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.

```
# library(readr)
# fbpImg <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS4/fbp-img.dat")

# senateVote <- read.table(file = "C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS3/senat
#                         sep="\t",
#                         quote="\"",
#                         header = TRUE)
```

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.

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

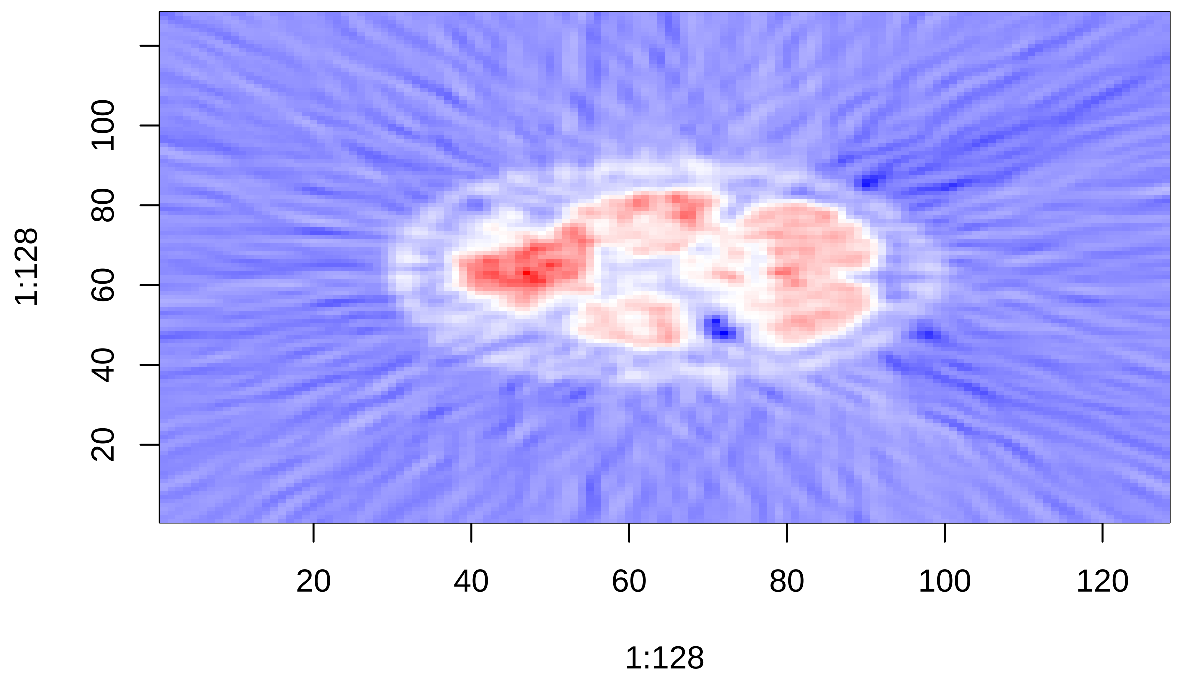
(b):

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

```
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)
```

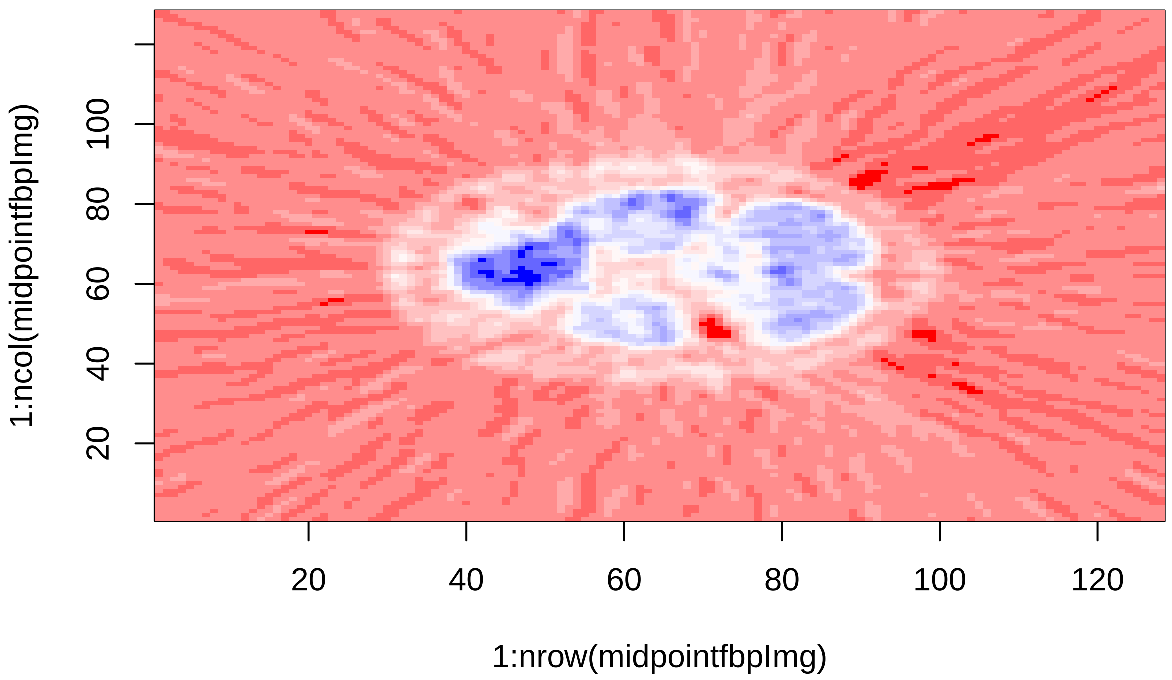
(c):

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)
```

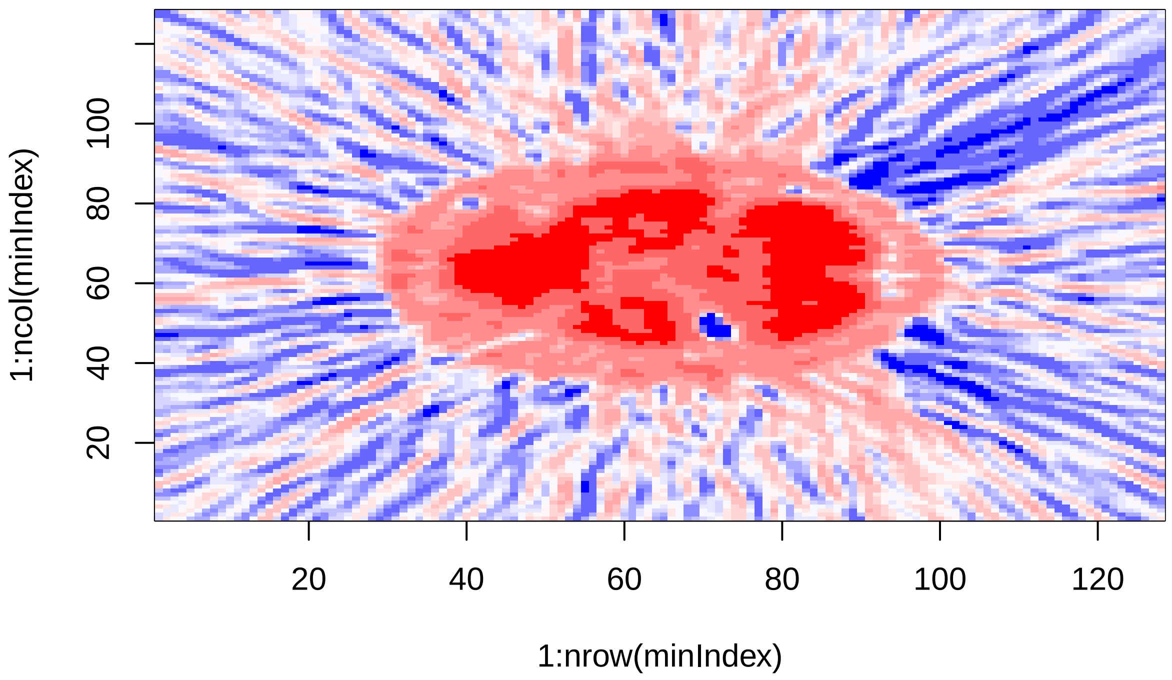
```r
# Calculate the midpoint of each bin
midpoints <- (breaks[-1] + breaks[-length(breaks)]) / 2

# 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.

```
# 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)
      )
```



```r
# 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.

Similar but different, neh, samurai?

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
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
# Step 3: Convert the data to a matrix, excluding the first column (gene probeset)
gene_matrix <- as.matrix(geneDf[, -1])

# Step 4: Reshape the matrix into a 3D array: genes x time points x replications
gene_array <- array(gene_matrix, dim = c(22810, 11, 2))

# Step 5: Calculate the mean abundance level at each time-point
mean_abundance <- apply(gene_array, c(1, 2), mean)
```

(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
mean_of_means <- apply(mean_abundance, 1, mean)
```

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
# Step 1: Create a matrix of dimension 22810 x 11 with each row being a copy of mean_of_means
mean_replicated_matrix <- matrix(mean_of_means, nrow = 22810, ncol = 11, byrow = TRUE)

# Step 2: Subtract the replicated mean matrix from the mean abundance matrix
mean_abundance_centered <- mean_abundance - mean_replicated_matrix
```

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
# Step 1: Calculate the standard deviation for each gene (row) in the centered abundance matrix
std_dev <- apply(mean_abundance_centered, 1, sd)

# Step 2: Create a matrix of the standard deviations with the same dimensions as mean_abundance_centere
std_dev_matrix <- matrix(rep(std_dev, each = 11), nrow = 22810, ncol = 11, byrow = TRUE)

# Step 3: Scale the measurements for each gene by dividing by the standard deviation
scaled_measurements <- mean_abundance_centered / std_dev_matrix
```

(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
micromeans <- geneDfMicro
```

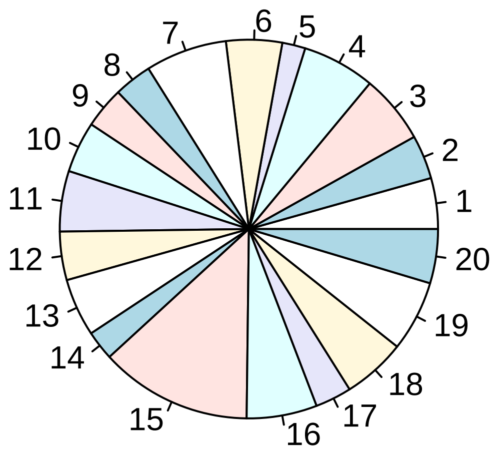
```r
# Step 3: Convert the data to a matrix
micromeans_matrix <- as.matrix(micromeans)

# Step 4: Calculate the standard deviation for each row (measurement) in the micromeans matrix
std_dev_micromeans <- apply(micromeans_matrix, 1, sd)

# Step 5: Create a matrix of the standard deviations with the same dimensions as micromeans_matrix
std_dev_micromeans_matrix <- matrix(rep(std_dev_micromeans, each = 11), nrow = 20, ncol = 11, byrow = T

# Step 6: Scale the measurements for each row by dividing by the standard deviation
scaled_micromeans <- micromeans_matrix / std_dev_micromeans_matrix
```

(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
# Step 1: Create a 3D array of mean abundance levels replicated for 20 means
mean_abundance_array <- array(rep(mean_abundance, each = 20), dim = c(22, 810, 20))

# Step 2: Create a 3D array of micromeans (mean vectors) replicated for each gene
# micromeans_array <- array(rep(micromeans_matrix, times = 22810), dim = c(20, 11))
micromeans_array <- array(rep(micromeans_matrix, times = 22810), dim = c(22810, 11, 20))

# Step 3: Calculate the Euclidean distances
# Initialize the distances array
euclidean_distances <- array(0, dim = c(22810, 20))  # Now only need to track gene and mean indices

# Loop through genes and means
for (gene_idx in 1:22810) {
  for (mean_idx in 1:20) {
    euclidean_distances[gene_idx, mean_idx] <- sqrt(sum((mean_abundance[gene_idx, ] - micromeans_matrix
  }
}

# Find the closest mean for each gene
closest_means_indices <- apply(euclidean_distances, 1, which.min)

# Tabulate the frequency distribution
frequency_distribution <- table(closest_means_indices)

# Display the frequencies using a pie chart
pie(frequency_distribution, main = "Frequency Distribution of Closest Means")
```

Frequency Distribution of Closest Means



Notes: Data Preparation:

Mean Abundance Matrix: Ensure you have a matrix (mean_abundance) with dimensions 22810 × 11 22810×11, where rows represent different genes and columns represent mean abundance levels across 11 time points. Micromeans Matrix: Define or load a matrix (micromeans_matrix) representing mean vectors for the 20 conditions or time points (e.g., dimensions 20 × 11 20×11).

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)
```

```
## [1] 2000  256
```

```
dim(t(ziptrain))
```

```
## [1]  256 2000
```

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

dim(ziptrain3D)
```

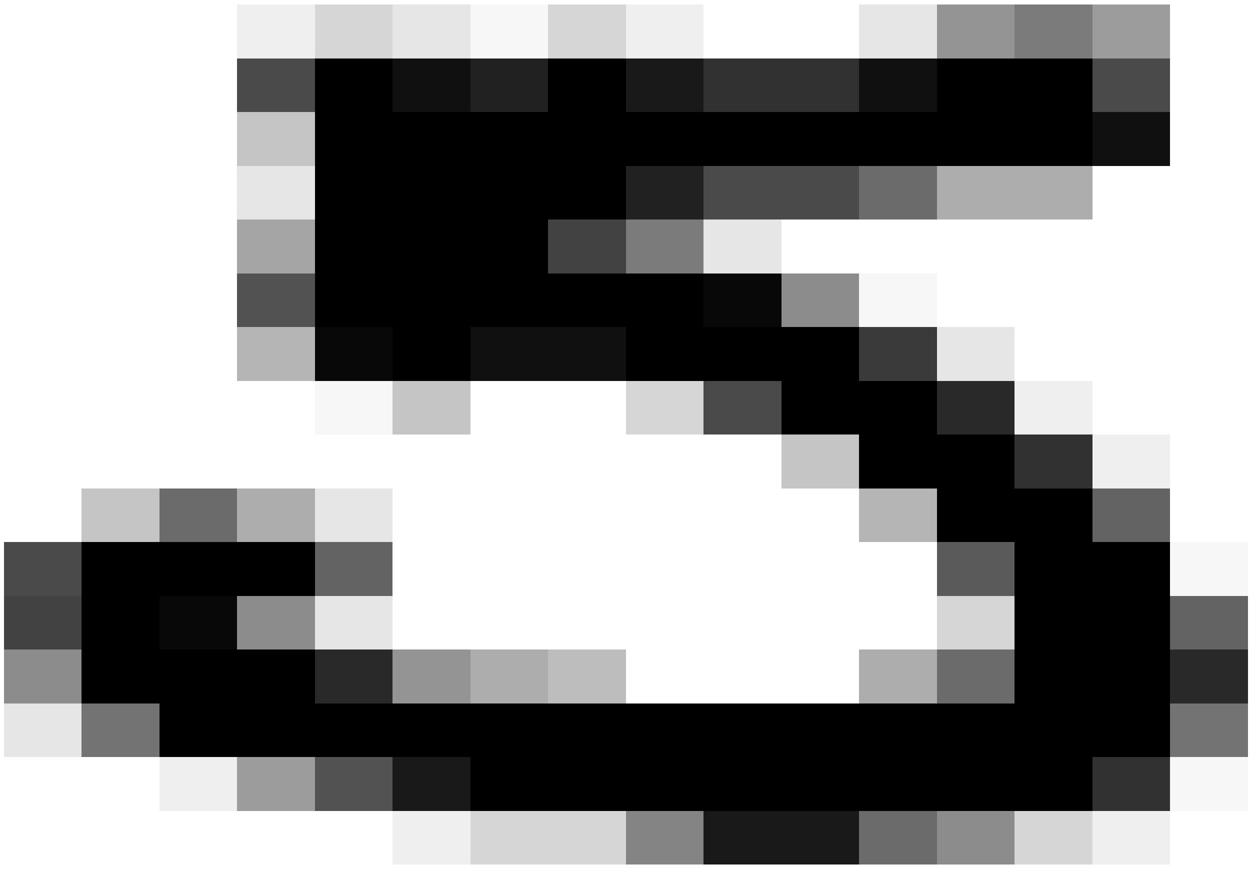
```
## [1]   16   16 2000
```

(b):

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

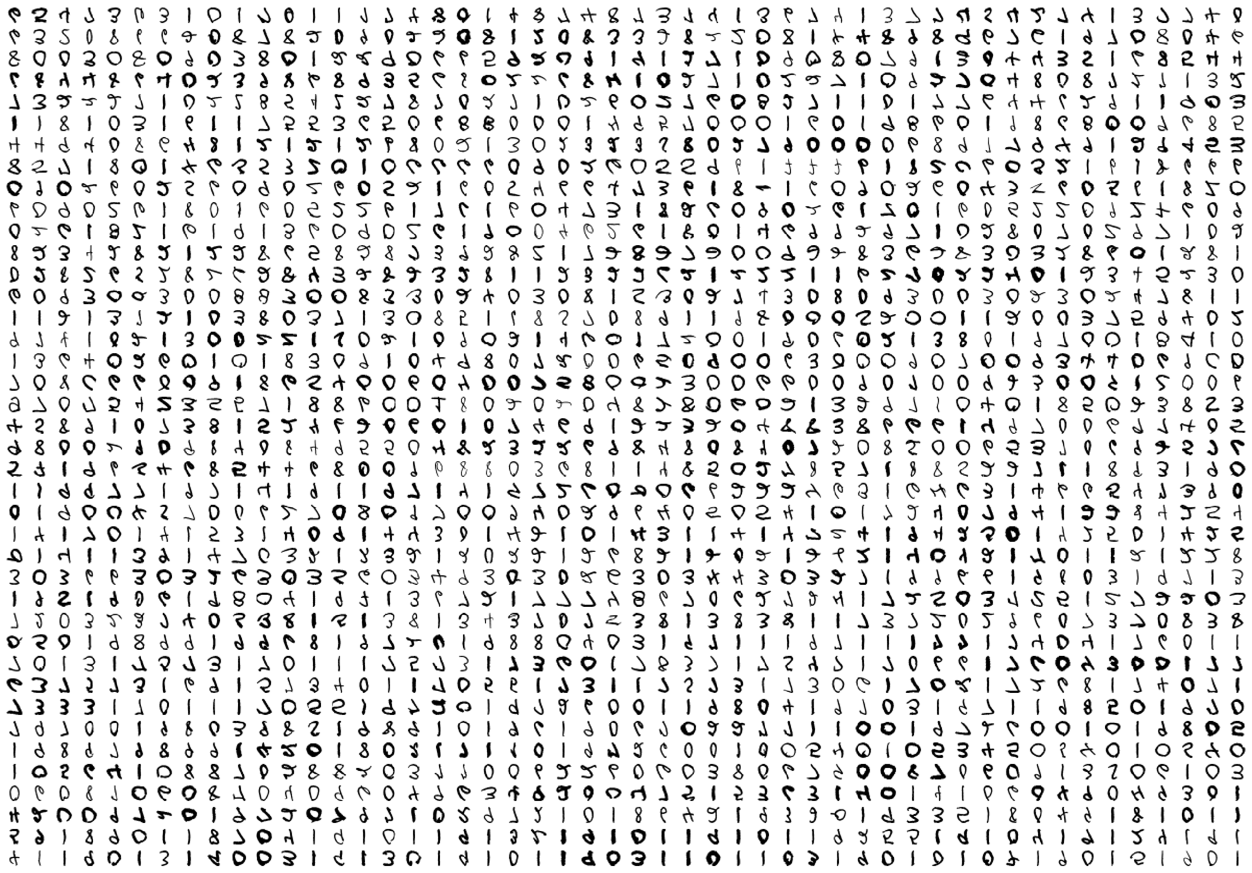
i. If we image the second record, we note that without any changes, the image mirrors the digit "5". Fix these by reversing the appropriate dimension of the array. Also note that the white space around the axes is wasted space and should be removed. To substantially reduce the white space, consider par(mar = 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),
    pin = c(5.2/64, 6.5/80)
    )
# , pin = c(5.2/50, 6.5/40))

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



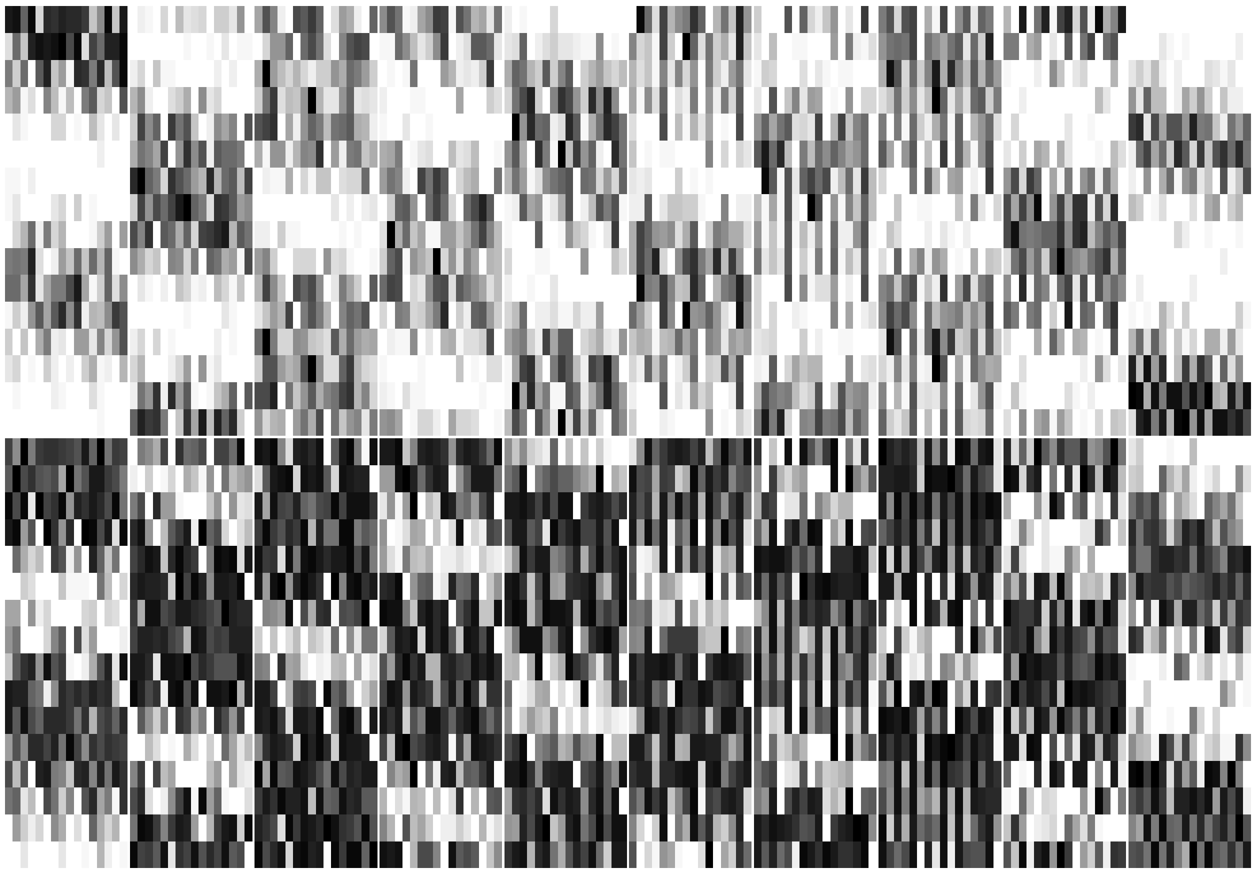
```
## NULL
```

(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.

```
ziptrain_data <- matrix(scan("ziptrain.dat"), ncol = 256, byrow = TRUE)

# Step 2: Reshape the dataset into a 3D array (16x16x2000)
ziptrain_array <- array(t(ziptrain_data), dim = c(16, 16, 2000))

ziptrain_matrix <- matrix(t(matrix(ziptrain_array)), nrow = 2000, ncol = 256, byrow = TRUE)
```

```
# Step 1: Read the pixel data from the file
ziptrain_data <- matrix(scan("ziptrain.dat"), ncol = 256, byrow = TRUE)  # Read in the 2000x256 pixel d

# Step 2: Use zipdigit as the digit labels
digit_labels <- zipdigit  # This is your 2000x1 matrix

mean_images <- sapply(0:9, function(digit) {
  digit_data <- ziptrain_data[digit_labels == digit, ]  # Select rows corresponding to the current digi
  colMeans(digit_data)
})

stddev_images <- sapply(0:9, function(digit) {
  digit_data <- ziptrain_data[digit_labels == digit, ]  # Select rows corresponding to the current digi
  apply(digit_data, 2, sd)
})

# Step 3: Reshape the results back into 16x16x10 arrays
mean_images_array <- array(t(mean_images), dim = c(16, 16, 10))
stddev_images_array <- array(t(stddev_images), dim = c(16, 16, 10))

# Step 4: Display the mean and standard deviation images
par(mfrow = c(2, 10), mar = rep(0.05, 4))

# Display mean images
for (digit in 0:9) {
  image(mean_images_array[,,digit+1], axes = FALSE, col = gray(31:0/31))
}

# Display standard deviation images
for (digit in 0:9) {
  image(stddev_images_array[,,digit+1], 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
# Assuming ziptrain_data and zipdigit are already defined

# Step 1: Compute mean images if not done yet
mean_images <- array(0, dim = c(256, 10))  # 256 pixels, 10 digits

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

# Step 2: Create a matrix to hold deviations
deviation_matrix <- ziptrain_data  # Start with the original data

# Step 3: Subtract the mean image for each digit
for (digit in 0:9) {
  mean_digit <- matrix(mean_images[, digit + 1], nrow = 16, ncol = 16)  # Reshape to 16x16
  deviation_matrix[zipdigit == digit, ] <-
    deviation_matrix[zipdigit == digit, ] - as.vector(t(mean_digit))
}

# Step 4: Perform Singular Value Decomposition
svd_result <- svd(deviation_matrix)

# Step 5: Define a function to reconstruct images using the first k singular values
reconstruct_images <- function(U, D, V, k) {
  Dk <- diag(D[1:k])  # Keep the first k singular values
  Yk <- U[, 1:k] %*% Dk %*% t(V[, 1:k])  # Ensure correct dimensions
  return(Yk)
}

# Step 6: Define k values to consider
k_values <- c(25, 50, 75)

# Step 7: Set up plotting layout for reconstructed images
par(mfrow = c(length(k_values), 10), mar = rep(0.05, 4))

# Step 8: Loop through k values and reconstruct images
for (k in k_values) {
  Yk <- reconstruct_images(svd_result$u, svd_result$d, svd_result$v, k)

  # Add back the mean for each digit and display the images
  for (digit in 0:9) {
    mean_digit <- matrix(mean_images[, digit + 1], nrow = 16, ncol = 16)  # Reshape to 16x16
    reconstructed_image <- Yk[zipdigit == digit, ] + as.vector(t(mean_digit))

    # Display the reconstructed image for this digit
    image(matrix(reconstructed_image[1, ], nrow = 16, ncol = 16)[, 16:1], axes = FALSE, col = gray(31:0
  }
}
```