stat602_hw2

4.3

First we read in the data set and remove the irrelavant columns.

```
house <- read_excel("~/work/homework/AmesHousingData.xlsx")
house <- data.frame(Price = house$Price, Size = house$Size,
                    Fireplace = house$Fireplace,
                    Basementbath = house$`Bsmt Bath`, Land = house$Land, intersect = 1)
head(house)
```

```
##    Price Size Fireplace Basementbath  Land intersect
## 1 141900 1762         0            1  7200         1
## 2 124000 2104         0            0  6300         1
## 3 109900 1040         0            0  6240         1
## 4 105000 1220         0            0  6120         1
## 5  84900 1177         0            0  7218         1
## 6 200000 2234         2            1 12975         1
```

Now we need a matrix of all the possible regressor combinations.

```
regMat <- expand.grid(c(TRUE,FALSE), c(TRUE,FALSE),
                      c(TRUE,FALSE), c(TRUE,FALSE))
regMat <- cbind(TRUE,regMat)

head(regMat)
```

```
##   TRUE  Var1  Var2  Var3 Var4
## 1 TRUE  TRUE  TRUE  TRUE TRUE
## 2 TRUE FALSE  TRUE  TRUE TRUE
## 3 TRUE  TRUE FALSE  TRUE TRUE
## 4 TRUE FALSE FALSE  TRUE TRUE
## 5 TRUE  TRUE  TRUE FALSE TRUE
## 6 TRUE FALSE  TRUE FALSE TRUE
```

Then we construct a formula so that from each row of regMat we can determine the the regressor combinations.

```
formu <- function(vec)
{
  vec <- as.matrix(vec)
  regressors <- c("intersect","Size", "Fireplace", "Basementbath", "Land")
  out <- as.formula(paste(c("Price ~ 0", regressors[vec]), collapse=" + "))
```

```
  return(out)
}

formu(regMat[1,])
```

```
## Price ~ 0 + intersect + Size + Fireplace + Basementbath + Land
## <environment: 0xcb794c8>
```

Cross-validation using caret package

We will repeat the 8-fold crossval times.

```
ctrl <- trainControl(method = "repeatedcv", repeats = 5, number = 8)

RMSE <- numeric(16)
for(i in 1:length(RMSE)){
  model_caret <- train(formu(regMat[i,]), data = house, trControl = ctrl, method = "lm")
  RMSE[i] = model_caret$results[,2]
}
RMSE
```

```
##  [1] 21709.31 26670.86 24488.29 32482.33 22305.24 27720.10 25184.11
##  [8] 34037.32 22336.93 27915.77 26790.50 36653.70 23441.72 28786.40
## [15] 27497.23 36665.69
```

```
order(RMSE)
```

```
##  [1]  1  5  9 13  3  7  2 11 15  6 10 14  4  8 12 16
```

```
regMat
```

```
##    TRUE  Var1  Var2  Var3  Var4
## 1  TRUE  TRUE  TRUE  TRUE  TRUE
## 2  TRUE FALSE  TRUE  TRUE  TRUE
## 3  TRUE  TRUE FALSE  TRUE  TRUE
## 4  TRUE FALSE FALSE  TRUE  TRUE
## 5  TRUE  TRUE  TRUE FALSE  TRUE
## 6  TRUE FALSE  TRUE FALSE  TRUE
## 7  TRUE  TRUE FALSE FALSE  TRUE
## 8  TRUE FALSE FALSE FALSE  TRUE
## 9  TRUE  TRUE  TRUE  TRUE FALSE
## 10 TRUE FALSE  TRUE  TRUE FALSE
## 11 TRUE  TRUE FALSE  TRUE FALSE
## 12 TRUE FALSE FALSE  TRUE FALSE
## 13 TRUE  TRUE  TRUE FALSE FALSE
## 14 TRUE FALSE  TRUE FALSE FALSE
## 15 TRUE  TRUE FALSE FALSE FALSE
## 16 TRUE FALSE FALSE FALSE FALSE
```

We can see that the full model has the smallest mean squared prediction error.

4.4

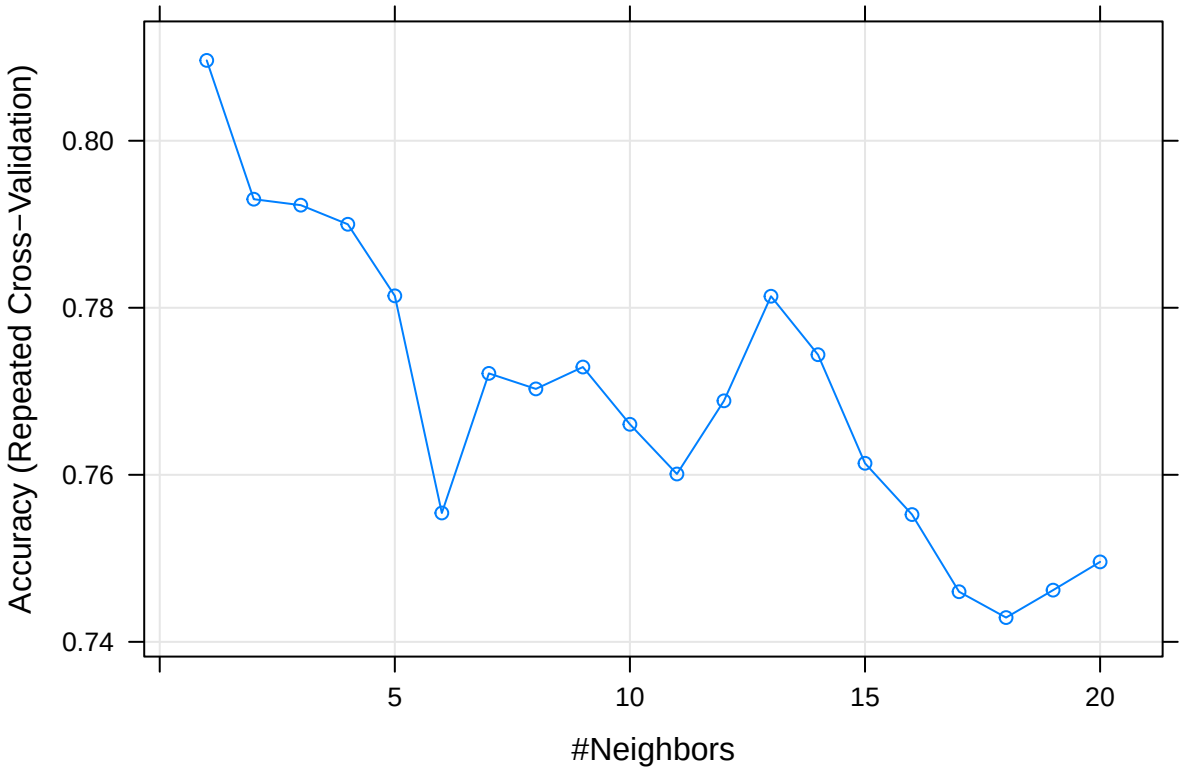
First read in the dataset:

```
glass <- read.csv("~/work/homework/glass.data", header=FALSE)
names(glass) <- c("Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type")
glass_sub <- subset(glass, Type %in% c(1,2))
glass_sub$Type <- as.factor(glass_sub$Type)
```

(a)

The best number of neighbors for this prediction task is 1.

```
ctrl <- trainControl(method="repeatedcv",repeats = 10, number = 10)
knnFit <- train(Type ~ . - Id, data = glass_sub, method = "knn", trControl = ctrl, preProcess = c("cent
plot(knnFit)
```



(b)

See hand-written solution.

5.1

(a)

```
X <- c(2, 4, 3, 5, 1, 4, 3, 4, 2, 3, 7, 5, 6, 4, 4, 2, 5, 1, 2, 4)
X <- matrix(X, nrow = 5)

qrObj <- qr(X)
svdObj <- svd(X)
```

The Q matrix and R matrix are:

```
qr.Q(qrObj)
```

```
##            [,1]        [,2]       [,3]        [,4]
## [1,] -0.2696799  0.57022480  0.7723433  0.04477265
## [2,] -0.5393599 -0.06579517 -0.1252449  0.56476381
## [3,] -0.4045199  0.37283929 -0.3548604 -0.70772210
## [4,] -0.6741999 -0.50442963  0.1043707 -0.12567762
## [5,] -0.1348400  0.52636136 -0.5009794  0.40295387
```

```
qr.R(qrObj)
```

```
##           [,1]      [,2]       [,3]      [,4]
## [1,] -7.416198 -6.067799 -10.247838 -5.528439
## [2,]  0.000000  4.145096   5.987360  2.280899
## [3,]  0.000000  0.000000   1.064581 -1.231574
## [4,]  0.000000  0.000000   0.000000  3.566102
```

The colums of Q matrix give the bases for $C(\mathbf{X})$.

The SVD results are:

```
svdObj
```

```
## $d
## [1] 16.5812586  3.7842977  3.3820909  0.5499212
##
## $u
##            [,1]       [,2]       [,3]        [,4]
## [1,] -0.4993567  0.5283913  0.1647824 -0.66384835
## [2,] -0.5040186 -0.5891454  0.1259937 -0.11349453
## [3,] -0.4582533  0.4878118 -0.2526166  0.64627429
## [4,] -0.3913976 -0.3254921 -0.6846703 -0.08815998
## [5,] -0.3652670 -0.1726408  0.6514474  0.34782427
##
## $v
##            [,1]       [,2]        [,3]        [,4]
## [1,] -0.4047825 -0.4324356 -0.79720334  0.11669381
## [2,] -0.4354972  0.2981978  0.18084854  0.82988796
## [3,] -0.7111504  0.4458822  0.03987313 -0.54209258
## [4,] -0.3751780 -0.7247528  0.57460488 -0.06167786
```

The U has columns spanning $C(\mathbf{X})$.

(b)

The eigen decomposition of $X^{'}X$ is:

```
(V <- svdObj$v)
```

```
##            [,1]       [,2]        [,3]        [,4]
## [1,] -0.4047825 -0.4324356 -0.79720334  0.11669381
## [2,] -0.4354972  0.2981978  0.18084854  0.82988796
## [3,] -0.7111504  0.4458822  0.03987313 -0.54209258
## [4,] -0.3751780 -0.7247528  0.57460488 -0.06167786
```

```
(D_sq <- diag(svdObj$d) %*% diag(svdObj$d))
```

```
##          [,1]     [,2]     [,3]      [,4]
## [1,] 274.9381  0.00000  0.00000 0.0000000
## [2,]   0.0000 14.32091  0.00000 0.0000000
## [3,]   0.0000  0.00000 11.43854 0.0000000
## [4,]   0.0000  0.00000  0.00000 0.3024134
```

The eigen decomposition of $XX^{'}$ is:

```
(U <- svdObj$u)
```

```
##            [,1]       [,2]       [,3]        [,4]
## [1,] -0.4993567  0.5283913  0.1647824 -0.66384835
## [2,] -0.5040186 -0.5891454  0.1259937 -0.11349453
## [3,] -0.4582533  0.4878118 -0.2526166  0.64627429
## [4,] -0.3913976 -0.3254921 -0.6846703 -0.08815998
## [5,] -0.3652670 -0.1726408  0.6514474  0.34782427
```

```
(D_sq <- diag(svdObj$d) %*% diag(svdObj$d))
```

```
##          [,1]     [,2]     [,3]      [,4]
## [1,] 274.9381  0.00000  0.00000 0.0000000
## [2,]   0.0000 14.32091  0.00000 0.0000000
## [3,]   0.0000  0.00000 11.43854 0.0000000
## [4,]   0.0000  0.00000  0.00000 0.3024134
```

(c)

rank = 1

```
( rank1approx <- best_approx(X, 1) )
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 3.351583 3.605900 5.888298 3.106459
## [2,] 3.382873 3.639565 5.943271 3.135461
## [3,] 3.075706 3.309089 5.403616 2.850758
## [4,] 2.626983 2.826318 4.615269 2.434854
## [5,] 2.451600 2.637627 4.307144 2.272298
```

rank = 2

```
( rank2approx <- best_approx(X, 2) )
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 2.486889 4.202174 6.779880 1.657251
## [2,] 4.346989 2.974732 4.949176 4.751299
## [3,] 2.277419 3.859570 6.226726 1.512847
## [4,] 3.159640 2.459010 4.066050 3.327575
## [5,] 2.734121 2.442807 4.015839 2.745797
```

(d)

First center the matrix:

```
X_center <- scale(X, center = TRUE, scale = FALSE)

X_center_scale <- scale(X, center = T, scale = T)
```

```
( svdObj2 <- svd(X_center) )
```

```
## $d
## [1] 3.8322510 3.3823499 2.0106498 0.4804676
##
## $u
##            [,1]       [,2]        [,3]        [,4]
## [1,] -0.5712672  0.1656304  0.28534764 -0.60398428
## [2,]  0.5089398  0.1143288  0.69563542  0.20976309
## [3,] -0.4996900 -0.2490239 -0.08995577  0.69296822
## [4,]  0.3402207 -0.6851706 -0.32383532 -0.33154457
## [5,]  0.2217967  0.6542353 -0.56719197  0.03279753
##
## $v
##            [,1]        [,2]      [,3]         [,4]
## [1,]  0.3436766 -0.80716469 0.4461250  0.177042475
## [2,] -0.3682374  0.17791687 0.2582385  0.875248373
## [3,] -0.5751820  0.03345965 0.6822505 -0.450089261
## [4,]  0.6445566  0.56188184 0.5184782  0.003988055
```

```
( svdObj3 <- svd(X_center_scale) )
```

```
## $d
## [1] 3.0431883 2.1387235 1.3876419 0.4892004
##
## $u
##            [,1]         [,2]       [,3]        [,4]
## [1,] -0.5985980 -0.003516761  0.1400083 -0.64966586
## [2,]  0.3012174  0.224786627  0.7869330  0.19868439
## [3,] -0.4397787 -0.343527972 -0.1595103  0.68054368
## [4,]  0.5757526 -0.580732932 -0.2401616 -0.27125742
## [5,]  0.1614067  0.702991038 -0.5272694  0.04169521
```

```
##
## $v
##            [,1]       [,2]      [,3]         [,4]
## [1,]  0.3592298 -0.6917233 0.5565679  0.287584734
## [2,] -0.6339580  0.1305970 0.1900625  0.738185562
## [3,] -0.5983454 -0.1695573 0.4907728 -0.610225552
## [4,]  0.3332176  0.6897200 0.6428456 -0.001368531
```

The principal component directions:

```
svdObj2$v
```

```
##            [,1]        [,2]      [,3]         [,4]
## [1,]  0.3436766 -0.80716469 0.4461250  0.177042475
## [2,] -0.3682374  0.17791687 0.2582385  0.875248373
## [3,] -0.5751820  0.03345965 0.6822505 -0.450089261
## [4,]  0.6445566  0.56188184 0.5184782  0.003988055
```

```
svdObj3$v
```

```
##            [,1]       [,2]      [,3]         [,4]
## [1,]  0.3592298 -0.6917233 0.5565679  0.287584734
## [2,] -0.6339580  0.1305970 0.1900625  0.738185562
## [3,] -0.5983454 -0.1695573 0.4907728 -0.610225552
## [4,]  0.3332176  0.6897200 0.6428456 -0.001368531
```

The "loadings" of the first principal component:

```
svdObj2$v[,1]
```

```
## [1]  0.3436766 -0.3682374 -0.5751820  0.6445566
```

```
svdObj3$v[,1]
```

```
## [1]  0.3592298 -0.6339580 -0.5983454  0.3332176
```

The principal components:

```
X_center %*% svdObj2$v
```

```
##            [,1]       [,2]       [,3]        [,4]
## [1,] -2.1892395  0.5602201  0.5737342 -0.29019489
## [2,]  1.9503851  0.3867000  1.3986792  0.10078437
## [3,] -1.9149375 -0.8422861 -0.1808695  0.33294879
## [4,]  1.3038112 -2.3174867 -0.6511194 -0.15929643
## [5,]  0.8499806  2.2128526 -1.1404244  0.01575815
```

```
X_center_scale %*% svdObj3$v
```

```
##            [,1]         [,2]       [,3]        [,4]
## [1,] -1.8216464 -0.007521379  0.1942814 -0.31781681
## [2,]  0.9166614  0.480756438  1.0919812  0.09719649
## [3,] -1.3383294 -0.734711340 -0.2213431  0.33292225
## [4,]  1.7521236 -1.242027160 -0.3332583 -0.13269924
## [5,]  0.4911909  1.503503442 -0.7316611  0.02039731
```

(e)

The rank = 1 approximation:

```
( rank1approx_center <- best_approx(X_center, 1) )
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.7523905  0.8061597  1.2592112 -1.4110888
## [2,]  0.6703018 -0.7182046 -1.1218265  1.2571337
## [3,] -0.6581193  0.7051515  1.1014376 -1.2342856
## [4,]  0.4480894 -0.4801120 -0.7499288  0.8403802
## [5,]  0.2921185 -0.3129946 -0.4888936  0.5478606
```

```
( rank1approx_center_scale <- best_approx(X_center_scale, 1) )
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.6543898  1.1548473  1.0899737 -0.6070047
## [2,]  0.3292921 -0.5811248 -0.5484801  0.3054477
## [3,] -0.4807678  0.8484446  0.8007832 -0.4459550
## [4,]  0.6294151 -1.1107727 -1.0483750  0.5838385
## [5,]  0.1764504 -0.3113944 -0.2939018  0.1636735
```

The rank = 2 approximation:

```
( rank1approx_center <- best_approx(X_center, 2) )
```

```
##             [,1]       [,2]       [,3]       [,4]
## [1,] -1.20458032  0.9058323  1.2779560 -1.0963113
## [2,]  0.35817116 -0.6494042 -1.1088876  1.4744134
## [3,]  0.02174434  0.5552946  1.0732550 -1.7075509
## [4,]  2.31868287 -0.8924320 -0.8274711 -0.4617735
## [5,] -1.49401805  0.0807092 -0.4148523  1.7912223
```

```
( rank1approx_center_scale <- best_approx(X_center_scale, 2) )
```

```
##              [,1]       [,2]       [,3]       [,4]
## [1,] -0.649187041  1.1538651  1.0912490 -0.6121924
## [2,] -0.003258318 -0.5183395 -0.6299958  0.6370351
## [3,]  0.027449121  0.7524935  0.9253588 -0.9527000
## [4,]  1.488554221 -1.2729777 -0.8377803 -0.2728124
## [5,] -0.863557983 -0.1150414 -0.5488317  1.2006698
```

(f)

```
cov_matrix_center <- 1/5 * t(X_center) %*% X_center
cov_matrix_center_scale <- 1/5 * t(X_center_scale) %*% X_center_scale

( eigenX <- eigen( cov_matrix_center ) )
```

```
## eigen() decomposition
## $values
## [1] 2.93722954 2.28805812 0.80854251 0.04616983
##
## $vectors
##            [,1]        [,2]       [,3]         [,4]
## [1,]  0.3436766  0.80716469 -0.4461250  0.177042475
## [2,] -0.3682374 -0.17791687 -0.2582385  0.875248373
## [3,] -0.5751820 -0.03345965 -0.6822505 -0.450089261
## [4,]  0.6445566 -0.56188184 -0.5184782  0.003988055
```

```
( best_approx(cov_matrix_center, 1) )
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  0.3469268 -0.3717198 -0.5806216  0.6506523
## [2,] -0.3717198  0.3982846  0.6221155 -0.6971509
## [3,] -0.5806216  0.6221155  0.9717365 -1.0889408
## [4,]  0.6506523 -0.6971509 -1.0889408  1.2202816
```

```
( best_approx(cov_matrix_center, 2) )
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  1.8376307 -0.7003038 -0.6424162 -0.3870534
## [2,] -0.7003038  0.4707118  0.6357364 -0.4684178
## [3,] -0.6424162  0.6357364  0.9742981 -1.0459245
## [4,] -0.3870534 -0.4684178 -1.0459245  1.9426472
```

```
( eigenX <- eigen( cov_matrix_center_scale ) )
```

```
## eigen() decomposition
## $values
## [1] 1.85219898 0.91482763 0.38510998 0.04786341
##
## $vectors
##            [,1]       [,2]       [,3]         [,4]
## [1,]  0.3592298 -0.6917233 -0.5565679  0.287584734
## [2,] -0.6339580  0.1305970 -0.1900625  0.738185562
## [3,] -0.5983454 -0.1695573 -0.4907728 -0.610225552
## [4,]  0.3332176  0.6897200 -0.6428456 -0.001368531
```

```
( best_approx(cov_matrix_center_scale, 1) )
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  0.2390190 -0.4218135 -0.3981182  0.2217114
## [2,] -0.4218135  0.7444038  0.7025869 -0.3912696
## [3,] -0.3981182  0.7025869  0.6631191 -0.3692900
## [4,]  0.2217114 -0.3912696 -0.3692900  0.2056571
```

```
( best_approx(cov_matrix_center_scale, 2) )
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  0.6767468 -0.5044563 -0.2908210 -0.2147486
## [2,] -0.5044563  0.7600067  0.6823293 -0.3088662
## [3,] -0.2908210  0.6823293  0.6894201 -0.4762764
## [4,] -0.2147486 -0.3088662 -0.4762764  0.6408530
```

Problem 2

Making up a matrix:

```
mother_wavelet <- function(x){

  value = ( x > 0 && x <= 1/2 ) - ( x > 1/2 && x <= 1)

  return(value)
}

haar_basis <- function(x){
  index <- 1;
  row <- numeric(16)
  for(m in 0:3)
    for( j in 0:( 2^m - 1 ) ){
      row[index] = sqrt(2^m) * mother_wavelet( (2^m) * ( x - j / (2^m) ) )
      index = index + 1
    }

  return(row)
}

x_trans <- ( data2$x - min(data2$x) )/( max(data2$x) - min(data2$x) )
X_h <- t( apply(as.matrix(x_trans), 1, haar_basis) )
```

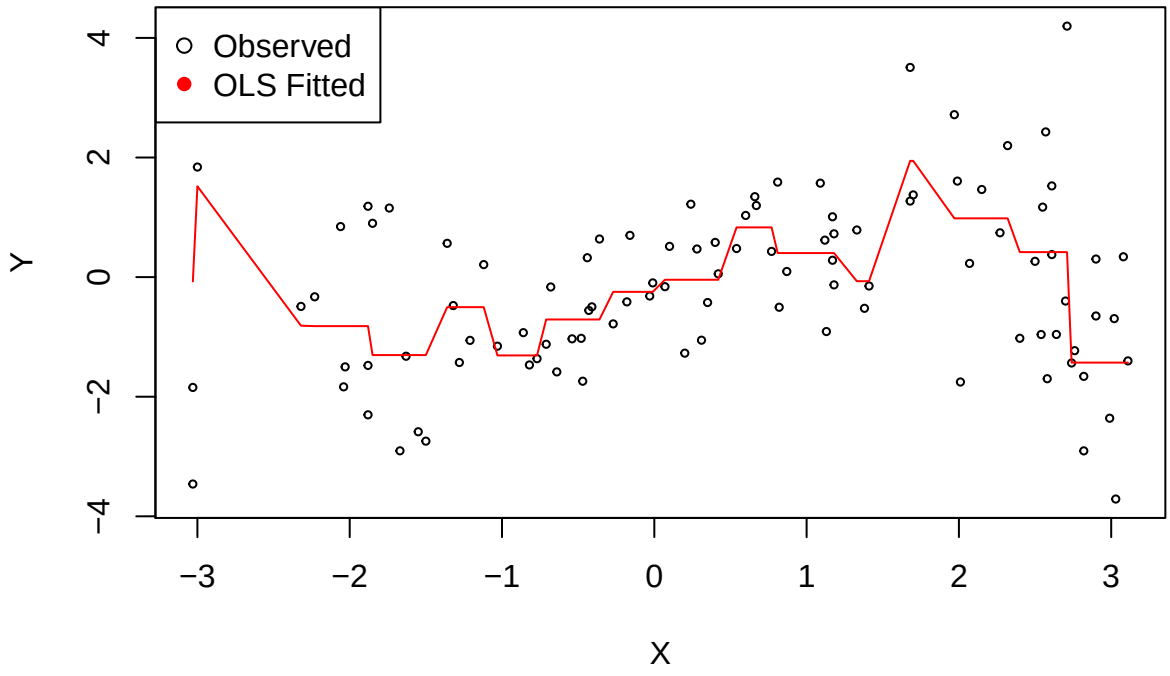
(a)

```
( fit_ols <- lm(y~X_h, data = data2) )
```

```
##
## Call:
## lm(formula = y ~ X_h, data = data2)
##
## Coefficients:
```

```
## (Intercept)         X_h1         X_h2         X_h3         X_h4
##    -0.07184     -0.45131      0.11927     -0.07014      0.35370
##         X_h5         X_h6         X_h7         X_h8         X_h9
##    -0.10709      0.05676      0.49231      0.41207      0.08564
##        X_h10        X_h11        X_h12        X_h13        X_h14
##     0.14280     -0.08164     -0.15481      0.08319      0.16989
##        X_h15        X_h16
##     0.32665           NA
```

```
plot(x = data2$x, y = data2$y, ylab = "Y", xlab = "X",
     pch = 1, cex = 0.5)
lines(x = sort( data2$x ), y = fit_ols$fitted.values[order(data2$x)], pch = 16,
       col = "red", cex = 0.5)
legend("topleft", legend = c("Observed", "OLS Fitted"), col = c("black", "red"),
       pch = c(1, 16))
```



(b)

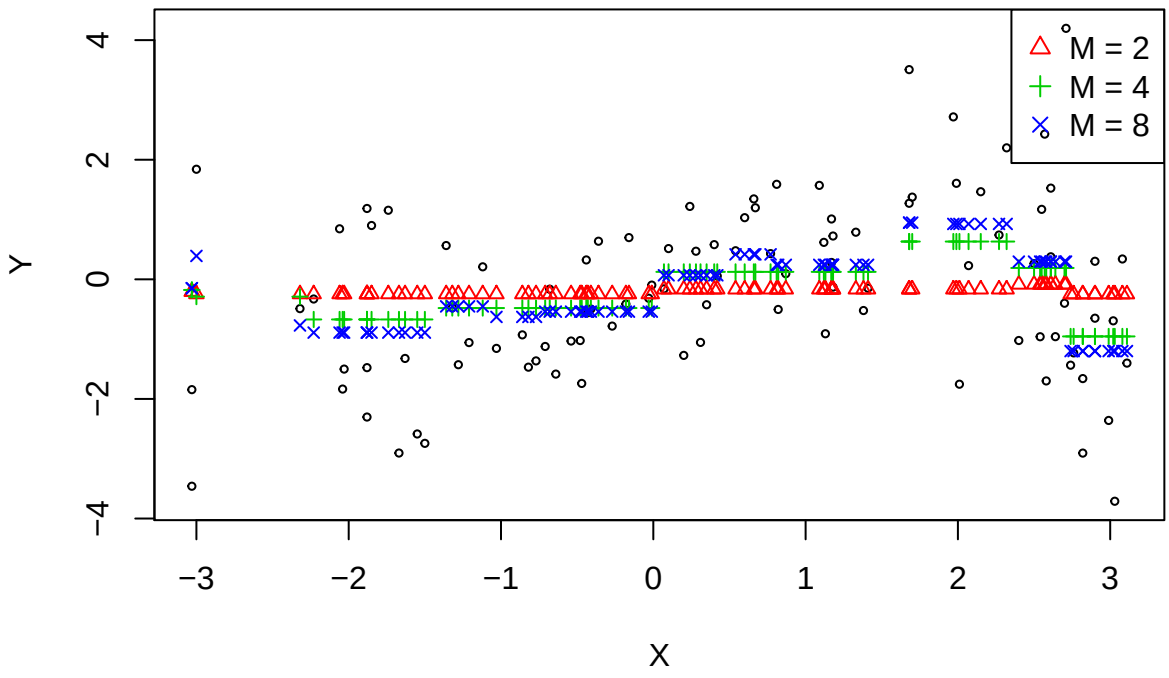
```
y_centered <- data2$y - mean( data2$y )

get_lambda <- function(M)
{
  fit_lasso <- glmnet(X_h, y_centered, standardize = T, pmax = M,nlambda = 20000)
```

```r
  return( min(fit_lasso$lambda) )
}

get_predicted <- function(lambda)
{
  fit_lasso <- glmnet(X_h, y_centered, standardize = T, lambda = lambda)
  value = predict(fit_lasso, X_h)

  return(value)
}

get_coefficient_vector <- function(lambda){
  fit_lasso <- glmnet(X_h, y_centered, standardize = T, lambda = lambda)
  return(fit_lasso$beta)
}

M <- c(2,4,8)
lambda <- apply(as.matrix(M), 1, get_lambda)
value <- apply(as.matrix(lambda), 1, get_predicted) + mean(data2$y)

( coef <- apply( as.matrix(lambda), 1, get_coefficient_vector) )
```

```
## [[1]]
## 16 x 1 sparse Matrix of class "dgCMatrix"
##              s0
## V1  -0.03994013
## V2   .
## V3   .
## V4   .
## V5   .
## V6   .
## V7   .
## V8   .
## V9   .
## V10  .
## V11  .
## V12  .
## V13  .
## V14  .
## V15  0.02880334
## V16  .
##
## [[2]]
## 16 x 1 sparse Matrix of class "dgCMatrix"
##              s0
## V1  -0.30242980
## V2   .
## V3   .
## V4   0.09544829
## V5   .
## V6   .
## V7   0.25349740
## V8   .
```

```
## V9   .
## V10  .
## V11  .
## V12  .
## V13  .
## V14  .
## V15  0.20237856
## V16  .
##
## [[3]]
## 16 x 1 sparse Matrix of class "dgCMatrix"
##               s0
## V1  -0.391467689
## V2   .
## V3   .
## V4   0.175157628
## V5   .
## V6   .
## V7   0.347304902
## V8   0.205501442
## V9   .
## V10  0.031114449
## V11  .
## V12 -0.061899306
## V13  .
## V14  0.003836725
## V15  0.264369490
## V16  .
```

```
plot(x = data2$x, y = data2$y, ylab = "Y", xlab = "X",
     pch = 1, cex = 0.5)

points(x = data2$x, y = value[,1], cex = 0.75, pch = 2, col = 2)
points(x = data2$x, y = value[,2], cex = 0.75, pch = 3, col = 3)
points(x = data2$x, y = value[,3], cex = 0.75, pch = 4, col = 4)

legend("topright", legend = c("M = 2", "M = 4", "M = 8"), col = c(2,3,4), pch = c(2,3,4))
```



Problem 3

```
knot <- c(0, .1, .3, .5, .7, .9, 1)
K <- length(knot)

only_positive <- function(x){
  value <- ifelse(x >= 0, x, 0)

  return(value)
}

make_row <- function(x){
  row <- numeric(K)
  row[1] = 1
  row[2] = x

  for(j in 1:(K - 2)){
    value = only_positive( ( x - knot[j] )^3 )
    -( knot[K] - knot[j] )/( knot[K] - knot[K-1] ) * only_positive( (x - knot[K -1])^3 )
    +( knot[K-1] - knot[j] )/( knot[K] - knot[K-1] ) * only_positive( (x - knot[K])^3 )

    row[j+2] = value
  }
```

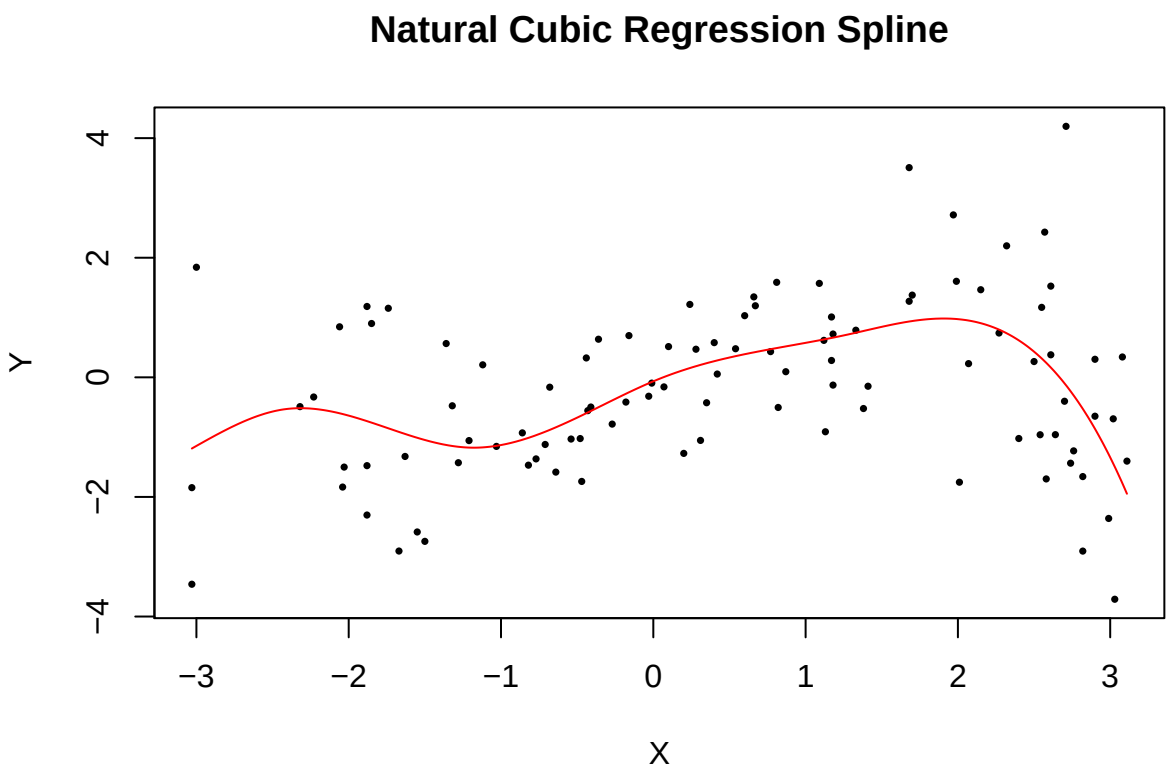
```
  return(row)
}

x_trans <- ( data2$x - min(data2$x) )/( max(data2$x) - min(data2$x) )
X_h_3 <- t( apply(as.matrix(x_trans), 1, make_row) )
```

Then to find $\hat{\beta}^{OLS}$:

```
fit_ols <- lm(data2$y ~ 0 + X_h_3)
fit_ols$coefficients
```

```
##      X_h_31      X_h_32      X_h_33      X_h_34      X_h_35      X_h_36
##   -1.190293    8.843849 -226.408590  434.236477 -343.350843  193.448300
##      X_h_37
## -228.230208
```

```
x_plot <- seq(from = min( data2$x ), to = max( data2$x), length.out = 500)
x_plot_trans <- ( x_plot - min(data2$x) )/( max(data2$x) - min(data2$x) )
x_plot_h <- t( apply(as.matrix(x_plot_trans), 1, make_row) )
y_plot <- x_plot_h %*% fit_ols$coefficients

plot(x = data2$x, y = data2$y, pch = 16, cex = 0.5,
     ylab = "Y", xlab = "X", main = "Natural Cubic Regression Spline")
lines(x = x_plot, y = y_plot, col = "red")
```



Problem 4

(a)

```
y <- c(0, 1.5, 2, 0.5, 0, -0.5,
       0, 1.5, 3.5, 4.5, 3.5)

Omiga <- matrix(0, nrow = N, ncol = N)

for( j in 1:(N-2) )
  for( k in j:(N-2)){
    if(k == j){
      Omiga[j+2, k+2] = 12 * (x[N-1] - x[j])^2 * (x[N] - x[j])
    } else {
      Omiga[j+2, k+2] = 6 * ( x[N-1] - x[k] )^2 * ( 2*x[N-1] + x[k] - 3*x[j] ) +
        12 * ( x[N-1] - x[k] ) * ( x[N-1] - x[j] ) * ( x[N] - x[N-1] )
      Omiga[k+2, j+2] = Omiga[j+2, k+2]
    }
  }
```

```
H_matrix <- matrix(nrow = N, ncol = N)
for(i in 1:N)
  for(j in 1:N){
    H_matrix[i, j] <- compute_H(i, j)
  }
H_matrix
```

```
##       [,1] [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11]
##  [1,]    1  0.0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##  [2,]    1  0.1 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##  [3,]    1  0.2 0.008 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##  [4,]    1  0.3 0.027 0.008 0.001 0.000 0.000 0.000 0.000 0.000 0.000
##  [5,]    1  0.4 0.064 0.027 0.008 0.001 0.000 0.000 0.000 0.000 0.000
##  [6,]    1  0.5 0.125 0.064 0.027 0.008 0.001 0.000 0.000 0.000 0.000
##  [7,]    1  0.6 0.216 0.125 0.064 0.027 0.008 0.001 0.000 0.000 0.000
##  [8,]    1  0.7 0.343 0.216 0.125 0.064 0.027 0.008 0.001 0.000 0.000
##  [9,]    1  0.8 0.512 0.343 0.216 0.125 0.064 0.027 0.008 0.001 0.000
## [10,]    1  0.9 0.729 0.512 0.343 0.216 0.125 0.064 0.027 0.008 0.001
## [11,]    1  1.0 0.990 0.720 0.504 0.336 0.210 0.120 0.060 0.024 0.006
```

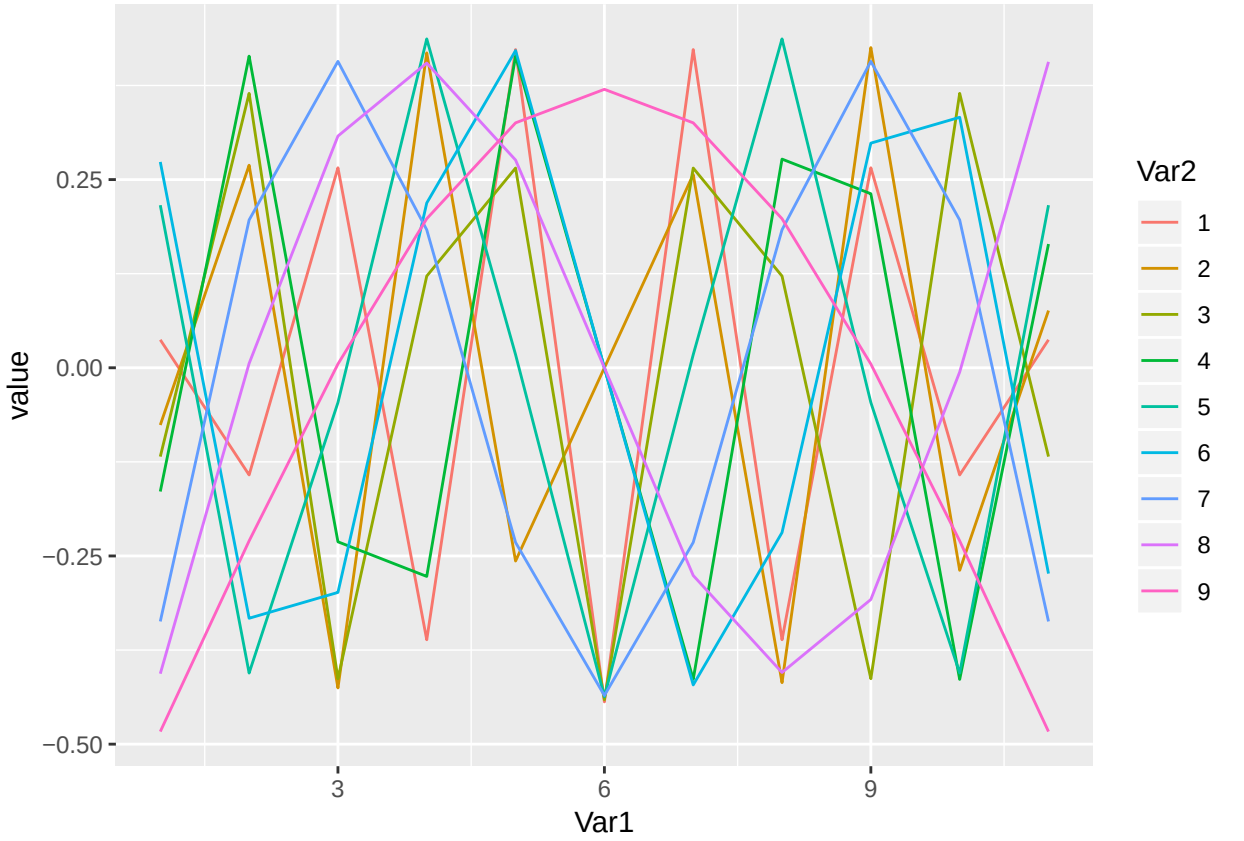
```
K_matrix <- solve( t(H_matrix) ) %*% Omiga %*% solve( H_matrix )
K_matrix <- ( K_matrix + t(K_matrix) )/2
```

(b)

```
K_eigen <- eigen(K_matrix); K_eigen$values
```

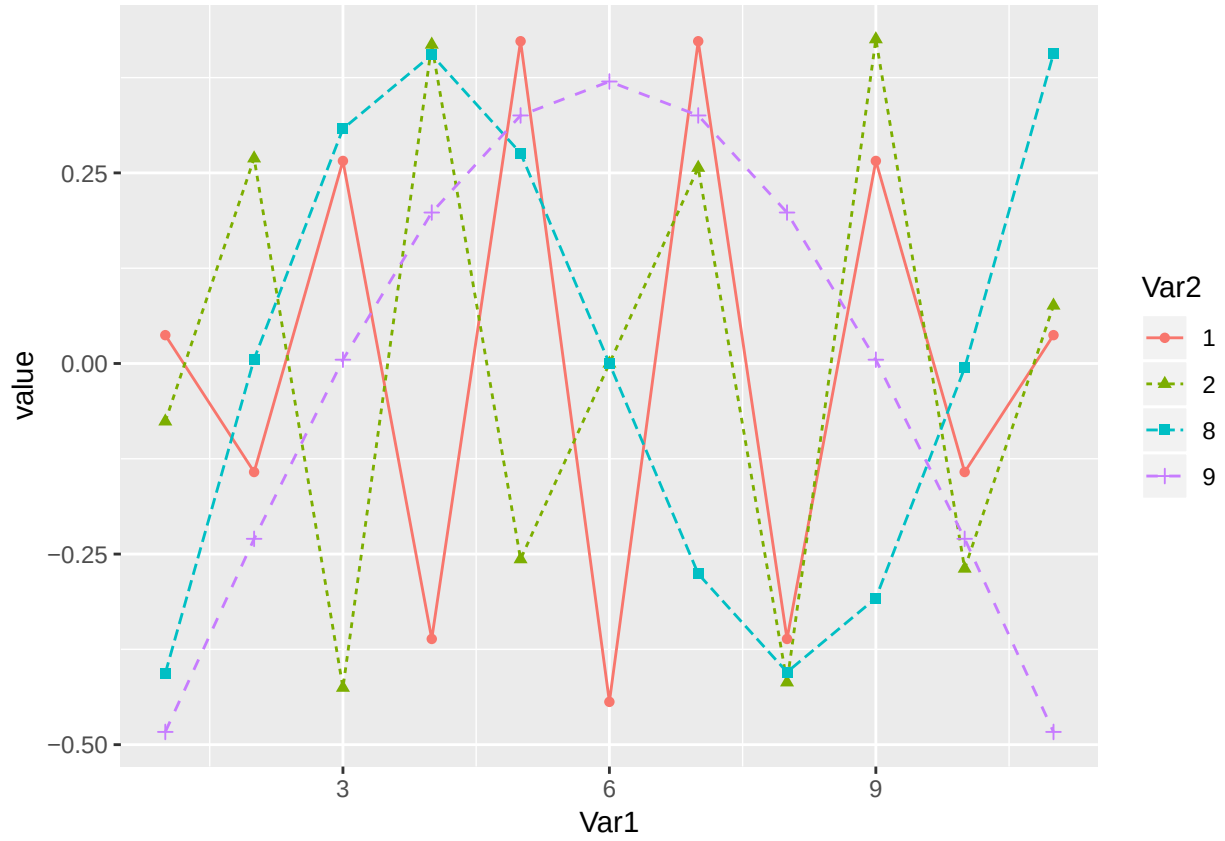
```
##  [1]  4.366292e+04  3.333961e+04  2.200669e+04  1.282846e+04  6.610687e+03
##  [6]  2.938555e+03  1.060305e+03  2.713642e+02  3.508288e+01 -1.958878e-12
## [11] -3.244671e-12
```

```
reshape2::melt(K_eigen$vectors) %>% mutate(Var2 = as.factor(Var2)) %>%
  filter(Var2 %in% 1:9) %>% ggplot() + geom_line(aes(x = Var1, y = value, col = Var2))
```



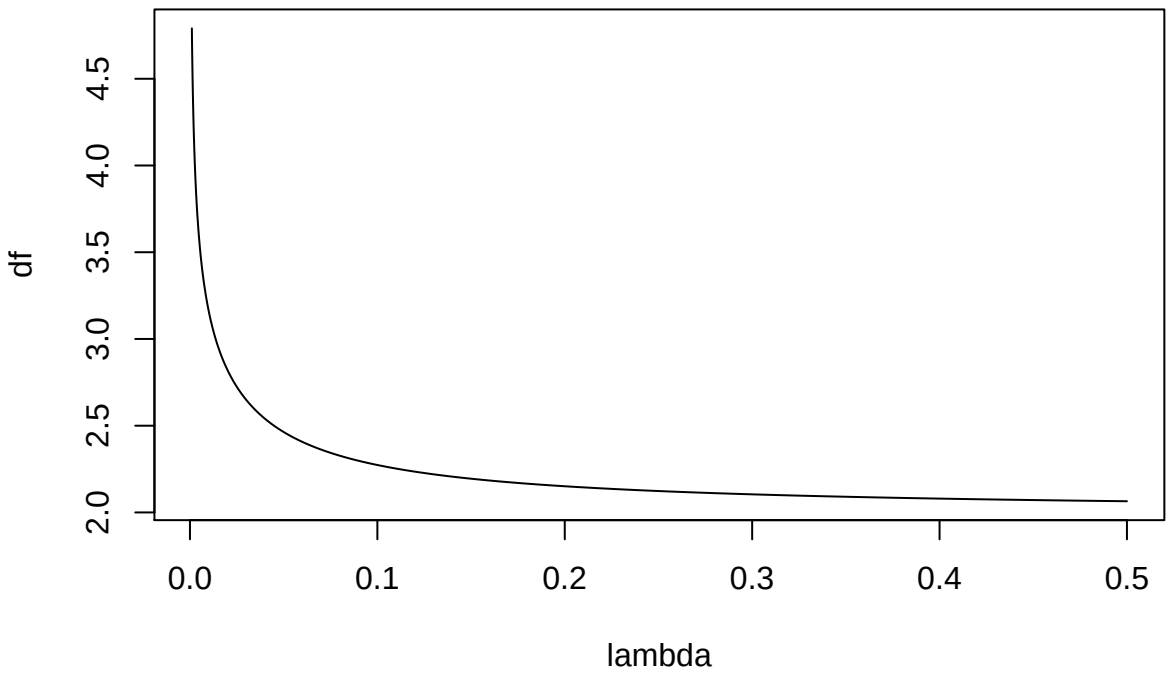
A large eigen value of K corresponds to a small eigen value of S_λ. From the plot below, we can draw the conclusion that the `Y[c(2,3,6,9)]` will get suppressed most in the smoothing.

```
reshape2::melt(K_eigen$vectors) %>%
  mutate(Var2 = as.factor(Var2)) %>%
  filter( Var2 %in% c(1:2, 8:9) ) %>%
  ggplot(aes( x = Var1, y = value, col = Var2)) + geom_line(aes( linetype = Var2 )) +geom_point(aes(pch
```



(c)

```
compute_df <- function(lambda, eta_array, target = 0){
  eta_array[10:11] = 0
  value <- sum( 1/(1 + lambda * eta_array) )

  return( value-target )
}

lambda <- as.matrix( seq(from = 0.001, to = 0.5, by = 0.0001) )
df <- apply(lambda, 1, compute_df, eta_array = K_eigen$values)

plot(x = lambda, y = df, type = "l")
```




```
df <- c(2.5, 3, 4, 5)
lambda_roots_4 <- numeric(length(df))

for(i in 1:length(df)){
  lambda_roots_4[i] <- uniroot(function(x) compute_df(x, eta_array = K_eigen$values, target = df[i]), in
}
lambda_roots_4
```

```
## [1] 0.0448424394 0.0137079634 0.0026479555 0.0008216407
```

Problem 5

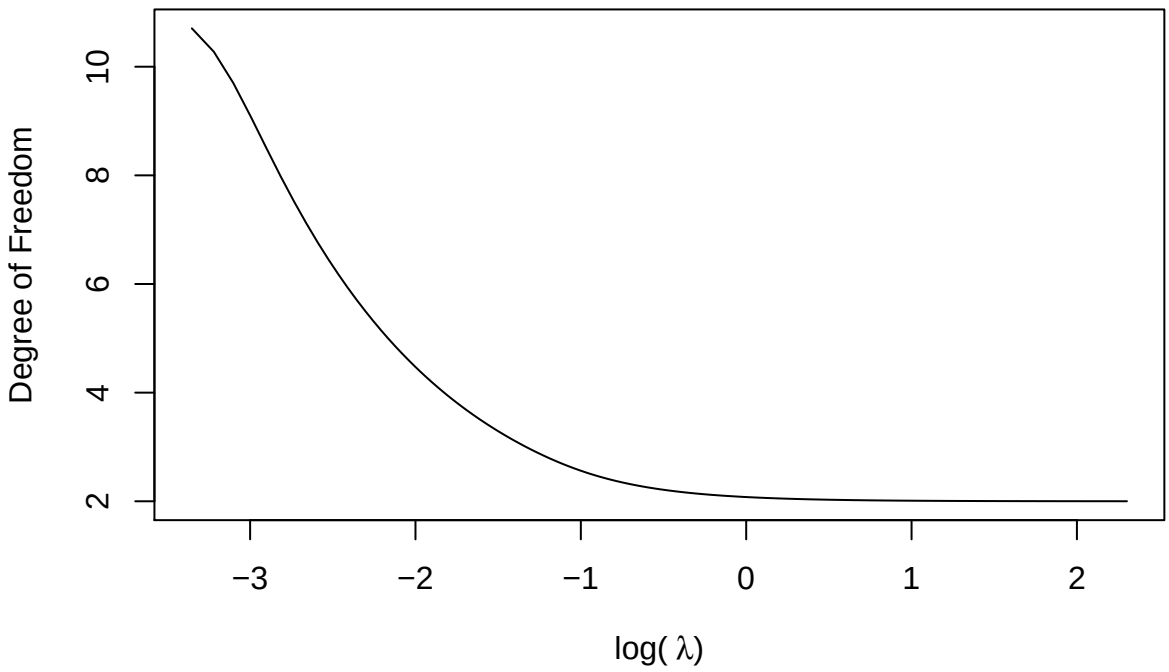
(a)

```
x <- seq(0,1,by=0.1)
N <- length(x)
B <- as.matrix( cbind(1,x) )

compute_K_lambda <- function(lambda, x0, xi){
  value <- dnorm( (x0 - xi)/lambda )

  return(value)
```

```
}

compute_W <- function(x0, lambda){
  value <- numeric(N)
  for(i in 1:N){
    value[i] <- compute_K_lambda(lambda, x0, x[i])
  }

  return(diag(value))
}

compute_I <- function(lambda, x0){
  W <- compute_W(x0, lambda)

  part1 <- matrix(c(1,x0), nrow = 1)
  part2 <- solve( t(B) %*% W %*% B)
  part3 <- t(B) %*% W

  value <- part1 %*% part2 %*% part3

  return(value)
}

get_trace_L <- function(lambda, target = 0){
  L <- apply(as.matrix( x ), 1, compute_I, lambda = lambda)

  return( tr(L) - target )
}
```

This is the plot of effective degree of freedom v.s. $log(\lambda)$:

```
lambda <- seq(0.035,10,length.out = 2000)
trace_I <- apply(as.matrix( lambda ), 1,get_trace_L)
plot(log(lambda), trace_I, ty = "l", ylab = "Degree of Freedom",
     xlab = expression( paste( "log(",~lambda,")" ) )
     )
```



Find out the value of alpha by solving the root:

```
df <- c(2.5, 3, 4, 5)
lambda_roots <- numeric(length(df))

for(i in 1:length(df)){
  lambda_roots[i] <- uniroot(function(x) get_trace_L(x, target = df[i]), interval = c(0.05,4))$root
}

lambda_roots
```

```
## [1] 0.3915776 0.2642124 0.1609244 0.1146486
```

(b)

```
lambda5 <- lambda_roots[3]
lambda4 <- lambda_roots_4[3]

L_lambda5 <- t( apply(as.matrix( x ), 1, compute_I, lambda = lambda5) )
S_lambda4 <- solve( diag(1, nrow = 11) + lambda4*K_matrix )
```

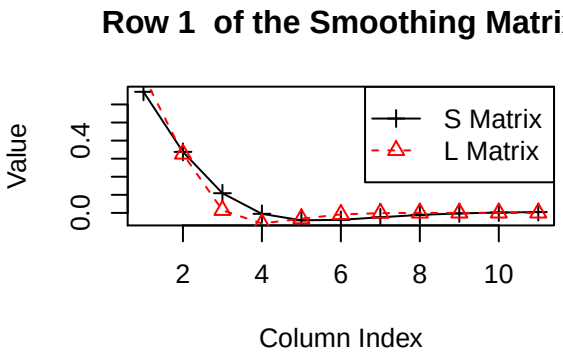
The matrix difference is:

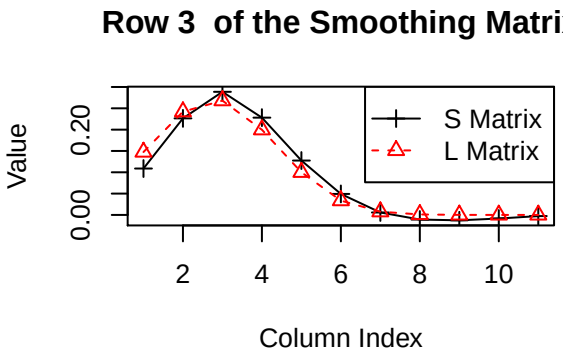
```
L_lambda5 - S_lambda4
```

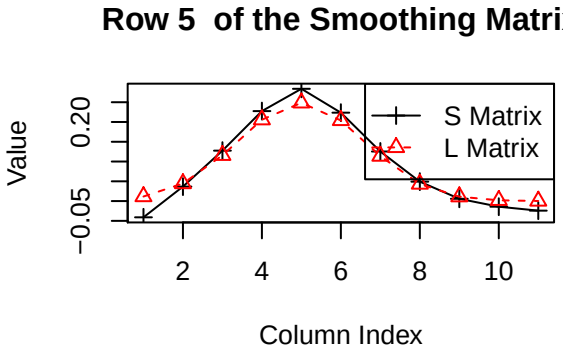
```
##               [,1]         [,2]         [,3]         [,4]         [,5]
##  [1,]  0.090027536 -0.011577964 -0.094409573 -0.053444610  0.009269582
##  [2,]  0.028821277  0.023347628 -0.019283615 -0.042654653 -0.026454779
##  [3,]  0.038764026  0.015740216 -0.020661092 -0.028294980 -0.027429832
##  [4,]  0.052673668  0.007797074 -0.015630840 -0.036983145 -0.025077386
##  [5,]  0.052563434  0.007659622 -0.011710767 -0.021773047 -0.035424474
##  [6,]  0.040489059  0.013278246 -0.005564228 -0.012537779 -0.019086010
##  [7,]  0.024457299  0.016186744  0.005322851 -0.006004389 -0.011640453
##  [8,]  0.011086811  0.013664341  0.012591005  0.004407607 -0.008059208
##  [9,]  0.002671861  0.008414849  0.012715407  0.011936735  0.001739538
## [10,] -0.001816245  0.003047093  0.008382383  0.013235838  0.013155994
## [11,] -0.004492881 -0.001816568  0.002662322  0.010931083  0.022859007
##                [,6]         [,7]         [,8]         [,9]        [,10]
##  [1,]  0.0299920680  0.022859007  0.010931083  0.002662322 -0.001816568
##  [2,]  0.0002190787  0.013155994  0.013235838  0.008382383  0.003047093
##  [3,] -0.0155967283  0.001739538  0.011936735  0.012715407  0.008414849
##  [4,] -0.0164699248 -0.008059208  0.004407607  0.012591005  0.013664341
##  [5,] -0.0196368197 -0.011640453 -0.006004389  0.005322851  0.016186744
##  [6,] -0.0331585750 -0.019086010 -0.012537779 -0.005564228  0.013278246
##  [7,] -0.0196368197 -0.035424474 -0.021773047 -0.011710767  0.007659622
##  [8,] -0.0164699248 -0.025077386 -0.036983145 -0.015630840  0.007797074
##  [9,] -0.0155967283 -0.027429832 -0.028294980 -0.020661092  0.015740216
## [10,]  0.0002190787 -0.026454779 -0.042654653 -0.019283615  0.023347628
## [11,]  0.0299920680  0.009269582 -0.053444610 -0.094409573 -0.011577964
##               [,11]
##  [1,] -0.004492881
##  [2,] -0.001816245
##  [3,]  0.002671861
##  [4,]  0.011086811
##  [5,]  0.024457299
##  [6,]  0.040489059
##  [7,]  0.052563434
##  [8,]  0.052673668
##  [9,]  0.038764026
## [10,]  0.028821277
## [11,]  0.090027536
```

The plots are:

```
par(mfrow = c(2,2))

for(i in c(1,3,5)){
  plot(x = 1:11, y = S_lambda4[i, ], xlab = "Column Index", col = 1,
     main = paste("Row", i, " of the Smoothing Matrix"), pch = 3, ylab = "Value")
  lines(x = 1:11, y = S_lambda4[i, ], col = 1, lty = 1)

  points(x = 1:11, y = L_lambda5[i, ], pch = 2, col = 2)
  lines(x = 1:11, y = L_lambda5[i, ], col = 2, lty = 2)

  legend("topright", legend = c("S Matrix", "L Matrix"), pch = c(3, 2), col = c(1, 2), lty = c(1, 2))
}
```







From the plots above, the two smoothing matrices are close to each other when the effective degree of freedom is the same.

6

(a)

The largest effective degrees of freedom I can get for the locally weighted linear regression smoother with tricube kernel is about 4.72. If I try to go over it, the effective degrees of freedom will jump to 11, which means the fitted values are the same as the observed values.

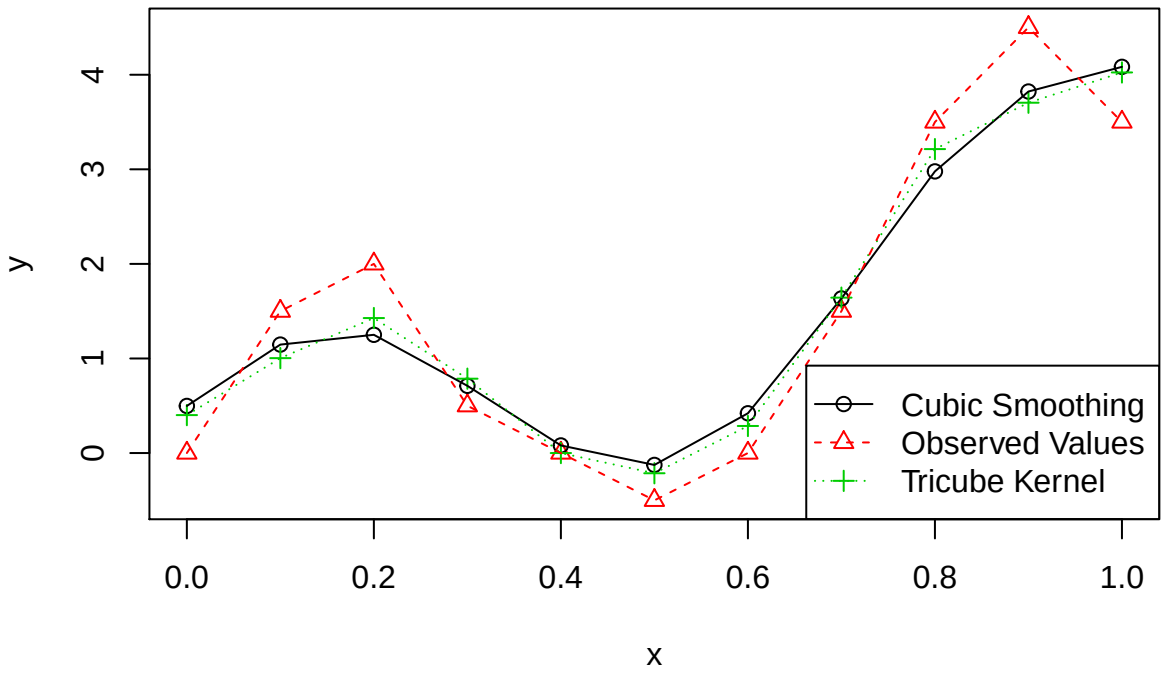
```
lambda_df5 <- uniroot(function(lambda) smooth.spline(x, y, lambda = lambda)$df-5,
                      interval = c(0.0001, 1) )$root
lambda_df9 <- uniroot(function(lambda) smooth.spline(x, y, lambda = lambda)$df-9,
                      interval = c(0.000001, 1))$root
cubic_smoothing_5 <- smooth.spline(x, y, lambda = lambda_df5)
cubic_smoothing_9 <- smooth.spline(x, y, lambda = lambda_df9)

tricube_5 <- loess(y~x, degree = 1, enp.target = 5)
( tricube_9 <- loess(y~x, degree = 1, enp.target = 9) )
```

```
## Call:
## loess(formula = y ~ x, enp.target = 9, degree = 1)
##
```

```
## Number of Observations: 11
## Equivalent Number of Parameters: 11
## Residual Standard Error: NaN
```

This is the plot for approximately 5 effective degrees of freedom:

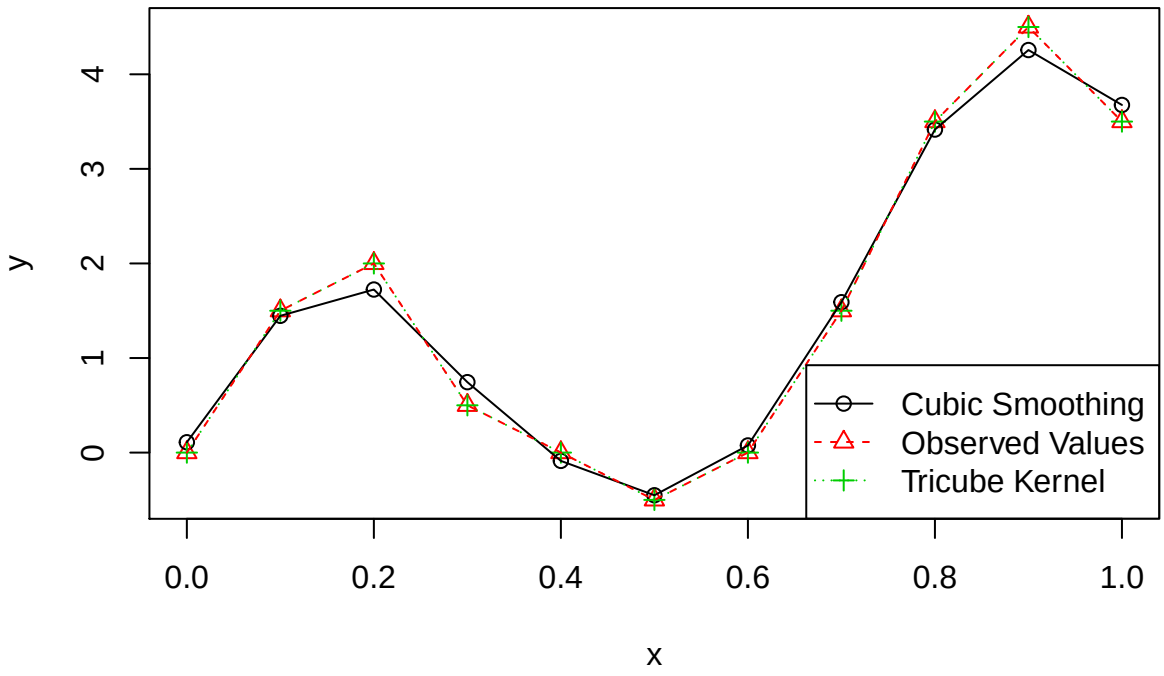
```
plot(x, y, pch = 2, col = 2)
points(x, cubic_smoothing_5$y, ylab = "Y", xlab = "x")
points(x, tricube_5$fitted, pch = 3, col = 3)

lines(x, cubic_smoothing_5$y)
lines(x, tricube_5$fitted, lty = 3, col = 3)
lines(x, y, lty = 2, col = 2)
legend("bottomright", legend = c("Cubic Smoothing", "Observed Values", "Tricube Kernel"),
       pch = c(1, 2, 3), col = c(1, 2, 3), lty = c(1, 2, 3))
```



This is the plot for approximately 9 effective degrees of freedom:

```
plot(x, y, pch = 2, col = 2)
points(x, cubic_smoothing_9$y, ylab = "Y", xlab = "x")
points(x, tricube_9$fitted, pch = 3, col = 3)

lines(x, cubic_smoothing_9$y)
lines(x, tricube_9$fitted, lty = 3, col = 3)
lines(x, y, lty = 2, col = 2)
```

```
legend("bottomright", legend = c("Cubic Smoothing", "Observed Values", "Tricube Kernel"),
       pch = c(1, 2, 3), col = c(1, 2, 3), lty = c(1, 2, 3))
```



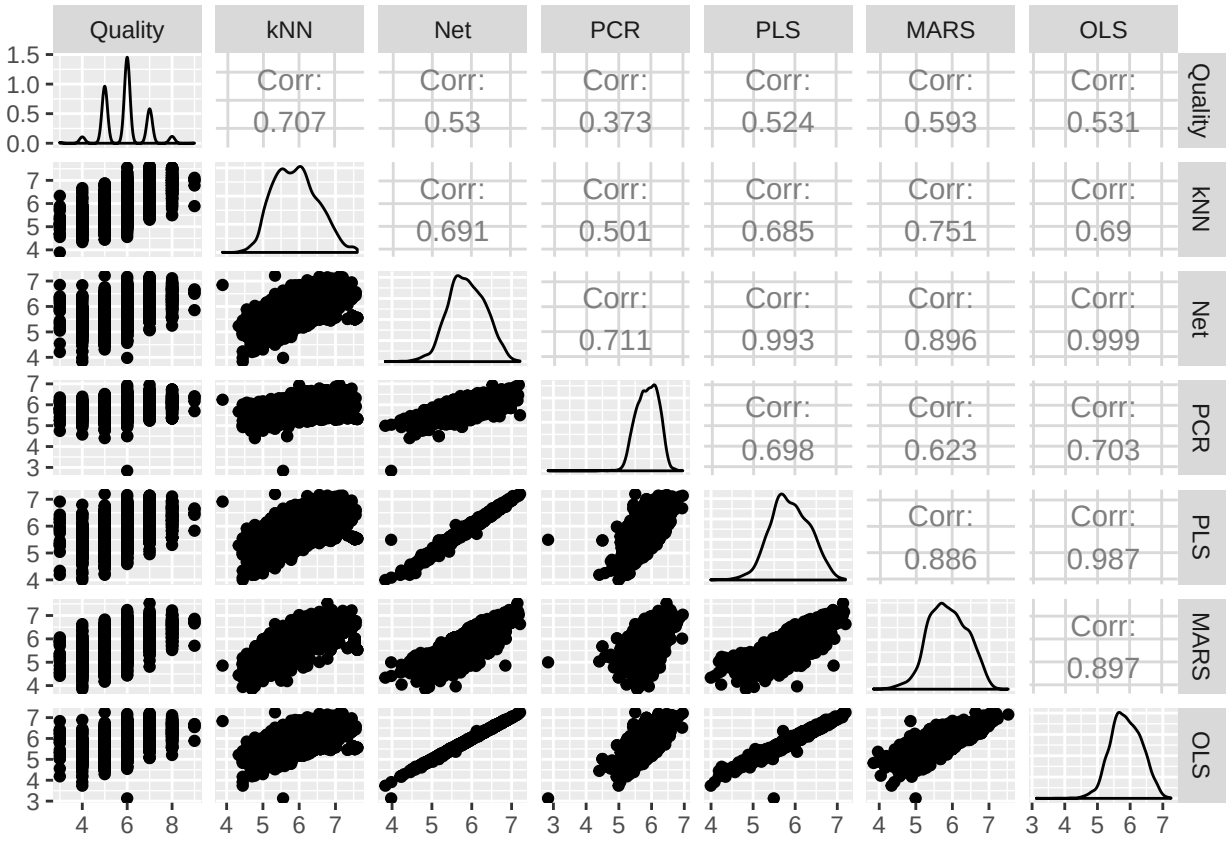
We can see that, the fitted value from locally weighted linear regression is the same as the observed one. The reason for that is explained above. Maybe it is caused by some numerical issues.

Problem 7

```
wineknn<- train(y=wine$quality,
                x=wine[,-12],
                method="knn",
                preProcess=c("center","scale"),
                trControl=trainControl(method="repeatedcv",repeats=100,number=10))

winenet<- train(y=wine$quality,
                x=wine[,-12],
                method="glmnet",
                preProcess=c("center","scale"),
                trControl=trainControl(method="repeatedcv",repeats=100,number=10))

winepcr<- train(y=wine$quality,
                x=wine[,-12],
                method="pcr",
```

```
                preProcess=c("center","scale"),
                trControl=trainControl(method="repeatedcv",repeats=100,number=10))

winepls<- train(y=wine$quality,
                x=wine[,-12],
                method="pls",
                preProcess=c("center","scale"),
                trControl=trainControl(method="repeatedcv",repeats=100,number=10))

winemars<- train(y=wine$quality,
                 x=wine[,-12],
                 method="earth",
                 preProcess=c("center","scale"),
                 trControl=trainControl(method="repeatedcv",repeats=100,number=10))

wineols<- lm(quality~.,data=wine)
```

```
winepred<- sapply(list(wineknn,winenet,winepcr,winepls,winemars,wineols),predict)
colnames(winepred)<- c("kNN","Net","PCR","PLS","MARS","OLS")

res<- data.frame(cbind(Quality=wine$quality,winepred))

ggpairs(res)
```



```
round(cor(res),digits=2)
```

```
##         Quality  kNN  Net  PCR  PLS MARS  OLS
## Quality    1.00 0.71 0.53 0.37 0.52 0.59 0.53
## kNN        0.71 1.00 0.69 0.50 0.69 0.75 0.69
## Net        0.53 0.69 1.00 0.71 0.99 0.90 1.00
## PCR        0.37 0.50 0.71 1.00 0.70 0.62 0.70
## PLS        0.52 0.69 0.99 0.70 1.00 0.89 0.99
## MARS       0.59 0.75 0.90 0.62 0.89 1.00 0.90
## OLS        0.53 0.69 1.00 0.70 0.99 0.90 1.00
```

Problem 8

(a)

```
x <- seq(0, 1, by = 0.1)
y <- c(0, 1.5, 2, 0.5, 0, -0.5, 0, 1.5, 3.5, 4.5, 3.5)

x_std <- scale(x, center = T, scale = T)
y_ctr <- scale(y, center = T, scale = FALSE)

K <- as.matrix( -1/2 * dist(x_std)^2 ) %>% exp

G <- matrix(nrow = 11, ncol = 11)
for(i in 1:11)
  for(j in 1:11){
    G[i, j] = K[i, j] - colMeans(K)[i] - rowMeans(K)[j] + mean(K)
  }
G
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
##  [1,]  0.72982684  0.59948554  0.40416853  0.17923959 -0.03591672
##  [2,]  0.59948554  0.55801817  0.44007432  0.26283433  0.05920771
##  [3,]  0.40416853  0.44007432  0.41100439  0.31113756  0.15519990
##  [4,]  0.17923959  0.26283433  0.31113756  0.30014465  0.22158014
##  [5,] -0.03591672  0.05920771  0.15519990  0.22158014  0.23188956
##  [6,] -0.20963926 -0.13330264 -0.02578076  0.08828844  0.17597102
##  [7,] -0.32445509 -0.28406202 -0.19532794 -0.06972905  0.06564248
##  [8,] -0.37719216 -0.37623188 -0.32344136 -0.21663027 -0.06972905
##  [9,] -0.37505911 -0.40766663 -0.39430890 -0.32344136 -0.19532794
## [10,] -0.33090034 -0.38745656 -0.40766663 -0.37623188 -0.28406202
## [11,] -0.25955782 -0.33090034 -0.37505911 -0.37719216 -0.32445509
##              [,6]        [,7]        [,8]        [,9]       [,10]
##  [1,] -0.20963926 -0.32445509 -0.37719216 -0.37505911 -0.33090034
##  [2,] -0.13330264 -0.28406202 -0.37623188 -0.40766663 -0.38745656
##  [3,] -0.02578076 -0.19532794 -0.32344136 -0.39430890 -0.40766663
##  [4,]  0.08828844 -0.06972905 -0.21663027 -0.32344136 -0.37623188
##  [5,]  0.17597102  0.06564248 -0.06972905 -0.19532794 -0.28406202
##  [6,]  0.20892640  0.17597102  0.08828844 -0.02578076 -0.13330264
##  [7,]  0.17597102  0.23188956  0.22158014  0.15519990  0.05920771
```

```
##  [8,]  0.08828844  0.22158014  0.30014465  0.31113756  0.26283433
##  [9,] -0.02578076  0.15519990  0.31113756  0.41100439  0.44007432
## [10,] -0.13330264  0.05920771  0.26283433  0.44007432  0.55801817
## [11,] -0.20963926 -0.03591672  0.17923959  0.40416853  0.59948554
##             [,11]
##  [1,] -0.25955782
##  [2,] -0.33090034
##  [3,] -0.37505911
##  [4,] -0.37719216
##  [5,] -0.32445509
##  [6,] -0.20963926
##  [7,] -0.03591672
##  [8,]  0.17923959
##  [9,]  0.40416853
## [10,]  0.59948554
## [11,]  0.72982684
```

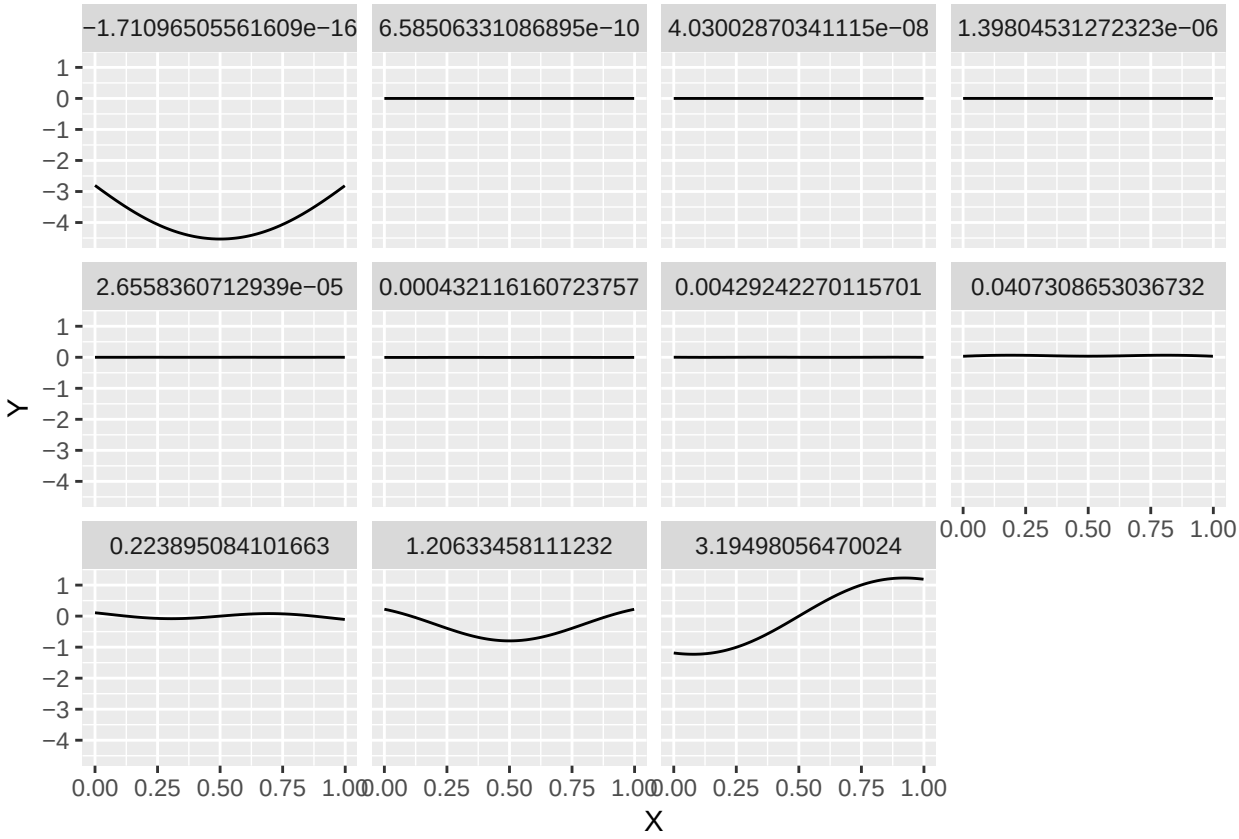
(c)

```
G_eigen <- eigen(G)

compute_T <- function(z, x){
  return( exp(-1/2 * ( x - z )^2 ) )
}

compute_S <- function(z, i){
  T_xi <- compute_T( z, x_std[i] )

  M <- 0
  for(j in 1:11){
    M <- M + compute_T( z, x_std[j] )
  }
  M <- M/11

  return(T_xi + M)
}
```

The function is becoming closer to 0 with decreasing eigenvalues, but the smallest eigen value is an outlier.

```
X_vector <- NULL
Y_vector <- NULL
Eigen_vector <- NULL

for(i in 1:11){
  eigen_value <- G_eigen$values[i]


  x_plot <- seq(from = min( x_std ), to = max( x_std ), by = 0.01)
  y_plot <- numeric( length(x_plot) )

  Eigen_vector <- c( Eigen_vector, rep(eigen_value, times = length(x_plot)) )
  X_vector <- c(X_vector, x_plot)
```

```
  for(ii in 1:length(x_plot) ){
    y_value <- 0
    for(jj in 1:11){
      y_value <- y_value + G_eigen$vectors[,i][jj] * compute_S( x_plot[ii], jj )
    }
    y_plot[ii] <- y_value
  }

  Y_vector <- c(Y_vector, y_plot)
}

X_vector <- ( sd(x) * X_vector + mean(x) )

plot_frame <- data.frame(X = X_vector, Y = Y_vector, Lambda = Eigen_vector)

ggplot(data = plot_frame, mapping = aes(x = X, y = Y) )+ geom_line() + facet_wrap(~Lambda)
```



(d)

The eigen decomposition is equivalent to SVD in this problem. The following is the plot:

```
compute_S065 <- function(y)
{
  value <- compute_T(y, 0.65)
```

```
  for(i in 1:11){
    value <- value - 1/11 * compute_T( y, x_std[i] )
  }

  return(value)
}

for(i in 1:length(x_plot))
{
  y_plot[i] <- compute_S065(x_plot[i])
}

kernel_inner <- function(x, y){
  return( exp(-(x - y)^2/2) )
}

S_65_S_xj <- function(j){
  value <- kernel_inner(0.65, x_std[j])

  part2 <- 0
  for(i in 1:11){
    part2 <- part2 + kernel_inner(x_std[i], 0.65) + kernel_inner(x_std[i], x_std[j])
  }
  part2 <- -1/11 * part2

  part3 <- 0
  dict <- expand.grid(x_std, x_std)
  for(i in 1:121){
    part3 <- part3 + kernel_inner(dict[i,1], dict[i,2])
  }
  part3 <- part3 / 121

  return(value + part2 + part3)
}

compute_s_weight <- function(y, j)
{
  weight <- rowSums(G_eigen$vectors)
  value <- weight[j] * S_65_S_xj(j) * compute_S(y, j)
  return(value)
}

final_function <- function(y){
  value = 0;
  for(i in 1:11){
    value = value + compute_s_weight(y, i)
  }
  return(value)
}

y_proj <- numeric(length(x_plot))
for(i in 1:length(x_plot)){
  y_proj <- final_function(x_plot)
```

```
}

data.frame(X = sd(x) * x_plot+mean(x), Original = y_plot, Projection = y_proj) %>% reshape2::melt(id.va
```