HW4

2024-09-29

Homework 4

Due 5 October

The total points on this homework is 175. Out of these, 5 points are reserved for clarity of presentation, punctuation and commenting with respect to the code.

The homework for this week are mainly exercises in using the apply() function, and its benefits in many cases where the problem can be "reduced" (which may mean, expanded for some cases) to operations at the margins of an array. Some of the problems are for a bigger dataset that is more cumbersome to notice operations on, so you are advised to try it out first on a small array (say of dimension 3x4x5) and then moving to the given problem once you are confident of your approach to solving the problem.

Q1

The file fbp-img.dat on Canvas is a 128×128 matrix containing the results of an image of fluorodeoxyglucse (FDG-18) radio-tracer intake reconstructed using Positron Emission Tomography (PET).

(a):

Read in the data as a matrix.

```r
fbpImg <- scan("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/fbp-img.dat") |>
  matrix(nrow = 128, ncol = 128)
```

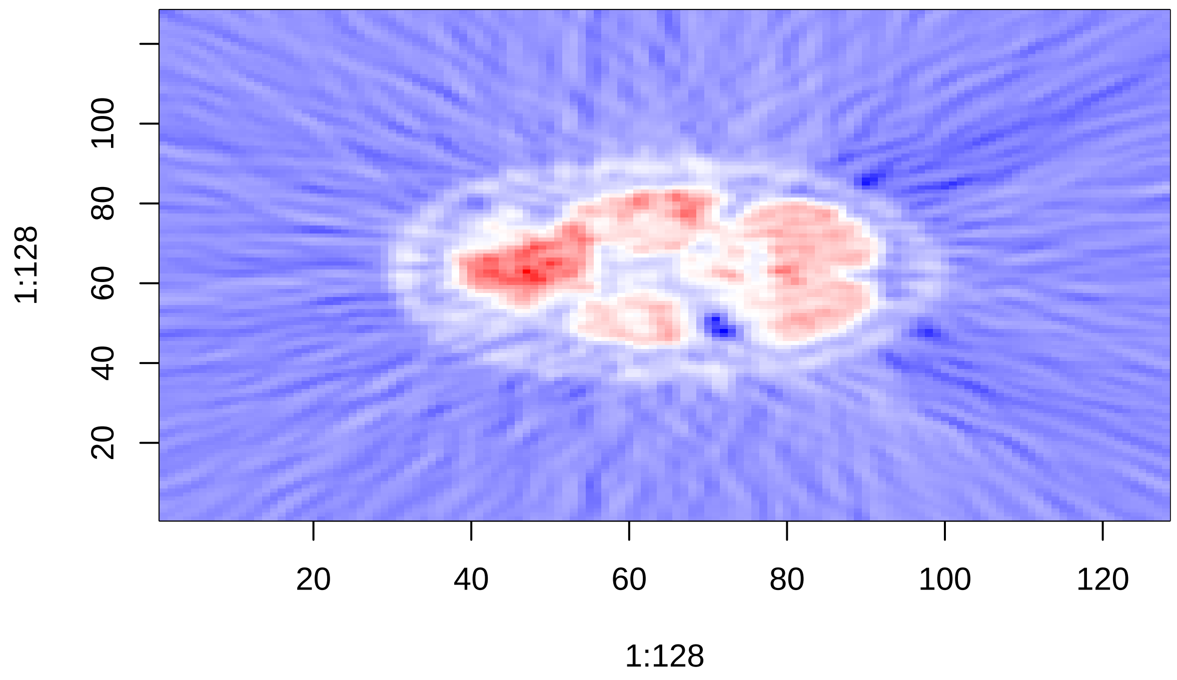
(b):

Use the image() function in R with your choice of color map to image the data.

```r
require(wesanderson)
```

```
## Loading required package: wesanderson
```

```r
image(1:128,
      1:128,
      fbpImg[, 128:1],
      col=colorspace::diverge_hsv(256)
      )
```



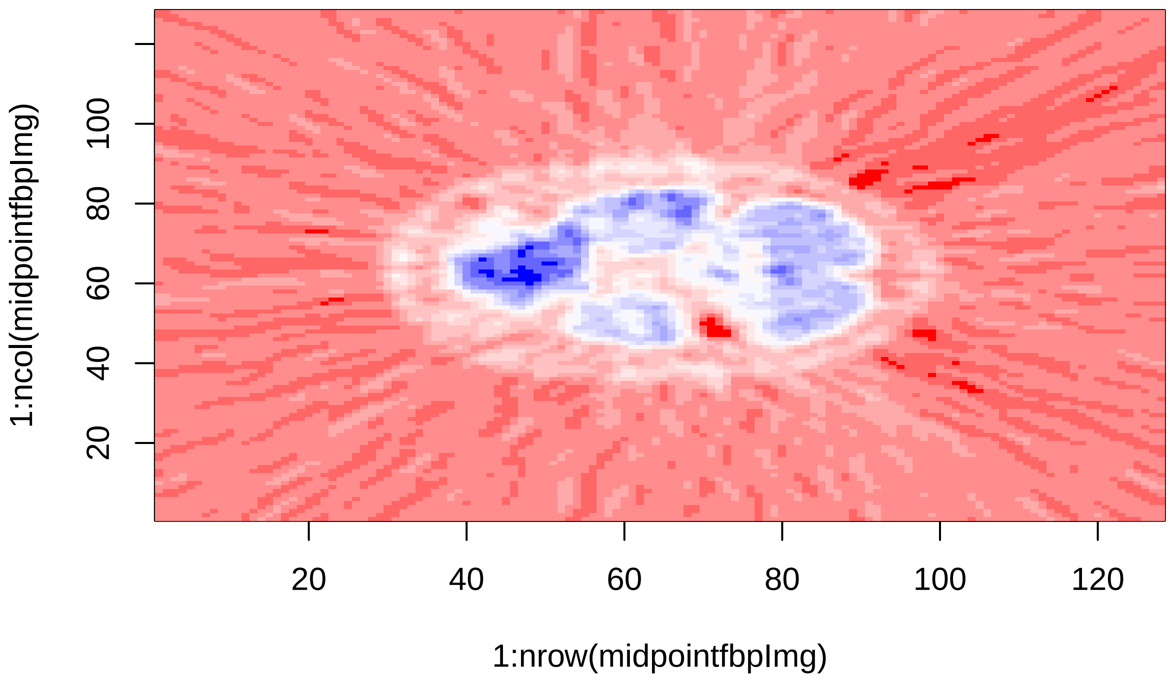
```r
# wes_palette(n = 256, name = "AsteroidCity3", type = "continuous")
# colorspace::diverge_hsv(256)
```

(c):

We will now (albeit, somewhat crudely) compress the data in the following two ways:

i. Obtain the range of values in the matrix, subdividing into 16 bins. Bin the values in the matrix and replace each value with the mid-point of each bin. Image these new binned matrix values.

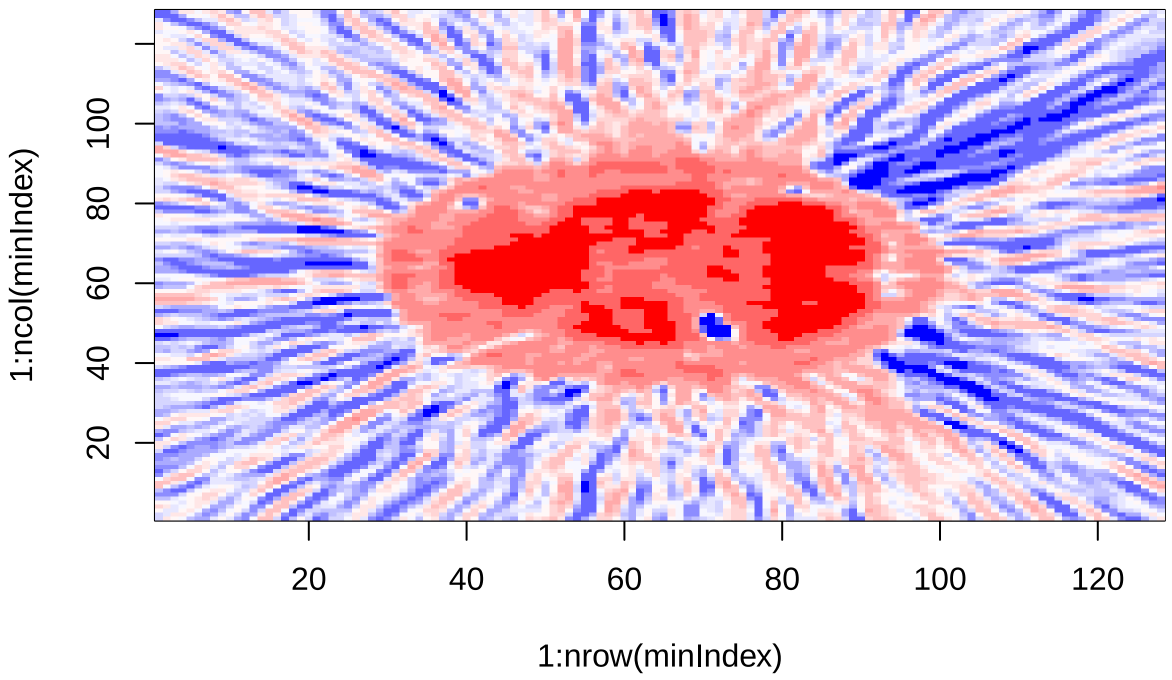
```r
rangefbpImg <- range(fbpImg)

# 16 bins implies 17 total lengths
breaks <- seq(to = min(rangefbpImg),
              from = max(rangefbpImg),
              length.out = 17)

# Calculate the midpoint of each bin
midpoints <- (breaks[-1] + breaks[-length(breaks)]) / 2

# Bin matrix using values derived
# replace each bin with its midpoint
# gives index values
binnedfbpImg <- cut(fbpImg,
                     breaks = breaks,
                     labels = FALSE,
                     include.lowest = TRUE)

# Replace binned values with corresponding midpoints
midpointMat <- midpoints[binnedfbpImg]

# Reshape the vector back into a matrix
midpointfbpImg <- matrix(midpointMat,
                         nrow = nrow(fbpImg),
                         ncol = ncol(fbpImg))

image(1:nrow(midpointfbpImg),
      1:ncol(midpointfbpImg),
      midpointfbpImg[, 128:1],
      col=colorspace::diverge_hsv(256)
      )
```



```
# wes_palette(n = 256, name = "AsteroidCity3", type = "continuous")
# colorspace::diverge_hsv(256)
```

ii. In this second case, we will group according to the 16 equally-spaced quantile bins, which can be obtained using the quantile() function in R with appropriate arguments. Use these obtained quantile bins to group the data, replacing each entry in the matrix with its mid-point. Image these new binned matrix values.

```r
# 16 bins implies 17 total lengths
qtFbpImg <- quantile(fbpImg,
                     probs = seq(0, 1, length = 17)
                     )
midQtFbpImg <- (qtFbpImg[-1] + qtFbpImg[-length(qtFbpImg)])/2

# stack the matrix 16 times on top of each other
arrFbpImg <- array(fbpImg,
                   dim = c(dim(fbpImg),
                              length(midQtFbpImg)
                           )
                 )
# dim(arrFbpImg)
midArr <- array(rep(midQtFbpImg,
                    each = prod(dim(fbpImg))
                    ),
                 dim = dim(arrFbpImg)
                 )
# dim(midArr)
sqDiffArr <- (arrFbpImg - midArr) ^ 2

minIndex <- apply(X = sqDiffArr,
                  MARGIN = c(1, 2),
                  FUN = which.min)
# dim(minIndex)
image(1:nrow(minIndex),
      1:ncol(minIndex),
      minIndex[, 128:1],
      col=colorspace::diverge_hsv(256)
      )
```



```
# col=wes_palette(n = 256, name = "AsteroidCity3", type = "continuous")
# colorspace::diverge_hsv(256)
```

iii. Comment, and discuss similarities and differences on the differences in the two images thus obtained with the original image.

The two images do appear similar, in that you can clearly visualize the outline of the brain and the distinct regions of the brain. However, the quantile-based image (second image) more clearly visualizes some of the smaller areas whose values differ considerably from the larger regions of the brain.

Q2

Microarray gene expression data. The file, accessible on Canvas at diurnaldata.csv contains gene expression data on 22,810 genes from Arabidopsis plants exposed to equal periods of light and darkness in the diurnal cycle. Leaves were harvested at eleven time-points, at the start of the experiment (end of the light period) and subsequently after 1, 2, 4, 8 and 12 hours of darkness and light each. Note that there are 23 columns, with the first column representing the gene probeset. Columns 2–12 represent measurements on gene abundance taken at 1, 2, 4, 8 and 12 hours of darkness and light each, while columns 13-23 represent the same for a second replication.

(a):

Read in the dataset. Note that this is a big file, and can take a while, especially on a slow connection.

```r
require(readr)
```

```
## Loading required package: readr
```

```r
# I downloaded it locally!
geneDf <- read.csv("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/diurnaldata.csv")
```

(b):

For each gene, calculate the mean abundance level at each time-point, and store the result in a matrix. One way to achieve this is to create a three-dimensional array or dimension 22,810 $\times$ 11 $\times$ 2 and to use apply over it with the appropriate function and over the appropriate margins. Note that you are only asked present the commands that you use here. From now on, we will use this dataset averaged over the two replications for each gene.

```r
# Exclude first column
geneMat <- as.matrix(geneDf[, -1])

geneArray <- array(geneMat, dim = c(22810, 11, 2))

meanGene <- apply(X = geneArray, MARGIN = c(1, 2), FUN = mean)
dim(meanGene)
```

```
## [1] 22810    11
```

```r
# meanGene
```

(c):

Standardization. Gene data are compared to each other by way of correlations. (Correlation between any two sequences is related, by means of an affine (linear) transformation, to the Euclidean distance between the sequences, after standardization to have mean zero and standard deviation 1).

i. In pursuance of the objective, use the apply function to calculate the mean of the mean abundance level over all time-points for each gene.

```r
meanGeneAllTimes <- apply(X = meanGene, MARGIN = 1, FUN = mean)
length(meanGeneAllTimes)
```

```
## [1] 22810
```

```r
# meanGeneAllTimes
```

ii. Set up a matrix of dimension 22,810 × 11 of the means of the mean abundance levels, where each row is a replicated version of the first one. Use this to eliminate the mean effect from the matrix stored in the previous part.

```r
meanRepGene <- matrix(meanGeneAllTimes, nrow = 22810, ncol = 11, byrow = TRUE)

meanGeneNoMeanEffect <- meanGene - meanRepGene

dim(meanGeneNoMeanEffect)
```

```
## [1] 22810    11
```

iii. Use the apply function again to calculate the standard deviation of each row of the matrix in the part (b) above, and proceed to obtain the scaled measurements on the genes.

```r
stdGene <- apply(X = meanGeneNoMeanEffect, MARGIN = 1, FUN = sd)

stdGeneMat <- matrix(rep(stdGene, each = 11),
                     nrow = 22810,
                     ncol = 11,
                     byrow = TRUE)

scaledGenes <- meanGene / stdGeneMat
dim(scaledGenes)
```

```
## [1] 22810    11
```

(d):

The file micromeans.dat contains a 20 × 11 matrix of measurements. Read in this dataset, and standardize as above.

```r
geneDfMicro <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/micromeans.da
  as.matrix(nrow = 20,
            ncol = 11)

stdMicro <- apply(X = geneDfMicro, MARGIN = 1, FUN = sd)

stdMicroMat <- matrix(rep(stdMicro, each = 11),
                                    nrow = 20,
                                    ncol = 11,
                                    byrow = TRUE)
```

```r
scaledMicro <- geneDfMicro / stdMicroMat

dim(scaledMicro)
```

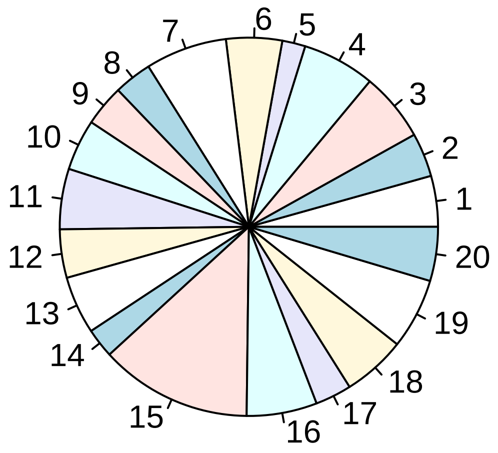
```
## [1] 20 11
```

(e):

We will now identify which of these means is closest to each gene. To do so, set up an three-dimensional array (of dimension 22,810×11×20) with 20 replicated datasets (of the 22,810 diurnal mean abundance levels over 11 time-points). Next, set up replications of the 20 mean vectors into a three-dimensional array (of dimension 22, 810 × 11 × 20), with the replications occurring along the first dimension. Using apply and the above, obtain a 22,810 × 20 matrix of Euclidean distances for each gene and mean. Finally, obtain the means to which each gene is closest to. Report the frequency distribution, and tabulate the frequencies, and display using a piechart.

```r
meanGeneArr <- array(rep(meanGene, each = 20),
                             dim = c(22, 810, 20)
                             )
microGeneArr <- array(rep(geneDfMicro, times = 22810),
                          dim = c(22810, 11, 20)
                          )

distancesArr <- array(0,
                      dim = c(22810, 20)
                      )

for (gene_index in 1:22810) {
  for (micro_index in 1:20) {
    distancesArr[gene_index, micro_index] <-
      sqrt(sum(
        (meanGene[gene_index, ] - geneDfMicro[micro_index, ])^2)
        )
  }
}

closestMeansIndex <- apply(X = distancesArr, MARGIN = 1, FUN = which.min)

frequencyTable <- table(closestMeansIndex)

pie(frequencyTable, main = "Frequency Distribution of Genes by Their Closest Means")
```

Frequency Distribution of Genes by Their Closest Means



Q3

The United States Postal Service has had a long-term project to automating the recognition of handwritten digits for zip codes. The file ziptrain.dat has data on different numbers of specimens for each digit. Each observation is in the form of a 256-dimensional vector of pixel intensities. These form a 16x16 image of pixel intensities for each digit. Each digit is determined to be a specimen of the actual digit given in the file zipdigit.dat. The objective is to clean up the dataset to further distinguish one digit from the other.

(a):

Read in the dataset as a vector. This dataset has 2000 rows and 256 columns. Convert and store the dataset as a three-dimensional array of dimensions $16 \times 16 \times 2000$.

```
ziptrain <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/ziptrain.dat")
# zipdigit <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/zipdigit.dat")
# dim(ziptrain)
# dim(t(ziptrain))

ziptrain3D <- array(t(ziptrain), dim = c(16, 16, 2000))

dim(ziptrain3D)
```

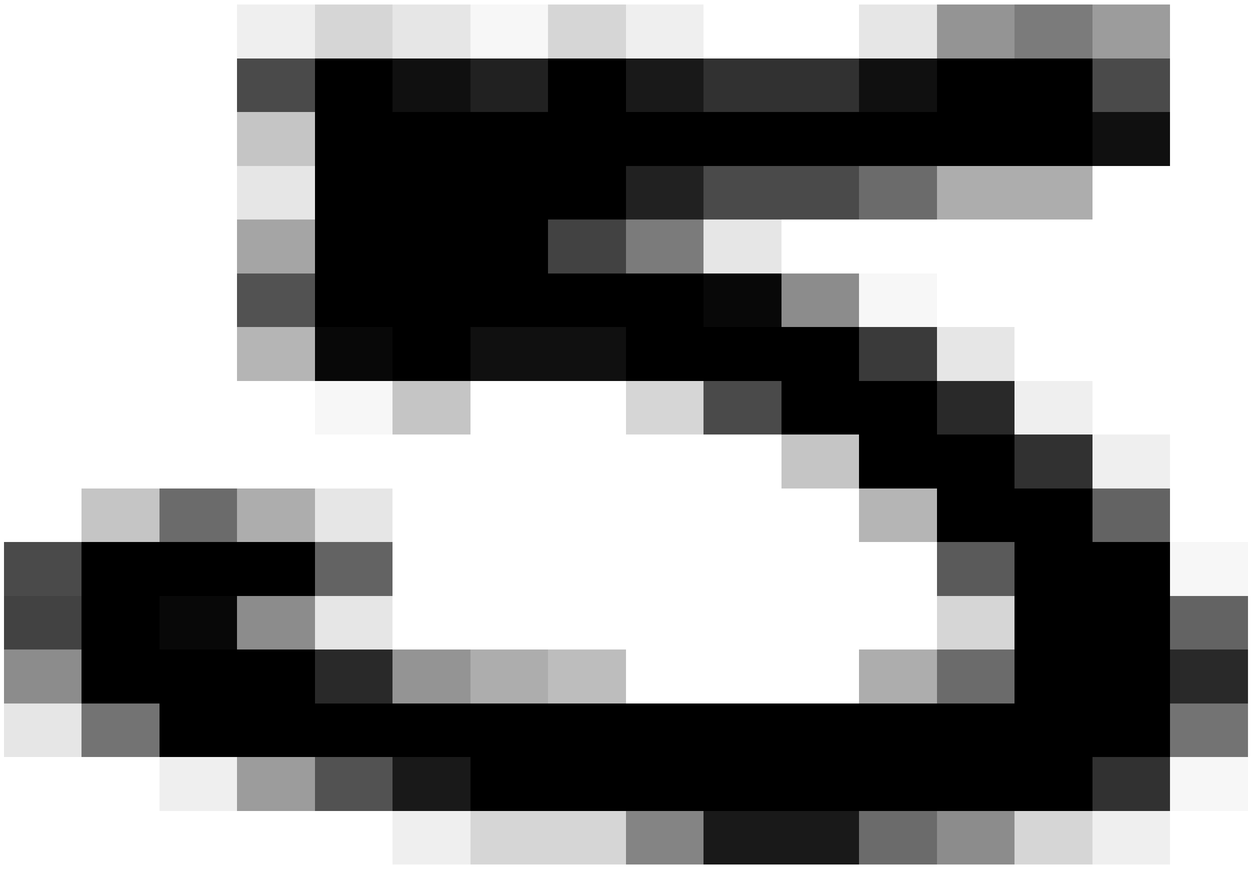
```
## [1]   16   16 2000
```

(b):

Our objective is to image all the 2000 pictures in a 40×50 display. We will hasten this display process by setting up the stage for using apply.

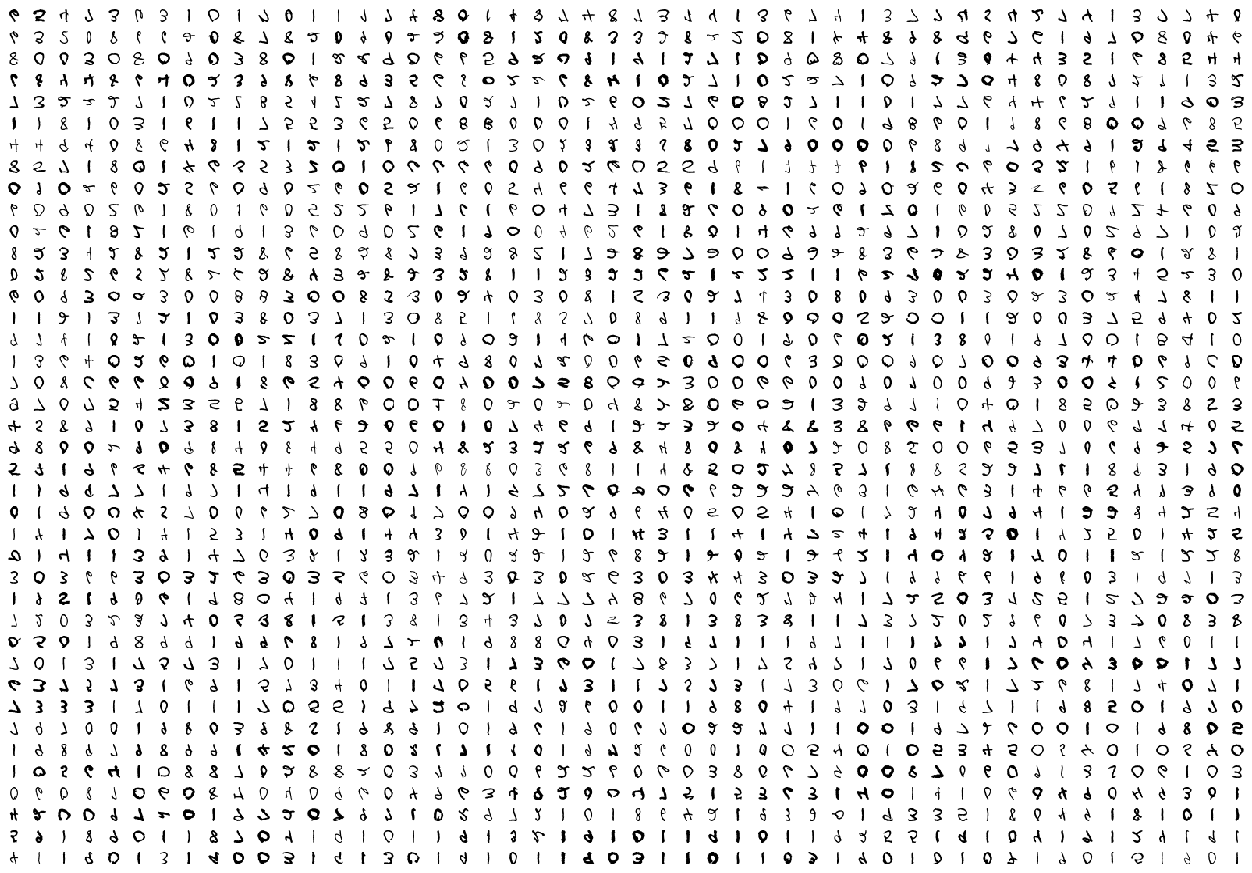
i. If we image the second record, we note that without any changes, the image mirrors the digit "5". Fix these by reversing the appropriate dimension of the array. Also note that the white space around the axes is wasted space and should be removed. To substantially reduce the white space, consider par(mar = rep(0.05, 4)). Further, the axes conveys no information so consider its elimination, using axes = F in the call to the image() function. Finally, we may as well use a gray scale here so consider using col = gray(31:0/31) in the image argument. Display the second record as an image, after addressing all these issues.

```
properArray <- ziptrain3D[, 16:1, ]

par(mar = rep(0.05, 4))

image(properArray[,, 2],
      axes = FALSE,
      col = gray(31:0/31)
      )
```



ii. Using par(mfrow = c(40,50)) and a total image size of 5.2”×6.5”, image all the 2000 digits using apply and the experience you gained in part i. above.

```r
par(mfrow = c(40, 50),
    mar = rep(0.05, 4),
    # this pin size works, I guess
    # pin = c(5.2/50, 6.5/40)
    # pin = c(5.2/40, 6.5/50)
    # pin = c(5.2/100, 6.5/80)
    pin = c(5.2/80, 6.5/100)
    )

apply(X = properArray, MARGIN = 3, FUN = function(x) {
  image(x[, 16:1],
        axes = FALSE,
        col = gray(31:0/31))
})
```



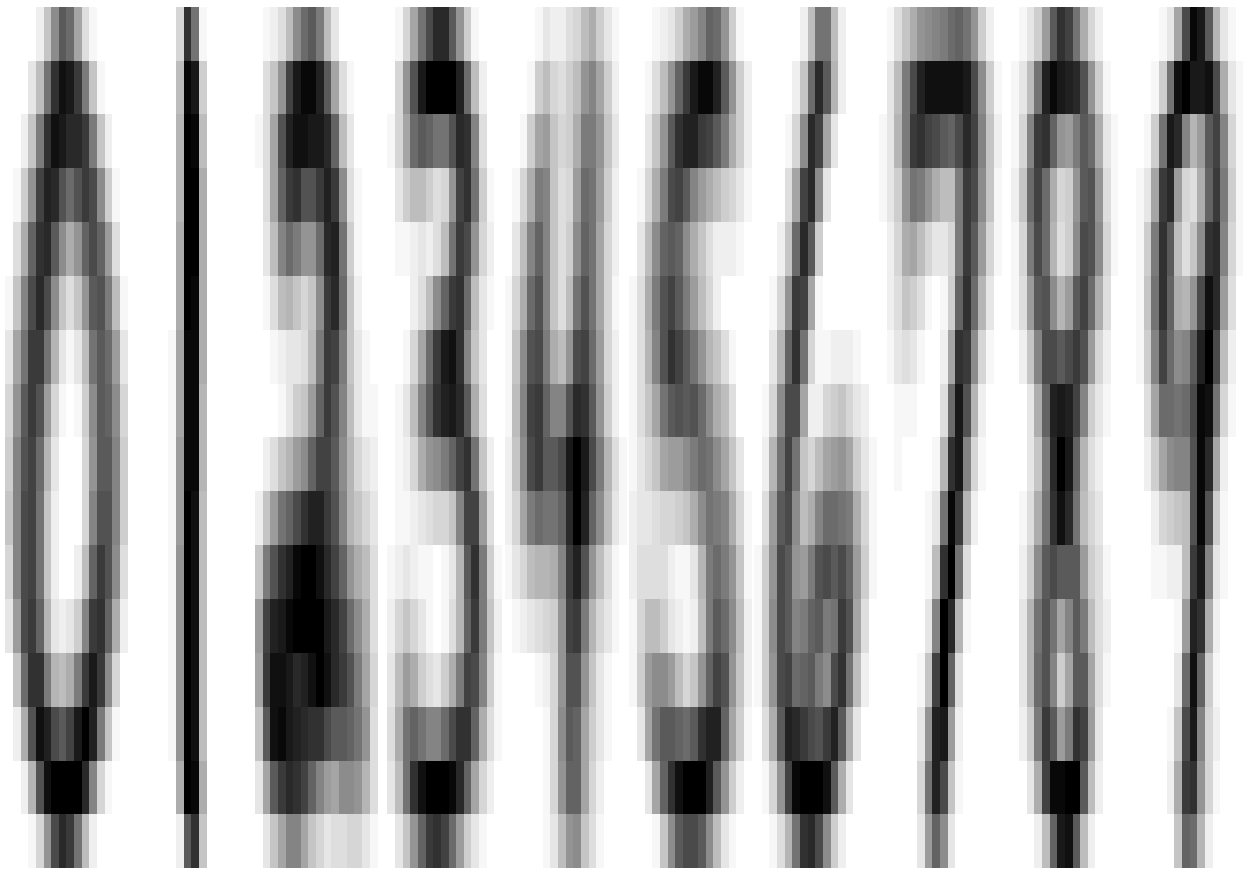
```
## NULL
```

(c):

We now compute the mean and standard deviation images of the digits. To do so, we convert the array back to a matrix, and calculate the means for the ten digits, convert back to a three-dimensional array. Do exactly that. Also, compute and display the standard deviation images in exactly a similar way.

```r
zipMat <- matrix(scan("ziptrain.dat"),
                        nrow = 2000,
                        ncol = 256,
                        byrow = TRUE)

zipdigit <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/zipdigit.dat")

meanDigits <- sapply(X = 0:9,
                     function(x) {
                        digitDf <- zipMat[zipdigit == x, ]
                        # Select rows corresponding to the current digit
                        colMeans(digitDf)
                        }
                      )



# meanDigitsArr <- array(t(meanDigits), dim = c(16, 16, 10))
```

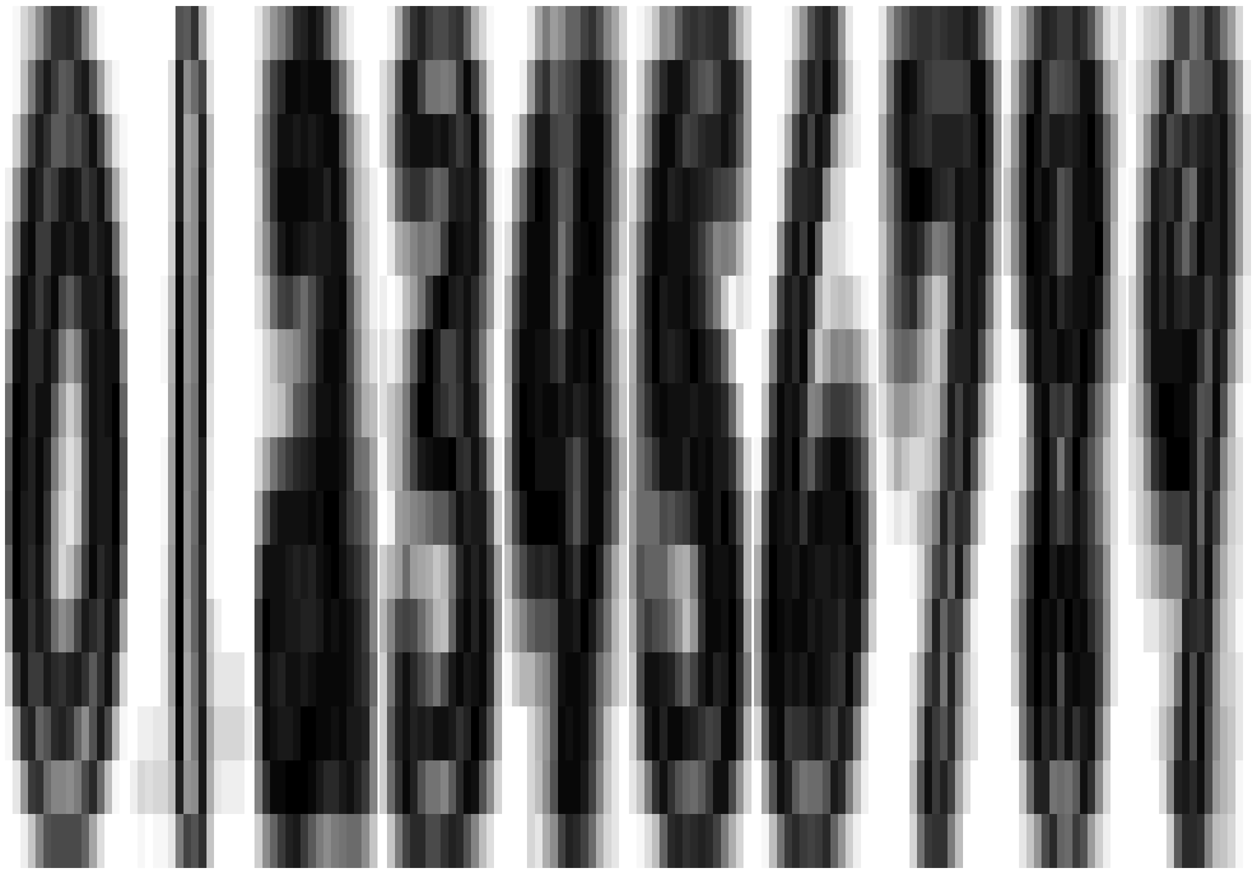
```
meanDigitsArr <- array(meanDigits, dim = c(16, 16, 10))
par(mfrow = c(1, 10), mar = rep(0.05, 4))

# for (x in 0:9) {
    # image(meanDigitsArr[,,x+1],
    # axes = FALSE,
    # col = gray(31:0/31))
# }

# mean images
apply(X = meanDigitsArr, MARGIN = 3, FUN = function(x) {
  image(x[, 16:1],
        axes = FALSE,
        col = gray(31:0/31))
})
```



```
## NULL
```

```
stdDigits <- sapply(X = 0:9,
                    FUN = function(x) {
                      digitDf <- zipMat[zipdigit == x, ]
                      apply(X = digitDf, MARGIN = 2, FUN = sd)
                      }
                    )
```

```r
# stdDigitsArr <- array(t(stdDigits), dim = c(16, 16, 10))
stdDigitsArr <- array(stdDigits, dim = c(16, 16, 10))

par(mfrow = c(1, 10), mar = rep(0.05, 4))

# for (x in 0:9) {
    # image(stdDigitsArr[,,x+1],
    # axes = FALSE,
    # col = gray(31:0/31))
# }

# std dev images
apply(X = stdDigitsArr, MARGIN = 3, FUN = function(x) {
  image(x[, 16:1],
        axes = FALSE,
        col = gray(31:0/31))
})
```



```
## NULL
```

```r
# x1 <- zipMat[zipdigit == 1,]
# x2 <- colMeans(x1)
# x3 <- array(t(x2), dim = c(16, 16, 1))
# image(x3[,,],
#         axes = FALSE,
#        col = gray(31:0/31)
#       )
```

```r
# x1 <- zipMat[zipdigit == 2,]
# x2 <- colMeans(x1)
# x3 <- array(t(x2), dim = c(16, 16, 1))
# image(x3[,,],
#        axes = FALSE,
#        col = gray(31:0/31)
#       )
```

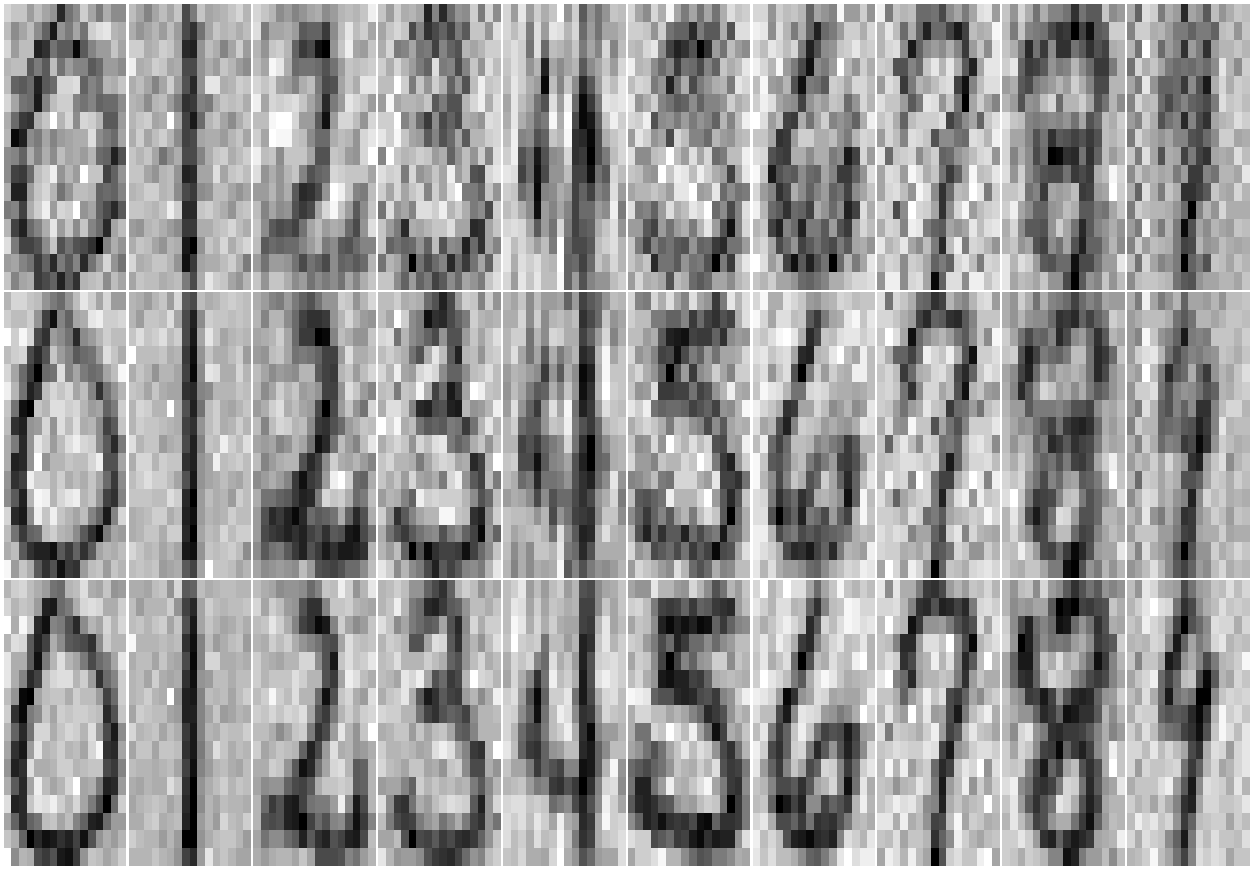
(d):

Our final act will involve cleaning up the dataset. To do so, remove the mean digit effect from each record. Then, perform a singular value decomposition on the 2000 × 256 matrix of deviations Y (that is the records with the correspoding digit mean effect removed). (Recall that the SVD is Y = U DV ’ and is obtained via the svd() function in R.) Replace D with a diagonal matrix Dk of the first k eigenvalues of D and the remainder as zero. Then let Yk = U DkV ’. Add back the mean and display the resulting images of all the digits for k = 25, 50, 75 in the same manner as (b)ii. Comment on the displays.

```r
ziptrain <- matrix(scan("ziptrain.dat"), ncol = 256, byrow = TRUE)
zipdigit <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/zipdigit.dat")

meanImages <- array(0, dim = c(256, 10))  # 256 pixels, 10 digits

# for (digit in 0:9) {
#   meanImages[, digit + 1] <- colMeans(ziptrain_data[zipdigit == digit, ])
# }

meanImages <- sapply(X = 0:9,
       FUN = function(x) {
         meanImages[, x + 1] <- colMeans(ziptrain[zipdigit == x, ])
         }
       )

deviationMat <- ziptrain

# I swear I cannot make this work using sapply
# this for loop is holding onto my sanity
for (digit in 0:9) {
  meanDigit <- matrix(meanImages[, digit + 1], nrow = 16, ncol = 16)
  deviationMat[zipdigit == digit, ] <-
    deviationMat[zipdigit == digit, ] - as.vector(t(meanDigit))
}

svdDf <- svd(deviationMat)

reconstruct_images <- function(u, d, v, k) {
  Dk <- diag(d[1:k])
  Yk <- u[, 1:k] %*% Dk %*% t(v[, 1:k])
  return(Yk)
}
```

```r
k_values <- c(25, 50, 75)
```

```r
# for (k in k_values) {
#   Yk <- reconstruct_images(svdDf$u, svdDf$d, svdDf$v, k)
#
#   for (digit in 0:9) {
#     meanDigit <- matrix(meanImages[, digit + 1], nrow = 16, ncol = 16)
#     reconstructedImg <- Yk[zipdigit == digit, ] + as.vector(t(meanDigit))
#     image(matrix(reconstructedImg[1, ], nrow = 16, ncol = 16)[, 16:1], axes = FALSE, col = gray(31:0/
#   }
# }

par(mfrow = c(length(k_values), 10), mar = rep(0.05, 4))

sapply(k_values[1:3],
       function(k) {
         Yk <- reconstruct_images(svdDf$u, svdDf$d, svdDf$v, k)
         sapply(0:9, function(x) {
           meanDigit <- matrix(meanImages[, x + 1], nrow = 16, ncol = 16)
           reconstructedImg <- Yk[zipdigit == x, ] + as.vector(t(meanDigit))
           image(matrix(reconstructedImg[1, ],
                        nrow = 16,
                        ncol = 16)[, 16:1],
                 axes = FALSE,
                 col = gray(31:0/31))
           }
         )
         }
       )
```



```
##       [,1] [,2] [,3]
##  [1,] NULL NULL NULL
##  [2,] NULL NULL NULL
##  [3,] NULL NULL NULL
##  [4,] NULL NULL NULL
##  [5,] NULL NULL NULL
##  [6,] NULL NULL NULL
##  [7,] NULL NULL NULL
##  [8,] NULL NULL NULL
##  [9,] NULL NULL NULL
## [10,] NULL NULL NULL
```

Comments

It appears the digits are more clearly delineated for larger values of k, though there is some “noise” surrounding the digits. Generally speaking, the digits displayed above are legible and would typically be understood in terms of which digit they denote, e.g. it is unlikely someone would confuse a “5” for a “6” in the above images, or vice versa.