MAST90083 Assignment2

Youran Zhou

2021/10/9

```
library(HRW)
```

```
## Warning: package 'HRW' was built under R version 4.0.5
```

```
library(splines)
library(pracma)
```

```
## Warning: package 'pracma' was built under R version 4.0.5
```

```
library(SpatialExtremes)
```

```
## Warning: package 'SpatialExtremes' was built under R version 4.0.5
```

```
##
## Attaching package: 'SpatialExtremes'
```

```
## The following objects are masked from 'package:pracma':
##
##     kriging, logit
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.5
```

```
library("plot.matrix")
```

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

```
library("png")
library("fields")
```

```
## Warning: package 'fields' was built under R version 4.0.5
```

```
## Loading required package: spam
```

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

```
## Loading required package: dotCall64
```

```
## Warning: package 'dotCall64' was built under R version 4.0.5
```

```
## Loading required package: grid
```

```
## Spam version 2.7-0 (2021-06-25) 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
```

```
## Warning: package 'viridis' was built under R version 4.0.5
```

```
## Loading required package: viridisLite
```

```
## Warning: package 'viridisLite' was built under R version 4.0.5
```

```
## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code
```

```
data(WarsawApts)
head(WarsawApts)
```

```
##   surface    district n.rooms floor construction.date areaPerMzloty
## 1      20 Srodmiescie       1     7              1970      95.23810
## 2      27        Wola       1     1              1962     117.39845
## 3      28     Mokotow       1     1              1950     114.28571
## 4      28     Mokotow       2     4              1968     114.28571
## 5      30 Srodmiescie       1     1              1952      93.75586
## 6      32     Mokotow       2     3              2007      81.63265
```

```
WarsawApts <- WarsawApts[order(WarsawApts$construction.date),]

n = dim(WarsawApts)[1]
```

Question 1

(1)

```
x = WarsawApts$construction.date

y = WarsawApts$areaPerMzloty


x_unique = unique(x)

a = seq(0,1,length=22)[2:21]

#(k = quantile(x,a))
(k = quantile(x_unique,a))
```

```
## 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
```

(2)

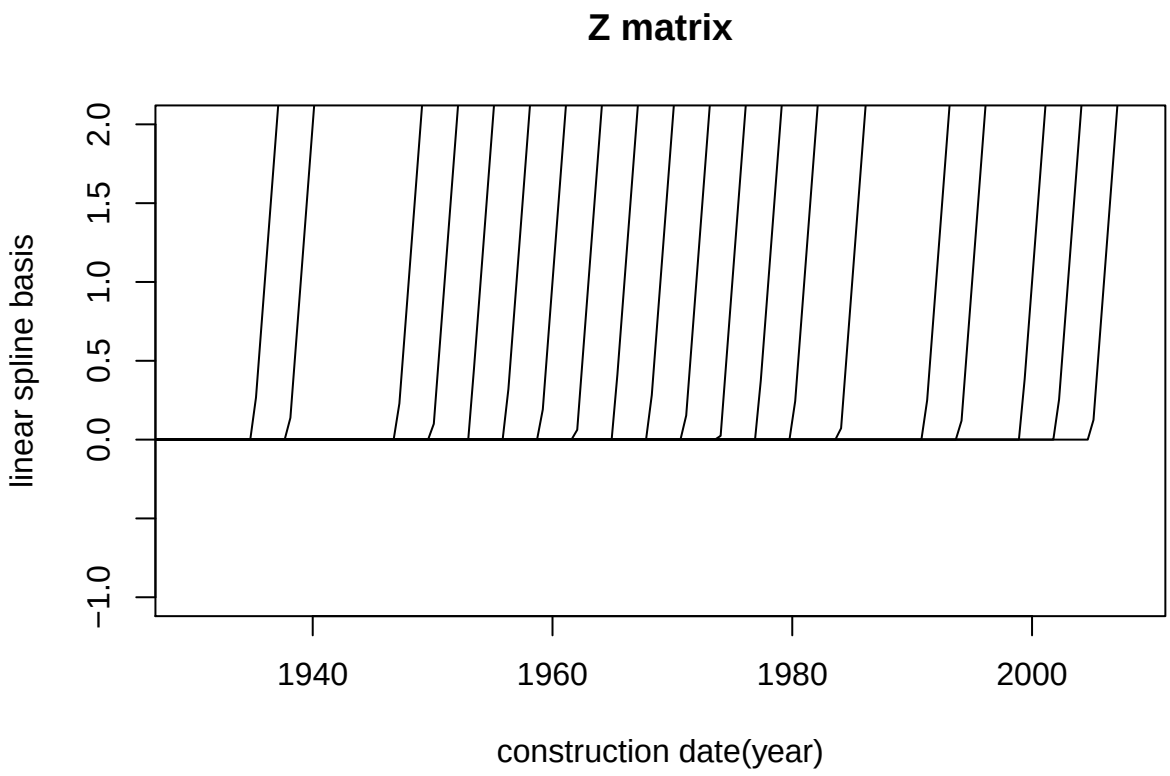
```
Splus = function(x, knot){

  result_matrix = matrix(0, length(x), length(knot))

  for (i in 1:length(knot)){

    for (j in 1:length(x)){

      result = x[j] - knot[i]


      if (result <= 0){
          result = 0
      }

      result_matrix[j,i] = result

    }

  }
  return(result_matrix)

}
```

Z is a matrix contains 20 basis function.

$$Z = \begin{bmatrix} (x-k_1)_+ \\ (x-k_2)_+ \\ \vdots \\ (x-k_{20})_+ \end{bmatrix}$$

```
x_range = seq(min(x)-200,max(x)+200,length=1000)

plot_Z = Splus(x_range,k)
```

```
plot(x_range,plot_Z[,1],typ = "l",ylim = c(-1,2),xlim = c(min(x),max(x)),main = "Z matrix",xlab = "cons

for (i in 2:dim(plot_Z)[2]){
    lines(x_range,plot_Z[,i])

}
```



(3)

```
Z = Splus(x,k)
ones = rep(1,n)

C = cbind(ones,x,Z)

D = diag(1, length(k)+2,length(k)+2)
D[1,1] = 0
D[2,2] = 0

lambda = seq(0,50,length=100)
```

```
fit_spline = function(D,C,y,lambda){
  fit_y= matrix(0, nrow = length(y), ncol = length(lambda))
  fit_df = c()
  fit_RSS = c()
  fit_GCV = c()
  for (i in 1:length(lambda)){

    inver = solve(t(C) %*% C + lambda[i] * D)

    predict_y = C%*% inver %*% t(C) %*% y

    df = sum(diag(inver %*% t(C) %*% C))
```

```
    RSS = sum((predict_y - y)^2)

    GCV = RSS / (1- df/n)^2


    fit_y[,i] = predict_y
    fit_df = c(fit_df, df)
    fit_RSS = c(fit_RSS, RSS)
    fit_GCV = c(fit_GCV, GCV)

  }


  return(list(fit_y = fit_y,fit_df = fit_df, fit_RSS = fit_RSS, fit_GCV = fit_GCV))

}
```

```
fitted = fit_spline(D,C,y,lambda)
```

(4)

```
fitted$fit_RSS
```

```
##   [1] 119307.9 119359.8 119475.7 119620.3 119776.3 119934.8 120091.3 120243.5
##   [9] 120390.4 120531.5 120666.9 120796.7 120921.1 121040.5 121155.1 121265.3
##  [17] 121371.4 121473.6 121572.2 121667.4 121759.5 121848.7 121935.1 122019.0
##  [25] 122100.4 122179.5 122256.5 122331.5 122404.5 122475.7 122545.1 122613.0
##  [33] 122679.2 122744.0 122807.4 122869.4 122930.1 122989.5 123047.8 123104.9
##  [41] 123161.0 123216.0 123270.0 123323.0 123375.0 123426.2 123476.5 123526.0
##  [49] 123574.6 123622.5 123669.6 123716.0 123761.6 123806.6 123850.9 123894.5
##  [57] 123937.5 123979.9 124021.8 124063.0 124103.7 124143.8 124183.4 124222.5
##  [65] 124261.0 124299.1 124336.7 124373.9 124410.5 124446.8 124482.6 124518.0
##  [73] 124552.9 124587.5 124621.7 124655.4 124688.8 124721.9 124754.6 124786.9
##  [81] 124818.9 124850.5 124881.8 124912.8 124943.5 124973.8 125003.9 125033.6
##  [89] 125063.1 125092.3 125121.1 125149.8 125178.1 125206.2 125234.0 125261.5
##  [97] 125288.9 125315.9 125342.7 125369.3
```

```
fitted$fit_df
```

```
##   [1] 22.00000 21.28811 20.68134 20.15438 19.68998 19.27589 18.90308 18.56472
##   [9] 18.25550 17.97123 17.70855 17.46471 17.23744 17.02486 16.82535 16.63758
##  [17] 16.46036 16.29270 16.13373 15.98268 15.83888 15.70174 15.57073 15.44539
##  [25] 15.32530 15.21008 15.09939 14.99293 14.89042 14.79161 14.69628 14.60419
##  [33] 14.51518 14.42906 14.34567 14.26485 14.18648 14.11041 14.03655 13.96476
##  [41] 13.89496 13.82704 13.76092 13.69652 13.63375 13.57255 13.51284 13.45456
##  [49] 13.39766 13.34207 13.28774 13.23462 13.18266 13.13183 13.08207 13.03334
##  [57] 12.98561 12.93885 12.89301 12.84806 12.80398 12.76074 12.71830 12.67664
##  [65] 12.63573 12.59555 12.55608 12.51729 12.47917 12.44169 12.40484 12.36858
##  [73] 12.33292 12.29783 12.26329 12.22929 12.19581 12.16284 12.13037 12.09838
##  [81] 12.06685 12.03579 12.00516 11.97498 11.94521 11.91585 11.88690 11.85834
##  [89] 11.83016 11.80235 11.77491 11.74782 11.72108 11.69468 11.66860 11.64286
##  [97] 11.61742 11.59230 11.56748 11.54296
```

```r
fitted$fit_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
```

(5)

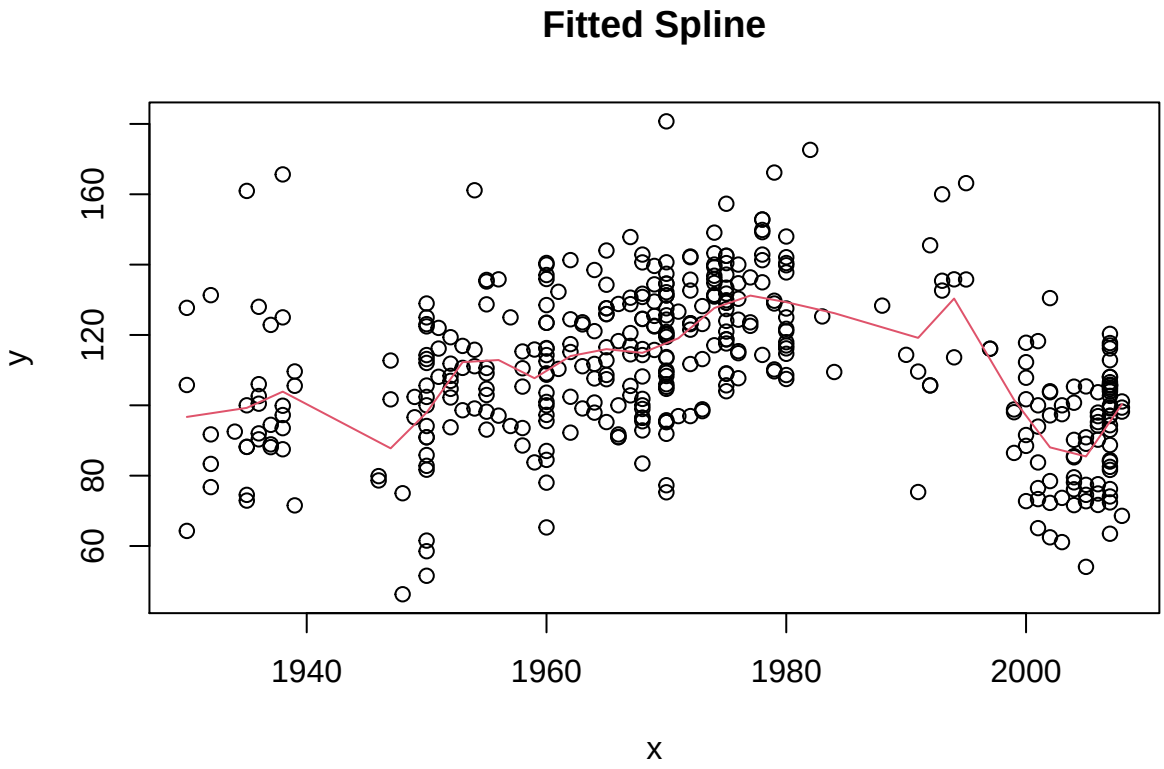
```r
min = which(fitted$fit_GCV == min(fitted$fit_GCV))

(min_lambda = lambda[min])
```

```
## [1] 8.585859
```

```r
(MSE = fitted$fit_RSS[min]/n)
```

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

```r
plot(x,y,xlab = "x",ylab = "y",main = "Fitted Spline")


lines(x,fitted$fit_y[,min],col = 2)
```



60 Basis

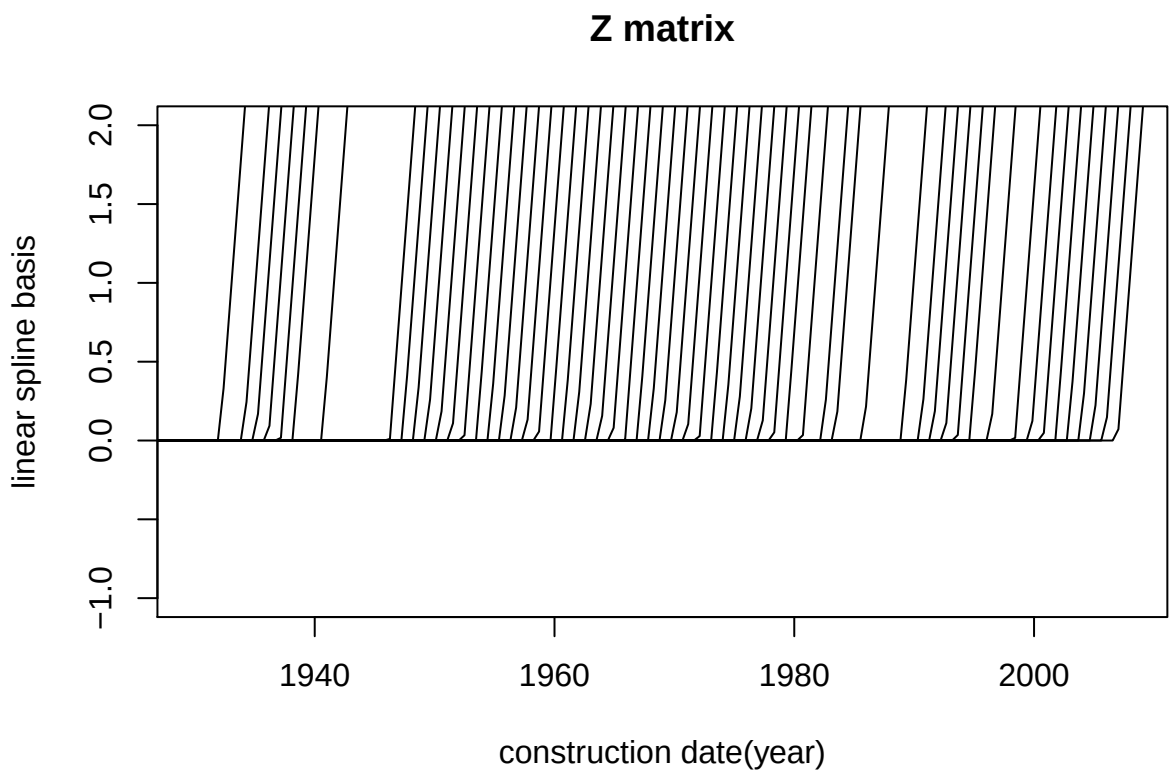
```
a = seq(0,1,length=62)[2:61]

(k_60 = quantile(x_unique,a))
```

```
## 1.639344% 3.278689% 4.918033% 6.557377% 8.196721% 9.836066% 11.47541% 13.11475%
##  1932.066  1934.066  1935.098  1936.131  1937.164  1938.197  1940.607  1946.262
##  14.7541% 16.39344% 18.03279% 19.67213% 21.31148% 22.95082% 24.59016% 26.22951%
##  1947.295  1948.328  1949.361  1950.393  1951.426  1952.459  1953.492  1954.525
## 27.86885%  29.5082% 31.14754% 32.78689% 34.42623% 36.06557% 37.70492% 39.34426%
##  1955.557  1956.590  1957.623  1958.656  1959.689  1960.721  1961.754  1962.787
## 40.98361% 42.62295%  44.2623% 45.90164% 47.54098% 49.18033% 50.81967% 52.45902%
##  1963.820  1964.852  1965.885  1966.918  1967.951  1968.984  1970.016  1971.049
## 54.09836%  55.7377% 57.37705% 59.01639% 60.65574% 62.29508% 63.93443% 65.57377%
##  1972.082  1973.115  1974.148  1975.180  1976.213  1977.246  1978.279  1979.311
## 67.21311% 68.85246%  70.4918% 72.13115% 73.77049% 75.40984% 77.04918% 78.68852%
##  1980.689  1982.377  1983.410  1985.770  1988.951  1990.508  1991.541  1992.574
## 80.32787% 81.96721% 83.60656%  85.2459% 86.88525% 88.52459% 90.16393% 91.80328%
##  1993.607  1994.639  1996.344  1998.410  1999.738  2000.770  2001.803  2002.836
## 93.44262% 95.08197% 96.72131% 98.36066%
##  2003.869  2004.902  2005.934  2006.967
```

```
#(k_60 = quantile(x,a))
```

```
plot_Z_60 = Splus(x_range,k_60)

plot(x_range,plot_Z_60[,1],typ = "l",ylim = c(-1,2),xlim = c(min(x),max(x)),main = "Z matrix",xlab = "c
```

```
for (i in 2:dim(plot_Z_60)[2]){
    lines(x_range,plot_Z_60[,i])

}
```




```
Z_60 = Splus(x,k_60)


C_60 = cbind(ones,x,Z_60)

D_60 = diag(1, length(k_60)+2,length(k_60)+2)
D_60[1,1] = 0
D_60[2,2] = 0
```

```
fitted_60 = fit_spline(D_60,C_60,y,lambda)
```

```
min_60 = which(fitted_60$fit_GCV == min(fitted_60$fit_GCV))

(min_lambda_60 = lambda[min_60])
```

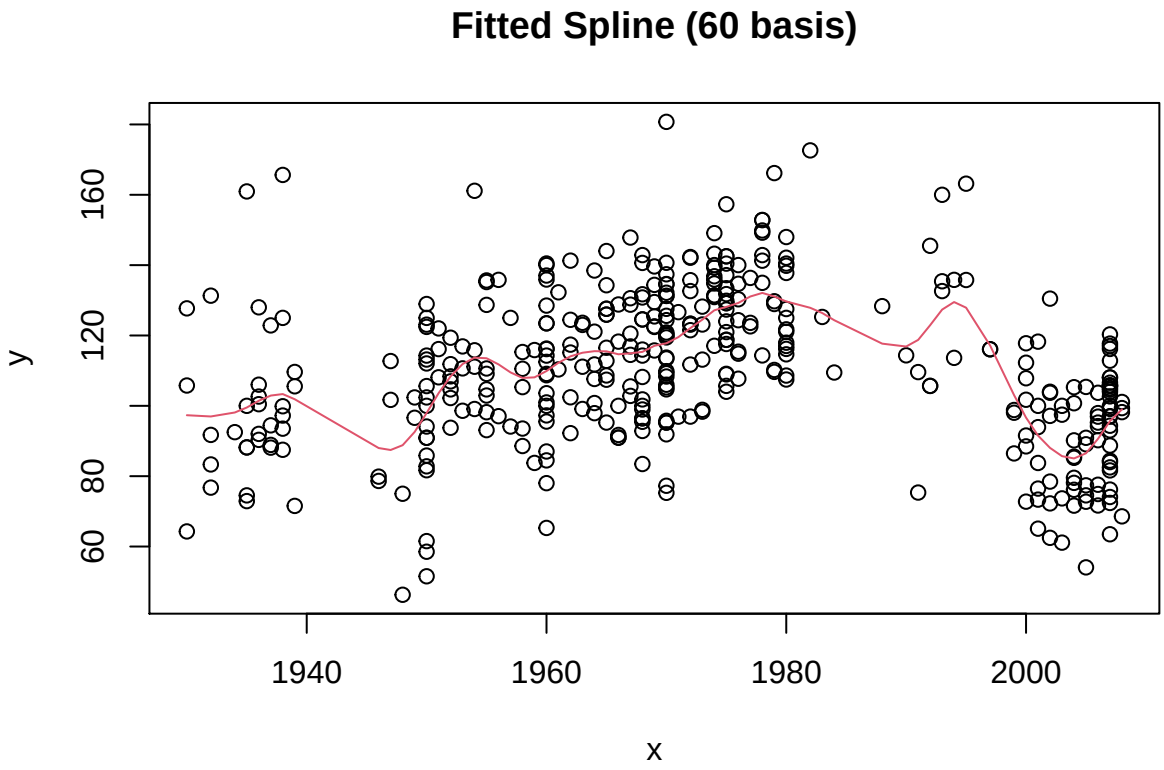
```
## [1] 31.31313
```

```
(MSE_60 = fitted_60$fit_RSS[min_60]/n)
```

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

```
plot(x,y,xlab = "x",ylab = "y",main = "Fitted Spline (60 basis)")
```

```
lines(x,fitted_60$fit_y[,min_lambda_60],col=2)
```



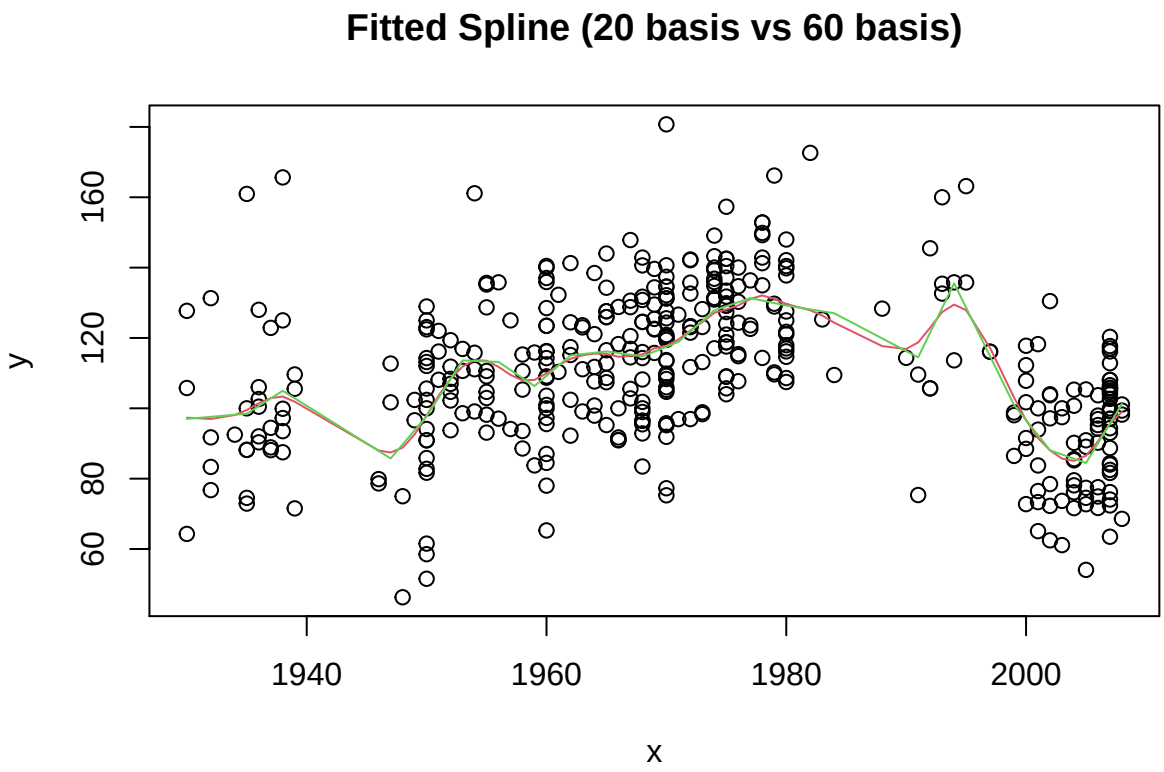

```
(MSE)
```

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

```
(MSE_60)
```

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

```
plot(x,y,xlab = "x",ylab = "y",main = "Fitted Spline (20 basis vs 60 basis)")

lines(x,fitted_60$fit_y[,min_lambda_60],col=2)
lines(x,fitted$fit_y[,min_lambda],col=3)
```



Increasing the number of basis does not help to improve the result.

Question 2

(1)

```
k_list = c(3,6,9,12,15,18)

k_list = c(3,7,11,15,19,23)
```

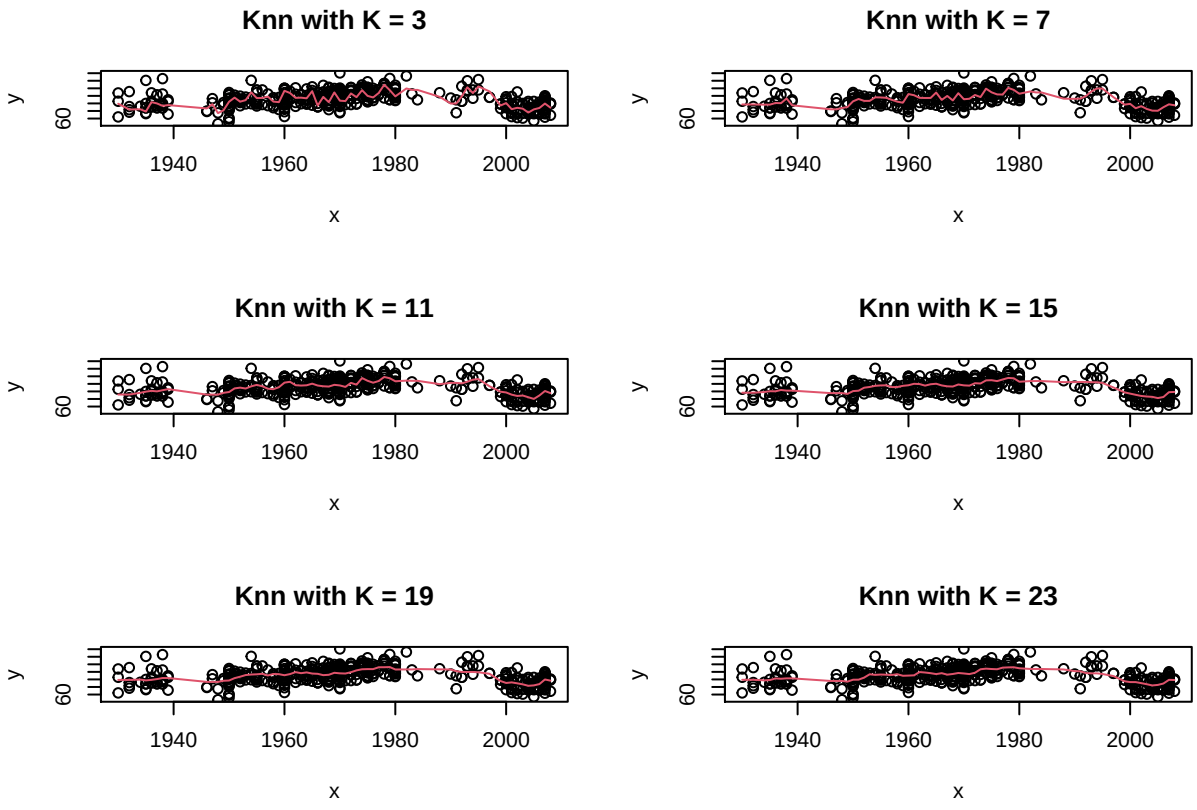
```
KNN = function(k,current_X, X){

  d = abs(current_X - X)

  index = sort(d, index.return=TRUE)$ix[1:k]

  return(mean(y[index]))
}
```

(2)

```
KNN_prediction = matrix(0,length(x),length(k_list))
KNN_MSE = c()
colnames(KNN_prediction) = k_list
```

```
for (i in 1:length(k_list)){
  predict_y = c()
  for (current_x in x){

    predict_y = c(predict_y, KNN(k_list[i],current_x,x))

  }

  KNN_prediction[,i] = predict_y

  KNN_MSE = c(KNN_MSE,sum((predict_y - y)^2)/n)

}
```

```
par(mfrow=c(3,2))

plot(x,y,main = "Knn with K = 3")
lines(x,KNN_prediction[,1],col = 2)

plot(x,y,main = "Knn with K = 7")
lines(x,KNN_prediction[,2],col = 2)

plot(x,y,main = "Knn with K = 11")
lines(x,KNN_prediction[,3],col = 2)

plot(x,y,main = "Knn with K = 15")
lines(x,KNN_prediction[,4],col = 2)

plot(x,y,main = "Knn with K = 19")
lines(x,KNN_prediction[,5],col = 2)

plot(x,y,main = "Knn with K = 23")
lines(x,KNN_prediction[,6],col = 2)
```




```
rbind(k_list,KNN_MSE)
```

```
##             [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## k_list    3.0000   7.0000  11.0000  15.0000  19.0000  23.0000
## KNN_MSE 333.3363 293.9181 297.3904 300.3149 302.7898 309.1673
```

When K = 7, we have the minimum MSE. Overall there is not a significant difference between using KNN and spline, but we can choose the different values of K to get slightly different results. Choosing an approximate K value could help us to get a better result. From the result we could say as the K value increases, the result is getting worse. However, the K value can not be too small since it will easily be affected by noisy data.

(3)

```
accommodate_Kernels = function(x){
  if (abs(x) < 1) {

    Epanechnikov = 3/4*(1-x^2)
    Biweight     = 15/16*(1-x^2)^2
    Triweight    = 35/32*(1-x^2)^3
    Uniform      = 1/2
    Tricube      = 70/81*(1-(abs(x))^3)^3

  }else{

    Epanechnikov = 0
    Biweight     = 0
```

```
    Triweight    = 0
    Uniform      = 0
    Tricube      = 0
  }

    Gaussian = (2*pi)^(-1/2)*exp(-x^2/2)

    return(c(Epanechnikov,Gaussian,Biweight,Triweight, Uniform, Tricube))
}
```

(4)

```
h = 2
K_h = function(x_i,x_j){
  return((x_j - x_i)/h )
}
```

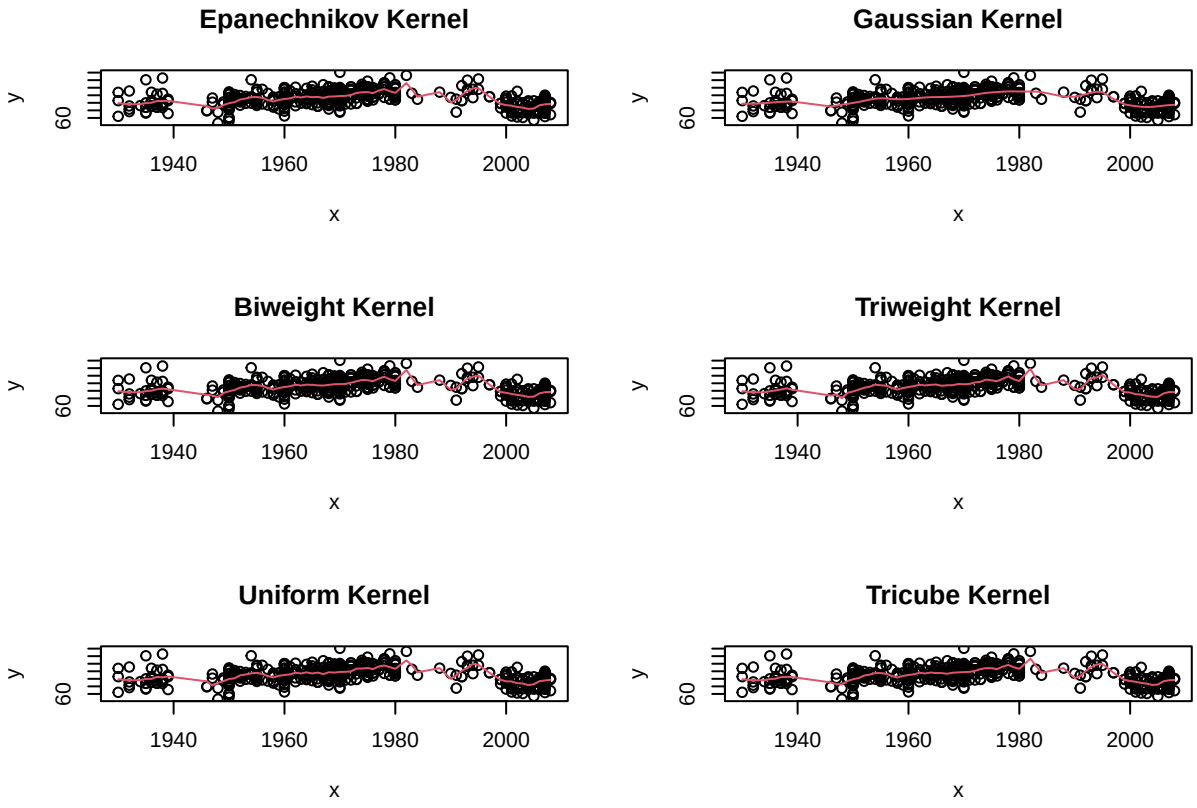
```
Kernel_predict = matrix(0,length(x),6)

colnames(Kernel_predict) <- c("Epanechnikov","Gaussian","Biweight","Triweight","Uniform","Tricube")

for (i in 1:length(x)){
  Kernel_weight = matrix(0,length(x),6)
  numerator = 0

  for (j in 1:length(x)){

    value =  K_h(x[i],x[j])
    weight = accommodate_Kernels(value)
    numerator = numerator + (weight * y[j])
    Kernel_weight[j,] = accommodate_Kernels(value)
  }
  predict_y = numerator / colSums(Kernel_weight)


  Kernel_predict[i,] = predict_y

}
```

```
Kernel_predict[1:10,]
```

```
##       Epanechnikov  Gaussian Biweight Triweight  Uniform  Tricube
##  [1,]     99.25909  97.64359 99.25909  99.25909 99.25909 99.25909
##  [2,]     99.25909  97.64359 99.25909  99.25909 99.25909 99.25909
##  [3,]     99.25909  97.64359 99.25909  99.25909 99.25909 99.25909
##  [4,]     95.78344  97.39443 95.78344  95.78344 95.78344 95.78344
##  [5,]     95.78344  97.39443 95.78344  95.78344 95.78344 95.78344
##  [6,]     95.78344  97.39443 95.78344  95.78344 95.78344 95.78344
##  [7,]     95.78344  97.39443 95.78344  95.78344 95.78344 95.78344
##  [8,]     96.56033  99.14823 96.32920  96.05920 96.75294 96.47429
##  [9,]     99.44371 100.37180 99.14416  98.86186 99.74902 99.32478
## [10,]     99.44371 100.37180 99.14416  98.86186 99.74902 99.32478
```

(5)

```
par(mfrow=c(3,2))

plot(x,y,main = "Epanechnikov Kernel")
lines(x,Kernel_predict[,1],col = 2)

plot(x,y,main = "Gaussian Kernel")
lines(x,Kernel_predict[,2],col = 2)

plot(x,y,main = "Biweight Kernel")
lines(x,Kernel_predict[,3],col = 2)

plot(x,y,main = "Triweight Kernel")
lines(x,Kernel_predict[,4],col = 2)

plot(x,y,main = "Uniform Kernel")
lines(x,Kernel_predict[,5],col = 2)

plot(x,y,main = "Tricube Kernel")
lines(x,Kernel_predict[,6],col = 2)
```




```
(Kernel_MSE = colSums((Kernel_predict - y)^2)/n)
```

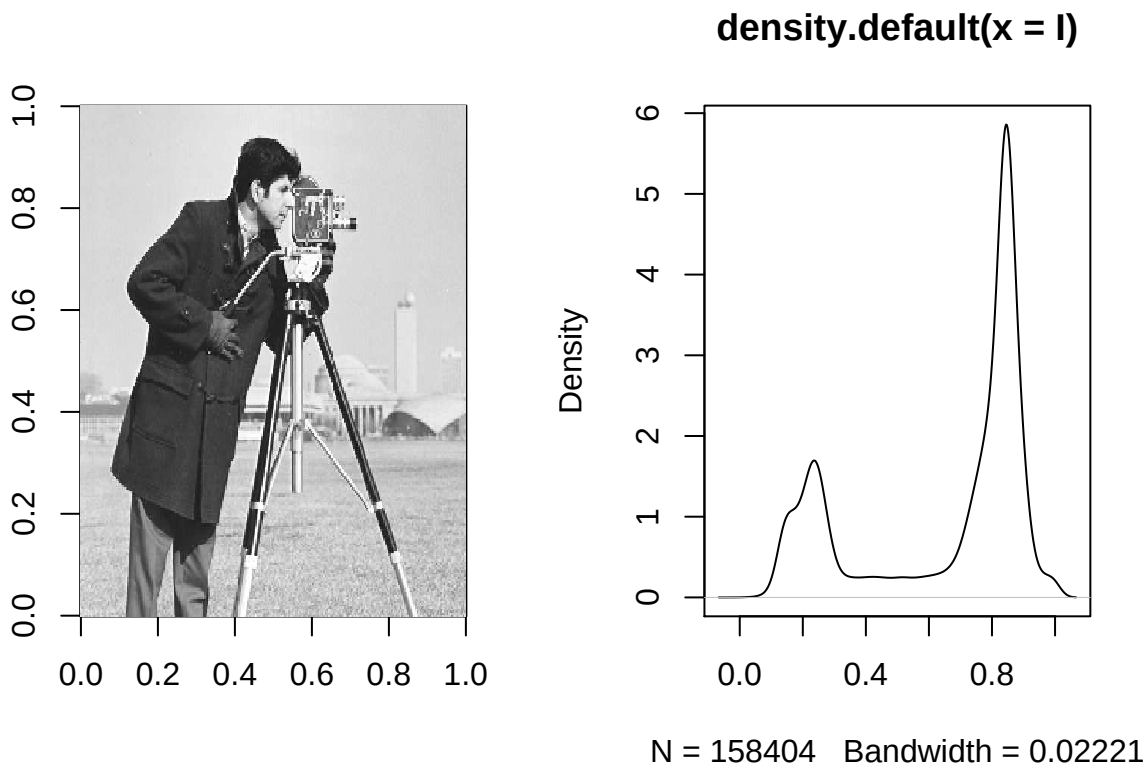
```
## Epanechnikov     Gaussian     Biweight    Triweight      Uniform      Tricube
##     285.5042     300.2880     278.8030     272.9163     292.8334     282.7872
```

Triweight Kernel could provide the minimum MSE.

Question 3

(1)

```
I = readPNG("CM.png")
I = I[,,1] # only first channel
I = t(apply(I, 2, rev))
par(mfrow = c(1,2))
image(I,col = gray((0:255)/255))
plot(density(I))
```



(2)

```
lappend <- function (lst, ...){
lst <- c(lst, list(...))
  return(lst)
}
```

```
# The source code comes from MAST30027, edit by me
EM_algo <- function(X, p, mu,sigma, epsilon=1e-6) {
```

```
  m_list = list(mu)
  c_list = list(sigma)
  e_list = list(0)


  criteria = 1


  n = 1


  while((criteria > epsilon)){


    n = n + 1

    result = EM_step(X, p, mu, sigma)

    p = result$w.new
    mu = result$mu.new
    sigma = result$sigma.new

    m_list = lappend(m_list,mu)
    c_list = lappend(c_list,sigma)



    e = sum(m_list[n][[1]] - m_list[n-1][[1]]) +
        sum(c_list[n][[1]] - c_list[n-1][[1]])



    e_list = lappend(e_list,e)

    criteria = abs(e - e_list[n-1][[1]])



    print(round(c(mu,sigma,p),4))

  }

  return(list(mu=mu, sigma = sigma, p = p))
}
```

```
EM_step <- function(X, p, mu,sigma) {


  # E-step:

  prob.y_z = Prob.y_z(X, p, mu,sigma)

  # compute posterior P(Z=k|y)
  posterior = prob.y_z / rowSums(prob.y_z)
```

```
  # M-step
  w.new = colSums(posterior)/sum(posterior)

  mu.new = colSums(posterior*X)/colSums(posterior)

  sigma.new = update_sigma(posterior,mu.new,X)


  return(list(w.new=w.new, mu.new=mu.new, sigma.new = sigma.new))
}
```

```
# compute P(y, Z=k)
Prob.y_z <- function(X, p, mu,sigma) {

  L = matrix(NA, nrow=length(X), ncol= length(p))
  for(k in seq_len(ncol(L))) {

    L[, k] = dnorm(X, mean=mu[k], sd=sigma[k])*p[k]
  }

  return(L)
}
```

```
update_sigma = function(p,mu,X){


  mu_matrix = t(replicate(length(X),mu))
  X_matrix = replicate(length(mu),X)
  update_P = p * (X_matrix - mu_matrix)^2

  return(sqrt(colSums(update_P)/colSums(p)))

}
```

(3)

```
EM <- EM_algo(sort(c(I)),
              p=c(0.3,0.3, 0.4),
              mu=c(0, 0.5, 1),
              sigma=c(0.1, 0.1, 0.1),
              epsilon=1e-6)
```

```
## [1] 0.1937 0.5001 0.8436 0.0449 0.1980 0.0472 0.1676 0.2277 0.6048
## [1] 0.2052 0.5208 0.8392 0.0466 0.2139 0.0476 0.1870 0.2017 0.6114
## [1] 0.2099 0.5455 0.8378 0.0482 0.2166 0.0476 0.2015 0.1875 0.6111
## [1] 0.2117 0.5664 0.8373 0.0495 0.2139 0.0474 0.2110 0.1803 0.6087
## [1] 0.2126 0.5831 0.8372 0.0506 0.2096 0.0470 0.2173 0.1771 0.6056
## [1] 0.2132 0.5963 0.8373 0.0513 0.2055 0.0466 0.2217 0.1760 0.6022
## [1] 0.2136 0.6071 0.8374 0.0519 0.2019 0.0461 0.2249 0.1764 0.5987
## [1] 0.2139 0.6162 0.8376 0.0523 0.1990 0.0457 0.2274 0.1776 0.5951
## [1] 0.2142 0.6238 0.8378 0.0527 0.1965 0.0452 0.2292 0.1793 0.5915
## [1] 0.2145 0.6304 0.8379 0.0530 0.1944 0.0447 0.2307 0.1814 0.5879
## [1] 0.2147 0.6362 0.8381 0.0533 0.1926 0.0443 0.2319 0.1836 0.5845
```

```
## [1] 0.2149 0.6413 0.8383 0.0535 0.1910 0.0438 0.2329 0.1860 0.5811
## [1] 0.2151 0.6459 0.8384 0.0537 0.1896 0.0434 0.2337 0.1883 0.5779
## [1] 0.2152 0.6499 0.8386 0.0538 0.1883 0.0430 0.2344 0.1907 0.5749
## [1] 0.2154 0.6536 0.8388 0.0540 0.1871 0.0426 0.2351 0.1930 0.5719
## [1] 0.2155 0.6569 0.8389 0.0541 0.1860 0.0422 0.2357 0.1952 0.5691
## [1] 0.2156 0.6599 0.8391 0.0543 0.1849 0.0419 0.2362 0.1973 0.5665
## [1] 0.2157 0.6627 0.8392 0.0544 0.1840 0.0416 0.2366 0.1994 0.5640
## [1] 0.2159 0.6652 0.8393 0.0545 0.1830 0.0413 0.2371 0.2013 0.5616
## [1] 0.2160 0.6675 0.8395 0.0546 0.1822 0.0410 0.2375 0.2032 0.5594
## [1] 0.2161 0.6696 0.8396 0.0547 0.1813 0.0407 0.2378 0.2049 0.5573
## [1] 0.2161 0.6716 0.8397 0.0548 0.1805 0.0405 0.2382 0.2066 0.5553
## [1] 0.2162 0.6734 0.8398 0.0549 0.1798 0.0402 0.2385 0.2082 0.5534
## [1] 0.2163 0.6750 0.8399 0.0550 0.1791 0.0400 0.2388 0.2097 0.5516
## [1] 0.2164 0.6766 0.8401 0.0551 0.1785 0.0398 0.2390 0.2111 0.5499
## [1] 0.2165 0.6780 0.8401 0.0551 0.1778 0.0396 0.2393 0.2124 0.5483
## [1] 0.2166 0.6794 0.8402 0.0552 0.1773 0.0394 0.2395 0.2137 0.5468
## [1] 0.2166 0.6806 0.8403 0.0553 0.1767 0.0393 0.2397 0.2149 0.5454
## [1] 0.2167 0.6818 0.8404 0.0554 0.1762 0.0391 0.2399 0.2160 0.5440
## [1] 0.2167 0.6828 0.8405 0.0554 0.1757 0.0390 0.2401 0.2171 0.5427
## [1] 0.2168 0.6838 0.8406 0.0555 0.1752 0.0388 0.2403 0.2181 0.5415
## [1] 0.2169 0.6848 0.8406 0.0555 0.1748 0.0387 0.2405 0.2191 0.5404
## [1] 0.2169 0.6857 0.8407 0.0556 0.1744 0.0385 0.2406 0.2200 0.5393
## [1] 0.2170 0.6865 0.8408 0.0556 0.1740 0.0384 0.2408 0.2209 0.5383
## [1] 0.2170 0.6872 0.8408 0.0557 0.1737 0.0383 0.2409 0.2217 0.5373
## [1] 0.2171 0.6880 0.8409 0.0557 0.1733 0.0382 0.2411 0.2225 0.5364
## [1] 0.2171 0.6886 0.8409 0.0558 0.1730 0.0381 0.2412 0.2232 0.5356
## [1] 0.2171 0.6893 0.8410 0.0558 0.1727 0.0380 0.2413 0.2239 0.5348
## [1] 0.2172 0.6899 0.8410 0.0559 0.1724 0.0379 0.2414 0.2246 0.5340
## [1] 0.2172 0.6904 0.8411 0.0559 0.1722 0.0378 0.2415 0.2252 0.5333
## [1] 0.2173 0.6909 0.8411 0.0559 0.1719 0.0378 0.2416 0.2258 0.5326
## [1] 0.2173 0.6914 0.8412 0.0560 0.1717 0.0377 0.2417 0.2264 0.5319
## [1] 0.2173 0.6919 0.8412 0.0560 0.1714 0.0376 0.2418 0.2269 0.5313
## [1] 0.2173 0.6923 0.8412 0.0560 0.1712 0.0375 0.2419 0.2274 0.5307
## [1] 0.2174 0.6927 0.8413 0.0561 0.1710 0.0375 0.2419 0.2279 0.5302
## [1] 0.2174 0.6931 0.8413 0.0561 0.1708 0.0374 0.2420 0.2283 0.5297
## [1] 0.2174 0.6935 0.8413 0.0561 0.1707 0.0374 0.2421 0.2287 0.5292
## [1] 0.2175 0.6938 0.8414 0.0561 0.1705 0.0373 0.2421 0.2291 0.5287
## [1] 0.2175 0.6941 0.8414 0.0562 0.1703 0.0373 0.2422 0.2295 0.5283
## [1] 0.2175 0.6944 0.8414 0.0562 0.1702 0.0372 0.2423 0.2299 0.5279
## [1] 0.2175 0.6947 0.8415 0.0562 0.1701 0.0372 0.2423 0.2302 0.5275
## [1] 0.2175 0.6950 0.8415 0.0562 0.1699 0.0371 0.2424 0.2305 0.5271
## [1] 0.2176 0.6952 0.8415 0.0562 0.1698 0.0371 0.2424 0.2308 0.5267
## [1] 0.2176 0.6955 0.8415 0.0562 0.1697 0.0370 0.2425 0.2311 0.5264
## [1] 0.2176 0.6957 0.8415 0.0563 0.1696 0.0370 0.2425 0.2314 0.5261
## [1] 0.2176 0.6959 0.8416 0.0563 0.1695 0.0370 0.2425 0.2317 0.5258
## [1] 0.2176 0.6961 0.8416 0.0563 0.1694 0.0369 0.2426 0.2319 0.5255
## [1] 0.2176 0.6963 0.8416 0.0563 0.1693 0.0369 0.2426 0.2321 0.5253
## [1] 0.2176 0.6964 0.8416 0.0563 0.1692 0.0369 0.2426 0.2323 0.5250
## [1] 0.2177 0.6966 0.8416 0.0563 0.1691 0.0369 0.2427 0.2325 0.5248
## [1] 0.2177 0.6968 0.8416 0.0563 0.1690 0.0368 0.2427 0.2327 0.5246
## [1] 0.2177 0.6969 0.8417 0.0564 0.1689 0.0368 0.2427 0.2329 0.5243
## [1] 0.2177 0.6971 0.8417 0.0564 0.1689 0.0368 0.2428 0.2331 0.5241
## [1] 0.2177 0.6972 0.8417 0.0564 0.1688 0.0368 0.2428 0.2333 0.5239
## [1] 0.2177 0.6973 0.8417 0.0564 0.1687 0.0367 0.2428 0.2334 0.5238
```

```
## [1] 0.2177 0.6974 0.8417 0.0564 0.1687 0.0367 0.2428 0.2336 0.5236
## [1] 0.2177 0.6975 0.8417 0.0564 0.1686 0.0367 0.2429 0.2337 0.5234
## [1] 0.2177 0.6976 0.8417 0.0564 0.1686 0.0367 0.2429 0.2338 0.5233
## [1] 0.2177 0.6977 0.8417 0.0564 0.1685 0.0367 0.2429 0.2340 0.5231
## [1] 0.2177 0.6978 0.8417 0.0564 0.1685 0.0367 0.2429 0.2341 0.5230
## [1] 0.2178 0.6979 0.8417 0.0564 0.1684 0.0366 0.2429 0.2342 0.5229
## [1] 0.2178 0.6980 0.8418 0.0564 0.1684 0.0366 0.2429 0.2343 0.5228
## [1] 0.2178 0.6981 0.8418 0.0564 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 0.6982 0.8418 0.0565 0.1683 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 0.2347 0.5222
## [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 0.1681 0.0366 0.2430 0.2349 0.5221
## [1] 0.2178 0.6985 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 0.6986 0.8418 0.0565 0.1681 0.0365 0.2431 0.2351 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 0.5217
## [1] 0.2178 0.6988 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 0.5216
## [1] 0.2178 0.6988 0.8418 0.0565 0.1680 0.0365 0.2431 0.2353 0.5216
## [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
```

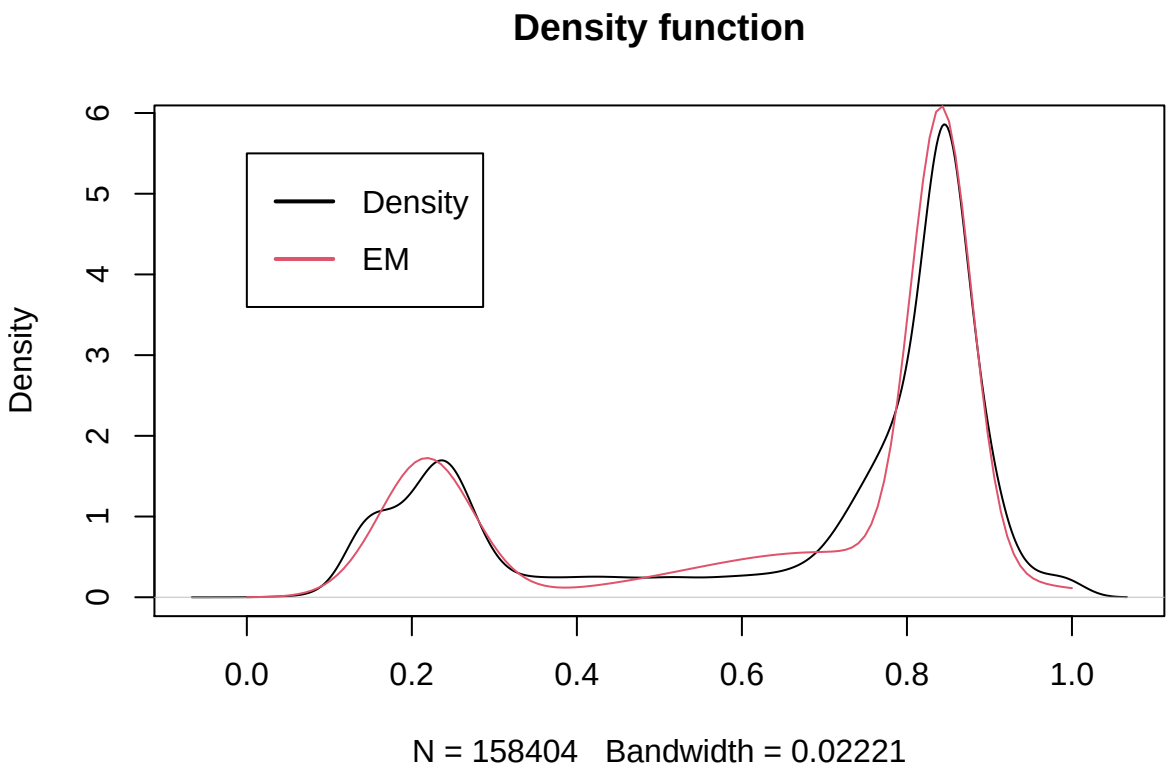
```
EM
```

```
## $mu
## [1] 0.2178312 0.6989283 0.8418349
##
## $sigma
## [1] 0.05651521 0.16790799 0.03648020
##
## $p
## [1] 0.2431212 0.2354512 0.5214276
```

(4)

```
EM_result = c()

for (i in sort(c(I))){

  EM_result = c( EM_result,
       dnorm(i,mean = EM$mu[1],sd = EM$sigma[1]) * EM$p[1] +   dnorm(i,mean = EM$mu[2],sd = EM$sigma[2])
  + dnorm(i,mean = EM$mu[3],sd = EM$sigma[3])*EM$p[3])
}
```

```
plot(density(I), main = "Density function")
lines(sort(c(I)),EM_result,lty = 1,col = 2)
legend(0,5.5,c("Density","EM"), lwd=c(2,2), col=c("black",2), y.intersp=1.5)
```

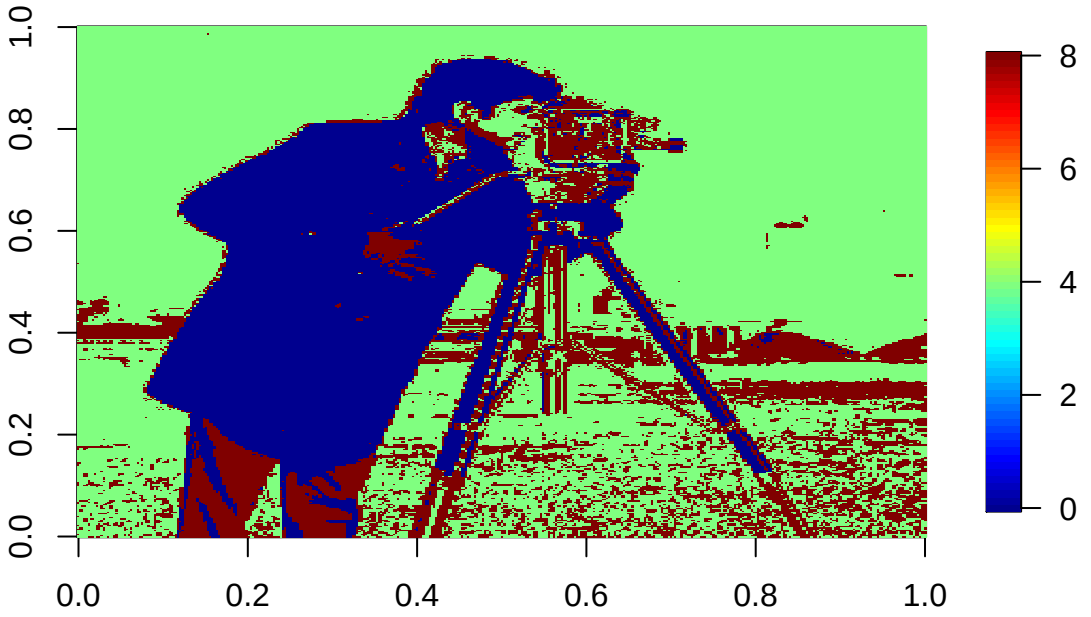



```
z_predict = data.frame(Prob.y_z(c(I), EM$p, EM$mu,EM$sigma))
image1 = apply(z_predict,1,which.max)


image1[image1 == 1] = 0
image1[image1 == 2] = 8
image1[image1 == 3] = 4
```

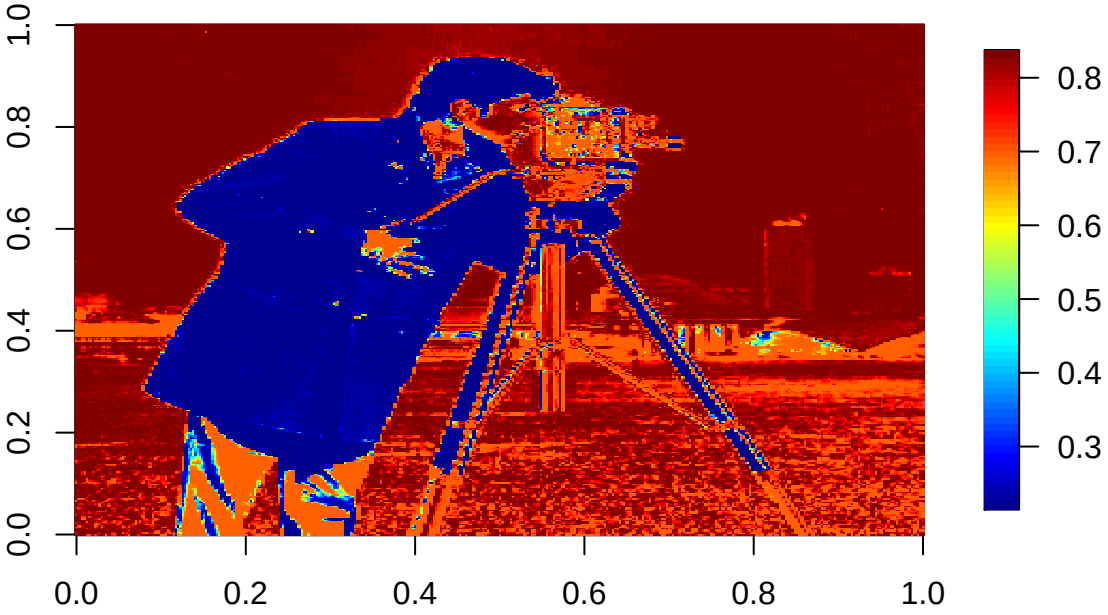
```
posterior = z_predict / rowSums(z_predict)
mu = t(replicate(length(c(I)),EM$mu))

image2 = rowSums(posterior * mu)
```

(5)

```
image.plot(matrix(image1,nrow = 398,ncol = 398))
```




```
image.plot(matrix(image2,nrow = 398,ncol = 398))
```


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