**EXP NO: 1**

# GENERATION OF SIGNALS

**AIM**

Program to generate the following signals using MATLAB.

1. Unit impulse signal
2. Unit pulse signal
3. Unit ramp signal
4. Bipolar pulse
5. Triangular signal

# TOOLS REQUIRED:-

MATLAB

# PROGRAM

%% Impulse signal clc;

clear all;

close all; p=[0]; q=[1];

subplot(5,1,1);

stem(p,q);

xlabel('n');

ylabel('x(n)'); title('impulse'); t=1:0.1:100;

%% Bipolar SQUARE WAVE a=0.7\*square(t); subplot(5,1,2);

plot(t,a);

title('square');

xlabel('time'); ylabel('amplitude');

%%RAMP WAVE t=-10:10;

b=(t>=0).\*t;

subplot(5,1,3);

plot(t,b);

title('ramp');

xlabel('time'); ylabel('amplitude');

% PULSE WAVE

fs = 100E9; % sample freq

D = [2.5 10 17.5]' \* 1e-9; % pulse delay times t = 0 : 1/fs : 2500/fs; % signal evaluation time w = 1e-9; % width of each pulse(in nano seconds) yp = 1\*pulstran(t,D,@rectpuls,w);

subplot(5,1,4); plot(t\*1e9,yp); axis([0 25 -0.2 2]);

xlabel('Time (ns)'); ylabel('Amplitude');

%%The first pulse occur at 2.5ns with a width 1ns

% Sawtooth wave T = 10\*(1/50);

fs = 1000;

t = 0:1/fs:T-1/fs;

x = sawtooth(2\*pi\*50\*t,1/2); subplot(5,1,5);

plot(t,x); title('triangular'); xlabel('time'); ylabel('amplitude'); grid on;

**RESULT**

Basic continuous signals Unit impulse signal, Unit pulse signal, Unit ramp signal, Bipolar pulse, Triangular signal were generated.

# PROGRAM

**DFT**

#GENERATE AND APPRECIATE DFT MATRIX

import numpy as np from scipy import fft

import matplotlib.pyplot as plt import time

import random import math

N=int(input('how many point dft:'))

#program for direct computation of DFT start1=time.time()

V\_N=np.empty((N, N), dtype=np.cdouble); W=np.exp(-1j\*2\*np.pi/N)

k= np.arange(N)

for n in np.arange(N):

V\_N[:, n]= W\*\*(k\*n) np.round(V\_N)

xn = random.sample(range(0, 1500), N)

X=V\_N@xn; # @ is the matrix multiplication operator np.round(X)

end1=time.time()

#program for calculation of DFT using FFT function start2=time.time()

y=fft.fft(xn, axis=0) end2=time.time() print("Input sequence=",xn) print("Direct DFT of xn=",X) print("FFT of xn=",y) t1=end1 - start1

print("Runtime of the direct computation=",t1) t2=end2 - start2

print(f"Runtime of the fft computation=",t2) eff=100-((t2/t1)\*100)

print("Computational saving of FFT as compared to direct DFT=",eff, "%")

#to plot real and imaginary parts of V\_N plt.subplot(1, 3, 1) plt.title('$\mathrm{Re}(\mathrm{DFT}\_N)$')

plt.imshow(V\_N.real) plt.xlabel('Time (sample, index $n$)') plt.ylabel('Frequency (index $k$)') plt.subplot(1, 3, 2)

plt.title('$\mathrm{Im}(\mathrm{DFT}\_N)$') plt.imshow(V\_N.imag)

plt.xlabel('Time (samples, index $n$)') plt.ylabel('Frequency (index $k$)')

#to find value of gamma(no. of stages) gamma=math. log2(N) print("\u03B3=",gamma)

# CIRCULAR CONVOLUTION

#CIRCULAR CONVOLUTION

import numpy as np from scipy import signal g=np.array([1, 5, 0])

h=np.array([2, 3, 6, 7, 9, 10]) def circonv(g, h):

N1=g.size N2=h.size N=max(N1,N2)

y=np.zeros(N) if N1>N2:

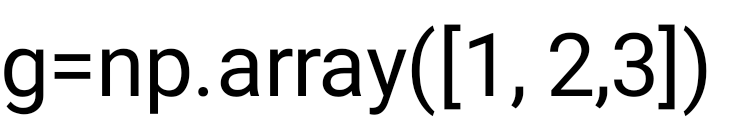
h=np.append(h,np.zeros(N1-N2)) elif N2>N1: g=np.append(g,np.zeros(N2-N1))

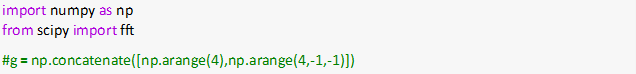
htr=np.concatenate([[h[0]], h[:0:-1]])#circular time-reversal for n in np.arange(N):

y[n] =np.sum(g\*htr) htr=np.roll(htr,1)#circular shift by 1 unit return y

print(circonv(g,h))

**PARSEVAL'S THEOROM**

# Verify Parseval’s relation for a sequence g[n]



print(g)

LHS = np.sum(g\*\*2) G = fft.fft(g)

RHS = 1/G.size \* np.sum(np.abs(G)\*\*2) print(LHS, RHS)

**SWITCH LED**

## Program:

**#include** "types.h"

**#include** "evmc6748.h"

**#include** "evmc6748\_gpio.h" **#include** "vi6748.h"

## int main(void)

{

uint8\_t \*XinSeq,i; XinSeq=(uint8\_t\*)0x80010000;

**EVMC6748\_lpscTransition**(PSC1, DOMAIN0, LPSC\_GPIO, PSC\_ENABLE); EVMC6748\_pinmuxConfig(PINMUX\_MCASP\_REG\_18, PINMUX\_MCASP\_MASK\_18,

PINMUX\_MCASP\_VAL\_18);

EVMC6748\_pinmuxConfig(PINMUX\_MCASP\_REG\_19, PINMUX\_MCASP\_MASK\_19, PINMUX\_MCASP\_VAL\_19);

EVMC6748\_pinmuxConfig(PINMUX\_MCASP\_REG\_1, PINMUX\_MCASP\_MASK\_1, PINMUX\_MCASP\_VAL\_1);

**for**(i=8;i<=15;i++)

{

VSK\_GPIO\_setDir(8, i, GPIO\_OUTPUT); VSK\_GPIO\_setDir(0, (i-8), GPIO\_INPUT);

}

**while**(1) {

**for**(i=8;i<=15;i++)

{

**GPIO\_getInput**(0,(i-8), XinSeq);

**GPIO\_setOutput**(8, i, OUTPUT\_HIGH );

}

}

}

**Department Of Electronics & Communication Engineering**

## Linear Convolution Program:

**#include**<math.h> **#include**<stdio.h>

## void main()

{

**int** \*Xn,\*Hn,\*Output;

**int** \*XnLength,\*HnLength;

**int** i,k,n,l,m;

Xn=(**int** \*)0x80010000; //input x(n) Hn=(**int** \*)0x80011000; //input h(n) XnLength=(**int** \*)0x80012000; //x(n) length HnLength=(**int** \*)0x80012004; //h(n) length Output=(**int** \*)0x80013000; // output address

l=\*XnLength; // copy x(n) from memory address to variable l m=\*HnLength; // copy h(n) from memory address to variable m

**for**(i=0;i<(l+m-1);i++) // memory clear

{

Output[i]=0; // o/p array Xn[l+i]=0; // i/p array Hn[m+i]=0; // i/p array

}

**for**(n=0;n<(l+m-1);n++)

{

**for**(k=0;k<=n;k++)

{

Output[n] =Output[n] + (Xn[k]\*Hn[n-k]); // convolution operation.

}

}

}

Freq, Nyquist Frequency, order

1. Choose the Window Type - hamming
2. Approximate the Window Length
3. Find the Appropriate Ideal Filter
4. Apply timeshift and multiply with window
5. Plot the magnitude response and phase response of the filter.

**Output:** The following waveform is obtained:

**Result:** Implemented magnitude and phase response of FIR Low Pass Filter using Hamming Window Method

## FIR LPF Program:

import numpy as np

import matplotlib.pyplot as plt from scipy import signal N=50

w=np.hamming(N)

i= np.arange(-(N-1)/2,(N-1)/2+1) wc=0.1\*np.pi hd=wc/np.pi\*np.sinc(wc/np.pi\*i) h=hd\*w plt.figure(figsize=(15,5)) plt.subplot(121)

plt.stem(h , linefmt = "Green" , markerfmt = 'D') plt.title('Impulse Response')

w, H=signal.freqz(h,1); plt.subplot(122)

plt.plot(w/np.pi , abs(H) , label = "Magnitude Response" , color = 'Magenta' , linewidth = 1.5 ) plt.title('Magnitude Response');

**Overlap add Program: #include** <stdio.h>

**#include** <math.h>

**#define** size(x) **sizeof**(x)/**sizeof**(\*x)

**#define** PI 3.141592653589 //Pi, 12 decimal places

**#define** NS 4 //Fourier transform points

**#define** MS 2 //The number of butterfly operations, N = 2^M

//#define N 64 //Fourier transform points

//#define M 6 //The number of butterfly operations, N = 2^M

**int** xsample[] = {3,-1,0,1,3,2,0,1,2,1}; // input sample Xn

**int** hsample[] = {1, 1, 1}; // input impulse response Hn

//yn = {3,2,2,0,4,6,5,,3,3,4,3,1}; // output sample

**typedef double** ElemType; //The data type of the original data sequence can be set here

**typedef struct** //Define complex structure

{

ElemType real,imag;

}complex;

complex data[NS],xndata[NS],hndata[NS]; //Define the storage unit, the original data and

//negative results are used

ElemType result[NS]; //Store the modulus of the complex number result after FFT

// Allocates a 2D array that can be accessed in the form arr[r][c].

// The caller is responsible for calling free() when done.

**void**\*\* **malloc2d**(size\_t rows, size\_t cols, size\_t element\_size)

{ size\_t header = rows \* **sizeof**(**void**\*); size\_t body = rows \* cols \* element\_size; size\_t needed = header + body;

**void**\*\* mem = malloc(needed);

**if** (!mem) {

**return** NULL;

}

size\_t i;

**for** ( i = 0; i < rows; i++) {

**void**\* col\_mem = mem + header + i\*rows\*cols\*element\_size; mem[i] = col\_mem;

}

**return** mem;

}

**void** \* **my\_malloc**(size\_t s)

{

size\_t \* ret = malloc(**sizeof**(size\_t) + s);

\*ret = s;

**return** &ret[1];

}

**void** \* **my\_realloc**(**void** \*ptr,size\_t s)

{

size\_t \* ret = realloc(ptr,**sizeof**(size\_t) + s);

\*ret = s;

**return** &ret[1];

}

**void my\_free**(**void** \* ptr){ free( (size\_t\*)ptr - 1);}

size\_t **allocated\_size**(**void** \* ptr){ **return** ((size\_t\*)ptr)[-1]/**sizeof**(ptr);} **int stagecnt**(**int** X,**int** L){

// Computes quotient

**int** quo = X / L;

// Computes remainder

**int** rem = X % L; **int** temp; **if**(rem == 0) temp = quo;

## else

temp = quo + 1;

**return** temp;}

// Find maximum between two numbers.

**int max**(**int** num1, **int** num2){

**return** (num1 > num2 ) ? num1 : num2;}

// Find minimum between two numbers.

**int min**(**int** num1, **int** num2){

**return** (num1 > num2 ) ? num2 : num1;}

//Index

**void ChangeSeat**(complex \*DataInput)

{

**int** nextValue,nextM,i,k,j=0; complex temp;

nextValue=NS/2; //Indexing operation, that is, changing the natural order

**else** { yn[i]=y\_n[h][zp];} n\_d++; h=n\_d/L;

zp++;

**if**(zp >= N) zp=L;

}

}

**printf**("OVERLAP ADD FFT\_IFFT:");

**for**(i=0; i<(X+L); i++){**printf**("%f ",yn[i]);} free(y\_n); free(yn);

**return** 0;

}

}

}

# MATLAB PROGRAM

close All clear All clc

N=input('Enter the length of x(n) : '); x=rand(1,N); % Random N Numbers h=input('Enter the values of h(n)='); L=length(h);

N1=length(x); M=length(h); lc=conv(x,h);

x=[x zeros(1,mod(-N1,L))]; N2=length(x);

h=[h zeros(1,L-1)];

H=fft(h,L+M-1); S=N2/L;

index=1:L; X=[zeros(M-1)]; for stage=1:S

xm=[x(index) zeros(1,M-1)]; % Selecting sequence to process X1=fft(xm,L+M-1);

Y=X1.\*H;

Y=ifft(Y);

Z=X((length(X)-M+2):length(X))+Y(1:M-1); %Samples Added in every stage X=[X(1:(stage-1)\*L) Z Y(M:M+L-1)];

index=stage\*L+1:(stage+1)\*L;

end i=1:N1+M-1; X=X(i);

figure() subplot(2,1,1) stem(lc);

title('Convolution Using inbuilt function') xlabel('n');

ylabel('y(n)'); subplot(2,1,2) stem(X);

title('Convolution Using Overlap Add Method') xlabel('n');

ylabel('y(n)');

**float** \*yn = (**float**\*)malloc((xc) \* **sizeof**(**float**)); h=0;**int** n\_d = 0;

**int** zp=L;

**for**(i=0; i<(xc); i++){

**if**(i<L)

yn[i]=0;

## else

{

yn[i]=y\_n[h][zp];

zp++;

**if**(zp >= N) zp=L; h++;

n\_d++; h=n\_d/L;

}

}

**printf**("\n\n");

**printf**("OVERLAP SAVE FFT\_IFFT:");

**for**(i=0; i<(xc); i++){**printf**("%f ",yn[i]);} free(y\_n); free(yn);

**return** 0;

}

# MATLAB PROGRAM

close All close All clear All clc

N=input('Enter the length of x(n) : '); x=rand(1,N); % Random N Numbers h=input('Enter the values of h(n)='); L=length(h);

N1=length(x); M=length(h); lc=conv(x,h);

x=[x zeros(1,mod(-N1,L)) zeros(1,L)]; N2=length(x);

h=[h zeros(1,L-1)];

H=fft(h,L+M-1); S=N2/L;

index=1:L;

xm=x(index); % For first stage Special Case x1=[zeros(1,M-1) xm]; %zeros appeded at Start point X=[];

for stage=1:S X1=fft(x1,L+M-1); Y=X1.\*H;

Y=ifft(Y); index2=M:M+L-1;

Y=Y(index2); %Discarding Samples X=[X Y];

index3=(((stage)\*L)-M+2):((stage+1)\*L); % Selecting Sequence to process if(index3(L+M-1)<=N2)

x1=x(index3); end

end; i=1:N1+M-1; X=X(i);

figure() subplot(2,1,1) stem(lc);

title('Convolution Using inbuilt function') xlabel('n');

ylabel('y(n)'); subplot(2,1,2) stem(X);

title('Convolution Using Overlap Save Method') xlabel('n');

ylabel('y(n)');