Statistics 216
Homework 3 Solutions, 62 total points.

1. *This problem is worth 14 points: one point for each correct answer (a, b, and c i-v) and one point for each explanation that correctly explains either the given answer or the correct answer.*

   (a) The best subset model will have the smallest training RSS. This is how best subset is defined.

   (b) It is impossible to know which model will have the smallest test RSS. Training is always done independently of the test set, so we could have anything happen in the test data.

      i. TRUE. In moving from k to $k+1$ variables, forward stepwise will pick one more variable to include.

      ii. TRUE. In moving from $k+1$ to k variables, backward stepwise will pick one more variable to exclude.

      iii. FALSE. The variables selected by forward and backward stepwise are unrelated.

      iv. FALSE. Same reason as above.

      v. FALSE. It is possible to have no overlap between these models. Sometimes a group of variables will be very predictive of a response even when no single variable is.

2. *This problem is worth 15 points: three points for each correct sketch (partial credit of 1 point for a sketch that matches an incorrect explanation). Note that there are many acceptable answers and artistic quality of sketches should not impact grading.*

   In general, when we set $\lambda = \infty$, it means that the m^{th} derivative of $\hat{g}$ must be 0 everywhere. This also implied that all higher derivatives must be 0 as well. We are left with a polynomial of degree $m-1$.

   (a) $\hat{g} = 0$: a horizontal line at $y = 0$.

   (b) $\hat{g} = \bar{y}$: a horizontal line at the average value of y.

   (c) $\hat{g}$ is the best linear fit (equivalent to the least squares regression line).

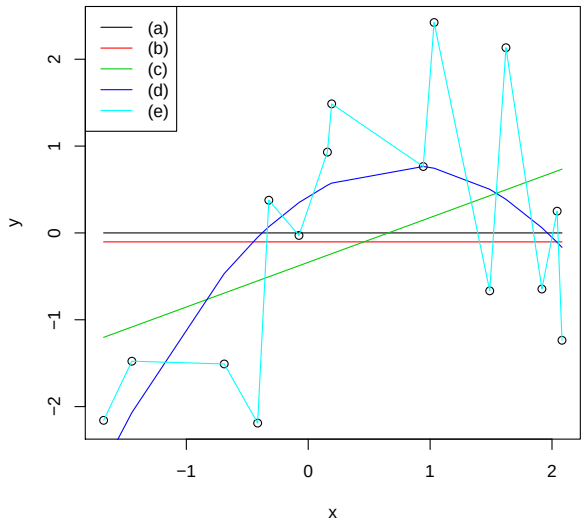
   (d) $\hat{g}$ is the best quadratic fit (equivalent to least squares with a quadratic term)

   (e) Here we are not putting any penalty on g, which means that $\hat{g}$ can be any function that perfectly fits the points in our sample (or as close as possible in the case of repeated x's).

   Here is one possible solution done in `R`:

```
set.seed(1337)
x = rnorm(15)
y = rnorm(15)+.2*x -.4*x^2 + sin(x)
plot(x,y)
lines(sort(x),predict(lm(y~0)), col = 1)
lines(sort(x),predict(lm(y~1))[order(x)], col = 2)
lines(sort(x),predict(lm(y~x))[order(x)], col = 3)
lines(sort(x),                          # this plot is inexact
      predict(lm(y~x+I(x^2)))[order(x)], # because we only calculated at
      col = 4)                          # x.  Should be smooth.
lines(sort(x),y[order(x)], col = 5)
legend("topleft",
       legend=paste("(",letters[1:5],")",sep=""),
   col = 1:5,
   lty = 1)
```

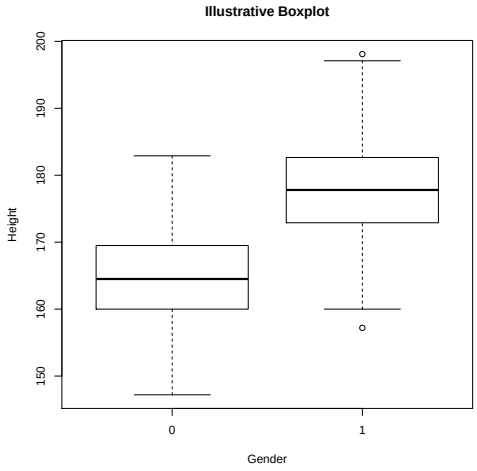
FIGURE 1. Sketches for Problem 2



3. *This problem is worth 15 points: two points each for parts a, b, d, and f. Three points for e. Four points for c.*

   (a) We can recover the encoding of gender in many ways. A simple one would be to look at boxplots of height by gender, knowing that men are taller on average than women. This reveals that a gender value of 1 corresponds to men.

FIGURE 2. Boxplot for 3a



   (b) Although all of the variables are in the same units, we think that they have different meanings. A change of 1cm in my wrist girth has drastic implications on my size in the way that a change of 1cm in my chest girth does not. If we did not scale the variables, the ones with larger variance would control almost all of the pls and pcr fits.

   (c) Here is the (trimmed) output from running summary():

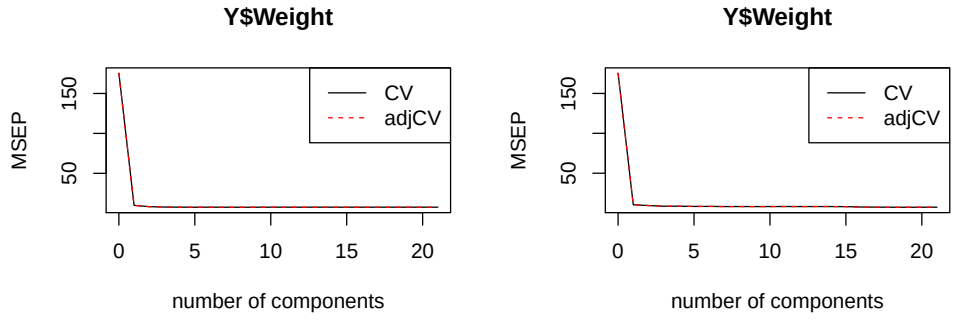
```
> summary(pls.fit)
```

```
...

TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X           64.45    74.37    79.74    81.94    83.52    85.99    88.60
Y$Weight    94.40    95.40    95.83    96.15    96.30    96.33    96.34
          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X           89.62    90.62     91.68     92.75     93.56     94.17     94.86
Y$Weight    96.35    96.35     96.35     96.35     96.35     96.35     96.35
          15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
X            95.94     96.65     97.58     98.19     98.77     99.51
Y$Weight     96.35     96.35     96.35     96.35     96.35     96.35
          21 comps
X           100.00
Y$Weight     96.35

> summary(pcr.fit)
...

TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X           64.47    75.73    80.44    84.43    86.82    88.64    90.21
Y$Weight    94.09    94.70    95.27    95.29    95.47    95.48    95.64
          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X           91.69    92.97     94.10     95.04     95.91     96.72     97.50
Y$Weight    95.64    95.66     95.66     95.68     95.71     95.72     95.79
          15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
X            98.10     98.54     98.96     99.34     99.60     99.84
Y$Weight     95.88     96.23     96.24     96.32     96.34     96.34
          21 comps
X           100.00
Y$Weight     96.35
```

As we can see, for a given number of components, pcr ends up with a better % variance explained of X and a worse % variance explained of Weight compared to pls. This is to be expected because pcr optimizes for % variance explained of X while pls optimizes for % variance explained of X AND Weight.

(d) To decide on a number of components to use in each model I examined the cross validated error:

FIGURE 3. CV Err for 3d



By examining that plot and the output of the summaries (which contains the CV error as well), I decided on using 8 components for pls and 19 components for pcr. These were the models with minimal CV error. Model selection can also be done with a 1-se rule or even potentially by sight as it seems that more than 2-3 components does not cause much improvement. Note that there are many correct answers on this problem depending on random seeds (including seeds used for the cross validation).

(e) Unfortunately, both of these methods still require all 21 measurements. That is because even though we only need 8 components for pls, the components themselves are constructed from the original 21 variables. Instead, I did a forward stepwise regression and selected the model that minimizes C_p. This resulted in selecting 13 components.

(f) My pls model had a test error around 8.65, pcr had 9.27, and forward stepwise 8.63. It seems like pcr is doing a little worse than the other two models.

Code:

```
load('body.RData')

# a
pdf("prob3a.pdf")
with(Y, boxplot(Height~Gender))
title(main = "Illustrative Boxplot", xlab = "Gender", ylab="Height")
dev.off()

# b
set.seed(1)
test = sample(1:nrow(X), 200)
train = (1:nrow(X))[-test]

require(pls)
pls.fit = plsr(Y$Weight ~ .,data = X, scale = TRUE, validation="CV", subset=train)
pcr.fit = pcr(Y$Weight ~ .,data = X, scale = TRUE, validation="CV", subset=train)

# c
summary(pls.fit)
summary(pcr.fit)

# d
pdf("prob3d.pdf", width = 8, height = 3)
par(mfcol=c(1,2))
validationplot(pls.fit, val.type="MSEP", legendpos="topright")
validationplot(pcr.fit, val.type="MSEP", legendpos="topright")
dev.off()

ncomp.pls = 8
ncomp.pcr = 19

# e
require(leaps)
reg.fit = regsubsets(Y$Weight~., data=X, method="forward", nvmax = 21, subset=train)
reg.sum = summary(reg.fit)
myid = which.min(reg.sum$cp)

predict.regsubsets = function(object, newdata, id, ...) {
```

```
    newdata = cbind(`(Intercept)` = 1, newdata)
    coefi = coef(object, id = id)
    #browser()
    do.call(cbind, newdata[, names(coefi)]) %*% coefi
}

# f
mean((Y$Weight[test] - predict(pls.fit,
                               newdata = X[test,],
                               id = ncomp.pls))^2)
mean((Y$Weight[test] - predict(pcr.fit,
                               newdata = X[test,],
                               id = ncomp.pcr))^2)
mean((Y$Weight[test] - predict(reg.fit,
                               newdata = X[test,],
                               id = myid))^2)
```

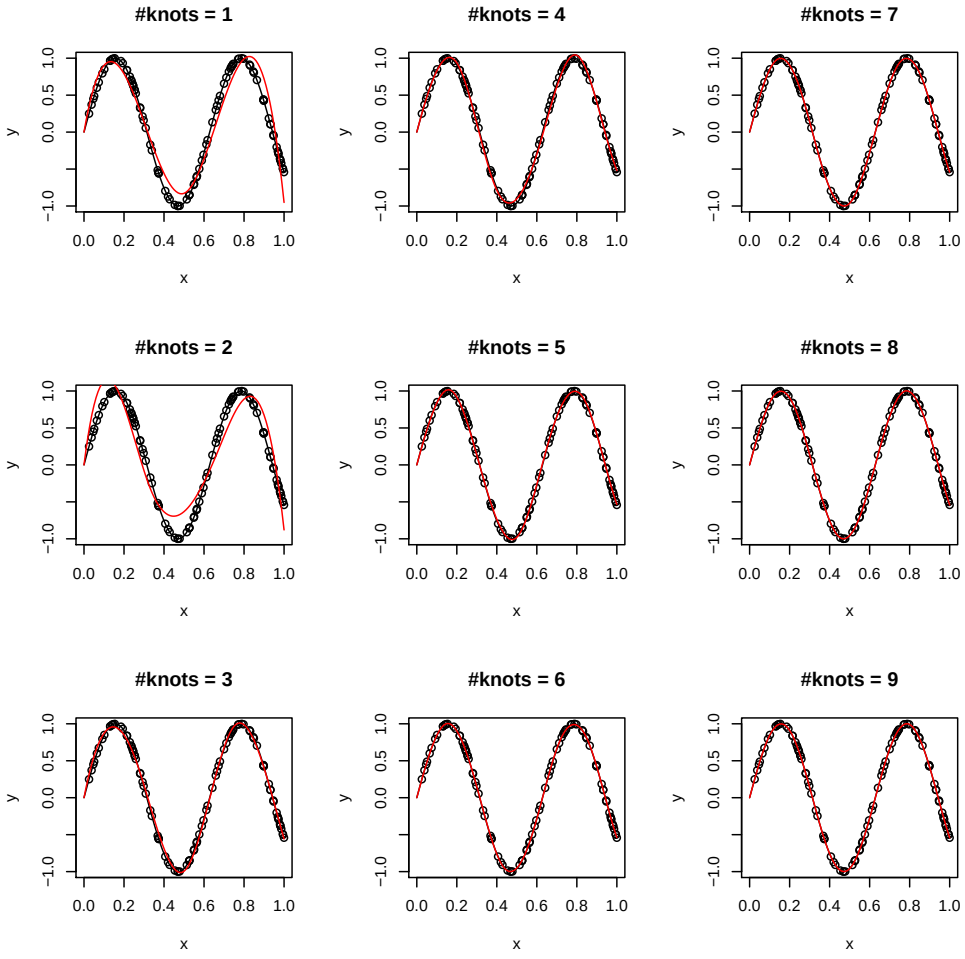
4. *This problem is worth 18 points: two points each for parts a, b, c, and g. Five points each for e and f. No points for d. For part f, there should not be a lot of redundant code in the solution. Ideally the student wrote a function that takes x, y, and k and does all the fitting/plotting. Take off up to 3 points for needlessly repeating code instead of putting it into a function to be reused*

   Code:

```
# a
h = function(x, z){ifelse(x > z, (x-z)^3, 0)}
#h = function(x, z){max((x-z)^3, 0)}

# b
hs = function(xs, z){sapply(xs, h, z=z)}
#hs = function(xs, z){ifelse(xs > z, (xs-z)^3, 0)}
#hs = function(xs, z){pmax((xs-z)^3, 0)}

# c
splinebasis = function(xs, zs){
cbind(xs, xs^2, xs^3, sapply(zs, hs, xs = xs))
}

# d
set.seed(1337)
x = runif(100)
y = sin(10*x)

# e/f
plot(x,y)
curve(sin(10*x))

pdf("prob4ef.pdf")
par(mfcol=c(3,3))
for(ii in 1:9){
k = ii
mymod = lm(y~splinebasis(x,zs=(1:k)/(k+1)) - 1)
curve(sin(10*x), main=paste("#knots =",ii), ylab = "y")
points(x,y)
```

Plots for parts e and f:

FIGURE 4. Fitted Spline Plots for 4e/f



As we can see, for all $k \geq 3$ the splines do a pretty good job fitting the curve. That said, as we let the number of knots go to infinity, we do NOT expect the fitted curve to continually become a better approximation to the true curve. Eventually we will be able to perfectly interpolate out training points, but we can't get a perfect approximation to the true curve unless the number of training points is increasing as well.