Assignment2

QINGTAN

01/10/2022

Q1.

1.1

Vector k will be an output in this question.

```
library(HRW)
data('WarsawApts')
x <- WarsawApts$construction.date
y <- WarsawApts$areaPerMzloty
k_22 <- seq(0, 1, length=22)
k_20 = k_22[2:21]
k = quantile(unique(x), k_20)
k
```

```
## 4.761905%  9.52381% 14.28571% 19.04762% 23.80952% 28.57143% 33.33333% 38.09524%
##      1935      1938      1947      1950      1953      1956      1959      1962
## 42.85714% 47.61905% 52.38095% 57.14286% 61.90476% 66.66667% 71.42857% 76.19048%
##      1965      1968      1971      1974      1977      1980      1984      1991
## 80.95238% 85.71429% 90.47619%  95.2381%
##      1994      1999      2002      2005
```

1.2

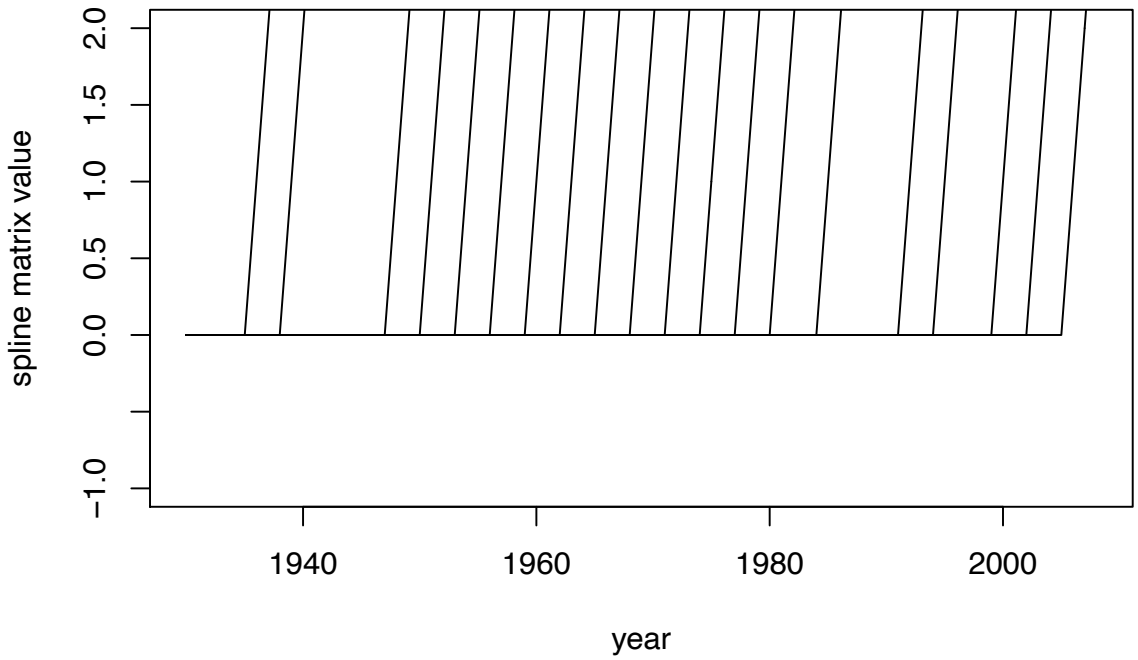
An initialized matrix should be built firstly:

```
l_x = length(x)
l_k = length(k)
Z_m = matrix(0, l_x, l_k)
```

Then we put values in this matrix and draw the plot:

```
for (i in 1:l_x){
  for (j in 1:l_k){
    val = x[i] - k[j]
    if (val > 0){
      Z_m[i,j] <- val
    }else{
```

```
      Z_m[i,j] <- 0
    }
  }
}

# use the first column of Z_m to define the characters of this plot, then build a loop for other column
plot(x[order(x)], Z_m[order(Z_m[,1]),1], ylim=c(-1,2), type='l', xlab='year', ylab = 'spline matrix valu
for (j in 2:l_k){
  lines(x[order(x)], Z_m[order(Z_m[,j]), j])
}
```



1.3

We should add 1 column at the first position to create matrix C:

```
add_one = rep(1, l_x)
C = cbind(add_one, x, Z_m)
```

Then matrix D will be created

```
D = matrix(0, ncol=22, nrow=22)
for (i in 3:22){
  for (j in 3:22){
    if (i == j){
```

```
      D[i,j] <- 1
    }
  }
}
```

Using 100 parameters to get our estimate beta and estimate value y:

```
beta_result = matrix(ncol=22, nrow=100)
y_result = matrix(ncol=100, nrow=409)
tuning = seq(0, 50, length=100)
for (i in 1:length(tuning)){
  beta_result[i, ] = solve(t(C)%*%C + tuning[i]*D)%*%t(C)%*%y
  y_result[ ,i] = C%*%beta_result[i, ]
}
```

Here is head of estimated beta value matrix:

```
head(beta_result)
```

```
##            [,1]        [,2]     [,3]      [,4]     [,5]     [,6]      [,7]
## [1,]  223.84853 -0.06541680 3.105945 -5.577324 7.212058 1.051256 -6.066614
## [2,]   88.22370  0.00479189 2.874079 -5.344824 7.016958 1.123769 -6.017647
## [3,]  -32.05278  0.06705608 2.666028 -5.134034 6.839415 1.177265 -5.942086
## [4,] -139.17256  0.12251017 2.478429 -4.941829 6.676133 1.218094 -5.852090
## [5,] -234.93182  0.17208387 2.308538 -4.765713 6.524811 1.250082 -5.754639
## [6,] -320.81895  0.21654757 2.154080 -4.603647 6.383750 1.275657 -5.653874
##           [,8]     [,9]     [,10]      [,11]    [,12]    [,13]     [,14]
## [1,] -2.789303 7.309482 -4.472600 -0.1416538 1.757304 1.778337 -1.915616
## [2,] -2.600559 6.854383 -4.046430 -0.3223312 1.793392 1.750160 -1.916435
## [3,] -2.464407 6.473800 -3.690691 -0.4644901 1.817234 1.724219 -1.907766
## [4,] -2.362814 6.147891 -3.388226 -0.5777335 1.831992 1.700573 -1.894311
## [5,] -2.284786 5.863650 -3.127333 -0.6686992 1.839939 1.678989 -1.878555
## [6,] -2.223317 5.612136 -2.899679 -0.7421516 1.842742 1.659154 -1.861844
##          [,15]     [,16]     [,17]    [,18]     [,19]    [,20]     [,21]
## [1,] -2.075714 1.6860223 -4.739524 17.74080 -23.37947 6.709464 0.8013691
## [2,] -2.001712 1.2341927 -3.792505 15.58043 -21.11652 5.640377 1.4411330
## [3,] -1.954892 0.9091498 -3.081072 13.87159 -19.31851 4.822914 1.9156639
## [4,] -1.924178 0.6690235 -2.532338 12.48346 -17.85174 4.181519 2.2765752
## [5,] -1.903230 0.4878436 -2.100132 11.33171 -16.62978 3.667834 2.5565119
## [6,] -1.888280 0.3487778 -1.753871 10.35945 -15.59418 3.249507 2.7770466
##         [,22]
## [1,] 8.466636
## [2,] 8.075135
## [3,] 7.764532
## [4,] 7.509522
## [5,] 7.294255
## [6,] 7.108356
```

Here is the head of estimated y value matrix:

```
head(y_result)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 117.36075 117.38161 117.40255 117.42328 117.44370 117.46378 117.48353
## [2,] 116.89455 116.49031 116.15176 115.86317 115.61376 115.39582 115.20363
## [3,]  97.58322  97.59205  97.60682  97.62515  97.64566  97.66751  97.69018
## [4,] 114.71448 114.71045 114.71237 114.71856 114.72783 114.73929 114.75231
## [5,] 109.03626 108.94160 108.83828 108.73182 108.62526 108.52029 108.41784
## [6,]  95.89903  95.85689  95.81750  95.78017  95.74444  95.71005  95.67679
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] 117.50296 117.52207 117.54089 117.55941 117.57766 117.59564 117.61334
## [2,] 115.03284 114.88009 114.74268 114.61845 114.50566 114.40283 114.30877
## [3,]  97.71332  97.73670  97.76016  97.78359  97.80692  97.83009  97.85307
## [4,] 114.76641 114.78124 114.79654 114.81210 114.82778 114.84347 114.85909
## [5,] 108.31841 108.22223 108.12939 108.03985 107.95355 107.87039 107.79024
## [6,]  95.64454  95.61318  95.58265  95.55287  95.52380  95.49540  95.46763
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
## [1,] 117.63079 117.64797 117.66490 117.68158 117.69800 117.71418 117.73011
## [2,] 114.22245 114.14299 114.06966 114.00182 113.93892 113.88047 113.82605
## [3,]  97.87583  97.89835  97.92063  97.94264  97.96440  97.98589  98.00711
## [4,] 114.87457 114.88987 114.90495 114.91980 114.93439 114.94872 114.96278
## [5,] 107.71298 107.63845 107.56655 107.49713 107.43008 107.36528 107.30261
## [6,]  95.44046  95.41385  95.38779  95.36225  95.33721  95.31265  95.28854
##          [,22]     [,23]     [,24]     [,25]     [,26]    [,27]     [,28]
## [1,] 117.74580 117.76125 117.77646 117.79143 117.80617 117.8207 117.83498
## [2,] 113.77530 113.72788 113.68350 113.64192 113.60288 113.5662 113.53168
## [3,]  98.02807  98.04877  98.06921  98.08939  98.10932  98.1290  98.14843
## [4,] 114.97657 114.99009 115.00335 115.01634 115.02907 115.0415 115.05378
## [5,] 107.24197 107.18327 107.12641 107.07130 107.01786 106.9660 106.91568
## [6,]  95.26488  95.24165  95.21883  95.19641  95.17437  95.1527  95.13138
##          [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
## [1,] 117.84904 117.86289 117.87652 117.88994 117.90315 117.91616 117.92896
## [2,] 113.49916 113.46849 113.43952 113.41215 113.38625 113.36171 113.33846
## [3,]  98.16763  98.18658  98.20531  98.22381  98.24208  98.26014  98.27798
## [4,] 115.06577 115.07752 115.08906 115.10037 115.11147 115.12237 115.13306
## [5,] 106.86680 106.81932 106.77315 106.72826 106.68459 106.64208 106.60069
## [6,]  95.11042  95.08979  95.06948  95.04949  95.02980  95.01041  94.99131
##          [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
## [1,] 117.94157 117.95398 117.96620 117.97824 117.99009 118.00176 118.01326
## [2,] 113.31640 113.29545 113.27555 113.25661 113.23859 113.22143 113.20508
## [3,]  98.29561  98.31304  98.33026  98.34729  98.36412  98.38076  98.39722
## [4,] 115.14357 115.15389 115.16403 115.17400 115.18380 115.19345 115.20293
## [5,] 106.56037 106.52109 106.48279 106.44545 106.40902 106.37347 106.33878
## [6,]  94.97248  94.95393  94.93563  94.91760  94.89981  94.88226  94.86494
##          [,43]     [,44]     [,45]     [,46]     [,47]     [,48]    [,49]
## [1,] 118.02458 118.03574 118.04672 118.05755 118.06822 118.07873 118.0891
## [2,] 113.18948 113.17459 113.16037 113.14679 113.13380 113.12137 113.1095
## [3,]  98.41349  98.42959  98.44551  98.46125  98.47683  98.49225  98.5075
## [4,] 115.21227 115.22145 115.23050 115.23942 115.24820 115.25685 115.2654
## [5,] 106.30490 106.27181 106.23949 106.20789 106.17701 106.14681 106.1173
## [6,]  94.84785  94.83099  94.81434  94.79790  94.78167  94.76563  94.7498
##          [,50]     [,51]     [,52]     [,53]     [,54]     [,55]     [,56]
## [1,] 118.09930 118.10936 118.11928 118.12906 118.13870 118.14820 118.15758
```

```
## [2,] 113.09809 113.08717 113.07671 113.06667 113.05704 113.04779 113.03891
## [3,]  98.52259  98.53753  98.55231  98.56694  98.58143  98.59577  98.60997
## [4,] 115.27379 115.28209 115.29027 115.29835 115.30632 115.31419 115.32196
## [5,] 106.08838 106.06011 106.03243 106.00534 105.97880 105.95282 105.92736
## [6,]  94.73415  94.71868  94.70340  94.68830  94.67337  94.65860  94.64400
##          [,57]     [,58]     [,59]     [,60]     [,61]     [,62]     [,63]
## [1,] 118.16683 118.17595 118.18494 118.19382 118.20257 118.21121 118.21974
## [2,] 113.03037 113.02216 113.01426 113.00667 112.99935 112.99231 112.98552
## [3,]  98.62403  98.63795  98.65174  98.66540  98.67892  98.69232  98.70559
## [4,] 115.32964 115.33722 115.34471 115.35211 115.35943 115.36666 115.37382
## [5,] 105.90241 105.87796 105.85399 105.83048 105.80744 105.78483 105.76265
## [6,]  94.62956  94.61528  94.60116  94.58718  94.57335  94.55966  94.54612
##          [,64]     [,65]     [,66]     [,67]     [,68]     [,69]     [,70]
## [1,] 118.22816 118.23646 118.24466 118.25276 118.26075 118.26864 118.27644
## [2,] 112.97898 112.97267 112.96659 112.96071 112.95505 112.94957 112.94428
## [3,]  98.71875  98.73178  98.74469  98.75748  98.77016  98.78273  98.79518
## [4,] 115.38089 115.38789 115.39482 115.40167 115.40845 115.41517 115.42181
## [5,] 105.74089 105.71953 105.69856 105.67798 105.65777 105.63793 105.61843
## [6,]  94.53271  94.51944  94.50630  94.49329  94.48041  94.46765  94.45501
##          [,71]     [,72]     [,73]     [,74]     [,75]     [,76]     [,77]
## [1,] 118.28414 118.29174 118.29925 118.30667 118.31401 118.32125 118.32841
## [2,] 112.93917 112.93423 112.92945 112.92483 112.92035 112.91602 112.91183
## [3,]  98.80753  98.81977  98.83190  98.84393  98.85585  98.86768  98.87941
## [4,] 115.42840 115.43491 115.44137 115.44776 115.45410 115.46038 115.46660
## [5,] 105.59929 105.58047 105.56198 105.54381 105.52595 105.50839 105.49113
## [6,]  94.44249  94.43009  94.41781  94.40564  94.39357  94.38162  94.36977
##          [,78]     [,79]     [,80]     [,81]     [,82]     [,83]     [,84]
## [1,] 118.33549 118.34249 118.34940 118.35624 118.36300 118.36968 118.37629
## [2,] 112.90776 112.90382 112.90001 112.89630 112.89271 112.88923 112.88584
## [3,]  98.89103  98.90257  98.91400  98.92535  98.93660  98.94776  98.95884
## [4,] 115.47277 115.47888 115.48494 115.49095 115.49691 115.50282 115.50868
## [5,] 105.47415 105.45745 105.44103 105.42488 105.40898 105.39334 105.37795
## [6,]  94.35803  94.34639  94.33486  94.32342  94.31208  94.30083  94.28968
##          [,85]     [,86]     [,87]     [,88]     [,89]     [,90]     [,91]
## [1,] 118.38283 118.38929 118.39569 118.40202 118.40828 118.41447 118.42060
## [2,] 112.88256 112.87937 112.87627 112.87325 112.87032 112.86748 112.86470
## [3,]  98.96982  98.98072  98.99153  99.00226  99.01291  99.02347  99.03396
## [4,] 115.51449 115.52026 115.52598 115.53166 115.53729 115.54288 115.54843
## [5,] 105.36280 105.34789 105.33321 105.31875 105.30452 105.29051 105.27670
## [6,]  94.27862  94.26766  94.25678  94.24599  94.23529  94.22467  94.21413
##          [,92]     [,93]     [,94]     [,95]     [,96]     [,97]     [,98]
## [1,] 118.42667 118.43268 118.43862 118.44450 118.45033 118.45609 118.46180
## [2,] 112.86201 112.85938 112.85682 112.85433 112.85191 112.84954 112.84724
## [3,]  99.04436  99.05469  99.06494  99.07511  99.08521  99.09524  99.10519
## [4,] 115.55394 115.55941 115.56484 115.57023 115.57558 115.58090 115.58618
## [5,] 105.26311 105.24972 105.23652 105.22352 105.21071 105.19809 105.18564
## [6,]  94.20368  94.19331  94.18302  94.17281  94.16267  94.15261  94.14263
##          [,99]    [,100]
## [1,] 118.46746 118.47306
## [2,] 112.84499 112.84280
## [3,]  99.11507  99.12488
## [4,] 115.59142 115.59663
## [5,] 105.17338 105.16129
## [6,]  94.13272  94.12288
```

1.4

We make two lists to store degrees of freedom and generalization cross validation:

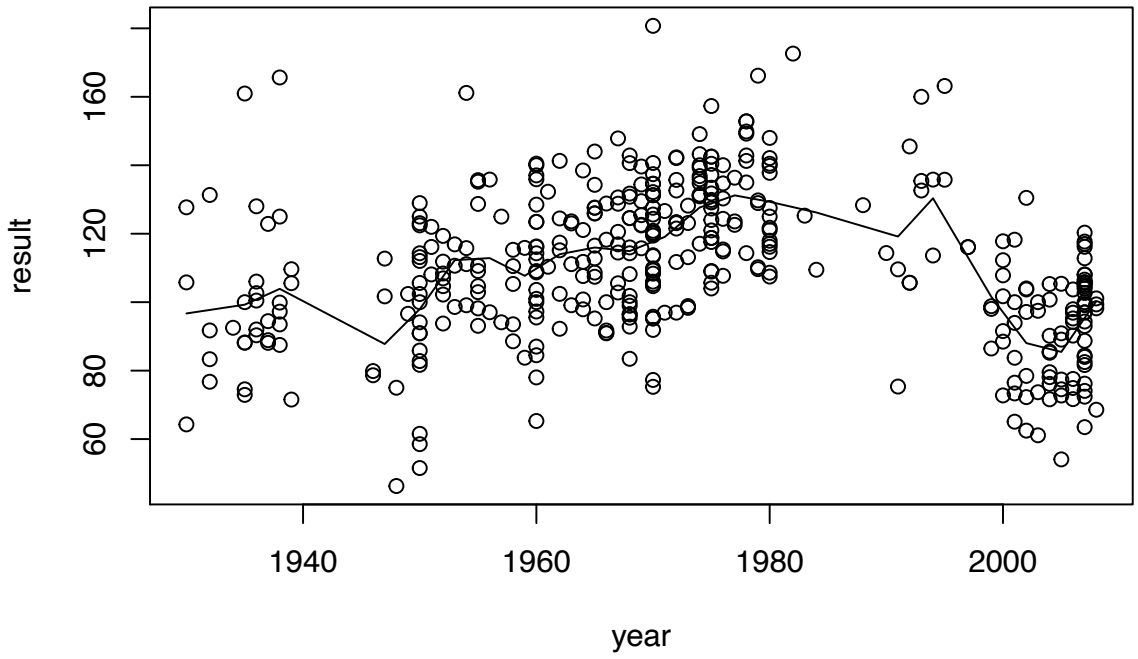
```
library(lava)
sum = rep(0,100)
deg_free = rep(0,100)
gcv = rep(0,100)
for (i in 1:100){
  sum[i] = 0
  for (j in 1:409){
    sum[i] = sum[i] + (y_result[j,i] - y[j])^2
  }
  deg_free[i] = tr(solve(t(C)%*%C + tuning[i]*D)%*%t(C)%*%C)
  gcv[i] = sum[i]/(1-deg_free[i]/409)^2
}
gcv
```

```
##   [1] 133258.2 132827.0 132540.8 132341.8 132198.4 132092.2 132011.9 131950.2
##   [9] 131902.4 131865.1 131836.0 131813.5 131796.2 131783.2 131773.9 131767.5
##  [17] 131763.7 131762.1 131762.3 131764.2 131767.5 131772.1 131777.8 131784.4
##  [25] 131791.9 131800.1 131809.1 131818.6 131828.7 131839.3 131850.3 131861.6
##  [33] 131873.4 131885.4 131897.7 131910.3 131923.1 131936.0 131949.2 131962.5
##  [41] 131975.9 131989.4 132003.1 132016.8 132030.6 132044.5 132058.5 132072.4
##  [49] 132086.4 132100.5 132114.5 132128.6 132142.6 132156.7 132170.7 132184.8
##  [57] 132198.8 132212.8 132226.8 132240.7 132254.7 132268.6 132282.4 132296.2
##  [65] 132310.0 132323.7 132337.4 132351.0 132364.6 132378.1 132391.6 132405.0
##  [73] 132418.4 132431.7 132444.9 132458.1 132471.3 132484.4 132497.4 132510.4
##  [81] 132523.3 132536.1 132548.9 132561.6 132574.3 132586.9 132599.4 132611.9
##  [89] 132624.3 132636.7 132649.0 132661.2 132673.4 132685.5 132697.6 132709.6
##  [97] 132721.5 132733.4 132745.2 132757.0
```

1.5

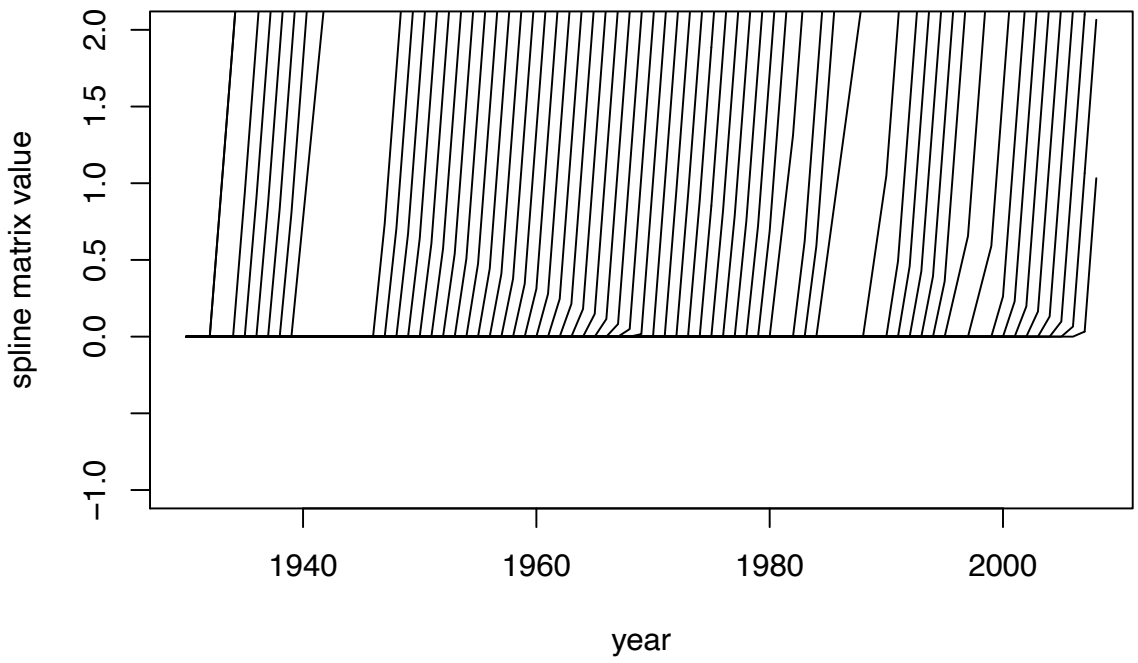
Firstly, find the best parameter from the cross validation test and draw the lines for 20 knots:

```
best_beta = beta_result[which(gcv==min(gcv))]
best_y = y_result[, which(gcv==min(gcv))]

# plot points firstly
plot(x, y, xlab='year', ylab='result')
x_line = x[order(x)]
y_line = best_y[order(x)]
lines(x_line, y_line)
```



Then we change it to 60 knots:

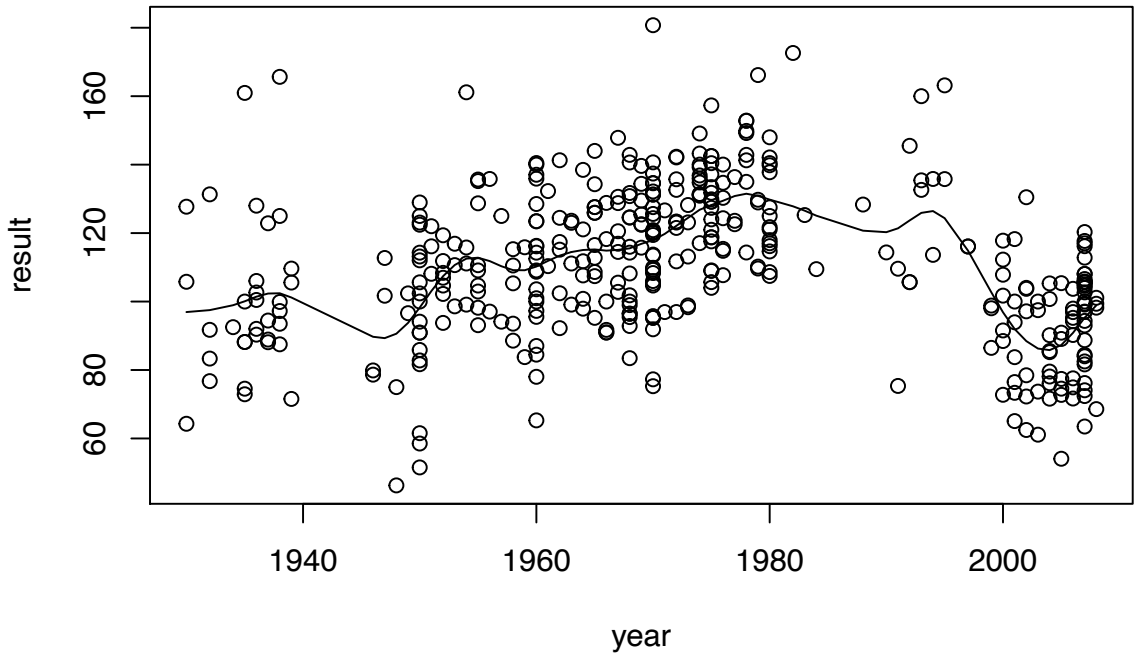
```
k_62 <- seq(0, 1, length=62)
k_60 = k_62[2:61]
new_k = quantile(unique(x), k_60)

l_x = length(x)
newl_k = length(new_k)
newZ_m = matrix(0, l_x, newl_k)

for (i in 1:l_x){
  for (j in 1:newl_k){
    val = x[i] - new_k[j]
    if (val > 0){
      newZ_m[i,j] <- val
    }else{
      newZ_m[i,j] <- 0
    }
  }
}

# use the first column of Z_m to define the characters of this plot, then build a loop for other column
plot(x[order(x)], newZ_m[order(newZ_m[,1]),1], ylim=c(-1,2), type='l', xlab='year', ylab = 'spline matr
for (j in 1:newl_k){
  lines(x[order(x)], newZ_m[order(newZ_m[,j]), j])
}
```



Then we draw the lines in the plot graph:

```
nC = cbind(add_one, x, newZ_m)
nD = matrix(0, ncol=62, nrow=62)
for (i in 3:62){
  for (j in 3:62){
    if (i == j){
      nD[i,j] <- 1
    }
  }
}
nbeta_result = matrix(ncol=62, nrow=100)
ny_result = matrix(ncol=100, nrow=409)
for (i in 1:length(tuning)){
  nbeta_result[i, ] = solve(t(nC)%*%nC + tuning[i]*nD)%*%t(nC)%*%y
  ny_result[ ,i] =nC%*%nbeta_result[i, ]
}

ndeg_free = rep(0,100)
ngcv = rep(0,100)
for (i in 1:100){
  nsum = 0
  for (j in 1:409){
    nsum = nsum + (ny_result[j,i] - y[j])^2
  }
  ndeg_free[i] = tr(solve(t(nC)%*%nC + tuning[i]*nD)%*%t(nC)%*%nC)
```

```
  ngcv[i] = nsum/(1-ndeg_free[i]/409)^2
}

nbest_beta = nbeta_result[which(ngcv==min(ngcv))]
nbest_y = ny_result[, which(ngcv==min(ngcv))]

# plot points firstly
plot(x, y, xlab='year', ylab='result')
nx_line = x[order(x)]
ny_line = nbest_y[order(x)]
lines(nx_line, ny_line)
```



Here is the residual calculation, this is the mean square error of 20 knots:

```
res20 = sum((y - best_y)^2)/409
res20
```

```
## [1] 297.0014
```

This is the mean square error of 60 knots:

```
res60 = sum((y - nbest_y)^2)/409
res60
```

```
## [1] 297.467
```

Conclusion: we can find that the MSE of 60 knots is a little larger than the MSE of 20 knots, which means that the increase of basis do not make MSE lower, the performance is even worse. So the performance are not always better when increasing the number of basis.

Q2.

2.1

The output is head of the result matrix we want:

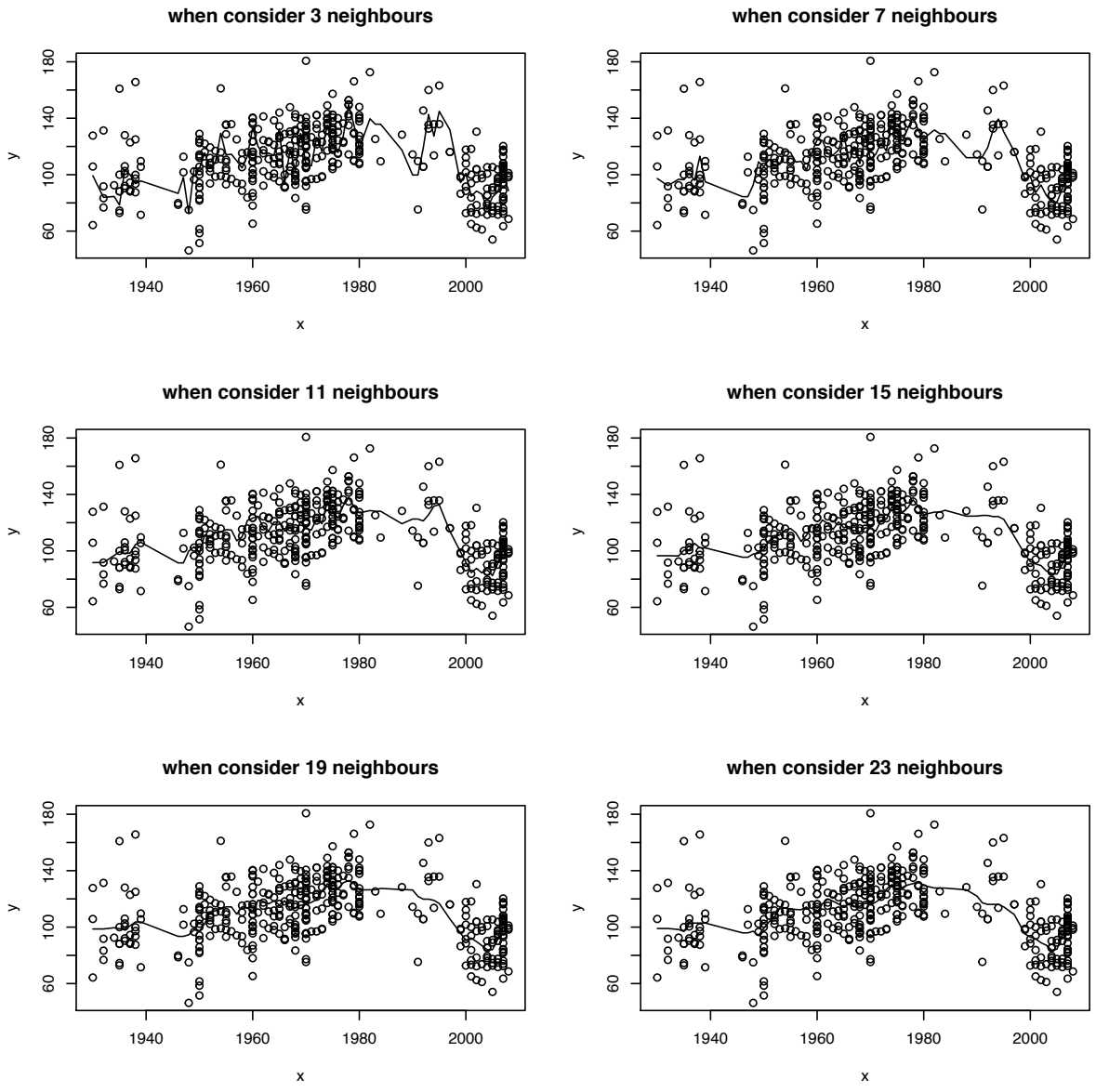
```
k_num = seq(3, 23, length=6)

# fill in the distance matrix
distance = matrix(ncol=409, nrow=409)
for (i in 1:409){
  distance[,i] = abs(x - x[i])
}
index_matrix = matrix(ncol=409, nrow=409)

# fill in the index matrix
for (i in 1:409){
  index_matrix[, i] = sort(distance[, i], index.return=TRUE)$ix
}

# create the result matrix
result_matrix = matrix(ncol=6, nrow=409)

# fill the result matrix
for (i in 1:409){
  for (j in 1:6){
    sum = 0
    for (k in index_matrix[1:k_num[j], i]){
      sum = sum + y[k]
    }
    result_matrix[i, j] = sum / k_num[j]
  }
}
head(result_matrix)
```

```
##           [,1]      [,2]      [,3]     [,4]      [,5]      [,6]
## [1,] 107.24234 111.05154 115.85830 116.0398 116.10845 115.78676
## [2,] 114.76050 114.75642 116.48079 119.3693 118.23894 119.95862
## [3,] 104.43510 105.61586  98.36686 100.8799  97.09665  99.21578
## [4,] 103.24733 107.89768 112.87928 112.9604 113.74375 114.71653
## [5,] 104.65925 106.74102 110.02975 109.8012 110.28319 107.09947
## [6,]  99.63521  97.71037 101.13791  97.9760  99.84684  98.84334
```

2.2

Firstly, here is the six plots we get:

```
par(mfrow=c(3,2))
for (i in 1:6){
  plot(x, y, main=paste('when consider', k_num[i], 'neighbours'))
  lines(x[order(x)], result_matrix[order(x), i])
}
```



Then we calculate the MSE results for these six k values, and store them into a list (from k=1 to k=6):

```
KMSE_result = rep(0, 6)
for (i in 1:6){
  KMSE_result[i] = sum((y - result_matrix[, i])^2)/409
}
KMSE_result
```

```
## [1] 331.9637 298.1435 301.1193 302.2849 305.0559 306.4572
```

Then we find the k value with minimun MSE value:

```
bestK = k_num[which(KMSE_result==min(KMSE_result))]
bestK
```

```
## [1] 7
```

And the minimum MSE value is:

```
min(KMSE_result)
```

```
## [1] 298.1435
```

Conclusion: When k=7 we find the best model, which has the lowest mean square error (298.1435). We already get that the lowest value for linear spline basis is 297.0014, which is greater than 298.1435. So the model we get from linear spline basis is better than the KNN model, and linear spline basis is more precise than KNN model.

2.3

The x in six methods formula is defined as $(x - x_0)/h$, then we have this function:

```
accomm_func = function(xvalue, x0, h){
  x = (xvalue - x0) / h
  if (abs(x) < 1){
    epan = 3/4*(1-x^2)
    gaus = (2*pi)^(-1/2)*exp(-x^2/2)
    biwe = 15/16*(1-x^2)^2
    triw = 35/32*(1-x^2)^3
    unif = 1/2
    tric = 70/81*(1-(abs(x))^3)^3
  }else{
    epan = 0
    gaus = (2*pi)^(-1/2)*exp(-x^2/2)
    biwe = 0
    triw = 0
    unif = 0
    tric = 0
  }
  value_matrix = cbind(epan, gaus, biwe, triw, unif, tric)
  return(value_matrix)
}
```

2.4

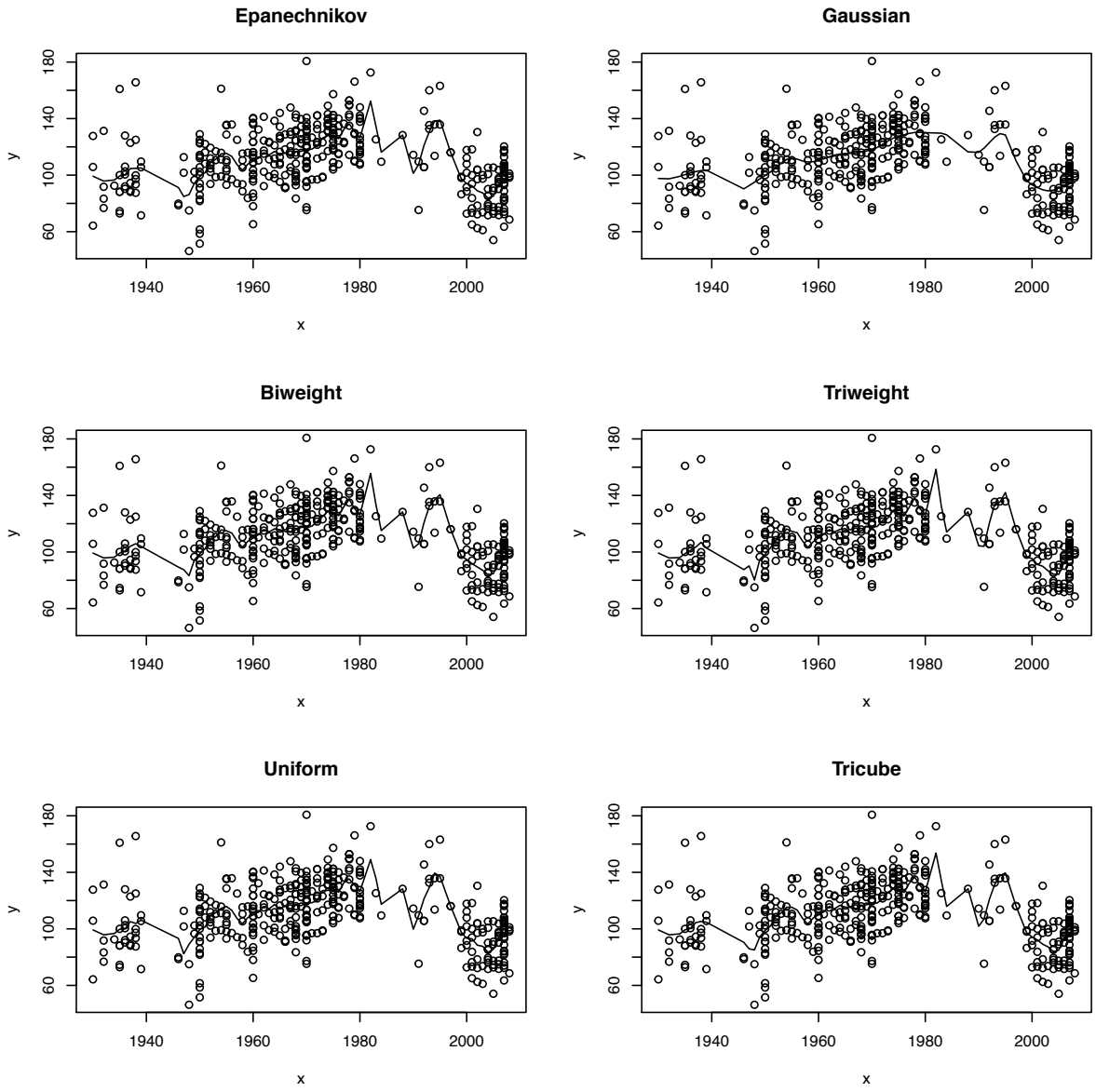
We output the head of the result matrix:

```r
matrix1 = matrix(nrow=409, ncol=6)
for (i in 1:409){
  for (j in 1:6){
    numer = 0
    denom = 0
    for (k in 1:409){
      numer = numer + y[k]*accomm_func(x[i], x[k], 2)[j]
      denom = denom + accomm_func(x[i], x[k], 2)[j]
    }
    # denominator cannot be zero
    if (denom == 0){
      matrix1[i, j] = 0
    }else{
      matrix1[i, j] = numer/denom
    }
  }
}
head(matrix1)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 117.95666 118.09604 117.67234 117.43990 118.29647 117.83858
## [2,] 115.95580 112.96006 115.88138 115.81268 116.03332 115.92606
## [3,]  99.06648  99.70322  98.63752  98.29080  99.58629  98.88768
## [4,] 116.52969 116.35816 115.72450 114.99319 117.38311 116.20629
## [5,] 108.82392 105.33943 108.47259 108.15466 109.19777 108.68265
## [6,]  96.05928  93.83312  96.43660  96.75179  95.61977  96.21490
```

2.5

```r
title = c("Epanechnikov","Gaussian","Biweight","Triweight","Uniform","Tricube")
par(mfrow=c(3,2))
for (i in 1:6){
  #plot points firstly
  plot(x, y, main=title[i])
  #add lines
  lines(x[order(x)], matrix1[order(x), i])
}
```



Then we calculate and compare their MSE:

```
AMSE = rep(0, 6)
for (i in 1:6){
  AMSE[i] = sum((matrix1[,i]-y)^2)/409
}
AMSE
```

```
## [1] 285.5042 300.2880 278.8030 272.9163 292.8334 282.7872
```

Then we get the smallest MSE value:

```
min(AMSE)
```

```
## [1] 272.9163
```

The method we use to get best performance is:

```
title[which(AMSE==min(AMSE))]
```

```
## [1] "Triweight"
```

Conclusion: The best performance model is related to triweight method, its mean square error is 272.9163.

Q3.

3.1

We use the code as the assignment provided:

```
library('plot.matrix')
```

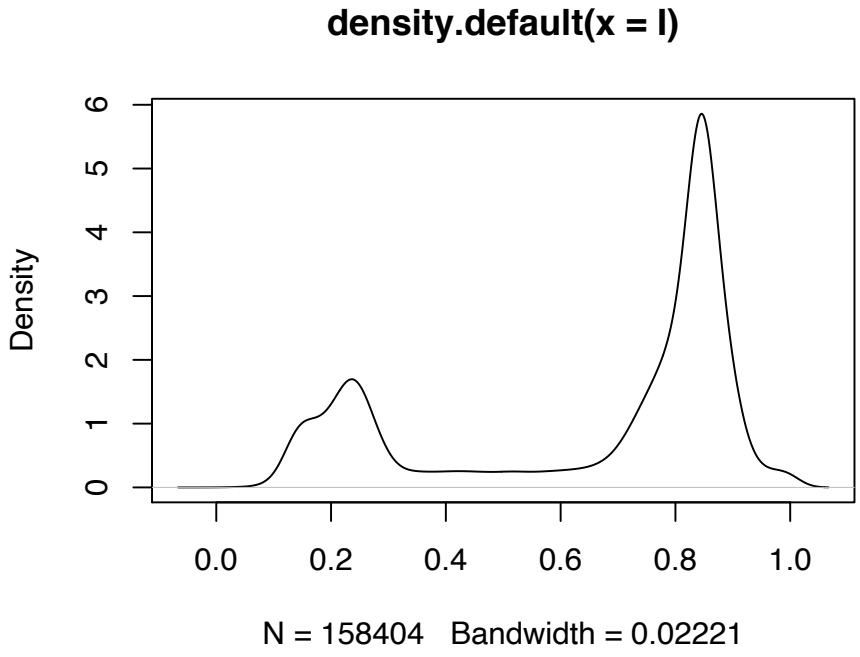
```
## Warning: package 'plot.matrix' was built under R version 4.0.5
```

```
library('png')
library('fields')
```

```
## Loading required package: spam

## Warning: package 'spam' was built under R version 4.0.5

## Spam version 2.8-0 (2022-01-05) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: viridis

## Loading required package: viridisLite

##
## Try help(fields) to get started.
```

```
##
## Attaching package: 'fields'

## The following object is masked from 'package:lava':
##
##     surface
```

```
I <- readPNG('/Users/apple/Desktop/MAST90083/CM-3.png')
I = I[ , , 1]
I = t(apply(I, 2, rev))
par(mfrow=c(2, 1))
image(I, col = gray((0:255)/255))
plot(density(I))
```



3.2

We convert I into a one dimensional vector

```
vec = as.vector(I)
```

3.3

We make the function at first

```
EM <- function(I, mean, var, prob){
  len = length(I)

  norm1 = (1/(sqrt(2*pi)*var[1]))*exp(-0.5*(I-mean[1])^2/(var[1])^2)
  norm2 = (1/(sqrt(2*pi)*var[2]))*exp(-0.5*(I-mean[2])^2/(var[2])^2)
  norm3 = (1/(sqrt(2*pi)*var[3]))*exp(-0.5*(I-mean[3])^2/(var[3])^2)

  denom = c()
  percent1 = c()
  percent2 = c()
  percent3 = c()
  error = 0
  #error[1] = 0
  stop = 1
  #loop = 1

  for (i in 1:len){
    denom[i] = prob[1]*norm1[i] + prob[2]*norm2[i] + prob[3]*norm3[i]
    percent1[i] = prob[1]*norm1[i]/denom[i]
    percent2[i] = prob[2]*norm2[i]/denom[i]
    percent3[i] = prob[3]*norm3[i]/denom[i]
  }

  sum_m = sum(mean)
  sum_v = sum(var)

  while (abs(stop) >= 10^(-6)){
    # initialize the output value
    m = c()
    c = c()
    p = c()

    # get the mean value for three normal distributions
    m[1] = sum(I*percent1)/sum(percent1)
    m[2] = sum(I*percent2)/sum(percent2)
    m[3] = sum(I*percent3)/sum(percent3)

    # get the variance for three normal distribution
    c[1] = sqrt(sum(percent1*(I-m[1])^2) / sum(percent1))
    c[2] = sqrt(sum(percent2*(I-m[2])^2) / sum(percent2))
    c[3] = sqrt(sum(percent3*(I-m[3])^2) / sum(percent3))

    # get the probability parameter for three normal distribution
    p[1] = sum(percent1)/len
    p[2] = sum(percent2)/len
    p[3] = sum(percent3)/len
```

```
    # the normal values we get after this loop
    norm1 = (1/(sqrt(2*pi)*c[1]))*exp(-0.5*(I-m[1])^2/(c[1])^2)
    norm2 = (1/(sqrt(2*pi)*c[2]))*exp(-0.5*(I-m[2])^2/(c[2])^2)
    norm3 = (1/(sqrt(2*pi)*c[3]))*exp(-0.5*(I-m[3])^2/(c[3])^2)

    # print the result
    print(round(c(m,c,p), 4))

    # get the new percentage in this loop using the normal value just calculated
    for (i in 1:len){
      denom[i] = p[1]*norm1[i] + p[2]*norm2[i] + p[3]*norm3[i]
      percent1[i] = p[1]*norm1[i]/denom[i]
      percent2[i] = p[2]*norm2[i]/denom[i]
      percent3[i] = p[3]*norm3[i]/denom[i]
    }

    # the criteria difference calculation
    nor_md = sum(m) - sum_m
    nor_vd = sum(c) - sum_v
    error_new = nor_md + nor_vd
    stop = error_new - error

    # renew them
    sum_m = sum(m)
    sum_v = sum(c)
    error = error_new
  }
}
```

Then we put parameters in this function, for the mean value, we can see that there are three peaks in the plot, which are around 0.16, 0.22, 0.85, so the mean value can be (0.16, 0.22, 0.85); for variance value, we can see that peaks in 0.22 is more wider, so we assume the variance input is (0.05, 0.20, 0.05); for initial probability value, we assume the probabilities are the same, so it is (1/3, 1/3, 1/3). Then we put them into this function to get the output

```
EM(vec, c(0.15,0.25,0.85), c(0.05,0.20,0.03), c(1/3,1/3,1/3))
```

```
## [1] 0.1911 0.4833 0.8430 0.0440 0.2314 0.0396 0.1284 0.2858 0.5858
## [1] 0.2027 0.5250 0.8402 0.0448 0.2403 0.0416 0.1589 0.2613 0.5798
## [1] 0.2083 0.5581 0.8390 0.0465 0.2364 0.0424 0.1842 0.2361 0.5796
## [1] 0.2108 0.5835 0.8385 0.0482 0.2281 0.0428 0.2011 0.2188 0.5802
## [1] 0.2120 0.6023 0.8383 0.0497 0.2191 0.0430 0.2119 0.2080 0.5801
## [1] 0.2127 0.6164 0.8382 0.0507 0.2111 0.0431 0.2191 0.2016 0.5793
## [1] 0.2133 0.6272 0.8383 0.0516 0.2045 0.0430 0.2240 0.1981 0.5779
## [1] 0.2138 0.6356 0.8384 0.0522 0.1994 0.0429 0.2275 0.1964 0.5761
## [1] 0.2143 0.6424 0.8386 0.0528 0.1953 0.0427 0.2301 0.1959 0.5740
## [1] 0.2146 0.6479 0.8387 0.0532 0.1921 0.0424 0.2320 0.1962 0.5717
## [1] 0.2149 0.6526 0.8389 0.0535 0.1896 0.0422 0.2335 0.1971 0.5694
## [1] 0.2152 0.6565 0.8390 0.0538 0.1876 0.0419 0.2346 0.1983 0.5671
## [1] 0.2154 0.6600 0.8391 0.0540 0.1859 0.0416 0.2355 0.1997 0.5648
## [1] 0.2156 0.6630 0.8393 0.0542 0.1845 0.0414 0.2362 0.2013 0.5625
## [1] 0.2158 0.6657 0.8394 0.0544 0.1833 0.0411 0.2368 0.2028 0.5603
## [1] 0.2159 0.6681 0.8395 0.0546 0.1822 0.0408 0.2373 0.2044 0.5582
```

```
## [1] 0.2160 0.6702 0.8397 0.0547 0.1813 0.0406 0.2378 0.2060 0.5562
## [1] 0.2161 0.6722 0.8398 0.0548 0.1804 0.0404 0.2382 0.2075 0.5543
## [1] 0.2162 0.6740 0.8399 0.0549 0.1796 0.0401 0.2385 0.2090 0.5525
## [1] 0.2163 0.6757 0.8400 0.0550 0.1789 0.0399 0.2388 0.2104 0.5508
## [1] 0.2164 0.6772 0.8401 0.0551 0.1782 0.0397 0.2391 0.2118 0.5491
## [1] 0.2165 0.6786 0.8402 0.0552 0.1776 0.0395 0.2394 0.2130 0.5476
## [1] 0.2166 0.6799 0.8403 0.0552 0.1770 0.0394 0.2396 0.2143 0.5461
## [1] 0.2166 0.6811 0.8404 0.0553 0.1765 0.0392 0.2398 0.2154 0.5447
## [1] 0.2167 0.6822 0.8405 0.0554 0.1760 0.0390 0.2400 0.2165 0.5434
## [1] 0.2168 0.6833 0.8405 0.0554 0.1755 0.0389 0.2402 0.2176 0.5422
## [1] 0.2168 0.6843 0.8406 0.0555 0.1751 0.0387 0.2404 0.2186 0.5410
## [1] 0.2169 0.6852 0.8407 0.0556 0.1746 0.0386 0.2406 0.2195 0.5399
## [1] 0.2169 0.6860 0.8407 0.0556 0.1742 0.0385 0.2407 0.2204 0.5389
## [1] 0.2170 0.6868 0.8408 0.0557 0.1739 0.0384 0.2409 0.2213 0.5379
## [1] 0.2170 0.6876 0.8409 0.0557 0.1735 0.0383 0.2410 0.2221 0.5369
## [1] 0.2171 0.6883 0.8409 0.0557 0.1732 0.0382 0.2411 0.2228 0.5360
## [1] 0.2171 0.6889 0.8410 0.0558 0.1729 0.0381 0.2412 0.2236 0.5352
## [1] 0.2172 0.6895 0.8410 0.0558 0.1726 0.0380 0.2414 0.2242 0.5344
## [1] 0.2172 0.6901 0.8411 0.0559 0.1723 0.0379 0.2415 0.2249 0.5337
## [1] 0.2172 0.6906 0.8411 0.0559 0.1720 0.0378 0.2416 0.2255 0.5330
## [1] 0.2173 0.6911 0.8412 0.0559 0.1718 0.0377 0.2417 0.2261 0.5323
## [1] 0.2173 0.6916 0.8412 0.0560 0.1716 0.0377 0.2417 0.2266 0.5317
## [1] 0.2173 0.6921 0.8412 0.0560 0.1713 0.0376 0.2418 0.2271 0.5311
## [1] 0.2174 0.6925 0.8413 0.0560 0.1711 0.0375 0.2419 0.2276 0.5305
## [1] 0.2174 0.6929 0.8413 0.0561 0.1709 0.0375 0.2420 0.2281 0.5300
## [1] 0.2174 0.6933 0.8413 0.0561 0.1708 0.0374 0.2420 0.2285 0.5294
## [1] 0.2174 0.6936 0.8414 0.0561 0.1706 0.0373 0.2421 0.2289 0.5290
## [1] 0.2175 0.6939 0.8414 0.0561 0.1704 0.0373 0.2422 0.2293 0.5285
## [1] 0.2175 0.6942 0.8414 0.0562 0.1703 0.0372 0.2422 0.2297 0.5281
## [1] 0.2175 0.6945 0.8414 0.0562 0.1701 0.0372 0.2423 0.2300 0.5277
## [1] 0.2175 0.6948 0.8415 0.0562 0.1700 0.0371 0.2423 0.2304 0.5273
## [1] 0.2175 0.6951 0.8415 0.0562 0.1699 0.0371 0.2424 0.2307 0.5269
## [1] 0.2176 0.6953 0.8415 0.0562 0.1697 0.0371 0.2424 0.2310 0.5266
## [1] 0.2176 0.6956 0.8415 0.0563 0.1696 0.0370 0.2425 0.2312 0.5263
## [1] 0.2176 0.6958 0.8415 0.0563 0.1695 0.0370 0.2425 0.2315 0.5260
## [1] 0.2176 0.6960 0.8416 0.0563 0.1694 0.0370 0.2426 0.2318 0.5257
## [1] 0.2176 0.6962 0.8416 0.0563 0.1693 0.0369 0.2426 0.2320 0.5254
## [1] 0.2176 0.6963 0.8416 0.0563 0.1692 0.0369 0.2426 0.2322 0.5251
## [1] 0.2176 0.6965 0.8416 0.0563 0.1691 0.0369 0.2427 0.2324 0.5249
## [1] 0.2177 0.6967 0.8416 0.0563 0.1691 0.0368 0.2427 0.2326 0.5247
## [1] 0.2177 0.6968 0.8416 0.0564 0.1690 0.0368 0.2427 0.2328 0.5245
## [1] 0.2177 0.6970 0.8417 0.0564 0.1689 0.0368 0.2427 0.2330 0.5242
## [1] 0.2177 0.6971 0.8417 0.0564 0.1688 0.0368 0.2428 0.2332 0.5241
## [1] 0.2177 0.6972 0.8417 0.0564 0.1688 0.0368 0.2428 0.2333 0.5239
## [1] 0.2177 0.6974 0.8417 0.0564 0.1687 0.0367 0.2428 0.2335 0.5237
## [1] 0.2177 0.6975 0.8417 0.0564 0.1686 0.0367 0.2428 0.2336 0.5235
## [1] 0.2177 0.6976 0.8417 0.0564 0.1686 0.0367 0.2429 0.2338 0.5234
## [1] 0.2177 0.6977 0.8417 0.0564 0.1685 0.0367 0.2429 0.2339 0.5232
## [1] 0.2177 0.6978 0.8417 0.0564 0.1685 0.0367 0.2429 0.2340 0.5231
## [1] 0.2178 0.6979 0.8417 0.0564 0.1684 0.0367 0.2429 0.2341 0.5230
## [1] 0.2178 0.6980 0.8417 0.0564 0.1684 0.0366 0.2429 0.2342 0.5228
## [1] 0.2178 0.6980 0.8418 0.0564 0.1684 0.0366 0.2430 0.2343 0.5227
## [1] 0.2178 0.6981 0.8418 0.0565 0.1683 0.0366 0.2430 0.2344 0.5226
## [1] 0.2178 0.6982 0.8418 0.0565 0.1683 0.0366 0.2430 0.2345 0.5225
```

```
## [1] 0.2178 0.6983 0.8418 0.0565 0.1682 0.0366 0.2430 0.2346 0.5224
## [1] 0.2178 0.6983 0.8418 0.0565 0.1682 0.0366 0.2430 0.2347 0.5223
## [1] 0.2178 0.6984 0.8418 0.0565 0.1682 0.0366 0.2430 0.2348 0.5222
## [1] 0.2178 0.6985 0.8418 0.0565 0.1682 0.0366 0.2430 0.2348 0.5221
## [1] 0.2178 0.6985 0.8418 0.0565 0.1681 0.0365 0.2430 0.2349 0.5220
## [1] 0.2178 0.6986 0.8418 0.0565 0.1681 0.0365 0.2430 0.2350 0.5220
## [1] 0.2178 0.6986 0.8418 0.0565 0.1681 0.0365 0.2431 0.2350 0.5219
## [1] 0.2178 0.6987 0.8418 0.0565 0.1680 0.0365 0.2431 0.2351 0.5218
## [1] 0.2178 0.6987 0.8418 0.0565 0.1680 0.0365 0.2431 0.2352 0.5218
## [1] 0.2178 0.6987 0.8418 0.0565 0.1680 0.0365 0.2431 0.2352 0.5217
## [1] 0.2178 0.6988 0.8418 0.0565 0.1680 0.0365 0.2431 0.2353 0.5217
## [1] 0.2178 0.6988 0.8418 0.0565 0.1680 0.0365 0.2431 0.2353 0.5216
## [1] 0.2178 0.6988 0.8418 0.0565 0.1679 0.0365 0.2431 0.2353 0.5215
## [1] 0.2178 0.6989 0.8418 0.0565 0.1679 0.0365 0.2431 0.2354 0.5215
## [1] 0.2178 0.6989 0.8418 0.0565 0.1679 0.0365 0.2431 0.2354 0.5215
## [1] 0.2178 0.6989 0.8418 0.0565 0.1679 0.0365 0.2431 0.2355 0.5214
```

3.4

Firstly we use the values for mean, variance and probability which in the last output line of section 3.3

```
mean = c(0.2178, 0.6989, 0.8418)
var = c(0.0565, 0.1679, 0.0365)
prob = c(0.2431, 0.2355, 0.5214)
```

Then we use dnorm function to generate random values

```
norm1 = dnorm(vec, mean = mean[1], sd = var[1], log = FALSE)
norm2 = dnorm(vec, mean = mean[2], sd = var[2], log = FALSE)
norm3 = dnorm(vec, mean = mean[3], sd = var[3], log = FALSE)
```

In (i) step, from figure 3a we see that the mean of blue color must be 0.2178 (value=0), the mean of green color must be 0.8418 (value=4), the mean of red color must be 0.6989 (value=8).

Then we do (i) and (ii) steps in this loop

```
pre_i = c()
pre_ii = c()

for (i in 1:length(vec)){
  denom = prob[1]*norm1[i] + prob[2]*norm2[i] + prob[3]*norm3[i]
  percent1 = prob[1]*norm1[i]/denom
  percent2 = prob[2]*norm2[i]/denom
  percent3 = prob[3]*norm3[i]/denom

  #(i)
  if (max(percent1, percent2, percent3)==percent1){
    pre_i[i] = 0 #blue
  }
  if (max(percent1, percent2, percent3)==percent2){
    pre_i[i] = 8 #red
  }
  if (max(percent1, percent2, percent3)==percent3){
```

```
    pre_i[i] = 4 #green
  }
  #(ii)
  pre_ii[i] = percent1*mean[1] + percent2*mean[2] + percent3*mean[3]
}
```

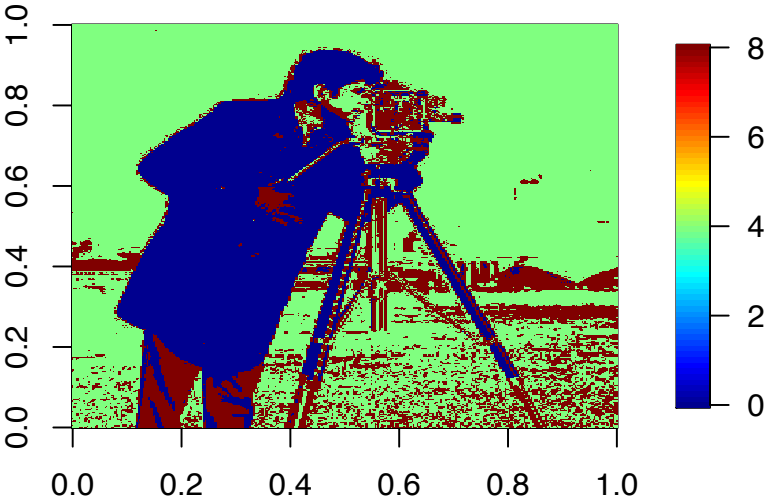
Then they can be converted into matrix, in order to draw them in the last question.

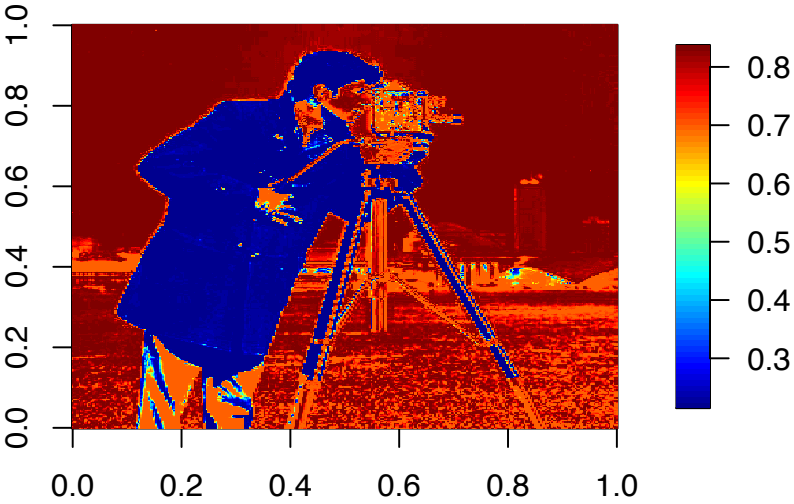
```
pre_i_matrix = matrix(pre_i, ncol=398, nrow=398)
pre_ii_matrix = matrix(pre_ii, ncol=398, nrow=398)
```

3.5

We draw the each pixel's pdf estimation graph in the first line, then we draw the posterior pdf in the second line.

```
par(mfrow=c(2,1))
image.plot(pre_i_matrix)
image.plot(pre_ii_matrix)
```


0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8


0.0
0.2
0.4
0.6
0.8
1.0
0.3
0.4
0.5
0.6
0.7
0.8