EX NO : **DATE :**	**IMAGE ENHANCEMENT TECHNIQUE**

AIM : The aim of the program is to implement Image enhancement Technique using MATLAB

ALGORITHM :

1. Read the image from the specified file path into MATLAB.
2. Apply Histogram Equalization to enhance the contrast of the image using histeq.
3. Apply Contrast Stretching to increase contrast by adjusting the intensity values with imadjust and stretchlim.
4. Apply Gaussian Filtering to smooth the image and reduce noise using imgaussfilt with a standard deviation of 2.
5. Display the images in a 2x2 grid: the original image, histogram-equalized image, contrast-stretched image, and Gaussian-filtered image. Use subplot and imshow for visualization.

PROGRAM CODE:

```
% Read the image
I = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\APPLE.jpg');

% Histogram equalization
J1 = histeq(I);

% Contrast stretching
J2 = imadjust(I, stretchlim(I), []);

% Gaussian filtering
J3 = imgaussfilt(I, 2); % Standard deviation = 2

% Display original and enhanced images
subplot(2, 2, 1);
imshow(I);
title('Original Image');

subplot(2, 2, 2);
imshow(J1);
title('Enhanced Image - Histogram Equalization');

subplot(2, 2, 3);
imshow(J2);
title('Enhanced Image - Contrast Stretching');

subplot(2, 2, 4);
imshow(J3);
title('Enhanced Image - Gaussian Filtering');
```

OUTPUT :

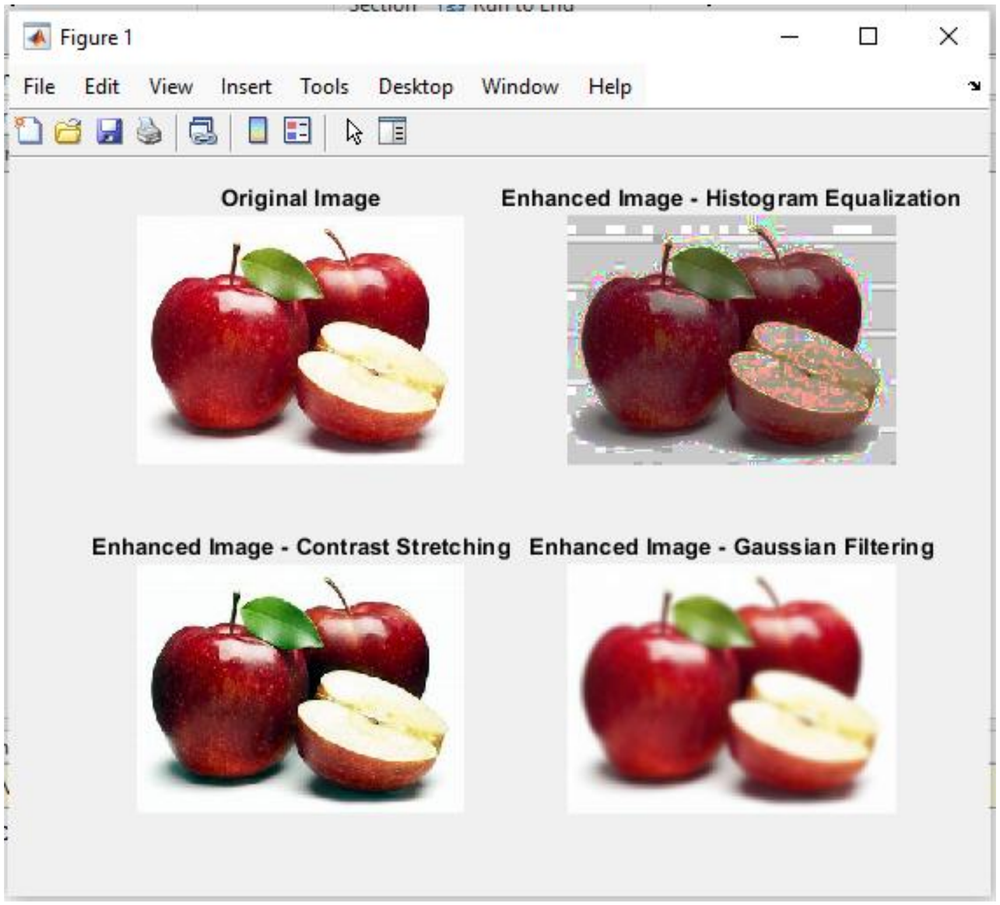


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
Enhanced Image - Histogram Equalization
Enhanced Image - Contrast Stretching
Enhanced Image - Gaussian Filtering

RESULT : Thus the program is to implement Image enhancement Technique using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**HISTOGRAM EQUALIZATION**

AIM : The aim of the program is to implement Histogram Equalization using MATLAB

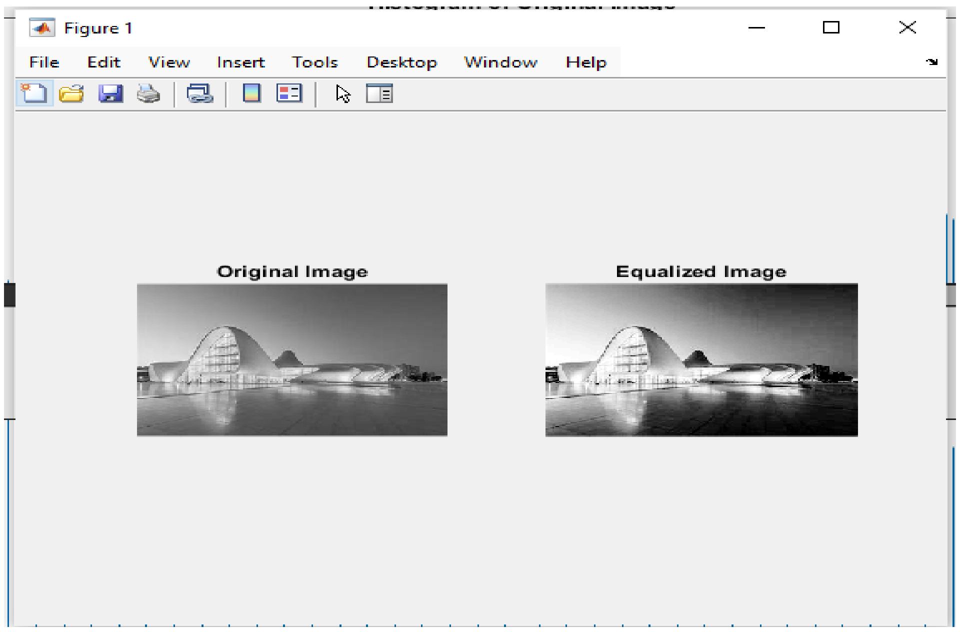
ALGORITHM :

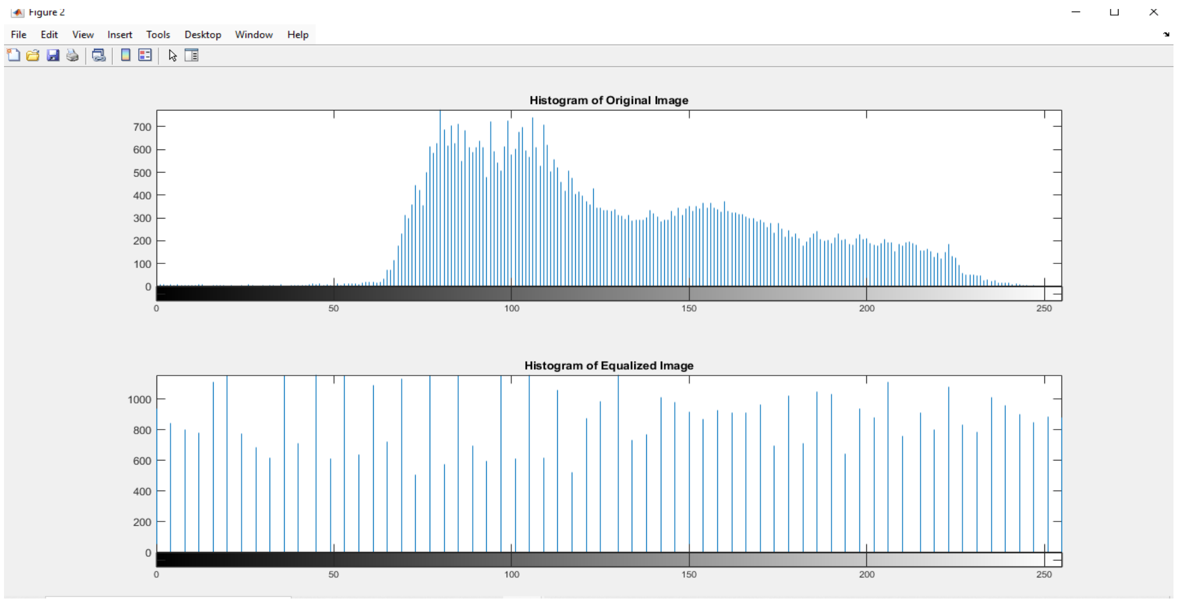
1. **Read the Input Image:** Load the image from the specified file path into MATLAB.
2. **Convert to Grayscale (if needed):** Check if the image is in RGB format (3 dimensions). If so, convert it to grayscale using rgb2gray.
3. **Perform Histogram Equalization:** Enhance the contrast of the grayscale image using histeq.
4. **Display Images:**
   - Create a figure with two subplots.
   - In the first subplot, display the original grayscale image with the title "Original Image".
   - In the second subplot, display the histogram-equalized image with the title "Equalized Image".
5. **Plot Histograms:**
   - Create another figure with two subplots.
   - In the first subplot, display the histogram of the original image using imhist, with the title "Histogram of Original Image".
   - In the second subplot, display the histogram of the equalized image using imhist, with the title "Histogram of Equalized Image".
6. **Enhance Visibility:** Maximize the figure window to fit the screen using set(gcf, 'Position', get(0, 'Screensize')).

PROGRAM CODE :

```
def compute_mean(numbers):
    return sum(numbers) / len(numbers)

def compute_variance(numbers):
    mean = compute_mean(numbers)
    squared_diff = [(x - mean) ** 2 for x in numbers]
    variance = sum(squared_diff) / len(numbers)
    return variance

def compute_standard_deviation(numbers):
    variance = compute_variance(numbers)
    standard_deviation = variance ** 0.5
    return standard_deviation

if __name__ == "__main__":
    # Taking user input for a list of numbers
    input_data = input("Enter a list of numbers separated by spaces: ")

    try:
        # Convert the user input into a list of floats
        data = [float(num) for num in input_data.split()]

        # Calculate the measures of dispersion
        variance = compute_variance(data)
        standard_deviation = compute_standard_deviation(data)

        print(f"Data: {data}")
        print(f"Variance: {variance}")
        print(f"Standard Deviation: {standard_deviation}")
    except ValueError:
print("Invalid input! Please enter a list of numbers separated by spaces.")
```

OUTPUT :





RESULT : Thus the Histogram Equalization program using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**IMAGE RESTORATION**

AIM : The aim of the program is to implement Image Restoration using MATLAB

ALGORITHM :

1. **Read the Degraded Image:**
   - ➢ Load the image from the specified file path into MATLAB.
2. **Convert to Double Precision:**
   - ➢ Convert the image to double precision using im2double for accurate processing.
3. **Display the Degraded Image:**
   - ➢ Create a figure and display the degraded image with the title "Degraded Image".
4. **Simulate the Blur:**
   - ➢ Define a Point Spread Function (PSF) using fspecial to simulate motion blur.
   - ➢ Convolve the degraded image with the PSF using imfilter to create a blurred image.
5. **Add Noise:**
   - ➢ Add Gaussian noise to the blurred image using imnoise with a specified noise variance to simulate a noisy environment.
6. **Display the Blurred and Noisy Image:**
   - ➢ Create a new figure and display the image with added noise and blur with the title "Blurred and Noisy Image".

7. **Estimate Noise Power Spectrum:**
   - Calculate the power spectrum of the noisy image using the Fourier transform (fft2).
   - Estimate the noise power by averaging the power spectrum.
8. **Perform Wiener Deconvolution:**
   - Apply Wiener de convolution to the blurred and noisy image using deconvwnr, with the estimated noise power and the PSF to restore the image.
9. **Display the Restored Image:**
   - Create a figure and display the restored image obtained from Wiener deconvolution with the title "Restored Image using Wiener Deconvolution".

PROGRAM CODE:

```
% Read the degraded image
degradedImage = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\jasmine.jpg');

% Convert to double precision for processing
degradedImage = im2double(degradedImage);

% Display the degraded image
figure;
imshow(degradedImage);
title('Degraded Image');

% Simulate the blur (convolution with a PSF)
PSF = fspecial('motion', 20, 45); % Example PSF (motion blur)
blurredImage = imfilter(degradedImage, PSF, 'conv', 'circular');

% Add noise to the blurred image
noiseVariance = 0.0001;
blurredNoisyImage = imnoise(blurredImage, 'gaussian', 0, noiseVariance);

% Display the blurred and noisy image
figure;
imshow(blurredNoisyImage);
title('Blurred and Noisy Image');

% Estimate the noise power spectrum (assuming Gaussian noise)
% This step helps in estimating the regularization parameter for Wiener filtering
N = numel(blurredNoisyImage);
noisePower = abs(fft2(blurredNoisyImage)).^2 / N;
estimatedNoisePower = mean(noisePower(:));

% Wiener deconvolution
wienerDeconvolution = deconvwnr(blurredNoisyImage, PSF, estimatedNoisePower);

% Display the restored image
figure;
imshow(wienerDeconvolution);
title('Restored Image using Wiener Deconvolution');

% Optional: Compare with original if available
% originalImage = imread('original_image.jpg');
% figure;
% imshow(originalImage);
% title('Original Image');
```

OUTPUT :

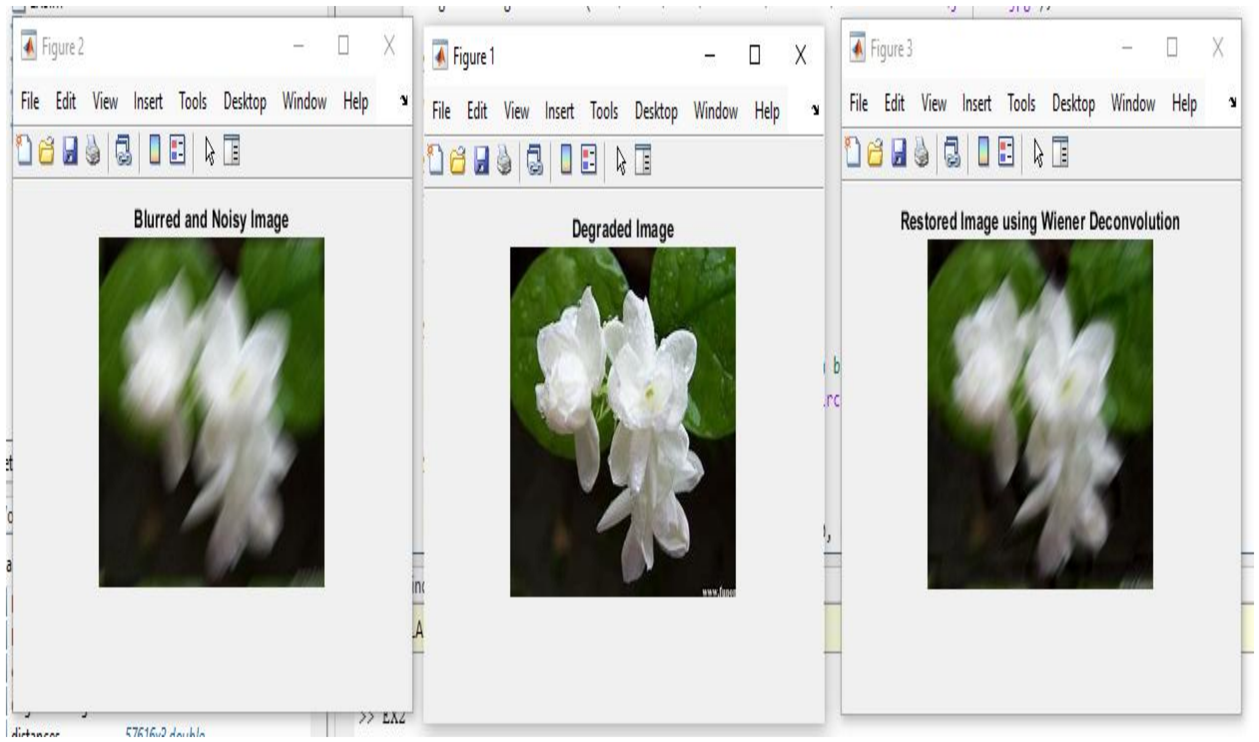


Figure 2
File Edit View Insert Tools Desktop Window Help
Blurred and Noisy Image
Figure 1
File Edit View Insert Tools Desktop Window Help
Degraded Image
Figure 3
File Edit View Insert Tools Desktop Window Help
Restored Image using Wiener Deconvolution

RESULT : Thus the Image Restoration program using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**IMAGE FILTERING**

AIM : The aim of the program is to implement Image Filtering using MATLAB.

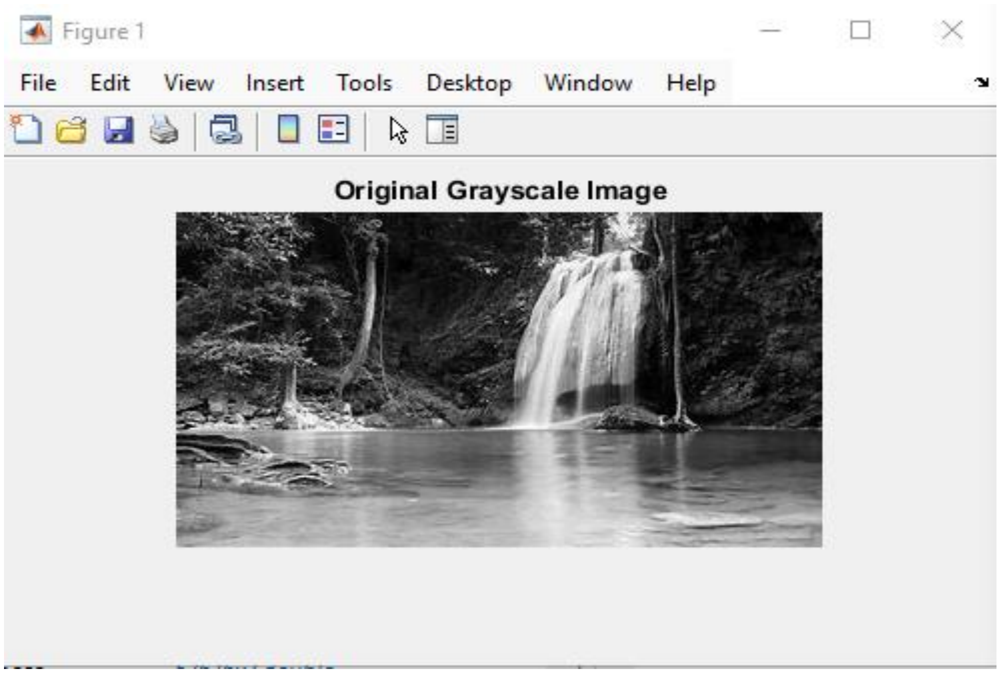
ALGORITHM :

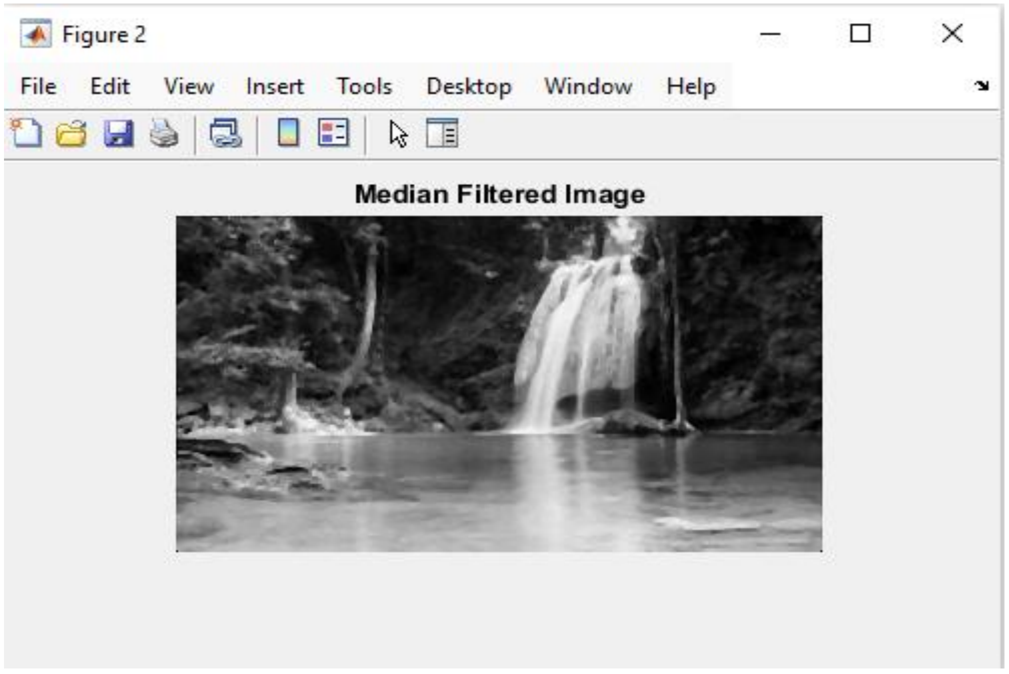
1. **Read the Input Image:**
   - Load the image from the specified file path into MATLAB.
2. **Convert to Grayscale (if necessary):**
   - Check if the image is RGB (3 channels). If so, convert it to grayscale using rgb2gray. If the image is already grayscale, keep it unchanged.
3. **Convert to Double Precision:**
   - Convert the grayscale image to double precision using im2double for further processing.
4. **Display the Original Image:**
   - Create a figure and display the original grayscale image with the title "Original Grayscale Image".
5. **Apply Median Filter:**
   - Apply a median filter to the grayscale image using medfilt2 with a 3x3 neighborhood to reduce noise.
6. **Display the Median Filtered Image:**
   - Create a new figure and display the median-filtered image with the title "Median Filtered Image".

PROGRAM CODE :

```
% Read the input image
originalImage = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\waterfall.jpg');

% Convert to grayscale if the image is RGB
if size(originalImage, 3) == 3
originalImageGray = rgb2gray(originalImage);
else
originalImageGray = originalImage; % Already grayscale
end

% Convert to double precision for processing
originalImageGray = im2double(originalImageGray);

% Display the original image
figure;
imshow(originalImageGray);
title('Original Grayscale Image');

% Apply median filter (noise reduction)
medianFiltered = medfilt2(originalImageGray, [3 3]); % 3x3 neighborhood

% Display the median filtered image
figure;
imshow(medianFiltered);
title('Median Filtered Image');
```

OUTPUT :





RESULT : Thus the Image Filtering program using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**EDGE DETECTION USING OPERATORS (ROBERTS, PREWITTS AND SOBELS OPERATORS)**

AIM : The aim of the program is to implement Edge detection using Operators (Roberts, Prewitt s and Sobels operators) using MATLAB

ALGORITHM :

1. **Clear Workspace and Set Up Environment:**
   - Clear variables, command window, and close all figures.
2. **Read the Image:**
   - Load the image from the specified file path into MATLAB.
3. **Convert to Grayscale (if necessary):**
   - Check if the image is in RGB format (3 channels). If so, convert it to grayscale using rgb2gray. If the image is already grayscale, use it as is.
4. **Convert to Double Precision:**
   - Convert the grayscale image to double precision using double for processing.
5. **Define Roberts Operators:**
   - Create the Roberts edge detection operators for detecting edges in the x and y directions.
6. **Apply Roberts Operators:**
   - Convolve the grayscale image with Roberts operators using imfilter to get edge responses in the x and y directions.
   - Compute the magnitude of the edges using the Euclidean norm.

7. **Define Prewitt Operators:**
   - Create the Prewitt edge detection operators for detecting edges in the x and y directions.

8. **Apply Prewitt Operators:**
   - Convolve the grayscale image with Prewitt operators using imfilter to get edge responses in the x and y directions.
   - Compute the magnitude of the edges using the Euclidean norm.

9. **Define Sobel Operators:**
   - Create the Sobel edge detection operators for detecting edges in the x and y directions.

10. **Apply Sobel Operators:**
    - Convolve the grayscale image with Sobel operators using imfilter to get edge responses in the x and y directions.
    - Compute the magnitude of the edges using the Euclidean norm.

11. **Normalize the Edge Detection Outputs:**
    - Normalize the edge detection results to the range [0, 1] using mat2gray for display purposes.

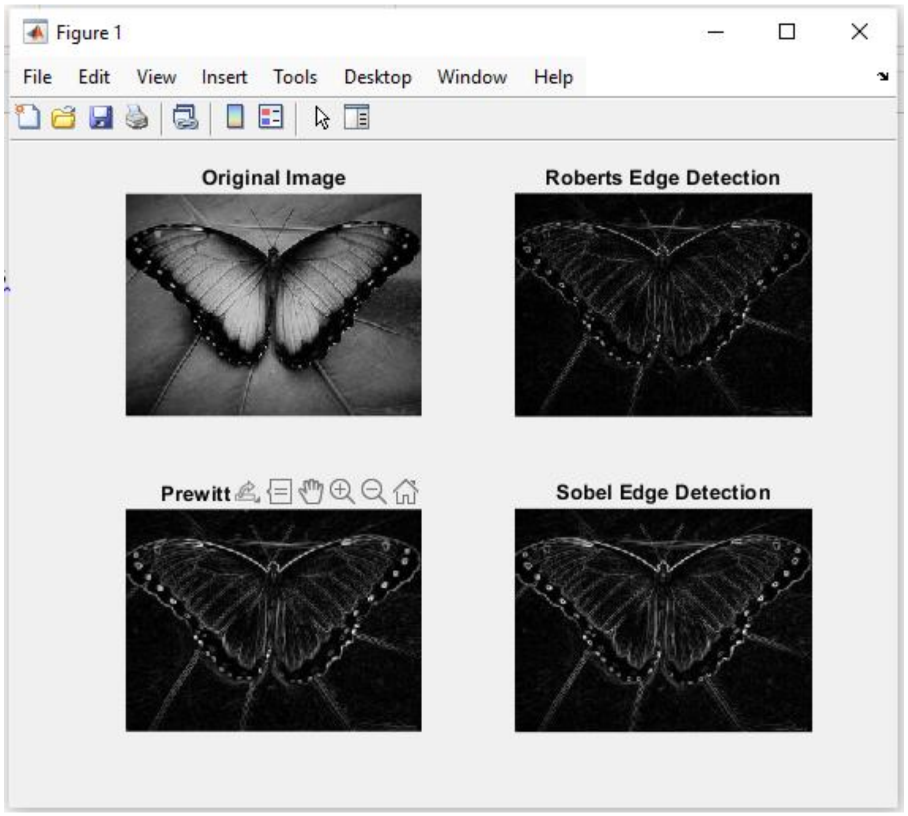
12. **Display the Results:**
    - Create a figure with a 2x2 subplot arrangement.
    - Display the original grayscale image in the first subplot.
    - Display the results of Roberts edge detection in the second subplot.
    - Display the results of Prewitt edge detection in the third subplot.
    - Display the results of Sobel edge detection in the fourth subplot.

PROGRAM CODE :

```
% Edge Detection using Roberts, Prewitt, and Sobel Operators
clear;
clc;
close all;

% Read the image
I = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\butterfly.jpg'); % Replace
with your image file

% Convert to grayscale if it's a color image
if size(I, 3) == 3
I_gray = rgb2gray(I);
else
I_gray = I;
end

% Convert to double for processing
I_gray = double(I_gray);

% Define Roberts operators
roberts_x = [1 0; 0 -1];
roberts_y = [0 1; -1 0];

% Apply Roberts operators
edge_roberts_x = imfilter(I_gray, roberts_x, 'replicate');
edge_roberts_y = imfilter(I_gray, roberts_y, 'replicate');
edge_roberts = sqrt(edge_roberts_x.^2 + edge_roberts_y.^2);

% Define Prewitt operators
prewitt_x = [-1 0 1; -1 0 1; -1 0 1];
prewitt_y = [-1 -1 -1; 0 0 0; 1 1 1];

% Apply Prewitt operators
edge_prewitt_x = imfilter(I_gray, prewitt_x, 'replicate');
edge_prewitt_y = imfilter(I_gray, prewitt_y, 'replicate');
edge_prewitt = sqrt(edge_prewitt_x.^2 + edge_prewitt_y.^2);

% DefineSobel operators
sobel_x = [-1 0 1; -2 0 2; -1 0 1];
sobel_y = [-1 -2 -1; 0 0 0; 1 2 1];

% ApplySobel operators
edge_sobel_x = imfilter(I_gray, sobel_x, 'replicate');
edge_sobel_y = imfilter(I_gray, sobel_y, 'replicate');
```

```
edge_sobel = sqrt(edge_sobel_x.^2 + edge_sobel_y.^2);

% Normalize the outputs to [0, 1] for display
edge_roberts = mat2gray(edge_roberts);
edge_prewitt = mat2gray(edge_prewitt);
edge_sobel = mat2gray(edge_sobel);

% Display the results
figure;
subplot(2,2,1);
imshow(I_gray, []);
title('Original Image');

subplot(2,2,2);
imshow(edge_roberts);
title('Roberts Edge Detection');

subplot(2,2,3);
imshow(edge_prewitt);
title('Prewitt Edge Detection');

subplot(2,2,4);
imshow(edge_sobel);
title('Sobel Edge Detection');
```

OUTPUT:



RESULT : Thus the Edge detection using Operators (Roberts, Prewitt s and Sobels operators) using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**IMAGE COMPRESSION**

AIM : The aim of the program is to implement Image Compression using MATLAB.

ALGORITHM :

1. **Clear Workspace and Set Up Environment:**

➢ Clear variables, command window, and close all figure windows.

2. **Read the Image:**

➢ Load the image from the specified file path into MATLAB.

3. **Convert to Grayscale (if necessary):**

➢ Check if the image is in RGB format (3 channels). If so, convert it to grayscale using rgb2gray. If the image is already grayscale, use it as is.

4. **Convert to Double Precision:**

➢ Convert the grayscale image to double precision using double for processing.

5. **Define Downsampling Factor:**

➢ Set the downsampling factor (compression level), where larger values result in higher compression.

6. **Downsample (Compress the Image):**

➢ Extract every downsampleFactor-th pixel from the grayscale image to create a downsampled (compressed) image.

7. Upsample (Reconstruct the Image):

- Use imresize with bilinear interpolation to upscale the downsampled image back to its original size.

8. Convert to uint8 for Display:

- Convert the reconstructed image to uint8 for visualization.

9. Display the Images:

- Create a figure with a specified size.
- Display the original grayscale image in the first subplot.
- Display the downsampled (compressed) image in the second subplot.
- Display the upsampled (reconstructed) image in the third subplot.
- Set axis visibility for better clarity.

10. Calculate and Display Compression Ratio:

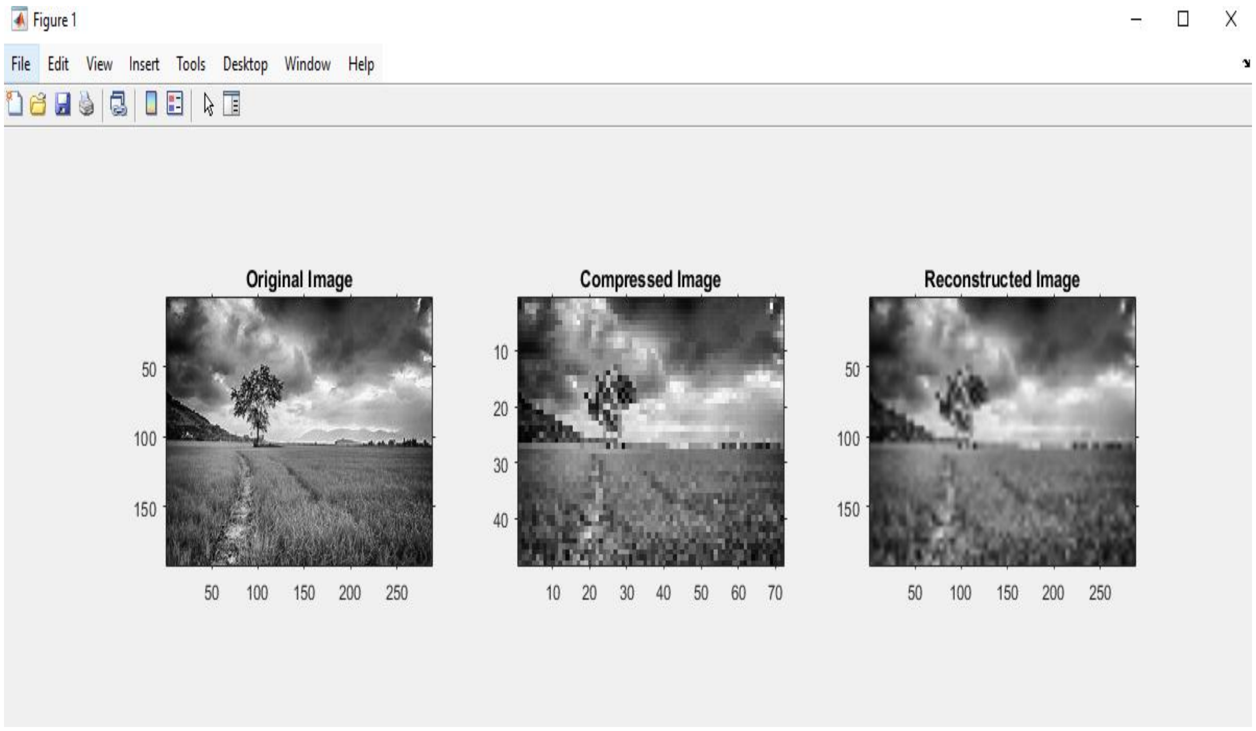
- Compute the size of the original image and the downsampled image.
- Calculate the compression ratio as the ratio of the original size to the compressed size.
- Print the compression ratio to the command window.

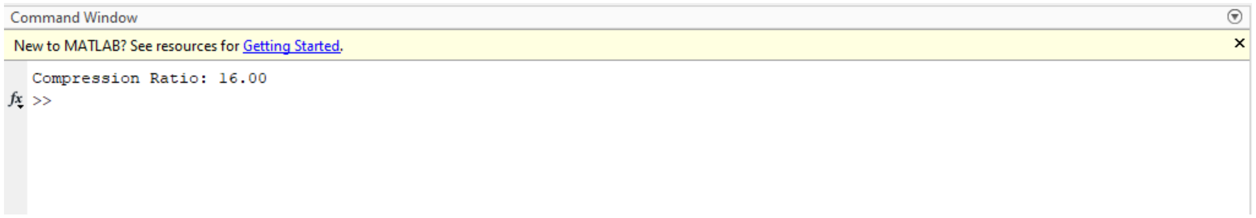
PROGRAM CODE :

```
% Simple Image Compression using Downsampling and Upsampling in MATLAB
clear;
clc;
close all;

% Read the image
I = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\farm.jpg'); % Replace with
your image file

% Convert to grayscale if it's a color image
if size(I, 3) == 3
I_gray = rgb2gray(I);
else
I_gray = I;
end

% Convert to double precision
I_gray = double(I_gray);

% Definedownsampling factor (compression level)
downsampleFactor = 4; % Larger values result in higher compression

% Downsample (compress the image)
I_downsampled = I_gray(1:downsampleFactor:end, 1:downsampleFactor:end);

% Upsample (reconstruct the image)
% Usingimresize to upscale the downsampled image back to original size
I_reconstructed = imresize(I_downsampled, [size(I_gray, 1), size(I_gray, 2)], 'bilinear');

% Convert to uint8 for display
I_reconstructed = uint8(I_reconstructed);

% Display the original and reconstructed images with larger figure size
figure('Position', [100, 100, 1200, 400]); % Set figure size (width x height)
subplot(1,3,1);
imshow(uint8(I_gray));
title('Original Image');
axis on; % Show axis for better visibility

subplot(1,3,2);
imshow(uint8(I_downsampled));
title('Compressed Image');
axis on; % Show axis for better visibility
```

```
subplot(1,3,3);
imshow(I_reconstructed);
title('Reconstructed Image');
axis on; % Show axis for better visibility

% Calculate compression ratio
originalSize = numel(I_gray);
compressedSize = numel(I_downsampled);
compressionRatio = originalSize / compressedSize;
fprintf('Compression Ratio: %.2f\n', compressionRatio);
```

OUTPUT :





```
Compression Ratio: 16.00
>>
```

RESULT : Thus the Image Compression program using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**IMAGE SUBRACTION**

AIM : The aim of the program is to implement Image Subtraction using MATLAB

ALGORITHM :

1. **Clear Workspace and Set Up Environment:**
   - Clear all variables, the command window, and close all figure windows to start fresh.
2. **Read the Two Images:**
   - Load the first image from the specified file path into variable I1.
   - Load the second image from the specified file path into variable I2.
3. **Convert to Grayscale (if necessary):**
   - Check if each image is in RGB format (3 channels). If so, convert each image to grayscale using rgb2gray. If the images are already grayscale, keep them unchanged.
4. **Get the Size of Each Image:**
   - Obtain the dimensions (number of rows and columns) of the grayscale images I1_gray and I2_gray.
5. **Determine the Size of the Smaller Image:**
   - Compute the dimensions (rows and columns) of the smaller image by taking the minimum number of rows and columns between the two images.
6. **Resize Both Images:**
   - Resize both grayscale images to match the dimensions of the smaller image using imresize.
7. **Convert to Double Precision:**
   - Convert the resized grayscale images to double precision using double for precise arithmetic operations.
8. **Perform Image Subtraction:**
   - Subtract pixel values of the second resized image from the first resized image to get the difference image.

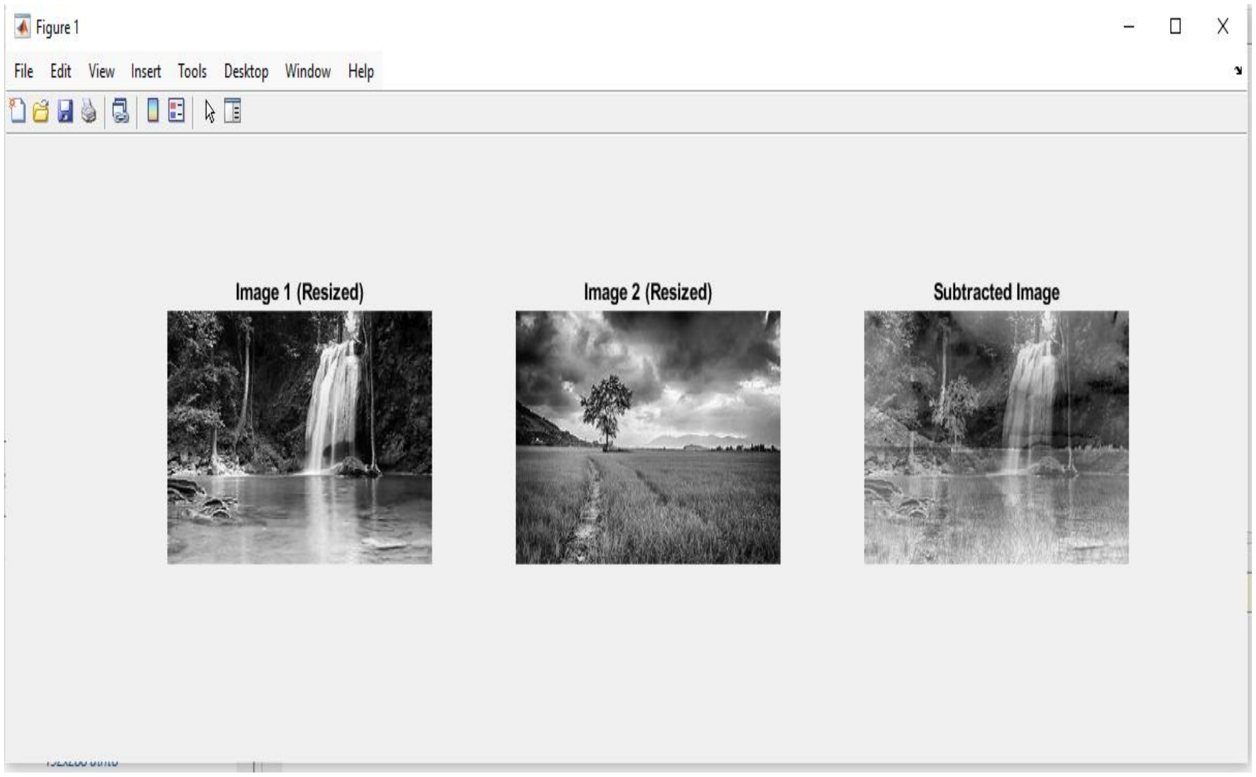
9. **Normalize the Result:**
   - Normalize the subtraction result to the range [0, 255] using mat2gray, then scale it to [0, 255] and convert it to uint8 for display.
10. **Display the Images:**
    - Create a figure with a specified size for better visibility.
    - Display the resized version of the first image in the first subplot.
    - Display the resized version of the second image in the second subplot.
    - Display the result of the image subtraction in the third subplot, showing the difference between the two images.

PROGRAM CODE :

```
% Image Subtraction with Size Adjustment in MATLAB
clear;
clc;
close all;

% Read the two images
I1 = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\waterfall.jpg'); % Replace
with your first image file
I2 = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\farm.jpg'); % Replace with
your second image file

% Convert to grayscale if necessary
if size(I1, 3) == 3
    I1_gray = rgb2gray(I1);
else
    I1_gray = I1;
end

if size(I2, 3) == 3
    I2_gray = rgb2gray(I2);
else
    I2_gray = I2;
end

% Get the size of each image
[rows1, cols1] = size(I1_gray);
[rows2, cols2] = size(I2_gray);

% Determine the size of the smaller image
rows = min(rows1, rows2);
cols = min(cols1, cols2);

% Resize both images to the size of the smaller image
I1_resized = imresize(I1_gray, [rows, cols]);
I2_resized = imresize(I2_gray, [rows, cols]);

% Convert to double for accurate subtraction
I1_resized = double(I1_resized);
I2_resized = double(I2_resized);

% Perform image subtraction
I_subtracted = I1_resized - I2_resized;

% Normalize the result to [0, 255]
```

```
I_subtracted = uint8(mat2gray(I_subtracted) * 255);

% Display the images
figure('Position', [100, 100, 1200, 400]); % Set figure size (width x height)
subplot(1,3,1);
imshow(uint8(I1_resized));
title('Image 1 (Resized)');

subplot(1,3,2);
imshow(uint8(I2_resized));
title('Image 2 (Resized)');

subplot(1,3,3);
imshow(I_subtracted);
title('Subtracted Image');
```

OUTPUT :



RESULT : Thus the Image Subtraction program using MATLAB has been implemented and executed successfully.

EX NO : DATE :	**BOUNDARY EXTRACTION USING MORPHOLOGY**

AIM : The aim of the program is to implement Boundary Extraction using morphology using MATLAB

ALGORITHM :

1. **Clear Workspace and Set Up Environment:**

➢ Clear all variables, the command window, and close all figure windows to start with a clean environment.

2. **Read the Image:**

➢ Load the image from the specified file path into MATLAB.

3. **Convert to Grayscale (if necessary):**

➢ Check if the image is in RGB format (3 channels). If so, convert it to grayscale using rgb2gray. If the image is already grayscale, use it as is.

4. **Convert to Binary Image:**

➢ Convert the grayscale image to a binary image using a specified threshold. Pixels with intensity values above the threshold are set to 1 (true), and others are set to 0 (false).

5. **Display the Binary Image:**

➢ Create a figure with a specified size.
➢ Display the binary image in the first subplot with the title "Binary Image".

6. **Perform Morphological Operations to Extract Boundaries:**

- Create a structuring element using strel (e.g., a disk-shaped element with a radius of 1).
- Erode the binary image using imerode to reduce the size of the foreground objects.
- Dilate the eroded image using imdilate to restore the size of the foreground objects to their original shape.
- Extract boundaries by computing the difference between the dilated image and the eroded image.

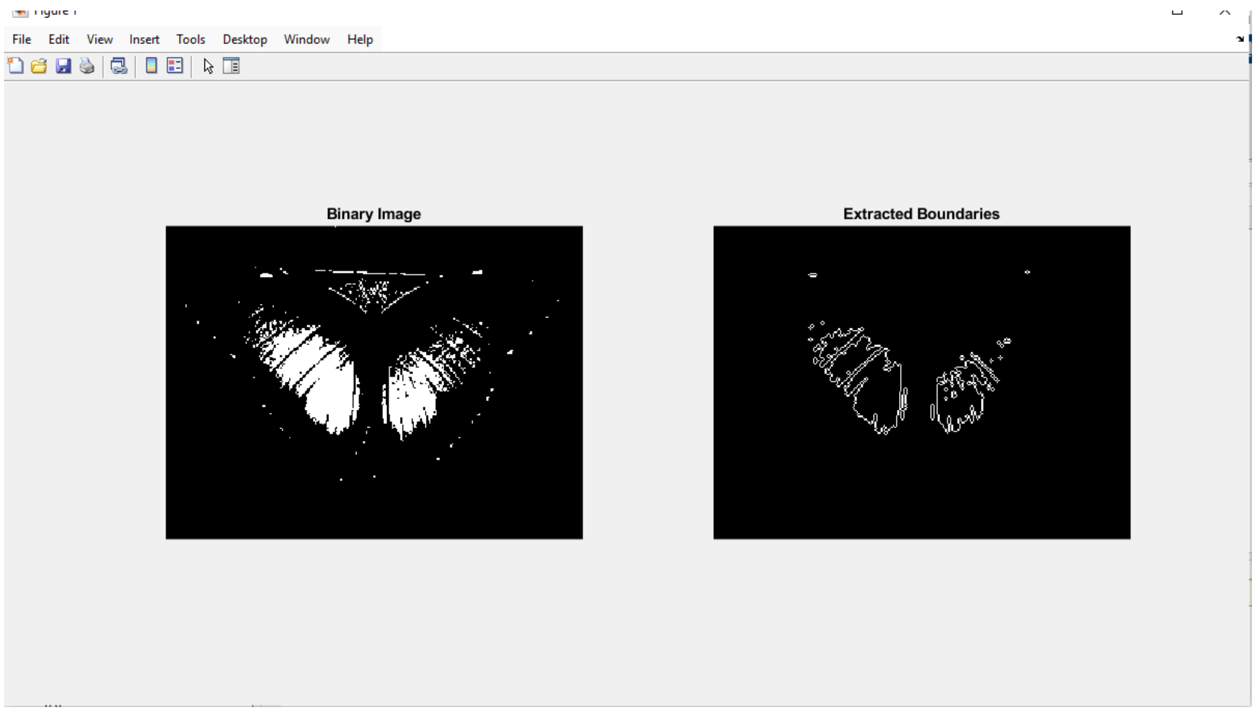
7. Display the Extracted Boundaries:

- Display the boundary-extracted image in the second subplot with the title "Extracted Boundaries".

PROGRAM CODE :

```
% Boundary Extraction using Morphology in MATLAB
clear;
clc;
close all;

% Read the image
I = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\butterfly.jpg'); % Replace
with your image file

% Convert to grayscale if it's a color image
if size(I, 3) == 3
I_gray = rgb2gray(I);
else
I_gray = I;
end

% Convert the grayscale image to a binary image
% Using a simple threshold; adjust as needed
threshold = 128; % Example threshold value
I_binary = I_gray> threshold;

% Display the original binary image
figure('Position', [100, 100, 1200, 600]); % Set figure size (width x height)
subplot(1,2,1);
imshow(I_binary);
title('Binary Image');

% Perform morphological operations to extract boundaries
% Create a structuring element
se = strel('disk', 1); % You can adjust the size as needed

% Erode the binary image
I_eroded = imerode(I_binary, se);

% Dilate the eroded image to get the boundaries
I_dilated = imdilate(I_eroded, se);

% Extract the boundaries
I_boundaries = I_dilated& ~I_eroded;

% Display the boundary extracted image
subplot(1,2,2);
imshow(I_boundaries);
title('Extracted Boundaries');
```

OUTPUT :



RESULT : Thus the Boundary Extraction program using MATLAB has been implemented and executed successfully.

EX NO : **DATE :**	**IMAGE SEGMENTATION**

AIM : The aim of the program is to implement Image Segmentation using MATLAB.

ALGORITHM :

1. **Clear Workspace and Set Up Environment:**

- Clear all variables, the command window, and close all figure windows to start with a clean environment.

2. **Read the Image:**

- Load the image from the specified file path into MATLAB.

3. **Convert to Grayscale (if necessary):**

- Check if the image is in RGB format (3 channels). If so, convert it to grayscale using rgb2gray. If the image is already grayscale, use it as is.

4. **Thresholding Segmentation:**

- Define a threshold value to segment the image into binary regions.
- Convert the grayscale image to a binary image using the threshold value (pixels with intensity values above the threshold are set to 1; others are set to 0).

5. **Custom K-means Clustering Segmentation:**

- Reshape the grayscale image into a 2D array where each row represents a pixel's intensity value.
- Define the number of clusters for segmentation.
- Initialize cluster centers randomly within the range of pixel intensity values (0 to 255).
- Implement the K-means clustering algorithm:

  - Compute distances from each pixel to each cluster center.
  - Assign each pixel to the closest cluster center based on distance.
  - Update cluster centers based on the mean intensity of pixels assigned to each cluster.
  - Check for convergence by comparing new and old cluster centers; terminate if the change is below a set tolerance.
- Reshape the cluster assignments back into the original image dimensions to produce the segmented image.

6. **Edge Detection-Based Segmentation:**

- Perform edge detection on the grayscale image using the Canny method to highlight edges.

7. **Display All Results:**

- Create a figure with a specified size for better visibility.
- Display the original grayscale image in the first subplot.
- Display the binary image obtained from thresholding in the second subplot.
- Display the segmented image from the custom K-means clustering in the third subplot.
- Display the edges detected using the Canny method in the fourth subplot.

PROGRAM CODE :

```
% Image Segmentation using Thresholding, Custom K-means, and Edge Detection
clear;
clc;
close all;

% Read the image
I = imread('C:\Users\SONY\OneDrive\Pictures\MATLAB PICTURES\butterfly.jpg'); % Replace
with your image file

% Convert to grayscale if it's a color image
if size(I, 3) == 3
I_gray = rgb2gray(I);
else
I_gray = I;
end

% --- Thresholding Segmentation ---
% Define a threshold value
threshold = 128; % Adjust as needed
I_binary = I_gray> threshold;

% --- Custom K-means Clustering Segmentation ---
% Reshape the image into a 2D array where each row is a pixel
[rows, cols] = size(I_gray);
I_reshaped = double(I_gray(:));

% Define the number of clusters
numClusters = 3; % Adjust as needed

% Initialize cluster centers randomly
numPixels = length(I_reshaped);
initialCenters = rand(numClusters, 1) * 255;

% K-means clustering algorithm
maxIter = 100; % Maximum number of iterations
tolerance = 1e-6; % Convergence tolerance
foriter = 1:maxIter
    % Compute distances from each pixel to each cluster center
distances = zeros(numPixels, numClusters);
for k = 1:numClusters
distances(:, k) = abs(I_reshaped - initialCenters(k));
end
```

```
    % Assign each pixel to the closest cluster center
    [~, idx] = min(distances, [], 2);

    % Compute new cluster centers
newCenters = arrayfun(@(k) mean(I_reshaped(idx == k)), 1:numClusters);

    % Check for convergence
if max(abs(newCenters - initialCenters)) < tolerance
break;
end

    % Update cluster centers
initialCenters = newCenters;
end

% Reshape the cluster indices to the original image size
I_segmented = reshape(idx, [rows, cols]);

% --- Edge Detection-Based Segmentation ---
% Perform edge detection using the Canny method
edges = edge(I_gray, 'canny');

% Display all results
figure('Position', [100, 100, 1800, 600]); % Set figure size (width x height)

% Original Image
subplot(1,4,1);
imshow(I_gray);
title('Original Image');

% Binary Image (Thresholding)
subplot(1,4,2);
imshow(I_binary);
title('Binary Image (Thresholding)');

% Segmented Image (Custom K-means)
subplot(1,4,3);
imshow(I_segmented, []);
title('Segmented Image (Custom K-means)');

% Edges (Canny)
subplot(1,4,4);
imshow(edges);
title('Edges (Canny)');
```

OUTPUT :

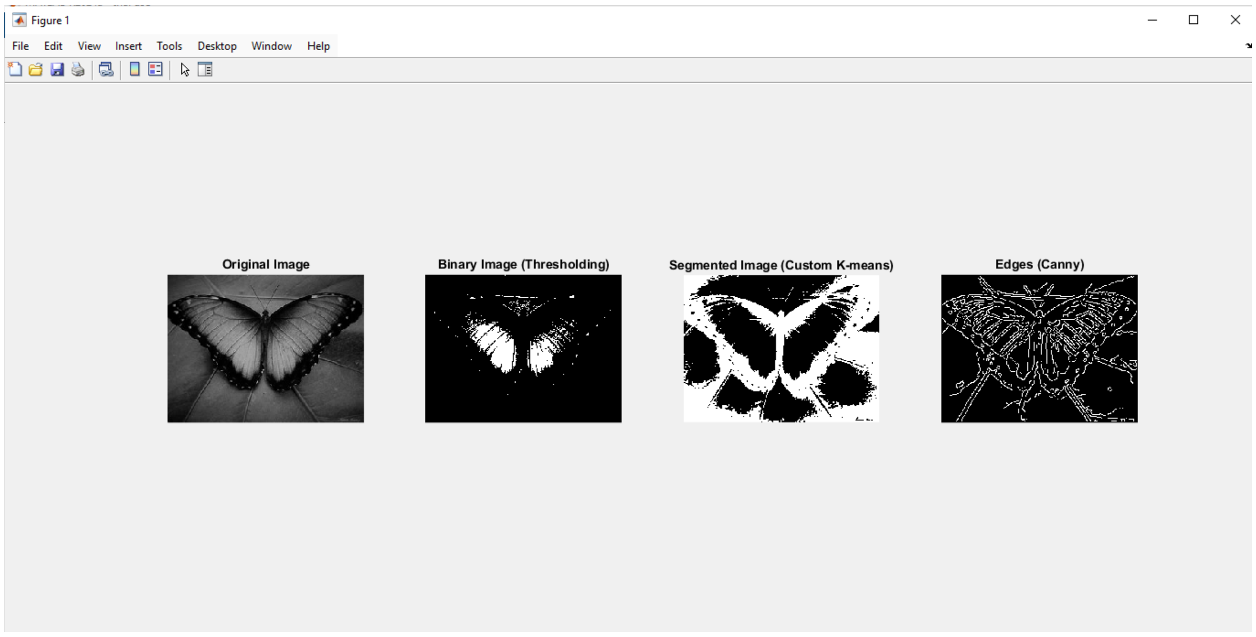


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
Binary Image (Thresholding)
Segmented Image (Custom K-means)
Edges (Canny)

RESULT : Thus the Image Segmentation program using MATLAB has been implemented and executed successfully.