MAST90083 Assignment 2

Haonan Zhong

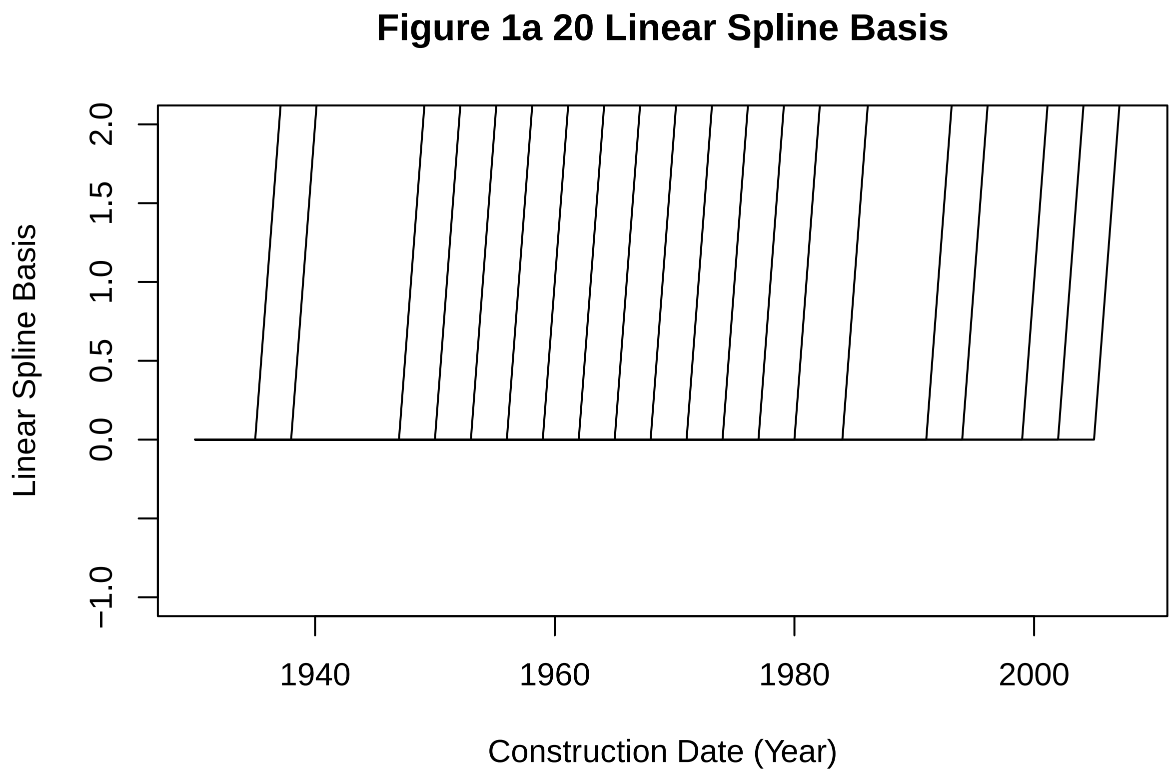
Question 1.1

```
library(HRW)
data(WarsawApts)

x <- WarsawApts$construction.date
y <- WarsawApts$areaPerMzloty

# Exclude extreme values of 0 and 1
probVec <- seq(0, 1, length = 22)[2:21]
k <- quantile(unique(x), probs = probVec)
```

Question 1.2

```
construct_basis <- function(x, knot) {
  z_matrix <- matrix(0, length(x), length(knot))
  for (i in 1:length(knot)) {
    for (j in 1:length(x)) {
      z_matrix[j, i] <-  x[j] - knot[i]
    }
  }
  z_matrix[ z_matrix < 0] <- 0
  return(z_matrix)
}

z <- construct_basis(sort(x), k)
```

```
plot(sort(x), z[,1], type = "l", ylim = c(-1, 2), xlab = "Construction Date (Year)",
     ylab = "Linear Spline Basis", main = "Figure 1a 20 Linear Spline Basis")
for (i in 2:dim(z)[2]) {
  lines(sort(x), z[, i])
}
```



Question 1.3

```
intercept <- rep(1, length(x))
Z <- construct_basis(x, k)
# Construct matrix C
C <- cbind(intercept, x, Z)

# Construct matrix D
D <- diag(1, dim(C)[2], dim(C)[2])
# Replace first two columns with 0s since we don't want to penalise them
D[, c(1, 2)] <- 0

# Tuning parameter
lambda <- seq(0, 50, length = 100)
```

Question 1.4

```
fit_penalised_spline <- function(X, D, lambda, y) {
  RSS_error <- rep(0, length(lambda))
  fitted_df <- rep(0, length(lambda))
  GCV <- rep(0, length(lambda))
  fitted_y <- matrix(0, length(y), length(lambda))
```

```r
  for (i in 1:length(lambda)) {
    betaHat <- solve(t(X) %*% X + lambda[i] * D) %*% t(X) %*% y
    fitted_y[, i] <- X %*% betaHat
    RSS_error[i] <- sum((y - fitted_y[, i])^2)
    fitted_df[i] <- sum(diag(solve(t(X) %*% X + lambda[i] * D) %*% t(X) %*% X))
    GCV[i] = RSS_error[i] / (1 - fitted_df[i]/length(y))^2
  }

  return(list(RSS_error = RSS_error, fitted_y = fitted_y, fitted_df = fitted_df, GCV = GCV))
}
```

```r
# Fit penalised spline regression for each lambda
penalised_spline_20 <- fit_penalised_spline(C, D, lambda, y)

# RSS error associated with 100 tuning parameters
(RSS_error_20 <- penalised_spline_20$RSS_error)
```

```
##   [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
```

```r
# Degrees of freedom associated with 100 tuning parameters
(df_20 <- penalised_spline_20$fitted_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
# Generalised CV associated with 100 tuning parameters
(GCV_20 <- penalised_spline_20$GCV)
```

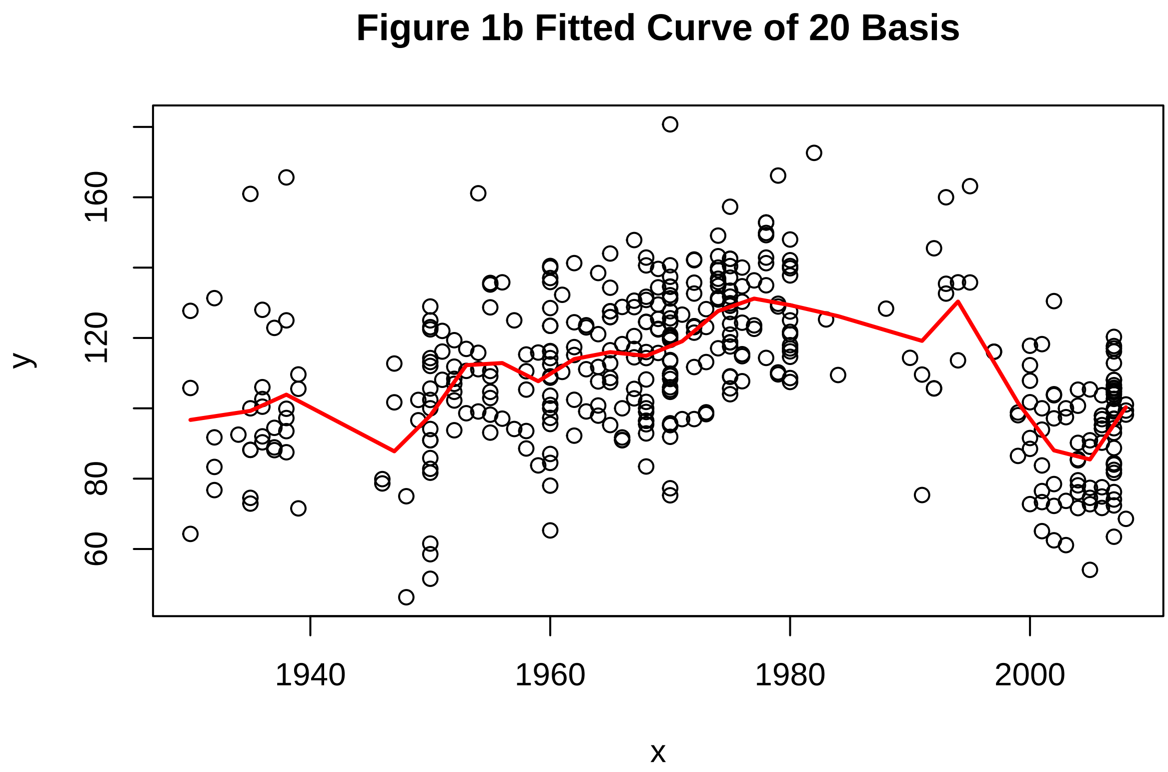
```
##   [1] 133258.2 132827.0 132540.8 132341.8 132198.4 132092.2 132011.9 131950.2
##   [9] 131902.4 131865.1 131836.0 131813.5 131796.2 131783.2 131773.9 131767.5
##  [17] 131763.7 131762.1 131762.3 131764.2 131767.5 131772.1 131777.8 131784.4
##  [25] 131791.9 131800.1 131809.1 131818.6 131828.7 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
```

Question 1.5

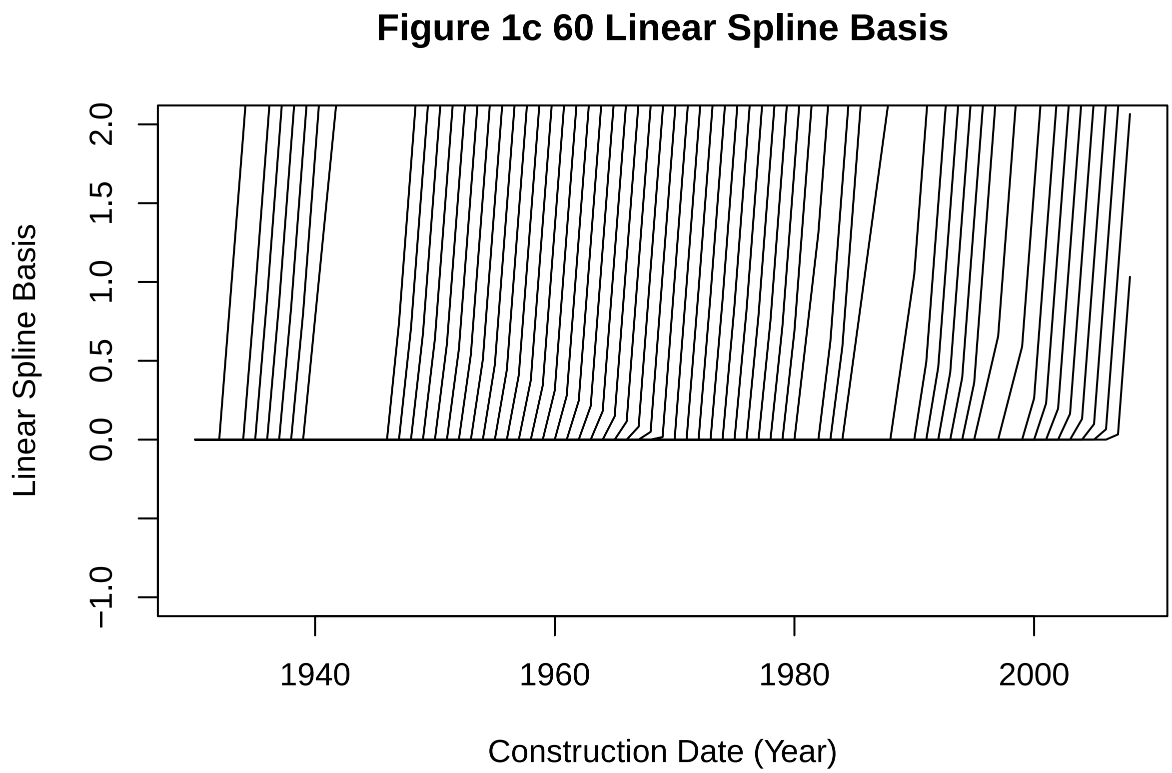
```r
# Obtain lambda corresponding to the minimum GCV
min_20 <- which(GCV_20 == min(GCV_20))
(min_lambda_20 <- lambda[min_20])
```

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

```r
# Overlay the true plot with a fit of y
plot(x, y, main = "Figure 1b Fitted Curve of 20 Basis")
lines(x[order(x)], penalised_spline_20$fitted_y[, min_20][order(x)], col = "red", lwd = 2)
```



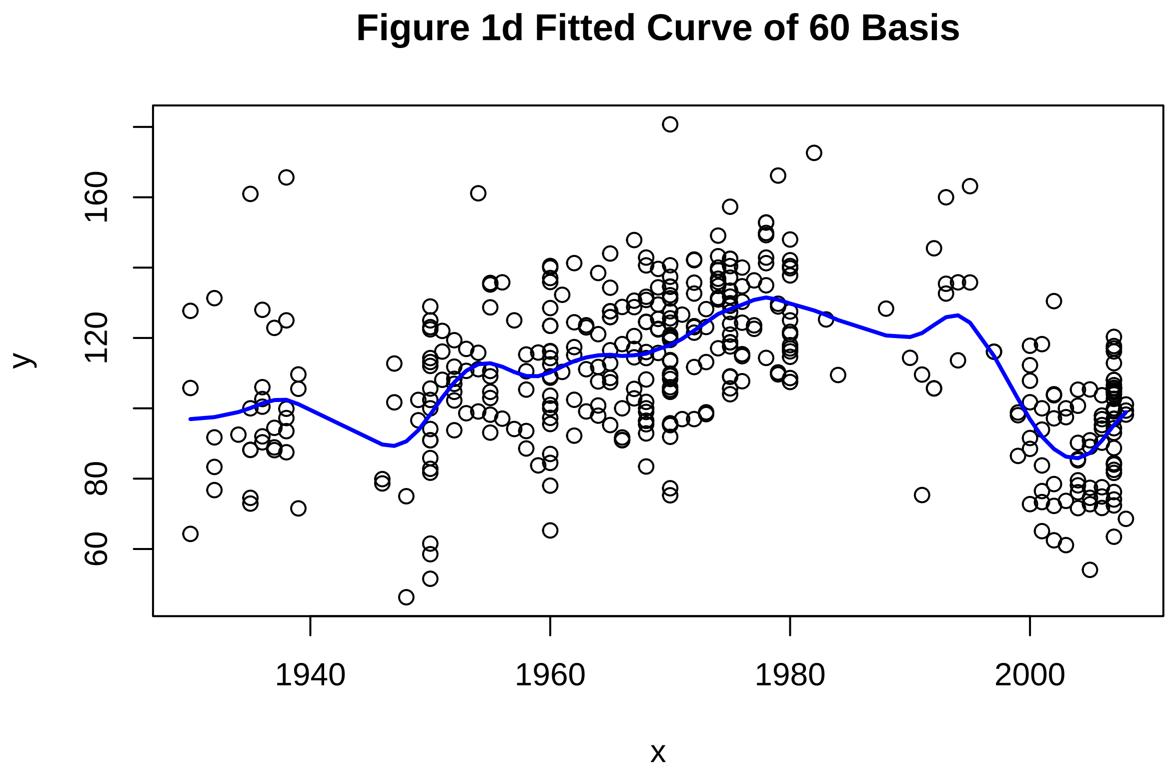

```
# Increase the number of basis to 60
# Exclude extreme values of 0 and 1
probVec <- seq(0, 1, length = 62)[2:61]
k <- quantile(unique(x), probs = probVec)

z <- construct_basis(sort(x), k)

plot(sort(x), z[,1], typ = "l", ylim = c(-1,2), xlab = "Construction Date (Year)",
     ylab = "Linear Spline Basis", main = "Figure 1c 60 Linear Spline Basis")
for (i in 2:dim(z)[2]) {
  lines(sort(x), z[, i])
}
```




```
intercept <- rep(1, length(x))
Z <- construct_basis(x, k)
# Construct matrix C
C <- cbind(intercept, x, Z)

# Construct matrix D
D <- diag(1, dim(C)[2], dim(C)[2])
# Replace first two columns with 0s since we don't want to penalise them
D[, c(1, 2)] <- 0
# Fit penalised spline regression for each lambda
penalised_spline_60 <- fit_penalised_spline(C, D, lambda, y)
GCV_60 <- penalised_spline_60$GCV

# Obtain lambda corresponding to the minimum GCV
min_60 <- which(GCV_60 == min(GCV_60))
(min_lambda_60 <- lambda[min_60])
```

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

```
# Overlay the true plot with a fit of y
plot(x, y, main = "Figure 1d Fitted Curve of 60 Basis")
lines(x[order(x)], penalised_spline_60$fitted_y[, min_60][order(x)], col = "blue", lwd = 2)
```




```
# Compute MSE for basis 20 and 60
paste("MSE for 20 Linear Spline Basis:", RSS_error_20[min_20]/length(y))
```

```
## [1] "MSE for 20 Linear Spline Basis: 297.00135815963"
```

```
paste("MSE for 60 Linear Spline Basis:", penalised_spline_60$RSS_error[min_60]/length(y))
```

```
## [1] "MSE for 60 Linear Spline Basis: 297.4669540763"
```

As we can see from Figure 2d. the fitted curve for 60 linear spline basis appears to be more smooth compared to the one for 20 linear spline basis. However, from the computed mean squared square errors, we cannot see an improvement for the 60 spline basis, and it is even slight higher than the MSE of 20 spline basis.

Question 2.1

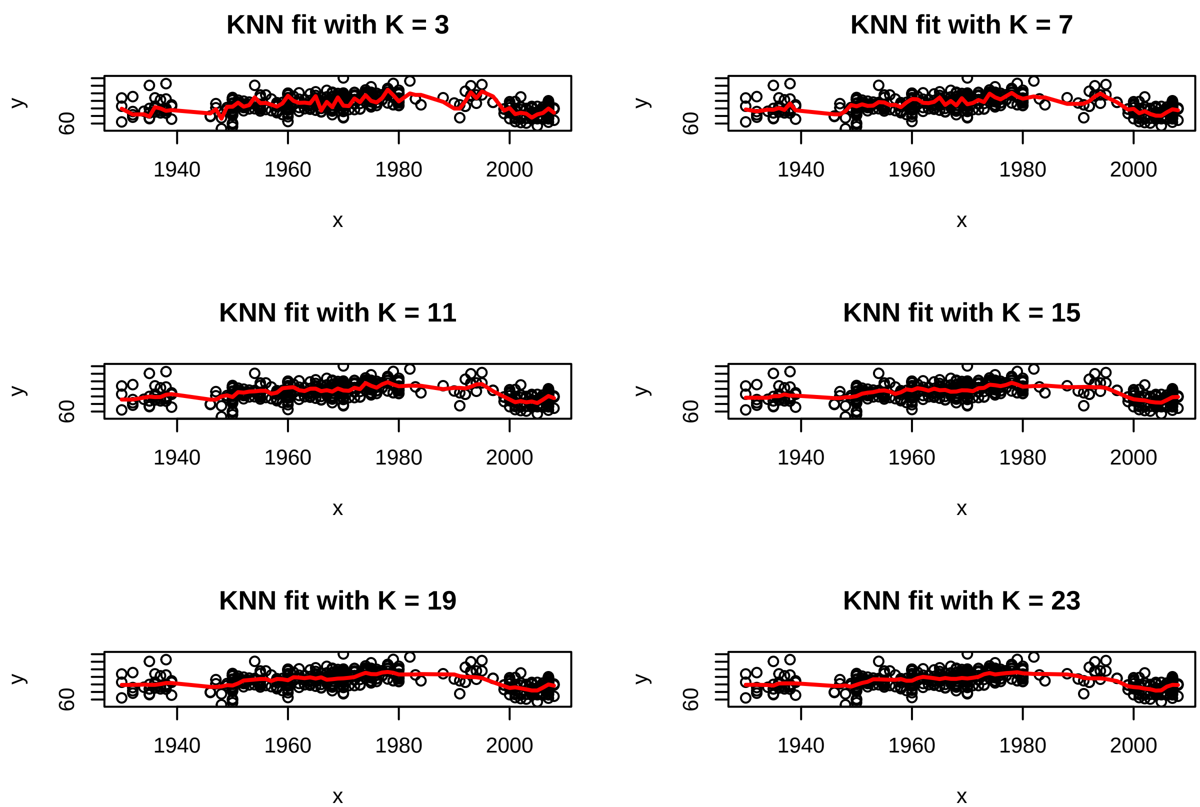
```
# Using 6 values of k selected from 3 to 23
k <- seq(3, 23, length = 6)

KNN <- function(x, cur_x, k, y) {
  abs_dif <- abs(cur_x - x)

  # Sort the absolute difference and find the k nearest neighbors
```

```
  indices <- sort(abs_dif, index.return = T)$ix[1:k]

  # Use these indices and take their mean as the i-th estimate.
  return(mean(y[indices]))
}
```

Question 2.2

```
prediction <- matrix(0, length(y), length(k))
MSE <- rep(0, length(k))

for (i in 1:length(k)) {
  y_hat <- rep(0, length(y))
  count <- 1
  for (j in x) {
    y_hat[count] <- KNN(x, j, k[i], y)
    count <- count + 1
  }
  prediction[, i] <- y_hat
  MSE[i] <- sum((y - y_hat)^2)/length(y)
}
```

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

for (i in 1:length(k)) {
  plot(x, y, main = paste("KNN fit with K =", k[i]))
  lines(x[order(x)], prediction[, i][order(x)], col = "red", lwd = 2)
}
```




```r
# Obtain k corresponding to the minimum MSE
rbind(k, MSE)
```

```
##         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## k     3.0000   7.0000  11.0000  15.0000  19.0000  23.0000
## MSE 331.9637 298.1435 301.1193 302.2849 305.0559 306.4572
```

As we can see from the table, the minimum mean squared error is 298.1435 when $K = 7$, which is quite similar to the one we obtained using penalised spline regression, but slightly higher.

Question 2.3

```r
kernel_name <- c('epanechnikov', 'gaussian', 'biweight', 'triweight', 'uniform', 'tricube')
# Function that accommodate all six kernels
kernel_collection <- function(x) {
  gaussian <- (2 * pi)^(-1/2) * exp(-(x^2)/2)

  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
```

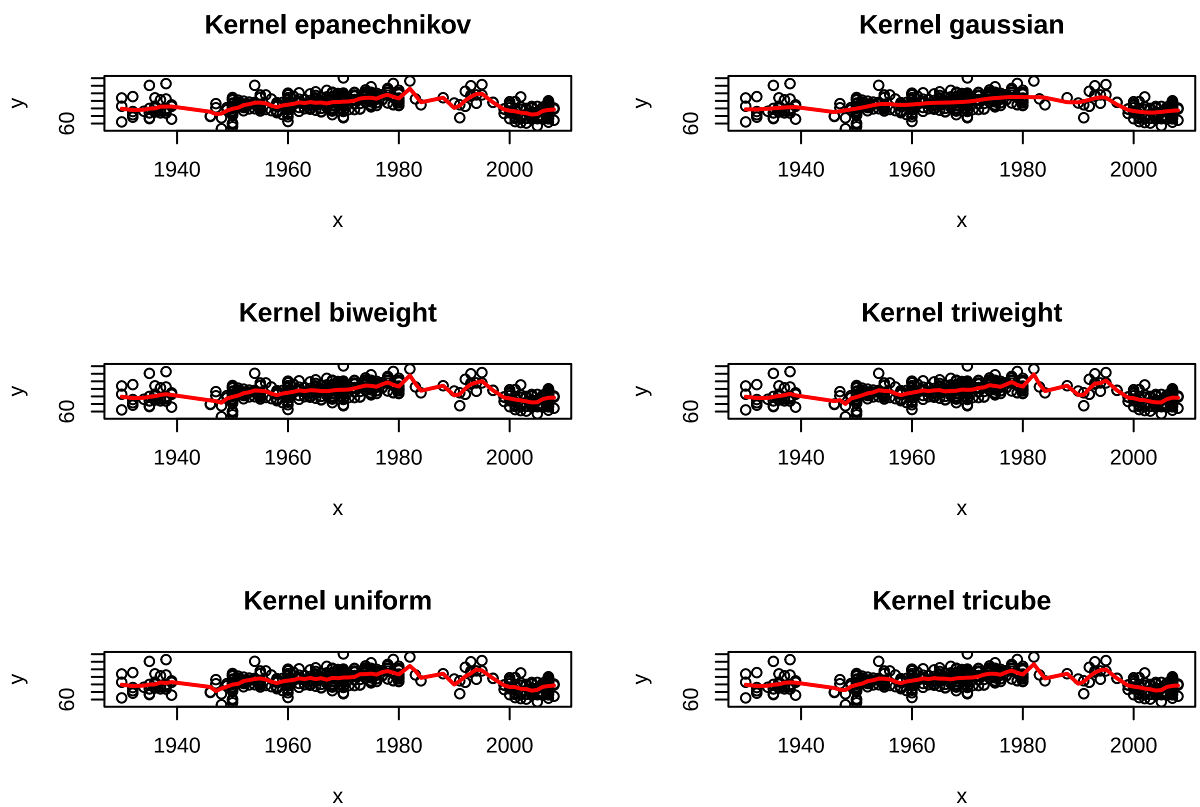
```
  }
  else {
    epanechnikov <- biweight <- triweight <- uniform <- tricube <- 0
  }
  return(c(epanechnikov, gaussian, biweight, triweight, uniform, tricube))
}
```

Question 2.4

```
# Bandwidth
h <- 2

kernel_regression <- function(x, y) {
  prediction <- matrix(0, length(y), 6)

  for (i in 1:length(y)) {
    weight <- matrix(0, length(y), 6)
    nw <- 0
    for (j in 1:length(y)) {
      weight[j, ] = kernel_collection((x[j] - x[i])/h)
      nw <- nw + (weight[j, ] * y[j])
    }
    prediction[i,] <-  nw / colSums(weight)
  }
  colnames(prediction) <- kernel_name
  return(prediction)
}
```

Question 2.5

```
prediction <- kernel_regression(x, y)
par(mfrow=c(3,2))

for (i in 1:6) {
  plot(x, y, main = paste("Kernel", kernel_name[i]))
  lines(x[order(x)], prediction[, i][order(x)], col = "red", lwd = 2)
}
```



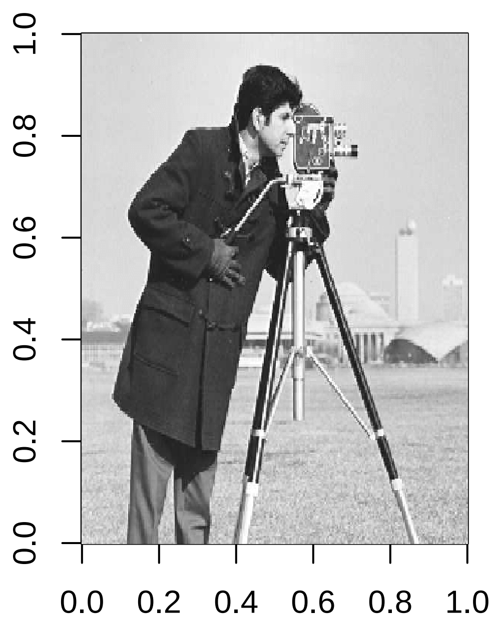

```r
# Estimate the MSE for each kernel
colSums(((y - prediction)^2)/length(y))
```

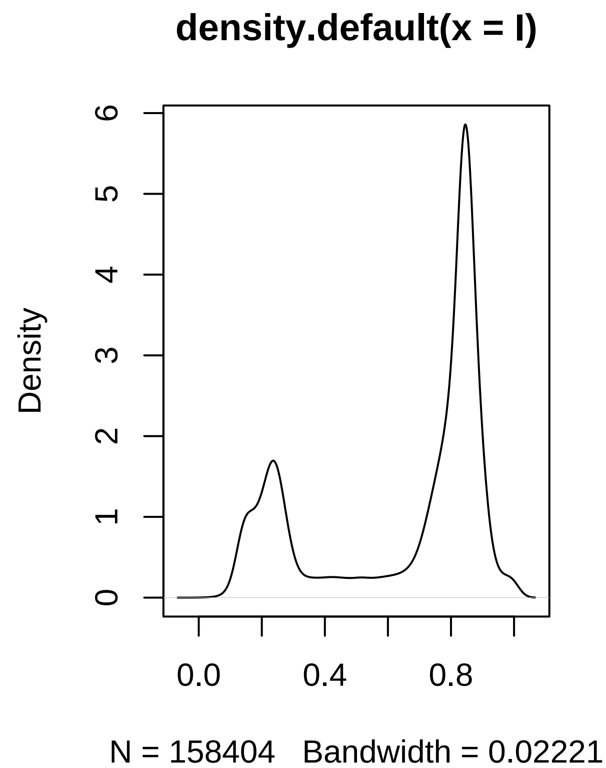
```
## epanechnikov     gaussian     biweight    triweight      uniform      tricube
##     285.5042     300.2880     278.8030     272.9163     292.8334     282.7872
```

As we can see, triweight kernel gives the lowest mean square error at 272.9163.

Question 3.1

```r
# Load libraries and image
suppressMessages(library(plot.matrix))
suppressMessages(library(png))
suppressMessages(library(fields))

I <- readPNG("CM.png")
I <- I[,, 1]
I <- t(apply(I, 2, rev))

par(mfrow=c(1, 2))
image(I, col = gray((0:255) / 255))
plot(density(I))
```





Question 3.2 and 3.3

```
compute.prob.x.z <- function(X, pi.curr, mu.curr, sigma.curr) {
  L <- matrix(NA, nrow = length(X), ncol = length(pi.curr))
  for (k in seq_len(ncol(L))) {
    L[, k] <- dnorm(X, mean = mu.curr[k], sd = sigma.curr[k]) * pi.curr[k]
  }
  return(list(p = L))
}

EM.iter <- function(X, pi.curr, mu.curr, sigma.curr) {

  # Expectation step
  p <- compute.prob.x.z(X, pi.curr, mu.curr, sigma.curr)$p

  # Compute the responsibility
  P_ik <- p / rowSums(p)

  # Maximisation step
  pi.new <- colSums(P_ik) / sum(P_ik)
  mu.new <- colSums(P_ik*X) / colSums(P_ik)

  temp <- P_ik * (replicate(length(mu.new), X) - t(replicate(length(X), mu.new)))^2
  sigma.new <- sqrt(colSums(temp)/colSums(P_ik))
```

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

EM <- function(X, p, mu, sigma, epsilon = 1e-6, max.iter=100) {
  p.curr <- p
  mu.curr <- mu
  sigma.curr <- sigma
  e.curr <- 0

  delta <- 1
  count <- 1
  while ((delta > epsilon)) {
    # run EM step to compute new values for the parameters
    EM.out <- EM.iter(X, p.curr, mu.curr, sigma.curr)
    count <- count + 1
    # print the new estimates
    print(round(c(EM.out$mu.new, EM.out$sigma.new, EM.out$pi.new), 4))

    # compute the stopping criteria
    e.new <- sum(EM.out$mu.new - mu.curr) +
      sum(EM.out$sigma.new - sigma.curr)
    delta <- abs(e.new - e.curr)

    # replace current values with new estimates
    p.curr <- EM.out$pi.new
    mu.curr <- EM.out$mu.new
    sigma.curr <- EM.out$sigma.new
    e.curr <- e.new
  }
  return(list(pi = EM.out$pi.new, mu = EM.out$mu.new, sigma = EM.out$sigma.new))
}
```

```r
# Define the initial values for the mixture model
p.init <- c(0.2, 0.3, 0.5)
mu.init <- c(0, 0.4, 0.7)
sigma.init <- c(0.2, 0.2, 0.2)
# Run EM
result <- EM(as.vector(I), p.init, mu.init, sigma.init)
```

```
## [1] 0.2024 0.3936 0.7970 0.0650 0.2464 0.1356 0.0852 0.2227 0.6921
## [1] 0.2102 0.3786 0.8197 0.0495 0.2216 0.0813 0.1395 0.1772 0.6832
## [1] 0.2120 0.3923 0.8289 0.0473 0.1888 0.0629 0.1732 0.1502 0.6766
## [1] 0.2119 0.4177 0.8322 0.0478 0.1785 0.0578 0.1886 0.1416 0.6697
## [1] 0.2115 0.4438 0.8336 0.0486 0.1772 0.0560 0.1985 0.1375 0.6640
## [1] 0.2115 0.4673 0.8344 0.0494 0.1774 0.0550 0.2061 0.1350 0.6589
## [1] 0.2119 0.4877 0.8350 0.0500 0.1775 0.0542 0.2120 0.1337 0.6544
## [1] 0.2122 0.5054 0.8354 0.0506 0.1777 0.0536 0.2165 0.1334 0.6501
## [1] 0.2126 0.5207 0.8358 0.0511 0.1779 0.0530 0.2200 0.1340 0.6460
## [1] 0.2130 0.5341 0.8362 0.0515 0.1784 0.0525 0.2227 0.1353 0.6420
## [1] 0.2133 0.5459 0.8364 0.0519 0.1792 0.0519 0.2249 0.1371 0.6380
## [1] 0.2136 0.5565 0.8367 0.0522 0.1803 0.0514 0.2265 0.1394 0.6341
## [1] 0.2138 0.5663 0.8368 0.0524 0.1816 0.0509 0.2278 0.1421 0.6301
## [1] 0.2140 0.5753 0.8370 0.0526 0.1831 0.0503 0.2288 0.1452 0.6260
```

```
## [1] 0.2142 0.5838 0.8371 0.0528 0.1847 0.0497 0.2296 0.1486 0.6218
## [1] 0.2144 0.5919 0.8372 0.0529 0.1863 0.0491 0.2302 0.1523 0.6175
## [1] 0.2145 0.5995 0.8373 0.0531 0.1877 0.0485 0.2307 0.1562 0.6131
## [1] 0.2146 0.6067 0.8374 0.0532 0.1888 0.0478 0.2312 0.1602 0.6086
## [1] 0.2147 0.6135 0.8375 0.0533 0.1896 0.0472 0.2317 0.1641 0.6042
## [1] 0.2148 0.6198 0.8377 0.0534 0.1901 0.0465 0.2321 0.1681 0.5998
## [1] 0.2149 0.6257 0.8378 0.0535 0.1903 0.0459 0.2325 0.1719 0.5956
## [1] 0.2150 0.6311 0.8379 0.0536 0.1901 0.0453 0.2330 0.1756 0.5915
## [1] 0.2151 0.6361 0.8381 0.0537 0.1898 0.0448 0.2334 0.1790 0.5875
## [1] 0.2152 0.6407 0.8382 0.0538 0.1892 0.0442 0.2339 0.1823 0.5838
## [1] 0.2152 0.6450 0.8384 0.0539 0.1885 0.0437 0.2344 0.1854 0.5803
## [1] 0.2153 0.6488 0.8385 0.0540 0.1877 0.0433 0.2348 0.1883 0.5769
## [1] 0.2154 0.6524 0.8387 0.0541 0.1869 0.0429 0.2353 0.1909 0.5738
## [1] 0.2155 0.6557 0.8388 0.0542 0.1860 0.0425 0.2357 0.1935 0.5708
## [1] 0.2156 0.6587 0.8390 0.0543 0.1851 0.0421 0.2362 0.1958 0.5680
## [1] 0.2157 0.6615 0.8391 0.0544 0.1842 0.0418 0.2366 0.1980 0.5654
## [1] 0.2158 0.6641 0.8393 0.0545 0.1833 0.0414 0.2370 0.2001 0.5629
## [1] 0.2159 0.6664 0.8394 0.0546 0.1825 0.0411 0.2373 0.2021 0.5606
## [1] 0.2160 0.6686 0.8395 0.0547 0.1817 0.0409 0.2377 0.2039 0.5584
## [1] 0.2161 0.6706 0.8397 0.0548 0.1809 0.0406 0.2380 0.2057 0.5563
## [1] 0.2162 0.6725 0.8398 0.0549 0.1801 0.0404 0.2383 0.2073 0.5544
## [1] 0.2163 0.6742 0.8399 0.0549 0.1794 0.0401 0.2386 0.2089 0.5525
## [1] 0.2164 0.6758 0.8400 0.0550 0.1788 0.0399 0.2389 0.2103 0.5508
## [1] 0.2164 0.6773 0.8401 0.0551 0.1781 0.0397 0.2392 0.2117 0.5491
## [1] 0.2165 0.6787 0.8402 0.0552 0.1775 0.0395 0.2394 0.2130 0.5476
## [1] 0.2166 0.6800 0.8403 0.0553 0.1770 0.0394 0.2396 0.2143 0.5461
## [1] 0.2167 0.6812 0.8404 0.0553 0.1765 0.0392 0.2398 0.2155 0.5447
## [1] 0.2167 0.6823 0.8405 0.0554 0.1760 0.0390 0.2400 0.2166 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 0.1750 0.0387 0.2404 0.2186 0.5410
## [1] 0.2169 0.6852 0.8407 0.0556 0.1746 0.0386 0.2406 0.2196 0.5399
## [1] 0.2169 0.6861 0.8407 0.0556 0.1742 0.0385 0.2407 0.2205 0.5388
## [1] 0.2170 0.6868 0.8408 0.0557 0.1739 0.0384 0.2409 0.2213 0.5378
## [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 0.0558 0.1732 0.0382 0.2411 0.2229 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 0.2243 0.5344
## [1] 0.2172 0.6901 0.8411 0.0559 0.1723 0.0379 0.2415 0.2249 0.5336
## [1] 0.2172 0.6907 0.8411 0.0559 0.1720 0.0378 0.2416 0.2255 0.5329
## [1] 0.2173 0.6912 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 0.0376 0.2417 0.2266 0.5316
## [1] 0.2173 0.6921 0.8412 0.0560 0.1713 0.0376 0.2418 0.2271 0.5310
## [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 0.5299
## [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 0.6940 0.8414 0.0561 0.1704 0.0373 0.2422 0.2293 0.5285
## [1] 0.2175 0.6943 0.8414 0.0562 0.1703 0.0372 0.2422 0.2297 0.5281
## [1] 0.2175 0.6946 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 0.2313 0.5263
## [1] 0.2176 0.6958 0.8416 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 0.6964 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 0.5244
## [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 0.5240
## [1] 0.2177 0.6973 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 0.5229
## [1] 0.2178 0.6980 0.8417 0.0564 0.1684 0.0366 0.2429 0.2342 0.5228
## [1] 0.2178 0.6981 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 0.2431 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 0.5216
## [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 0.2354 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 0.2354 0.5215
## [1] 0.2178 0.6989 0.8418 0.0565 0.1679 0.0365 0.2431 0.2355 0.5214
```

Question 3.4

```r
# Compute the probability of each pixels being in each class
predict_prob <- compute.prob.x.z(as.vector(I), result$pi, result$mu, result$sigma)

# Classify each pixels based on their pdf estimation
label <- rep(0, length(as.vector(I)))
for (i in 1:length(as.vector(I))) {
  label[i] <- which(predict_prob$p[i, ] == max(predict_prob$p[i, ]))
}

posterior <- predict_prob$p / rowSums(predict_prob$p)
mean <- t(replicate(length(as.vector(I)), result$mu))
posterior.mean <- rowSums(posterior * mean)

# Assign values based on figure 3a
```

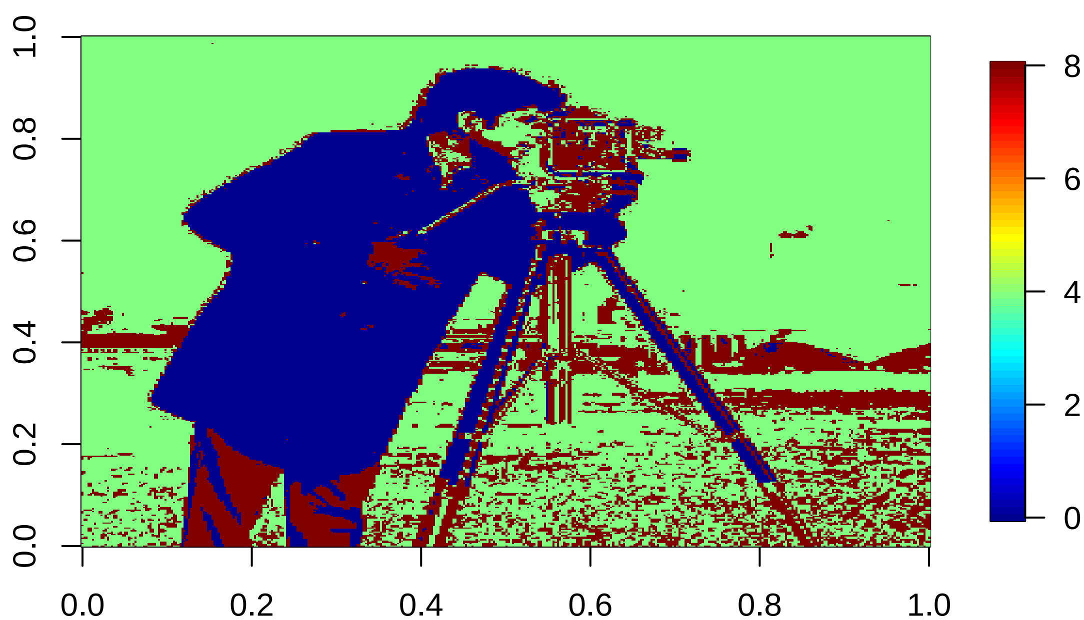
```r
label[label == 1] <- 0
label[label == 2] <- 8
label[label == 3] <- 4
```

Question 3.5

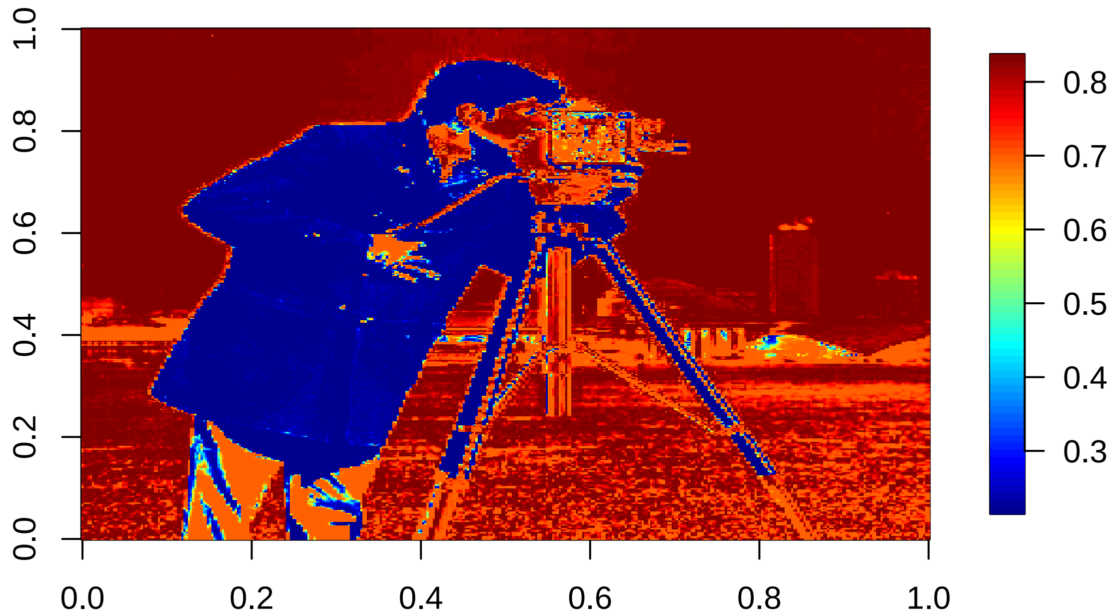
```r
# Check dimension for reshape
dim(I)
```

```
## [1] 398 398
```

```r
# Plot figure 3a
image.plot(matrix(label, 398, 398))
```




```r
# Plot figure 3b
image.plot(matrix(posterior.mean, 398, 398))
```



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