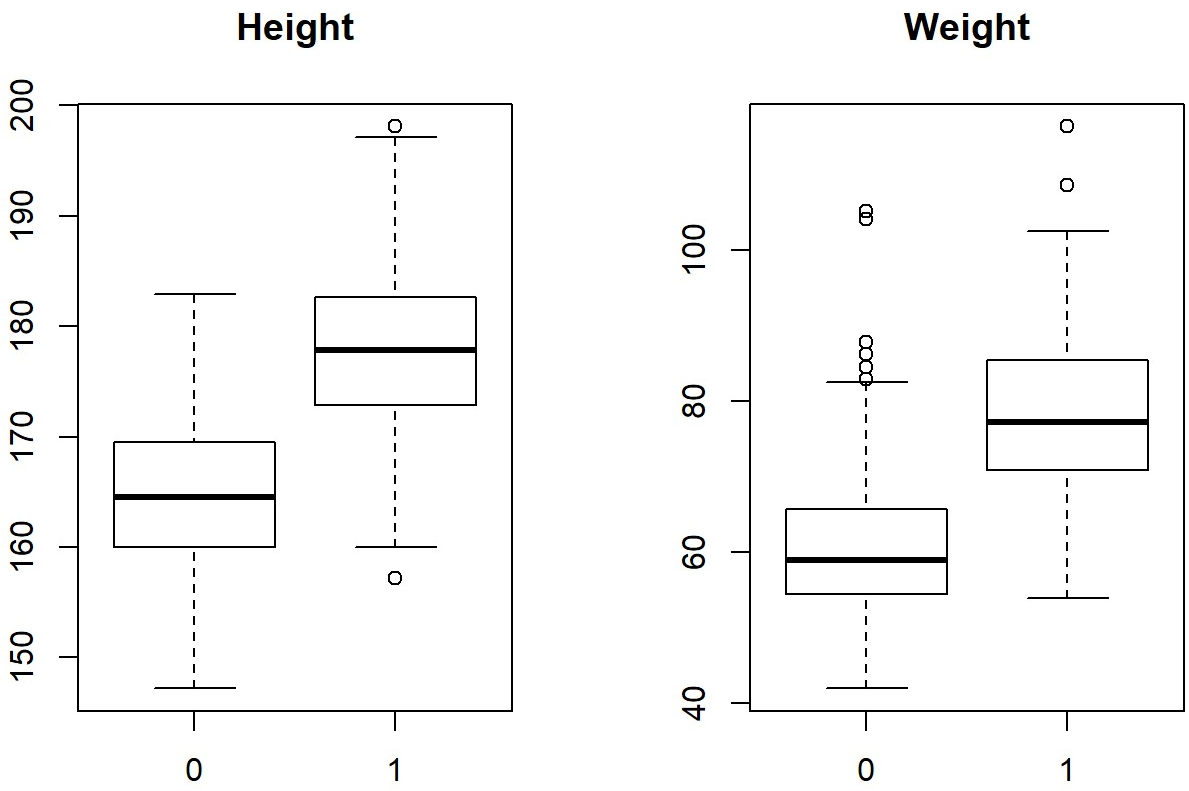
STATS 216: HW3

Xiangpeng Li

1.
    a. Best subset will have the smallest training RSS. Best subset will search all the possibilities combinations and find the best ones which have the lowest training RSS.
    b. More information needs to provide. We need to do cross validation or fit on the test data to get the test RSS.
    c.
        i. true. As we move forward, we will add a new predictor with smallest RSS to the current set.
        ii. true. As we move forward, we will remove a new predictor with smallest RSS from the current set.
        iii. false. They have different directions to choose predictors so they are not related.
        iv. false. The same reason as above.
        v. false. Subset selection picks are indenpendent at each step.
2.
    a. $\hat{g}$ will be a linear least square line across training data.
    b. $\hat{g}$ will be a piecewise cubic polynomial with knots and continuous first derivative.
    c. $\hat{g}$ will be a piecewise cubic polynomial with knots and continuous first and second derivatives.
    d. $\hat{g}$ will be a piecewise cubic polynomial with knots and continuous first, second and third derivatives.
    e. $\hat{g}$ will interpolate the training observation.
3.
    a.

```
setwd("D:/One Drive/OneDrive/Document/Study/Stanford/Introduction to Statistical Learning/hom
ework/hw3")
load("body.RData")
par(mfrow = c(1, 2))
attach(Y)
boxplot(Height ~ Gender, main = "Height")
boxplot(Weight ~ Gender, main = "Weight")
```



On average, `1` has higher height and heavier weight, so we can assume `1` denotes male and `0` denotes female.

b.

```
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
## set the seed to make your partition reproductible
set.seed(1)
traingIndex = sample(seq_len(nrow(X)), size = 307)

training = X[traingIndex, ]
trainRes = Y[traingIndex, ]

test = X[-traingIndex, ]
testRes = Y[-traingIndex, ]

pcr.fit = pcr(trainRes$Weight ~ ., data = training, scale = TRUE, validation = "CV")

plsr.fit = plsr(trainRes$Weight ~ ., data = training, scale = TRUE, validation = "CV")
```

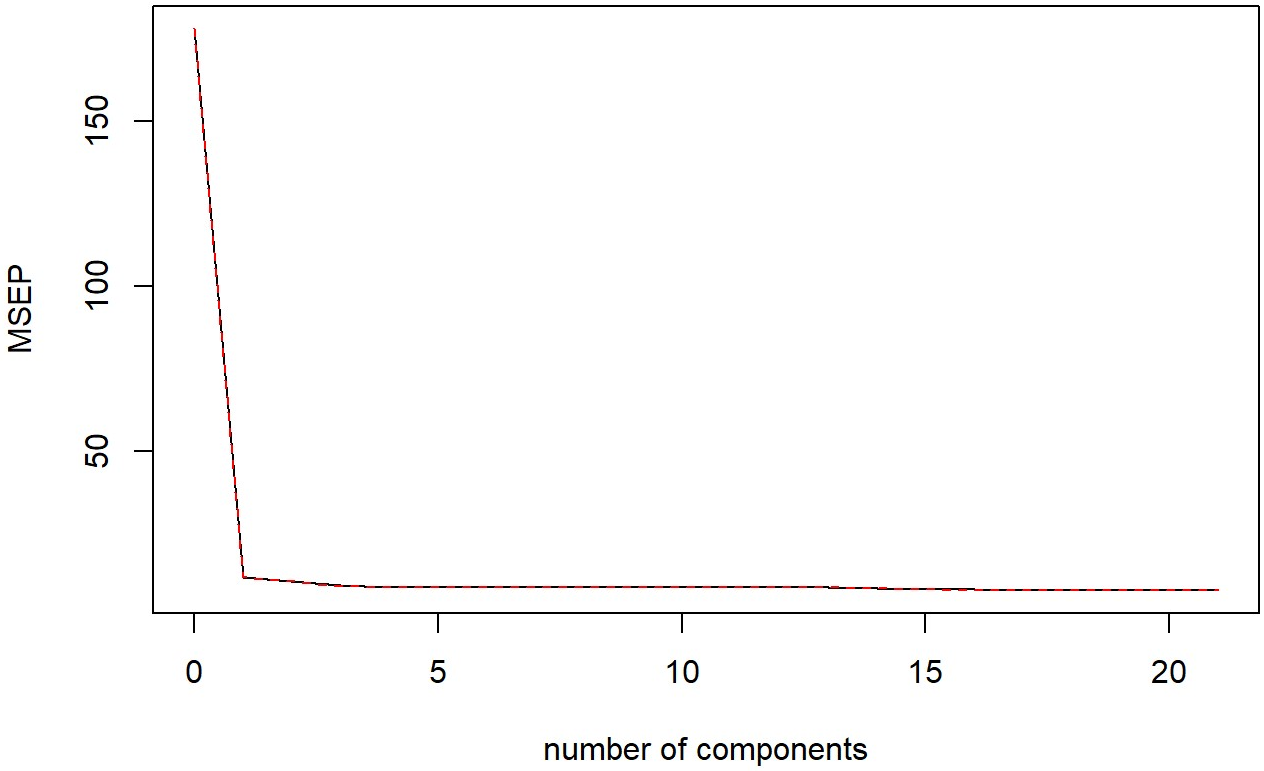
We choose to scale our variables because we can ensure all variables are on the same unit Otherwise the scale on which the variables are measured will have an effect on the final PCR model.

c.

```
summary(pcr.fit)
```

```
## Data:    X dimension: 307 21
##  Y dimension: 307 1
## Fit method: svdpc
## Number of components considered: 21
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           13.34    3.423    3.258    3.056    2.963    2.959    2.953
## adjCV        13.34    3.422    3.256    3.001    2.960    2.956    2.949
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       2.941    2.925    2.947     2.925     2.936     2.934     2.923
## adjCV    2.939    2.919    2.941     2.920     2.930     2.926     2.919
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        2.912     2.861     2.815     2.787     2.794     2.799
## adjCV     2.904     2.852     2.807     2.772     2.783     2.788
##        20 comps  21 comps
## CV        2.809     2.808
## adjCV     2.797     2.796
##
## TRAINING: % variance explained
##                   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X                   63.08    75.20    79.96    84.46    86.77    88.89
## trainRes$Weight     93.46    94.12    95.18    95.20    95.27    95.32
##                   7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X                   90.37    91.80    93.08     94.19     95.17     96.05
## trainRes$Weight     95.38    95.47    95.47     95.52     95.54     95.60
##                   13 comps  14 comps  15 comps  16 comps  17 comps
## X                    96.82     97.54     98.10     98.57     98.97
## trainRes$Weight      95.61     95.71     95.88     96.08     96.20
##                   18 comps  19 comps  20 comps  21 comps
## X                    99.34     99.60     99.82    100.00
## trainRes$Weight      96.20     96.21     96.21     96.23
```

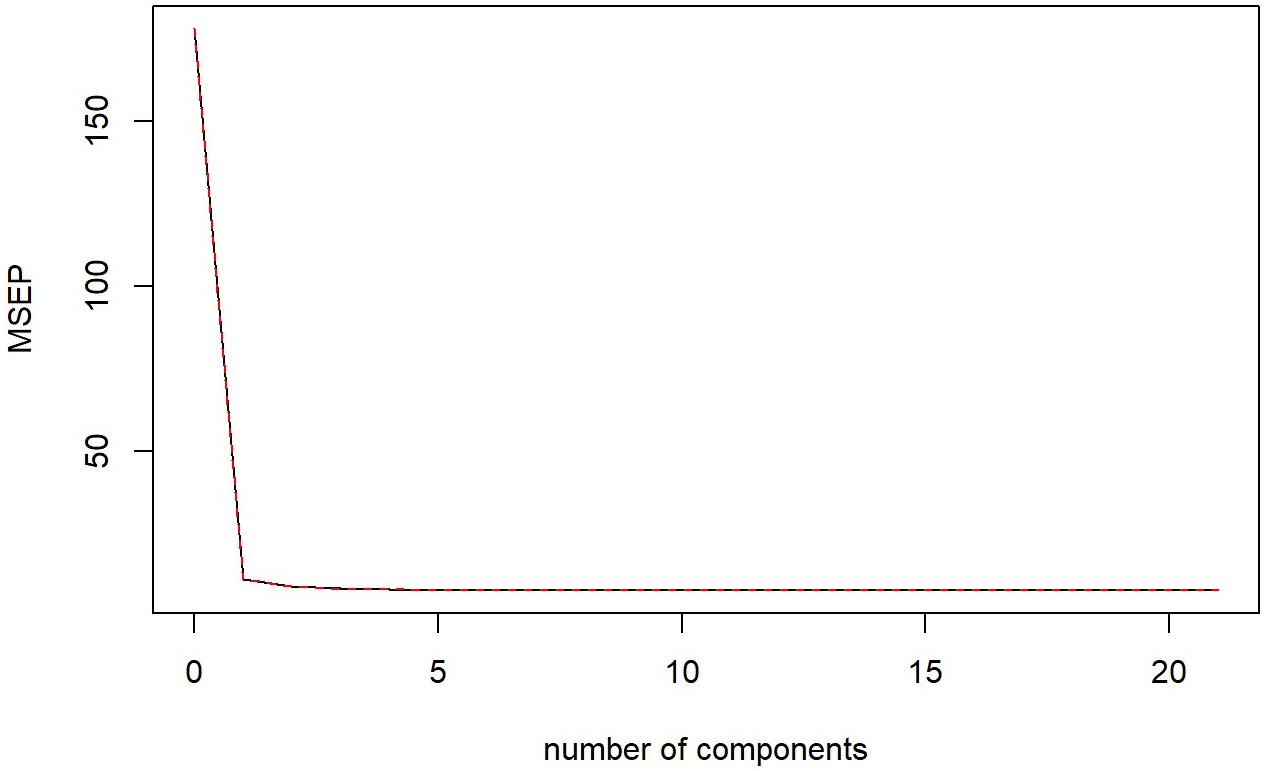
```
validationplot(pcr.fit, val.type = "MSEP", main = "PCR MSEP")
```




```
summary(plsr.fit)
```

```
## Data:    X dimension: 307 21
##  Y dimension: 307 1
## Fit method: kernelpls
## Number of components considered: 21
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           13.34    3.330    3.001    2.879    2.842    2.818    2.805
## adjCV        13.34    3.328    2.999    2.874    2.830    2.804    2.793
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       2.797    2.795    2.799     2.803     2.802     2.802     2.802
## adjCV    2.786    2.784    2.788     2.791     2.790     2.791     2.790
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        2.802     2.802     2.802     2.802     2.802     2.802
## adjCV     2.790     2.790     2.790     2.790     2.790     2.790
##        20 comps  21 comps
## CV        2.802     2.802
## adjCV     2.790     2.790
##
## TRAINING: % variance explained
##                   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X                   63.06    73.25    79.60    81.27    82.80    85.27
## trainRes$Weight     93.88    95.17    95.67    96.07    96.19    96.21
##                   7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X                   88.37    89.55    91.07     92.05     92.80     93.66
## trainRes$Weight     96.22    96.23    96.23     96.23     96.23     96.23
##                   13 comps  14 comps  15 comps  16 comps  17 comps
## X                    94.67     95.56     96.35     97.19     97.77
## trainRes$Weight      96.23     96.23     96.23     96.23     96.23
##                   18 comps  19 comps  20 comps  21 comps
## X                    98.57     99.01     99.66    100.00
## trainRes$Weight      96.23     96.23     96.23     96.23
```

```
validationplot(plsr.fit, val.type = "MSEP", main = "PLSR MSEP")
```



The lowest cross-validation occurs when $M = 17$ for PCR model and $M = 8$ for PLS model. Also we can see for the same number of components these two model are using, PLS tends to have slightly smaller cross-validation error and higher percentage of variance explained. That's because PLS searches for directions that explain variance in both predictors and response whereas PCR only considers predictors. However, it is just slight different in the two model (95.2% for PCR and 96.07% for PLS using 4 components), PLS made an small improvement on PCR predictions.

d.

As we can see from above figures descriping MESP for two models, MSEPs drop significantly around 2-4 components, and don't have a noticable changes as we inscrease the number of components, we can see the same pattern for cross-validation error and % variance explained. So we choose $M = 4$ to make predictions.

```
pcr.pred = predict(pcr.fit, newdata = test, ncomp = 4);

mean((pcr.pred - testRes)^2)
```

```
## [1] 4281.326
```

```
plsr.pred = predict(plsr.fit, newdata = test, ncomp = 4);

mean((plsr.pred - testRes)^2)
```

```
## [1] 4279.822
```

The test MSE for PLS model is bit smaller than PCR test MSE which provide another evidence that PLS is slightly better than PCR in this scenario.

e.

PCR and PLS are not feature selection models, it's because each of the components will be using a linear combination of all features. A typical feature selection model is LASSO, so we can use LASSO to fit this data to improve interpretion.

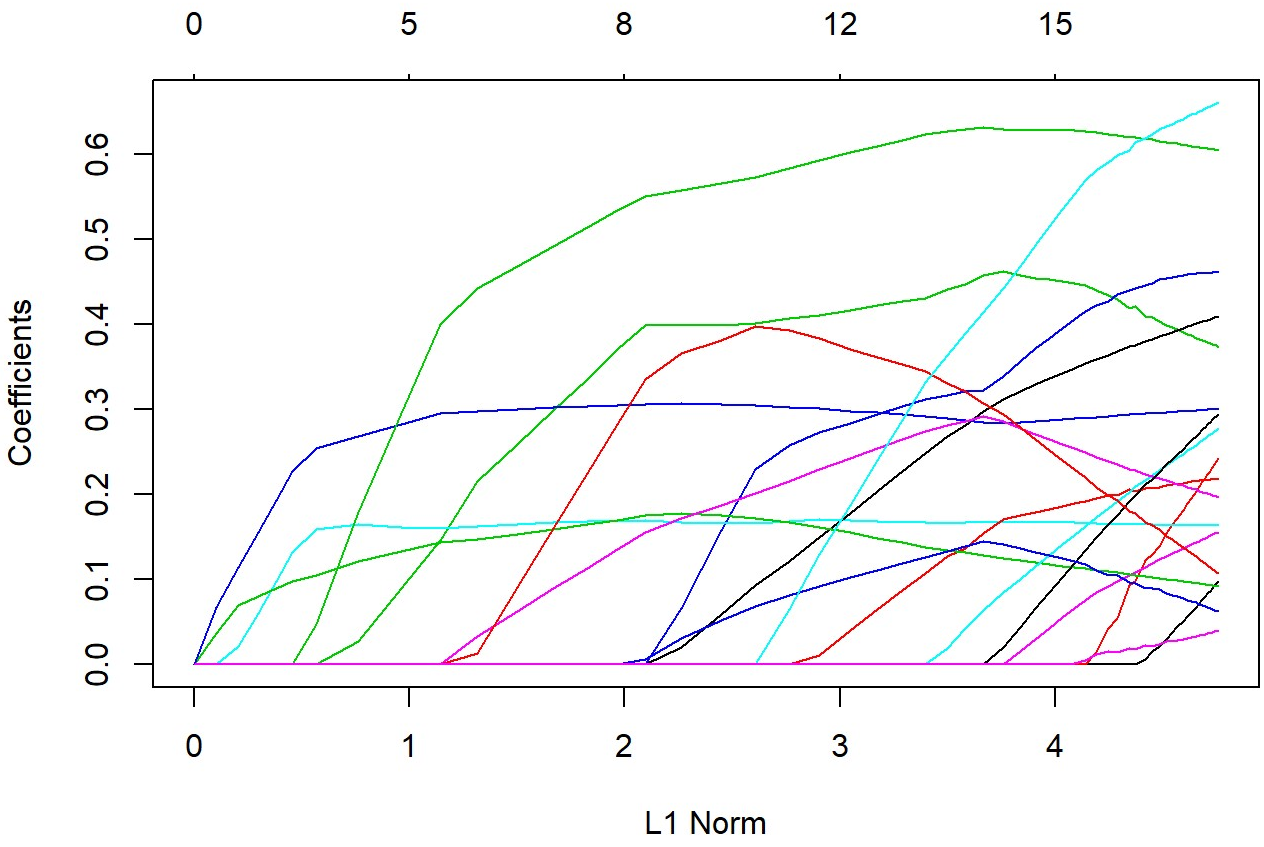
```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-13
```

```
dataFrame = data.frame("Weight" = Y$Weight, X)
x = model.matrix(Weight ~ ., dataFrame[traingIndex,])[, -1]
xTest = model.matrix(Weight ~ ., dataFrame[-traingIndex,])[, -1]

# fit the lasso model
lasso.fit = glmnet(x, trainRes$Weight, alpha = 1)
plot(lasso.fit)
```




```
# use cross-validation to get the tuning parameter
cv.out = cv.glmnet(x, trainRes$Weight, alpha = 1)
bestlam = cv.out$lambda.min

lasso.coef = predict(lasso.fit, s = bestlam, type = "coefficients")
# coefficient estimates using cross-validation
lasso.coef
```

```
## 22 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)         -100.45764492
## Wrist.Diam             0.09285864
## Wrist.Girth            0.23600867
## Forearm.Girth          0.37524915
## Elbow.Diam             0.46117144
## Bicep.Girth            .
## Shoulder.Girth         0.16318198
## Biacromial.Diam        0.15389404
## Chest.Depth            0.40779724
## Chest.Diam             0.21823430
## Chest.Girth            0.09231997
## Navel.Girth            .
## Waist.Girth            0.30016716
## Pelvic.Breadth         0.27411671
## Bitrochanteric.Diam    .
## Hip.Girth              0.19807114
## Thigh.Girth            0.29043305
## Knee.Diam              0.11038084
## Knee.Girth             0.60539062
## Calf.Girth             0.06393019
## Ankle.Diam             0.65893472
## Ankle.Girth            0.03846260
```

f.

```
# predictions using PCR
pcr.pred = predict(pcr.fit, newdata = test, ncomp = 4);
pcr.mse = mean((pcr.pred - testRes)^2)


# predictions using PLS
plsr.pred = predict(plsr.fit, newdata = test, ncomp = 4);
plsr.mse = mean((plsr.pred - testRes)^2)

# precitions using LASSO
lasso.pred = predict(lasso.fit, s = bestlam, newx = xTest)
lasso.mse = mean((lasso.pred - testRes)^2)

dataFrame = data.frame(matrix(ncol = 3, nrow = 0))
colnames(dataFrame) = c("PCR", "PLS", "LASSO")

dataFrame[1, ] = c(pcr.mse, plsr.mse, lasso.mse)
print(dataFrame)
```

```
##        PCR      PLS    LASSO
## 1 4281.326 4279.822 4280.238
```

Although LASSO only using 18 out of 21 features, it doesn't have a better predictions. Results show these three model have the similar performance. PLS has the lowest test MSE followed by LASSO, PCR has the highest test MSE.

4.

a.

```
h = function(x, z) {
  if (x > z) (x - z)^3
  else 0
}
```

b.

```
hs = function(xs, z) {
  sapply(xs, function (x) h(x, z))
}
```

c.

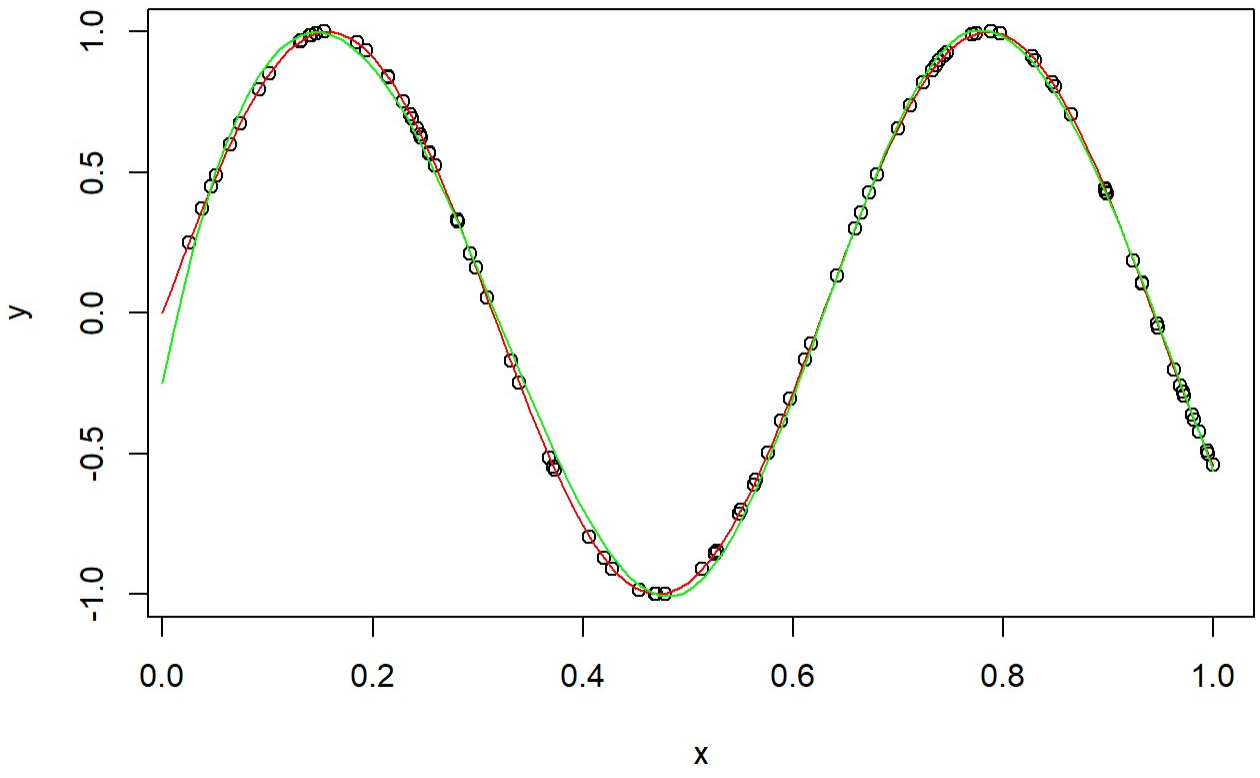
```
splinebasis = function(xs, zs) {
  row = length(xs);
  col = length(zs) + 3;
  res = matrix(nrow = row, ncol = col);
  res[, 1] = xs;
  res[, 2] = xs^2;
  res[, 3] = xs^3;
  for (i in 1:length(zs)) {
    res[, i+3] = hs(xs, zs[i]);
  }
  res;
}
```

d.

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

e.

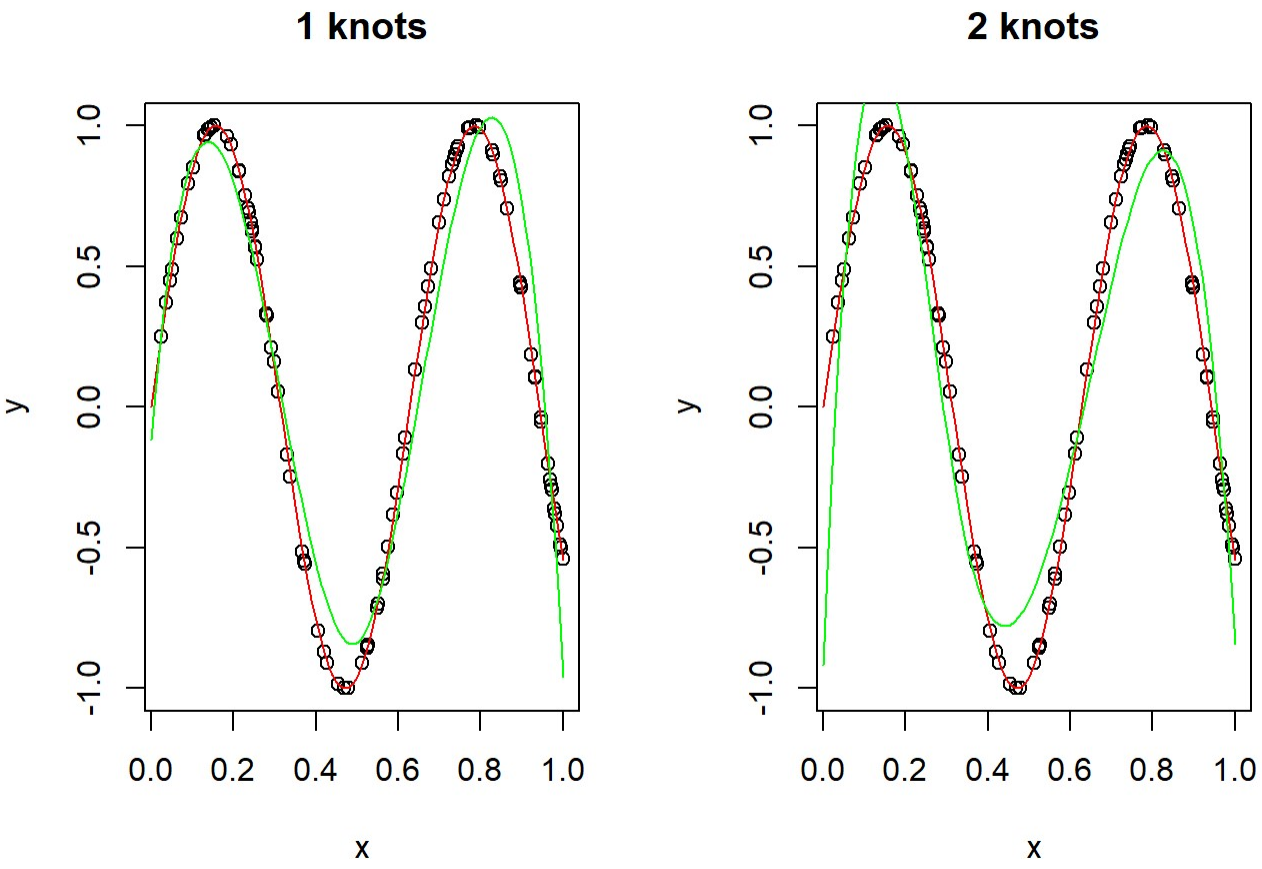
```
knots = c(1/4, 2/4, 3/4)
dataFrame = data.frame(y, splinebasis(x, knots))

# fit
lm.fit = lm(y ~ ., data = dataFrame)


intvl = seq(0, 1, len = 100)
nx = splinebasis(intvl, knots)
test = data.frame(nx)

lm.pred = predict(lm.fit, newdata = test)


plot(x, y)
curve(sin(10*x), 0, 1, add = TRUE, col = "red")
lines(intvl, lm.pred, col = "green")
```

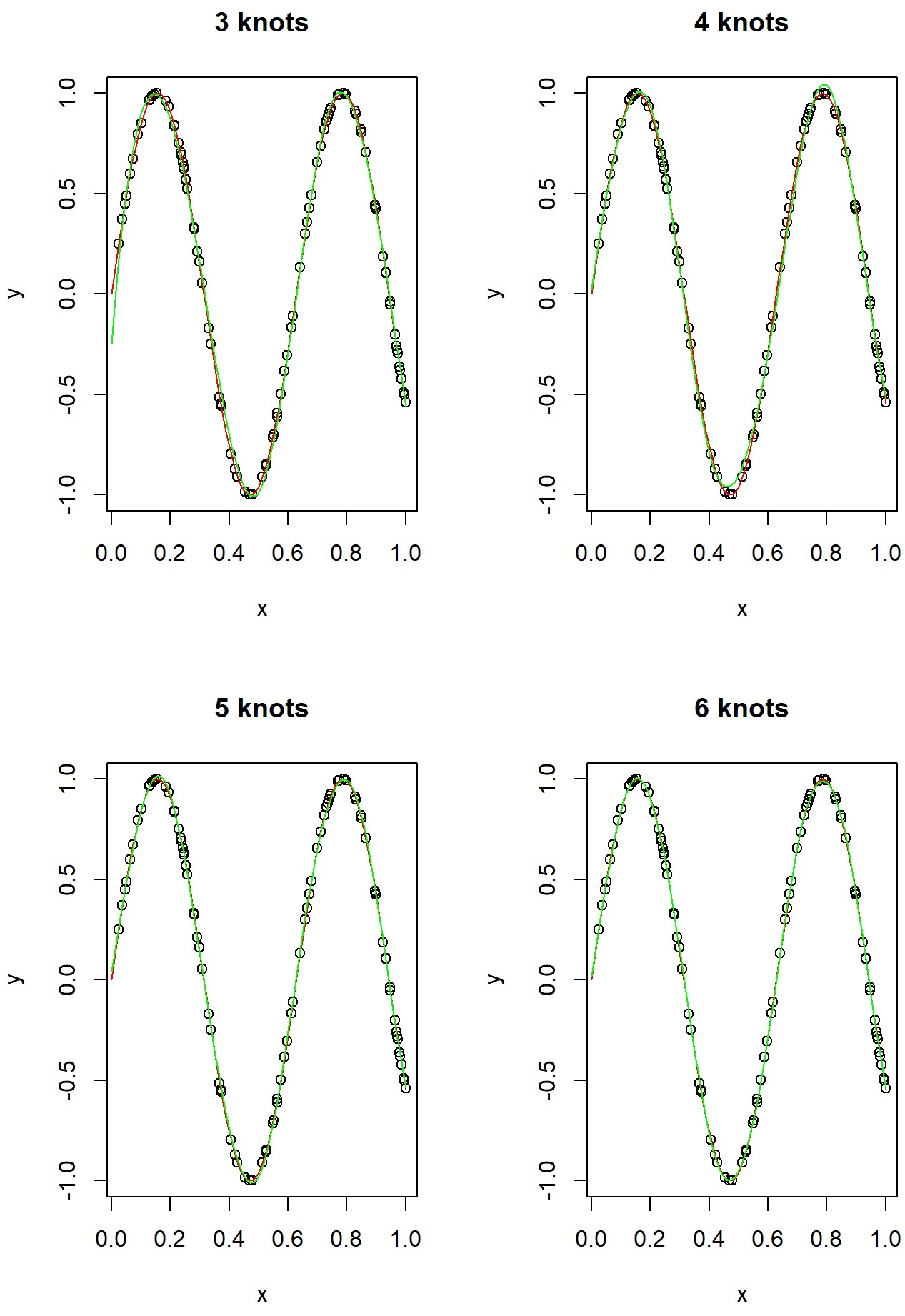


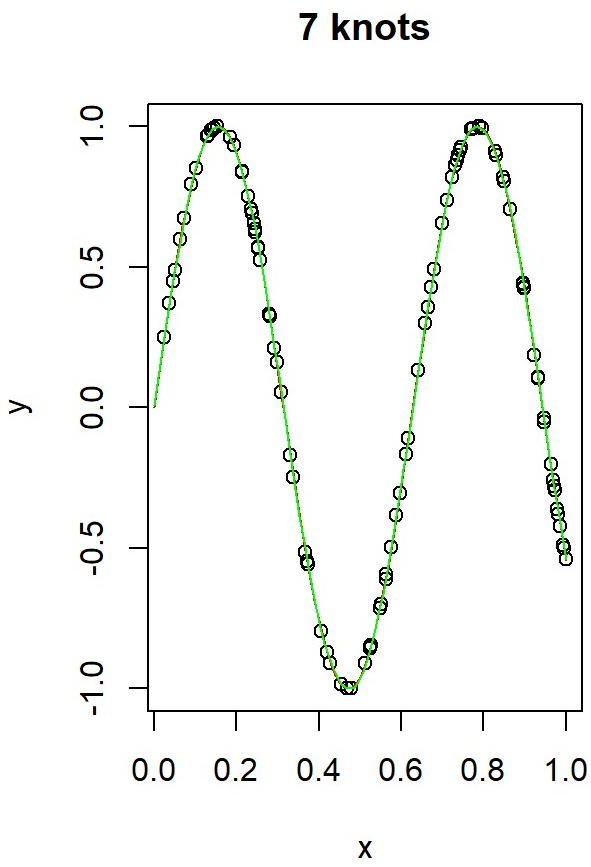
green line represents the fitted spline, each black circle represents the original points and red line represents the curve those points were generated from

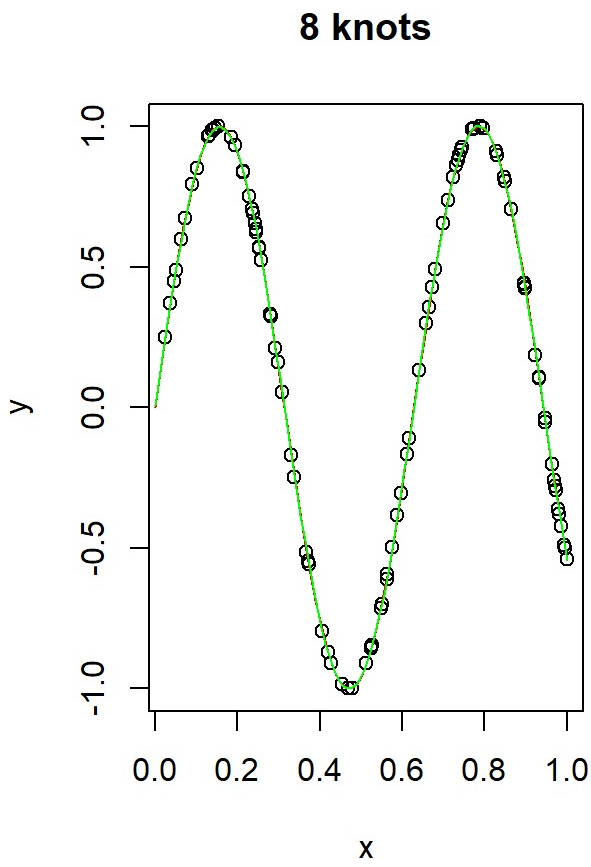
f.

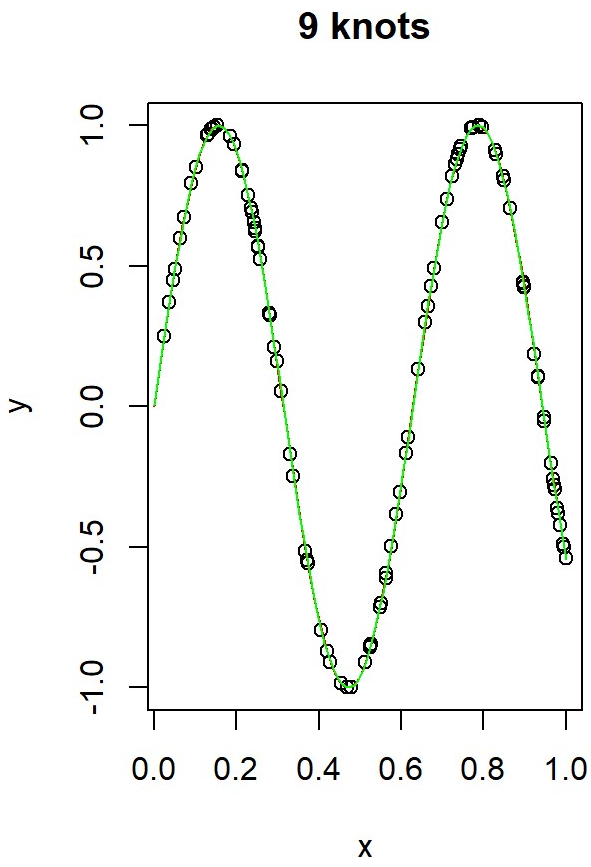
```
par(mfrow = c(1, 2))
for (k in 1:9) {
  knots = 1:k
  knots = knots / (k + 1)
  dataFrame = data.frame(y, splinebasis(x, knots))
  lm.fit = lm(y ~ ., data = dataFrame)
  nx = splinebasis(intvl, knots)
  test = data.frame(nx)
  lm.pred = predict(lm.fit, newdata = test)
  plot(x, y, main = paste(k, "knots"))
  curve(sin(10*x), 0, 1, add = TRUE, col = "red")
  lines(intvl, lm.pred, col = "green")
}
```




3 knots
4 knots
5 knots
6 knots
y
x
1.0
0.5
0.0
-0.5
-1.0
0.0
0.2
0.4
0.6
0.8
1.0


7 knots
y
x
1.0
0.5
0.0
-0.5
-1.0
0.0
0.2
0.4
0.6
0.8
1.0


8 knots
y
x
1.0
0.5
0.0
-0.5
-1.0
0.0
0.2
0.4
0.6
0.8
1.0


9 knots
y
x
1.0
0.5
0.0
-0.5
-1.0
0.0
0.2
0.4
0.6
0.8
1.0

g. Theoretically as we increase the number of knots, the fitted curve will continually approximate to the true curve and eventually interpolate the true curve. But in this case when knots are greater than 4, we can barely see the difference between these two curves.