Geometric Transformation of Shapes and Image Processing

Umer Farooq

2024-09-14

1. Geometric Transformation of Shapes Using Matrix Multiplication

Context: In computer graphics and data visualization, geometric transformations are fundamental. These transformations, such as translation, scaling, rotation, and reflection, can be applied to shapes to manipulate their appearance.

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.

Create a Shape: Define a simple shape (e.g., a square) using a set of point coordinates.

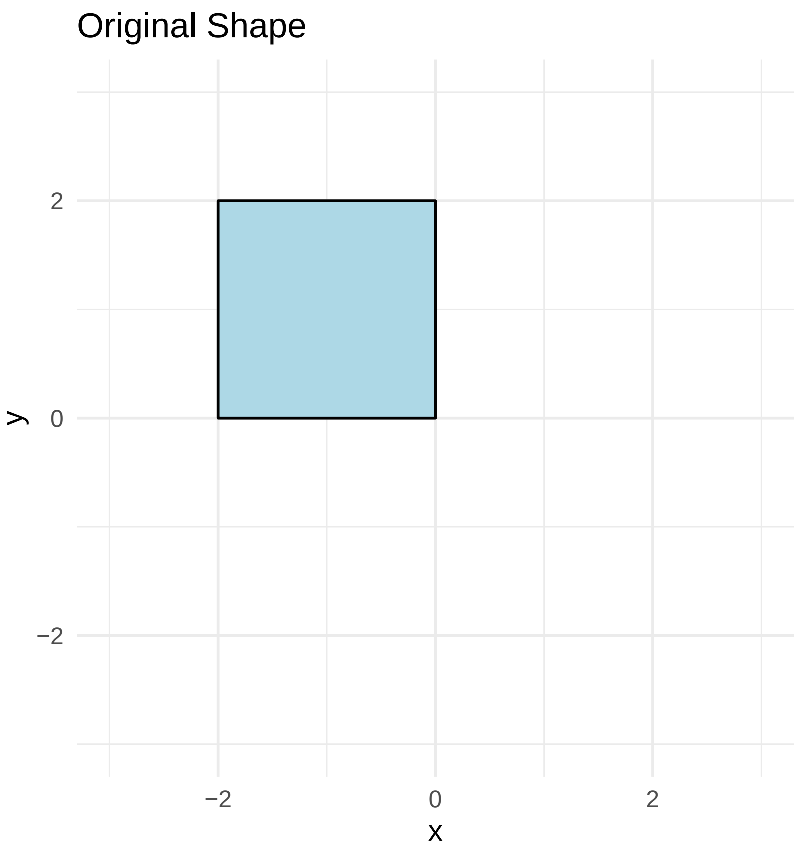
Apply Transformations:

Scaling: Enlarge or shrink the shape. Rotation: Rotate the shape by a given angle. Reflection: Reflect the shape across an axis. Animate Transformations: Use a loop to incrementally change the transformation matrix and visualize the effect on the shape over time.

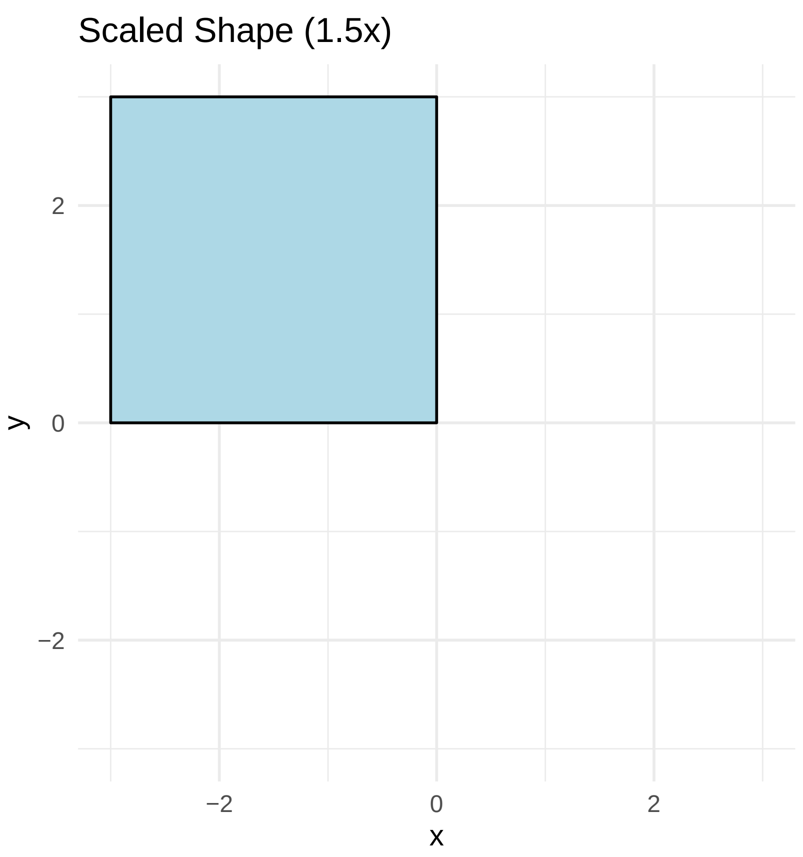
Plot: Display the original shape and its transformations in your compiled pdf. Demonstrate the effect of the transformations with fixed images

Answer:

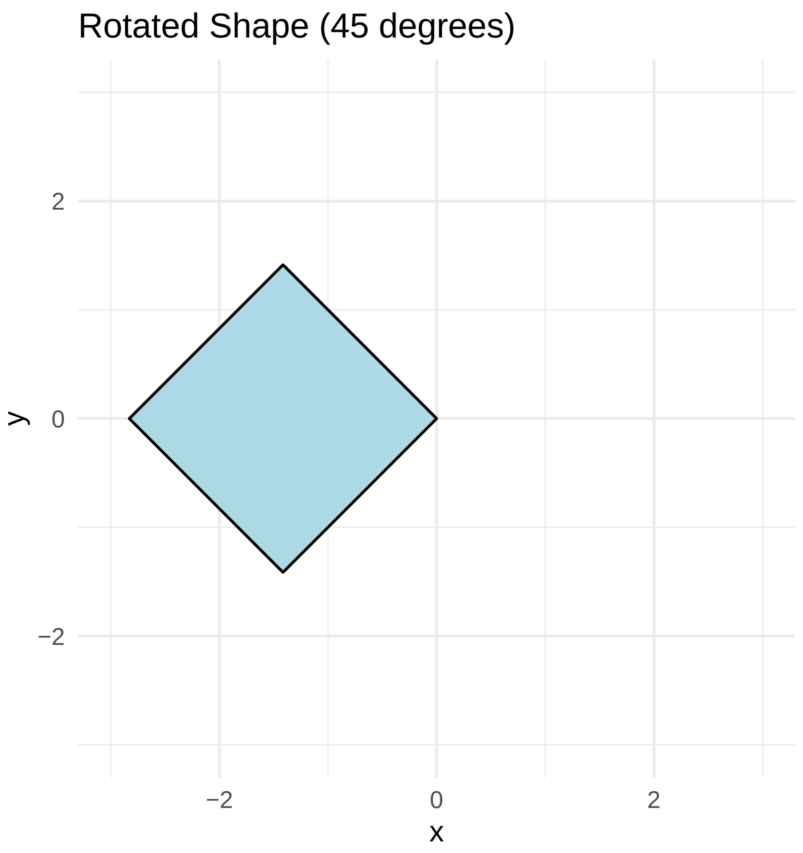
```r
# Plotting and defining square
square <- matrix(c(-2, 0, 0, 0, 0, 2, -2, 2), ncol = 2, byrow = TRUE)
plot_shape <- function(shape, title = "Shape") {
  shape_df <- data.frame(x = shape[, 1], y = shape[, 2])
  shape_df <- rbind(shape_df, shape_df[1, ])
  ggplot(shape_df, aes(x, y)) +
    geom_polygon(fill = 'lightblue', color = 'black') +
    xlim(-3, 3) + ylim(-3, 3) + coord_fixed() +
    ggtitle(title) +
    theme_minimal()
}

# Applying transformations
# Scaling
scaling_matrix <- function(sx, sy) {
  matrix(c(sx, 0, 0, sy), nrow = 2, byrow = TRUE)
}
# Rotation
rotation_matrix <- function(angle) {
  theta <- angle * pi / 180
  matrix(c(cos(theta), -sin(theta), sin(theta), cos(theta)), nrow = 2, byrow = TRUE)
}
# Reflection
reflection_matrix <- matrix(c(1, 0, 0, -1), nrow = 2)

# Transformation
apply_transformation <- function(shape, transformation_matrix) {
  t(transformation_matrix %*% t(shape))
}

# Creating an animated plot
frames <- list()
for (i in seq(0.5, 2, length.out = 10)) {
  scaled_shape <- apply_transformation(square, scaling_matrix(i, i))
  frames[[length(frames) + 1]] <- plot_shape(scaled_shape, title = paste("Scaling: ", round(i, 2)))
}
animation_plot <- frames[[1]]
for (i in 2:length(frames)) {
  animation_plot <- animation_plot + transition_states(i, transition_length = 1, state_length = 1)
}

plot_shape(square, "Original Shape")
```



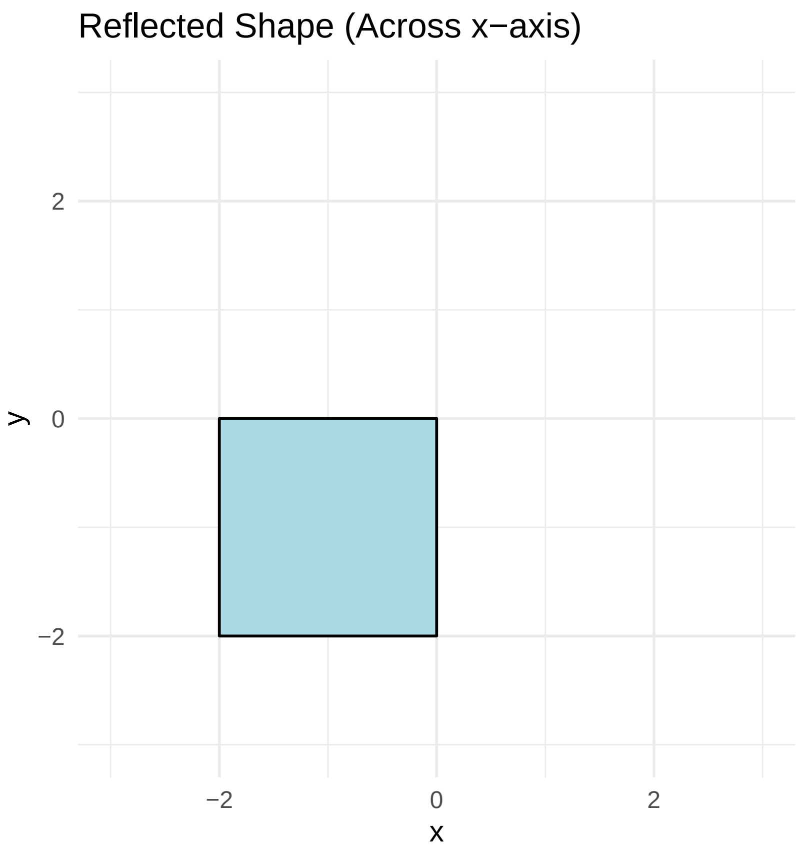
```r
scaled_shape <- apply_transformation(square, scaling_matrix(1.5, 1.5))
plot_shape(scaled_shape, "Scaled Shape (1.5x)")
```



```r
rotated_shape <- apply_transformation(square, rotation_matrix(45))
plot_shape(rotated_shape, "Rotated Shape (45 degrees)")
```



```
reflected_shape <- apply_transformation(square, reflection_matrix)
plot_shape(reflected_shape, "Reflected Shape (Across x-axis)")
```



The below code animate the scaling of the square but for the sake of rendering to PDF format the we will set the `eval` to FALSE in the beginning of code chunk which will not let the rendering to run our code for the PDF. We can follow the same process for rotation and reflection.

```r
square <- matrix(c(-1, -1, 1, -1, 1, 1, -1, 1), ncol = 2, byrow = TRUE)

# Convert the square points into a data frame for easier manipulation
square_df <- data.frame(x = square[, 1], y = square[, 2], state = "Original")

# Function to create a scaling matrix
scaling_matrix <- function(sx, sy) {
  matrix(c(sx, 0, 0, sy), ncol = 2, byrow = TRUE)
}

# Function to apply transformation to the shape
apply_transformation <- function(shape, transformation_matrix) {
  shape %*% transformation_matrix
}

# Create a scaling sequence
scaling_frames <- lapply(seq(0.5, 2, length.out = 10), function(scale) {
  scaled_shape <- apply_transformation(square, scaling_matrix(scale, scale))
  data.frame(x = scaled_shape[, 1], y = scaled_shape[, 2], state = paste("Scale", round(scale, 2)))
})

# 2. Rotation Animation (rotating from 0 to 45 degrees)
```

```r
rotation_frames <- lapply(seq(0, 45, length.out = 10), function(angle) {
  rotated_shape <- apply_transformation(square, rotation_matrix(angle))
  data.frame(x = rotated_shape[, 1], y = rotated_shape[, 2], state = paste("Rotation", round(angle, 2),
})

# 3. Reflection Animation (reflecting across y-axis)
reflection_frame <- apply_transformation(square, reflection_matrix)
reflection_df <- data.frame(x = reflection_frame[, 1], y = reflection_frame[, 2], state = "Reflected ac

# Combine all frames for animation
animation_data <- do.call(rbind, c(list(square_df), scaling_frames, rotation_frames, list(reflection_df

# Closing the polygon by repeating the first point for each state
animation_data <- animation_data[order(animation_data$state),]
animation_data <- rbind(animation_data, animation_data[1:4, ])  # Closing the original shape for the fi

# Create the combined animation
p <- ggplot(animation_data, aes(x = x, y = y, group = state)) +
  geom_polygon(aes(fill = state), color = "black") +
  coord_fixed() +
  xlim(-3, 3) + ylim(-3, 3) +
  labs(title = 'Transformation: {closest_state}') +
  theme_minimal() +
  transition_states(state, transition_length = 2, state_length = 1) +
  ease_aes('linear')

# Saving the animation as a GIF
animate(p, width = 500, height = 500, duration = 10, fps = 20, output = gif_file)
```

---

2. Matrix Properties and Decomposition

a) Proof that $AB \neq BA$

In general, matrix multiplication is **not commutative**, meaning $AB \neq BA$ for arbitrary matrices A and B. To show this, consider:

Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$. The product AB results in a matrix $C \in \mathbb{R}^{m\times p}$, where each element of C is defined as:

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$$

For the reverse product BA, assuming $B \in \mathbb{R}^{n\times p}$ and $A \in \mathbb{R}^{p\times m}$, the result is a matrix $D \in \mathbb{R}^{n\times m}$:

$$D_{ij} = \sum_{k=1}^{p} B_{ik}A_{kj}$$

Since the dimensions and ordering of multiplication differ, $AB \neq BA$ in most cases unless A and B commute, which is rare.

Example

Consider the matrices:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Calculating AB and BA:

$$AB = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$BA = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 23 & 34 \\ 31 & 46 \end{bmatrix}$$

Clearly, $AB \neq BA$.

---

b) Prove that $A^T A$ is always symmetric.

Given any matrix $A \in \mathbb{R}^{m \times n}$, the matrix $A^T A$ is always symmetric. To prove this, recall the definition of symmetry: a matrix M is symmetric if $M = M^T$.

Now consider $A^T A$:

$$(A^T A)^T = A^T (A^T)^T = A^T A$$

Since $(A^T)^T = A$, we have $(A^T A)^T = A^T A$, proving that $A^T A$ is symmetric.

---

c) Proof that $\det(A^T A) \geq 0$

Let $A \in \mathbb{R}^{m \times n}$. We aim to show that the determinant of $A^T A$ is non-negative. To do this, we use the fact that for any matrix A, $A^T A$ is **positive semi-definite**, meaning that for any non-zero vector $x \in \mathbb{R}^n$, we have:

$$x^T A^T A x = (Ax)^T (Ax) = \|Ax\|^2 \geq 0$$

Since $A^T A$ is positive semi-definite, its eigenvalues are non-negative. The determinant of a matrix is the product of its eigenvalues, and because all eigenvalues are non-negative, we conclude that:

$$\det(A^T A) \geq 0$$

This completes the proof.

---

d) Singular Value Decomposition and Image Compression

Singular Value Decomposition (SVD) allows us to decompose a matrix $A \in \mathbb{R}^{m \times n}$ as:

Task: 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. You can choose any open-access grayscale image that is appropriate for a professional program.

Instructions:

- Read an Image: Convert a grayscale image into a matrix.
- Perform SVD: Factorize the image matrix A into $U\Sigma V^T$ using R's built-in svd() function.
- Compress the Image: Reconstruct the image using only the top k singular values and vectors.
- Visualize the Result: Plot the original image alongside the compressed versions for various values of k (e.g., k = 5, 20, 50).

```r
image_path <- "/Users/umerfarooq/Desktop/Data 605/Homework 1/downloaded_image.jpeg"
```

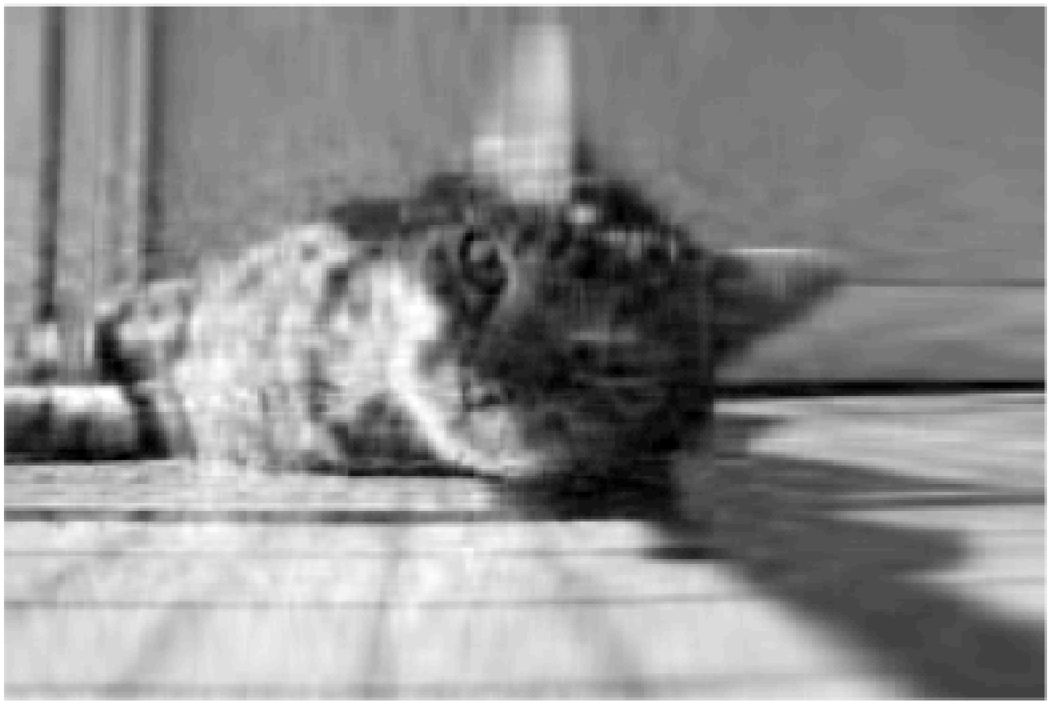
```r
library(imager)

# Function to rotate the matrix 90 degrees clockwise
rotate <- function(img_matrix) {
  t(apply(img_matrix, 2, rev))
}

compress_image_svd <- function(image_path, k_values) {
  img <- load.image(image_path)
  img_gray <- grayscale(img)
  img_matrix <- as.matrix(img_gray)

  rotated_original <- rotate(img_matrix)
  png("original_image.png")
  plot(as.raster(rotated_original), main="Original Image")
  dev.off()

  svd_result <- svd(img_matrix)
  U <- svd_result$u
  D <- diag(svd_result$d)
  V <- svd_result$v
  dev.new()
  par(mfrow=c(1, length(k_values) + 1), mar=c(2, 2, 2, 2))
  rotated_original <- rotate(img_matrix)
  plot(as.raster(rotated_original), main="Original Image")

  for (k in k_values) {
    D_k <- D
    D_k[(k+1):nrow(D_k), (k+1):ncol(D_k)] <- 0
    img_compressed <- U %*% D_k %*% t(V)
    img_compressed[img_compressed < 0] <- 0
    img_compressed[img_compressed > 1] <- 1
    rotated_compressed <- rotate(img_compressed)
    # Dynamically create filename with the k value
    file_name <- paste0("compressed_image_K_", k, ".png")

    # Save the image to a PNG file
    png(file_name)
    plot(as.raster(rotated_compressed), main=paste("Compressed Image (k =", k, ")"))
    dev.off()  # Close the PNG device
  }
}

k_values <- c(1, 5, 10, 20)  # Different values of k for compression
compress_image_svd(image_path, k_values)
```

Orginal Image:

```
knitr::include_graphics(c("original_image.png"))
```

K = 20:

```
knitr::include_graphics(c("compressed_image_k_20.png"))
```



K = 10:

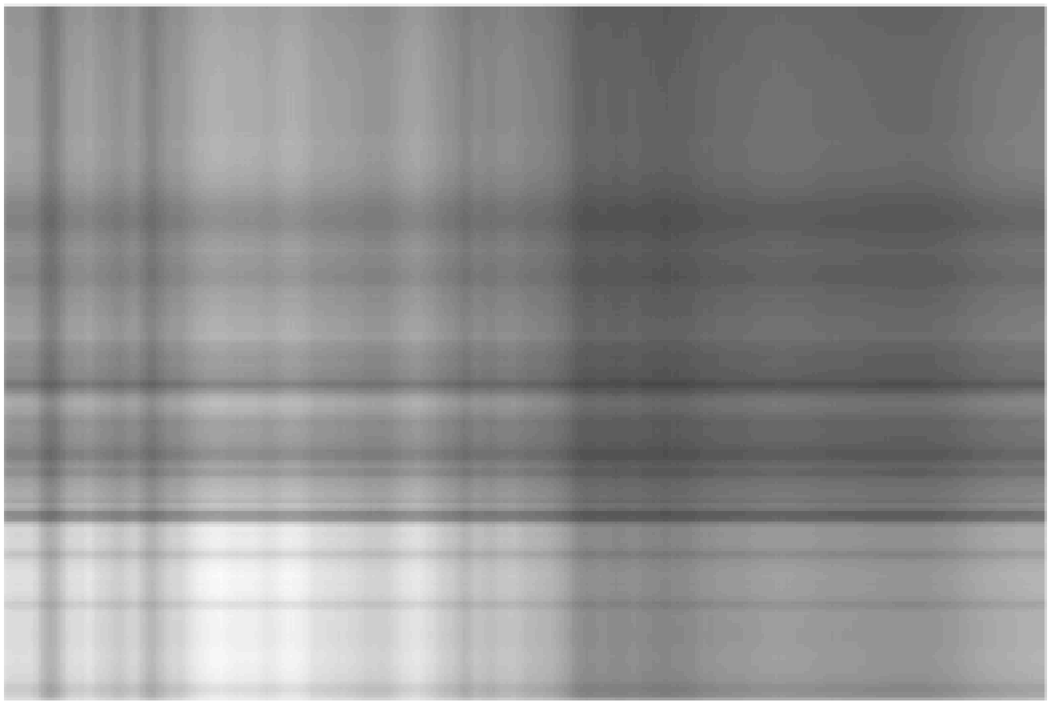
```
knitr::include_graphics(c( "compressed_image_k_10.png"))
```

K = 5:

```
knitr::include_graphics(c("compressed_image_k_5.png"))
```

K = 1:

```
knitr::include_graphics(c("compressed_image_k_1.png"))
```



---

3. Matrix Rank, Properties, and Eigenspace

Determine the Rank of the Given Matrix:

Find the rank of the matrix A. Explain what the rank tells us about the linear independence of the rows and columns of matrix A. Identify if there are any linear dependencies among the rows or columns.

$$A = \begin{bmatrix} 2 & 4 & 1 & 3 \\ -2 & -3 & 4 & 1 \\ 5 & 6 & 2 & 8 \\ -1 & -2 & 3 & 7 \end{bmatrix}$$

```r
A <- matrix(c(2,-2,5,-1,
              4,-4,6,-2,
              1,4,2,3,
              3,1,8,7), nrow = 4, ncol = 4)
```

```r
rref(A)
```

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

The rank(A) = 4 because there are 4 linear independent column or using the definition for the book dimension of the column space is 4. There are no linear dependencies among the rows or columns of this matrix because the rank is equal to the number of rows and columns (i.e., 4). This means the matrix has full row and column rank, indicating no linear dependencies.

Matrix Rank Boundaries:

Given an m x n matrix where m > n, determine the maximum and minimum possible rank, assuming that the matrix is non-zero.

Given an m×n matrix A where m > n, the rank of a matrix is the number of linearly independent rows or columns. The rank provides insight into the number of independent equations or relationships the matrix represents.

Maximum Possible Rank: The maximum rank of an m × n matrix is the smaller of the two dimensions,min(m,n). Since m > n, the maximum rank is n, meaning that the matrix can have n linearly independent columns. Reason: There can be at most n independent vectors in the column space or row space of the matrix. Minimum Possible Rank: The minimum rank of a non-zero matrix is 1. This occurs when the matrix has exactly one non-zero row or column. Reason: A non-zero matrix cannot have a rank of 0, as that would imply all rows and columns are zero.

Prove that the rank of a matrix equals the dimension of its row space (or column space). Provide an example to illustrate the concept.

The rank of a matrix is defined as the number of linearly independent rows or columns in the matrix. To prove that the rank of a matrix equals the dimension of its row space (or column space), consider the following:

Definition:

Row space: The span of all row vectors of the matrix. Column space: The span of all column vectors of the matrix. Proof:

1. Consider an m×n matrix A.
2. Perform Gaussian elimination (or row reduction) on A, transforming it into row echelon form or reduced row echelon form (RREF).

   - The row reduction does not change the row space of the matrix, but it simplifies the matrix so that we can easily identify the linearly independent rows.
   - The number of non-zero rows in RREF is equal to the rank of the matrix.

3. Since row reduction preserves the linear dependencies among the rows, the number of non-zero rows in the RREF form corresponds to the number of linearly independent rows in the original matrix.

   - This number is exactly the dimension of the row space.

4. Therefore, the rank of the matrix is the number of linearly independent rows, which is the dimension of the row space.

5. A similar argument holds for the column space. Using the fact that the rank of a matrix is invariant under transposition (i.e.,rank(A)=rank(A^T)), the rank also equals the dimension of the column space.

Thus, the rank of a matrix is equal to the dimension of its row space or column space.

Example: 3×2 Matrix:

Consider the following 3×2 matrix:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 5 & 10 \end{bmatrix}$$

Step 1: Perform row reduction to find the rank. Subtract multiples of the first row from the second and third rows to eliminate the entries below the first pivot.

$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The resulting matrix has only one non-zero row. Hence, the rank of A is 1.

Step 2: The dimension of the row space is the number of linearly independent rows. In this case, there is one linearly independent row.

Step 3: The dimension of the column space is also 1, as there is only one linearly independent column.

Thus, the rank of A is 1, which is equal to both the dimension of its row space and the dimension of its column space.

Rank and Row Reduction:

Determine the rank of matrix B. Perform a row reduction on matrix B and describe how it helps in finding the rank. Discuss any special properties of matrix B (e.g., is it a rank-deficient matrix?).

$$B = \begin{bmatrix} 2 & 5 & 7 \\ 4 & 10 & 14 \\ 1 & 2.5 & 3.5 \end{bmatrix}$$

```r
B <- matrix(c(2,4,1,5,10,2.5,7,14,3.5), nrow = 3, ncol = 3)
```

```r
rref(B)
```

```
##      [,1] [,2] [,3]
## [1,]    1  2.5  3.5
## [2,]    0  0.0  0.0
## [3,]    0  0.0  0.0
```

The rank of matrix B is 1, as determined through row reduction. This indicates that matrix B is rank-deficient, meaning its rank is less than the number of its rows and columns. Specifically, matrix B has 3 rows and 3 columns, but only one of the rows is linearly independent. The second and third rows are scalar multiples of the first row: the second row is 2× the first row, and the third row is 0.5× the first row. These linear dependencies among the rows show that the matrix does not have full rank, resulting in a rank of 1. Therefore, matrix B has a low rank, with significant redundancy in its rows and columns, which directly impacts its ability to span a higher-dimensional space.

Eigenvalues and Eigenvectors of Matrix A

Given matrix A:

$$A = \begin{pmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{pmatrix}$$

Step 1: Characteristic Polynomial

The eigenvalues are the solutions to the characteristic equation:

$$\det(A - \lambda I) = 0$$

First, find $A - \lambda I$:

$$A - \lambda I = \begin{pmatrix} 3-\lambda & 1 & 2 \\ 0 & 5-\lambda & 4 \\ 0 & 0 & 2-\lambda \end{pmatrix}$$

Now, calculate the determinant:

$$\det(A - \lambda I) = (3-\lambda)(5-\lambda)(2-\lambda)$$

So the characteristic equation is:

$$(3-\lambda)(5-\lambda)(2-\lambda) = 0$$

Step 2: Eigenvalues

Solving the characteristic equation, we find the eigenvalues:

$$\lambda_1 = 3, \quad \lambda_2 = 5, \quad \lambda_3 = 2$$

Step 3: Eigenvectors

Eigenvector for $\lambda_1 = 3$:

Solve $(A - 3I)\mathbf{v} = 0$:

$$A - 3I = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 0 & -1 \end{pmatrix}$$

Eigenvector for $\lambda_1 = 3$:

$$\mathbf{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Eigenvector for $\lambda_2 = 5$:

$$\mathbf{v}_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$$

Eigenvector for $\lambda_3 = 2$:

$$\mathbf{v}_3 = \begin{pmatrix} \frac{1}{3} \\ -\frac{4}{3} \\ 1 \end{pmatrix}$$

Step 4: Verify Linear Independence

The eigenvectors are linearly independent because the determinant of the matrix formed by the eigenvectors is non-zero:

$$\det(V) = 2$$

Diagonalization of Matrix:

- Determine if matrix A can be diagonalized. If it can, find the diagonal matrix and the matrix of eigenvectors that diagonalizes A.

To determine if matrix \(A \) can be diagonalized, we first compute its eigenvalues and eigenvectors.

Consider the matrix A:

$$A = \begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix}$$

To determine if matrix A can be diagonalized, we need to find its eigenvalues and eigenvectors.

Step 1: Characteristic Polynomial

The characteristic polynomial is obtained by solving the determinant of $A - \lambda I$, where λ is the eigenvalue and I is the identity matrix.

$$A - \lambda I = \begin{bmatrix} 3-\lambda & 1 & 2 \\ 0 & 5-\lambda & 4 \\ 0 & 0 & 2-\lambda \end{bmatrix}$$

The determinant is given by:

$$\det(A - \lambda I) = (3-\lambda)(5-\lambda)(2-\lambda)$$

Thus, the characteristic equation is:

$$(3-\lambda)(5-\lambda)(2-\lambda) = 0$$

Step 2: Eigenvalues

The solutions to the characteristic equation give the eigenvalues:

$$\lambda_1 = 3, \quad \lambda_2 = 5, \quad \lambda_3 = 2$$

Since the matrix has 3 distinct eigenvalues, it can be diagonalized.

Step 3: Finding Eigenvectors

For each eigenvalue λ, solve $(A - \lambda I)v = 0$ to find the corresponding eigenvector.

Eigenvector for $\lambda_1 = 3$

$$(A - 3I)v_1 = 0 \quad \Rightarrow \quad \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 0 & -1 \end{bmatrix} v_1 = 0$$

This gives the eigenvector:

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Eigenvector for $\lambda_2 = 5$

$$(A - 5I)v_2 = 0 \quad \Rightarrow \quad \begin{bmatrix} -2 & 1 & 2 \\ 0 & 0 & 4 \\ 0 & 0 & -3 \end{bmatrix} v_2 = 0$$

This gives the eigenvector:

$$v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Eigenvector for $\lambda_3 = 2$

$$(A - 2I)v_3 = 0 \quad \Rightarrow \quad \begin{bmatrix} 1 & 1 & 2 \\ 0 & 3 & 4 \\ 0 & 0 & 0 \end{bmatrix} v_3 = 0$$

This gives the eigenvector:

$$v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Step 4: Diagonal Matrix and Matrix of Eigenvectors

The matrix P, which diagonalizes A, is formed by the eigenvectors as columns:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The diagonal matrix D is formed by the eigenvalues along the diagonal:

$$D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Thus, A can be diagonalized as:

$$A = PDP^{-1}$$

Discuss the geometric interpretation of the eigenvectors and eigenvalues in the context of transformations. For instance, how does matrix A stretch, shrink, or rotate vectors in R^3?

In the context of transformations, eigen vectors and eigenvalues give us a clear geometric interpretation of how a matrix acts on vectors in space. For matrix A, the transformation it performs in R^3 can be understood through the effects of its eigenvalues and eigen vectors. The matrix can stretch, shrink, or even reverse the direction of vectors depending on the eigenvalues and the direction defined by the eigen vectors.The eigenvalues of a matrix A represent how much vectors along the direction of the eigen vectors are stretched or shrunk. For instance, if is an eigenvalue of matrix A and v is its corresponding eigen vector, then applying the matrix transformation A to v gives:

Av = λ v

This equation means that the vector v is scaled by the factor , without changing its direction. Geometrically, the eigenvalue tells us how much the vector v is stretched if $\lambda > 1$,shrunk if $0 < \lambda < 1$, or reflected if $\lambda < 0$. In the case of matrix A, the eigenvalues are:

$\lambda_1 = 3$, $\lambda_2 = 5$, $\lambda_3 = 2$

These eigenvalues indicate that the matrix will stretch vectors along the directions of their corresponding eigen vectors by factors of 3, 5, and 2, respectively. The eigenvectors define the directions in space that remain unchanged except for scaling by the eigenvalues. For matrix A, the eigen vectors are:

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

These eigen vectors correspond to the x-axis, y-axis, and z-axis, respectively. This means that matrix A preserves the directions of these axes. When the matrix acts on a vector along the x-axis, such as

$$\begin{bmatrix}1\\0\\0\end{bmatrix}$$

it stretches the vector by a factor of 3. Similarly, vectors along the y-axis and z-axis are stretched by factors of 5 and 2, respectively. Geometrically, this means that matrix A performs a non-uniform scaling (anisotropic transformation), as it stretches vectors differently along each axis. Since all the eigenvalues of matrix A are positive, the transformation involves only stretching and shrinking, with no reflection or rotation. This is because eigenvalues less than 0 would indicate a reversal of direction, and complex eigenvalues would indicate rotation. Therefore, matrix A stretches vectors in the positive directions of the eigenvectors, preserving their orientation in space.

In summary, matrix A stretches vectors along the x-axis by a factor of 3, along the y-axis by a factor of 5, and along the z-axis by a factor of 2. This transformation is purely scaling, and since all eigenvalues are positive, there is no flipping or rotation involved. Eigenvalues represent the magnitude of scaling, while eigenvectors provide the invariant directions along which the scaling occurs.

---

4. Project: Eigenfaces from the LFW (Labeled Faces in the Wild) Dataset:

In this section we will load the LFW data set. Once we load the data set then we have to Convert the images to grayscale and resize them to a smaller size (e.g., 64x64) to reduce computational complexity. We will flatten each image into a vector. After flattening each image we will compute the PCA on the flattened images and determine the number of principal components required to account for 80% of the variability. Then we will visualize the first few eigenfaces (principal components) and discuss their significance. Reconstruct some images using the computed eigenfaces and compare them with the original images. Let's begin our process by loading the data set.

Loading the LFW Dataset:

```r
image_dir <- "/Users/umerfarooq/Desktop/Data 605/Homework 1/lfw-deepfunneled"
```

Preprocess the Images in R:

```r
# Function to ensure the file path has an extension
add_extension_if_missing <- function(file_path, extension = ".jpg") {
  if (!grepl("\\.[a-zA-Z]{3,4}$", file_path)) {
    return(paste0(file_path, extension))
  }
  return(file_path)
}

# Function to preprocess a single image
preprocess_image <- function(image_path) {
  image_path <- add_extension_if_missing(image_path, ".jpg")
  img <- tryCatch({
    readImage(image_path)          # Read image
  }, error = function(e) {
    message("Error reading image: ", image_path)
    return(NULL)
  })
  if (is.null(img)) return(NULL)
  img <- channel(img, "gray")           # Convert to grayscale
  img <- resize(img, w = 64, h = 64)    # Resize to 64x64
  img <- as.vector(img)                 # Flatten into a vector
  return(img)
}

image_files <- list.files(image_dir, pattern = "\\.(jpg|png|jpeg)$", full.names = TRUE, recursive = TRU

image_data <- lapply(image_files, preprocess_image)
image_data <- Filter(Negate(is.null), image_data)

image_matrix <- do.call(rbind, image_data)

dim(image_matrix)
```

```
## [1] 13233  4096
```

Applying PCA:

Once we have the pre-processed images as a matrix, we can apply PCA to find the principal components. We'll use the `prcomp` function in R for this.

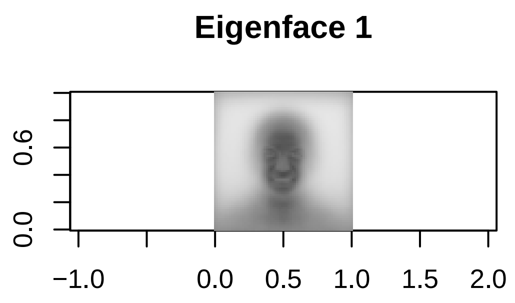
```
pca_result <- prcomp(image_matrix, center = TRUE, scale. = TRUE)


explained_variance <- summary(pca_result)$importance[2, ]

cum_variance <- cumsum(explained_variance)
num_components <- which(cum_variance >= 0.80)[1]

cat("Number of principal components needed for 80% variance:", num_components)
```

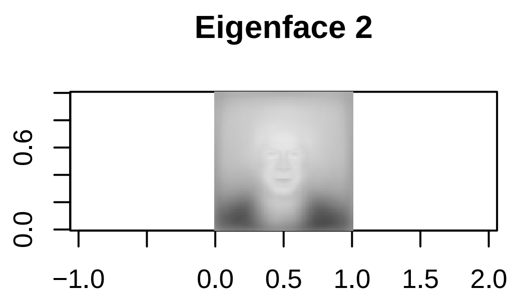
```
## Number of principal components needed for 80% variance: 72
```

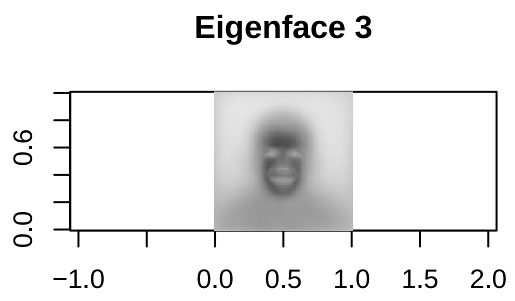
Visualizing Eigenfaces:

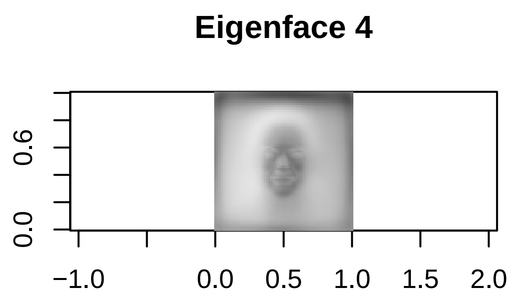
Eigenfaces are the principal components (or eigenvectors) of the covariance matrix of the images. We can visualize the first few eigenfaces to see the dominant features.

```
# Extract the first few eigenfaces (principal components)
eigenfaces <- pca_result$rotation[, 1:num_components]

# Reshape and visualize the first 4 eigenfaces (64x64 images)
par(mfrow=c(2, 2))  # Display 4 images in a 2x2 grid
for (i in 1:4) {
  eigenface_matrix <- matrix(eigenfaces[, i], 64, 64)

  # Rotate the eigenface by 180 degrees (flip both rows and columns)
  rotated_eigenface_matrix  <- eigenface_matrix[64:1, 64:1]

  # Plot the flipped eigenface matrix
  image(rotated_eigenface_matrix, col = gray.colors(256), main = paste("Eigenface", i), asp = 1)
}
```









Significance:

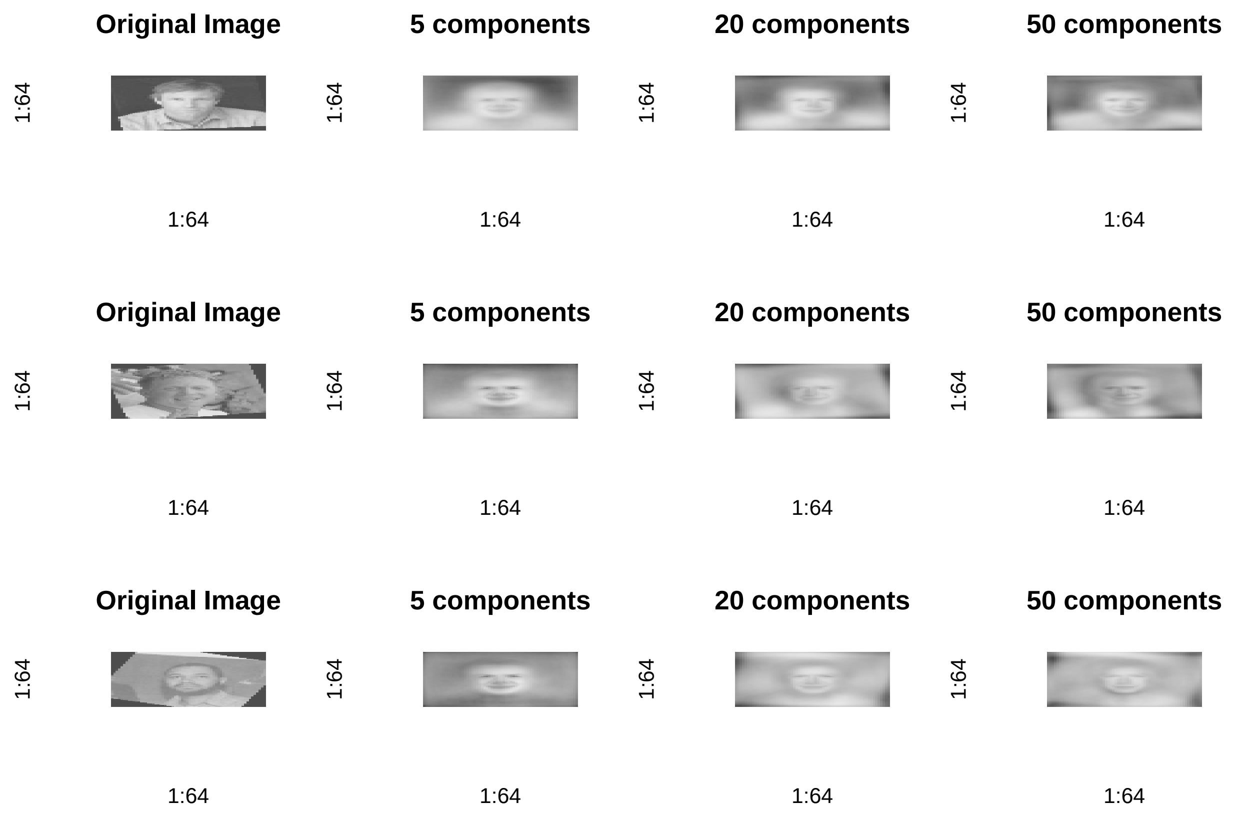
Eigenfaces represent the key features of faces extracted through Principal Component Analysis (PCA), serving as a way to reduce the dimensionality of facial image data by capturing the most important variations across images. They allow us to focus on the most significant aspects of faces, such as overall structure, lighting, and key features like eyes, nose, and mouth, while discarding less important details. By projecting images onto these eigenfaces, we can efficiently reconstruct faces with fewer components, making image processing faster and more efficient. Eigenfaces also play a crucial role in face recognition, as they help identify faces by comparing the most meaningful variations instead of comparing pixel-by-pixel. As more eigenfaces are used for reconstruction, the accuracy and detail of the reconstructed images increase, reflecting how much of the original information is captured.

Reconstruct Images Using Eigenfaces:

To reconstruct an image, we project the image onto the eigenfaces and use the principal components to approximate the original image

```
reconstruct_image <- function(image_vector, k) {
  projection <- image_vector %*% eigenfaces[, 1:k]
  reconstructed <- projection %*% t(eigenfaces[, 1:k])
  return(reconstructed)
}

# Function to plot original and reconstructed images
```

```r
plot_image <- function(image_vector, title) {
  # Reshape the vector back into a 64x64 image
  image_matrix <- matrix(image_vector, nrow = 64, ncol = 64)

  # Plot the image, correcting the orientation
  image(1:64, 1:64, image_matrix[, 64:1], col = gray.colors(256), axes = FALSE, main = title)
}

# Choose three images to reconstruct
image_indices <- c(1, 2, 3)  # Indices of the images to use

# Number of components to use for reconstruction
components <- c(5, 20, 50)

# Plot each selected image and its reconstructions
par(mfrow = c(length(image_indices), length(components) + 1))  # Adjust the layout for multiple images

for (i in image_indices) {
  original_image <- image_matrix[i, ]  # Select the image

  # Plot the original image
  plot_image(original_image, "Original Image")

  # Reconstruct and plot with different components
  for (k in components) {
    reconstructed_image <- reconstruct_image(original_image, k)
    plot_image(reconstructed_image, paste(k, " components", sep = ""))
  }
}
```


Original Image
5 components
20 components
50 components
1:64
Original Image
5 components
20 components
50 components
1:64
Original Image
5 components
20 components
50 components
1:64