LAB MANUAL

Experiment 1

AIM: Generation of the following discrete signals using MATLAB. (i) Unit step (ii) unit impulse (iii) unit ramp
(iv) SINC (v) Gaussian

Objective:

1. To generate all basic signals and plot the same as a function of time.
2. To plot the 2D and 3D signals such as LIDAR and Image

Theory:

There are several elementary or basic signals which are used to model a large number of physical signals which occur in nature. These elementary signals are also called Standard Signals. Some of these signals are described below

i) Unit Impulse Function:
An ideal impulse function is a function that is zero everywhere but at the origin, where it is infinitely high.

$$\delta(n) = 1 \; at \; n = 0$$

ii) Unit Step Function:
The unit step function, u(t) is defined as

$$u(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}$$

That is, *u* is a function of time *t*, and *u* has a value of zero when time is negative and a value of one when time is positive.

iii) Unit Ramp Function:

A ramp function or ramp signal is a type of standard signal which starts at □ = 0 and increases linearly with time. The unit ramp function has unit slope.

$$r(t) = \begin{cases} 0 & \text{for } t<0 \\ t & \text{for } t\geq 0 \end{cases}$$

$$r(t) = t.u(t)$$

iv) Sinc Pulse:

A sinc function is an even function with a unit area. A sinc pulse passes through zero at all positive and negative integers (i.e., t=±1,±2,…), but at time t=0, it reaches its maximum of 1. This is a very desirable property in a pulse, as it helps to avoid inter symbol interference, a major cause of degradation in digital transmission systems. The product of a sinc function and any other signal would also guarantee zero crossings at all positive and negative integers.

The normalized sinc function is commonly defined for $x \neq 0$ by

$$\operatorname{sinc} x = \frac{\sin(\pi x)}{\pi x}$$

v) Gaussian Pulse

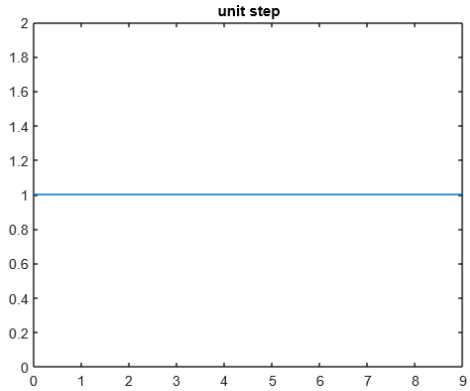
In one dimension, the Gaussian function is the probability density function of the normal distribution,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)},$$

MATLAB Code:

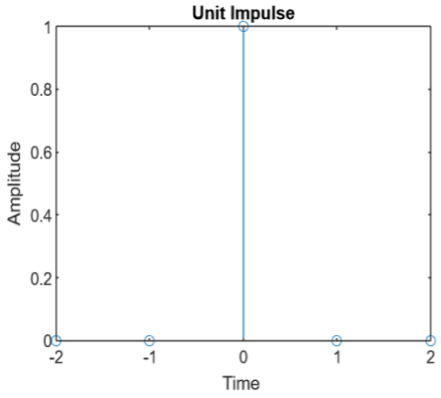
i) Impulse signal

```
%Generation of UNIT impulse signal
clc; close all; clear all;
n=-2:1:2;
y=[zeros(1,2),ones(1,1),zeros(1,2)]
figure(1)
stem(n,y);
xlabel("Time ")
ylabel("Amplitude")
title('unit impulse');
```



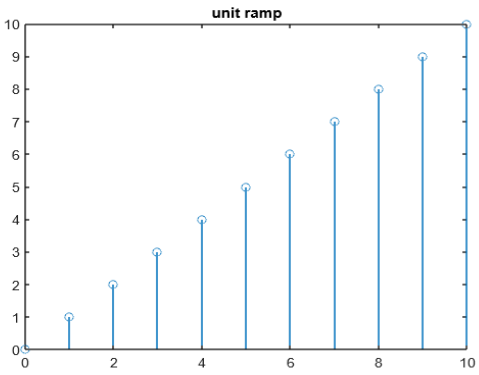
ii) %Generation of UNIT step signal

```
clc; close all; clear all;
n=input('enter the n value');
t=0:1:n-1;
y=ones(1,n);
figure(2)
plot(t,y);
title('unit step');
xlabel("Time (sec)")
ylabel("Amplitude")
```



iii)%Generation of unit RAMP signal

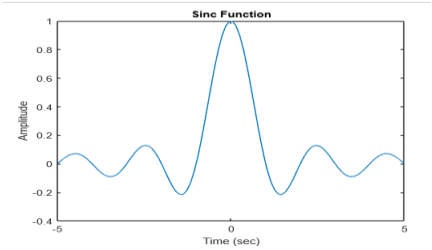
```
clc; close all; clear all;
n=input('enter the n value');
t=0:n;
y=t;
figure(3)
stem(y,t);
title('unit ramp');
xlabel("Time (sec)")
ylabel("Amplitude")
```



iv) %Generation of sinc pulse

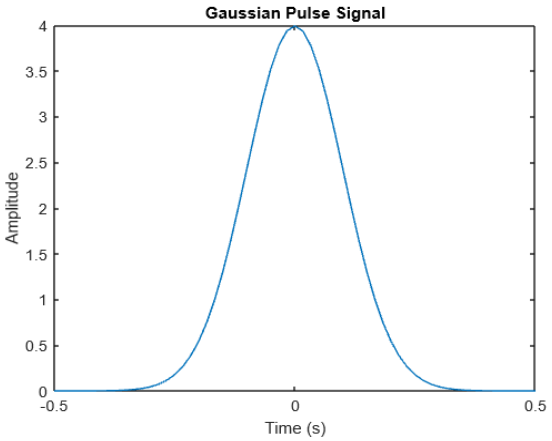
```
t1 = linspace(-5,5);
y1 = sinc(t1);
plot(t1,y1)
xlabel("Time (sec)")
ylabel("Amplitude")
```

title("Sinc Function")



v)%Generation of Gaussian pulse

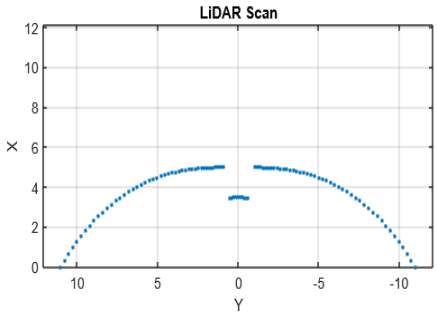
```
Fs = 60; % sampling freq
t = -.5:1/Fs:.5;
x = 1/(sqrt(2*pi*0.01))*(exp(-t.^2/(2*0.01)));
figure(1);
plot(t,x);
title('Gaussian Pulse Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```



vi) Generation of 2D LIDAR

```
x = linspace(-2,2);
ranges = abs((1.5).*x.^2 + 5);
```

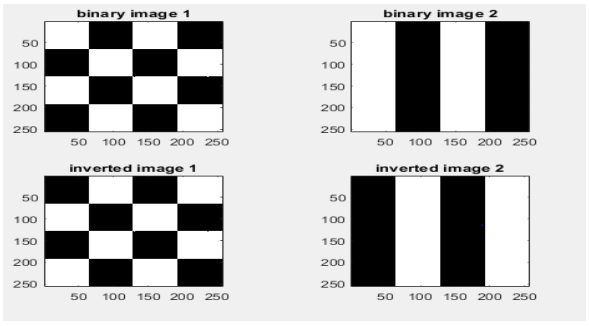
```
ranges(45:55) = 3.5;
angles = linspace(-pi/2,pi/2,numel(ranges));
scan = lidarScan(ranges,angles);
plot(scan)
```



vii) Generation of 2D Binary image:

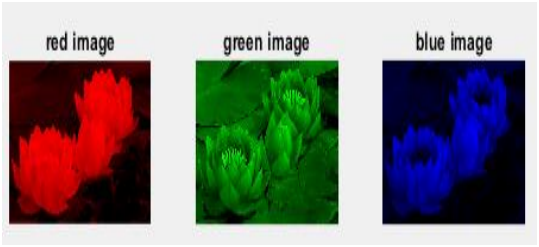
```
% create black and white image
clc;% clear command window
clear all;% clear workspace
close all;% clear all figures
w = ones(64,64);
b = zeros(64,64);
bin1= [w b w b
       b w b w
       w b w b
       b w b w];
bin2 = [w b w b
        w b w b
        w b w b
        w b w b];
subplot(2,2,1);subimage(bin1); title('binary image 1');
subplot(2,2,2);subimage(bin2); title('binary image 2');
imwrite(bin1,'bin_image1.tif');
```

```
imwrite(bin2,'bin_image2.tif');
i1 = not(bin1);
i2 = not(bin2);
% for block & white image use subimage
subplot(2,2,3);subimage(i1); title('inverted image 1');
subplot(2,2,4);subimage(i2); title('inverted image 2');
%rgb2gray
clc; % clear command window
clear all;% clear workspace
close all;% clear all figures
  I1=imread('D:\waterlily.jpg');
figure;
imshow(I1);
title('color image');
I2=rgb2gray(I1);
figure;
imshow(I2);
title('Gray image');
subplot(2,2,1);subimage(I1); title('Color Image');
subplot(2,2,2);subimage(I2); title('Gray Image');
```



% reading and displaying color image

```
clc;% clear command window
clear all; close all;
a = imread('D:\waterlily.jpg');
[row col dim] = size(a);
figure(1); imshow(a); title('original image');
red = a(:,:,1);% gray scale image of the red plane
green = a(:,:,2);% gray scale image of the green plane
blue = a(:,:,3);% gray scale image of the blue plane
plane = zeros(row,col);
RED = cat(3,red,plane,plane);
GREEN = cat(3,plane,green,plane);
BLUE =  cat(3,plane,plane,blue);
figure(3);
subplot(1,3,1);imshow(RED),title('red image');
subplot(1,3,2);imshow(GREEN),title('green image');
subplot(1,3,3);imshow(BLUE),title('blue image');
```



Practice Questions:

Write a code in Python to generate all the basic signals and plot the same using appropriate library functions.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 2

AIM: Perform basic operations: time shifting, time scaling and time reversal for the basic signals and plot them as a function of time.

Objective:

1. To perform basic operations on the dependent variable of signals and observe the behavior of signals for different values of amplitude.
2. To perform basic operations on independent variables of signals as a function of time with appropriate shifting and scaling.

Theory:

Basic operations on Signals

The basic set of signal operations can be broadly classified as below.

1. Basic Signal Operations Performed on Dependent Variables

In this transformation, only the quadrature axis values are modified i.e magnitude of the signal changes, with no effects on the horizontal axis values or periodicity of signals like.

- Amplitude scaling of signals
- Addition of signals.
- Multiplication of signals.
- Differentiation of signals.
- Integration of signals.

2. Basic Signal Operations Performed on Independent Variables

- Time Shifting
- Time Scaling
- Time Reversal

Time Shifting

A signal x(t) may be shifted in time by replacing the independent variable *t* by either $t-t_0$ or $t+t_0$. Here t_0 is called the *shifting factor*. Shifting in time may result in time delay or time advancement.

If the independent variable t is replaced by $t-t_0$, the signal is shifted to the right, and the time shift results in a delay of the signal by t_0 units of time. This type of time shifting is known as Right side shifting. This can be achieved by adding t_0 value to every instant in signal x(t).

If the independent variable t is replaced by $t+t_0$, the signal is shifted to the left and the time shift results in an advancement of the signal by t_0 units of time. This type of time shifting is known as Left side shifting. This can be achieved by subtracting t_0 value to every time instant in signal x(t).

Time Scaling

A signal x(t) may be scaled in time by replacing the independent variable *t* with *at*. Here '*a'* is called the *scaling factor*. Time scaling may result in signal compression or signal expansion.

If the independent variable *t* is replaced by *at* and a>1, the signal is *compressed*. This can be achieved by dividing every instant in signal x(t) by '*a*'.

If the independent variable *t* is replaced by *at* and 0<a<1, the signal is *expanded*. This can be achieved by dividing every instant in signal x(t) by '*a*'.

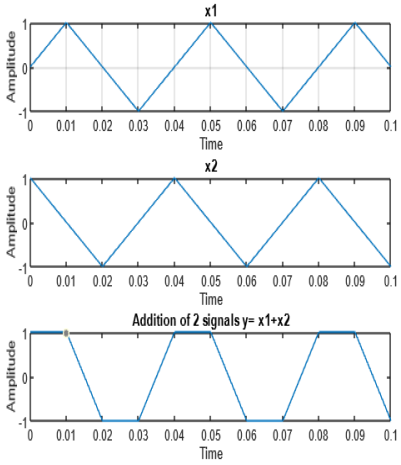
Time Reversal

If the independent variable *t* is replaced by '$-t$' , this operation is known as time reversal of the signal about the y-axis or amplitude axis. This can be achieved by taking a mirror image of the signal x(t) about the y-axis or by rotating x(t) by 180° about the y-axis. Hence, time reversal is known as *folding* or *reflection*.

MATLAB Code:

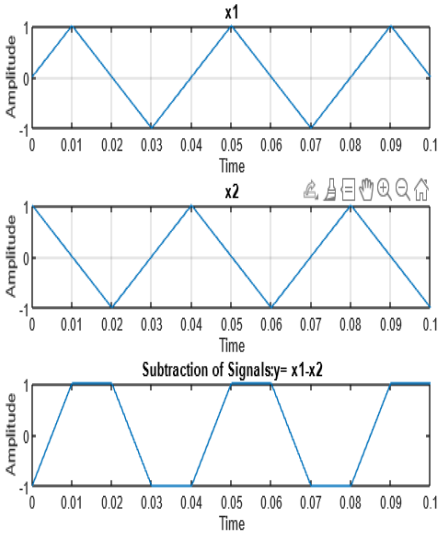
i) Addition of two signals

```
t=0:0.01:0.1
f=25;
t1=2*pi*f*t;
x1=sin(t1);
subplot(3,1,1)
plot(t,x1)
title('x1')
xlabel('Time')
ylabel('Amplitude')
grid on;
x2=cos(t1)
subplot(3,1,2)
plot(t,x2)
title('x2')
xlabel('Time')
ylabel('Amplitude')
y=x1+x2
subplot(3,1,3)
plot(t,y)
title('Addition of 2 signals y= x1+x2')
```



```
xlabel('Time')
ylabel('Amplitude')
```

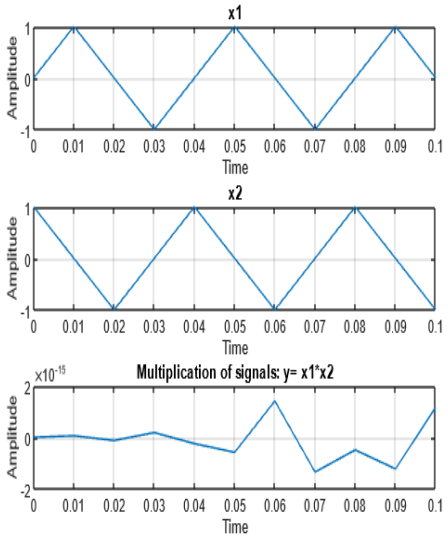
ii) Subtraction of Signals




```
t=0:0.01:0.1
f=25;
t1=2*pi*f*t;
x1=sin(t1);
subplot(3,1,1)
plot(t,x1)
title('x1')
xlabel('Time')
ylabel('Amplitude')
grid on;
x2=cos(t1)
subplot(3,1,2)
grid on;
plot(t,x2)
title('x2')
xlabel('Time')
ylabel('Amplitude')
grid on;
y=x1-x2 % Here the subtraction takes place
subplot(3,1,3)
grid on;
plot(t,y)
title('Subtraction of Signals:y= x1-x2')
xlabel('Time')
ylabel('Amplitude')
```

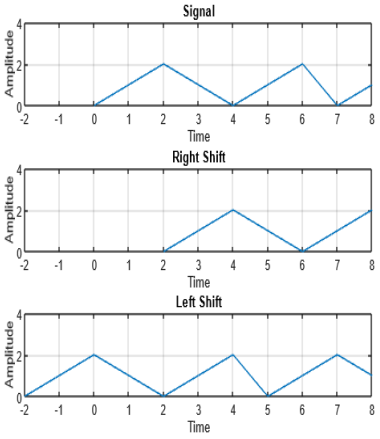
iii) Multiplication of Signals

```
t=0:0.01:0.1
f=25;
t1=2*pi*f*t;
x1=sin(t1);
subplot(3,1,1)
plot(t,x1)
title('x1')
xlabel('Time')
ylabel('Amplitude')
grid on;
x2=cos(t1)
subplot(3,1,2)
plot(t,x2)
title('x2')
xlabel('Time')
ylabel('Amplitude')
grid on;
y=x1.*x2 % Here the multiplication takes place
subplot(3,1,3)
plot(t,y)
title('Multiplication of signals: y= x1*x2')
xlabel('Time')
ylabel('Amplitude')
grid on;
```

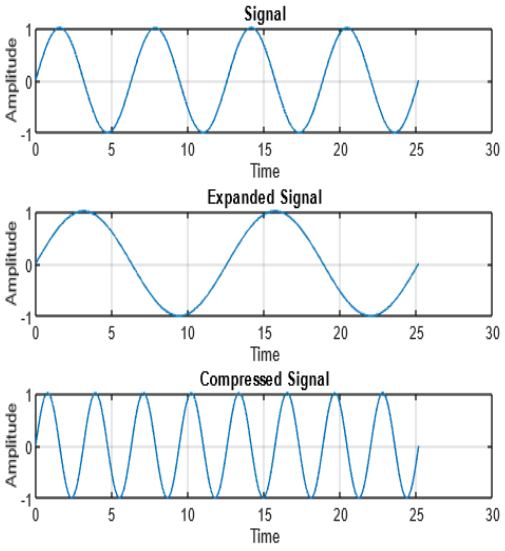


i) Time Shifting of Signals

```
t=0:10;
x=[0 1 2 1 0 1 2 0 1 2 1 ];
subplot(3,1,1)
plot(t,x)
title('Signal')
xlabel('Time')
ylabel('Amplitude')
```

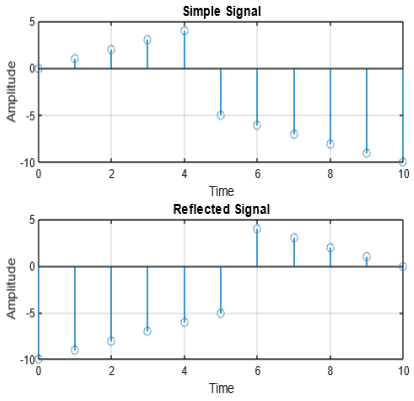


```
grid on;
axis([-2 8 0 4]);
subplot(3,1,2)
plot(t+2,x)
title('Left Shift')
xlabel('Time')
ylabel('Amplitude')
grid on;
axis([-2 8 0 4]);
subplot(3,1,3)
plot(t-2,x)
title('Right Shift')
xlabel('Time')
ylabel('Amplitude')
grid on;
axis([-2 8 0 4]);
```



ii)Time Scaling of Signals

```
t=0:0.01:8*pi
x=sin(t);
subplot(3,1,1)
plot(t,x)
title('Signal')
xlabel('Time')
ylabel('Amplitude')
grid on;
y=sin(t/2)
subplot(3,1,2)
plot(t,y)
title('Expanded Signal')
xlabel('Time')
ylabel('Amplitude')
grid on;
z=sin(t*2)
subplot(3,1,3)
```



```
plot(t,z)
title('Compressed Signal')
xlabel('Time')
ylabel('Amplitude')
grid on;
```

iii)Time reversal

```
t=0:10;
x=[0 1 2 3 4 -5 -6 -7 -8 -9 -10];
subplot(2,1,1)
stem(t,x)
title('Simple Signal')
xlabel('Time')
ylabel('Amplitude')
grid on;
y=fliplr(x);
subplot(2,1,2)
stem(t,y)
title('Reflected Signal')
xlabel('Time')
ylabel('Amplitude')
grid on;
```

Practice Questions:

Write the MATLAB Code to sketch the following signals and verify the same using analytical method

a. r(t+2)-r(t+1)-r(t-2)+r(t-3)
b. u(n+2)-3u(n-1)+2u(n-5)
c. $\delta(t) + u(t-3) + r(t+4) - \delta(t-2)$

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 3

AIM: Perform the system to find given system is linear, stable and casual.

Objective:

1. Write a MATLAB program to find whether the given systems are linear casual and stable.

Theory:

A system is linear if and only if T [a1x1[n] + a2x2n]] = a1T [x1[n]] + a2T [x2[n]], for any arbitrary constants a1 and a2.

Let x[n] be bounded input sequence and h[n] be impulse response of the system and y[n], the output sequence. Necessary and sufficient boundary condition for stability is

8n = -8|h[n]| < 8.

Causal system is a system where the output depends on past or current inputs but not on the future inputs. The necessary condition is h[n] = 0; < 0, where h[n] is the impulse response.

Algorithm:

1. Start
2. Input the coefficients of x(n) and y(n)
3. Generate random signals x1 and x2.
4. Check for linearity using filter function of the given system and display
5. Generate impulse signal.

6. Check for causality using filter function of the given system and display

7. Obtain the absolute value of impulse response and check for the stability of the system and display.

8. Stop

Matlab Functions Used:

• RAND: Uniformly distributed pseudo random numbers. R=RAND (N) returns an N-by-N matrix containing pseudo-random values drawn from a uniform distribution on the unit interval.
RAND (M N)or RAND([M, N]) returns an M-by-N matrix
• DISP: Display array. DISP(X) displays the array, without printing the array name. In all other ways
it‟s the same as leaving the semicolon of an expression except that empty arrays don‟t display.
• ABS: Absolute value. ABS(X) is the absolute value of the elements of X.
• SUM: Sum of elements. S=SUM(X) is the sum of the elements of the vector X.

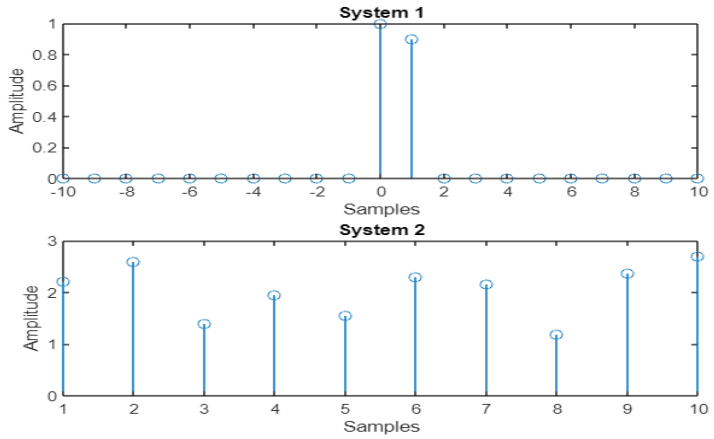
Linear stable causal system

```
%Program to find linearity of system 1
clc
clear all
close all
% System 1
% Linearity
b = [1 0];
a = [1 0.9];
x1= rand(1,10);
x2 = rand (1,10);
x3 = rand (1,10);
```

```
y2 = filter (b, a, 2.*x1);
y5 = filter (b, a,2.* x2);
y = filter (b, a, x1);
y0 = filter (b, a, x2);
y6 = y2+y5;
y7 = 2*y+2*y0;
if (y6-y7 == 0)
disp ('System 1 is Linear ')
else
disp ('System 1 is Non Linear ')
end;
% Causality--------%%%
n1= -10:1:10;
x = [zeros(1,10) 1 zeros(1,10)];
y1 = filter (a, b, x);
subplot (2, 1, 1);
stem (n1,y1);
title('System 1');
xlabel (' Samples ');
ylabel (' Amplitude ');
% Stability
T = abs (y1);
t = sum (T);
if (t < 1000)
disp ('System 1 is Stable ')
else
disp ('System 1 is Unstable ')
end;

%System 2
% 1 % Linearity
y8 = exp (x2);
y9 = exp (x3);
y10 = 5*y8+5*y9;
y11 = exp(5*x2+5*x3);
if (y11-y10==0)
disp ('System 2 is Linear ')
else
disp ('System 2 is Non Linear ')
end;
```

```
% 2 % Causality
Y8= exp (x);
subplot (2, 1, 2);
stem (y8);
title('System 2');
xlabel (' Samples ');
ylabel (' Amplitude ');
% 3 % Stability
T1= abs (y8);
t1 = sum (T1);
if (t1 < 1000)
disp ('System 2 is Stable ')
else
disp ('System 2 is Unstable ')
end;
```



Experiment 4

AIM: To write a MATLAB program to FT of basic signals. Also plot its magnitude and phase spectrum.

Objective:

1. To Generate basic signals Such as sinusoidal, rectangular and sawtooth signals
2. To find their Fourier transform and plot its magnitude and phase spectrum.

Theory:

The generalization of the complex Fourier series is known as the Fourier transform. The term "Fourier transform" can be used in the mathematical function, and it is also used in the representation of the frequency domain. The Fourier transform helps to extend the Fourier series to the non-periodic functions, which helps us to view any functions in terms of the sum of simple sinusoids.

Fourier Transform of a signal x(t) is given by

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$

And Inverse Fourier Transform of X(w) is given by

$$x(t) = \sum_{n=-\infty}^{+\infty} X(\omega)e^{j\omega t}\frac{d\omega}{2\pi} = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega$$

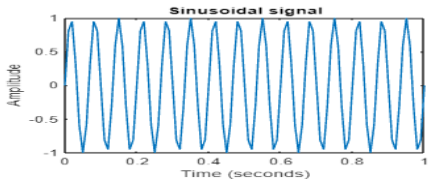
Fourier Transform of Basic Functions

FT of sine signal

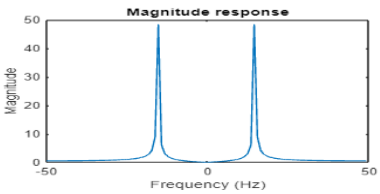
%% plotting of signal

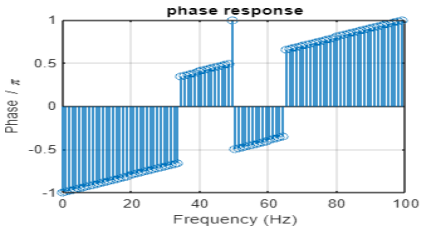
Ts=0.01

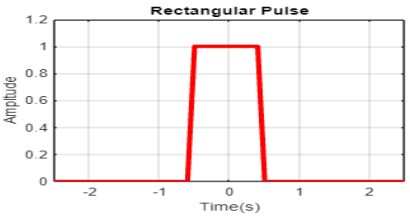
t = 0:Ts:1

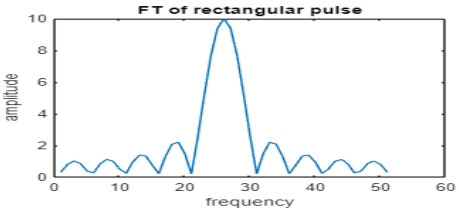


```
x = sin(2*pi*15*t)
subplot(2,2,1)
plot(t,x)
xlabel('Time (seconds)')
ylabel('Amplitude')
title('Sinusoidal signal')
% Fourier Transform of the signal
y = fft(x);
fs = 1/Ts;
f = (0:length(y)-1)*fs/length(y);
n = length(x);
fshift = (-n/2:n/2-1)*(fs/n);
z = fftshift(y);
subplot(2,2,2)
stem(fshift,abs(z))
xlabel('Frequency (Hz)')
ylabel('Magnitude')
title('Magnitude response')
theta = angle(z);
subplot(2,2,3)
stem(f,theta/pi)
xlabel("Frequency (Hz)")
ylabel("Phase /\pi")
title('phase response')
%%FT of rectangular pulse
clear all
close all
clc
%fs = 500;
T = 1;
t = -2.5 : 0.1 : 2.5;
x = rectpuls(t,T);
subplot(2,2,1)
plot(t,x,'r','Linewidth',3);
axis([-2.5 2.5 0 1.2])
```

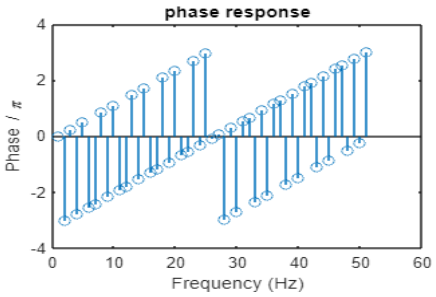


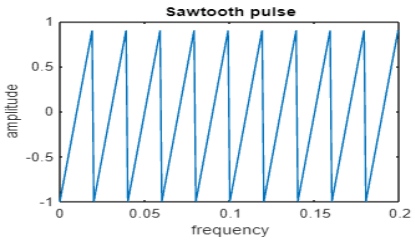


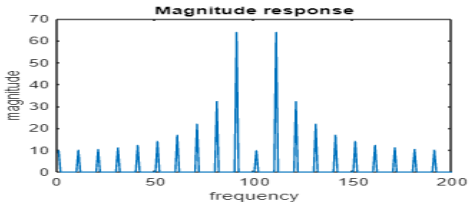


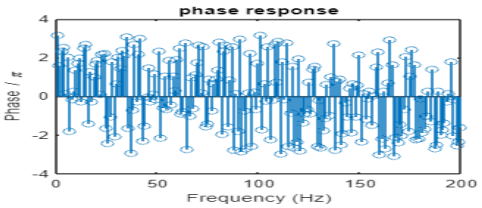


```
title({'Rectangular Pulse'})
xlabel({'Time(s)'});
ylabel('Ampltude');
grid
y=fft(x)
subplot(2,2,2)
plot(fftshift(abs(y)))
xlabel('frequency')
ylabel('amplitude')
title('Magnitude response')
theta = angle(y);
subplot(2,2,3)
stem(theta)
xlabel("Frequency (Hz)")
ylabel("Phase / \pi")
title('phase response')
% FT of sawtooth waveform
T = 10*(1/50);
fs = 1000;
t = 0:1/fs:T-1/fs;
x = sawtooth(2*pi*50*t);
subplot(2,2,1)
plot(t,x)
y=fft(x)
subplot(2,2,2)
plot(fftshift(abs(y)))
xlabel('frequency')
ylabel('amplitude')
title('Magnitude response')
theta = angle(y);
subplot(2,2,3)
stem(theta)
xlabel("Frequency (Hz)")
ylabel("Phase / \pi")
title('phase response')
```









Practice Questions:

1. Generate triangular signal and plot its magnitude and phase response
2. Generate an impulse signal (like siren/Hammer/Buzzer)and plot its magnitude and phase response.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
F	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 5

AIM: To write a MATLAB program for calculating DFT and IDFT discrete time sequences using analytical calculation and inbuilt function.

Objective:

2. To generate basic signals to find its frequency response.
3. To plot its magnitude and phase spectrum.

Theory:

DFT: Discrete Fourier transform is defined for sequences with finite length.

For a sequence x[n] with length N (x[n] for n=0,1, 2, ..., N-1), the discrete-time Fourier transform is

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} = \sum_{n=0}^{N-1} x[n] e^{-j\omega n}$$

X(w) the discrete-time Fourier transform is periodic with period 2π.

The usually considered frequency interval is (-π , π). There are infinitely many points in the interval. If x[n] has N points, we compute N equally spaced ω in the interval (-π , π). That is, we sample using the frequencies.

$$\omega = \omega_k = \frac{2\pi k}{N}, \qquad 0 \le k \le N-1$$

The above equation is known as the N-point DFT Analysis equation.

$$X(k) = X(\frac{k2\pi}{N}) = \sum_{n=0}^{N-1} x[n]e^{-j\frac{k2\pi}{N}n}$$

$$\omega = \frac{k2\pi}{N}, K = 0,1,...N-1$$

Inverse DFT: The DFT values (X(K), $0 \leq k \leq N-1$), uniquely define the sequence x[n] through the inverse DFT formula (IDFT) **$x(n) = IDFT\{X(k)\}$, $0 \leq N \leq n-1$**

The above equation is known as the Synthesis equation.

Matlab Program:

$$x(n) = IDFT\{X(k)\} = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, w_n^{-kn}, \quad 0 \leq N \leq n-1$$

Write a program to find the frequency response of the signal for

(i) n=10 samples

(ii) n=100 (10 samples plus zero padding)

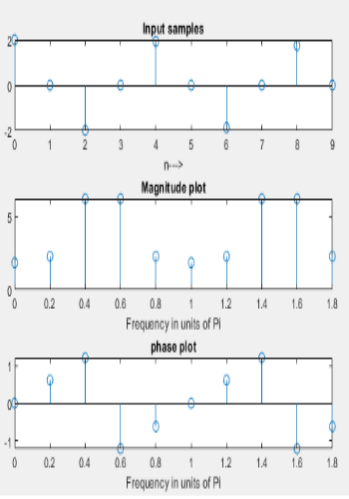
(iii) n=100

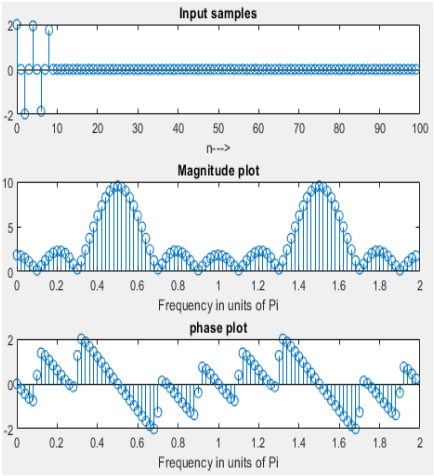
Comment on the frequency spectrum.

DFT and IDFT

Matlab code:

% ***DFT with 10 samples of the signal***
n=[0:1:99];
x=cos(0.48*pi*n)+cos(0.52*pi*n);
% Discrete time signal
% Taking only 10 samples






```
  n1=[0:1:9];
x1=x(1:1:10);
y1=fft(x1);
mag_y1=abs(y1);
phase_y1=angle(y1);
figure(1);
subplot(3,1,1);
stem(n1,x1);
xlabel('n--->');
title('Input samples');
subplot(3,1,2);
F=2*pi*n1/10;
stem(F/pi,mag_y1);
xlabel('Frequency in units of Pi');
title('Magnitude plot');
X1=ifft(y1);
subplot(3,1,3);
stem(F/pi,phase_y1);
```

DFT with 10 samples of the signal

```
xlabel('Frequency in units of Pi');
title('phase plot');
```

% 10 samples + Zero padding

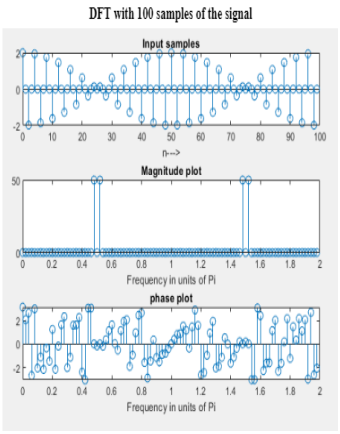
```
n2=[0:1:99];
x2=[x(1:1:10) zeros(1,90)];
y2=fft(x2);
mag_y2=abs(y2);
phase_y2=angle(y2);
figure(2);
subplot(3,1,1);
stem(n2,x2);
xlabel('n--->');
title('Input samples');
subplot(3,1,2);
F=2*pi*n2/100;
stem(F/pi,mag_y2);
```

```
xlabel('Frequency in units of Pi');
title('Magnitude plot');
X2=ifft(y2);
subplot(3,1,3);
stem(F/pi,phase_y2);
xlabel('Frequency in units of Pi');
```

DFT with 10 samples of the signal + padding with zeros

```
title('phase plot');
```

% 100 samples




```
n3=[0:1:99];
x=cos(0.48*pi*n3)+cos(0.52*pi*n3);
x3=x(1:1:100);
y3=fft(x3);
mag_y3=abs(y3);
phase_y3=angle(y3);
figure(3);
subplot(3,1,1);
stem(n3,x3);
xlabel('n--->');
title('Input samples');
subplot(3,1,2);
F=2*pi*n3/100;
%F1=2*pi*[-50:1:49]/100;
stem(F/pi,mag_y3);
xlabel('Frequency in units of Pi');
title('Magnitude plot');
X3=ifft(y3);
subplot(3,1,3);
stem(F/pi,phase_y3);
xlabel('Frequency in units of Pi');
title('phase plot');
```

Practice Questions:

1. Given x(n) = [1,2,3,4] , obtain DFT and IDFT using formula. Plot magnitude and phase plot.
2. Record speech signal and plot FFT.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 6

AIM: To write a Python program for linear and circular convolution of two discrete time sequences. Plot all the sequences and verify the result by analytical calculation.

Objective:

1. To convolve two discrete sequences using linear convolution
2. To convolve two discrete sequences using circular convolution

Theory: Linear Convolution**:** For a system with the impulse response h[n] the output for any arbitrary input x[n] is given by convolution defined as,

$$y[n] = x[n] * h[n]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Circular convolution**:** It is periodic convolution. and are the two finite duration sequences of length N, the circular convolution between though sequence is given by

$$x_3[n] = x_1[n] \otimes x_2[n]$$

$$x_3[n] = \sum_{m=0}^{N-1} x_2[m]x_1[((n-m))_N]$$

The linear convolution of an N-point vector, x, and a L-point vector, y, has length N+L-1.For the circular convolution of x and y to be equivalent, you must pad the vectors with zeros to length at least N+L-1 before you take the DFT. After you invert the product of the DFTs, retain only the first N+L-1 elements.

Python Code

(a) #Program to perform linear convolution

```
import numpy as np
#impulse response
h = [1,2,3,3,2,1]
#input response
x = [1,2,3,4,5]
y = np.convolve(x,h,mode='full')
print('Linear convolution using NumPy built-in function output response y=\n',y)
```

#Results

Linear convolution using NumPy built-in function output response y= [1 4 10 19 30 36 35 26 14 5]

(b) # Python program to compute circular convolution of two arrays

```
import numpy as np
import matplotlib.pyplot as plt
h = [1,2,3,4,5]
#input response
x = [1,2,1]
# Pad sequences to the same length
N=max(len(x), len(h))
x_padded = np.pad(x, (0, N-len(x)), mode='constant')
h_padded= np.pad(h, (0, N-len(h)), mode='constant')
# Perform circular convolution using np.fft.ifft()
X = np.fft.fft(x_padded)
H = np.fft.fft(h_padded)
Y = np.fft.ifft(X * H)
print("Circular Convolution Result:", np.real(Y))
```

 # Result of circular convolution is [15 9 8 12 16]

Practice question:

1. Generate a sinusoidal signal using python and plot the same
2. Generate two sinusoidal signals having 400Hz and 4000Hz using python and plot the convolution of two signals.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 7

AIM: To write a Python program for circular correlation of two discrete time sequences. Plot all the sequences and verify the result by analytical calculation.

Objective:

1. To do the autocorrelation of a discrete sequence
2. To do the cross correlation of two discrete sequences

Theory

The correlation operation measures the similarity between two sequences by computing the convolution of one sequence with a time-reversed version of the other sequence. There are two types of correlation: cross-correlation and auto-correlation.

Cross-correlation measures the similarity between two different sequences. Given two sequences, let's say x[n] and y[n], the cross-correlation is computed as:

$r\_xy[k] = \sum(x[n] * y[n-k])$

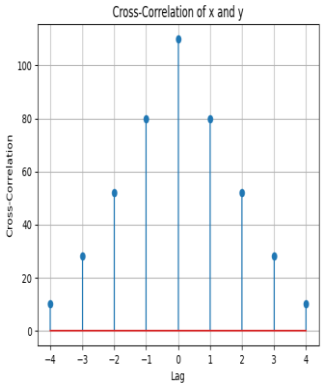
where k is the lag or shift applied to the second sequence y[n]. The resulting sequence r_xy[k] represents the similarity between the two sequences at each lag k.

Auto-correlation measures the similarity of a sequence with itself. Given a sequence x[n], the auto-correlation is computed as: $r\_xx[k] = \sum(x[n] * x[n-k])$

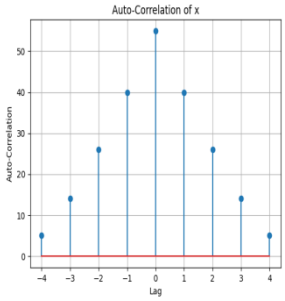
Similar to cross-correlation, the resulting sequence r_xx[k] represents the similarity of the sequence x[n] at each lag k.

Python Code

(a) #Program to perform auto correlation and cross correlation




```
import numpy as np
import matplotlib.pyplot as plt
# First sequence
x = [1, 2, 3, 4, 5]
# Second sequence
y = [2, 4, 6, 8, 10]
# Perform cross-correlation
cross_corr = np.correlate(x, y, mode='full')
print('cross correlation',cross_corr)
Perform auto-correlation
auto_corr = np.correlate(x, x, mode='full')
print('autocorrelation', auto_corr)
# Plotting the results
lags = np.arange(-len(x) + 1, len(x))
plt.stem(lags, cross_corr)
plt.xlabel('Lag')
plt.ylabel('Cross-Correlation')
plt.title('Cross-Correlation of x and y')
plt.grid(True)
plt.show()
plt.stem(lags, auto_corr)
plt.xlabel('Lag')
plt.ylabel('Auto-Correlation')
plt.title('Auto-Correlation of x')
plt.grid(True)
plt.show()
```



#RESULT

cross correlation [10 28 52 80 110 80 52 28 10]

auto correlation [5 14 26 40 55 40 26 14 5]

Linear convolution, circular convolution and correlation) without using inbuilt functions

```
import numpy as np
import matplotlib.pyplot as plt
#Defining functions
def linear_convolution_nf(x,y):
    x_len=len(x)
    y_len=len(y)
    result=[]
    for n in range(x_len+y_len-1):
        res_sum=0
        for k in range(x_len):
            if n-k<y_len and n-k>=0:
                res_sum+=x[k]*y[n-k]
        result.append(res_sum)
    return result

def circular_convolution_nf(x,y):
    if(len(x)!=len(y)):
        Max=max(len(x),len(y))
        x=np.pad(x,(0,Max-len(x)),mode='constant')
        y=np.pad(y,(0,Max-len(y)),mode='constant')
    N=len(x)
    result=[]
    for n in range(N):
        res_sum=0
        for k in range(N):
                res_sum+=x[k]*y[(n-k)%N]
        result.append(res_sum)
    return result

def circular_correlation_nf(x,y):
    N=len(x)
    res=[]
    for n in range(-(N-1),N):
        res_sum=0
        for k in range(N):
                if k-n>=0 and k-n<N:
```

```
                res_sum+=x[k] * y[(k-n)]
        res.append(res_sum)
    return res

#Inputs
x1=[1,2,3,4]
x2=[2,3,4,5]

#Printing Outputs
print("Linear convolution of x1 and x2
is:",linear_convolution_nf(x1,x2))
print("Circular convolution of x1 and x2
is:",circular_convolution_nf(x1,x2))
print("Circular correlation of x1 and x2
is:",circular_correlation_nf(x1,x2))
```

Practice question:

3. Generate two sinusoidal signals using python and plot the autocorrelation and cross correlation of the same.
4. Generate signals having two frequencies 400 Hz and 4000 Hz using python and plot the autocorrelation of the signals.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 8

AIM: Write a python code to extract features in the time domain for any signal.

Objective:

Extract time domain features from any audio signal (single tone/multitone)

Theory:

Feature extraction is the process of highlighting the most discriminating and impactful features of a signal. The evolution of features used in audio signal processing algorithms begins with features extracted in the time domain, which continue to play an important role in audio analysis and classification. Some of the basic features discussed here are:
i. read the audio file, ii. Measure the duration of the audio signal , iii. Amplitude variation of signal , iv.Spectrogram representation of Audio signal,v.MFCC, vi Chroma

i. #read audio file from google drive

```
import librosa   #librosa is a Python package for music and audio processing
import librosa.display   #
import IPython.display as ipd    #to play the audio signal
import matplotlib.pyplot as plt     #plot the audio signal in time domain
ipd.Audio('/content/drive/MyDrive/ColabNotebooks/trainingdatasets/cars001.wav)
#load a local WAV file
```

```
audio_path=('/content/drive/MyDrive/ColabNotebooks/trai
ningdatasets/cars001.wav')
x , sr = librosa.load(audio_path)   # 22050Hz
```

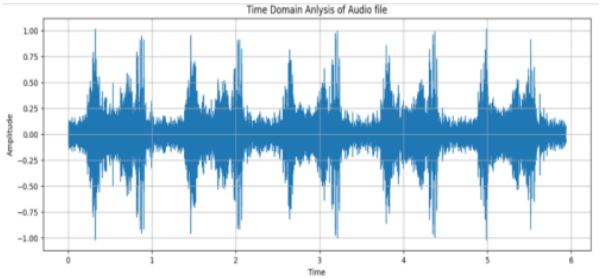
ii. #measure the duration of the audio signal

```
ipd.Audio('/content/drive/MyDrive/ColabNotebooks/traini
ngdatasets/cars001.wav')
#Duration of audio file
y=len(x)  #number of samples x
print(y)
#print(len(x))
y1=sr   #sampling frequency sr
print(y1)
duration_of_sound=y/y1    #number of samples/sampling
frequency
print(duration_of_sound,"second")
```

Output: 5.9346938775510205 second

iii. Amplitude variation of signal in time domain

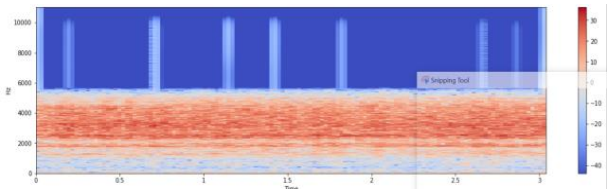
```
plt.figure(figsize=(14, 5))
#variable to change the x and y axix range
plt.grid(True )
#plotting the sampled signal
librosa.display.waveshow(x)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Time Domain Anlysis of Audio file")
```



iv**. Spectrogram representation of Audio signal :**

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time.

```
X = librosa.stft(x)
#converting into energy levels(dB)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(20, 5))
librosa.display.specshow(Xdb, sr=sr,
x_axis='time', y_axis='hz')
plt.colorbar()
```

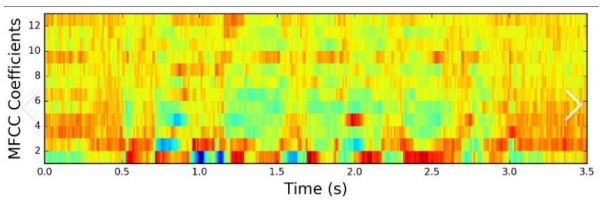


v. Mel Frequency Cepstral Coefficients (MFCC)

MFCC component : The **mfcc** function processes the entire speech data in a batch. Based on the number of input rows, the window length, and the overlap length, **mfcc** partitions the speech into 1551 frames and computes the cepstral features for each frame

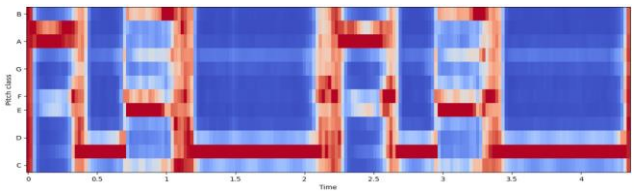
```
librosa.load('/content/drive/MyDrive/ColabNotebooks/traini
ng datasets/cars001.wav')
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
 #varable should be  y not x
plt.figure(figsize=(10,4))
librosa.display.specshow(mfccs, x_axis="time")
plt.colorbar()
```

```
plt.title('MFCC')
plt.tight_layout()
plt.show()
```



vi. **Chroma: We can use Chroma feature visualization to know how dominant the characteristics of a certain pitch {C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B} is present in the sampled frame.**

```
x, sr = librosa.load('/content/drive/MyDrive/Colab Notebooks/training datasets/cars014.wav')
hop_length = 512
S = np.abs(librosa.stft(y))
chroma = librosa.feature.chroma_stft(S=S, sr=sr)
plt.figure(figsize=(15, 5))
librosa.display.specshow(chroma, x_axis='time', y_axis='chroma', hop_length=hop_length, cmap='coolwarm')
```



Practice question:

1. Time domain analysis of various single tone signals and note the observations**.**

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 9

AIM: To write a python code to extract features in frequency domain for any signal.

Objective:

Extract frequency domain feature from any audio signal (single tone/multitone)

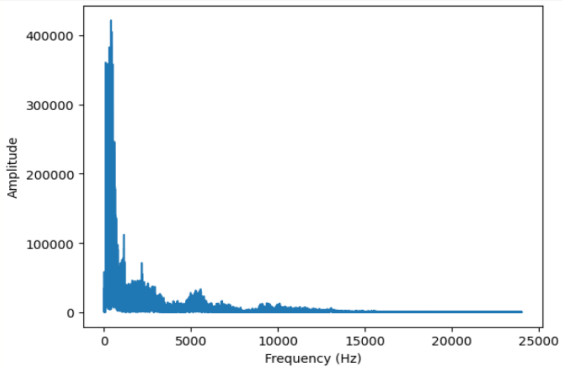
Theory:

The frequency domain-based audio feature extraction reveals the frequency content of the audio signals, band energy ratio, and so on. Some of the features analysed for the audio signals are:
i. measure the frequency of the signal using FFT, ii. mean, iii.standard deviation, vi skewness and kurtosis.

i. Read the audio file of ur choice

```
import scipy.io.wavfile as wavfile
import scipy
import scipy.fftpack as fftpk
import numpy as np
from matplotlib import pyplot as plt
s_rate, signal = wavfile.read('/content/drive/MyDrive/Colab Notebooks/training datasets/Copy of Sad(10).wav')
FFT = abs(fftpk.fft(signal))
freqs = fftpk.fftfreq(len(FFT), (1.0/s_rate))
plt.plot(freqs[range(len(FFT)//2)], FFT[range(len(FFT)//2)])
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
```

```
plt.show()
```



ii. #Measure of dispersion – variance – the second moment:

```
std = np.std(freqs)
maxv = np.amax(freqs)
minv = np.amin(freqs)
median = np.median(freqs)
```

iii. #Higher order moments – skewness and kurtosis:

```
skew = scipy.stats.skew(freqs)
kurt = scipy.stats.kurtosis(freqs)  # identify anomalies and outliers in many signal processing applications.
q1 = np.quantile(freqs, 0.25)
q3 = np.quantile(freqs, 0.75)
mode = scipy.stats.mode(freqs)[0][0]
iqr = scipy.stats.iqr(freqs)
print('mean as:', mean)
print('std as:',std)
print('shew as:',skew)
print('kurt as:',kurt)
print('q1 as:',q1)
```

Output:mean as: -5.206706237642656e-06

std as: 0.28867513457916105

shew as: 1.9759477879042912e-16

kurt as: -1.2000000002602538 q1 as: -0.25000260335311886

Practice questions:

1. Measure the frequency of various real time audio signals, compare the frequencies.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 10

AIM: To Develop a Simulink model to demonstrate Amplitude modulation and Demodulation.

Objective:

1. Designing Amplitude modulator and demodulator using Simulink
2. Understanding the waveforms of modulated and demodulated signals.
3. Observing the effects of the percent of modulation

Theory:

Modulation is a process by which some characteristic of a carrier is varied in accordance with a modulating wave. The message signal is referred to as the modulating wave and the result of the modulation process is referred to as the modulated wave.

Amplitude Modulation:

Consider a sinusoidal carrier wave c(t) defined by

$$c(t)=A_c \cos(2\pi f_c t)$$

Where is the carrier amplitude and fc is the carrier frequency.

For convenience phase of

the carrier wave is assumed to be zero.

Let m(t) denote the baseband signal that carries specification of the message.

Amplitude Modulation is defined as a process in which amplitude of the carrier wave c(t) is

varied linearly with the message signal m(t).

A standard form of an amplitude-modulated wave is defined as

$$S(t) = A_c\ [1 + K_a\ m(t)]\ Cos(2\pi\ fc\ t)$$

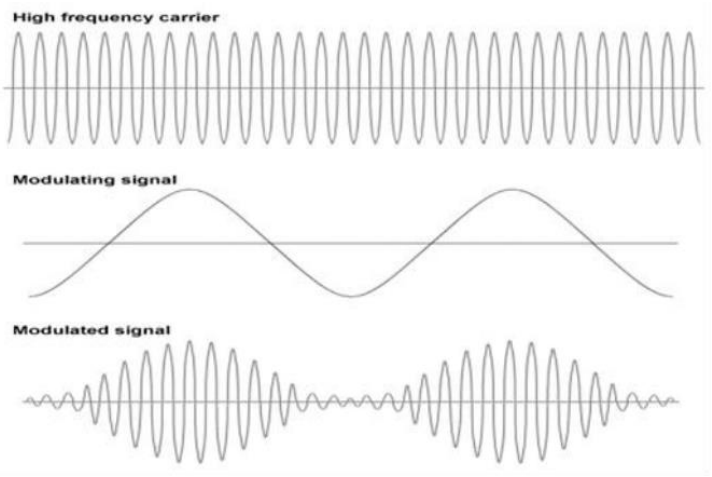
where is called the amplitude sensitivity of the modulator. The envelope of the modulated signal is given by,

$$a(t) = A_c\ |1 + K_a\ \ m(t)|$$

Percentage Modulation:

If the maximum absolute value of $K_a\ m(t)$ is multiplied by 100 then it is referred to as percentage modulation. If $K_a\ m(t) < 1,$ then the percentage modulation in is ≤ 100%, where as

$K_a\ m(t)$ > 1, then the percentage modulation is in excess of 100%. In first case will have one to one correspondence where as in the second case, modulated wave is said to suffer from envelope distortion and the wave is said to be over-modulated.



Conventional AM in Simulink

Procedure:

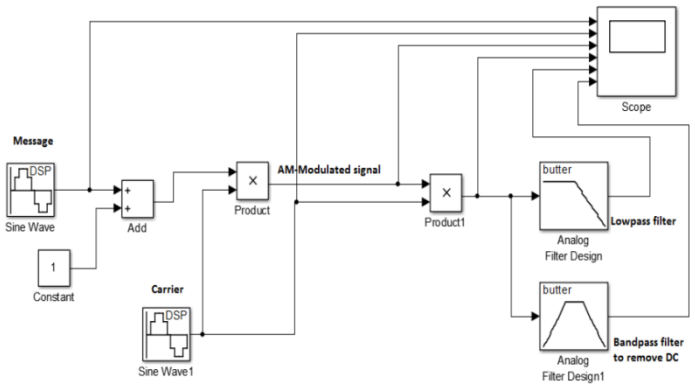
- Type Simulink in the Matlab Command Window to open a Simulink Library Browser
- Go to Signal Processing Blockset -> Signal Processing Sources
- Choose Sine wave as source. Right click on it and select add to untitled.

File -> save as -> give file name and save it with .mdl extension For ex: Std_AM.mdl

In this experiment Simulink is used to generate a Conventional Amplitude Modulator

(Transmitter and receiver). Construct the model as shown in the figure below by searching

each block in a Simulink Library Browser.



Input Parameters:

(a) Sine wave

Amplitude: 1 V

Frequency: 4 Hz

Phase offset: 0 radians

Sample mode: Discrete

Output complexity: Real
Computation method: Trigonometric fcn
Sample time: 1/500
Samples per frame: 1

(b) Constant
Constant value: 1

(c) Sine wave1
Amplitude: 1 V
Frequency: 50 Hz
Phase offset: 0 radians
Sample mode: Discrete
Output complexity: Real
Computation method: Trigonometric fcn
Sample time: 1/5000
Samples per frame: 1

(d) Add
List of signs: ++

(e) Product and Product1
Number of inputs: 2

(f) Analog Filter Design
Design method: Butterworth
Filter type: Lowpass
Filter order: 20
Passband edge frequency (rad/s): 2*3.142*8

(g) Analog Filter Design1
Design method: Butterworth
Filter type: Bandpass
Filter order: 20
Lower passband edge frequency (rad/s): 2*3.142*2

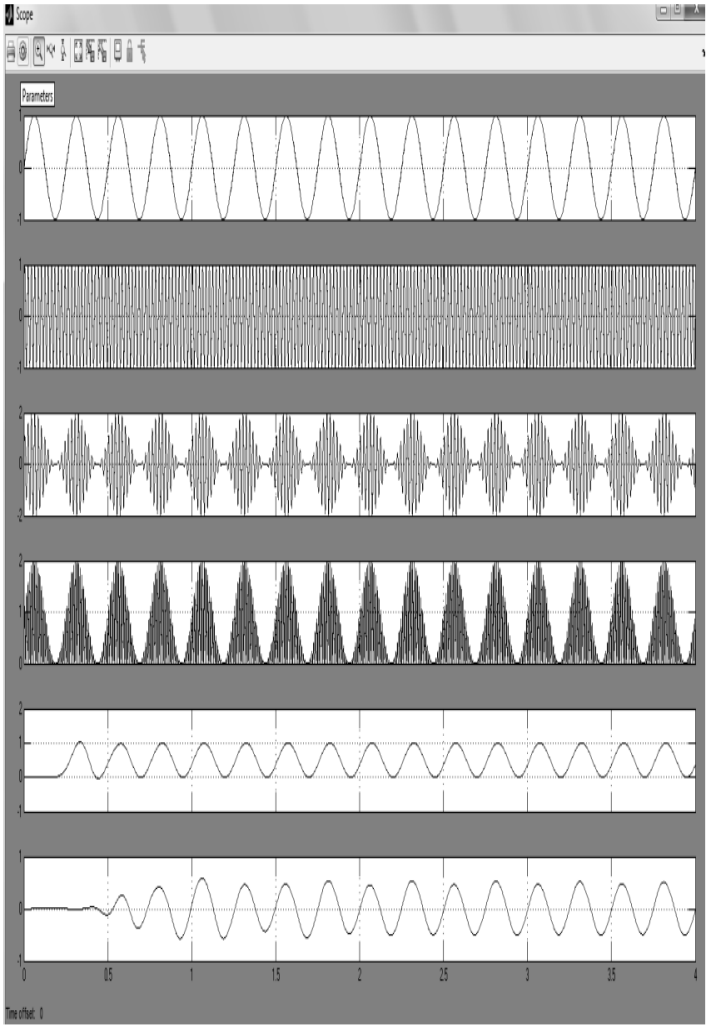
Upper passband edge frequency (rad/s): 2*3.142*8

(h) Scope

Go to Settings, then click on History and Uncheck the limit data points to last

Run time : 4sec

Waveforms:



Practice Questions:

1. AM modulation and Demodulation using MATLAB also perform frequency domain analysis
2. To Develop a Simulink model to demonstrate any variant Amplitude modulation and Demodulation.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

OPEN ENDED EXPERIMENTS

- To demonstrate classify two signals of various features using STM 32.
- To demonstrate real time applications using microcontroller.