Homework 2

Xingche Guo and Yueying Wang

3/3/2019

Problem 1

4.3

```
Housing_data <- read_xlsx('/Users/apple/Desktop/ISU 2019 spring/STAT 602/hw/hw2/AmesHousingData.xlsx',
                          col_names  = TRUE)

Y <- Housing_data['Price']
X <- Housing_data[,c('Size','Fireplace','Bsmt Bath','Land')]
n <- nrow(X)
CV_int <- train(y=Y$Price,x=data.frame(int=rep(1,n)),method="lm",
               trControl=trainControl(method="repeatedcv",repeats=100,number=8))

input <- list(1,2,3,4,c(1,2),c(1,3),c(1,4),c(2,3),c(2,4),c(3,4),
              c(1,2,3),c(1,2,4),c(1,3,4),c(2,3,4),c(1,2,3,4))
RMSPE <- lapply(input, FUN = function(var_ind){
  CV_tmp <- train(y=Y$Price, x=X[,var_ind], method='lm',
                  trControl=trainControl(method="repeatedcv",repeats=100,number=8))
  return(CV_tmp$results$RMSE)
})

Var_names <- sapply(input, FUN = function(var_ind){
  return(paste(colnames(X)[var_ind], collapse=','))
})
names(RMSPE) <- unlist(Var_names)
```

It seems that the model with all four variables has the smallest RMSPE.

```
RMSPE
```

```
## $Size
## [1] 27555.93
##
## $Fireplace
## [1] 28637.79
##
## $`Bsmt Bath`
## [1] 35778.46
##
## $Land
## [1] 33581.68
##
## $`Size,Fireplace`
## [1] 23344.96
##
## $`Size,Bsmt Bath`
## [1] 26735.65
```

```
##
## $`Size,Land`
## [1] 25304.33
##
## $`Fireplace,Bsmt Bath`
## [1] 28167.37
##
## $`Fireplace,Land`
## [1] 27672.83
##
## $`Bsmt Bath,Land`
## [1] 32574.79
##
## $`Size,Fireplace,Bsmt Bath`
## [1] 22731.82
##
## $`Size,Fireplace,Land`
## [1] 22327.4
##
## $`Size,Bsmt Bath,Land`
## [1] 24430.78
##
## $`Fireplace,Bsmt Bath,Land`
## [1] 27088.84
##
## $`Size,Fireplace,Bsmt Bath,Land`
## [1] 21807.69
```

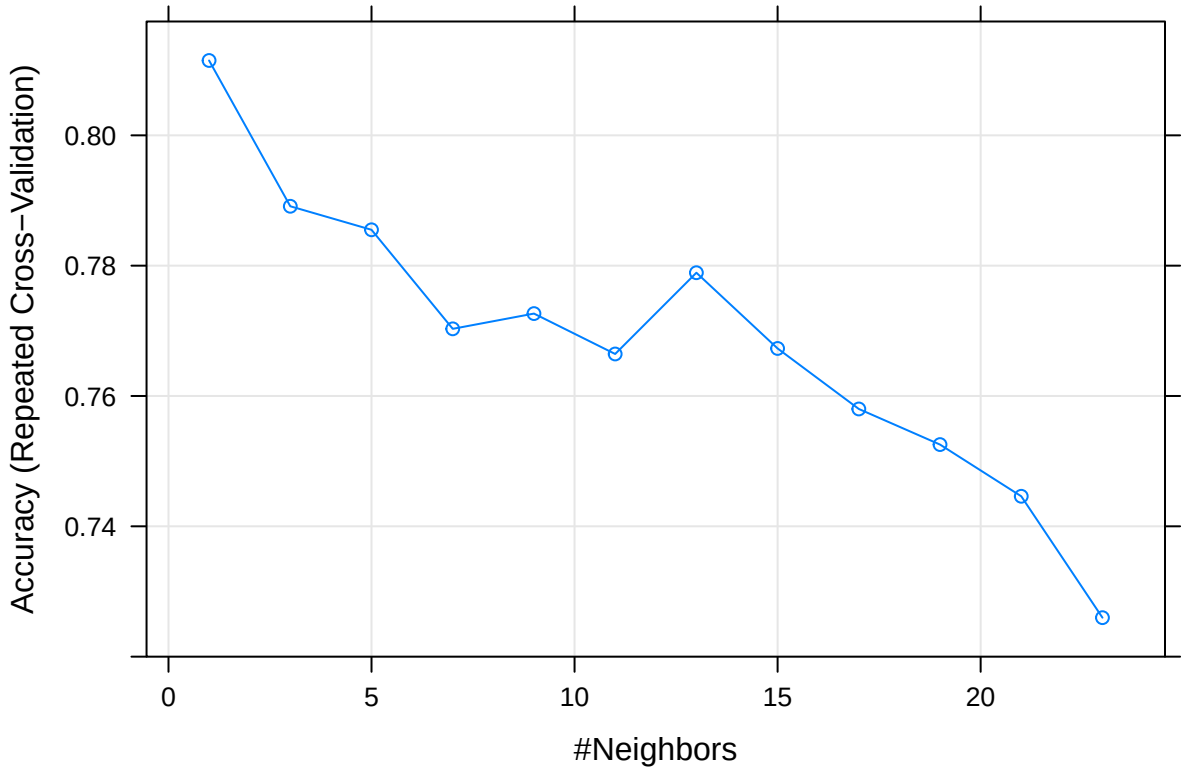
4.4

(a).

```
glass <- read.table(file = "/Users/apple/Desktop/ISU 2019 spring/STAT 602/hw/hw2/glass.data.txt",
                    sep = ",")
names(glass) <- c("ID", "RI", "Na", "Mg", "Al", "Si",
                  "K", "Ca", "Ba", "Fe", "Type")
glass1 <- glass[ (glass$Type==1) | (glass$Type==2)  ,-1]
glass1$Type <- as.factor( glass1$Type )


# define training control
train_control <- trainControl(method="repeatedcv", number=10, repeats = 50)

# find best n (in KNN)
grid = expand.grid(k = (2*(1:12)-1) )

train_knn <- train(Type~., data=glass1, trControl=train_control, method="knn",
                   preProcess = c("center", "scale"),
                   tuneGrid = grid)


train_knn$results
```

```
##     k  Accuracy     Kappa AccuracySD   KappaSD
## 1   1 0.8114857 0.6229377 0.09782653 0.1955420
## 2   3 0.7891143 0.5786942 0.10057300 0.2005931
## 3   5 0.7855048 0.5722567 0.10465406 0.2081806
## 4   7 0.7703048 0.5419399 0.10577896 0.2104647
## 5   9 0.7726476 0.5473279 0.10846361 0.2150935
## 6  11 0.7664476 0.5357521 0.10389117 0.2059242
## 7  13 0.7789143 0.5599157 0.10378756 0.2061964
## 8  15 0.7672952 0.5371540 0.09953091 0.1971601
## 9  17 0.7580095 0.5187649 0.10382346 0.2054952
## 10 19 0.7525429 0.5081478 0.10291389 0.2037101
## 11 21 0.7446095 0.4925552 0.10625434 0.2104828
## 12 23 0.7259810 0.4560607 0.11003359 0.2175359
```

```
plot(train_knn)
```



(b).

There is no need to make any modification. The frequency of Type = 2 in the training set is $\pi^* = 76/146$ while the real value is $\pi = 0.3$. So according to section 1.5.1, to better account for the difference between π^* and π, the modification of the classification rule is to classify the case to Type 2 if

$$t(x)(1-\pi^*)\pi > (k - t(x))\pi^*(1-\pi).$$

When $k = 1$, the modified rule is to classify it to glass Type 2 if

$$\Rightarrow t(x) > 0.7170.$$

Because $t(x) = 0$ or 1, it makes no difference whether we use $t(x) > \frac{k}{2} = \frac{1}{2}$ or $t(x) > 0.7170$ for classification.

5.1

(a)

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


# QR Decomposition
qr_fit <- qr(X)
Q <- qr.Q(qr_fit)     # Basis for C(X)
Q
```

```
##            [,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
```

```
R <- qr.R(qr_fit)
R
```

```
##           [,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
```

```
# Q %*% R

# Singular Value Decomposition
svd_fit <- svd(X)
U <- svd_fit$u       # Basis for C(X)
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 <- svd_fit$v
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 <- diag(svd_fit$d)
D
```

```
##          [,1]     [,2]     [,3]      [,4]
## [1,] 16.58126 0.000000 0.000000 0.0000000
## [2,]  0.00000 3.784298 0.000000 0.0000000
## [3,]  0.00000 0.000000 3.382091 0.0000000
## [4,]  0.00000 0.000000 0.000000 0.5499212
```

```
# U %*% D %*% t(V)
```

(b)

```
eigen(t(X)%*%X)
```

```
## eigen() decomposition
## $values
## [1] 274.9381384  14.3209094  11.4385389   0.3024134
##
## $vectors
##            [,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
```

```
eigen(X%*%t(X))
```

```
## eigen() decomposition
## $values
## [1] 2.749381e+02 1.432091e+01 1.143854e+01 3.024134e-01 7.035877e-15
##
## $vectors
##           [,1]       [,2]       [,3]        [,4]        [,5]
## [1,] 0.4993567 -0.5283913  0.1647824  0.66384835  0.05998042
## [2,] 0.5040186  0.5891454  0.1259937  0.11349453 -0.60837281
## [3,] 0.4582533 -0.4878118 -0.2526166 -0.64627429 -0.26562756
## [4,] 0.3913976  0.3254921 -0.6846703  0.08815998  0.51411787
## [5,] 0.3652670  0.1726408  0.6514474 -0.34782427  0.53982376
```

The eigenvectors of $X'X$ is V, the eigenvalues of $X'X$ is the square of the singular values of X; The first four eigenvectors of XX' is U, the first four eigenvalues of XX' is the square of the singular values of X, the last eigenvalue is 0.

(c)

```
rank_approx <- function(X, r){
  svd_fit.tmp <- svd(X)
  Ur <- matrix(svd_fit.tmp$u[,1:r], ncol = r)
  Vr <- matrix(svd_fit.tmp$v[,1:r], ncol = r)
  Dr <- diag(svd_fit.tmp$d[1:r], ncol = r)
  Xr <- Ur %*% Dr %*% t(Vr)
  return(Xr)
}

rank_approx(X, r=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_approx(X, r=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)

```
X1 <- apply(X, 2, FUN=function(x){
  return(x - mean(x))
})

# X1 <- scale(X, center = TRUE, scale = FALSE)


svd_fit1 <- svd(X1)
U1 <- svd_fit1$u
D1 <- diag(svd_fit1$d)
V1 <- svd_fit1$v    # V1's Columns are the Principal Components Directions
U1 %*% D1    # Columns of U1D1 are the Principal Components
```

```
##            [,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
```

```
# First column of V1 is the loadings for the 1st principal component
```

(e)

```
rank_approx(X1, r=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
```

```
rank_approx(X1, r=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
```

(f)

```
cov_X1 <- t(X1) %*% X1 / 5
# eigenvectors same as columns of V1
# eigenvalues are diag(D1^2/5)   merely the same as eigen(cov_X1), only sign difference

V1
```

```
##            [,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
```

```
diag(D1^2/5)
```

```
## [1] 2.93722954 2.28805812 0.80854251 0.04616983
```

```
eigen(cov_X1)
```

```
## 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
```

```
rank_approx(cov_X1, r=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
```

```
rank_approx(cov_X1, r=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
```

re-do (d), (e), (f) for standardized X

```
X2 <- apply(X1, 2, FUN = function(x){
  return(x/sqrt(sum(x^2)/length(x)))
})

# (d) standardized

svd_fit2 <- svd(X2)
U2 <- svd_fit2$u
D2 <- diag(svd_fit2$d)
V2 <- svd_fit2$v    # V2's Columns are the Principal Components Directions
U2 %*% D2    # Columns of U2D2 are the Principal Components
```

```
##            [,1]         [,2]       [,3]        [,4]
## [1,] -2.0366626 -0.008409157  0.2172132 -0.35532999
## [2,]  1.0248586  0.537502038  1.2208720  0.10866898
## [3,] -1.4962977 -0.821432250 -0.2474691  0.37221839
## [4,]  1.9589337 -1.388628580 -0.3725941 -0.14836226
## [5,]  0.5491681  1.680967950 -0.8180219  0.02280489
```

```
# First column of V2 is the loadings for the 1st principal component

# (e) standardized

rank_approx(X2, r=1)
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.7316300  1.2911586  1.2186277 -0.6786519
## [2,]  0.3681598 -0.6497173 -0.6132194  0.3415010
## [3,] -0.5375148  0.9485899  0.8953028 -0.4985928
## [4,]  0.7037074 -1.2418817 -1.1721189  0.6527513
## [5,]  0.1972776 -0.3481495 -0.3285922  0.1829925
```

```
rank_approx(X2, r=2)
```

```
##             [,1]       [,2]       [,3]       [,4]
## [1,] -0.72581318  1.2900603  1.2200535 -0.6844519
## [2,] -0.00364291 -0.5795212 -0.7043568  0.7122268
## [3,]  0.03068905  0.8413133  1.0345826 -1.0651510
## [4,]  1.66425421 -1.4232324 -0.9366669 -0.3050136
## [5,] -0.96548718 -0.1286202 -0.6136125  1.3423896
```

```
# (f) standardized

cov_X2 <- t(X2) %*% X2 / 5

# eigenvectors same as columns of V2
# eigenvalues are diag(D2^2/5)   merely the same as eigen(cov_X2), only sign difference

V2
```

```
##            [,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
```

```
diag(D2^2/5)
```

```
## [1] 2.31524873 1.14353454 0.48138748 0.05982926
```

```
eigen(cov_X2)
```

```
## eigen() decomposition
## $values
## [1] 2.31524873 1.14353454 0.48138748 0.05982926
##
## $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
```

```
rank_approx(cov_X2, r=1)
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  0.2987738 -0.5272669 -0.4976477  0.2771393
## [2,] -0.5272669  0.9305048  0.8782337 -0.4890870
## [3,] -0.4976477  0.8782337  0.8288988 -0.4616125
## [4,]  0.2771393 -0.4890870 -0.4616125  0.2570713
```

```
rank_approx(cov_X2, r=2)
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  0.8459335 -0.6305704 -0.3635263 -0.2684358
## [2,] -0.6305704  0.9500084  0.8529116 -0.3860827
## [3,] -0.3635263  0.8529116  0.8617751 -0.5953455
## [4,] -0.2684358 -0.3860827 -0.5953455  0.8010663
```

In the following Questions 2 and 3, the Harr wavelet basis and natural cubic spline basis is defined on $(0, 1]$. So we transform the variable x in the given data to lie within $(0, 1]$, do the prediction and then transform it back to get all the plots.

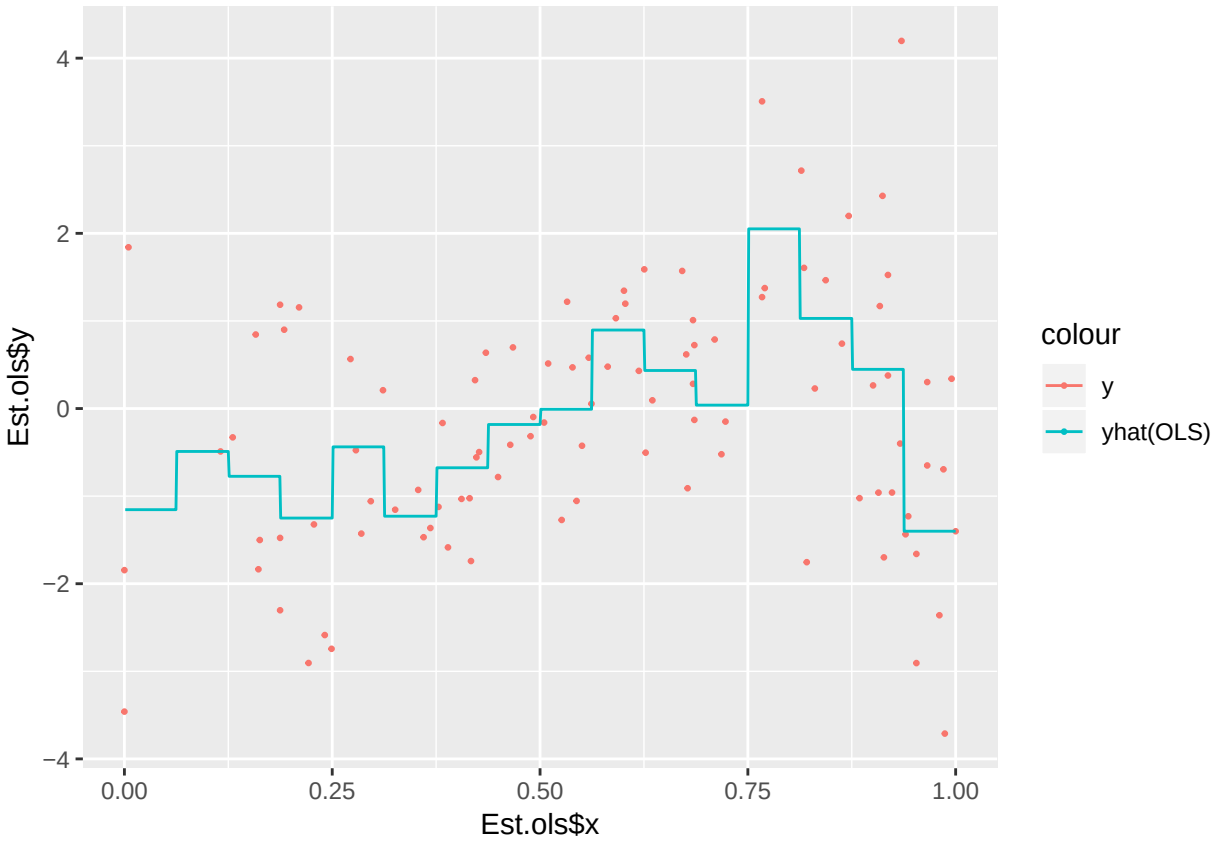
Problem 2

```
Q2data <- read_xlsx('/Users/apple/Desktop/ISU 2019 spring/STAT 602/hw/hw1/Problem3.4Data.xlsx',col_names

# Make Harr basis functions

# Father wavelet
Harr.F <- function(x){
  y <- (x > 0) & (x <= 1)
  return(as.numeric(y))
}

# Mother wavelet
Harr.M <- function(x){
  y = Harr.F(2*x) - Harr.F(2*x - 1)
  return(y)
```

```
}

# other wavelets
Harr.basis <- function(M, x){  # x: vector of locations to evaluate basis functions
  B <- cbind(Harr.F(x),Harr.M(x))
  for (m in 1:M){
    for (j in 0:(2^m-1)){
      psi_mj <- sqrt(2^m) * Harr.M(2^m * (x - j/(2^m)))
      B <- cbind(B, psi_mj)
    }
  }
  colnames(B) <- NULL
  return(B)
}


# Normalize x to [0,1]
x.01 <- (Q2data$x - min(Q2data$x) + 0.0001)/(max(Q2data$x) - min(Q2data$x) + 0.0001)

# compute Xh
Xh <- Harr.basis(M = 3, x = x.01)
```

(a)

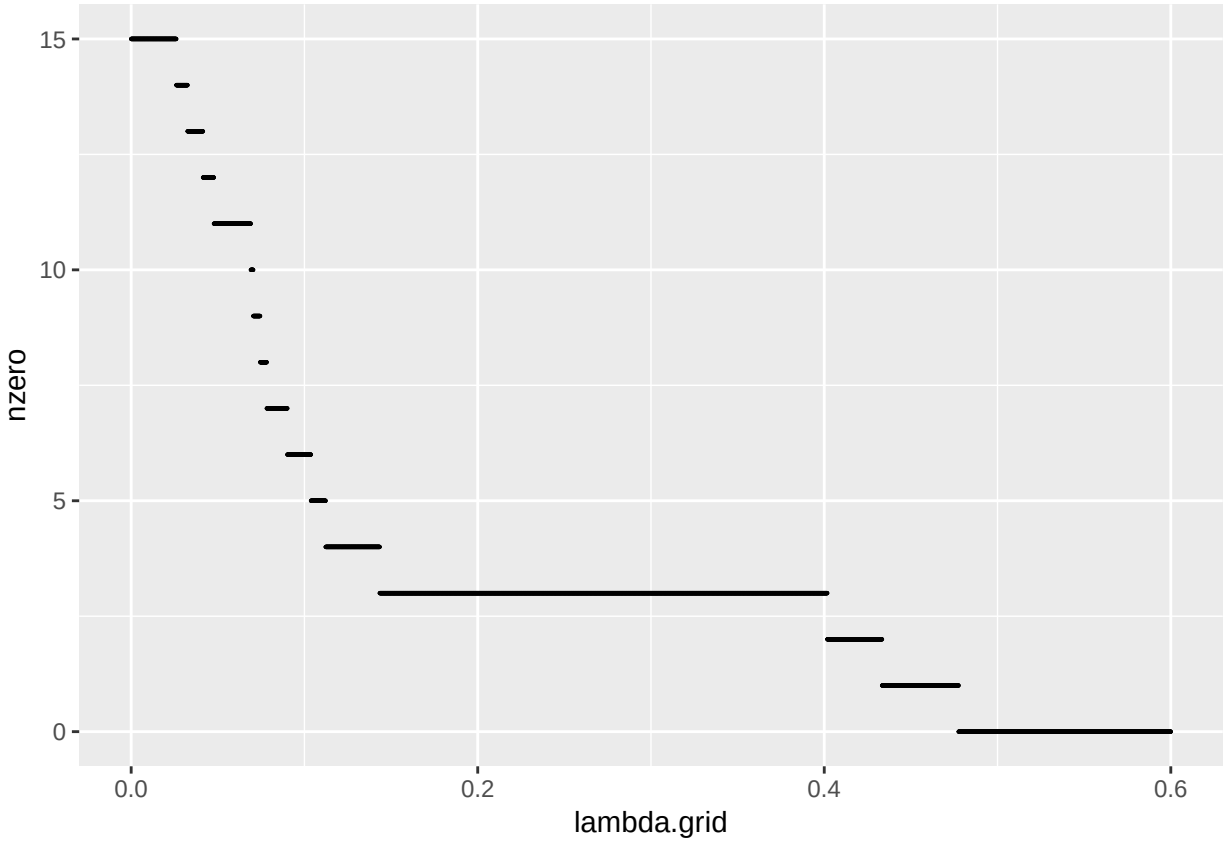
```
# compute beta_ols
Y <- Q2data$y
beta.ols <- solve(t(Xh)%*%Xh, t(Xh)%*%Y)
# compute yhat_ols
Yhat.ols <- Xh %*% beta.ols

# plot
x.grid <- seq(0.001,1,0.001)
Xh.full <- Harr.basis(M = 3, x = x.grid)
func.ols <- Xh.full %*% beta.ols
Est.ols <- data.frame(x=x.01, y = Y, yhat.ols = Yhat.ols)
ggplot() + geom_point(aes(x = Est.ols$x, y = Est.ols$y, color = 'y'),size = 0.5) +
  geom_line(aes(x = x.grid, y = func.ols, color = 'yhat(OLS)'))
```



(b)

```
# Center y and standardize the columns of Xh
Y.center <- Y - mean(Y)

std_X <- function(X){
  centers <- apply(X, 2, mean)
  scales <- apply(X, 2, FUN = function(x){
    x.center <- x - mean(x)
    return(sqrt(sum(x.center^2)/length(x)))
  })
  X.std <- sapply(1:ncol(X), FUN = function(i){
    x <- X[,i]
    x.std <- (x - (centers[i]))/scales[i]
    return(x.std)
  })
  return(list(X.std = X.std, centers = centers, scales = scales))
}

Xh.stdout <- std_X(Xh[,-1])     # no intercept
Xh.std <- Xh.stdout$X.std



# cross-validation to find lambda (that satisfies nzero M = 2, 4, 8)
```

```
lambda.grid <- seq(0.0001,1,0.0001)
cv.lasso <- cv.glmnet(x = Xh.std, y = Y.center, alpha = 1, lambda = seq(0,0.6,0.0001))
lambda.grid <- cv.lasso$lambda
nzero <- cv.lasso$nzero
ggplot() + geom_point(aes(x = lambda.grid, y = nzero),size = 0.1)
```




```
Lam.n2 <- min(lambda.grid[nzero==1])    # smallest lambda with 2 nonzero beta entries
Lam.n2
```

```
## [1] 0.4334
```

```
Lam.n4 <- min(lambda.grid[nzero==3])    # smallest lambda with 4 nonzero beta entries
Lam.n4
```

```
## [1] 0.1435
```

```
Lam.n8 <- min(lambda.grid[nzero==7])    # smallest lambda with 8 nonzero beta entries
Lam.n8
```

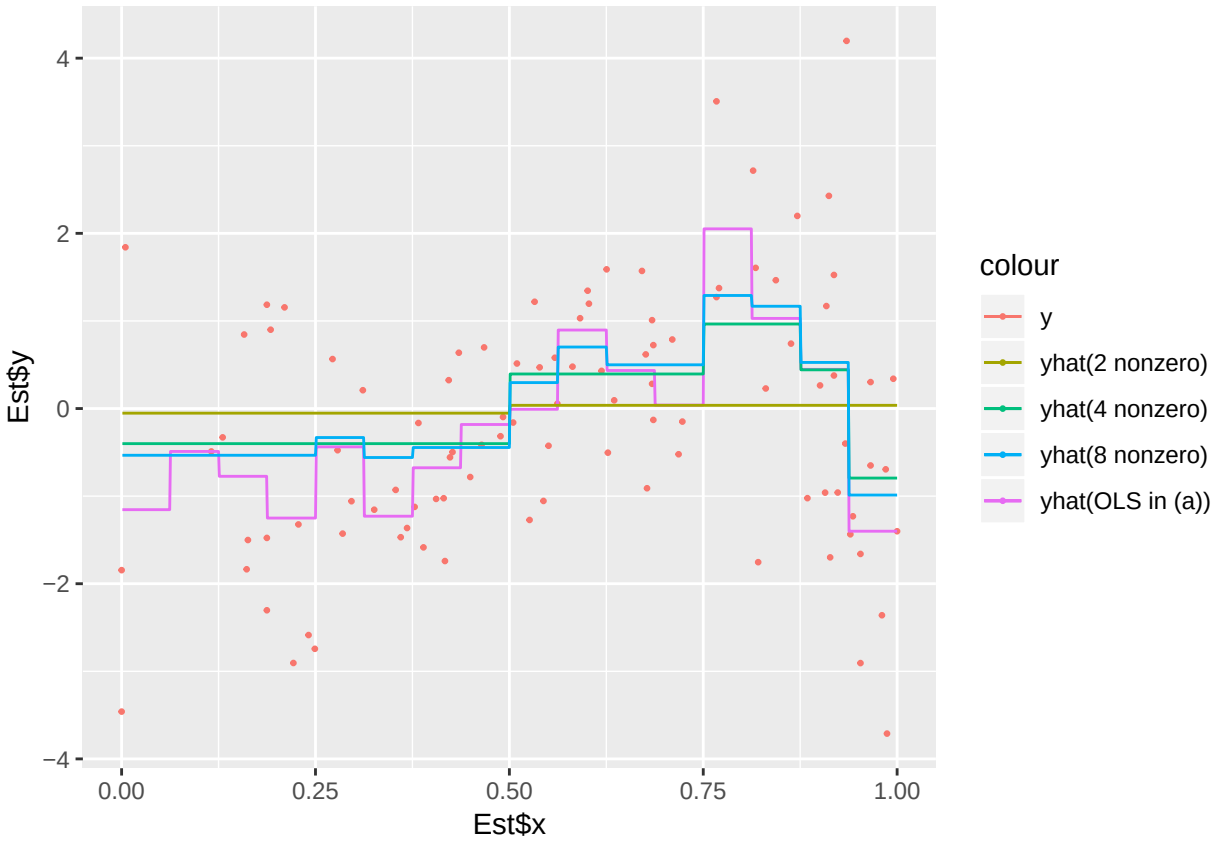
```
## [1] 0.0783
```

```
# lasso fit for certain lambdas
lasso.fit <- glmnet(x = Xh.std, y = Y.center, alpha = 1, lambda = c(Lam.n2, Lam.n4, Lam.n8))

# estimated parameters beta_lasso
param <- rbind(lasso.fit$a0,lasso.fit$beta)
colnames(param) <- c('2 nonzero entries', '4 nonzero entries', '8 nonzero entries')
param
```

```
## 16 x 3 sparse Matrix of class "dgCMatrix"
```

```
##     2 nonzero entries 4 nonzero entries 8 nonzero entries
##          -6.93383e-17     -6.900215e-17     -6.978930e-17
## V1       -4.41128e-02     -3.910728e-01     -4.861355e-01
## V2        .                .                -2.814342e-02
## V3        .                .                 .
## V4        .                .                 .
## V5        .                .                 .
## V6        .                .                 .
## V7        .                3.153276e-01      4.035633e-01
## V8        .                .                 .
## V9        .                .                 .
## V10       .                .                 3.416205e-02
## V11       .                .                 .
## V12       .                .                -7.552993e-02
## V13       .                .                 .
## V14       .                .                 1.932950e-02
## V15       .                2.898518e-01      3.550518e-01
```

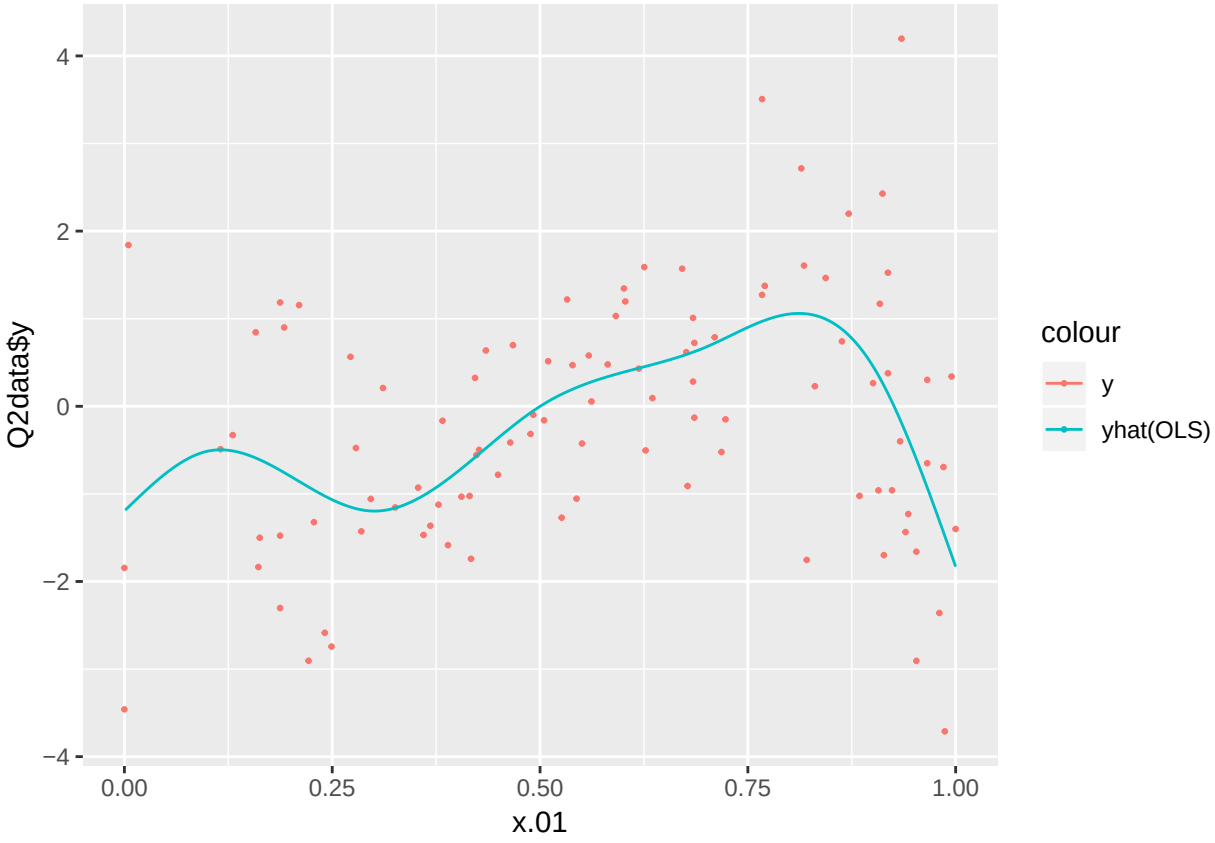
```r
# yhat_lasso
lasso.pred <- predict(lasso.fit , s = c(Lam.n2, Lam.n4, Lam.n8), newx = Xh.std)


# plot
Est <- data.frame(cbind(Est.ols, lasso.pred))
colnames(Est) <- c('x','y','yhat.ols','yhat.nzero2','yhat.nzero4','yhat.nzero8')

x.grid <- seq(0.001,1,0.001)
B.x <- Harr.basis(M=3, x.grid)[,-1]
B.x.trans <- sapply(1:ncol(B.x), FUN = function(i){ # center B.x using centers and scales from Xh
  x.tmp <- B.x[,i]
  scale.tmp <- Xh.stdout$scales[i]
  center.tmp <- Xh.stdout$centers[i]
  return((x.tmp - center.tmp)/scale.tmp)
})

func.lasso <- predict(lasso.fit , s = c(Lam.n2, Lam.n4, Lam.n8), newx = B.x.trans)

ggplot() + geom_point(aes(x = Est$x, y = Est$y, color = 'y'), size = 0.5) +
  geom_line(aes(x = x.grid, y = func.ols, color = 'yhat(OLS in (a))'), size = 0.5) +
  geom_line(aes(x = x.grid, y = func.lasso[,1], color = 'yhat(2 nonzero)'), size = 0.5) +
  geom_line(aes(x = x.grid, y = func.lasso[,2], color = 'yhat(4 nonzero)'), size = 0.5) +
  geom_line(aes(x = x.grid, y = func.lasso[,3], color = 'yhat(8 nonzero)'), size = 0.5)
```



Problem 3

```
pos.func <- function(a,b){   # vector a, scalar b;  return (a-b)_{+}
  c <- a - b
  return(c * (c > 0))
}

# function to provide natural cubic regression spline basis
N.Cubic <- function(x, knots){
  K <- length(knots)
  B <- cbind(rep(1, length(x)),x)
  for (j in 1:(K-2)){
    c1 <- (knots[K] - knots[j]) / (knots[K] - knots[K-1])
    c2 <- (knots[K-1] - knots[j]) / (knots[K] - knots[K-1])
    B.tmp <- pos.func(x, knots[j])^3 - c1 * pos.func(x, knots[K-1])^3 + c2 * pos.func(x, knots[K])^3
    B <- cbind(B, B.tmp)
  }
  colnames(B) <- NULL
  return(B)
}

# compute Xh
knots = c(0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0)
Xh.cubic <- N.Cubic(x.01, knots = knots)
```

```r
# compute beta_spline and yhat_spline
beta.cubic.ols <- solve(t(Xh.cubic)%*%Xh.cubic, t(Xh.cubic)%*%Y)
Yhat.cubic.ols <- Xh.cubic %*% beta.cubic.ols

# plot
x.grid <- seq(0.001,1,0.001)
Xh.cubic.full <- N.Cubic(x = x.grid, knots = knots)
func.cubic.ols <- Xh.cubic.full %*% beta.cubic.ols

ggplot() + geom_point(aes(x = x.01, y = Q2data$y, color = 'y'),size = 0.5) +
  geom_line(aes(x = x.grid, y = func.cubic.ols, color = 'yhat(OLS)'))
```



Problem 4

(a)

From notes, we know that:

$$\underset{N\times N}{\mathbf{H}} = \left(h_j\left(x_i\right)\right); \quad \underset{N\times N}{\Omega} = \left(\int_0^1 h_j''(t)h_l''(t)dt\right); \quad \mathbf{K} = \left(\mathbf{H}'\right)^{-1}\mathbf{\Omega}\mathbf{H}^{-1}$$

$$\because h''_{j+2}(x) = 6\,(x - x_j)\,I\,[x_j \le x \le x_{N-1}] + 6\,(x - x_N)\left(\frac{x_j - x_N}{x_N - x_{N-1}}\right) I\,[x_{N-1} \le x \le x_N]$$

$$\therefore \int_0^1 \left(h''_{j+2}(x)\right)^2 dx = 12\left((x_{N-1} - x_j)^3 + (x_N - x_{N-1})\,(x_N - x_j)^2\right)$$

$$\therefore \int_0^1 h''_{j+2}(x) h''_{k+2}(x) dx = 12\left(x_{N-1}^2 - x_k^3\right) - 18\,(x_j + x_k)\left(x_{N-1}^2 - x_k^2\right) + 36 x_j x_k\,(x_{N-1} - x_k) + 12\,(x_N - x_{N-1})\,(x_{N-1} - x_j)\,(x_{N-1} - x_k)$$

Therefore, we can code the matrix in R as follow:

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

H_mat <- function(x){
  n <- length(x)
  H_mat <- matrix(0, ncol = n, nrow = n)
  H_mat[,1] <- 1
  H_mat[,2] <- x
  z_ind <- rep(0,n)
  for (j in 1:(n-2)){
    z1 <- pmax(z_ind, x-x[j])
    z2 <- pmax(z_ind, x-x[n-1])
    z3 <- pmax(z_ind, x-x[n])
    H_mat[,2+j] <- z1^3 - ((x[n]-x[j])/(x[n]-x[n-1])) * z2^3 +
      ((x[n-1]-x[j])/(x[n]-x[n-1])) * z3^3
  }
  return(H_mat)
}

H <- H_mat(x)
H
```

```
##       [,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
```

```
Omega_mat <- function(x){
  n <- length(x)
  Omega_mat <- matrix(0, ncol = n, nrow = n)
  Omega_mat22 <- matrix(0, ncol = n-2, nrow = n-2)
  for (j in 1:(n-3)){
    for (k in (j+1):(n-2)){
      Omega_mat22[j,k] <- 12*(x[n-1]^3 - x[k]^3) - 18*(x[j]+x[k])*(x[n-1]^2 - x[k]^2) +
        36*x[j]*x[k]*(x[n-1]-x[k]) + 12*(x[n]-x[n-1])*(x[j]-x[n-1])*(x[k]-x[n-1])
    }
```

```
  }
  Omega_mat22_diag <-  12 * ( (x[n-1]-x[1:(n-2)])^3 + (x[n]-x[n-1])*(x[n]-x[1:(n-2)])^2 )
  Omega_mat22 <- Omega_mat22 + t(Omega_mat22)
  Omega_mat[3:n,3:n] <- Omega_mat22
  diag(Omega_mat)[3:n] <- Omega_mat22_diag
  return(Omega_mat)
}

Omega <- Omega_mat(x)
Omega
```

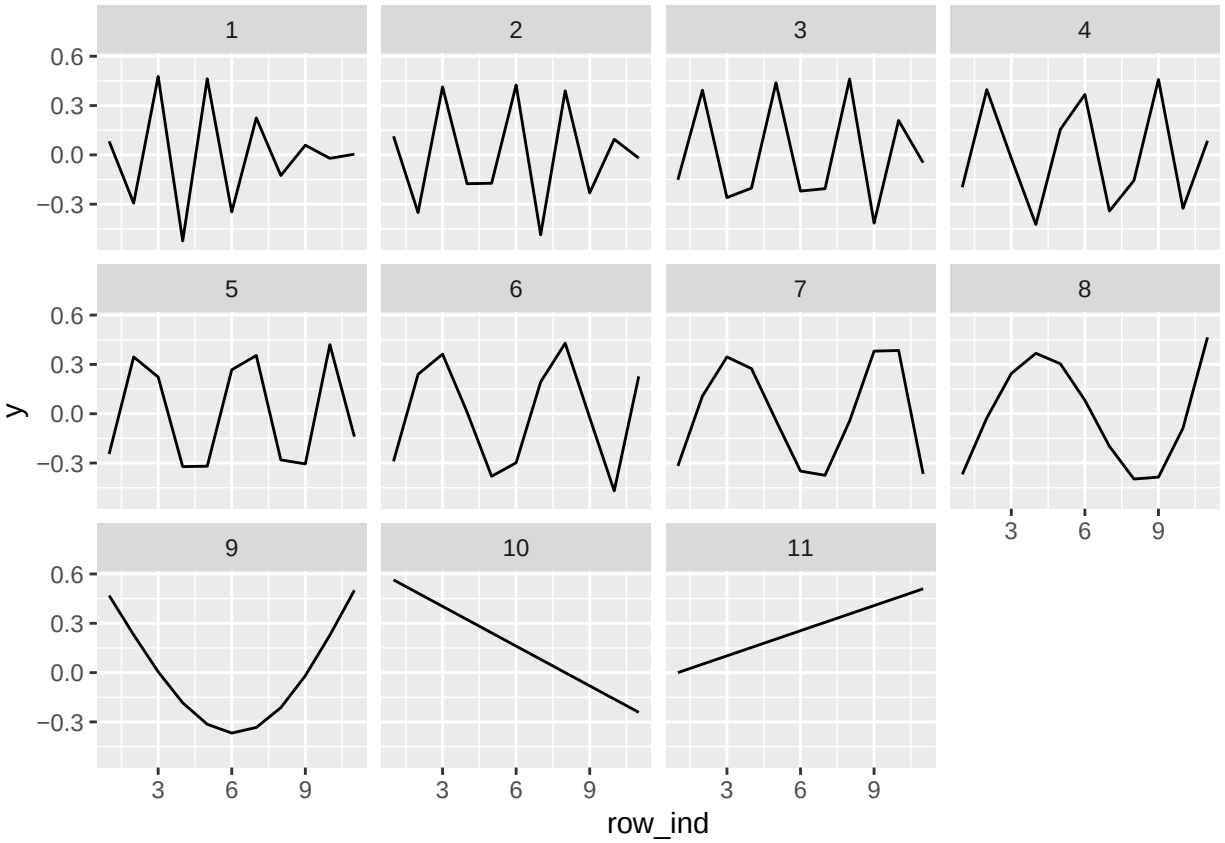
```
##       [,1] [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11]
##  [1,]    0    0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##  [2,]    0    0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##  [3,]    0    0 9.948 8.160 6.636 5.184 3.840 2.640 1.620 0.816 0.264
##  [4,]    0    0 8.160 7.116 5.670 4.464 3.330 2.304 1.422 0.720 0.234
##  [5,]    0    0 6.636 5.670 4.884 3.744 2.820 1.968 1.224 0.624 0.204
##  [6,]    0    0 5.184 4.464 3.744 3.180 2.310 1.632 1.026 0.528 0.174
##  [7,]    0    0 3.840 3.330 2.820 2.310 1.932 1.296 0.828 0.432 0.144
##  [8,]    0    0 2.640 2.304 1.968 1.632 1.296 1.068 0.630 0.336 0.114
##  [9,]    0    0 1.620 1.422 1.224 1.026 0.828 0.630 0.516 0.240 0.084
## [10,]    0    0 0.816 0.720 0.624 0.528 0.432 0.336 0.240 0.204 0.054
## [11,]    0    0 0.264 0.234 0.204 0.174 0.144 0.114 0.084 0.054 0.060
```

```
K <- solve(t(H))%*%Omega%*%solve(H)
round(K,1)
```

```
##           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
##  [1,]  128928.7  -380999.9   441797.1  -281864.4   126263.5   -45360.3
##  [2,] -380999.9  1178564.2 -1479041.3  1063657.1  -541204.2   214184.0
##  [3,]  441797.1 -1479041.3  2119302.8 -1842394.9  1148744.6  -542680.6
##  [4,] -281864.4  1063657.1 -1842394.9  2052766.4 -1658924.3   998295.3
##  [5,]  126263.5  -541204.2  1148744.6 -1658924.3  1765499.5 -1392434.2
##  [6,]  -45360.3   214184.0  -542680.6   998295.3 -1392434.2  1449518.8
##  [7,]   14688.8   -72598.0   206243.2  -460252.4   819954.4 -1116514.1
##  [8,]   -4464.6    22626.3   -67965.4   170988.2  -368805.4   637952.0
##  [9,]    1291.4    -6638.7    20544.3   -54672.7   132231.5  -274178.0
## [10,]    -337.5     1747.5    -5487.2    14985.0   -37974.1    86892.6
## [11,]      57.2     -297.0      937.4    -2583.2     6648.8   -15675.5
##             [,7]       [,8]       [,9]      [,10]     [,11]
##  [1,]   14688.8   -4464.6    1291.4    -337.5     57.2
##  [2,]  -72598.0   22626.3   -6638.7    1747.5   -297.0
##  [3,]  206243.2  -67965.4   20544.3   -5487.2    937.4
##  [4,] -460252.4  170988.2  -54672.7   14985.0  -2583.2
##  [5,]  819954.4 -368805.4  132231.5  -37974.1   6648.8
##  [6,] -1116514.1  637952.0 -274178.0   86892.6 -15675.5
##  [7,]  1129876.7 -837228.1  449607.3 -165877.9  32100.0
##  [8,]  -837228.1  802487.5 -542125.2  236587.4 -50052.8
##  [9,]   449607.3 -542125.2  449934.3 -229231.9  53237.7
## [10,]  -165877.9  236587.4 -229231.9  132245.0 -33548.9
## [11,]    32100.0  -50052.8   53237.7  -33548.9   9176.2
```

(b)

```
# eigen analysis of the matrix K
d_K <- eigen(K, symmetric = TRUE)$values
P_K <- eigen(K, symmetric = TRUE)$vectors

# plot the eigenvectors
rP_K <- as.numeric(P_K)
row_ind <- rep(1:11,11)
eig_vec_ind <- rep(1:11, each = 11)
eig_vec_ind <- as.factor(eig_vec_ind)
D4 <- data.frame(row_ind, y=rP_K, eig_vec_ind)

# all eigenvectors
 ggplot(data=D4) +
  geom_line(aes(x=row_ind, y = y)) +
   facet_wrap(~eig_vec_ind, nrow = 3)
```



Visually speaking, the "shape" of eigenvectors with larger eigenvalues are sharper. The components of a vector of observed values, Y get "most suppressed" in the spanned space of eigenvectors of $\boldsymbol{K}$ with larger eigenvalues. In fact, if the eigenvalue decomposition of $\boldsymbol{K}$ is $\boldsymbol{K} = U\Lambda U^T$, then,

$$\hat{Y} = U(I + \lambda\Lambda)^{-1}U^T Y = \sum_{i=1}^{N} \frac{< u_i, Y >}{1 + \lambda\Lambda_{ii}} u_i$$

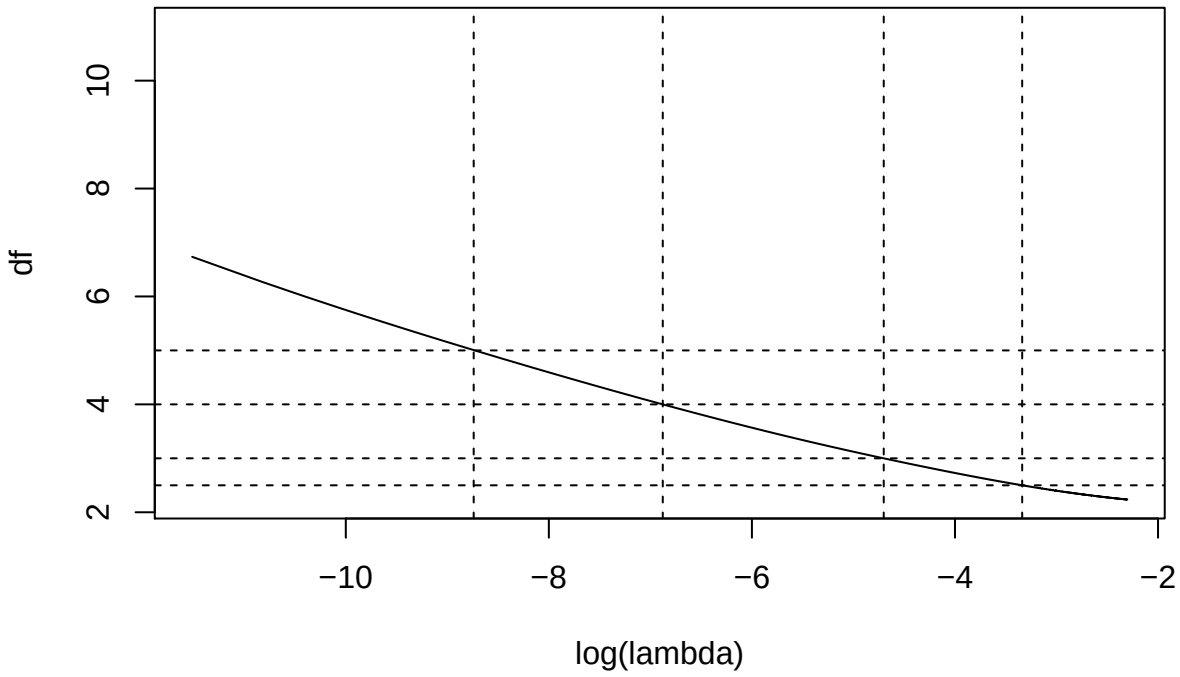
It's not hard to find that $\hat{Y}$ is shrunk more rapidly for larger Λ_{ii}.

(c)

$$\mathbf{S}_\lambda = \mathbf{H}\left(\mathbf{H}'\mathbf{H} + \lambda\boldsymbol{\Omega}\right)^{-1}\mathbf{H}' = \left(\mathbf{I} + \lambda\mathbf{K}\right)^{-1}$$

$$\mathrm{df}(\lambda) \equiv \mathrm{tr}\left(\mathbf{S}_\lambda\right)$$

```r
# function to compute S_lambda
S_lam <- function(lambda, H, Omega){
  S <- H %*% solve(t(H)%*%H + lambda*Omega) %*% t(H)
  return( S )
}

# function to compute trace(S_lambda)
trS_lambda <- function(lambda, H, Omega){
  S <- H %*% solve(t(H)%*%H + lambda*Omega) %*% t(H)
  return( sum( diag(S) ) )
}

# search lambda for certain df
lambda0 <- seq(0,0.1,0.00001)
df0 <- rep(0,length(lambda0))
for (i in 1:length(lambda0)){
  df0[i] <- trS_lambda(lambda0[i],H,Omega)
}

plot(log(lambda0), df0, type = "l", xlab = "log(lambda)", ylab = "df")
abline(h=2.5, lty=2)
abline(h=3, lty=2)
abline(h=4, lty=2)
abline(h=5, lty=2)
abline(v=log( lambda0[ which.min(abs(df0-2.5)) ] ), lty=2)
abline(v=log( lambda0[ which.min(abs(df0-3)) ] ), lty=2)
abline(v=log( lambda0[ which.min(abs(df0-4)) ] ), lty=2)
abline(v=log( lambda0[ which.min(abs(df0-5)) ] ), lty=2)
```




```
# df=2.5
lambda0[ which.min(abs(df0-2.5)) ]
```

```
## [1] 0.03551
```

```
# df=3
lambda0[ which.min(abs(df0-3)) ]
```

```
## [1] 0.00908
```

```
# df=4
lambda0[ which.min(abs(df0-4)) ]
```

```
## [1] 0.00103
```

```
# df=5
lambda0[ which.min(abs(df0-5)) ]
```

```
## [1] 0.00016
```

Problem 5

(a)

For kernel smoother,

$$\underset{N\times 2}{\mathbf{B}} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix}; \quad \underset{N\times N}{W}(x_0) = \operatorname{diag}\left(K_\lambda(x_0, x_1), \ldots, K_\lambda(x_0, x_N)\right);$$

$$\mathrm{l}'(x_0) = (1, x_0)\left(\mathbf{B}'\mathbf{W}(x_0)\mathbf{B}\right)^{-1}\mathbf{B}'\mathbf{W}(x_0); \quad \underset{N\times N}{\mathbf{L}_\lambda} = \begin{pmatrix} \mathrm{l}'(x_1) \\ \vdots \\ \mathrm{l}'(x_N) \end{pmatrix};$$

$$\widehat{\mathbf{Y}}_\lambda = \mathbf{L}_\lambda \mathbf{Y}; \quad \mathrm{df}(\lambda) = \mathrm{tr}\left(\mathbf{L}_\lambda\right).$$

Then for gaussian kernel, we wirte R to find the bandwidth for certain level of effective degrees of freedom.

```
x <- seq(0,1,0.1)

# function to compute l_lambda
l_lam <- function(x0, x, lambda){
  n <- length(x)
  B <- cbind(rep(1,n),x)
  W_vec <- dnorm(x, mean = x0, sd = lambda)
  W <- diag(W_vec)
  l <- t(c(1,x0)) %*% solve( t(B) %*% W %*% B ) %*% t(B) %*% W
  return( as.numeric(l) )
}

# function to compute L_lambda
L_lam <- function(x, lambda){
  n <- length(x)
  L <- matrix(0, ncol=n, nrow=n)
  for (i in 1:n){
    L[i,] <- l_lam(x[i], x, lambda)
  }
  return(L)
}

# function to compute trace(L_lambda)
tr_L_lam <- function(x, lambda){
  L <- L_lam(x, lambda)
  return( sum( diag(L) ) )
}


# search for certain df
log_lambda1 <- seq(-3,0,0.01)
df1 <- rep(0,length(log_lambda1))

for (i in 1:length(log_lambda1)){
  df1[i] <- tr_L_lam( x, exp(log_lambda1[i]) )
}


# df=2.5
exp( log_lambda1[ which.min(abs(df1-2.5)) ] )
```

```
## [1] 0.3906278
```

```
# df=3
exp( log_lambda1[ which.min(abs(df1-3)) ] )
```

```
## [1] 0.2644773
```

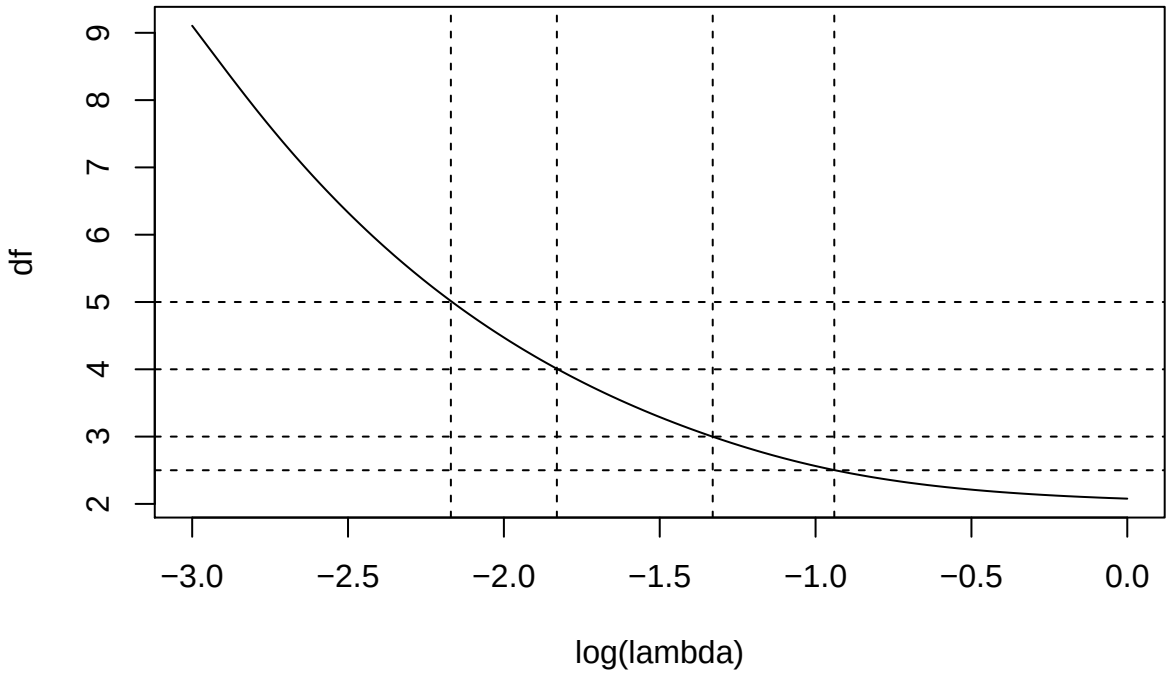
```
# df=4
exp( log_lambda1[ which.min(abs(df1-4)) ] )
```

```
## [1] 0.1604136
```

```
# df=5
exp( log_lambda1[ which.min(abs(df1-5)) ] )
```

```
## [1] 0.1141776
```

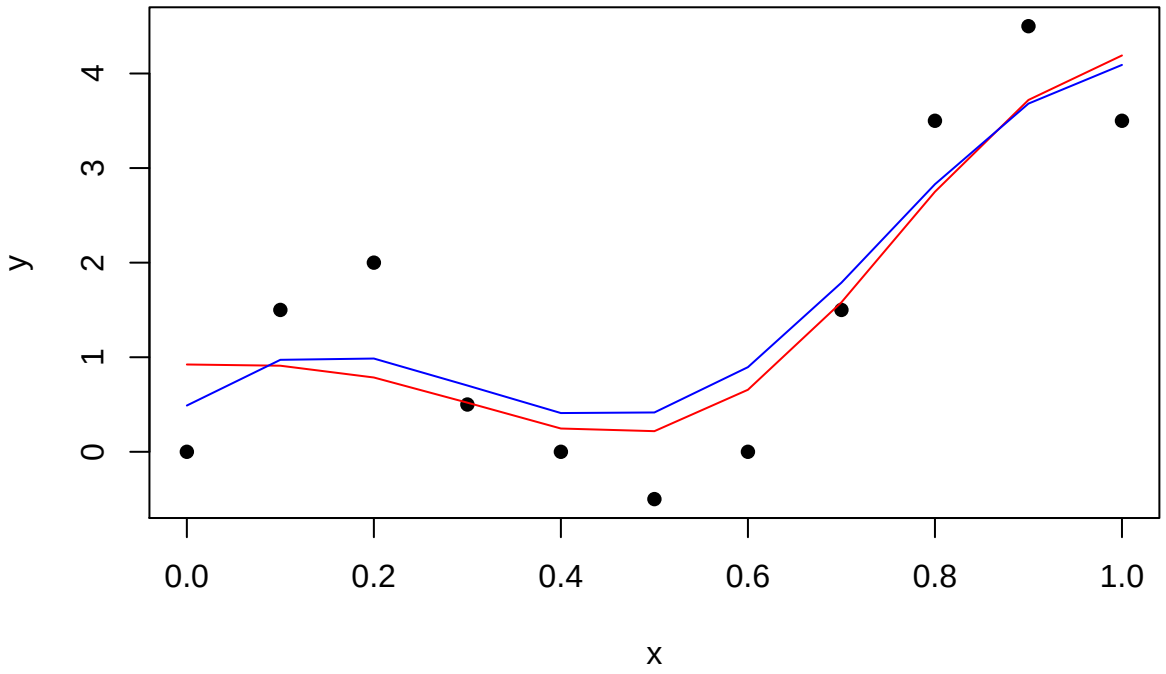
```
# plot
plot(log_lambda1, df1, type = "l", xlab = "log(lambda)", ylab = "df")

abline(h=2.5, lty=2)
abline(h=3, lty=2)
abline(h=4, lty=2)
abline(h=5, lty=2)
abline(v= log_lambda1[ which.min(abs(df1-2.5)) ] , lty=2)
abline(v= log_lambda1[ which.min(abs(df1-3)) ] , lty=2)
abline(v= log_lambda1[ which.min(abs(df1-4)) ] , lty=2)
abline(v= log_lambda1[ which.min(abs(df1-5)) ] , lty=2)
```



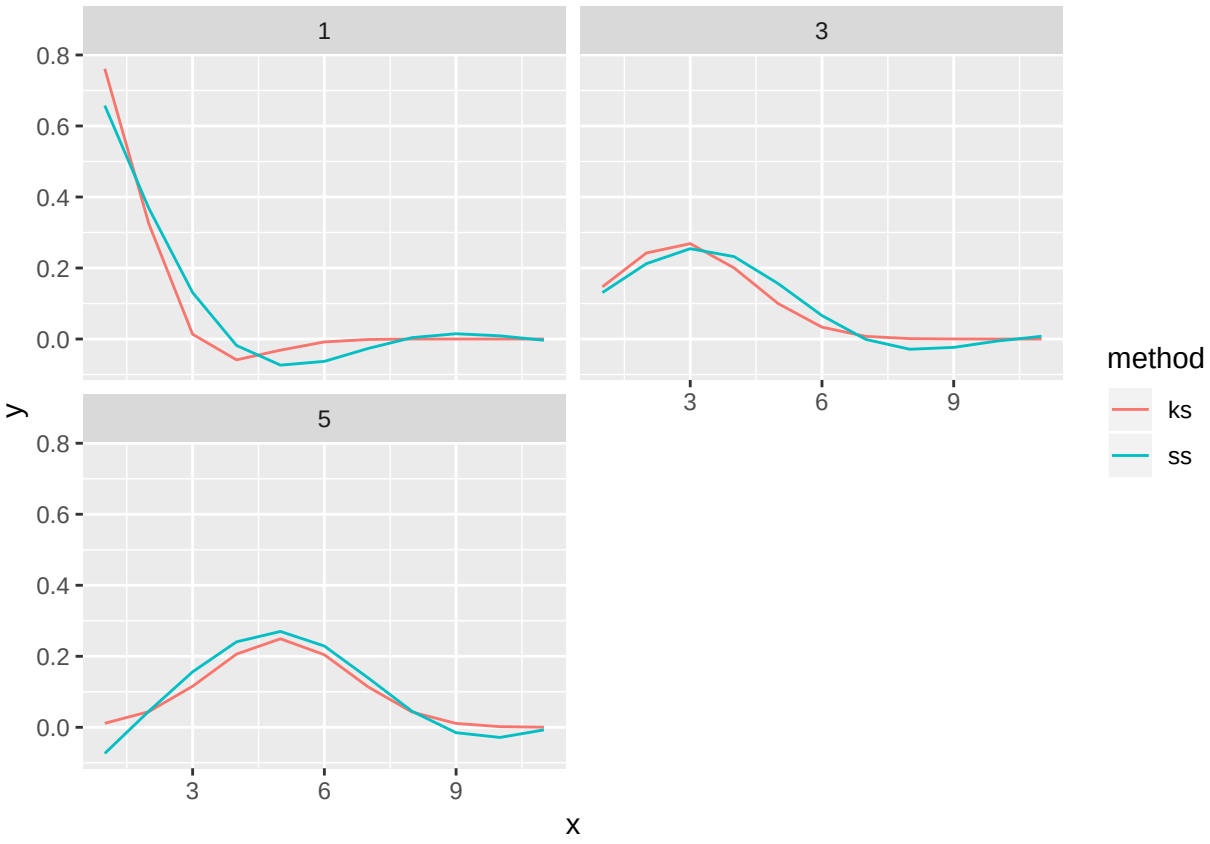
(b)

```
# find the smoothing penalty parameter and bandwidth corresponding to df = 4
l0 <- lambda0[ which.min(abs(df0-4)) ]
l1 <- exp( log_lambda1[ which.min(abs(df1-4)) ] )

# compute the spline/kernel smoothing matrix
S4 <- S_lam(l0,H,Omega)
L4 <- L_lam(x,l1)

# plot spline fit vs kernel fit ( spline:red \ kernel:blue )
plot(x,y, ylab = "y", pch = 16)
lines(x, S4%*%y, col="red")
```

```
lines(x, L4%*%y, col="blue")
```




```
# plot the 1st, 3rd, 5th row of the two smoothing matrices
D_p5 <- data.frame(method = rep( c("ss", "ks"), each = 11*3),
                   col_ind = rep(rep( c("1","3","5"), each=11 ),2),
                   y = c(S4[1,], S4[3,], S4[5,], L4[1,], L4[3,], L4[5,]),
                   x = rep(1:11,6))

ggplot(data=D_p5) + geom_line(aes(x=x,y=y,colour=method)) +
  facet_wrap(~col_ind, nrow=2)
```



Problem 6

We first search for the bandwidth with 5 and then 9 effective degrees of freedom for tricube kernel in R:

```r
# function to compute l_lambda (tricube)
l_lam_tricube <- function(x0, x, lambda){
  n <- length(x)
  B <- cbind(rep(1,n),x)

  tt <- abs(x-x0)/lambda
  W_vec <- rep(0,n)
  ind <- which(tt<1)
  W_vec[ind] <- (1 - tt[ind]^3)^3
  W <- diag(W_vec)
  l <- t(c(1,x0)) %*% solve( t(B) %*% W %*% B ) %*% t(B) %*% W
  return( as.numeric(l) )
}


# function to compute L_lambda (tricube)
L_lam_tricube <- function(x, lambda){
  n <- length(x)
  L <- matrix(0, ncol=n, nrow=n)
  for (i in 1:n){
    L[i,] <- l_lam_tricube(x[i], x, lambda)
  }
```

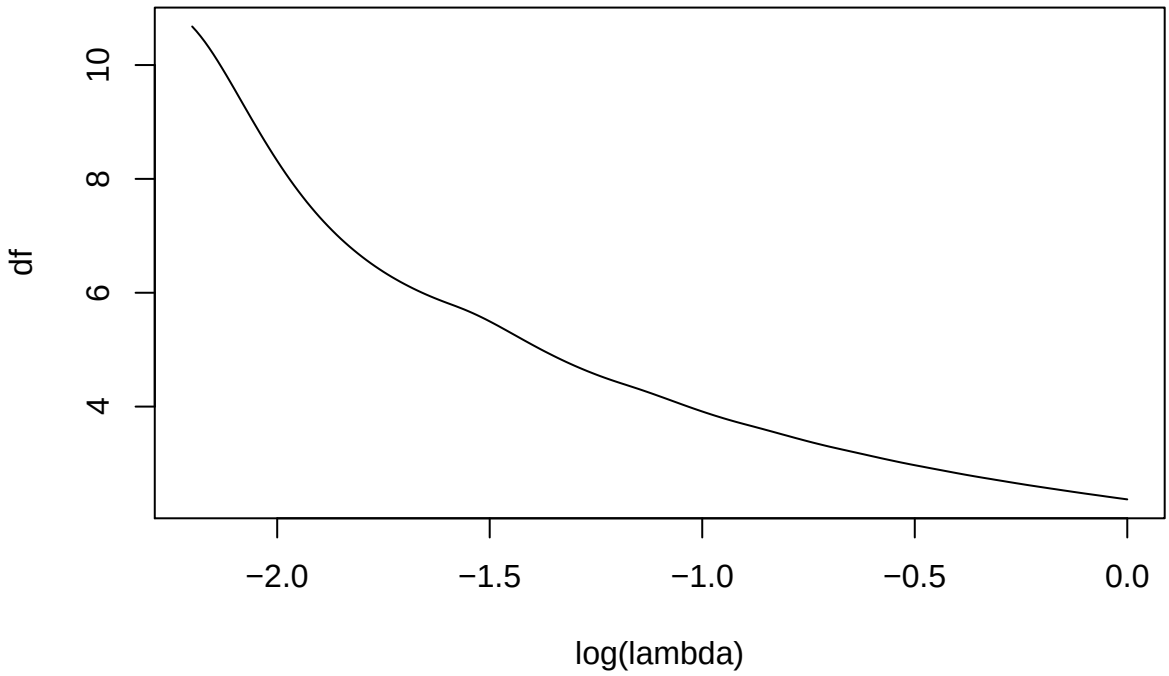
```
  return(L)
}


# function to compute trace(L_lambda) (tricube)
tr_L_lam_tricube <- function(x, lambda){
  L <- L_lam_tricube(x, lambda)
  return( sum( diag(L) ) )
}



# search for certain df (df=5 and df=9)
log_lambda2 <- seq(-2.2,0,0.01)
df2 <- rep(0,length(log_lambda2))

for (i in 1:length(log_lambda2)){
  df2[i] <- tr_L_lam_tricube( x, exp(log_lambda2[i]) )
}

# df=5
exp( log_lambda2[ which.min(abs(df2-5)) ] )
```

```
## [1] 0.2515786
```

```
# df=9
exp( log_lambda2[ which.min(abs(df2-9)) ] )
```

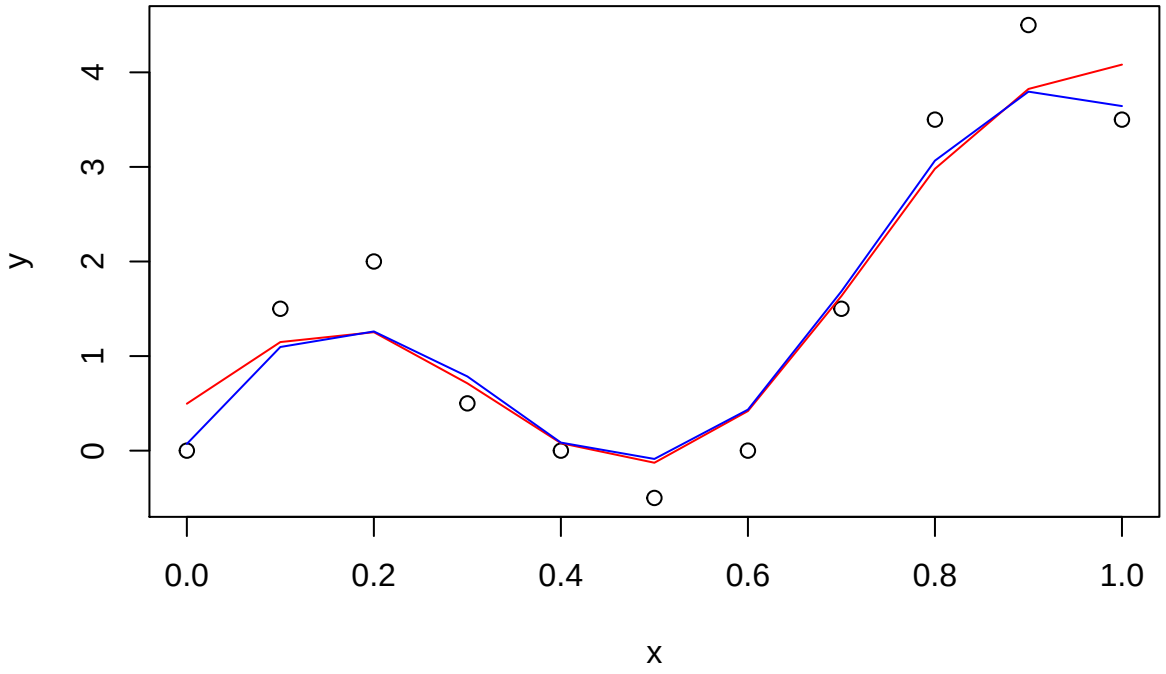
```
## [1] 0.127454
```

```
# plot
plot(log_lambda2, df2, type = "l", xlab = "log(lambda)", ylab = "df")
```

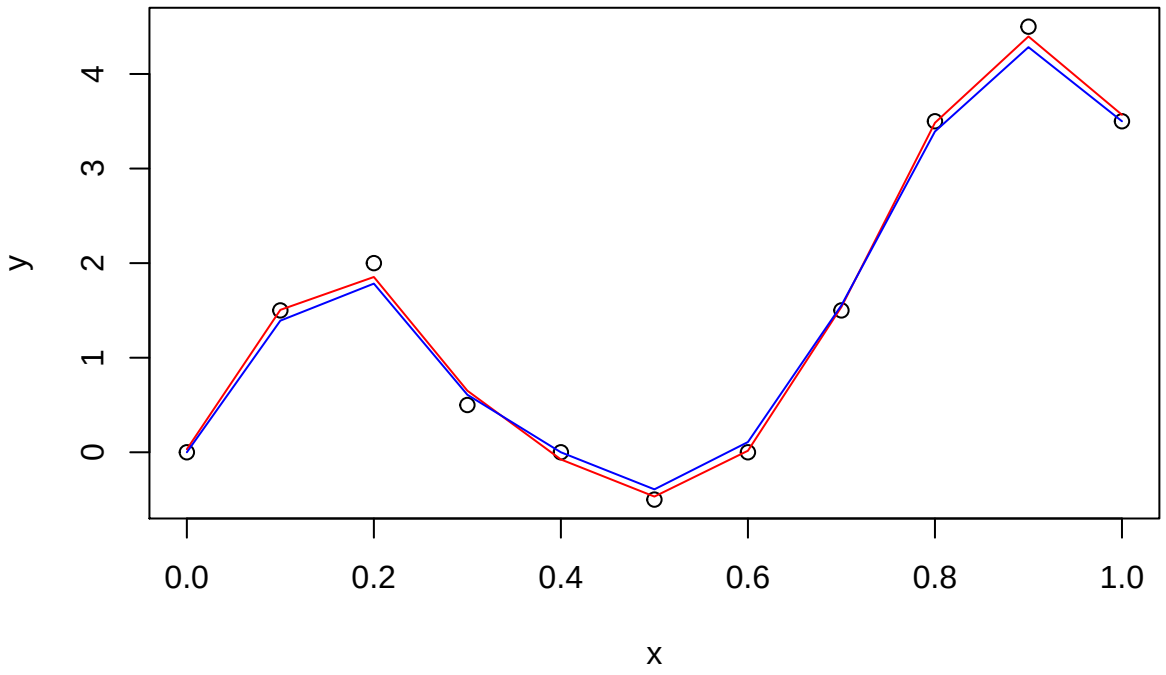


Then we fit the data by smoothing spline/ local polynomial:

```r
D6 <- data.frame(x,y)

# local polynomial fit
library(locpol)
fit_k5 <- locpol(y~x, data = D6, bw =  0.2515786, kernel = tricubK,
                 deg = 1)
fit_k9 <- locpol(y~x, data = D6, bw =   0.127454, kernel = tricubK,
                 deg = 1)


# smoothing spline fit
fit_s5 <- smooth.spline(x=x, y=y, df=5)
fit_s9 <- smooth.spline(x=x, y=y, df=9)


# compare for df=5 ( spline:red / kernel:blue )
plot(x,y)
lines(x, predict(fit_s5)$y, col = "red")
lines(x, y-fit_k5$residuals, col = "blue")
```

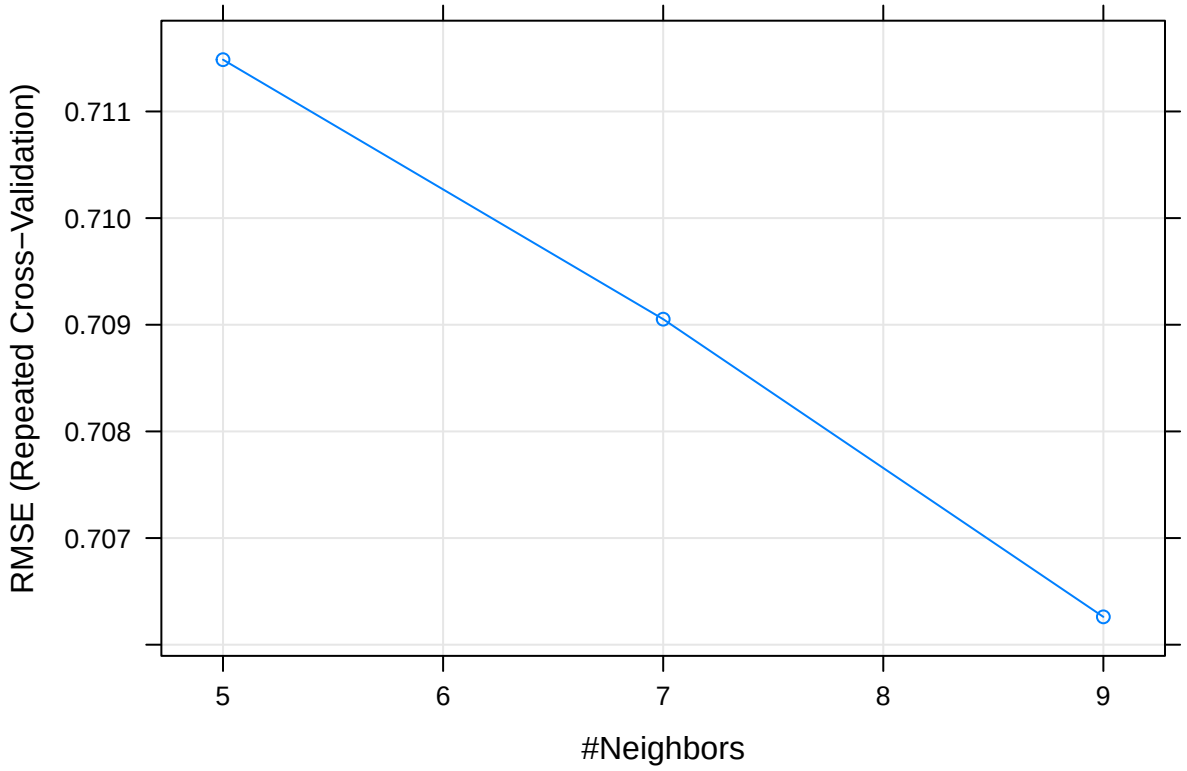



```r
# compare for df=9 ( spline:red / kernel:blue )
plot(x,y)
lines(x, predict(fit_s9)$y, col = "red")
lines(x, y-fit_k9$residuals, col = "blue")
```

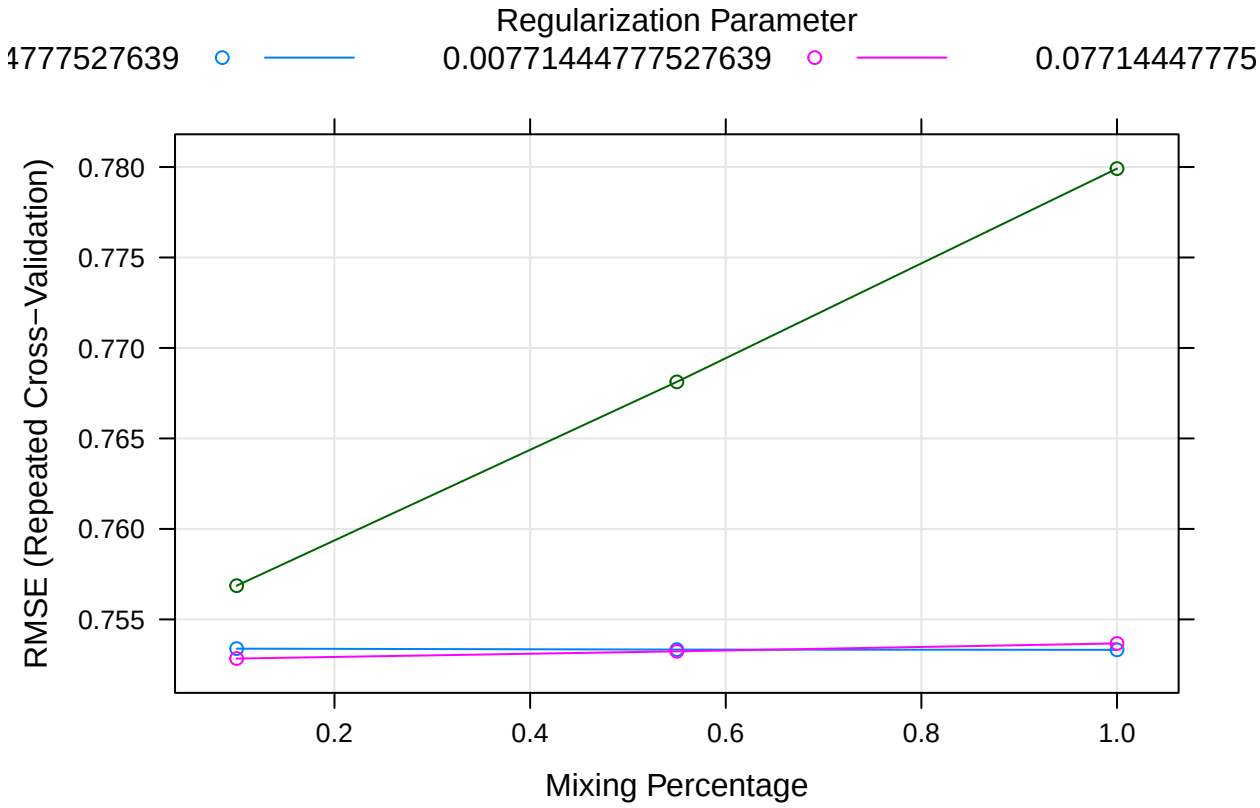


Problem 7

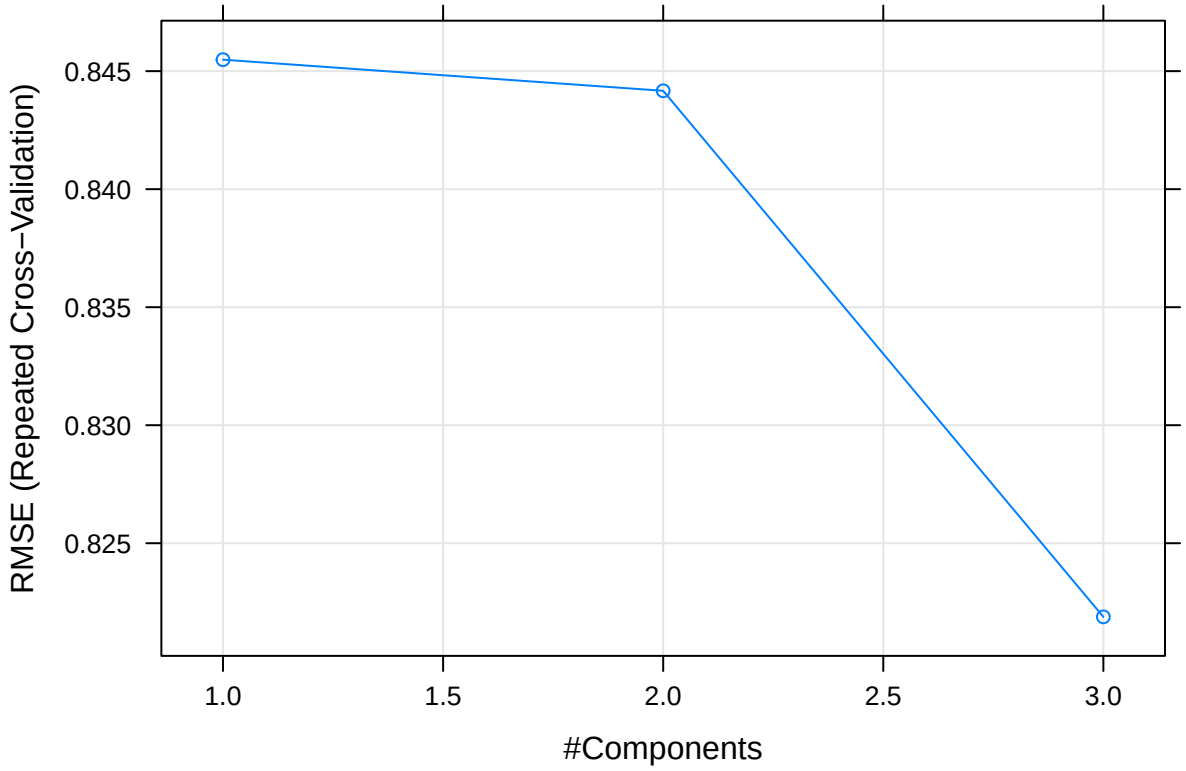
```
wine_data <- read.table('/Users/apple/Desktop/ISU 2019 spring/STAT 602/hw/hw2/winequality-white.csv', he

ols.fit <- lm(quality ~., data = wine_data)
# ols.fit$coefficients

knn.fit <- train(x = wine_data[,-12], y = wine_data$quality,
                 method = 'knn', preProcess = c('center', 'scale'),
                 trControl=trainControl(method="repeatedcv",repeats=10,number=10))
plot(knn.fit)
```



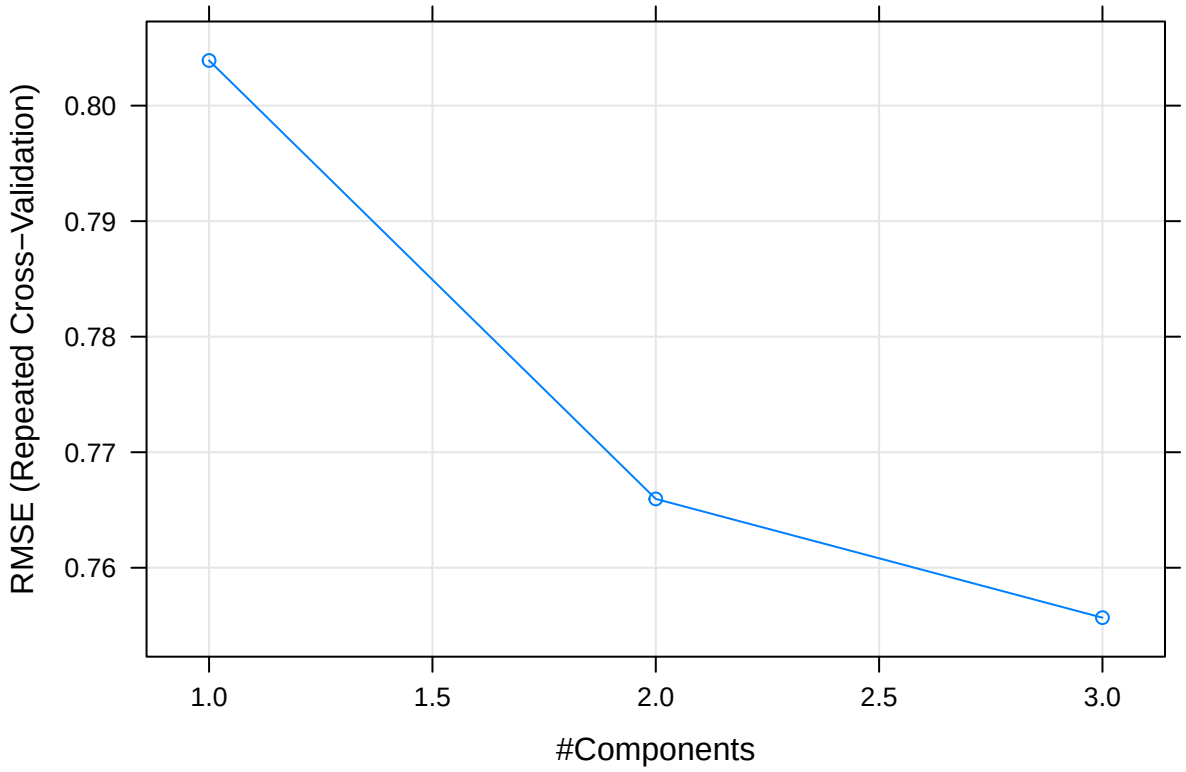

```
net.fit <- train(x = wine_data[,-12], y = wine_data$quality,
                 method = 'glmnet', preProcess = c('center', 'scale'),
                 trControl=trainControl(method="repeatedcv",repeats=10,number=10))
plot(net.fit)
```




```
pcr.fit <- train(x = wine_data[,-12], y = wine_data$quality,
                 method = 'pcr', preProcess = c('center', 'scale'),
                 trControl=trainControl(method="repeatedcv",repeats=10,number=10))
plot(pcr.fit)
```



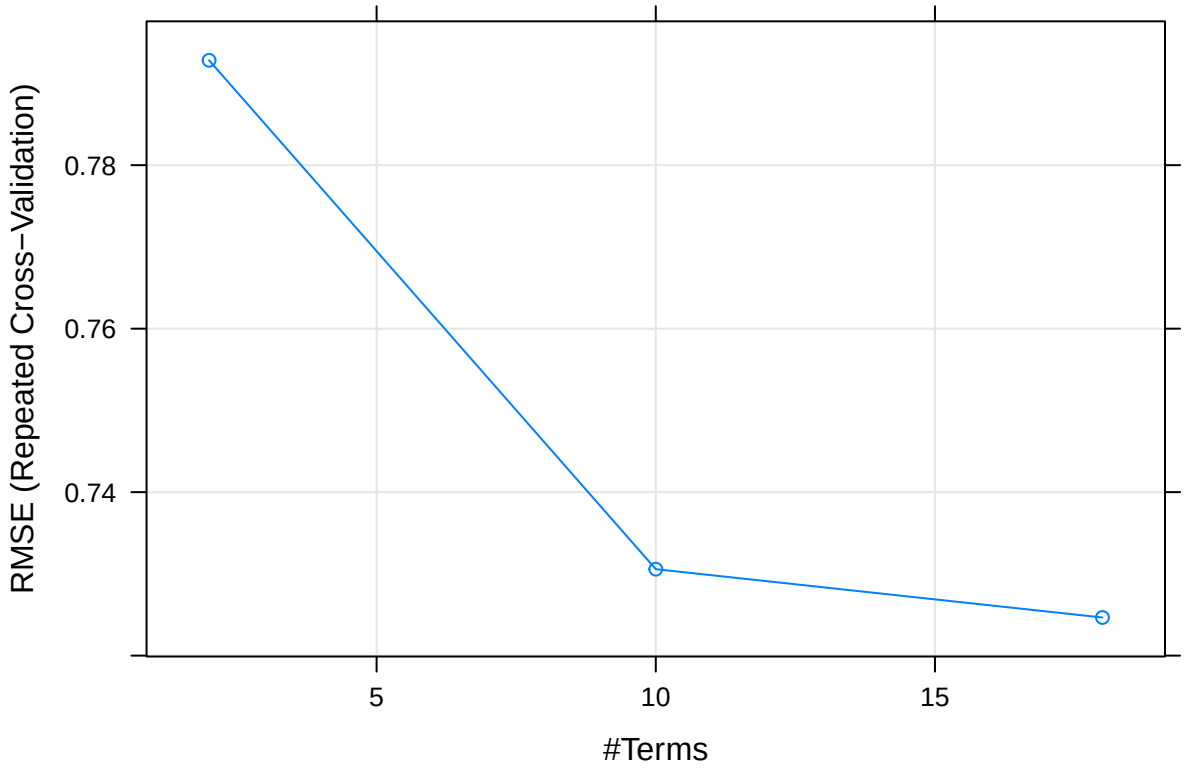

```
pls.fit <- train(x = wine_data[,-12], y = wine_data$quality,
                 method = 'pls', preProcess = c('center', 'scale'),
                 trControl=trainControl(method="repeatedcv",repeats=10,number=10))
plot(pls.fit)
```



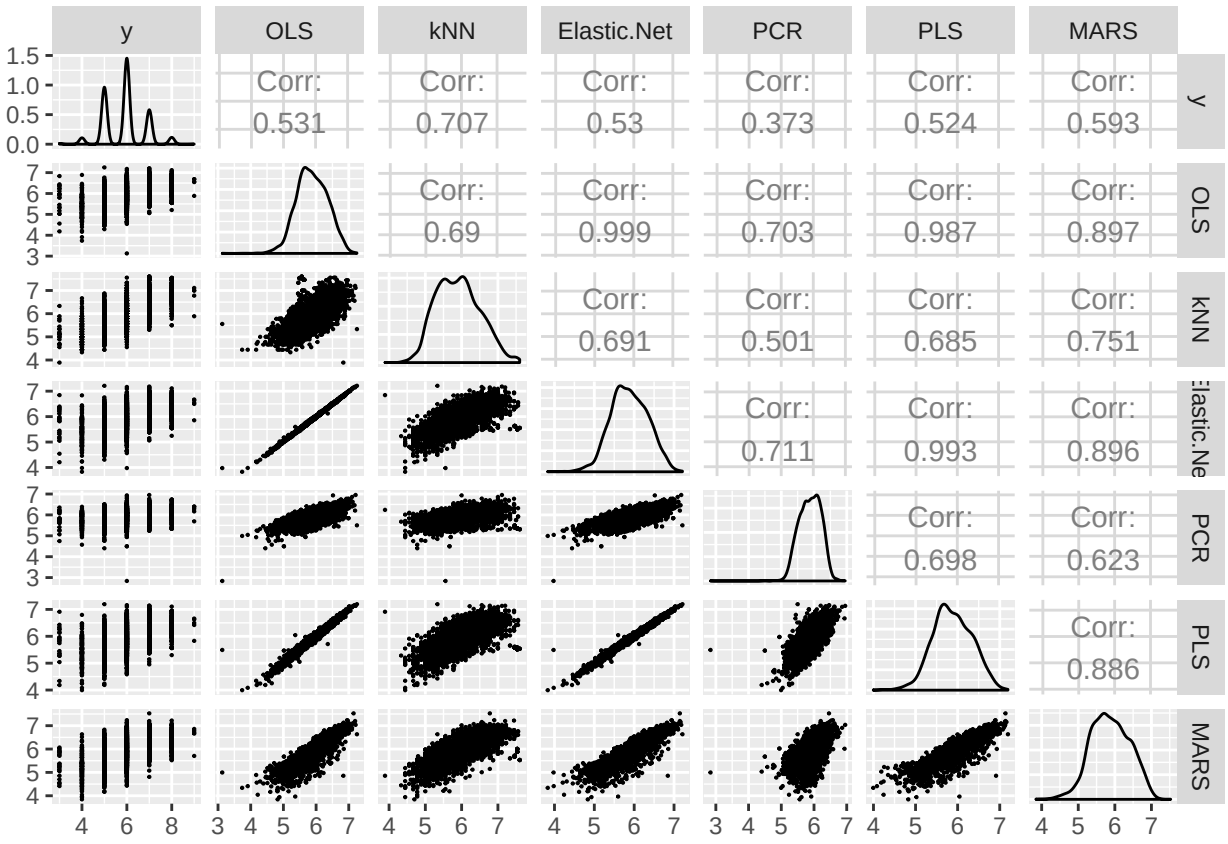
```
mars.fit <- train(x = wine_data[,-12], y = wine_data$quality,
                  method = 'earth', preProcess = c('center', 'scale'),
                  trControl=trainControl(method="repeatedcv",repeats=10,number=10))
```

```
## Loading required package: earth
## Loading required package: plotmo
## Loading required package: plotrix
## Loading required package: TeachingDemos
```

```
plot(mars.fit)
```




```
Model.fit <- list(ols.fit, knn.fit, net.fit, pcr.fit, pls.fit, mars.fit)
names(Model.fit) <- c('OLS', 'kNN', 'Elastic Net', 'PCR', 'PLS', 'MARS')

Model.pred <- sapply(Model.fit, FUN = predict)

Model.all <- data.frame(y = wine_data$quality, Model.pred)

ggpairs(Model.all,
        lower = list(continuous = wrap("points", size=0.1),
                     combo = wrap("dot", size=0.2) ))
```




```
Corr.y <- cor(Model.all)
round(Corr.y, digits = 2)
```

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

Problem 8

(a)

Denote function $T(x_i)(z) := T_i(z)$, then for L_2 norm:

$$\begin{aligned} < T_i, T_j >_{L_2} &= \int_{\mathbb{R}} \exp\{-\frac{(x_i - x)^2}{2} - \frac{(x_j - x)^2}{2}\} dx \\ &= \sqrt{\pi} \exp\{-\frac{(x_i - x_j)^2}{4}\} \end{aligned}$$

Define: $\rho_{ij} := \exp\{-\frac{(x_i - x_j)^2}{4}\}$, then:

$$
\begin{aligned}
U_1(x) &= T_1(x) \\
U_2(x) &= T_2(x) - \frac{< T_2, U_1 >}{< U_1, U_1 >} U_1(x) \\
U_3(x) &= T_3(x) - \frac{< T_3, U_1 >}{< U_1, U_1 >} U_1(x) - \frac{< T_3, U_2 >}{< U_2, U_2 >} U_2(x)
\end{aligned}
$$

Then:

$$
\begin{aligned}
U_1(x) &= T_1(x) \\
U_2(x) &= T_2(x) - \rho_{12} T_1(x) \\
U_3(x) &= T_3(x) - \frac{\rho_{23} - \rho_{12}\rho_{13}}{1 - \rho_{12}^2} T_2(x) - \frac{\rho_{13} - \rho_{12}\rho_{23}}{1 - \rho_{12}^2} T_1(x)
\end{aligned}
$$

By standardization, we will have:

$$
\begin{aligned}
Q_1(x) &= \pi^{-1/4} T_1(x) \\
Q_2(x) &= \pi^{-1/4} \frac{T_2(x) - \rho_{12} T_1(x)}{\sqrt{1 - \rho_{12}^2}} \\
Q_3(x) &= \pi^{-1/4} \frac{T_3(x) - \frac{\rho_{23} - \rho_{12}\rho_{13}}{1 - \rho_{12}^2} T_2(x) - \frac{\rho_{13} - \rho_{12}\rho_{23}}{1 - \rho_{12}^2} T_1(x)}{\sqrt{\frac{1 - \rho_{12}^2}{1 - \rho_{12}^2 - \rho_{13}^2 - \rho_{23}^2 + 2\rho_{12}\rho_{13}\rho_{23}}}}
\end{aligned}
$$

for kernel-based inner product:

$$
< T_i, T_j >_K = \mathcal{K}(x_i, x_j) := k_{ij}
$$

Thus, the orthonormal norm would be:

$$
\begin{aligned}
Q_1(x) &= T_1(x) \\
Q_2(x) &= \frac{T_2(x) - k_{12} T_1(x)}{\sqrt{1 - k_{12}^2}} \\
Q_3(x) &= \frac{T_3(x) - \frac{k_{23} - k_{12}k_{13}}{1 - k_{12}^2} T_2(x) - \frac{k_{13} - k_{12}k_{23}}{1 - k_{12}^2} T_1(x)}{\sqrt{\frac{1 - k_{12}^2}{1 - k_{12}^2 - k_{13}^2 - k_{23}^2 + 2k_{12}k_{13}k_{23}}}}
\end{aligned}
$$

Since $\rho_{ij} = \exp\{-\frac{(x_i - x_j)^2}{4}\} \neq \exp\{-\frac{(x_i - x_j)^2}{2}\} = k_{ij}$, in addition, $\pi^{-1/4}$ is multiplied to the orthonormal basis for L_2 norm, the two basis are not the same.

(b)

$$
\boldsymbol{G} = \boldsymbol{K} - \frac{1}{N}\boldsymbol{J}\boldsymbol{K} - \frac{1}{N}\boldsymbol{K}\boldsymbol{J} + \frac{1}{N^2}\boldsymbol{J}\boldsymbol{K}\boldsymbol{J}
$$

```
# center y, standardize x
x <- seq(0,1,0.1)
y <- c(0, 1.5, 2, 0.5, 0, -0.5, 0, 1.5, 3.5, 4.5, 3.5)

xs <- (x-mean(x))/sd(x)
z <- y - mean(y)
```

```r
# function to compute kernel matrix (gaussian kernel)
K_gauss <- function(x){
  n <- length(x)
  Ku <- matrix(0, ncol=n, nrow=n)
  for (i in 1:(n-1)){
    for (j in (i+1):n){
      Ku[i,j] <- exp(-((x[i]-x[j])^2)/2)
    }
  }
  K <- Ku + t(Ku) + diag(n)
  return(K)
}



# function to compute Gram matrix (gaussian kernel)
G_gauss <- function(x){
  n <- length(x)
  K <- K_gauss(x)
  G <- K
  k1 <- apply(K,2,mean)
  for (i in 1:n){
    G[i,] <- G[i,] - k1[i]
    G[,i] <- G[,i] - k1[i]
  }
  G <- G+mean(K)
  return(G)
}

# function to compute Gram matrix (gaussian kernel) : a slower version
G_gauss1 <- function(x){
  n <- length(x)
  K <- K_gauss(x)
  J <- matrix(1,ncol=n, nrow=n)
  return( K - (1/n)*J%*%K - (1/n)*K%*%J + (1/(n^2))*J%*%K%*%J )
}


G <- G_gauss(xs)
round(G,3)
```

```
##         [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
##  [1,]  0.730  0.599  0.404  0.179 -0.036 -0.210 -0.324 -0.377 -0.375
##  [2,]  0.599  0.558  0.440  0.263  0.059 -0.133 -0.284 -0.376 -0.408
##  [3,]  0.404  0.440  0.411  0.311  0.155 -0.026 -0.195 -0.323 -0.394
##  [4,]  0.179  0.263  0.311  0.300  0.222  0.088 -0.070 -0.217 -0.323
##  [5,] -0.036  0.059  0.155  0.222  0.232  0.176  0.066 -0.070 -0.195
##  [6,] -0.210 -0.133 -0.026  0.088  0.176  0.209  0.176  0.088 -0.026
##  [7,] -0.324 -0.284 -0.195 -0.070  0.066  0.176  0.232  0.222  0.155
##  [8,] -0.377 -0.376 -0.323 -0.217 -0.070  0.088  0.222  0.300  0.311
##  [9,] -0.375 -0.408 -0.394 -0.323 -0.195 -0.026  0.155  0.311  0.411
## [10,] -0.331 -0.387 -0.408 -0.376 -0.284 -0.133  0.059  0.263  0.440
## [11,] -0.260 -0.331 -0.375 -0.377 -0.324 -0.210 -0.036  0.179  0.404
##        [,10]  [,11]
```

```
##  [1,] -0.331 -0.260
##  [2,] -0.387 -0.331
##  [3,] -0.408 -0.375
##  [4,] -0.376 -0.377
##  [5,] -0.284 -0.324
##  [6,] -0.133 -0.210
##  [7,]  0.059 -0.036
##  [8,]  0.263  0.179
##  [9,]  0.440  0.404
## [10,]  0.558  0.599
## [11,]  0.599  0.730
```

(c)

```
# compute eigenvectors for G
eigen(G, symmetric = TRUE)$values
```

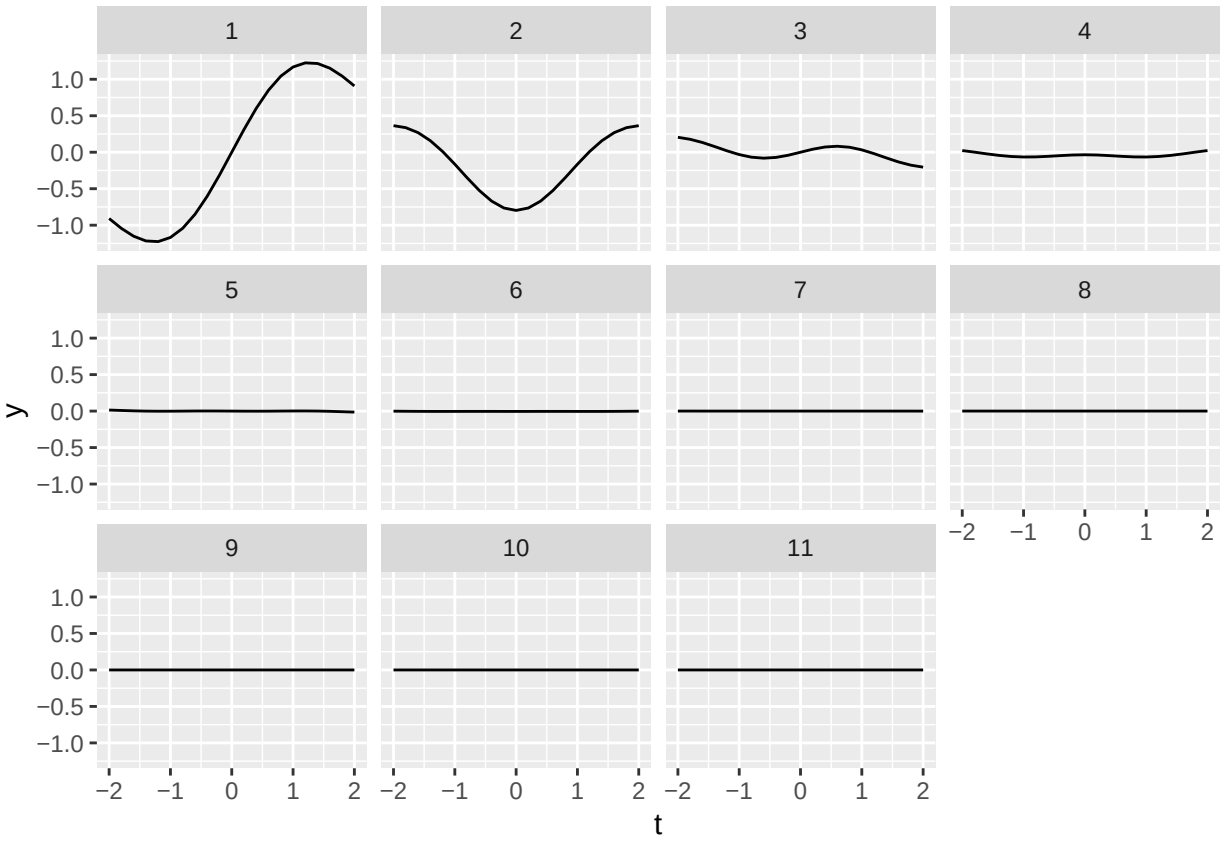
```
##  [1] 3.194981e+00 1.206335e+00 2.238951e-01 4.073087e-02 4.292423e-03
##  [6] 4.321162e-04 2.655836e-05 1.398045e-06 4.030029e-08 6.585064e-10
## [11] 8.077710e-17
```

```
PG <- eigen(G, symmetric = TRUE)$vectors
PG
```

```
##                [,1]        [,2]          [,3]       [,4]          [,5]
##  [1,] -3.717180e-01  0.43491586  4.854238e-01  0.4126071  3.309303e-01
##  [2,] -3.835607e-01  0.26285103  9.422307e-02 -0.2123822 -4.048419e-01
##  [3,] -3.495534e-01  0.04039045 -2.318789e-01 -0.3955577 -2.299833e-01
##  [4,] -2.673548e-01 -0.18323666 -3.646948e-01 -0.1710918  2.344339e-01
##  [5,] -1.450543e-01 -0.34946016 -2.621365e-01  0.1883085  3.445818e-01
##  [6,]  1.665335e-16 -0.41092106  8.673617e-17  0.3562325  9.381385e-15
##  [7,]  1.450543e-01 -0.34946016  2.621365e-01  0.1883085 -3.445818e-01
##  [8,]  2.673548e-01 -0.18323666  3.646948e-01 -0.1710918 -2.344339e-01
##  [9,]  3.495534e-01  0.04039045  2.318789e-01 -0.3955577  2.299833e-01
## [10,]  3.835607e-01  0.26285103 -9.422307e-02 -0.2123822  4.048419e-01
## [11,]  3.717180e-01  0.43491586 -4.854238e-01  0.4126071 -3.309303e-01
##               [,6]          [,7]        [,8]          [,9]        [,10]
##  [1,]  0.21475591  1.270373e-01 -0.05949242 -2.315359e-02  0.005920587
##  [2,] -0.47387424 -4.041439e-01  0.26194691  1.295145e-01 -0.041485683
##  [3,]  0.12219809  4.020033e-01 -0.46484406 -3.309767e-01  0.141354953
##  [4,]  0.37649906  8.847412e-02  0.33028609  4.824232e-01 -0.308872826
##  [5,] -0.05267738 -3.887093e-01  0.13694640 -3.747161e-01  0.479561393
##  [6,] -0.37380286  2.487240e-12 -0.40968586 -3.120431e-10 -0.552957700
##  [7,] -0.05267738  3.887093e-01  0.13694640  3.747161e-01  0.479561393
##  [8,]  0.37649906 -8.847412e-02  0.33028609 -4.824232e-01 -0.308872826
##  [9,]  0.12219809 -4.020033e-01 -0.46484406  3.309767e-01  0.141354953
## [10,] -0.47387424  4.041439e-01  0.26194691 -1.295145e-01 -0.041485683
## [11,]  0.21475591 -1.270373e-01 -0.05949242  2.315359e-02  0.005920587
##            [,11]
##  [1,] -0.3015113
##  [2,] -0.3015113
##  [3,] -0.3015114
##  [4,] -0.3015113
##  [5,] -0.3015115
```

```
##  [6,] -0.3015112
##  [7,] -0.3015115
##  [8,] -0.3015113
##  [9,] -0.3015114
## [10,] -0.3015113
## [11,] -0.3015113
```

```r
# function to compute S(xi)(t)
# return a matrix, row i is the value of S(xi)(t)
S_mat <- function(t, x){
  n <- length(x)
  S_mat <- matrix(0, ncol = length(t), nrow = n)
  T_mat <- matrix(0, ncol = length(t), nrow = n)
  for (j in 1:n){
    T_mat[j,] <- exp(-((x[j]-t)^2)/2)
  }
  for (i in 1:n){
    S_mat[i,] <- T_mat[i,] - apply(T_mat, 2, mean)
  }
  return(S_mat)
}

# function to compute the linear combinations of S(xi)(t),
# the coeficients are eigenvectors.
# return a matrix, row j is the value of sum(  uij * S(xi)(t)  )
PS_mat <- function(t,x,PG){
  S <- S_mat(t,x)
  n <- length(x)
  PS <- t(S)%*%PG
  return(PS)
}


# plot the linear combination of S(xi)(t),
t0 <- seq(-2,2,0.2)
PS <- PS_mat(t0, xs, PG)
nt <- length(t0)
D8 <- data.frame(t = rep(t0,11), y = as.numeric(PS),
                 eigen_ind = as.factor( rep(1:11, each = nt) )
                 )

ggplot(data = D8) + geom_line(aes(x = t, y = y)) +
  facet_wrap(~eigen_ind, nrow=3)
```

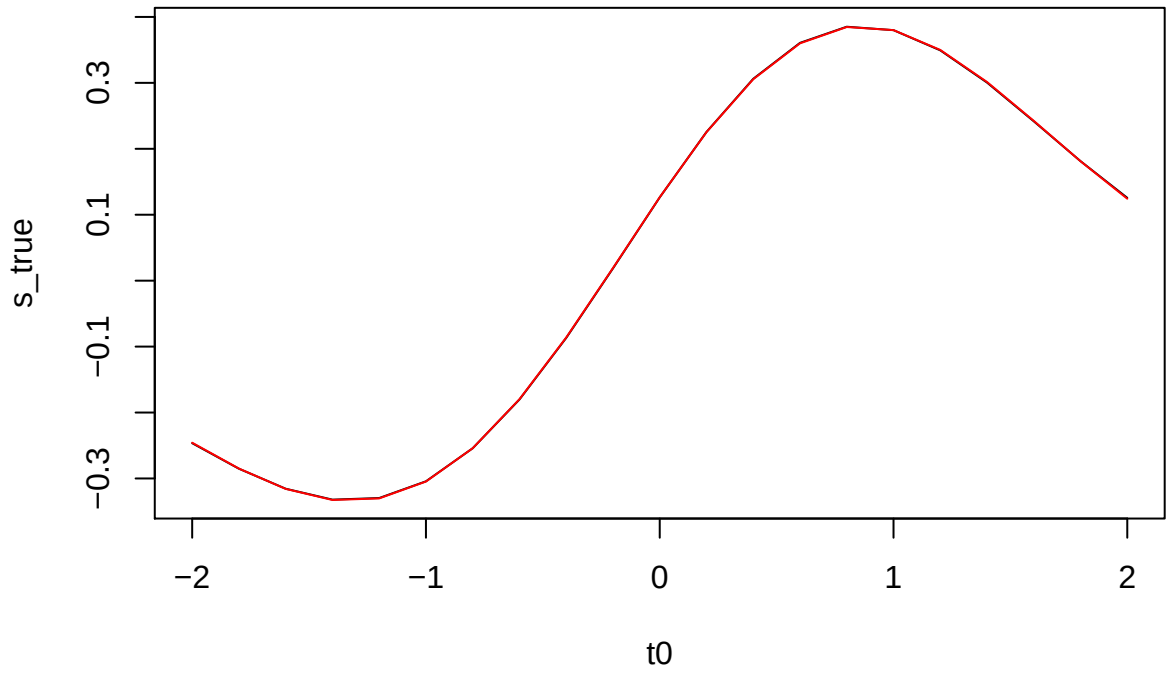


We find that the functions become more and more flat (include less information about S(xi)(t)) as the reduce of eigenvalues.

(d)

```r
# function to compute S(x0)(t) = T(x0)(t) - M(x)(t)
S_func <- function(t, x, x0){
  n <- length(x)
  T_mat <- matrix(0, ncol = length(t), nrow = n)
  for (j in 1:n){
    T_mat[j,] <- exp(-((x[j]-t)^2)/2)
  }
    S_func <- exp(-((x0-t)^2)/2) - apply(T_mat, 2, mean)
    return(S_func)
}


s_true <- S_func(t0, xs, 0.65)

# OLS fit using the function we have in (c):
# we only use the first four functions since the other functions
# are really close to the constant function (y=0)
s_fit <- fitted(lm(s_true~PS[,1:4]))

# plot ( true:black vs fitted:red )
plot(t0, s_true, type = "l")
```

```
lines(t0, s_fit, col = "red")
```



We find that the fitted function are really close to the true function.