Devon Zuegel // 3 Nov 2015 // STATS 202

Problem Set 5

web.stanford.edu/class/stats202/content/viewhw.html?hw5

Problem 1

Chapter 7, Exercise 2 (p. 298)

When `λ = ∞`, the penalty term dominates the objective function that we are minimizing, and the (least square) solution for `g` comes from minimizing the sum of squares with the constraint `g''(x) = 0` for all `x`. Here, `m` makes a difference.

However, when `λ = 0`, then the minimizing function interpolates the data; $y_i = f(x_i). Note that in this case the function does not have to be linear.

$$\hat{g} = \underset{g}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g^{(m)}(x)^2 \cdot dx \right)$$

Notes

- $\lambda \rightarrow$ non-negative tuning param
- $\sum (y_i - g(x_i))^2 \rightarrow$ loss fn that encourages g to fit the data well
- $\lambda \int g^{(m)}(x)^2 \cdot dx \rightarrow$ penalty term that punishes variability of g
- second deriv. ($m=2$) is a measure of its roughness
  - 2nd derivative of a straight line is 0
- when $\lambda = 0$, the penalty term has no impact
  - $\hookrightarrow$ g will be very jumpy, exactly interpolating the training observations
- when $\lambda \rightarrow \infty$, g will be perfectly smooth, a straight line that passes as close as poss. to the training pts
  - $\hookrightarrow$ g will be the linear least squares line (the loss fn effectively minimizes the RSS)
- for intermediate values of λ, g approx. the training observation but will be somewhat smooth
  - $\hookrightarrow$ λ effectively controls the bias-variance tradeoff

ⓐ $\lambda = \infty$, $m = 0$ $\longrightarrow$ 0^{th} derivative is the fn $g(x)$ itself, so we're basically punishing any non-constant behavior within the fn

$g(x)$ is a constant, horizontal line:

$\Rightarrow$ $g(x) = k$ where $k \in \mathbb{R}$

ⓑ $\lambda = \infty$, $m = 1$ $\longrightarrow$ goal is to minimize the area under the first derivative, so $g(x)$ would be quadratic

$g(x) = kx^2$

ⓒ $\lambda = \infty$, $m = 2$ $\longrightarrow$ goal is to min. the area under the curve of the 2nd derivative, so $g(x)$ would be cubic.

$g(x) = k \cdot x^3$

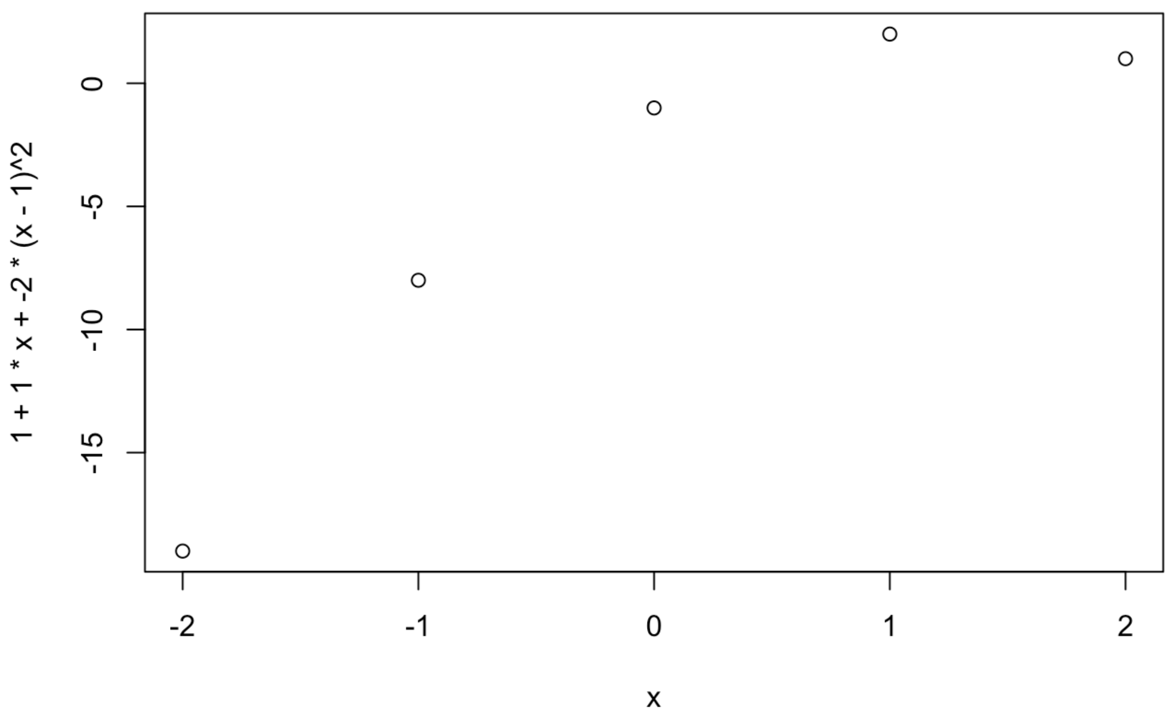
ⓓ $\lambda = \infty$, $m = 3$ $\longrightarrow$ $g(x) = k \cdot x^4$ (similar to last 2)

ⓔ $\lambda = 0$, $m = 3$ $\longrightarrow$ The penalty term has no impact. g will be very jumpy.

Problem 2

Chapter 7, Exercise 3 (p. 298)

```
x = -2:2
plot(x, 1 + 1*x + -2*(x-1)^2)
```



Problem 3

Chapter 7, Exercise 5 (p. 299)

Part A

$\widehat{g}_2$ would likely have the smaller training RSS since it's a higher-order polynomial.

Part B

$\widehat{g} *1*
$wouldlikelyhavethesmallertestRSS. The higher - order polynomial function$
\widehat{g} 2$ has an extra degree of freedom, causing it to possibly overfit the training data, resulting in worse test results.

Part C

The two functions are equivalent when `λ = 0`.

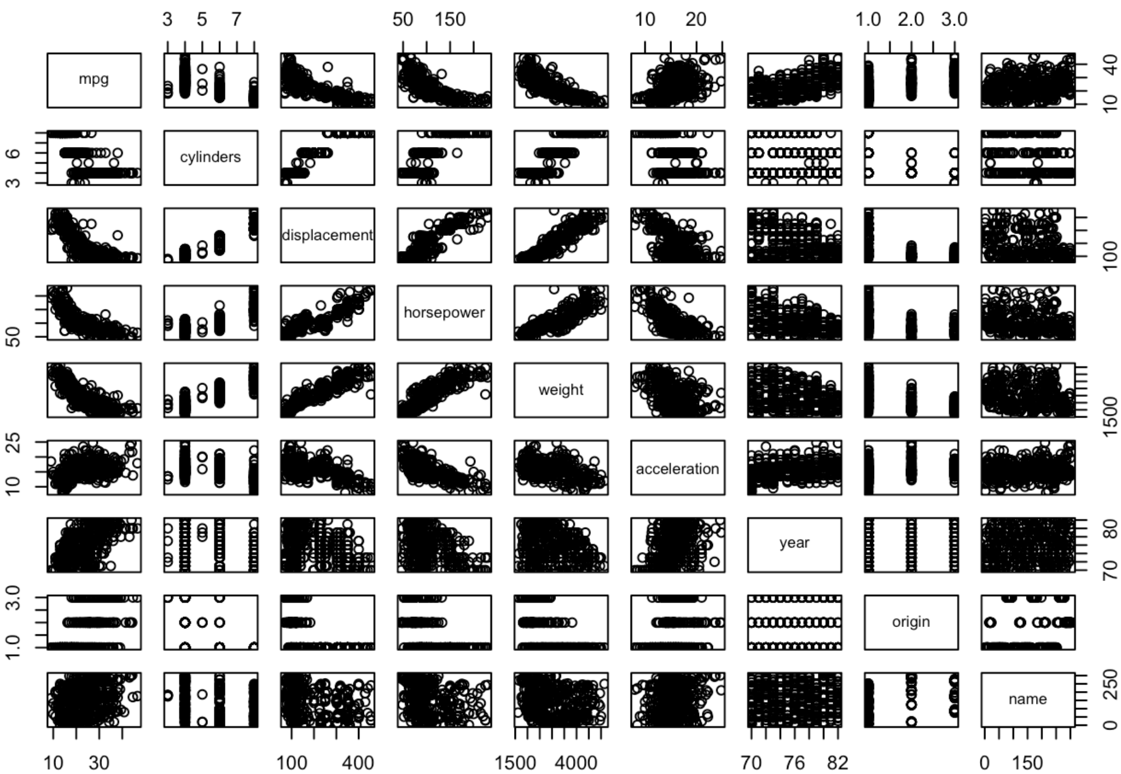
Problem 4

Chapter 7, Exercise 8 (p. 299). Find at least one non-linear estimate which does better than linear regression, and justify this using a t-test or by showing an improvement in the cross-validation error with respect to a linear model. You must also produce a plot of the predictor X vs. the non-linear estimate $\hat{f}(X)$

```
library(ISLR)
library(gam)
df = Auto
summary(df)
```

```
##       mpg          cylinders      displacement     horsepower   
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0  
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0  
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5  
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5  
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0  
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0  
##                                                                 
##      weight      acceleration        year           origin     
##  Min.   :1613   Min.   : 8.00   Min.   :70.00   Min.   :1.000  
##  1st Qu.:2225   1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000  
##  Median :2804   Median :15.50   Median :76.00   Median :1.000  
##  Mean   :2978   Mean   :15.54   Mean   :75.98   Mean   :1.577  
##  3rd Qu.:3615   3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000  
##  Max.   :5140   Max.   :24.80   Max.   :82.00   Max.   :3.000  
##                                                                 
##                  name    
##  amc matador       :  5  
##  ford pinto        :  5  
##  toyota corolla    :  5  
##  amc gremlin       :  4  
##  amc hornet        :  4  
##  chevrolet chevette:  4  
##  (Other)           :365  
```

```
plot(df)
```




```
fit = gam(mpg ~ s(horsepower, 3) + s(displacement, 3), data = df)
summary(fit)
```

```
##
## Call: gam(formula = mpg ~ s(horsepower, 3) + s(displacement, 3), data = df)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -11.0177  -2.1818  -0.5872   2.1790  16.8284
##
## (Dispersion Parameter for gaussian family taken to be 15.5188)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 5974.754 on 385.0001 degrees of freedom
## AIC: 2196.27
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                      Df  Sum Sq Mean Sq F value    Pr(>F)
## s(horsepower, 3)      1 15513.3 15513.3 999.641 < 2.2e-16 ***
## s(displacement, 3)    1   692.9   692.9  44.649 8.284e-11 ***
## Residuals           385  5974.8    15.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                    Npar Df Npar F     Pr(F)
## (Intercept)
## s(horsepower, 3)         2 20.761 2.740e-09 ***
## s(displacement, 3)       2 21.652 1.227e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Problem 5

Chapter 7, Exercise 9 (p. 299). In part (d), the book instructs you to fit a regression or cubic spline with 4 degrees of freedom. Use 7 degrees of freedom instead (3 knots)

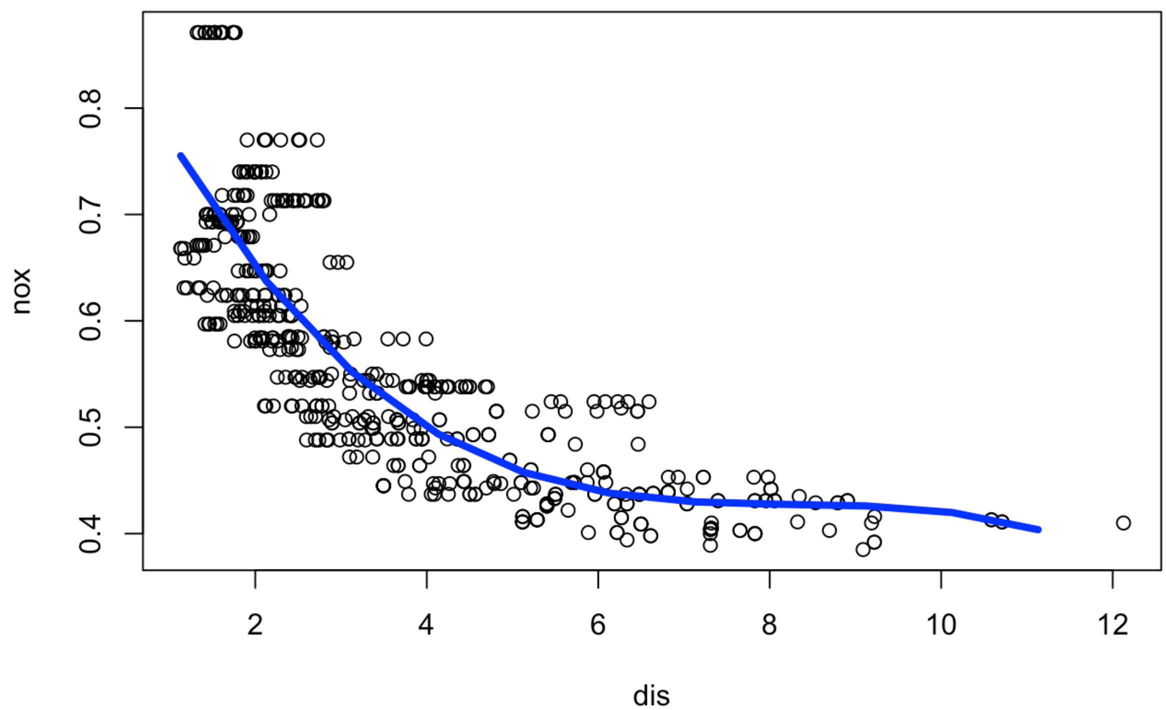
Part A

```
set.seed(8)
library(MASS)
library(boot)
library(splines)
df = Boston

# Cubic polynomial regression to predict nox using dis.
fit = lm(nox ~ poly(dis, 3), data = df)
summary(fit)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = df)
##
## 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
```

```
dis.range = range(df$dis)
dis.seq = seq(from = dis.range[1], to = dis.range[2])
prediction = predict(fit, list(dis = dis.seq))
plot(nox ~ dis, data = df)
lines(dis.seq, prediction, lwd = 4, col = 'blue')
```



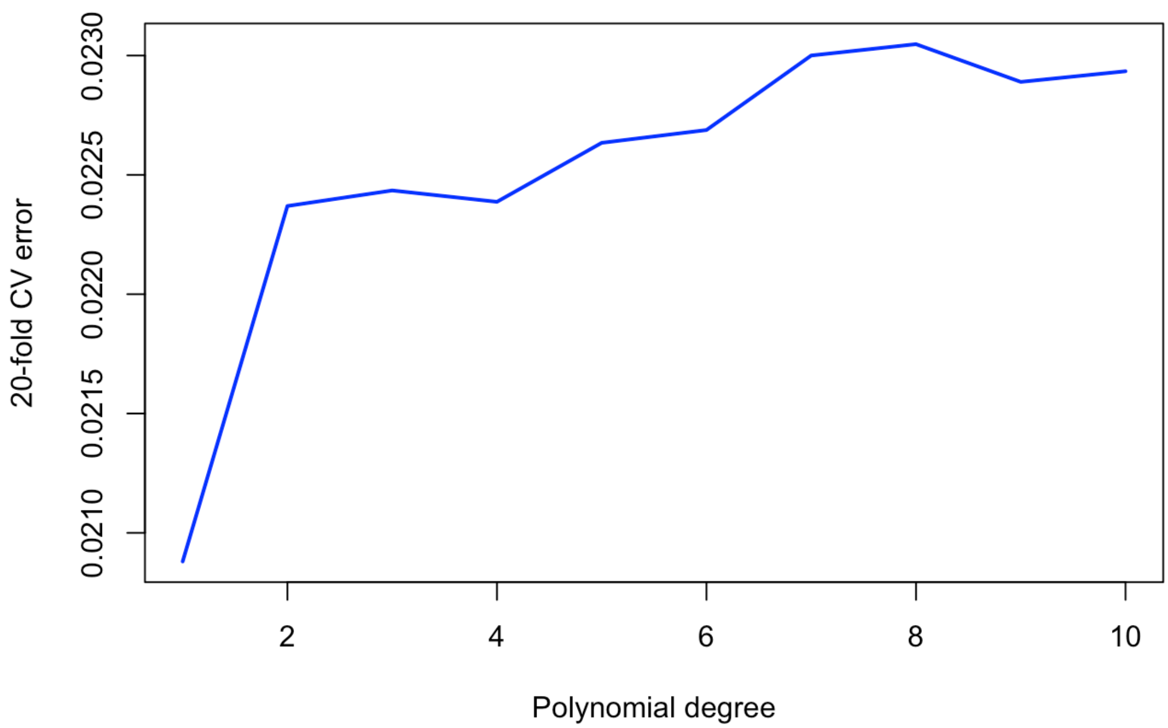
Part B

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

```
## [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484
## [8] 1.835630 1.833331 1.832171
```

Part C

```
NUM_FOLDS = 20
reps = rep(NA, N)
for (i in 1:N) {
    fit = glm(df$nox ~ poly(df$dis, i), data = df)
    reps[i] = cv.glm(df, fit, K = NUM_FOLDS)$delta[2]
}
plot(1:N, reps, xlab = 'Polynomial degree', ylab = '20-fold CV error', lwd = 2, col = 'blue', type = 'l')
```



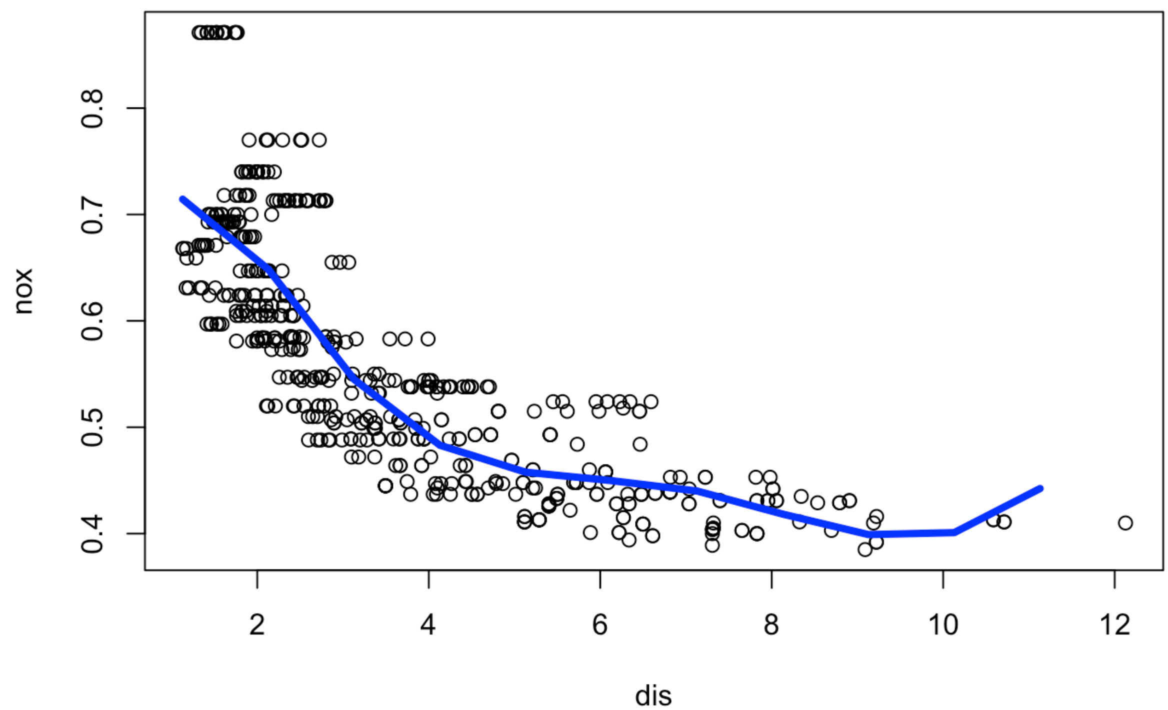
We pick `1` as the best polynomial degree, since it has the lowest cross-validation error.

Part D

```
fit = lm(nox ~ bs(dis, df = 4, knots = c(3, 7, 11)), data = df)
summary(fit)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4, knots = c(3, 7, 11)), data = df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.130710 -0.039850 -0.008357  0.027792  0.188518
##
## Coefficients:
##                                        Estimate Std. Error t value
## (Intercept)                            0.714346   0.015846  45.081
## bs(dis, df = 4, knots = c(3, 7, 11))1 -0.006626   0.024307  -0.273
## bs(dis, df = 4, knots = c(3, 7, 11))2 -0.296980   0.018293 -16.234
## bs(dis, df = 4, knots = c(3, 7, 11))3 -0.222840   0.033763  -6.600
## bs(dis, df = 4, knots = c(3, 7, 11))4 -0.379811   0.042317  -8.975
## bs(dis, df = 4, knots = c(3, 7, 11))5 -0.222959   0.086870  -2.567
## bs(dis, df = 4, knots = c(3, 7, 11))6 -0.304346   0.063378  -4.802
##                                       Pr(>|t|)
## (Intercept)                            < 2e-16 ***
## bs(dis, df = 4, knots = c(3, 7, 11))1   0.7853
## bs(dis, df = 4, knots = c(3, 7, 11))2  < 2e-16 ***
## bs(dis, df = 4, knots = c(3, 7, 11))3 1.05e-10 ***
## bs(dis, df = 4, knots = c(3, 7, 11))4  < 2e-16 ***
## bs(dis, df = 4, knots = c(3, 7, 11))5   0.0106 *
## bs(dis, df = 4, knots = c(3, 7, 11))6 2.08e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06137 on 499 degrees of freedom
## Multiple R-squared:  0.7229, Adjusted R-squared:  0.7196
## F-statistic:   217 on 6 and 499 DF,  p-value: < 2.2e-16
```

```r
prediction = predict(fit, list(dis = dis.seq))
plot(nox ~ dis, data = df)
lines(dis.seq, prediction, col = 'blue', lwd = 4)
```



I chose the knots based such that the data was roughly split into even pieces (since `dis`'s lower limit is ~1 and its upper limit is ~13).

Part E

```
NUM_DEGREES_OF_FREEDOM = 25
reps = rep(NA, NUM_DEGREES_OF_FREEDOM)

# Fit a regression spline for a range a degrees of freedom.
for (i in 1:NUM_DEGREES_OF_FREEDOM) {
    fit = lm(nox ~ bs(dis, df = i), data = df)
    reps[i] = sum(fit$residuals^2)
}
```

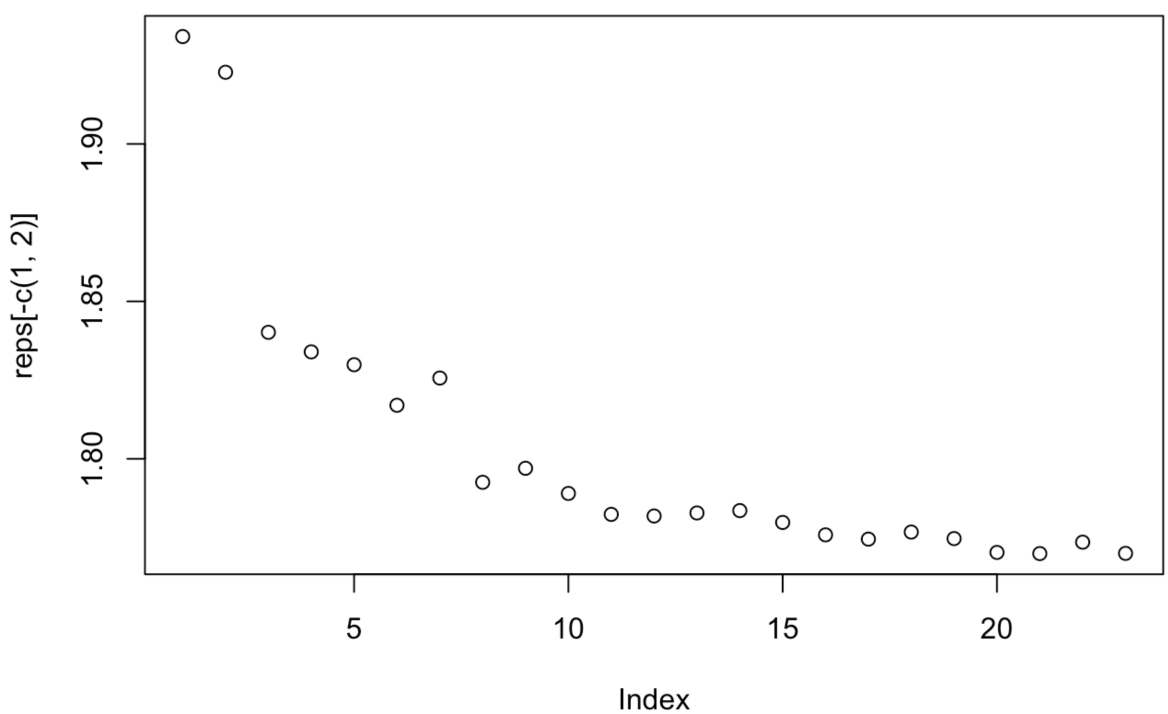
```
## Warning in bs(dis, df = i): 'df' was too small; have used 3
```

```
## Warning in bs(dis, df = i): 'df' was too small; have used 3
```

```
reps[-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
## [15] 1.779789 1.775838 1.774487 1.776727 1.774664 1.770263 1.769895
## [22] 1.773516 1.769957
```

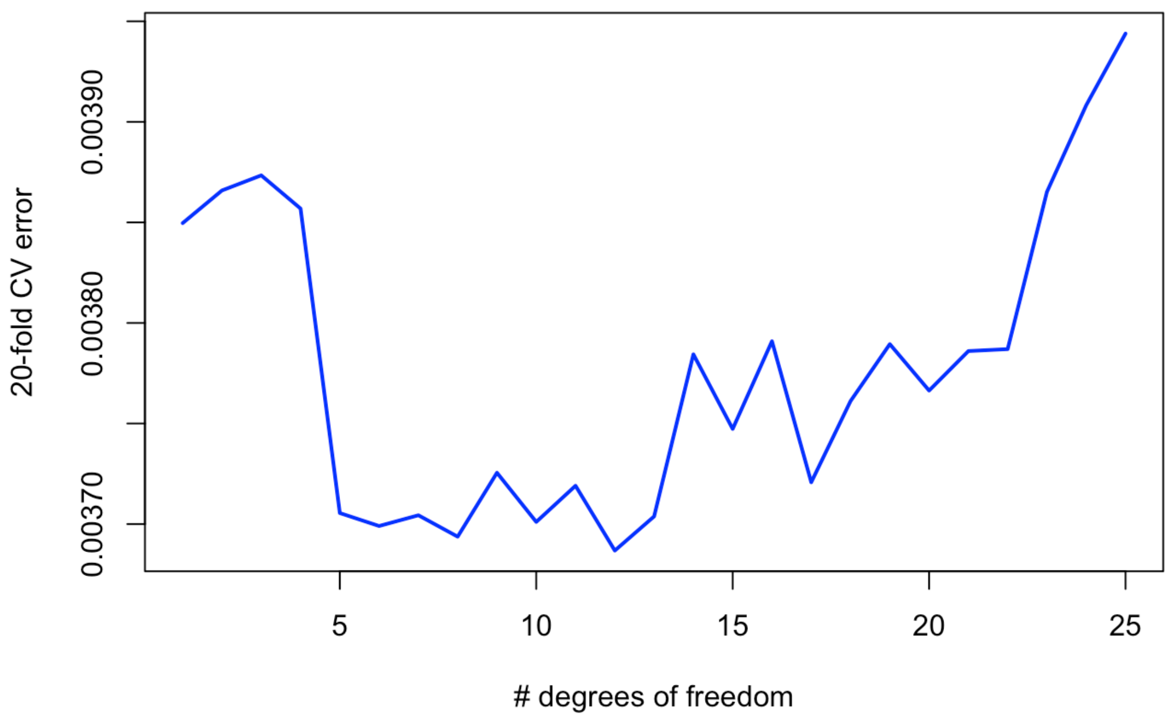
```
plot(reps[-c(1, 2)])
```



Decreases monotonically as the degree of freedom increases.

Part F

```
reps = rep(NA, NUM_DEGREES_OF_FREEDOM)
for (i in 1:NUM_DEGREES_OF_FREEDOM) {
    fit = glm(nox ~ bs(dis, df = i), data = df)
    reps[i] = cv.glm(df, fit, K = N)$delta[2]
}
plot(1:NUM_DEGREES_OF_FREEDOM, reps, lwd = 2, col = 'blue', xlab = '# degrees of freedom', ylab = '20
```



CV is at a minimum when we have 12 degrees of freedom.

Problem 6

Chapter 7, Exercise 11 (p. 300)

Part A

```
set.seed(51)    # Because 51 is edgy.

N  = 100
x1 = rnorm(N)
x2 = rnorm(N)
e  = rnorm(100, sd = 1)

y  = 1.2 + 2.3*x1 + 3.4*x2
```

Parts C & B

```
b1 = 5
a  = y - b1*x1
b2 = lm(a ~ x2)$coef[2]
```

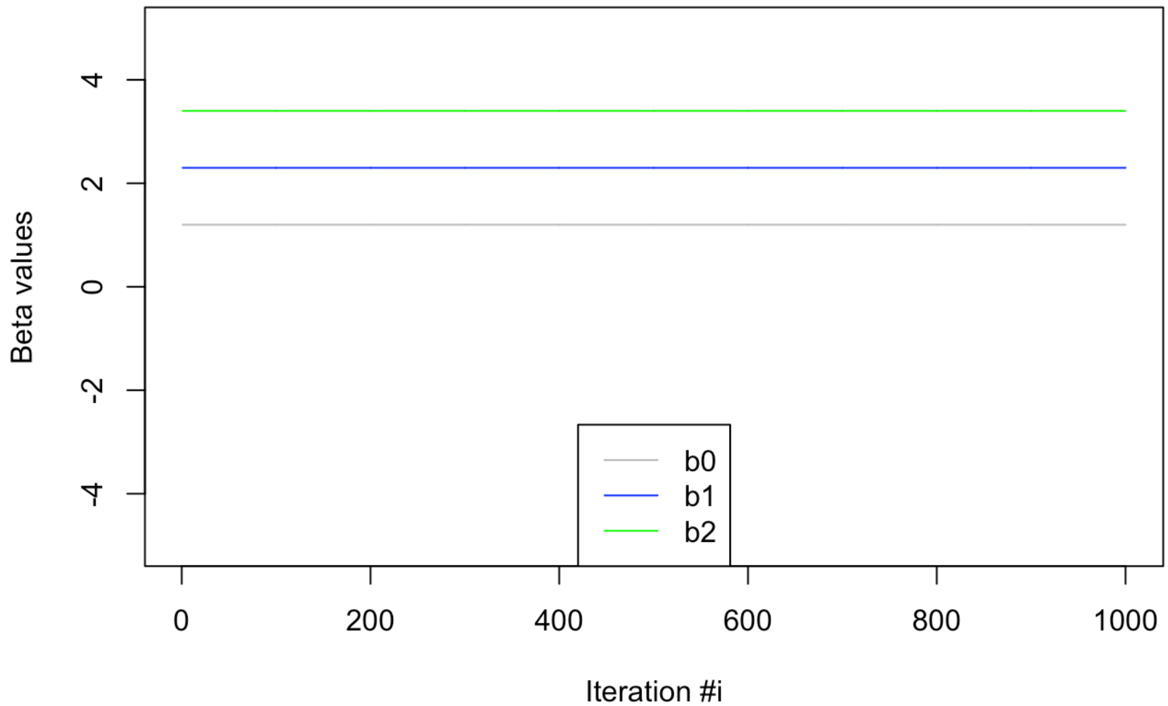
Part D

```
a  = y - b2*x2
b1 = lm(a ~ x1)$coef[2]
```

Part E

```
NUM_ITERATIONS = 1000
b0 = rep(NA, NUM_ITERATIONS)

for (i in 1:NUM_ITERATIONS) {
    a     = y - b1[i] * x1
    b2[i] = lm(a ~ x2)$coef[2]
    a     = y - b2[i] * x2
    fit   = lm(a ~ x1)
    if (i < NUM_ITERATIONS) {
        b1[i + 1] = fit$coef[2]
    }
    b0[i] = fit$coef[1]
}
plot (1:NUM_ITERATIONS, b0, type = 'l', xlab = 'Iteration #i', ylab = 'Beta values', ylim = c(-5, 5), cc
lines(1:NUM_ITERATIONS, b1, col = 'blue')
lines(1:NUM_ITERATIONS, b2, col = 'green')
legend('bottom', c('b0', 'b1', 'b2'), lty = 1, col = c('grey', 'blue', 'green'))
```

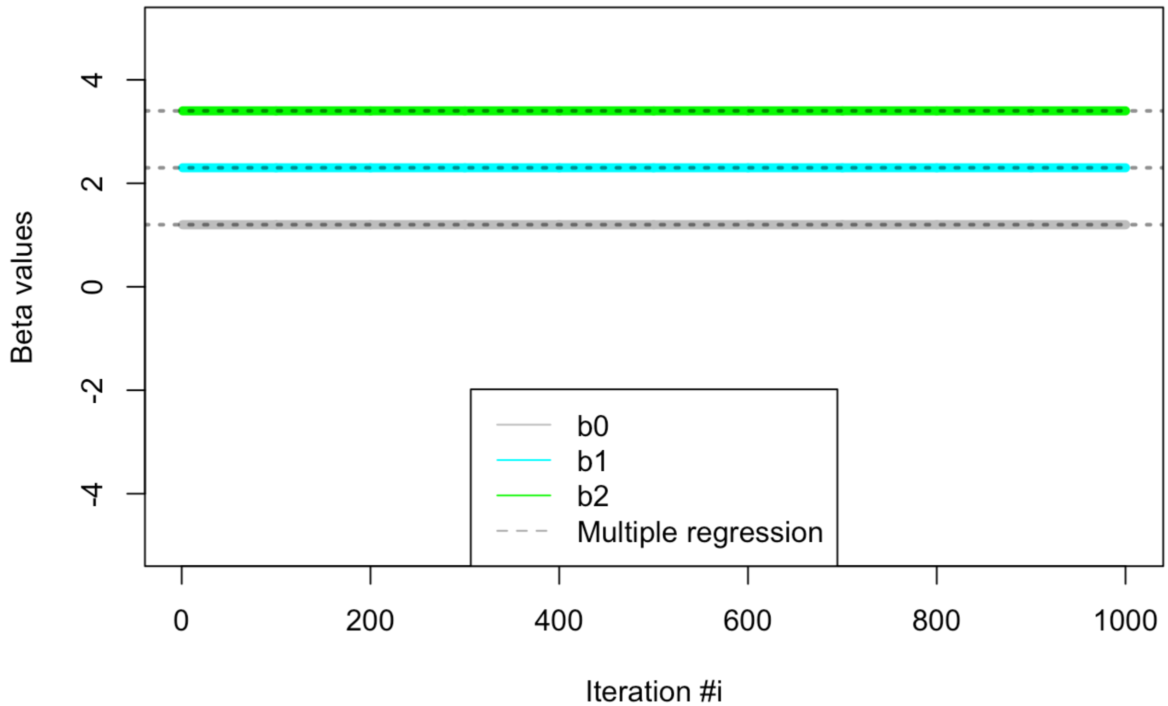


Part F

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).

```
NUM_ITERATIONS = 1000

lm.fit = lm(y ~ x1 + x2)
plot (1:NUM_ITERATIONS, b0, lwd = 5, type = 'l', xlab = 'Iteration #i', ylab = 'Beta values', ylim = c(-
lines(1:NUM_ITERATIONS, b1, lwd = 5, col = 'cyan')
lines(1:NUM_ITERATIONS, b2, lwd = 5, col = 'green')
abline(h = lm.fit$coef[1], lty = 'dotted', lwd = 2, col = rgb(0, 0, 0, alpha = 0.5))
abline(h = lm.fit$coef[2], lty = 'dotted', lwd = 2, col = rgb(0, 0, 0, alpha = 0.5))
abline(h = lm.fit$coef[3], lty = 'dotted', lwd = 2, col = rgb(0, 0, 0, alpha = 0.5))
legend('bottom', c('b0', 'b1', 'b2', 'Multiple regression'), lty = c(1, 1, 1, 2), col = c('grey', 'cyan', 'gre
```



The coefficients from multiple regression match the coefficients from backfitting perfectly.

Part G

Only really needed to do one iteration.