Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Engineering

Academic Year 2022-2023

NAME	**:**	**Junaid Girkar**
SAP ID	**:**	**60004190057**
DIVISION	**:**	**A/A2**
BRANCH	**:**	**Computer**
A.Y.	**:**	**Engineering 2022-23**
SEM	**:**	**VII**
SUB	**:**	**Digital Signal Processing Applications**

ACADEMIC YEAR – 2022-2023

List of Experiment

Sub- Digital Signal and Image Processing Lab **Sem- VII**

1. Construction of standard DT signals(Sine,Cosine,Unit Step, Unit Impulse, Ramp) and perform different operations (addition ,subtraction , multiplication & Time scaling > 1 & Time scaling <1) on the signal
2. To perform Discrete Auto/Cross Correlation between two signals
3. To perform Discrete Convolution (Linear and Circular)
4. To apply Discrete Fourier Transform on DT signal
5. To implement Image negative, Gray level Slicing and Thresholding
6. Implementation of Contrast Stretching ,Dynamic range compression & Bit plane Slicing
7. Implementation of Histogram Stretching
8. Implementation of Histogram Equalization
9. Implementation of Image smoothing/ Image sharpening
10. Implementation of Edge detection using Sobel and Prewitt masks

Prof. Shruti Dodani

Prof. Vivek Deodeshmukh Dr. Meera Narvekar

Subject Incharges HoD, Computer Dept

Experiment No. 1

Aim: Construction of standard DT signals(Sine,Cosine,Unit Step, Unit Impulse, Ramp) and perform different operations (addition ,subtraction , multiplication & Time scaling > 1 & Time scaling <1) on the signal

Objective: Develop a program to generate standard signals like unit impulse, unit step and unit ramp and perform different manipulation operations on them.

Software:MATLAB

Theory:

Signal Generation Standard Signals

1. Unit Impulse Sequence

$$\boxed{\delta[n]=\begin{cases}1 \ , n=0\\ 0 \ , n\neq 0\end{cases}}$$

2. Unit Step Sequence

$$u[n]=\begin{cases}0, & n<0\\ 1, & n\geq 0\end{cases}.$$

3. Unit Ramp Sequence

$$\text{ramp}[n]=\begin{Bmatrix}n & , & n\geq 0\\ 0 & , & n<0\end{Bmatrix}$$

Signal Manipulation

1. Time scaling Compression
2. Time scaling Expansion
3. Signal Addition
4. Signal Subtraction
5. Signal Multiplication

Problem Definition:

Plot a graph of all signal generation and manipulation techniques by taking time duration on x-axis and signal amplitude on y-axis.

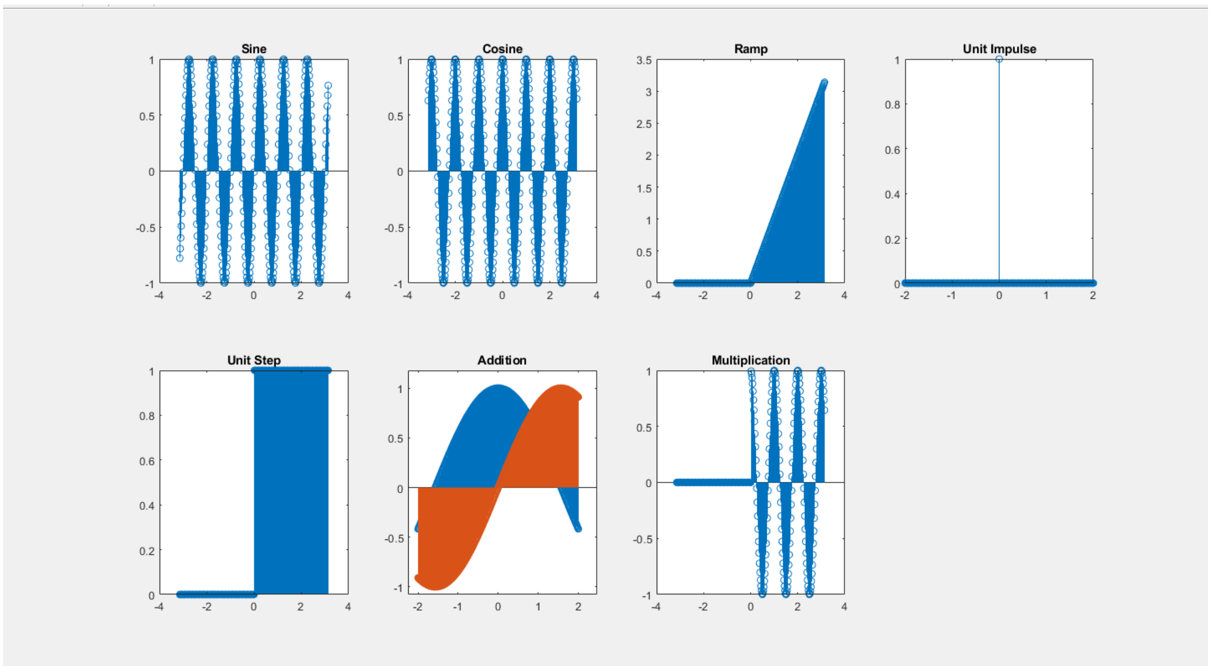
Experiment 1A:

Code:

```
n = [-pi: 0.02: pi];
t=-2:0.01:2;


sine_y = sin(2*n*pi);
cosine_y= cos(2*n*pi);
ramp_y= max(0,n);
unitImpulse_y = t==0;
unitstep_y = n>=0;

subplot(2,4,1),stem(n,sine_y);
title('Sine');

subplot(2,4,2),stem(n,cosine_y);
title('Cosine');

subplot(2,4,3),stem(n,ramp_y);
title('Ramp');

subplot(2,4,4),stem(t,unitImpulse_y);
title('Unit Impulse');

subplot(2,4,5),stem(n,unitstep_y);
title('Unit Step');
```

```
addition_t = -pi:0.01:(2*pi);
x = cos(t);
y = sin(t);
subplot(2,4,6),stem(t,x); hold on;
stem(t,y);
axis([0 2*pi -1.5 1.5])
title('Addition');


multiplication=cosine_y.*unitstep_y;
subplot(2,4,7),stem(n,multiplication);
title('Multiplication');
```

Output:



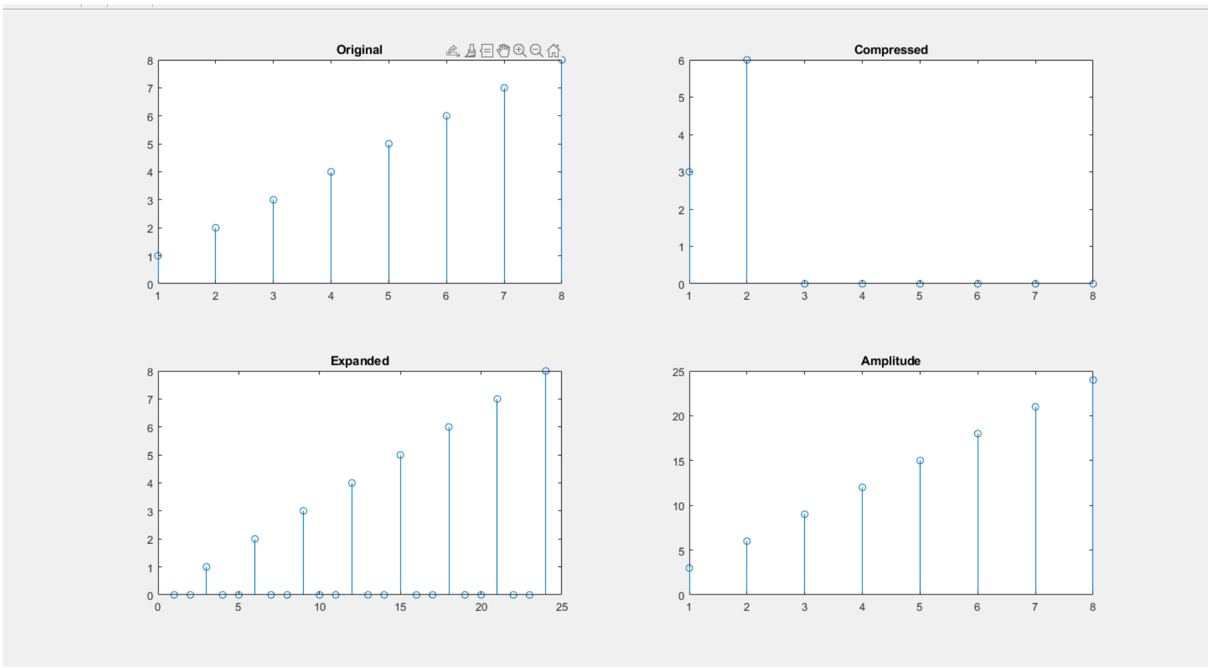
Experiment 1B:

Code:

```
signal = input("Enter Signal : ");
```

```
%signal = [1,2,3,4,5,6,7,8];x
signal_length = length(signal);
samples = 1:1:signal_length;
subplot(2,2,1),stem(samples,signal);
title('Original');

factor = input('Enter Factor : ');
%Create an array of zeros
compressed = zeros(1,signal_length);
for i = 1:signal_length
    %If new array ietm is out of bounds
    if i*factor > signal_length
        compressed(1,i) = 0;
    else
        compressed(1,i) = signal(1,factor*i);
    end
end
subplot(2,2,2),stem(samples,compressed);
title('Compressed');

samples = 1:1:factor*signal_length;
expanded = zeros(1,factor*signal_length);
for i=1:1:factor*signal_length
    if floor(i/factor) == i/factor
        expanded(1,i) = signal(1,i/factor);
    end
end
subplot(2,2,3),stem(samples,expanded);
title('Expanded');

samples = 1:1:signal_length;
subplot(2,2,4),stem(samples,factor*signal);
title('Amplitude');
```

Output:



Conclusion:

In this experiment, we learnt about various signal operations and plotted them using MATLAB's array operations. MATLAB treats signals as an array of numbers with the index linking them together and number of dimensions equal to the number of arrays.

Experiment No. 2

Aim: To perform Discrete Auto/Cross Correlation between two signals

Objective:

1. Calculate correlation of a DT signals and verify the results using mathematical formulation. .

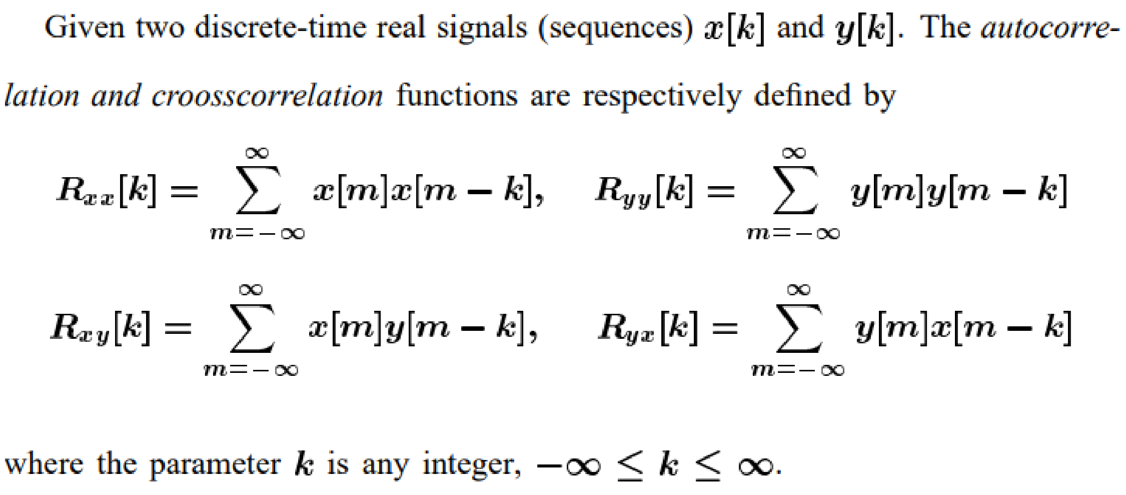
Input Specifications:

1. Length of first Signal L and signal values.

2. Length of second Signal M and signal values.

Theory:

Correlation : is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. This is also known as a sliding dot or sliding inner-product. It is commonly used for searching a long signal for a shorter, known feature.



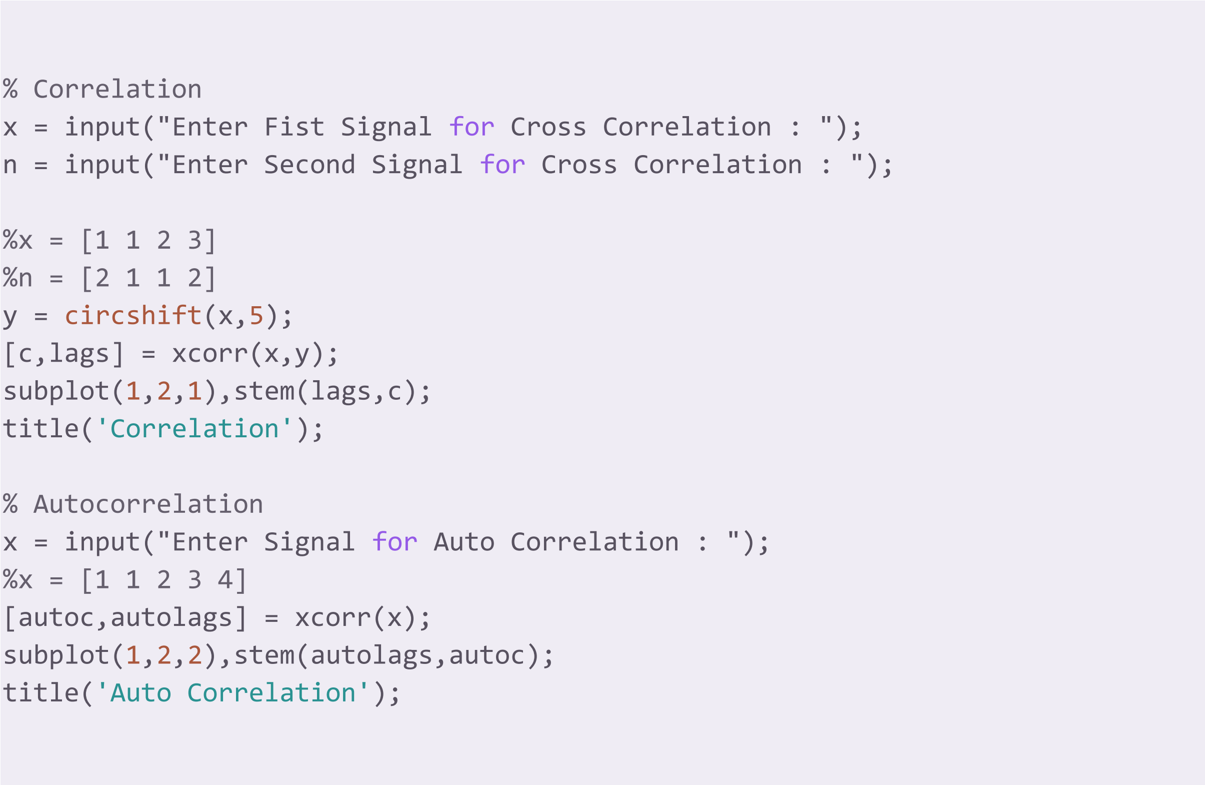
Given two discrete-time real signals (sequences) $x[k]$ and $y[k]$. The *autocorrelation and croosscorrelation* functions are respectively defined by

$$R_{xx}[k] = \sum_{m=-\infty}^{\infty} x[m]x[m-k], \quad R_{yy}[k] = \sum_{m=-\infty}^{\infty} y[m]y[m-k]$$

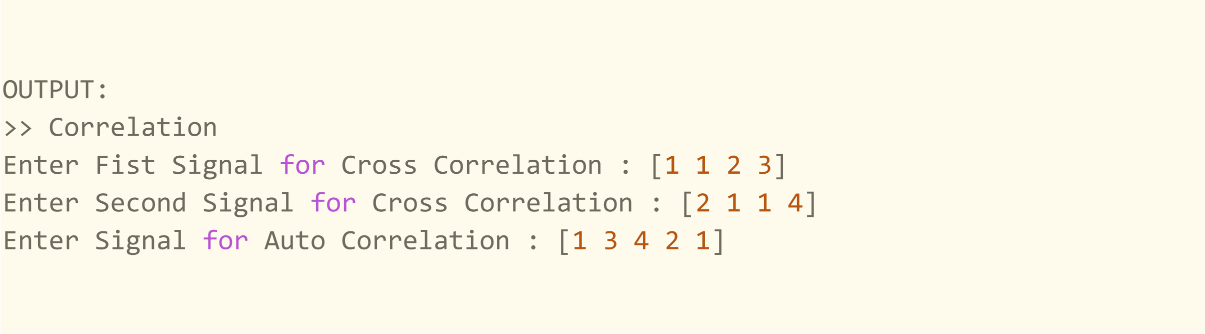
$$R_{xy}[k] = \sum_{m=-\infty}^{\infty} x[m]y[m-k], \quad R_{yx}[k] = \sum_{m=-\infty}^{\infty} y[m]x[m-k]$$

where the parameter k is any integer, $-\infty \leq k \leq \infty$.

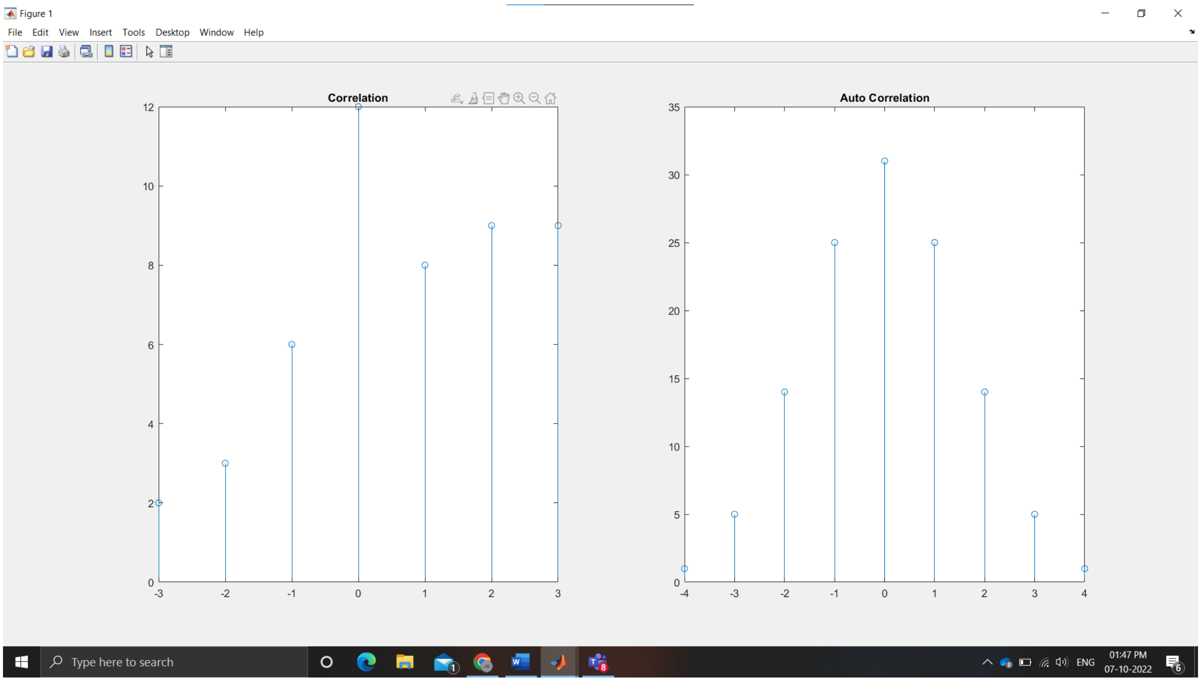
CODE:



```
% Correlation
x = input("Enter Fist Signal for Cross Correlation : ");
n = input("Enter Second Signal for Cross Correlation : ");

%x = [1 1 2 3]
%n = [2 1 1 2]
y = circshift(x,5);
[c,lags] = xcorr(x,y);
subplot(1,2,1),stem(lags,c);
title('Correlation');

% Autocorrelation
x = input("Enter Signal for Auto Correlation : ");
%x = [1 1 2 3 4]
[autoc,autolags] = xcorr(x);
subplot(1,2,2),stem(autolags,autoc);
title('Auto Correlation');
```



```
OUTPUT:
>> Correlation
Enter Fist Signal for Cross Correlation : [1 1 2 3]
Enter Second Signal for Cross Correlation : [2 1 1 4]
Enter Signal for Auto Correlation : [1 3 4 2 1]
```



CONCLUSION: We learnt about in build MATLAB functions for signal plotting and used them to implement Linear and Circular Convolution.

Problem Definition:

1. Find auto correlation of delayed input signal.

2. Find cross correlation of input signal and delayed input signal,

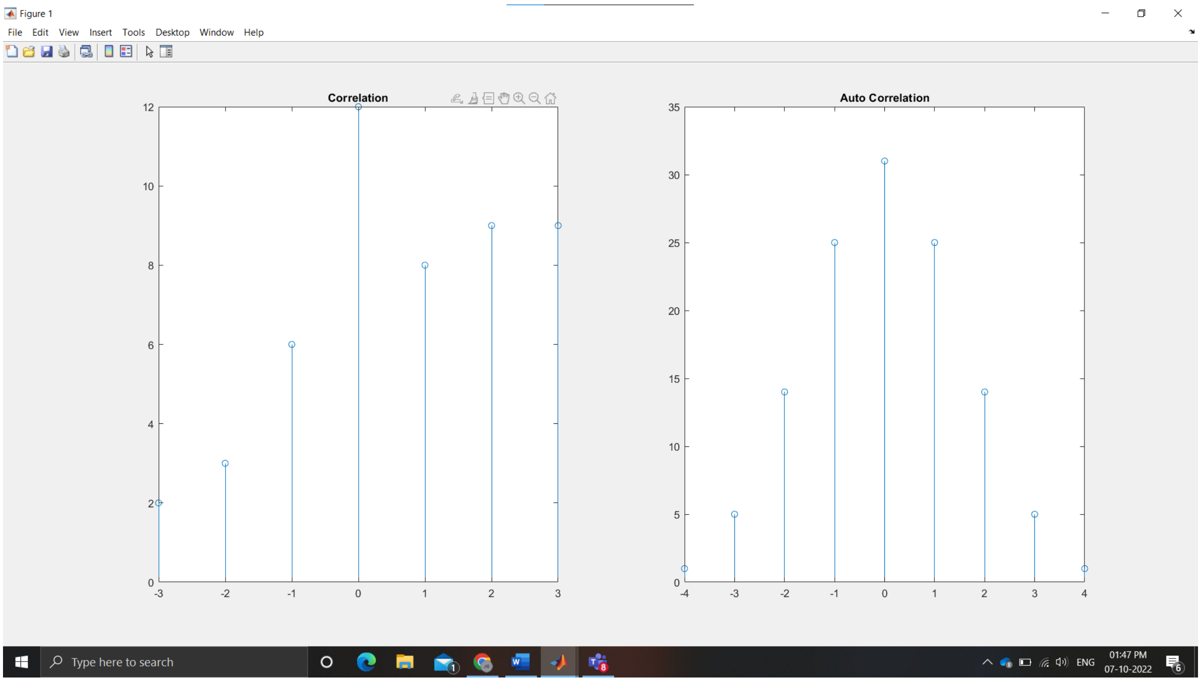
3. Compare the resultant signals. Give your conclusion.

CODE:

```
% Correlation
x = input("Enter Fist Signal for Cross Correlation : ");
n = input("Enter Second Signal for Cross Correlation : ");

%x = [1 1 2 3]
%n = [2 1 1 2]
y = circshift(x,5);
[c,lags] = xcorr(x,y);
subplot(1,2,1),stem(lags,c);
title('Correlation');

% Autocorrelation
x = input("Enter Signal for Auto Correlation : ");
%x = [1 1 2 3 4]
[autoc,autolags] = xcorr(x);
subplot(1,2,2),stem(autolags,autoc);
title('Auto Correlation');
```

OUTPUT:

```
>> Correlation
Enter Fist Signal for Cross Correlation : [1 1 2 3]
Enter Second Signal for Cross Correlation : [2 1 1 4]
Enter Signal for Auto Correlation : [1 3 4 2 1]
```



CONCLUSION: We learnt about in build MATLAB functions for signal plotting and used them to implement Cross-Correlation and Auto Correlation.

Experiment No. 3

Aim: To perform Discrete Convolution (Linear and Circular)

Objective:

1. Calculate Linear Convolution, Circular Convolution & verify the results using mathematical formulation.

Input Specifications:

1. Length of first Signal L and signal values.

2. Length of second Signal M and signal values.

Theory:

Convolution

The Discrete-Time Convolution (DTC) is one of the most important operations in a discrete-time signal analysis. The operation relates the output sequence y(n) of a linear-time invariant (LTI) system, with the input sequence x(n) and the unit sample sequence h(n)

$$\boxed{x_3(n) = \sum_{m=-\infty}^{+\infty} x_1(m)\, x_2(n-m)} \quad \text{or} \quad \boxed{x_3(n) = \sum_{m=-\infty}^{+\infty} x_2(m)\, x_1(n-m)}$$

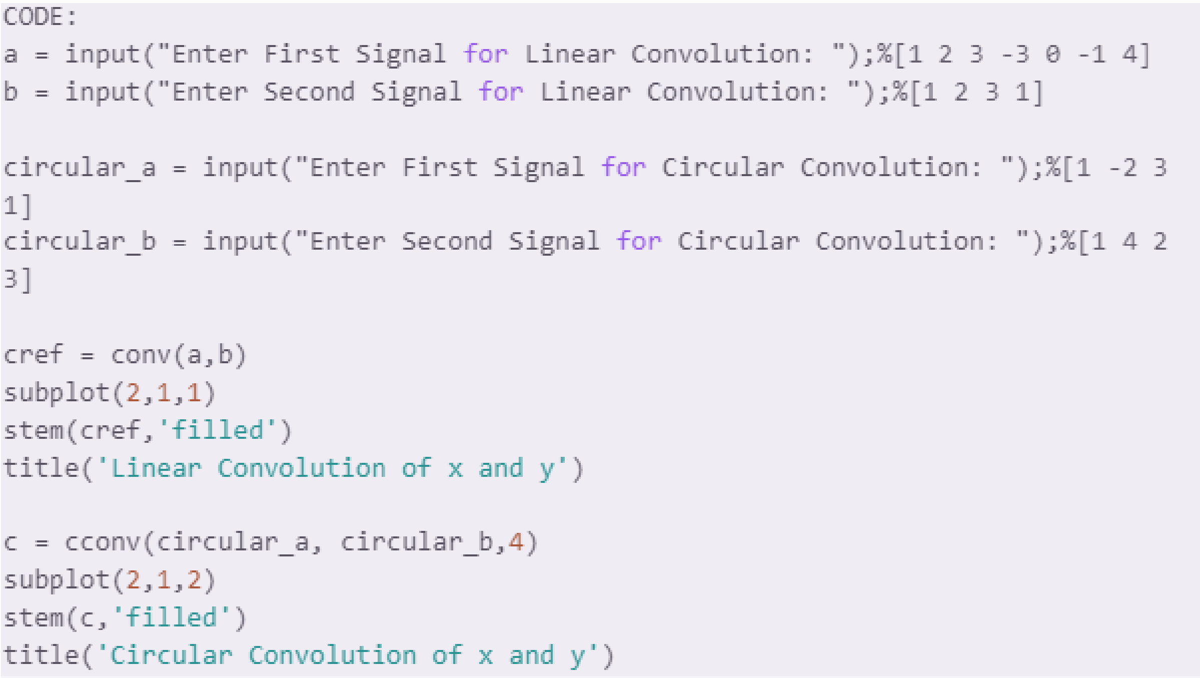
$$x_3(n) = x_1(n) * x_2(n) = x_2(n) * x_1(n)$$

Circular convolution, also known as cyclic convolution, is a special case of periodic convolution, which is **the convolution of two periodic functions that have the same period**.

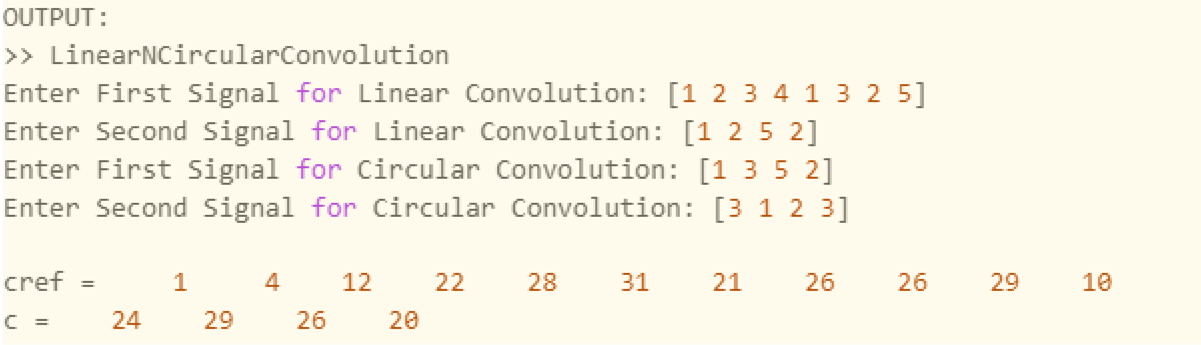
$$\boxed{x_3(n)=\sum_{m=0}^{N-1} x_1(m)\, x_2((n-m))_N} \quad \text{or} \quad \boxed{x_3(n)=\sum_{m=0}^{N-1} x_2(m)\, x_1((n-m))_N}$$

Problem Definition:

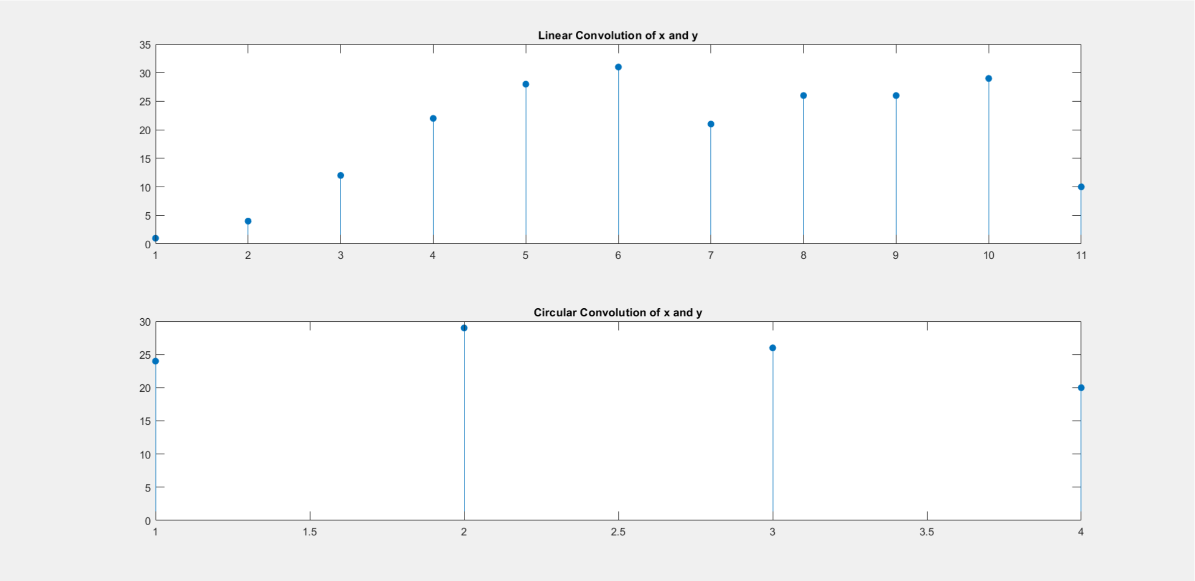
1. Find Linear Convolution and Circular Convolution of L point sequence x[n] and M point sequence h[n].



```
CODE:
a = input("Enter First Signal for Linear Convolution: ");%[1 2 3 -3 0 -1 4]
b = input("Enter Second Signal for Linear Convolution: ");%[1 2 3 1]

circular_a = input("Enter First Signal for Circular Convolution: ");%[1 -2 3
1]
circular_b = input("Enter Second Signal for Circular Convolution: ");%[1 4 2
3]

cref = conv(a,b)
subplot(2,1,1)
stem(cref,'filled')
title('Linear Convolution of x and y')

c = cconv(circular_a, circular_b,4)
subplot(2,1,2)
stem(c,'filled')
title('Circular Convolution of x and y')
```



```
OUTPUT:
>> LinearNCircularConvolution
Enter First Signal for Linear Convolution: [1 2 3 4 1 3 2 5]
Enter Second Signal for Linear Convolution: [1 2 5 2]
Enter First Signal for Circular Convolution: [1 3 5 2]
Enter Second Signal for Circular Convolution: [3 1 2 3]

cref =     1     4    12    22    28    31    21    26    26    29    10
c =    24    29    26    20
```



CONCLUSION: We learnt about in build MATLAB functions for signal plotting and used them to implement Linear and Circular Convolution.

Experiment No. 4

Aim: To apply Discrete Fourier Transform on DT signal

Objective:

1. Write a code to perform DFT of N point signal

2. Calculate DFT of a DT signal

3.Calculate FFT of same signal

Input Specifications:

1. Length of Signal N

2. Signal values

Theory:

Discrete Fourier Transform

Discrete Fourier transform (**DFT**) converts a finite list of equally spaced samples of a function into the list of coefficients of a finite combination of complex sinusoids, ordered by their frequencies, that has those same sample values. It can be said to convert the sampled function from its original domain (often time or position along a line) to the frequency domain.

The input samples are complex numbers (in practice, usually real numbers), and the output coefficients are complex as well. The frequencies of the output sinusoids are integer multiples of a fundamental frequency, whose corresponding period is the length of the sampling interval. The combination of sinusoids obtained through the DFT is therefore periodic with that same period. The DFT differs from the discrete-time Fourier transform (DTFT) in that its input and output sequences are both finite; it is therefore said to be the Fourier analysis of finite-domain (or periodic) discrete-time functions.

The sequence of **N** complex numbers $x_0, x_1, \ldots, x_{N-1}$ is transformed into an **N**-periodic sequence of complex numbers as follows

$$X_k \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z}$$

Problem Definition:

1. Take any four-point & eight-point sequence x[n].

- Find DFT X[k].

Code

```
clc;
signal_input=input("Enter Signal: ");
y=fft(signal_input);
x = (0:length(y)-1)*100/length(y);
m = abs(y);
y(m<1e-6) = 0;
subplot(1,1,1),stem(x, m);
title("Discrete Fourier Transform", 'FontSize', 10);
```

Output

```
Enter Signal: [1 3 4 6 2 6 4 9 12 2 5]
```

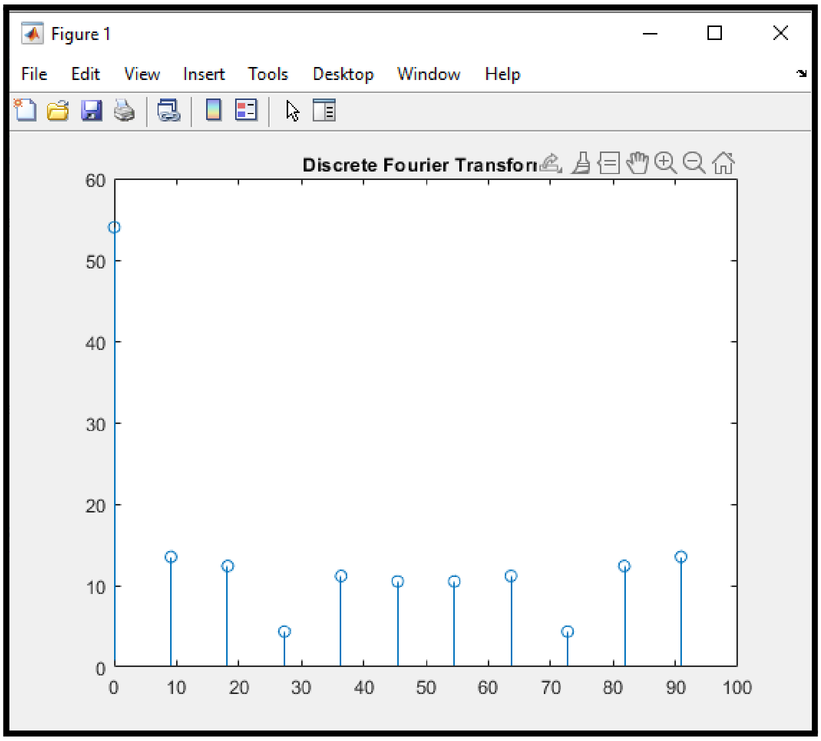


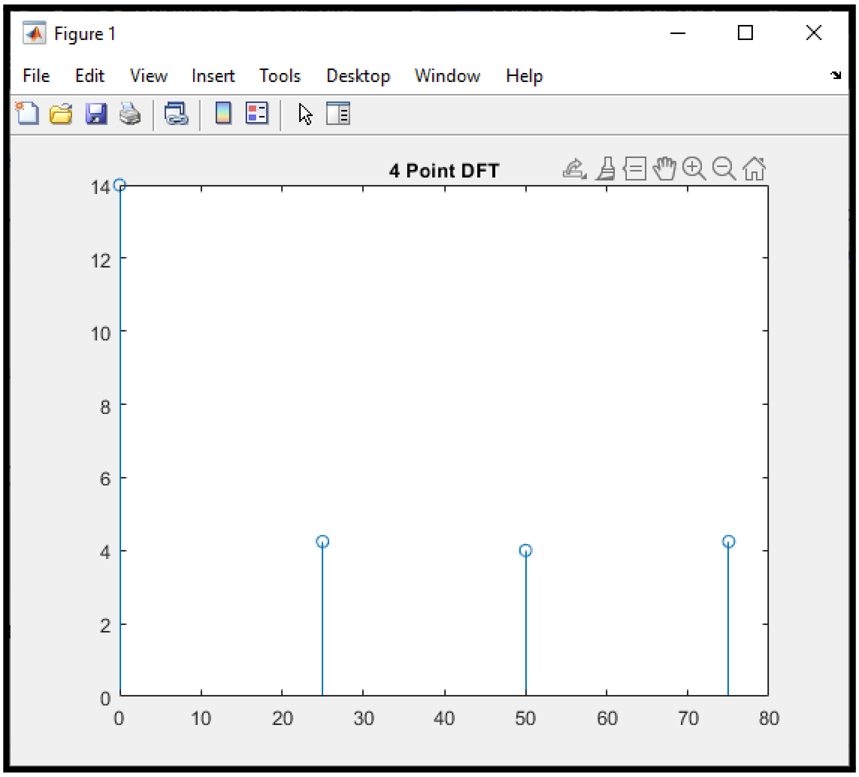
Figure 1
File Edit View Insert Tools Desktop Window Help
Discrete Fourier Transform
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100

Code (4 Point DFT)

```
clc;
signal_input=input("Enter Signal: ");
y=fft(signal_input, 4);
x = (0:length(y)-1)*100/length(y);
m = abs(y);
y(m<1e-6) = 0;
subplot(1,1,1),stem(x, m);
title("4 Point DFT", 'FontSize', 10);
```

Output

```
Enter Signal: [1 3 4 6 2 6 4 9 12 2 5]
```

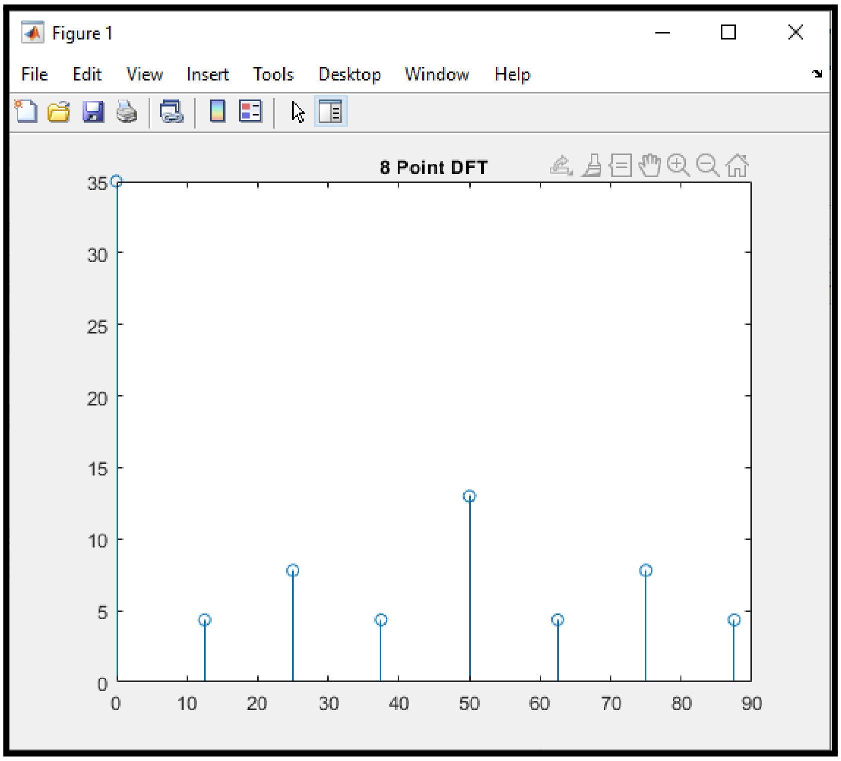


Code (8 Point DFT)

```
clc;
signal_input=input("Enter Signal: ");
y=fft(signal_input, 8);
x = (0:length(y)-1)*100/length(y);
m = abs(y);
y(m<1e-6) = 0;
subplot(1,1,1),stem(x, m);
title("8 Point DFT", 'FontSize', 10);
```

Output:

```
Enter Signal: [1 3 4 6 2 6 4 9 12 2 5]
```



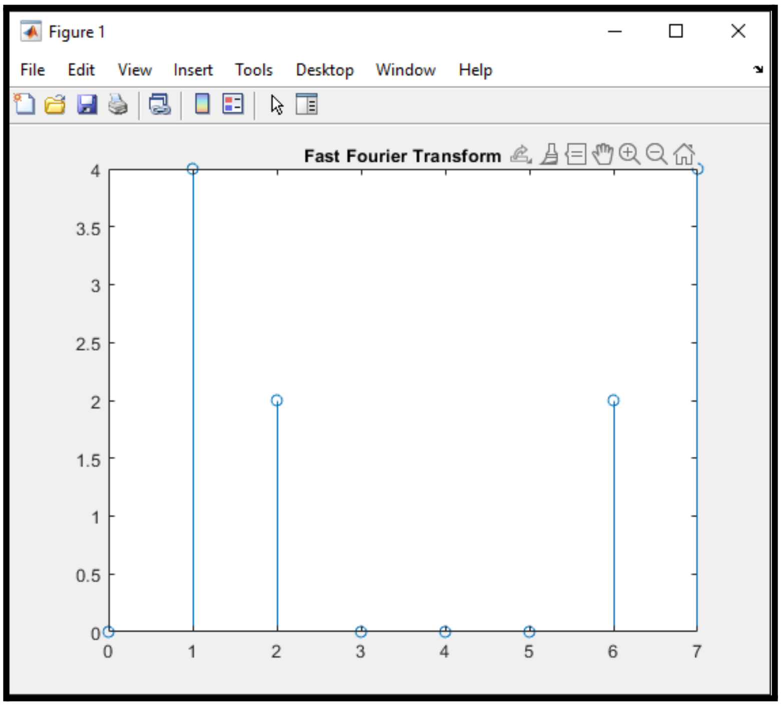
Code (FFT)

```
clc;
input_signal = input('Input N : ');
signal = zeros(1, input_signal);
for k = 0:input_signal-1
signal(1,k+1) = sin((pi/4)*k) + 0.5*sin((pi/2)*k);
end
FFT = fft(signal);
FFT_Value = abs(FFT);
disp(FFT_Value);
range = 0:input_signal-1;
subplot(1,1,1),stem(range, FFT_Value);
title("Fast Fourier Transform", 'FontSize', 10);
```

Output:

```
Input N : 8
```

```
    0.0000    4.0000    2.0000    0.0000    0.0000    0.0000    2.0000    4.0000
```



Conclusion

In this experiment, we learnt about Discrete Fourier Transform and Fast Fourier Transform. We then implemented these using MATLAB.

Experiment No. 5

Aim: To implement Image negative, Gray level Slicing and Thresholding

Objective:

1. Convert an Image from RGB to Gray Level Image

2. Apply Image Negative

3. Apply Gray level Slicing(with and without preserving background)

4. Apply Thresholding

Input Specifications:

- Image of size MxN

Theory:

Image Negative

The negative of an image with grey levels in the range [0, L-1] is obtained by the negative transformation given below.

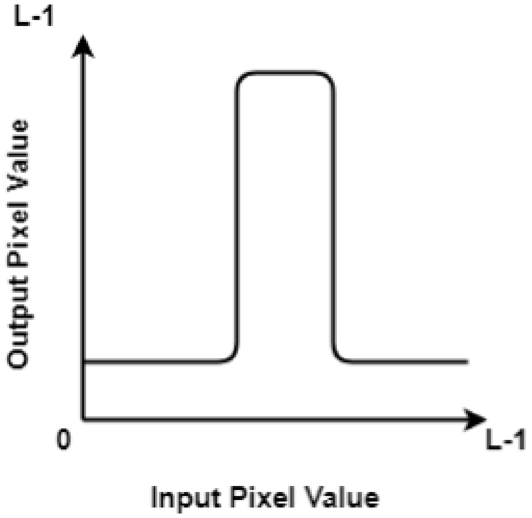
$$s = L - 1 - r.$$

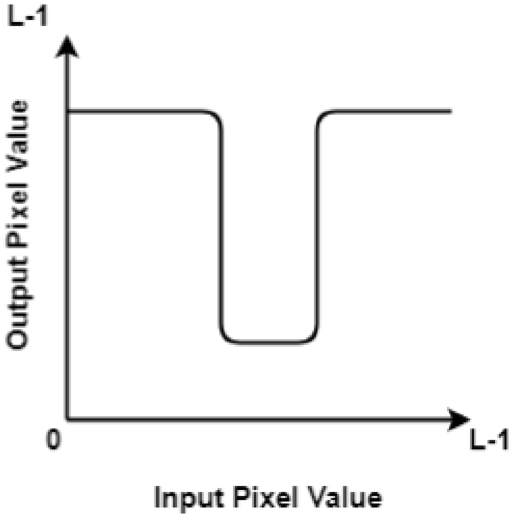
This expression results in reversing of the gray level intensities of the image thereby producing a negative like image. In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image. This is particularly useful for enhancing white or gray details embedded in dark regions of an image.

Gray Level Slicing (Intensity Level Slicing)

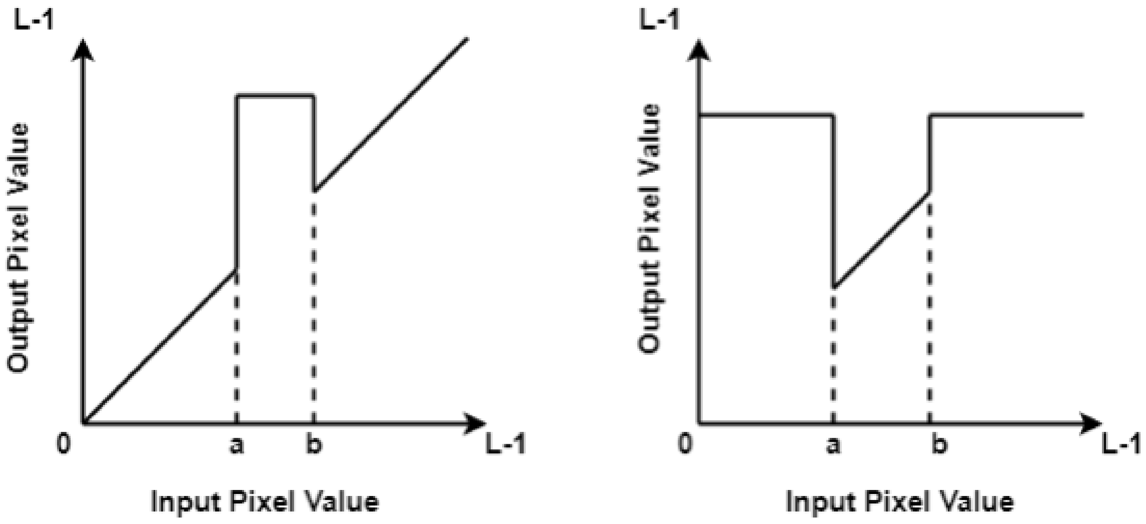
Intensity level slicing means highlighting a specific range of intensities in an image. In other words, we segment certain gray level regions from the rest of the image.

In the first type, we display the desired range of intensities in white and suppress all other intensities to black or vice versa. This results in a binary image. The transformation function for both the cases is shown below.





In the second type, we brighten or darken the desired range of intensities(a to b as shown below) and leave other intensities unchanged or vice versa. The transformation function for both the cases, first where the desired range is changed and second where it is unchanged, is shown below.



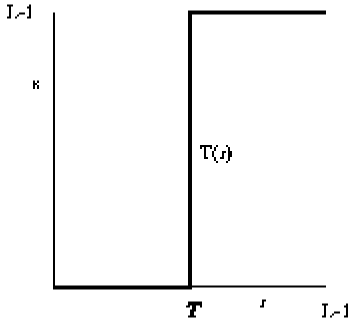
Thresholding

The simplest approach to segment an image is using thresholding.

A simple mapping function is defined by the thresholding operator:

$$s = \begin{cases} 0 & \text{if } r < T \\ L-1 & \text{if } r > T \end{cases}$$

The corresponding graph is shown below



Problem Definition:

1) Take a color image of size MxN
2) Convert Color image to Gray Scale Image
3) Find the highest range(L-1) of gray level in the range [0, L-1] from the image
4) Perform Image Negative.
5) Take Threshold values a and b as input.
6) Apply gray level slicing with and without preserving the background
7) Take Threshold value ‘T as input.
8) Apply thresholding on input image.
9) Conclude by specifying the applications where these operations can be used

Code:

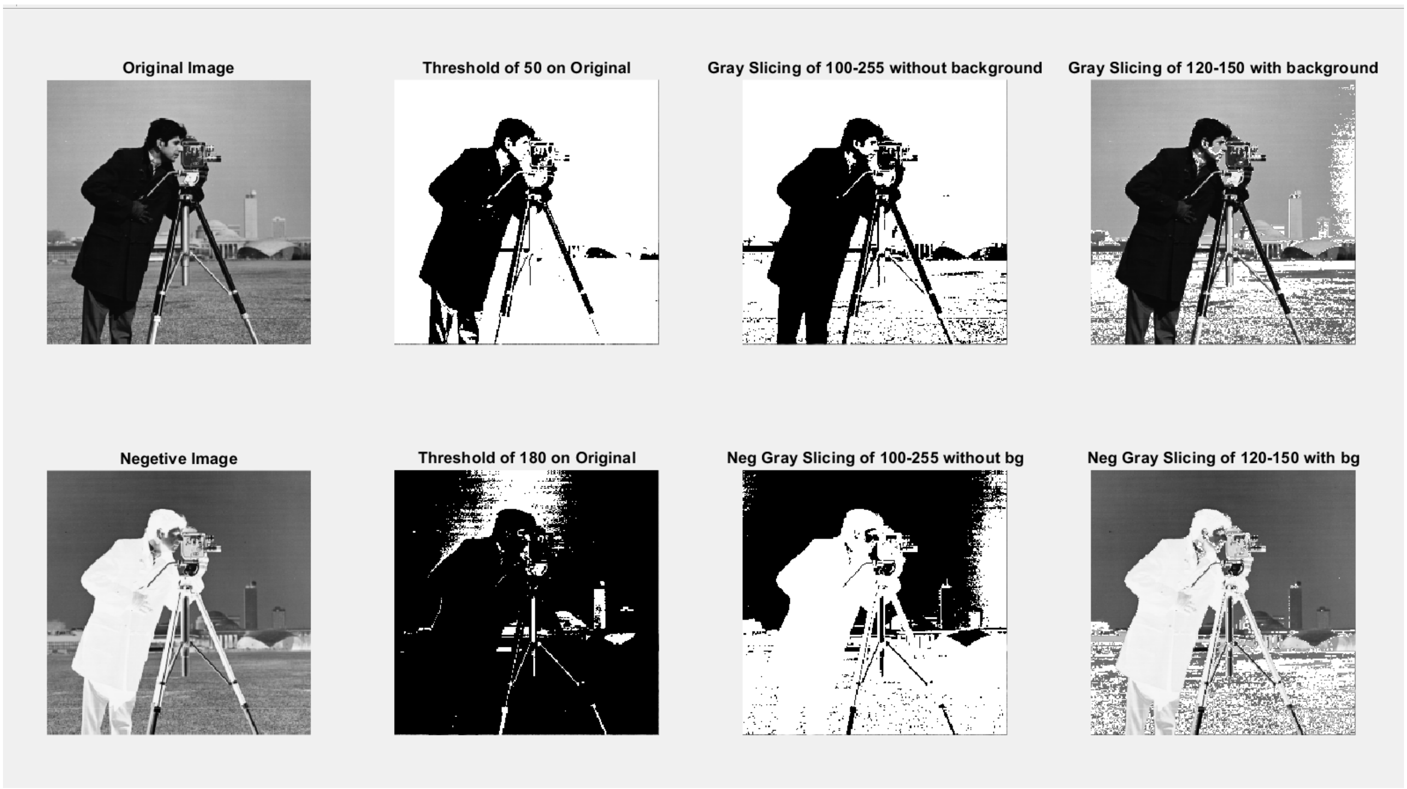
```
clc;
positiveImage = imread('CameraMan.tif');
negativeImage = 255 - positiveImage;
subplot(2,4,1), imshow(positiveImage)
title('Original Image')
subplot(2,4,5), imshow(negativeImage)
title('Negetive Image')
ThresholdImage = imread('CameraMan.tif');
[row , col] = size(ThresholdImage);
t1 = 50;
for i=1:row-1
  for j=1:col-1
    if ThresholdImage(i,j)<t1
      ThresholdImage(i,j) = 0;
    else
      ThresholdImage(i,j) = 255;
    end
  end
```

```
end
subplot(2,4,2), imshow(ThresholdImage)
title(['Threshold of ' , num2str(t1) , ' on Original'])
ThresholdImage2 = imread('CameraMan.tif');
t2 = 180;
for i=1:row-1
  for j=1:col-1
    if ThresholdImage2(i,j)<t2
      ThresholdImage2(i,j) = 0;
    else
      ThresholdImage2(i,j) = 255;
    end
  end
end
subplot(2,4, 6), imshow(ThresholdImage2)
title(['Threshold of ' , num2str(t2) , ' on Original'])
% GREY SLICING
g_slice = positiveImage;
[row, col] = size(positiveImage);
a1 = 100;
a2 = 255;
for i=1:row-1
  for j=1:col-1
    if g_slice(i,j)>a1 && g_slice(i,j)<a2
      g_slice(i,j) = 255;
    else
      g_slice(i,j) = 0;
    end
  end
end
subplot(2,4,3)
imshow(g_slice)
title(['Gray Slicing of ', num2str(a1), '-', num2str(a2), ' without background'])
neg_gslice = negativeImage;
for i=1:row-1
  for j=1:col-1
    if neg_gslice(i,j)>a1 && neg_gslice(i,j)<a2
```

```
      neg_gslice(i,j) = 255;
    else
      neg_gslice(i,j) = 0;
    end
  end
end
subplot(2,4,7)
imshow(neg_gslice)
title(['Neg Gray Slicing of ', num2str(a1), '-', num2str(a2), ' without bg'])
% GRAY SCLICING WITH BACKGROUND
g_slice2 = positiveImage;
a3=120;
a4=150;
for i=1:row-1
  for j=1:col-1
    if g_slice2(i,j)>a3 && g_slice2(i,j)<a4
      g_slice2(i,j) = 255;
    else
      g_slice2(i,j) = positiveImage(i,j);
    end
  end
end
subplot(2,4,4)
imshow(g_slice2)
title(['Gray Slicing of ', num2str(a3), '-', num2str(a4), ' with background'])
neg_g2 = negativeImage;
for i=1:row-1
  for j=1:col-1
    if neg_g2(i,j)>a3 && neg_g2(i,j)<a4
      neg_g2(i,j) = 255;
    else
      neg_g2(i,j) = negativeImage(i,j);
    end
  end
end
subplot(2,4,8)
imshow(neg_g2)
```

```
title(['Neg Gray Slicing of ', num2str(a3), '-', num2str(a4), ' with bg'])
```

Output:


Original Image
Threshold of 50 on Original
Gray Slicing of 100-255 without background
Gray Slicing of 120-150 with background
Negetive Image
Threshold of 180 on Original
Neg Gray Slicing of 100-255 without bg
Neg Gray Slicing of 120-150 with bg

Conclusion:

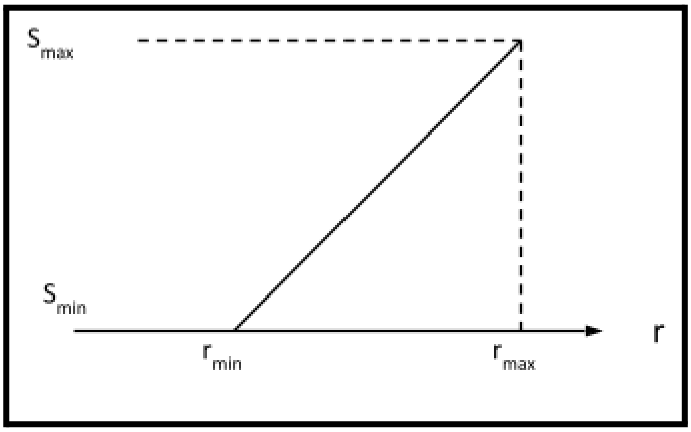
In this experiment, we explored image processing in matlab. We converted images to negative versions, modified pixels based on a threshold, and performed gray slicing of images based on a user-defined threshold.

Experiment No. 6

AIM: To perform Histogram Stretching on an image.

THEORY:
It is a method to increase the dynamic range of the image. Here we do not alter the basic shape of the histogram, but we spread it so as to cover the entire dynamic range. We do this by using a straight line equation having a slope

(Smax – Smin)/ (Rmax - Rmin)



smax = Maximum grey level of output image

smin = Minimum grey level of output image.

rmax = Maximum grey level of input image

rmin = Minimum grey level of input image.

$$S = T(r) = \frac{(smax - smin)}{(rmax - rmin)} (r - r_{min}) + smin$$

This transformation stretches and shifts the grey level range of input image to occupy the entire dynamic range (Smax , Smin).

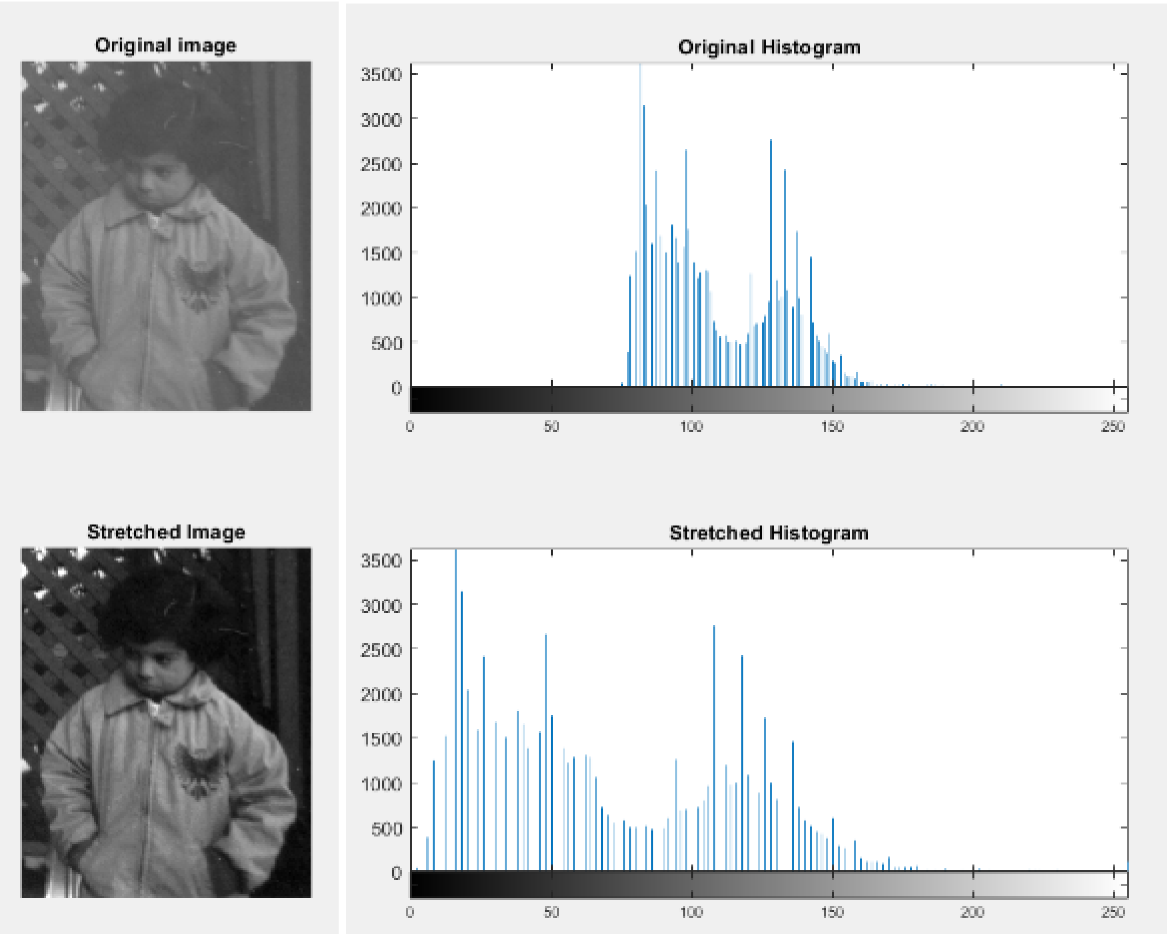
Problem Definition:

1) Take a color image of size MxN
2) Convert Color image to Gray Scale Image
3) Apply Stretching transformation
4) Plot the input image, its histogram, output image and its histogram
5) Conclude by specifying the applications where histogram stretching can be used

CODE:

```
Iimg=imread('pout.tif');
subplot(2,2,1)
imshow(Iimg);
title('Original image');
subplot(2,2,2)
imhist(Iimg);
title('Original Histogram');
rmin = min(min(Iimg));
rmax = max(max(Iimg));
smin = 0;
smax = 255;
y=(((smax-smin)/(rmax-rmin))*(Iimg-rmin))+smin;
subplot(2,2,3)
imshow(y);
title('Stretched Image');
subplot(2,2,4)
imhist(y);
title('Stretched Histogram');
```

OUTPUT:



CONCLUSION:

Histogram stretching is one of the best methods to improve contrast in an image using MATLAB. In this experiment, we have done image processing on ‘pouf.tif’ image and have displayed the histogram before and after stretching.

Experiment No. 7

Aim: To implement Histogram Equalization for an image

Objective:

Apply CDF for histogram equalization of an image

Input Specifications:

Colour Image of size MXN

Theory:

Histogram equalization is a method to process images in order to adjust the contrast of an image by modifying the intensity distribution of the histogram. The objective of this technique is to give a linear trend to the cumulative probability function associated to the image.

The processing of histogram equalization relies on the use of the cumulative probability function (cdf). The cdf is a cumulative sum of all the probabilities lying in its domain and defined by:

$$cdf(x) = \sum_{k=-\infty}^{x} P(k)$$

The idea of this processing is to give to the resulting image a linear cumulative distribution function.

Problem Definition:

1) Take a color image of size MxN
2) Convert Color image to Gray Scale Image
3) Count total no. of pixels associated with each pixel intensity
4) Calculate probability of each pixel intensity in a image matrix.
5) Calculate cumulative probability
6) Normalize the cumulative probability
7) Round off each value.

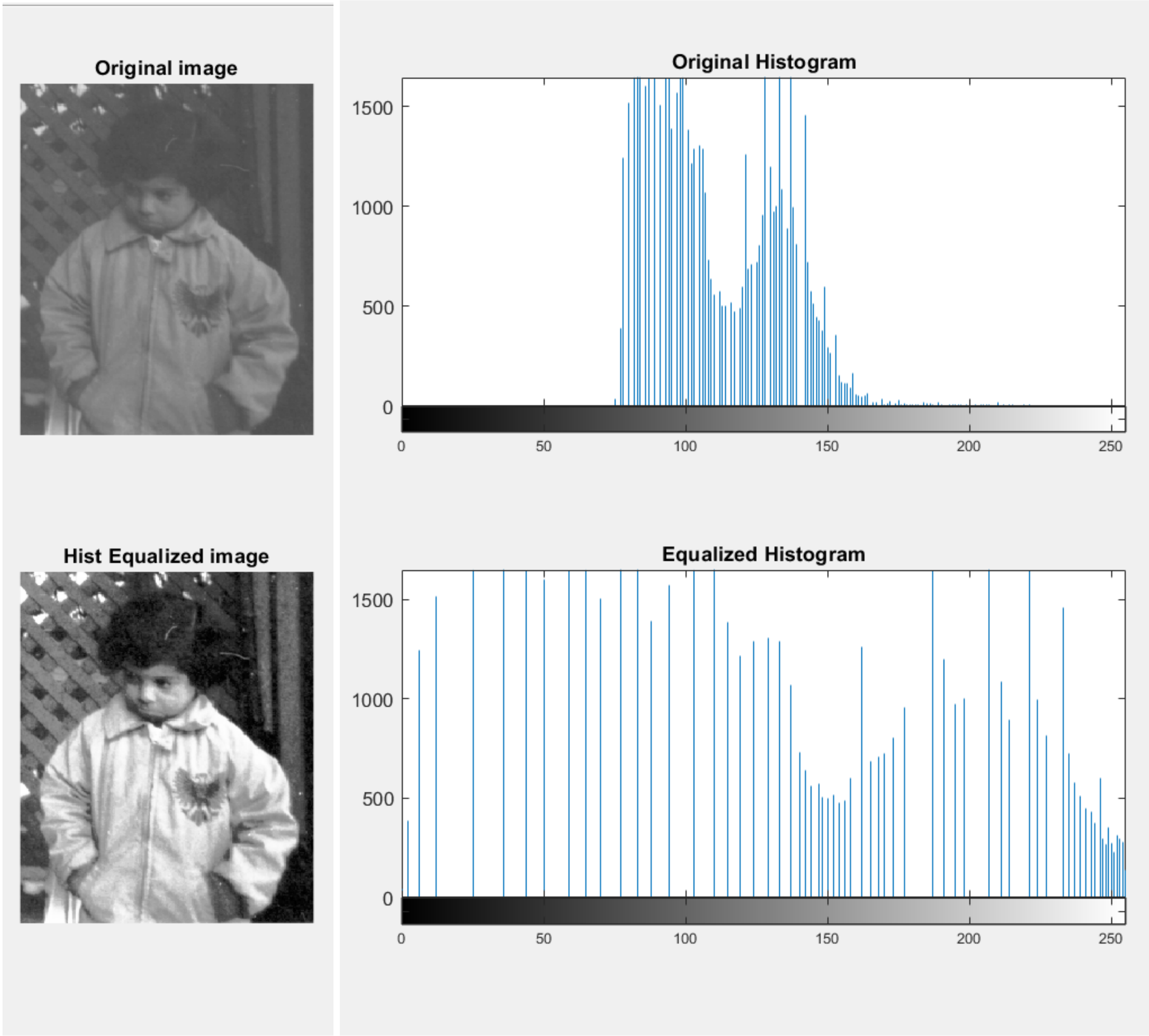
8) Plot the input image, its histogram, output image and its histogram
9) Conclude by specifying the applications where histogram equalization can be used

CODE:

```
GIm=imread('pout.tif');
numofpixels=size(GIm,1)*size(GIm,2);
subplot(2,2,1)
imshow(GIm);
title('Original image');
subplot(2,2,2)
imhist(GIm);
title('Original Histogram');
%--------------------------------------------------------
HIm=uint8(zeros(size(GIm,1),size(GIm,2)));
freq=zeros(256,1);
probf=zeros(256,1);
probc=zeros(256,1);
cum=zeros(256,1);
output=zeros(256,1);
%freq counts the occurrence of each pixel value.
%The probability of each occurrence is calculated by probf.
for i=1:size(GIm,1)
  for j=1:size(GIm,2)
     value=GIm(i,j);
     freq(value+1)=freq(value+1)+1;
     probf(value+1)=freq(value+1)/numofpixels;
  end
end
sum=0;
no_bins=255;
%The cumulative distribution probability is calculated.
for i=1:size(probf)
 sum=sum+freq(i);
 cum(i)=sum;
```

```
  probc(i)=cum(i)/numofpixels;
  output(i)=round(probc(i)*no_bins);
end
for i=1:size(GIm,1)
  for j=1:size(GIm,2)
        HIm(i,j)=output(GIm(i,j)+1);
  end
end
subplot(2,2,3)
imshow(HIm);
title('Hist Equalized image');
subplot(2,2,4)
imhist(HIm);
title('Equalized Histogram');
```

OUTPUT:



APPLICATIONS:

- To achieve better quality images in B&W color scales in medical applications like X-rays, MRI's and CT scans.
- To better detail in photographs that are either over or under-exposed.

CONCLUSION:

Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. In this practical, we did histogram equalization on an image and observed the changes in the histogram using MATLAB.

Experiment No. 8

Aim: To implement Image smoothing/ Image sharpening using low pass and high pass filter

Objective:

1) Apply Low pass filter to get image smoothening

2) Apply High pass filter to get image sharpening

Input Specifications:

Colour Image of size MXN

Theory:

Image Smoothening (Low Pass Filering)

A low pass filter is the basis for most smoothing methods. An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels

Using a low pass filter tends to retain the low frequency information within an image while reducing the high frequency information. An example is an array of ones divided by the number of elements within the kernel, such as the following 3 by 3 kernel:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Image Sharpening (High Pass Filering)

A high pass filter is the basis for most sharpening methods. An image is sharpened when contrast is enhanced between adjoining areas with little variation in brightness or darkness

A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information. The kernel of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels. The kernel array usually contains a single positive value at its center, which is completely surrounded by negative values. The following array is an example of a 3 by 3 kernel for a high pass filter:

$$\begin{bmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{bmatrix}$$

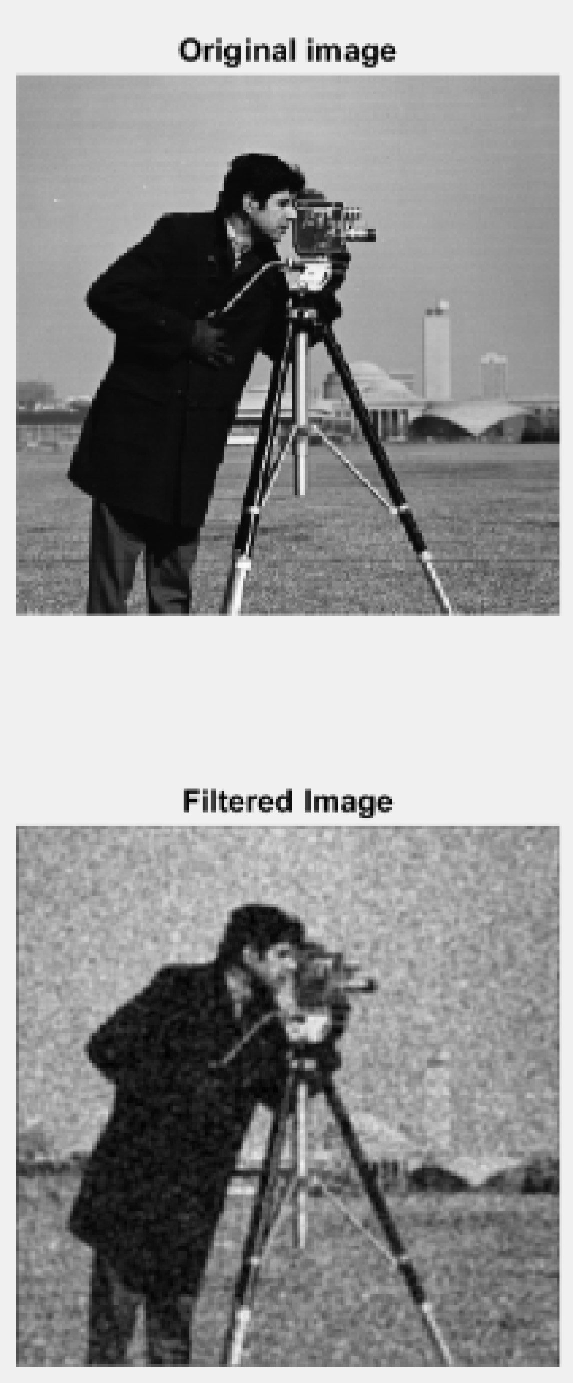
Problem Definition:

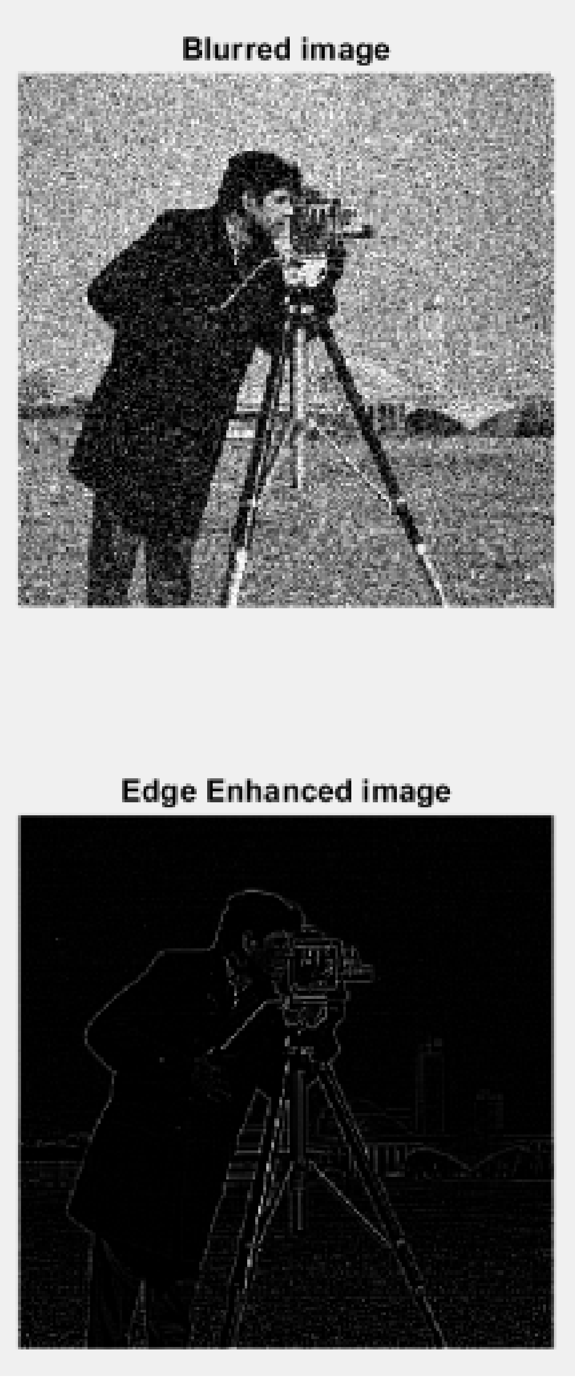
1) Take a color image of size MxN
2) Convert color image to Gray Scale Image
3) Perform convolution of image with low pass filter mask
4) Perform convolution of image with high pass filter mask
5) Conclude by specifying the applications where LPF and HPF can be used

CODE:

```
clc;
Iimg=imread('cameraman.tif');
subplot(2,2,1)
imshow(Iimg);
title('Original image');
Iimg = imnoise(Iimg,'gaussian',0,0.025);
subplot(2,2,2)
imshow(Iimg);
title('Blurred image');
filtered = filter2(fspecial('average',3),Iimg)/255;
subplot(2,2,3)
imshow(filtered);
title('Filtered Image');
original_image=imread('cameraman.tif');
kernel = 1/6*-1*ones(3);
kernel(2,2) = 1/6*8;
enhancedImage = imfilter(original_image, kernel);
subplot(2,2,4)
imshow(enhancedImage);
title('Edge Enhanced image');
```

OUTPUT:


Original image
Filtered Image


Blurred image
Edge Enhanced image

CONCLUSION:

In this experiment, we leant how to add noise to images using MATLAB, how to filter noise from images using a 3*3 mask and how to do edge detection using high pass filter in MATLAB.

Experiment No. 9

Aim: To implement Edge detection using Sobel and Prewitt masks

Objective:

Apply Sobel and Prewitt mask for edge detection

Input Specifications:

Colour Image of size MXN

Theory:

Sobel Mask

Following is the vertical Mask of Sobel Operator:

-1	0	1
-2	0	2
-1	0	1

This mask works exactly same as the Prewitt operator vertical mask. There is only one difference that is it has “2” and “-2” values in center of first and third column. When applied on an image this mask will highlight the vertical edges.

How it works

When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in a edge region.

As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also the center values of both the first and third column is 2 and -2 respectively.

This give more weight age to the pixel values around the edge region. This increase the edge intensity and it become enhanced comparatively to the original image.

Following is the horizontal Mask of Sobel Operator

-1	-2	-1
0	0	0
1	2	1

Above mask will find edges in horizontal direction and it is because that zeros column is in horizontal direction. When you will convolve this mask onto an image it would prominent horizontal edges in the image. The only difference between it is that it have 2 and -2 as a center element of first and third row.

How it works

This mask will prominent the horizontal edges in an image. It also works on the principle of above mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask is consist of zeros so it does not include the original values of edge in the image but rather it calculate the difference of above and below pixel intensities of the particular edge. Thus increasing the sudden change of intensities and making the edge more visible.

Prewitt Mask

The Prewitt operator is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Prewitt operator is either the corresponding gradient vector or the norm of this vector. The Prewitt operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical directions and is therefore relatively inexpensive in terms of computations

Following is the vertical Mask of Prewitt Operator:

-1	0	1
-1	0	1
-1	0	1

Following is the horizontal Mask of Prewitt Operator

-1	-1	-1
0	0	0
1	1	1

Problem Definition:

1) Take a color image of size MxN
2) Convert color image to Gray Scale Image
3) Apply prewitt and sobel mask vertically and horizontally.
4) Conclude by specifying the applications where LPF and HPF can be used

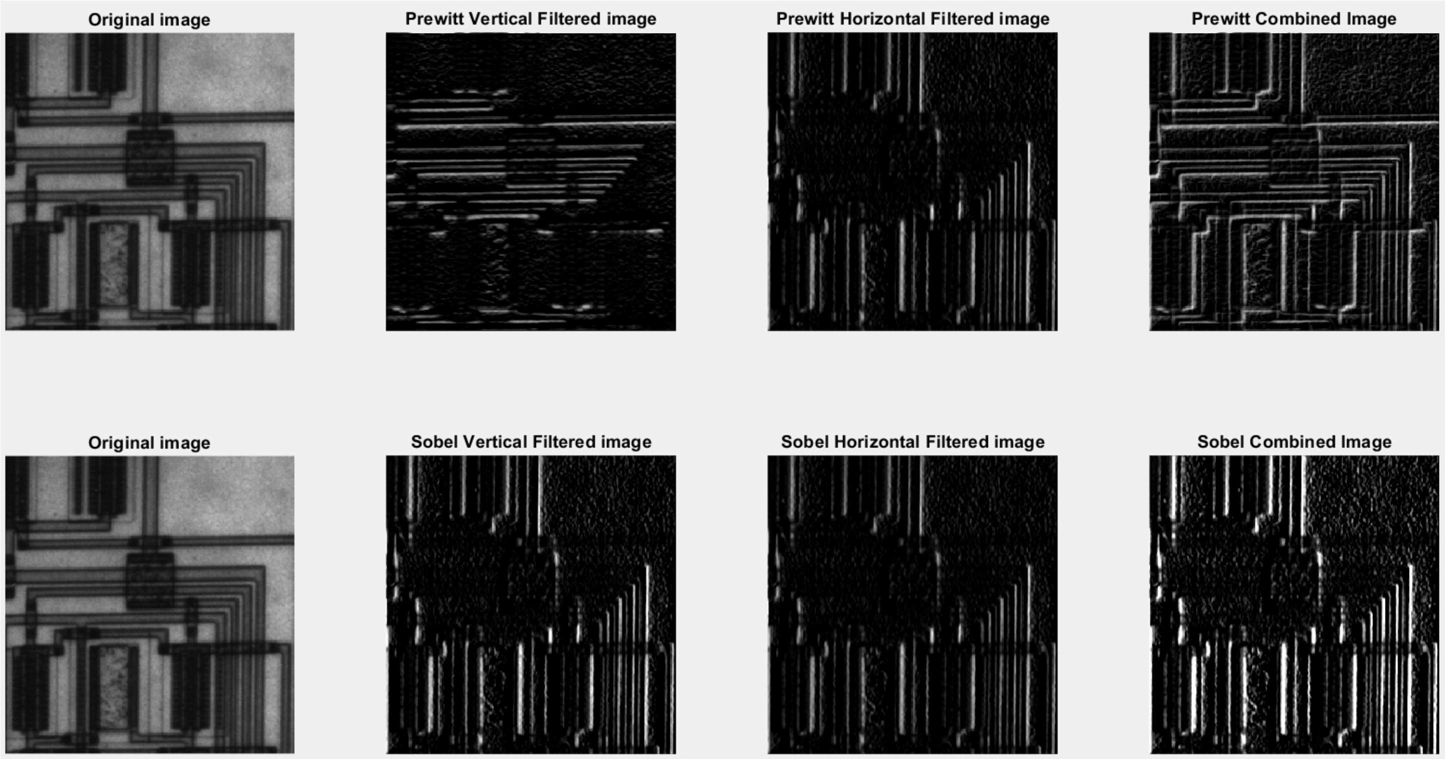
CODE:

```
clc;
Iimg=imread('circuit.tif');
subplot(2,4,1)
imshow(Iimg);
title('Original image');
kernel = ones(3);
kernel(1,1) = -1;
kernel(1,2) = -1;
kernel(1,3) = -1;
kernel(2,1) = 0;
kernel(2,2) = 0;
kernel(2,3) = 0;
kernel(3,1) = 1;
kernel(3,2) = 1;
kernel(3,3) = 1;
enhancedImage = imfilter(Iimg, kernel);
subplot(2,4,2)
imshow(enhancedImage);
title('Prewitt Vertical Filtered image');
vertical_kernel = zeros(3);
vertical_kernel(1,1) = -1;
vertical_kernel(1,2) = 0;
vertical_kernel(1,3) = 1;
vertical_kernel(2,1) = -1;
vertical_kernel(2,2) = 0;
vertical_kernel(2,3) = 1;
vertical_kernel(3,1) = -1;
vertical_kernel(3,3) = 0;
vertical_kernel(3,3) = 1;
horizontal_enhancedImage = imfilter(Iimg, vertical_kernel);
subplot(2,4,3)
imshow(horizontal_enhancedImage);
title('Prewitt Horizontal Filtered image');
combined_image = enhancedImage + horizontal_enhancedImage;
subplot(2,4,4)
imshow(combined_image);
```

```
title('Prewitt Combined Image');
% SOBEL
Iimg=imread('circuit.tif');
subplot(2,4,5)
imshow(Iimg);
title('Original image');
sobel_kernel = ones(3);
sobel_kernel(1,1) = -1;
sobel_kernel(1,2) = 0;
sobel_kernel(1,3) = 1;
sobel_kernel(2,1) = -2;
sobel_kernel(2,2) = 0;
sobel_kernel(2,3) = 2;
sobel_kernel(3,1) = -1;
sobel_kernel(3,2) = 0;
sobel_kernel(3,3) = 1;
sobel_vertical_enhancedImage = imfilter(Iimg, sobel_kernel);
subplot(2,4,6)
imshow(sobel_vertical_enhancedImage);
title('Sobel Vertical Filtered image');
sobel_horizontal_kernel = zeros(3);
sobel_horizontal_kernel(1,1) = -1;
sobel_horizontal_kernel(1,2) = 0;
sobel_horizontal_kernel(1,3) = 1;
sobel_horizontal_kernel(2,1) = -1;
sobel_horizontal_kernel(2,2) = 0;
sobel_horizontal_kernel(2,3) = 1;
sobel_horizontal_kernel(3,1) = -1;
sobel_horizontal_kernel(3,3) = 0;
sobel_horizontal_kernel(3,3) = 1;
sobel_horizontal_enhancedImage = imfilter(Iimg, sobel_horizontal_kernel);
subplot(2,4,7)
imshow(sobel_horizontal_enhancedImage);
title('Sobel Horizontal Filtered image');
sobel_combined_image = sobel_vertical_enhancedImage +
sobel_horizontal_enhancedImage;
subplot(2,4,8)
```

```
imshow(sobel_combined_image);
title('Sobel Combined Image');
```

OUTPUT:


Original image
Prewitt Vertical Filtered image
Prewitt Horizontal Filtered image
Prewitt Combined Image
Original image
Sobel Vertical Filtered image
Sobel Horizontal Filtered image
Sobel Combined Image

APPLICATIONS:

- In the case of audio electronics, Low pass filters are used to remove high frequencies from low frequency components.
- A low pass filter removes unwanted frequencies from interfering with the function of a bass speaker.
- A high pass filter will remove unwanted low frequencies from your tweeters.

CONCLUSION:

In this experiment, we learnt about Low pass filtering and High pass filtering. We then implemented Sobel and Prewitt filtering on images using MATLAB.