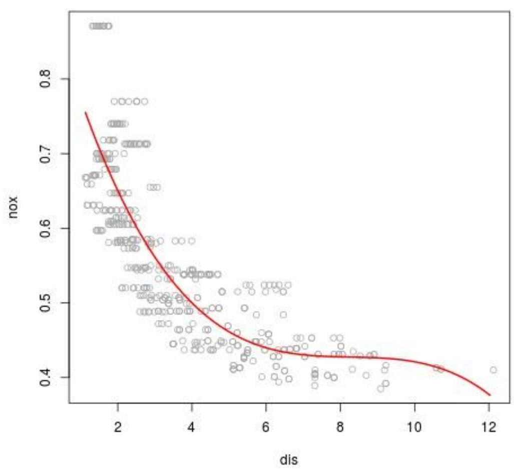
9)
a)

```
> set.seed(1)
> library(MASS)
> attach(Boston)
> 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))
> jpeg("9a.jpg")
> plot(nox ~ dis, data = Boston, col = "darkgrey")
> lines(dis.grid, lm.pred, col = "red", lwd = 2)
> dev.off()
```



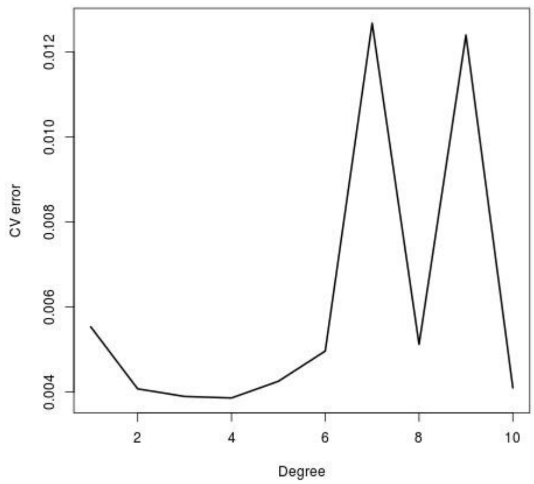
Summary shows that all polynomial terms are significant while predicting nox using dis. Plot shows a smooth curve fitting the data fairly well.

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

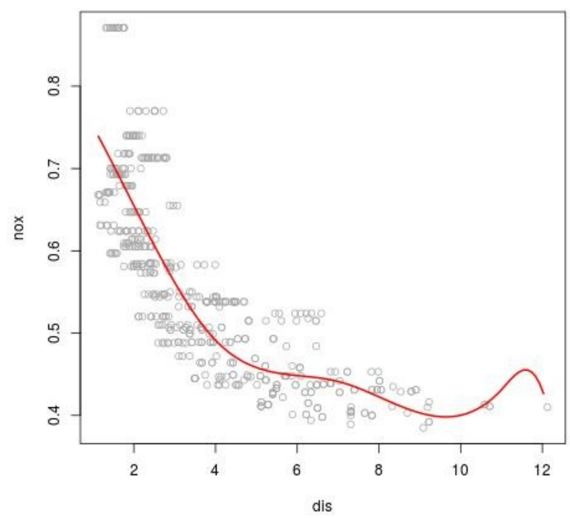
```
> jpeg("9b.jpg")
> plot(1:10, all.deltas, xlab = "Degree", ylab = "CV error", type = "l", pch = 20, lwd = 2)
> dev.off()
```

A 10-fold CV shows that the CV error reduces as we increase degree from 1 to 3, stay almost constant till degree 5, and the starts increasing for higher degrees. We pick 4 as the best polynomial degree.

d)
We see that dis has limits of about 1 and 13 respectively. We split this range in roughly equal 4 intervals and establish knots at [4,7,11]. Note: bs function in R expects either df or knots argument. If both are specified, knots are ignored.

```
> 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))
> jpeg("9c.jpg")
> plot(nox ~ dis, data = Boston, col = "darkgrey")
> lines(dis.grid, sp.pred, col = "red", lwd = 2)
> dev.off()



The summary shows that all terms in spline fit are significant. Plot shows that the spline fits data well except at the extreme values of disdis, (especially dis>10).

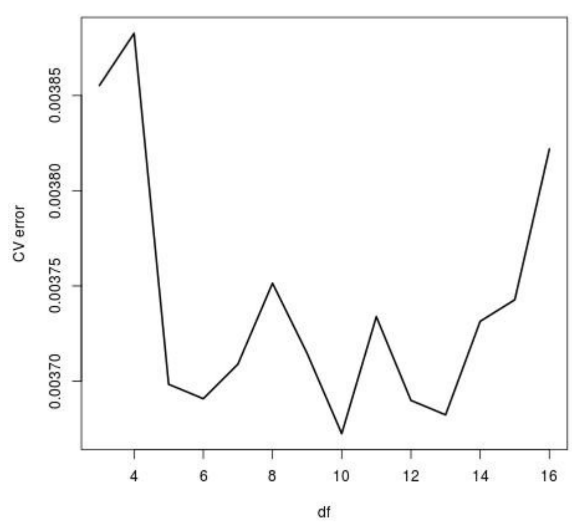
e)
We fit regression splines with dfs between 3 and 16.

```
> 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)
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]
+ }

> jpeg("9f.jpg")
> plot(3:16, all.cv[-c(1, 2)], lwd = 2, type = "l", xlab =
"df", ylab = "CV error")
> dev.off()
```



CV error is more jumpy in this case, but attains minimum at df=10. We pick 1010 as the optimal degrees of freedom.

11)

a)

We create variables according to the equation $Y=-2.1+1.3X_1+0.54X_2$.

```
> set.seed(1)
> X1 = rnorm(100)
> X2 = rnorm(100)
> eps = rnorm(100, sd = 0.1)
> Y = -2.1 + 1.3 * X1 + 0.54 * X2 + eps
```

b)

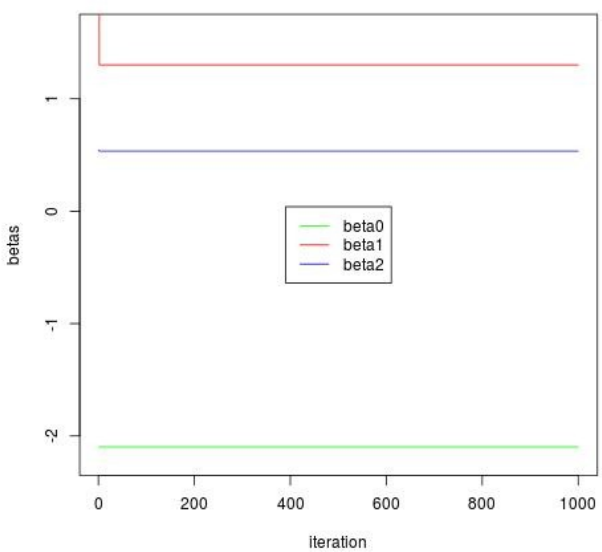
Create a list of 1000 beta^0, beta^1 and beta^2. Initialize first of the β^1 to 10.

```
> beta0 = rep(NA, 1000)
> beta1 = rep(NA, 1000)
> beta2 = rep(NA, 1000)
> beta1[1] = 10
```

c, d, e)

Accumulate results of 1000 iterations in the beta arrays.

```
> for (i in 1:1000) {
+     a = Y - beta1[i] * X1
+     beta2[i] = lm(a ~ X2)$coef[2]
+     a = Y - beta2[i] * X2
+     lm.fit = lm(a ~ X1)

+     if (i < 1000) {+         beta1[i + 1] =
lm.fit$coef[2]
+     }
+     beta0[i] = lm.fit$coef[1]
+ }

> jpeg("9cde.jpg")
> plot(1:1000, beta0, type = "l", xlab = "iteration",
ylab = "betas", ylim = c(-2.2,
+     1.6), col = "green")
> lines(1:1000, beta1, col = "red")
> lines(1:1000, beta2, col = "blue")
> legend("center", c("beta0", "beta1", "beta2"), lty
= 1, col = c("green", "red",
+     "blue"))
> dev.off()
```



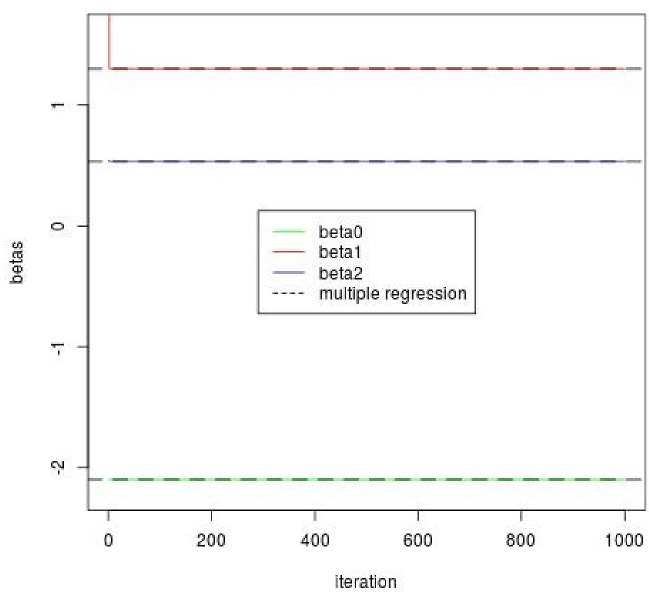
The coefficients quickly attain their least square values.

f)

```
> lm.fit = lm(Y ~ X1 + X2)
```

```
> jpeg("9f.jpg")
> plot(1:1000, beta0, type = "l", xlab = "iteration", ylab = "betas", ylim = c(-2.2,
+     1.6), col = "green")
> lines(1:1000, beta1, col = "red")
> lines(1:1000, beta2, col = "blue")
> abline(h = lm.fit$coef[1], lty = "dashed", lwd = 3, col = rgb(0, 0, 0, alpha = 0.4))
> abline(h = lm.fit$coef[2], lty = "dashed", lwd = 3, col = rgb(0, 0, 0, alpha = 0.4))
> abline(h = lm.fit$coef[3], lty = "dashed", lwd = 3, col = rgb(0, 0, 0, alpha = 0.4))
> legend("center", c("beta0", "beta1", "beta2", "multiple regression"), lty = c(1,
+     1, 1, 2), col = c("green", "red", "blue", "black"))
> dev.off()
```

Dotted lines show that the estimated multiple regression coefficients match exactly with the coefficients obtained using backfitting.



g)
When the relationship between Y and X's is linear, one iteration is sufficient to attain a good approximation of true regression coefficients.