DATA 605 Assignment 1

Amanda Fox

Load libraries:

```
# load libraries
library(tidyverse)
library(ggplot2)
library(patchwork)
library(jpeg)
library(RCurl)
```

1. Task: Create a simple shape (like a square or triangle) using point plots in R. Implement R code to apply different transformations (scaling, rotation, reflection) to the shape by left multiplying a transformation matrix by each of the point vectors. Demonstrate these transformations through animated plots.

To create a triangle, we use a 2x4 matrix: the first three columns represent three point vectors (x,y) and the fourth column repeats the first point vector to close the shape.

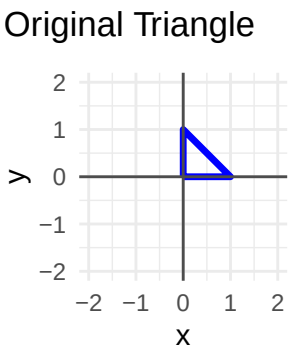
```
# create matrix defining triangle:
my_shape <- matrix(c(0,0,
                     1,0,
                     0,1,
                     0,0),
                   nrow = 2, byrow = TRUE)
my_shape
```

```
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    0
[2,]    0    1    0    0
```

```r
# prepare to plot: convert to df and label columns x & y
df_my_shape <- as.data.frame(t(my_shape))
colnames(df_my_shape) <- c("x","y")

df_my_shape
```

```
  x y
1 0 0
2 0 1
3 1 0
4 0 0
```

```r
# plot
plot_my_shape <- df_my_shape %>%
  ggplot(aes(x = x, y = y)) +
         geom_path(color = 'blue', linewidth = 1.2) +
         geom_hline(yintercept = 0, color = "gray30") +
         geom_vline(xintercept = 0, color = "gray30") +
         coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
         theme_minimal()

plot_my_shape + plot_annotation("Original Triangle")
```



Next, we can scale the shape by multiplying it by a transformation matrix. In this case, we set up our transformation matrix to multiply both x and y by 2:

```r
# create scaling transformation matrix multiplying x and y by 2
matrix_scale <- matrix(c(2,0,
                         0,2),
                       nrow = 2, byrow = TRUE)
matrix_scale
```

```
     [,1] [,2]
[1,]    2    0
[2,]    0    2
```

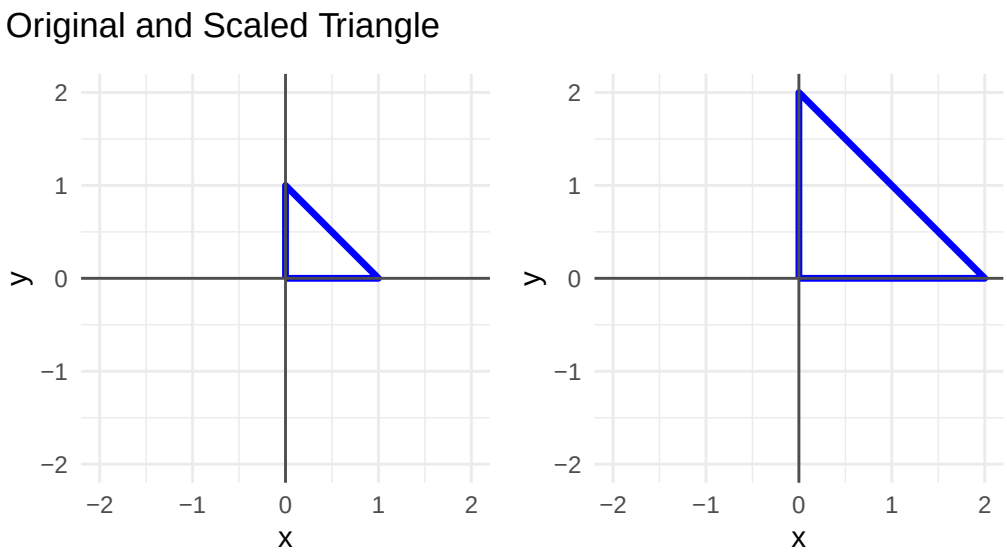
```
# multiply: transpose my_shape matrix so # rows match # columns
my_shape_scaled <- t(my_shape) %*% matrix_scale
my_shape_scaled
```

```
     [,1] [,2]
[1,]    0    0
[2,]    0    2
[3,]    2    0
[4,]    0    0
```

```
# convert to df with column names x,y
df_my_shape_scaled <- as.data.frame(my_shape_scaled)
colnames(df_my_shape_scaled) <- c("x","y")

df_my_shape_scaled
```

```
  x y
1 0 0
2 0 2
3 2 0
4 0 0
```

```
# plot
plot_my_shape_scaled <- df_my_shape_scaled %>%
  ggplot(aes(x = x, y = y)) +
  geom_path(color = 'blue', linewidth = 1.2) +
  geom_hline(yintercept = 0, color = "gray30") +
  geom_vline(xintercept = 0, color = "gray30") +
  coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
  theme_minimal()

plot_my_shape + plot_my_shape_scaled  + plot_annotation("Original and Scaled Triangle")
```



We can also use a transformation matrix to flip our shape over. In the example below, we create a transformation matrix to flip the shape over the x axis by keeping x values the same while multiplying y values by -1:

```
# create transformation matrix (x, y) → (x, -y)
matrix_flip <- matrix(c(1, 0,
                        0,-1),
                      nrow = 2, byrow = TRUE)
matrix_flip
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0   -1
```

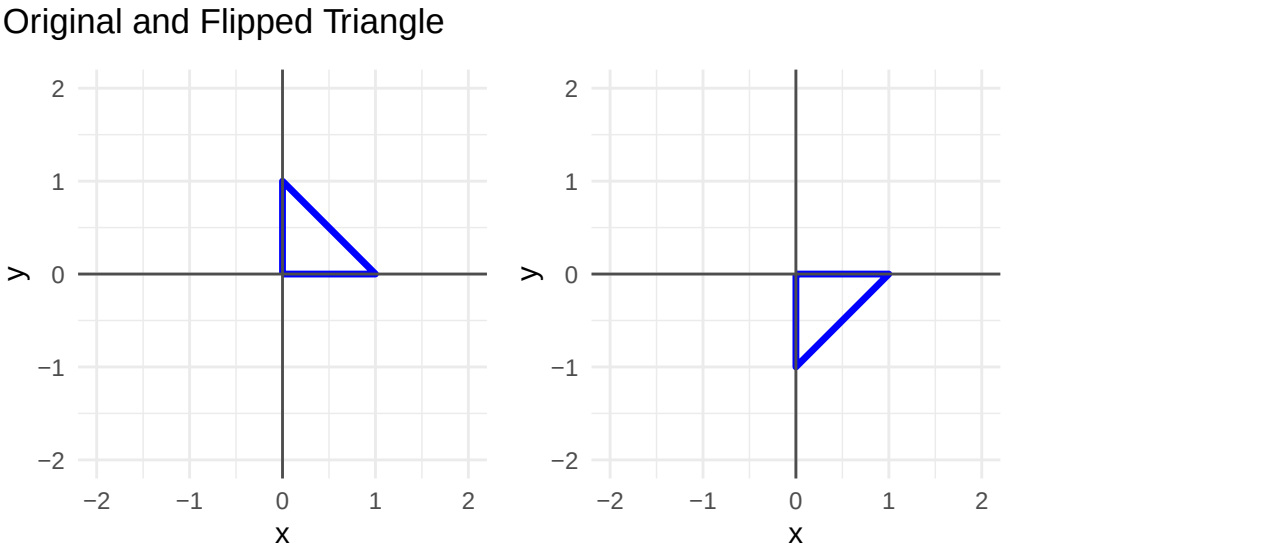
```
# multiply: transpose my_shape matrix so # rows match # columns
my_shape_flip <- t(my_shape) %*% matrix_flip
my_shape_flip
```

```
     [,1] [,2]
[1,]    0    0
[2,]    0   -1
[3,]    1    0
[4,]    0    0
```

```r
# convert to df with column names x,y
df_my_shape_flip <- as.data.frame(my_shape_flip)
colnames(df_my_shape_flip) <- c("x","y")

df_my_shape_flip
```

```
  x  y
1 0  0
2 0 -1
3 1  0
4 0  0
```

```r
# plot
plot_my_shape_flip <- df_my_shape_flip %>%
  ggplot(aes(x = x, y = y)) +
  geom_path(color = 'blue', linewidth = 1.2) +
  geom_hline(yintercept = 0, color = "gray30") +
  geom_vline(xintercept = 0, color = "gray30") +
  coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
  theme_minimal()

plot_my_shape + plot_my_shape_flip + plot_annotation("Original and Flipped Triangle")
```



Finally, we can use transformation matrices to rotate a shape. Below, we use a transformation matrix rotating our shape 90 degrees clockwise and repeat four times for a full 360 degrees:

```
# create transformation matrix (x, y) → (-y, x)
matrix_rotate_90 <- matrix(c(0,-1,
                             1, 0),
                           nrow = 2, byrow = TRUE)
matrix_rotate_90
```

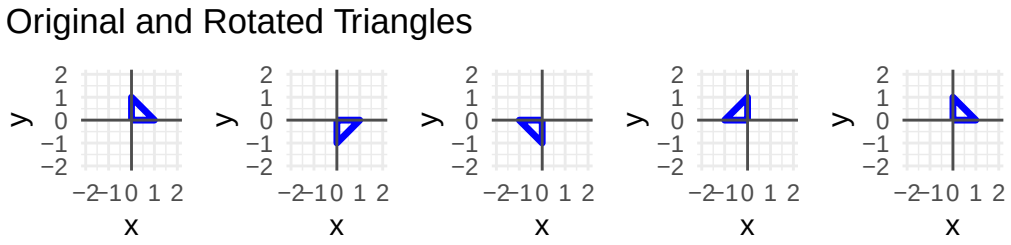
```
     [,1] [,2]
[1,]    0   -1
[2,]    1    0
```

```
# create list to hold plots for display
rotated_shapes <- list()
rotated_shapes[[1]] <- plot_my_shape

current_shape <- t(my_shape)
current_shape
```

```
     [,1] [,2]
[1,]    0    0
[2,]    0    1
[3,]    1    0
[4,]    0    0
```

```
# loop: multiply, convert to df, create and store plot

for (i in 2:5) {

  # multiply matrix
  current_shape <- current_shape %*% matrix_rotate_90
  current_shape

  # convert transformed matrix to df with column names x,y
  df_rotate_90 <- as.data.frame(current_shape)
  colnames(df_rotate_90) <- c("x","y")

  df_rotate_90

  # create and store plot
```

```
  rotated_shapes[[i]] <- df_rotate_90 %>%
    ggplot(aes(x = x, y = y)) +
    geom_path(color = 'blue', linewidth = 1.2) +
    geom_hline(yintercept = 0, color = "gray30") +
    geom_vline(xintercept = 0, color = "gray30") +
    coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
    theme_minimal()
  }

plot_display <- rotated_shapes[[1]] +
                rotated_shapes[[2]] +
                rotated_shapes[[3]] +
                rotated_shapes[[4]] +
                rotated_shapes[[5]]
plot_display +
  plot_layout(ncol = 5) +
  plot_annotation("Original and Rotated Triangles")
```



2. Proofs & SVD Example

A. Prove that $A \times B \neq B \times A$

```r
# --------------
# AB <> BA
# --------------

# define two 3x3 matrices A and B

A <- matrix(c(1,2,3,
              4,5,6,
              7,8,9),
            nrow = 3, byrow = TRUE)

B <- matrix(c(1, 0,-1,
              1, 2, 0,
              0,-1, 1),
            nrow = 3, byrow = TRUE)

A%*%B
```

```
     [,1] [,2] [,3]
[1,]    3    1    2
[2,]    9    4    2
[3,]   15    7    2
```

```r
B%*%A
```

```
     [,1] [,2] [,3]
[1,]   -6   -6   -6
[2,]    9   12   15
[3,]    3    3    3
```

B. Prove that $A^T * A$ is always symmetric.

The matrix $A^T \times A$ is always symmetric because by transposing any matrix A with dimensions $m \times n$, we get a matrix A^T with dimensions $n \times m$. This has two implications for multiplying $A^T \times A$:

1. The multiplication $(n \times m) \times (m \times n)$ always works because the number of columns in the first matrix (m) match the number of rows(m) in the second, which makes matrix multiplication possible.

2. The resulting matrix has dimensions $n \times n$, which is square and symmetric, because the dimensions are defined by the number of rows in the first matrix (n) and the number of columns in the second matrix (also n).

```
# Matrix A from above:
A
```

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

```
# Transpose of A:
t(A)
```

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

```
# Transpose of A times A:
t(A) %*% A
```

```
     [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

C. Prove that $det(A^T \times A)$ is non-negative

$A^T \times A$ results in a square matrix as shown above, so we can calculate a determinant.

We also know that $det(A^T \times A) = det(A)^2$, and also that, like any square, $det(A)^2$ must be ≥ 0.

Therefore, if $det(A^T \times A) = det(A)^2$ and $det(A)^2 \geq 0$ then $det(A^T \times A) \geq 0$.

```
# determinant of transpose of A times A:
det(t(A) %*% A)
```

```
[1] -1.534772e-12
```

D. Singular Value Decomposition (SVD) and Image Compression: Write an R function that performs Singular Value Decomposition (SVD) on a grayscale image (which can be represented as a matrix). Use this decomposition to compress the image by keeping only the top k singular values and their corresponding vectors. Demonstrate the effect of different values of k on the compressed image's quality.

This was a challenging but beneficial exercise demonstrating a practical application of matrix decomposition. I had never manipulated images before so R function help and online resources (ChatGPT for coding assistance; see Works Cited) were particularly helpful in problem-solving this exercise as I worked my way through understanding the data behind images and manipulating that data in R as matrices.

First, I used a black and white image that turned out to be encoded as RGB. I learned this is common for compatibility reasons, but all channels will have the same values for a grayscale image: this meant I could extract any channel in order to apply SVD with a simple R function.

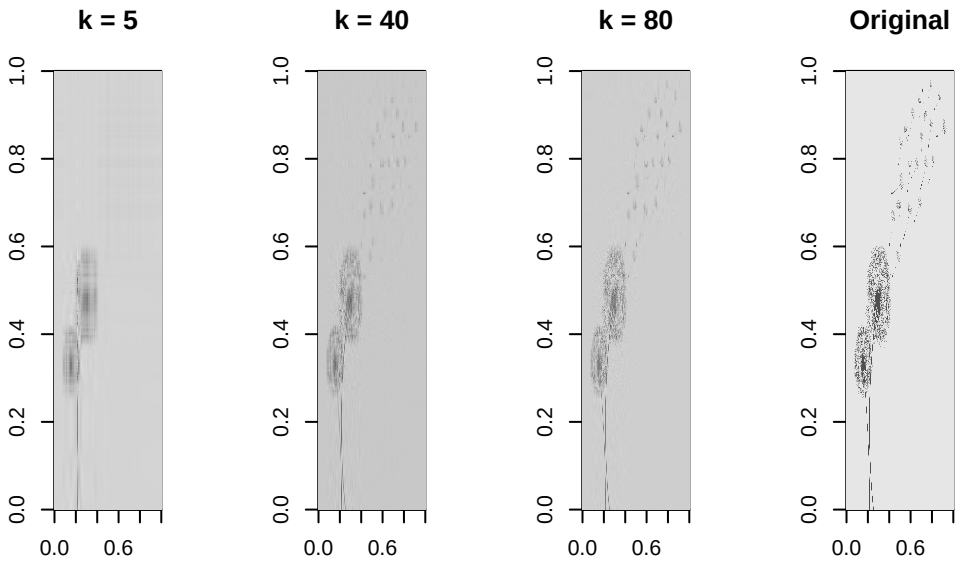
Then I extracted the components U, S, and V and created a function to reconstruct the image matrices for various values of k (ChatGPT). The first iteration resulted in appropriately compressed images but in shades of red and rotated -90 degrees. Changing to gray was easy but I found multiple explanations for rotating the images; a matrix approach was most helpful to my understanding in the context of this assignment. While I could not multiply by a transformation matrix directly, the earlier rotation exercise was helpful in understanding how to manipulate the x,y points as rows/columns in an image, while the image matrix expressed the intensity.

The resulting transformed images were compressed relative to the k value: low values for k (5, 40) showed significant loss of detail, while a higher value (80) was much closer to the original image, but still not as sharp.

```
# import image and convert to matrix

my_url <- "https://raw.githubusercontent.com/AmandaSFox/DATA605/main/OC1Y7T0.jpg"
my_image <- readJPEG(getURLContent(my_url, binary = TRUE))

# check if three channels (RGB) and if so, extract one channel to make it grayscale
if(length(dim(my_image)) == 3) {
  my_image <- my_image[, , 1]
  }

# validate image size and not three dimensions
dim(my_image)
```

```
[1] 1300 1300
```

```r
# apply SVD and display components
my_svd <- svd(my_image)

# Extract components
U <- my_svd$u
S <- diag(my_svd$d)
V <- my_svd$v

# Reconstruct image FUNCTION
reconstruct_image <- function(k) {
  U_k <- U[, 1:k]  # First k columns of U
  S_k <- S[1:k, 1:k]  # Top k singular values
  V_k <- V[, 1:k]  # First k columns of V
  img_k <- U_k %*% S_k %*% t(V_k)
  return(img_k)
}

# Substitute different k values and create three new matrices:
img_k5 <- reconstruct_image(5)
img_k40 <- reconstruct_image(30)
img_k80 <- reconstruct_image(80)

# create function to correct rotation:
rotate_90 <- function(image_matrix) {
  rotated_matrix <- t(image_matrix)[, ncol(image_matrix):1]
  return(rotated_matrix)
}

# setting to display all images in one row
par(mfrow = c(1,4))

# create four images, applying above rotation function, gray instead of red
my_image_k5 <- image(rotate_90(img_k5), col = gray.colors(256), main="k = 5")
my_image_k40 <- image(rotate_90(img_k40), col = gray.colors(256), main="k = 40")
my_image_k80 <- image(rotate_90(img_k80), col = gray.colors(256), main="k = 80")
my_image_orig <- image(rotate_90(my_image), col = gray.colors(256), main="Original")
```



Works Cited

Beezer, Robert. "A First Course in Linear Algebra." Open Textbook Library, 2015, open.umn.edu/opentextbooks/textbooks/5. Accessed Feb. 2025. ChatGPT. "ChatGPT." Chatgpt.com, 2024, chatgpt.com. Accessed Feb. 2025. Occasional R coding and LaTeX assistance. Layerace. Image by Layerace on Freepik. www.freepik.com/free-vector/dandelion-background-design_921206.htm. Accessed 18 Feb. 2025.