LABORATORY MANUAL

DIGITAL SIGNAL PROCESSING LAB (S5 T)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING TRIVANDRUM

THIRUVANANTHAPURUM

2019

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING TRIVANDRUM

THIRUVANANTHAPURAM

This is a controlled document of the department of Electronics and Communication Engineering, College of Engineering Trivandrum, Thiruvananthapuram. No part of this can be reproduced in any form by any means without the prior written permission of the Head of the Department, Electronics and Communication Engineering, College of Engineering Trivandrum, Thiruvananthapuram. This is prepared as per 2016 KTU B.Tech scheme.

Head of the Dept.

Dept. of ECE

CET

INDEX

Part A

EXPERIMENT NO 01

GENERATION OF WAVEFORMS

AIM

To generate continuous and discrete waveforms.

THEORY

In signal processing experiments, we need the help of some fundamental signals such as sine wave, square wave, ramp wave, unit step, unit impulse etc.

A digital signal can be either a deterministic signal that can be predicted with certainity, or a random signal that is unpredictable. Due to ease in signal generation and need for predictability, deterministic signal can be used for system simulation studies. Standard forms of some deterministic signals that are frequently used in DSP are discussed below:

1. **Impulse:** The simplest signal is the unit impulse signal which is defined as,

   $\delta(n) = 1$ for $n = 0$

   $= 0$ for $n \neq 0$

2. **Step:** The general form of step function is,

   $u(n) = 1$ for $n \geq 0$

   $= 0$ for $n < 0$

3. **Exponential**: The decaying exponential is a basic signal in DSP whose general form is,

   $x(n) = a^n$ for all n.

4. **Ramp:** This signal is given by,

   $r(n) = n$ for $n \geq 0$

   $= 0$ for $n < 0$

5. **Sine:** A sinusoidal sequence is defined as,

   $x(n) = \sin(n)$.

A continuous time signal is defined for all values of time *t*. Standard forms of some basic continuous time signals frequently used in DSP are discussed below:

1. **Impulse:** An impulse signal is given by,

   $$\int_{-\infty}^{\infty} \delta(t) = 1$$

   $$= 0 \text{ for } t \neq 0$$

2. **Sine:** A sinusoidal sequence is defined as,

   $x(t) = \sin(t)$

3. **Unit step signal:** The general form of step signal is,

   $$u(t) = A \text{ for } t \geq 0$$

   $$= 0 \text{ for } t < 0$$

   Where, A is the amplitude. If A=1, then it is a unit step signal.

4. **Ramp:** Ramp signal is given by,

   $$r(t) = t \text{ for } t \geq 0$$

   $$= 0 \text{ for } t < 0$$

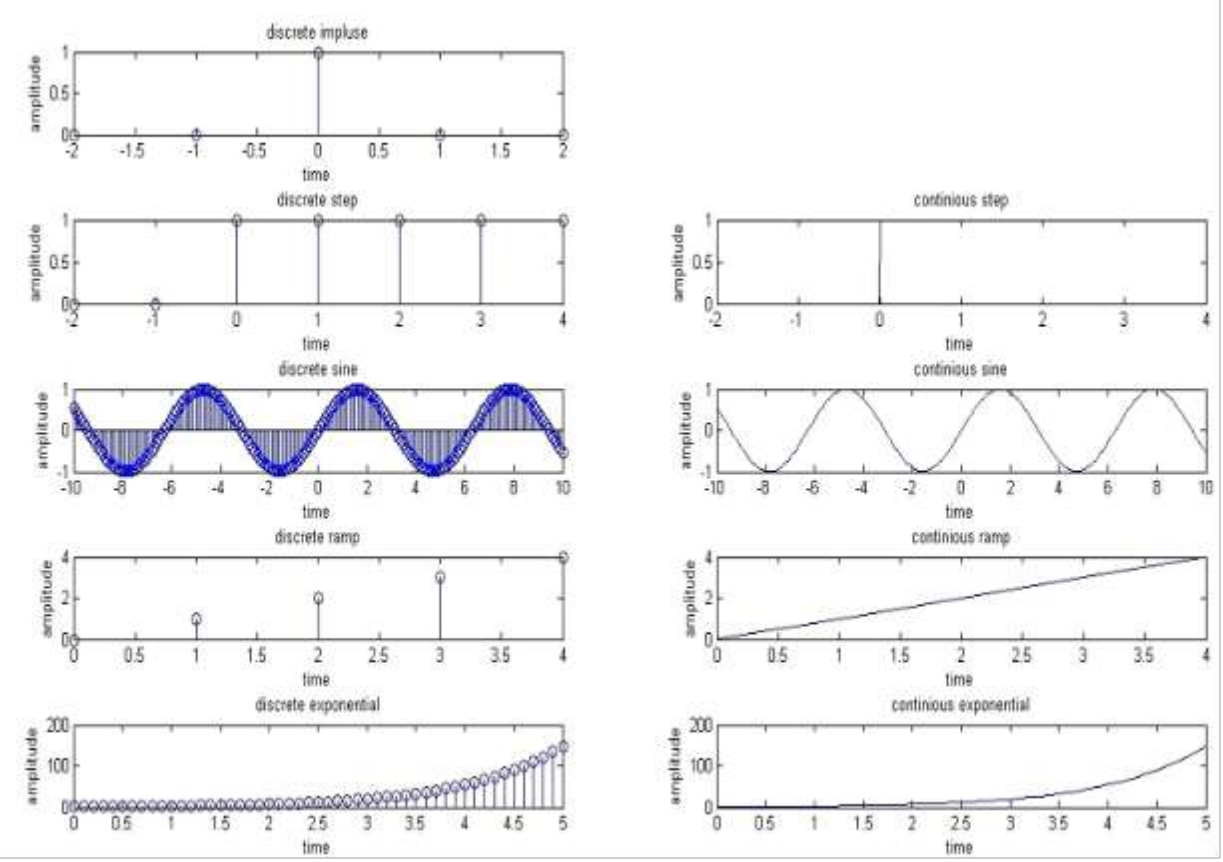
5. **Exponential :** Exponential signal is given by

   $x(t)=e^{at}$, a>0 for growing exponential, a<0 for decaying exponential

INSTRUCTIONS

1. Plot discrete impulse function of duration 5, symmetrically distributed on time axis
2. Plot discrete step function of duration 10, spread over positive and negative x axis
3. Plot continuous step function of duration 10, spread over positive and negative x axis
4. Plot discrete sine function, of 5 cycles, spread over positive and negative x axis
5. Plot continuous sine function of 5 cycles, spread over positive and negative x axis
6. Plot discrete and continuous ramp functions of duration 10, spread over positive and negative axis.
7. Plot discrete and continuous exponential functions of duration 50. Choose weight factor as your lab team number.
8. Parameters for simulation is selected based on the batch number (N)

SAMPLES OF EXPECTED OUTPUT



RESULT

The MATLAB program to generate continuous and discrete waveforms were executed and obtained output waveforms.

EXPERIMENT NO 02

VERIFICATION OF SAMPLING THEOREM.

AIM

Write a MATLAB program to verify the sampling theorem.

THEORY

Sampling Theorem: A band limited signal can be reconstructed exactly if it is sampled at a rate atleast twice the maximum frequency component in it. Figure 1 shows the spectrum of a signal g(t) that is band limited.

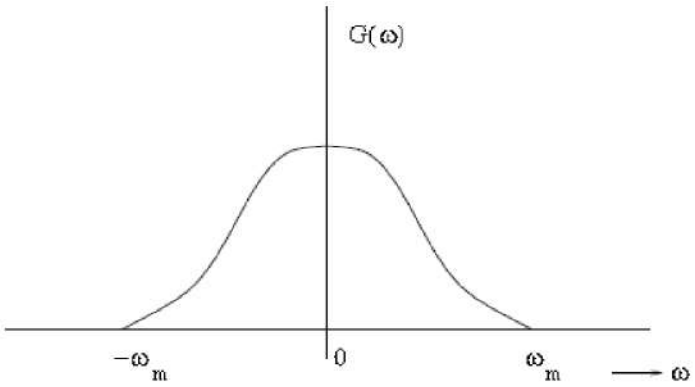


Figure 1: Spectrum of band limited signal g(t)

The maximum frequency component of g(t) is fm. To recover the signal g(t) exactly from its samples it has to be sampled at a rate fs=2fm. The minimum required sampling rate fs = 2fm is called Nyquist rate.

INSTRUCTIONS

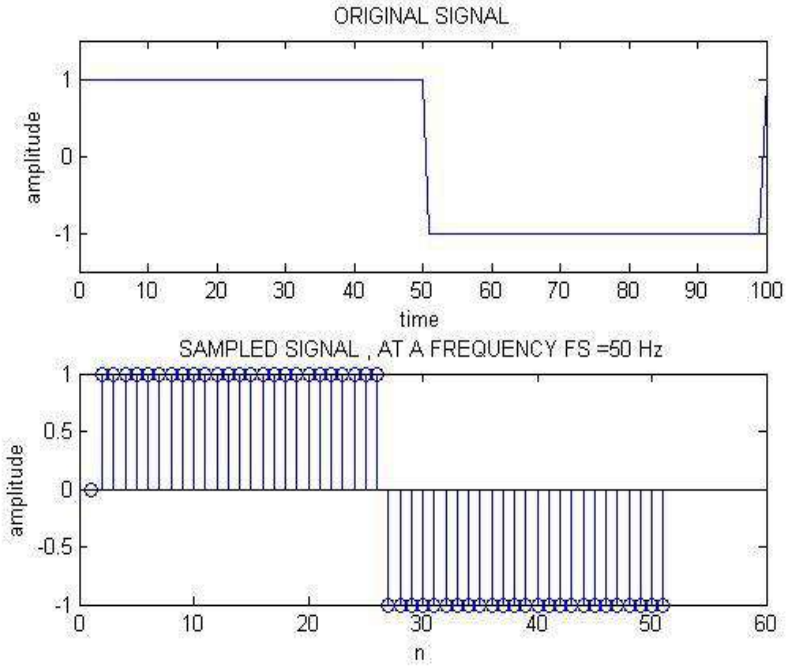
1. Make the program interactive with the user to accept signal frequency and sampling frequency
2. Plot at least one complete cycle of the signal to be sampled. Choose signal frequency as your lab team number in Hz.
3. Choose three different sampling (undersampling and oversampling) frequencies and plot the sampled sequence on the same time scale as the original continuous signal.
4. Choose an appropriate reconstruction filter and plot its impulse response.

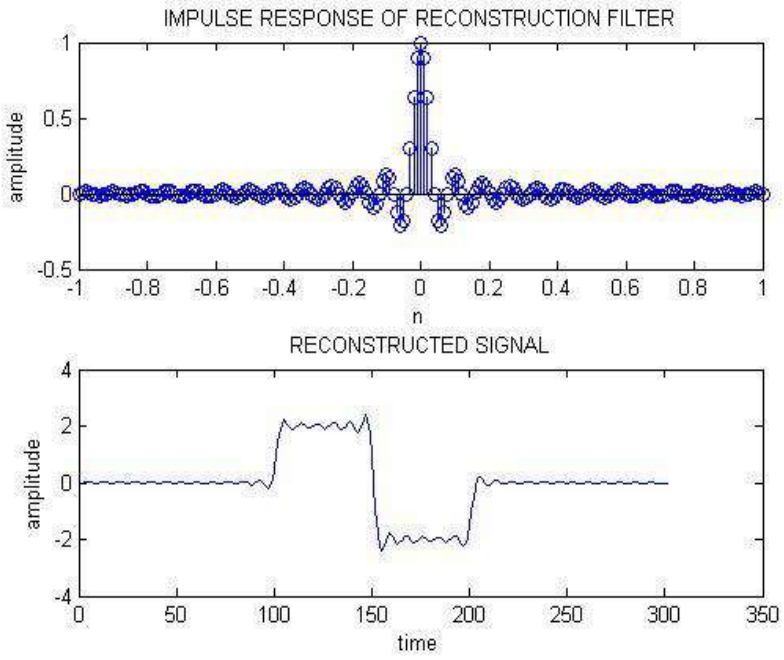
5. Plot the reconstructed signal on the same time scale.
6. Parameters for simulation is selected based on the batch number (N)

SAMPLES OF EXPECTED OUTPUT

THE SIGNAL FREQUENCY =1 Hz

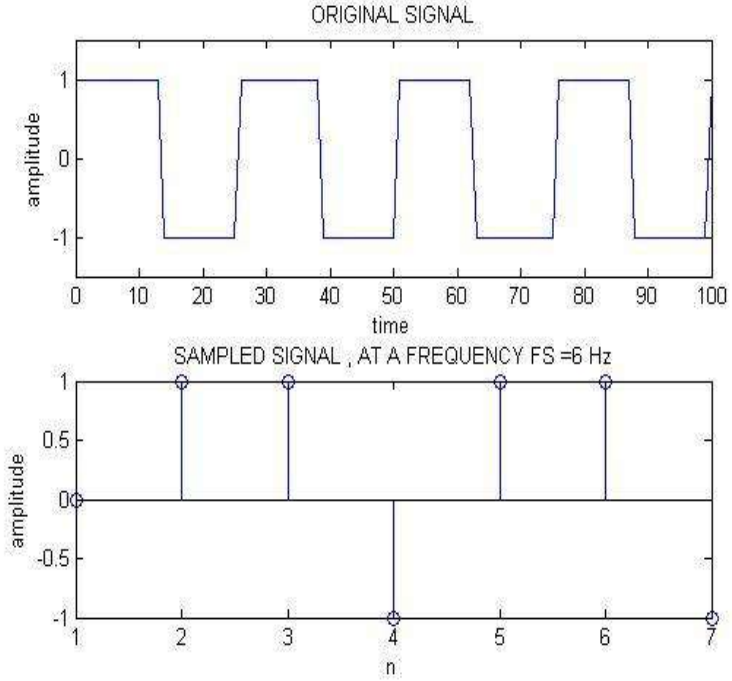
THE SAMPLING FREQUENCY =50 Hz

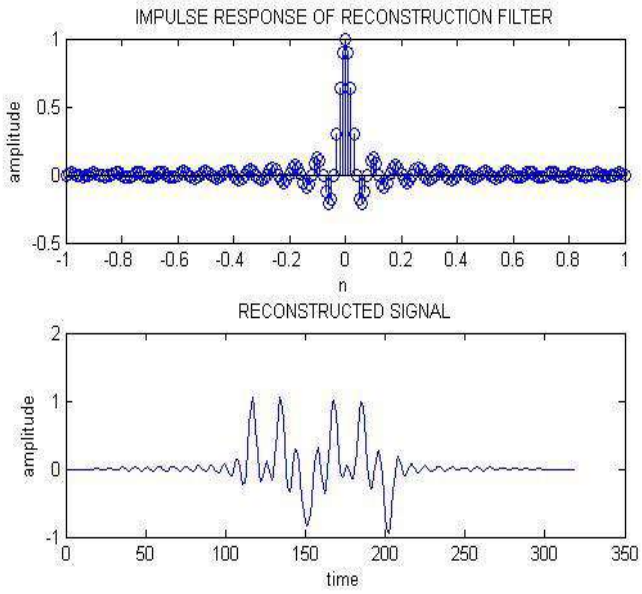




THE SIGNAL FREQUENCY = 4 Hz

THE SAMPLING FREQUENCY =6 Hz





RESULT

The MATLAB program to verify the sampling theory was executed and output waveform for fs>>signal frequency and fs<signal frequency is executed and the theorem is verified.

EXPERIMENT NO 03

TIME AND FREQUENCY RESPONSE OF AN LTI SYSTEM

AIM

Write a MATLAB program to obtain the time and frequency response of an LTI system.

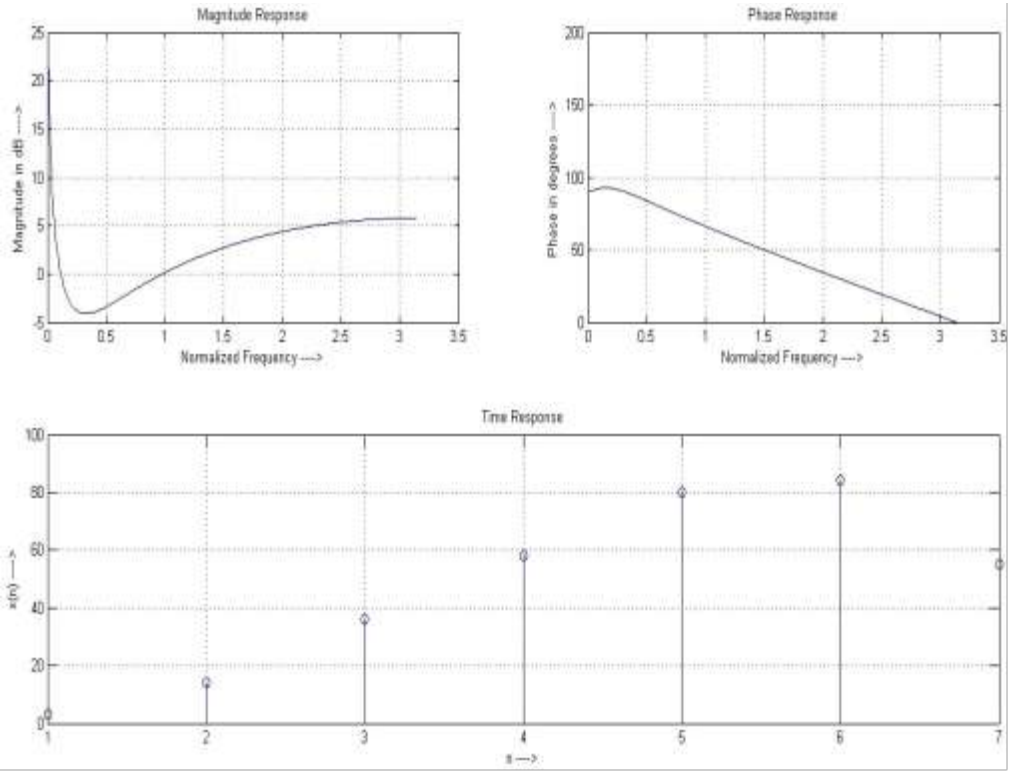
THEORY

Linear Time Systems (LTI Systems) are a class of systems used in DSP that are both linear and time invariant. Linear systems are systems whose output for a linear combination of inputs are the same as a linear combination of individual responses to those inputs. The time invariant systems are systems where the output does not depend on when an input was applied. These properties make LTI systems easy to represent and understand graphically.

INSTRUCTIONS

1. Define a digital filter frequency response using freqz() function, given the transfer function coefficients. eg: `b = [1,-1]; a = [0.9,-2,1]; [h,w] = freqz(a,b);`
2. Plot its magnitude and phase response in dB and degrees respectively

SAMPLES OF EXPECTED OUTPUT



RESULT

The MATLAB program to obtain the time and frequency response of an LTI system is executed and output is obtained.

EXPERIMENT NO 04

LINEAR CONVOLUTION , CIRCULAR CONVOLUTION

& LINEAR CONVOLUTION USING CIRCULAR CONVOLUTION

AIM

a) Write a MATLAB program to perform linear convolution of two input sequences without using inbuilt function and verify the result using MATLAB function.
b) Write a MATLAB program to perform circular convolution of two input sequences without using inbuilt function and verify the result using MATLAB function.
c) Write a MATLAB program to perform linear convolution using circular convolution of two input sequences without using inbuilt function and verify the result using MATLAB function.

THEORY

Convolution is used to find the output response of a digital system. The linear convolution of two continuous time signals x[n] and h[n] is defined by y[n] = x[n] * h[n]. The length of the output sequence y[n]= length of x[n]+length of h[n]-1.

Circular Convolution of two sequences x[n] and y[n], each of length N is given by ,

p[n] = x[n]©y[n].

If the length of sequence is not equal, zero padding is done to get the maximum length among the two. The resulting convolved signal would be zero outside the range n= 0,1,...,N-1.

In order to find the circular convolution of two given sequences without using inbuilt function, we make use of the circular convolution property of DFT.

If X(k) = DFT [x(n)], Y(k) = DFT[y(n)] then convolution property states that,

DFT[x (n)©y (n)] = X (k)Y(k)

INSTRUCTIONS

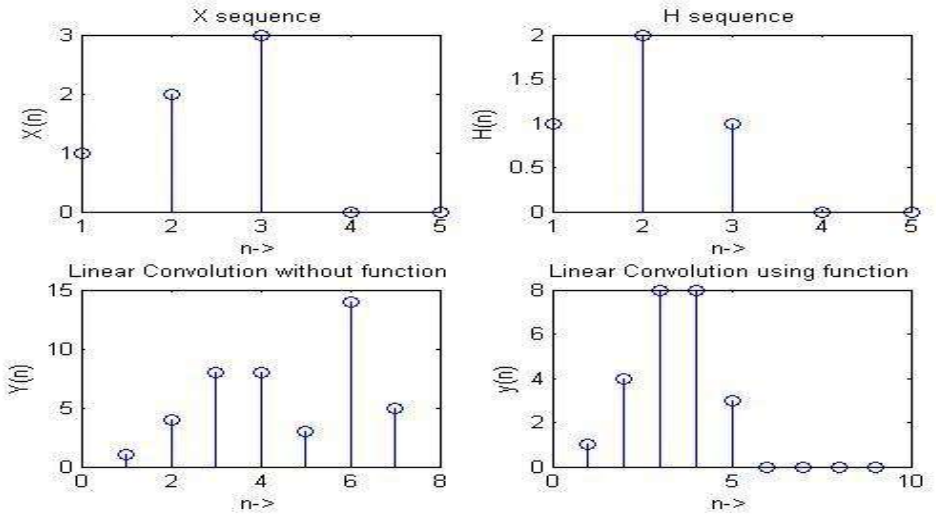
1. Interactively accept two input sequences from the user and perform linear convolution by user defined and builtin functions and verify the result
2. Interactively accept two input sequences from the user and perform circular convolution by user defined and builtin functions and verify the result
3. Define a user defined function that perform linear convolution using circular convolution and then verify the result.

SAMPLES OF EXPECTED OUTPUT

a) Linear convolution

The first sequence, X: [1 2 3]

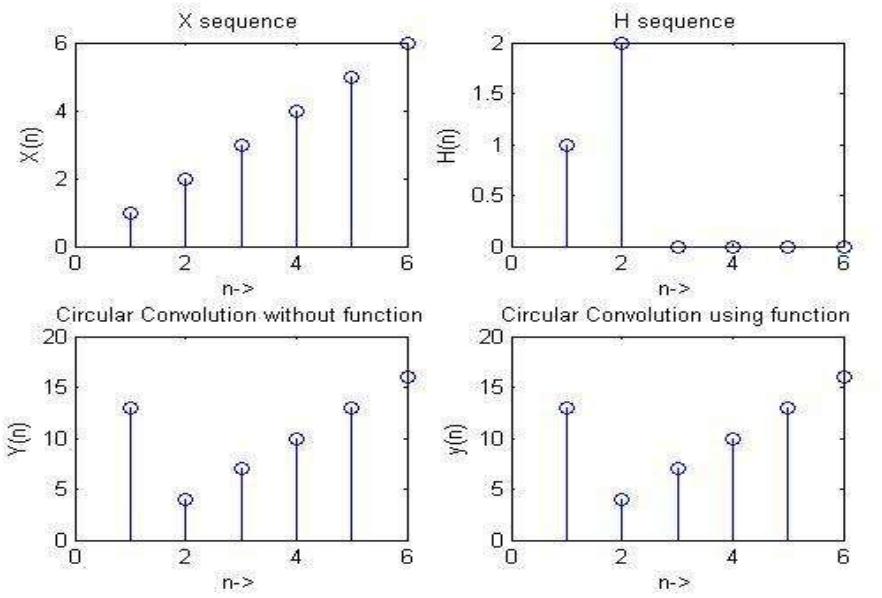
The second sequence, H: [1 2 1]



b) Circular convolution

The first sequence, X = [1 2 3 4 5 6]

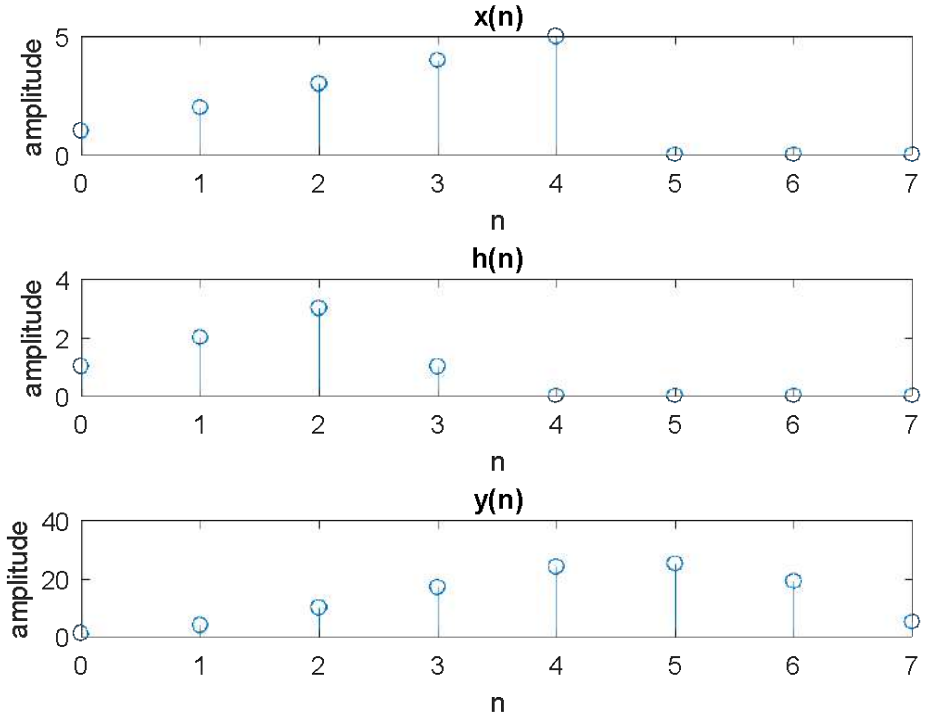
The second sequence, H = [1 2]



c)Linear convolution using circular convolution

The first sequence x(n)= [1 2 3 4 5]

The second sequence h(n)= [1 2 3 1]



RESULT

a) The MATLAB program to find the linear convolution of two input sequences is executed and output is verified using inbuilt function.

b) The MATLAB program to find the circular convolution of two input sequences is executed and output is verified using inbuilt function.

c) The MATLAB program to find the linear convolution using circular convolution of two input sequences is executed and output is verified using inbuilt function.

EXPERIMENT NO 05

DFT AND IDFT

AIM

Write a MATLAB program to find DFT and IDFT of input sequence.

THEORY

Discrete Fourier Transform is the transformation used to represent the finite duration frequencies. DFT of a discrete sequence x(n) is obtained by performing sampling operations in both time domain and frequency domain. It is the frequency domain representation of a discrete digital signal.

The DFT of a sequence x (n) of length N is given by the following equation,

$$X(k) = \left\{ \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}} ; 0 \le k \le N-1 \right\}$$

IDFT performs the reverse operation of DFT, to obtain the time domain sequence x(n) from frequency domain sequence X(k). IDFT of the sequence is given as,

$$x(n) = \frac{1}{N}$$

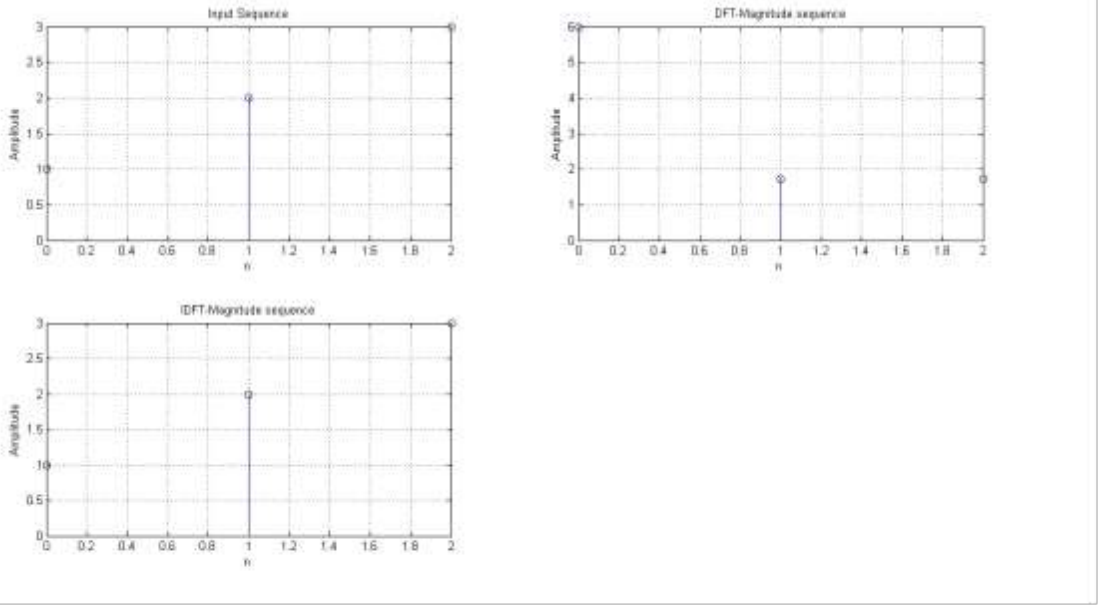
INSTRUCTIONS

1. Accept an input sequence from the user, interactively
2. Without using inbuilt functions, compute the DFT of the sequence. Plot the magnitude response.
3. Using the sequence obtained as DFT in the previous step as input, compute the IDFT without using inbuilt functions.
4. Plot the IDFT and verify it is same as the original input.

SAMPLES OF EXPECTED OUTPUT

N= 3

Sequence=[1 2 3]



RESULT

The MATLAB program to find DFT and IDFT of input sequence is executed and output is obtained.

EXPERIMENT NO 06

LINEAR CONVOLUTION USING DFT (OVERLAP-ADD & OVERLAP-SAVE METHODS)

AIM

To simulate linear convolution of two signals using overlap add and overlap save methods.

THEORY

Generally in discrete-time processing we talk about linearly convolving a sequence with a FIR filter. In contrast, the FFT performs circular convolution with a filter of equal or lesser length. In computing the DFT or FFT we often have to make linear convolution behave like circular convolution or vice versa. Two methods that make linear convolution look like circular convolution are overlap-add and overlap-save.

i. Overlap-Add Method

This procedure cuts the signal up into equal length segments with no overlap. Then it zero-pads the segments and takes the DFT of the segments. Part of the convolution result corresponds to the circular convolution. The tails that do not correspond to the circular convolution are added to the adjoining tail of the previous and subsequent sequence. This addition results in the aliasing that occurs in circular convolution.

ii. Overlap-Save Method

This procedure cuts the signal up into equal length segments with some overlap. Then it takes the DFT of the segments and saves the parts of the convolution that correspond to the circular convolution. Because there are overlapping sections, it is like the input is copied therefore there is not lost information in throwing away parts of the linear convolution.

INSTRUCTIONS

1. Interactively accept input sequence and impulse sequence from the user. Also accept the block length of processing
2. Perform zero padding followed by linear convolution using inbuilt functions fft() and ifft() for overlap add method
3. Repeat the same using overlap save algorithms

4. Plot the sequences in each case.

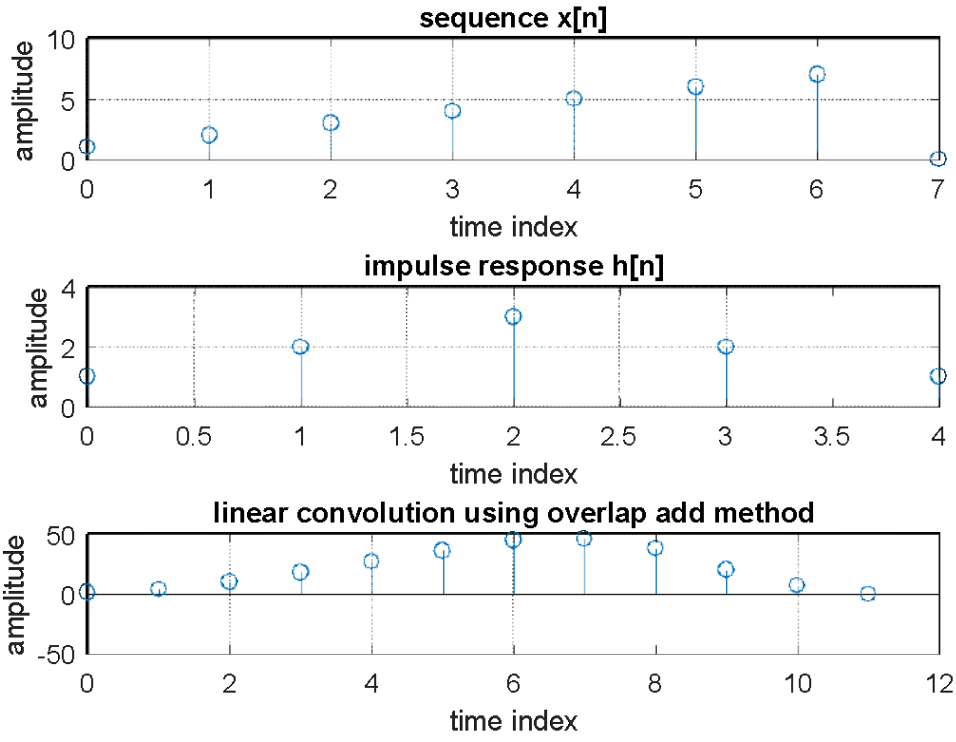
SAMPLES OF EXPECTED OUTPUT

Overlap-Add

The signal sequence: [1 2 3 4 5 6 7]

The impulse sequence:[1 2 3 2 1] '

The block length for processing L: 4

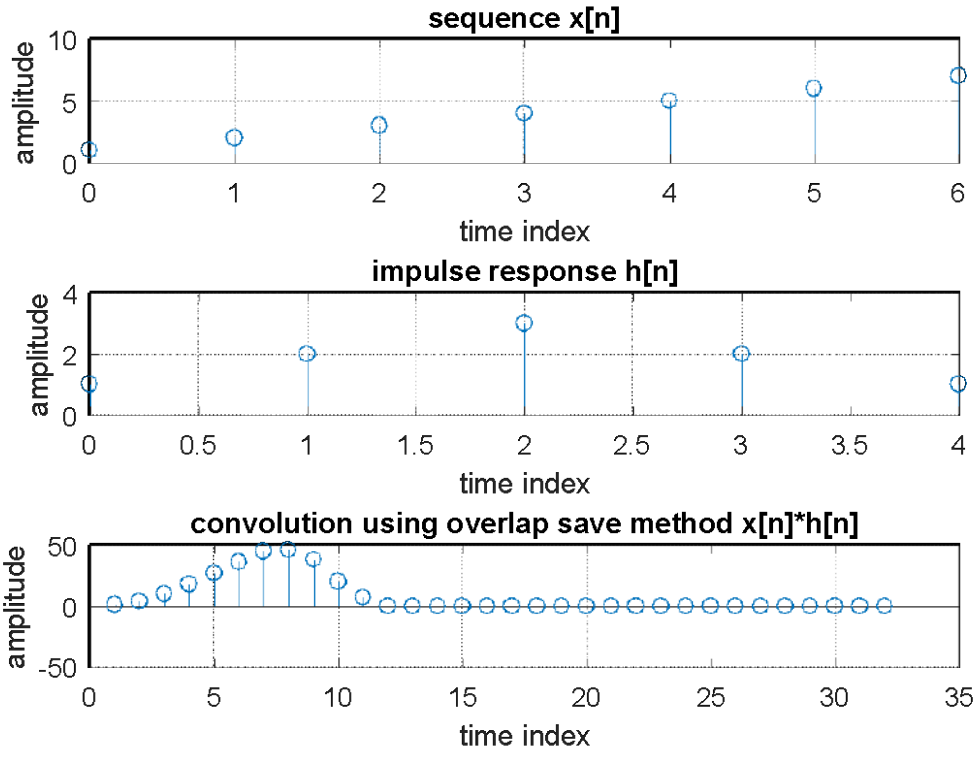


Overlap-Save

The signal sequence: [1 2 3 4 5 6 7]

The impulse sequence:[1 2 3 2 1] '

The block length for processing L: 4



RESULT

The linear convolution of two signals performed using overlap add and overlap save methods.

EXPERIMENT NO 07

DCT AND IDCT

AIM

Write a MATLAB program to find the DCT and IDCT for the given input.

THEORY

A D**iscrete Cosine Transform** (**DCT**) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. The use of cosine rather than sine functions is critical for compression, since it turns out that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions. In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers.

One-Dimensional DCT Equation can be expressed as,

$$X_C(k) = (c(k)/2) \quad \cos((2n+1)\pi k/2N), \quad \text{Where } k = 0, 1, 2, \ldots, N-1$$

The IDCT function is the inverse of the DCT function — The IDCT reconstructs a sequence from its discrete cosine transform (DCT) coefficients.

$$x(n) = \sum_{k=0}^{N-1} \left(c\frac{[k]}{2} \right) X_c[k] \quad \cos((2n+1))\pi k/N)$$

where, n = 0, 1, 2, …, N-1

X_c is the DCT result

c[k] =1 for k=0

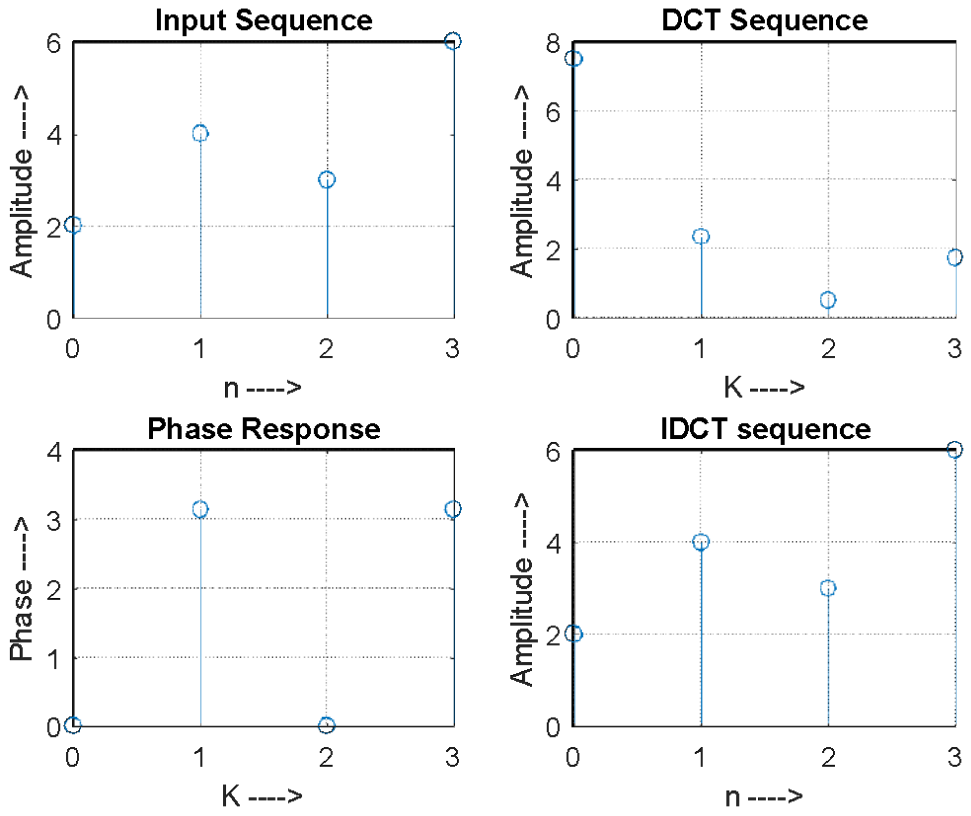
=2 for k=1,2,3,……,N-1

INSTRUCTIONS

1. Accept a sequence form the user.
2. Without using inbuilt functions, write a program to find the DCT of the input sequence
3. Using the DCT output, perform IDCT without using built in functions
4. Plot all input and output sequences

SAMPLE OF EXPECTED OUTPUT

```
The input sequence = [2 4 3 6]
```



RESULT

The MATLAB program to find the DCT and IDCT for the given input was executed and output obtained correctly.

EXPERIMENT NO 08

FFT AND IFFT

AIM

Write a MATLAB program to find the FFT and IFFT of the given input.

THEORY

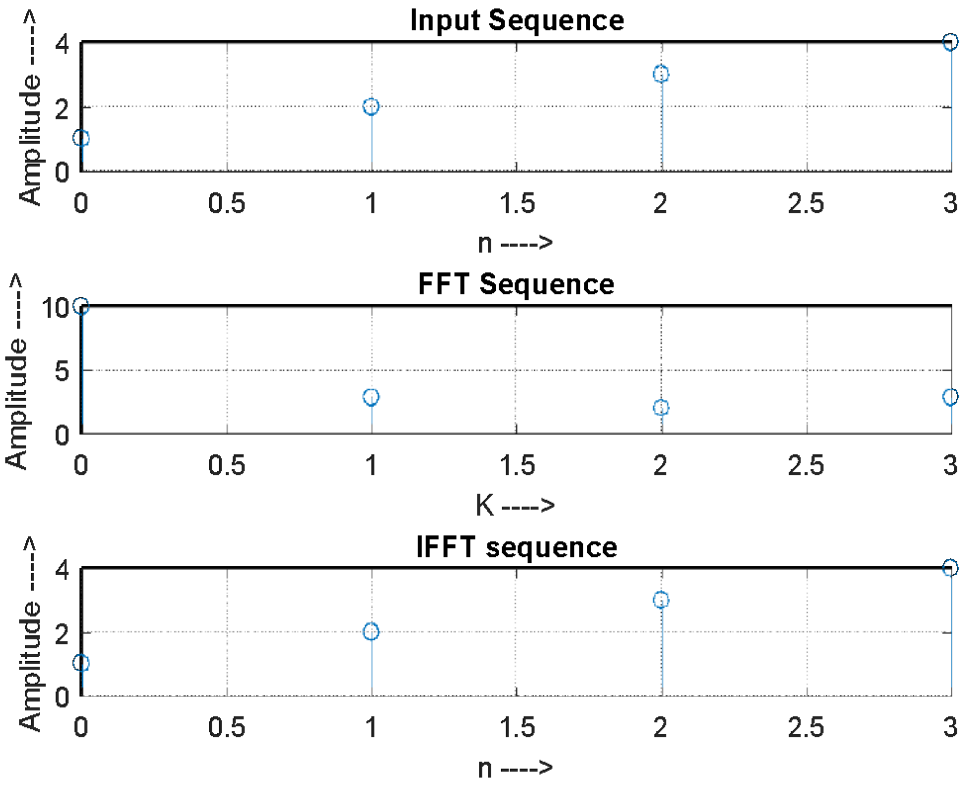
The most common tools used to perform Fourier analysis and synthesis are called the fast Fourier transform (FFT) and the inverse fast Fourier transform (IFFT). The FFT and IFFT are optimized (very fast) computer-based algorithms that perform a generalized mathematical process called the *discrete Fourier transform* (DFT). The DFT is the actual mathematical transformation that the data go through when converted from one domain to another (time to frequency). Basically, the DFT is just a slow version of the FFT.

INSTRUCTIONS

1. Accept a sequence form the user.
2. Without using inbuilt functions, write a program to find the FFT of the input sequence
3. Using the FFT output, perform IFFT without using built in functions
4. Plot all input and output sequences

SAMPLE OF EXPECTED OUTPUT

```
The input sequence = [1 2 3 4]
```



RESULT

The MATLAB program to find the FFT and IFFT of the given input sequence

EXPERIMENT NO 09

FIR AND IIR FILTER DESIGN USING FILTER DESIGN TOOLBOX

AIM

FIR and IIR filter design using Filter Design Toolbox.

THEORY

Filter Design and Analysis using FDATool of MATLAB:

The Filter Design and Analysis Tool (FDATool) is a powerful user interface for designing and analyzing filters quickly. FDATool enables you to design digital FIR or IIR filters by setting filter specifications, by importing filters from your MATLAB workspace, or by adding, moving or deleting poles and zeros. FDATool also provides tools for analyzing filters, such as magnitude and phase response and pole-zero plots. FDATool seamlessly integrates additional functionality from other MathWorks products.

Use FDATOOL in matlab.

If you type

>>fdatool

in command window, FDAtool will be opened. There you can select FIR or IIR filter, order of filter and cutoff frequency of a filter (either HPF, LPF or BPF). That code will automatically generate .m file for you.

GETTING STARTED:

Type fdatool at the MATLAB command prompt:

```
>> fdatool
```

A Tip of the Day dialog displays with suggestions for using FDATool. Then, the GUI diplays with a default filter.

The GUI has three main regions:

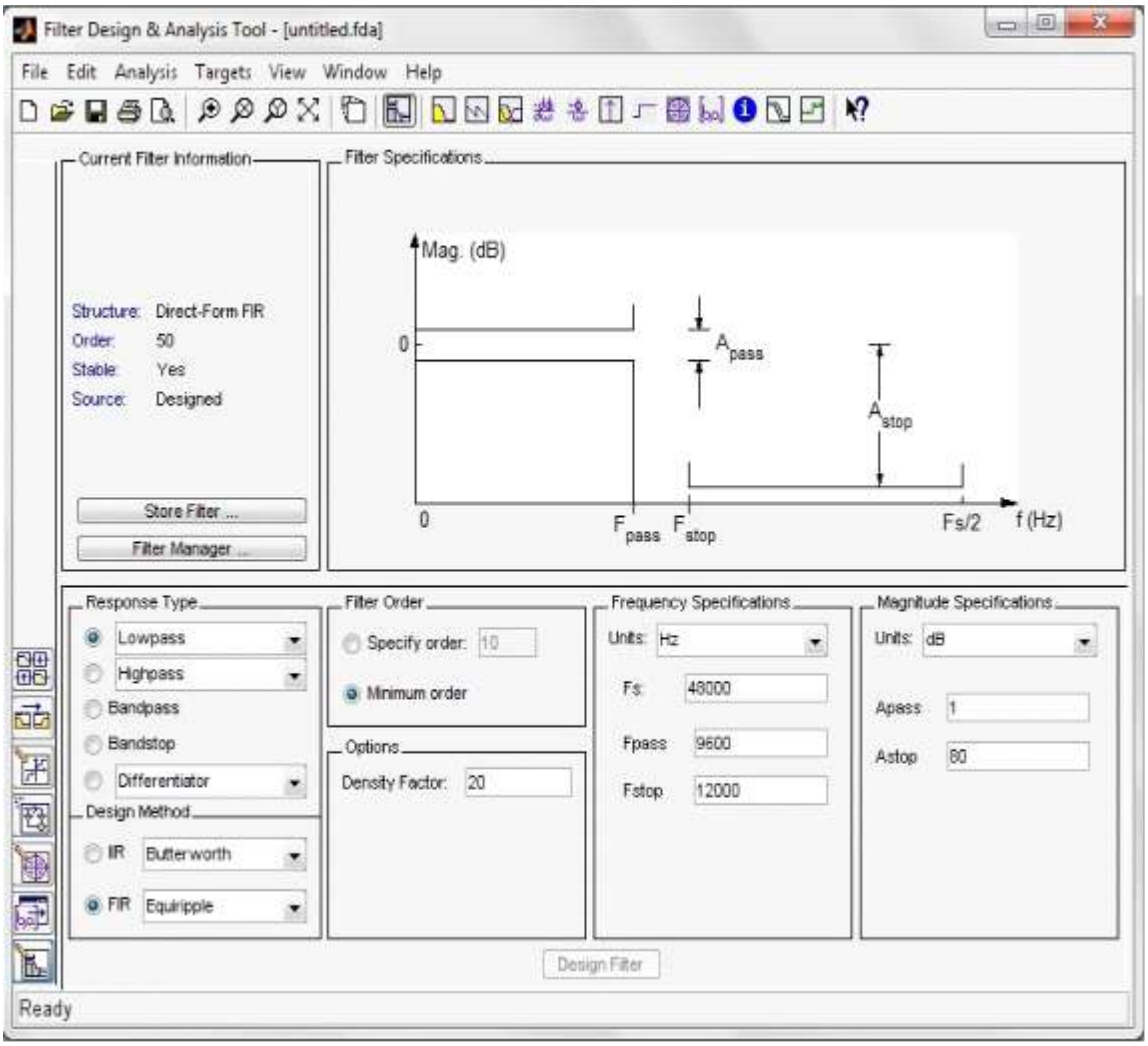
1.The Current Filter Information region

2.The Filter Display region and

3.The Design panel

The upper half of the GUI displays information on filter specifications and responses for the current filter. The Current Filter Information region, in the upper left, displays filter properties, namely the filter structure, order, number of sections used and whether the filter is stable or not. It also provides access to the Filter manager for working with multiple filters.

The Filter Display region, in the upper right, displays various filter responses, such as, magnitude response, group delay and filter coefficients.The lower half of the GUI is the interactive portion of FDATool. The Design Panel, in the lower half is where you define your filter specifications. It controls what is displayed in the other two upper regions. Other panels can be displayed in the lower half by using the sidebar buttons.



DESIGNING FILTER:

-

IIR FILTER

We will design a butterworth low pass filter with following specifications:

Pass band frequency: 4000

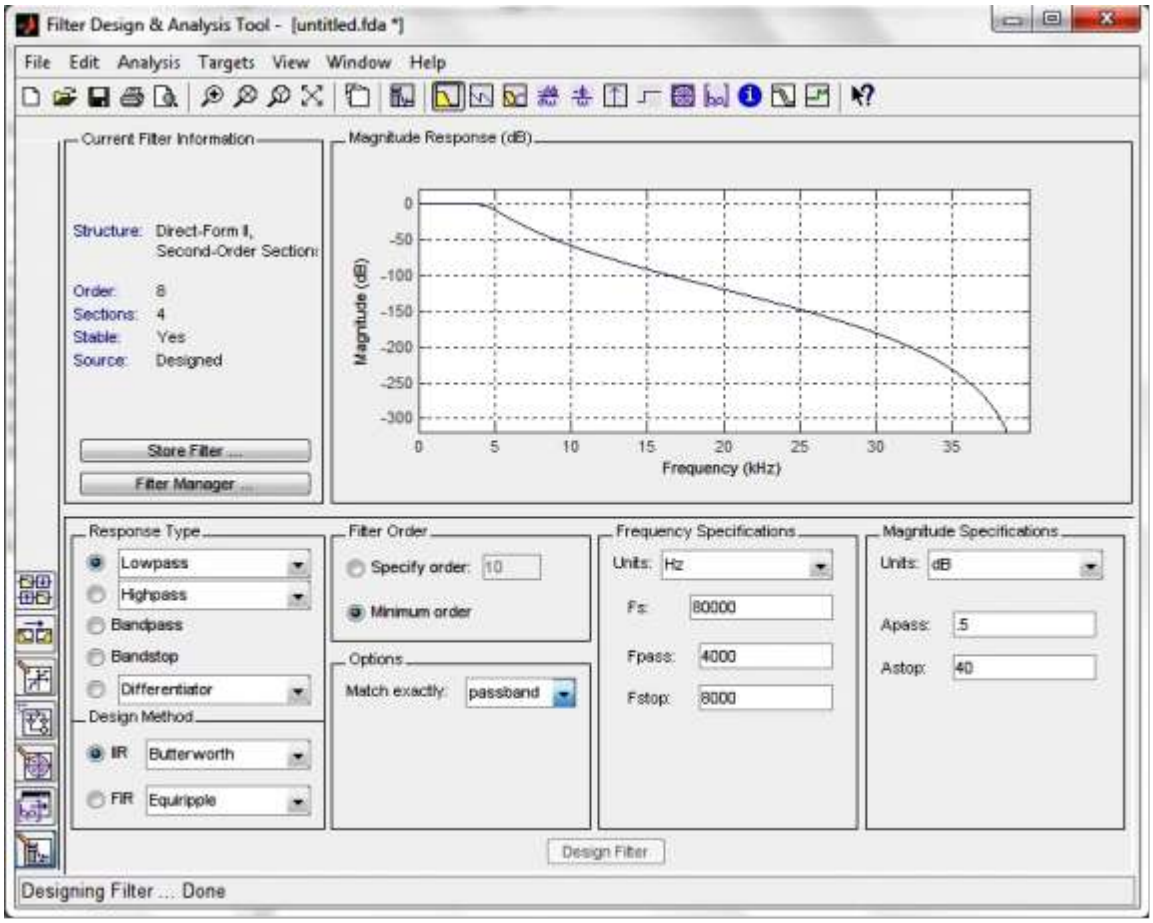
Stop band frequency: 8000

Pass band ripple: .5

Stop band ripple: 40

Sampling Frequency: 80000

To implement this design, we will use the following specifications:



1. Select **Lowpass**from the dropdown menu under **Response Type** and **BUTTERWORTH** under **IIR Design Method**. In general, when you change the Response Type or Design Method, the filter parameters and Filter Display region update automatically.

2. Select **Minimum order** in the **Filter Order.**

3. Select **Hz** in the Units pull down menu in the **Frequency Specifications** area. Then input sampling frequency,Pass band frequency and stop band frequency.

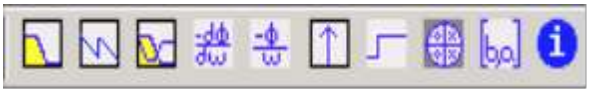
4. Select **Db**in the units pull down menu in the **Magnitude Specifications** area.Enter**0.5** for **A pass**and **40** for **Astop**in the **Magnitude Specifications** area.

5. After setting the design specifications, click the **Design Filter** button at the bottom of the GUI to design the filter.

The magnitude response of the filter is displayed in the Filter Analysis area after the coefficients are computed.

Viewing other Analyses:

Once you have designed the filter, you can view the following filter analyses in the display window by clicking any of the buttons on the toolbar:



In order from left to right, the buttons are

1. Magnitude response
2. Phase response
3. Magnitude and Phase responses
4. Group delay response
5. Phase delay response
6. Impulse response
7. Step response
8. Pole-zero plot
9. Filter Coefficients
10. Filter Information

FIR FILTER

We will design a FIR low pass filter using hanning window with following specifications:

Sampling Frequency: 80000

Cut-off Frequency : 4000

To implement this design, we will use the following specifications:

1. Select **Low pass** from the dropdown menu under **Response Type** and **Window** under **FIR Design Method**. In general, when you change the Response Type or Design Method, the filter parameters and Filter Display region update automatically.

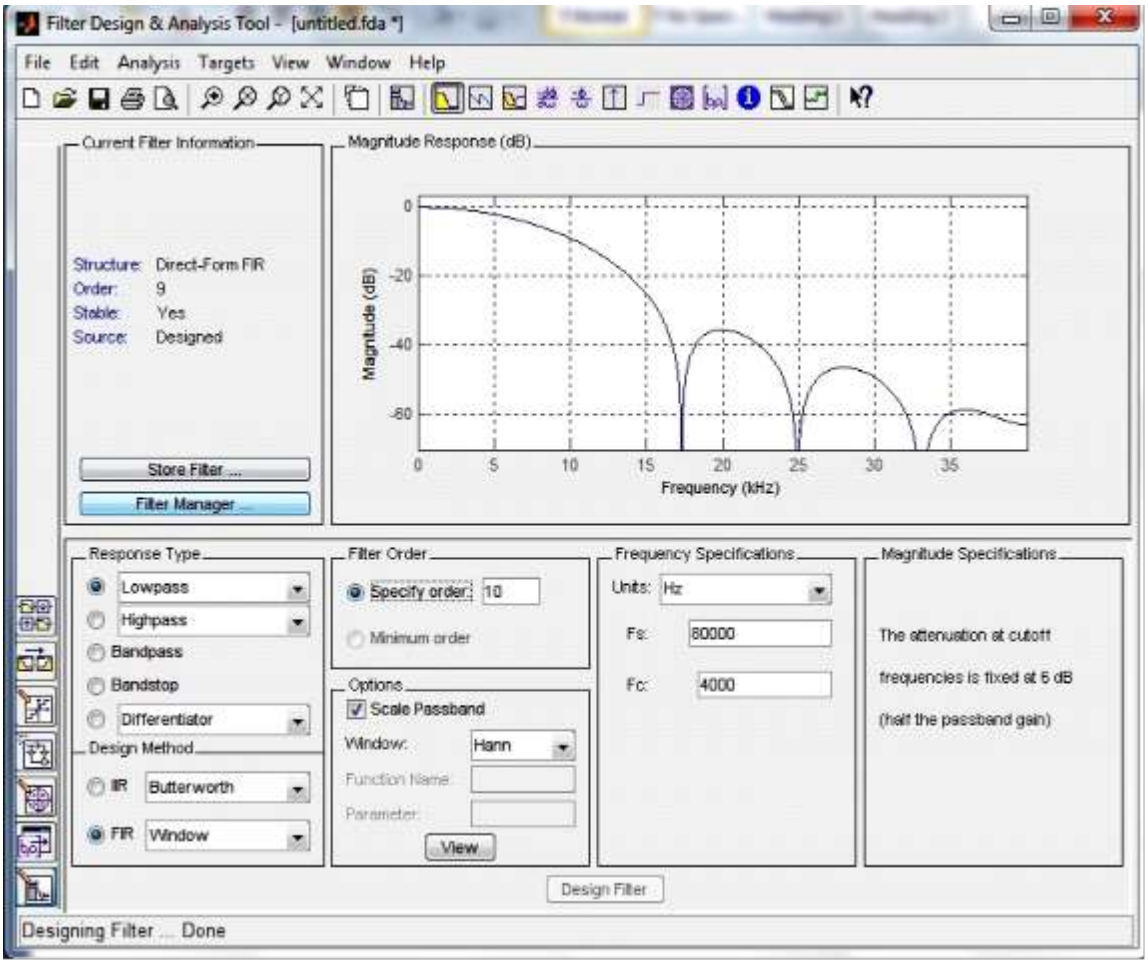
2. Enter **10** in **Specify order** under the **Filter Order.**

3. Select **Hann** in **Window.**

4. Select **Hz** in the Units pull down menu in the **Frequency Specifications** area. Then input sampling frequency and cut-off frequency.

5. After setting the design specifications, click the **Design Filter** button at the bottom of the GUI to design the filter.

The magnitude response of the filter is displayed in the Filter Analysis area after the coefficients are computed.



RESULT:

Designed FIR and IIR filter Using Filter Design Tool.

EXPERIMENT NO 10

FIR FILTER DESIGN (WINDOW METHOD)

AIM

To design and implement a finite impulse response filter of linear phase characteristics by window method.

THEORY

The rectangular window sequence is given by

$$\omega_R(n)=1\ for\ -\frac{N-1}{2}\le n\le\frac{N-1}{2}\ 0\ otherwise$$

In the design of FIR filters using any window technique, the order can be calculated using the formula given by

$$N=\frac{-20\log\left(\sqrt{\delta_p\delta_s}\right)}{14.6\frac{\left(f_s-f_p\right)}{F_s}}-13$$

Where δ_p is the pass band ripple, is the stop band ripple, f_p is the pass band frequency, f_s is the stop band frequency and F_s is the sampling frequency.

The equation for hamming window is given by

$$\omega_H(n)=0.54+\frac{0.46\cos 2\pi n}{N-1}\ for\ -\frac{N-1}{2}\le n\le\frac{N-1}{2}\ 0,\ otherwise$$

The hanning window sequence can be obtained by substituting α=0.5 in the raised cosine window function. The sequence is obtained as

INSTRUCTIONS

1. Interactively accept the filter type and window type from th user. Use `questdlg()` and `switch..case()` functions
2. Interactively accept the order, cutoff frequency and sampling frequency from the user

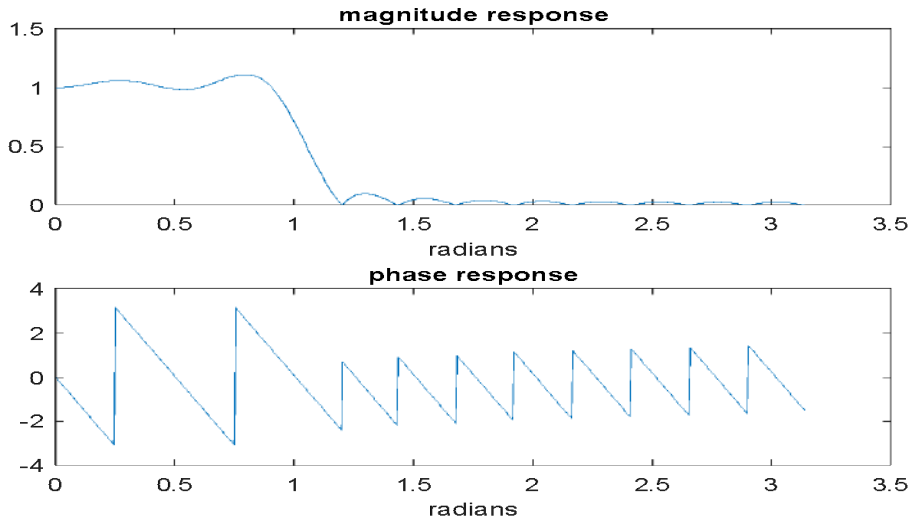
SAMPLES OF EXPECTED OUTPUT

1. Low pass Filter using Rectangular window

Order of the filter : 25

Cutoff frequency : 500

Sampling frequency : 3000

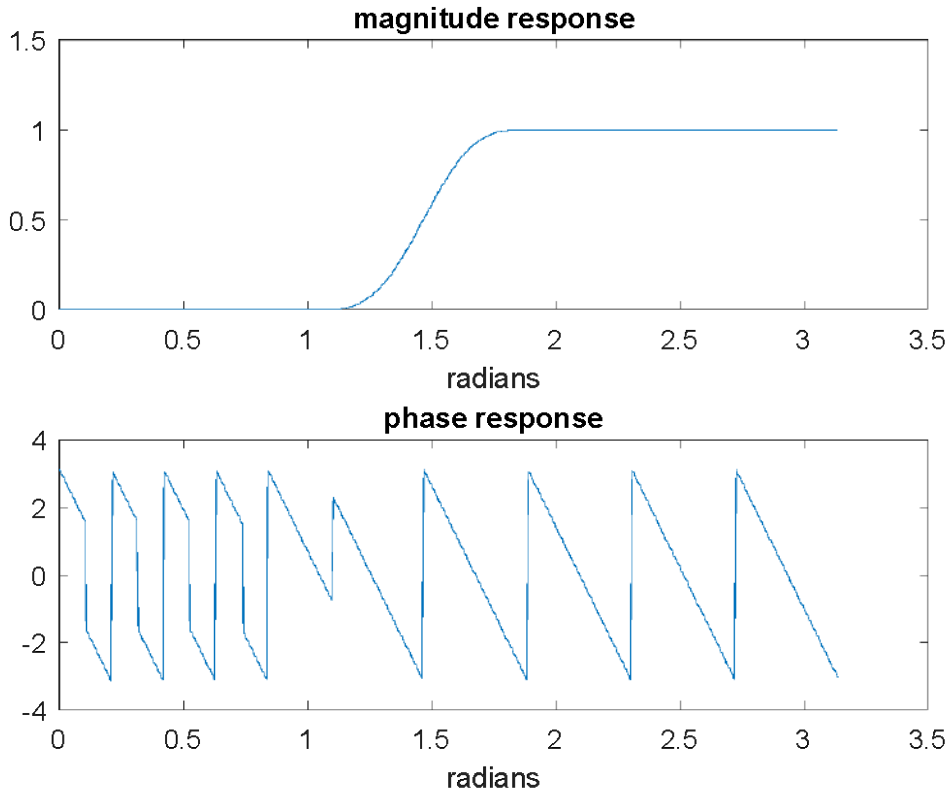


2. High pass Filter using Hamming window

The order of the filter : 30

The cutoff frequency : 700

Sampling frequency : 3000



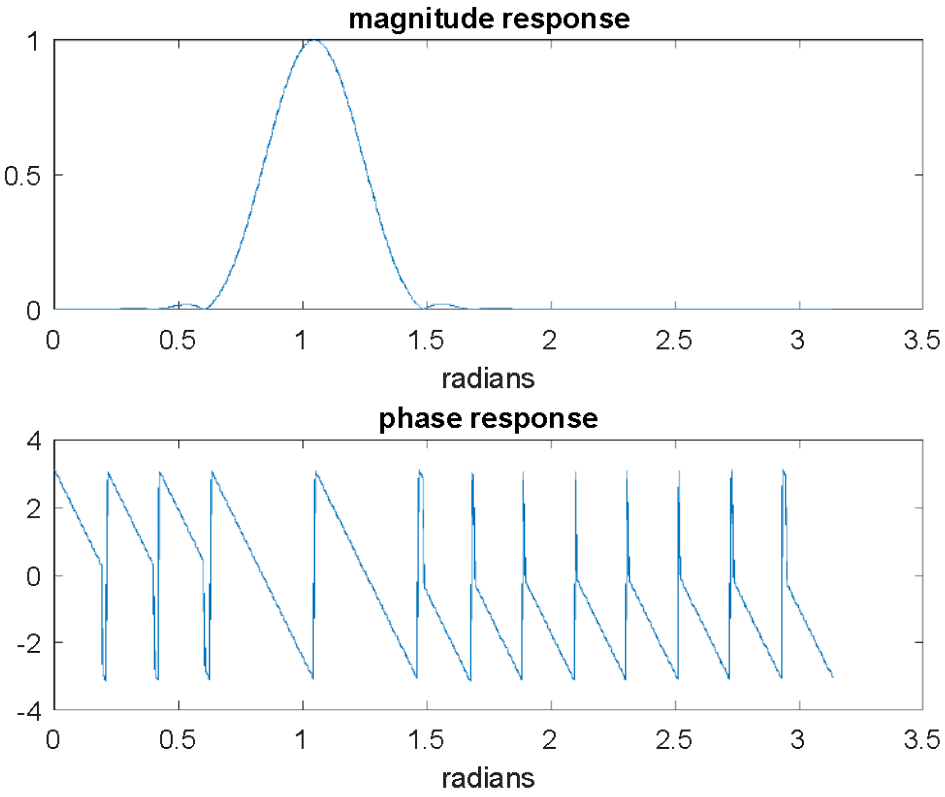
3. **Band pass Filter using Hanning window**

   The order of the filter :30

   The cutoff frequency1 : 460

   The cutoff frequency2 :540

   Sampling frequency: 3000



RESULT

Designed and implemented a finite impulse response filter of linear phase characteristics by window method.

EXPERIMENT NO: 11

IIR FILTER DESIGN

AIM

Write a MATLAB program to design an IIR Filter(Butterworth and Chebychev).

THEORY

The Butterworth filter is a type of signal processing filter designed to have as flat a frequency response as possible in the pass band so that it is also termed a maximally flat magnitude filter. The frequency response of the Butterworth filter is maximally flat (has no ripples) in the passband and rolls off towards zero in the stopband. When viewed on a logarithmic Bode plot the response slopes off linearly towards negative infinity. A first-order filter's response rolls off at −6 dB per octave (−20 dB per decade) (all first-order lowpass filters have the same normalized frequency response). A second-order filter decreases at −12 dB per octave, a third-order at −18 dB and so on. Butterworth filters have a monotonically changing magnitude function with ω, unlike other filter types that have non-monotonic ripple in the passband and/or the stopband.

Chebyshevfilters are analog or digital filters having a steeper roll-off and more pass band ripple (type I) or stop band ripple (type II) than Butterworth filters. Chebyshev filters have the property that they minimize the error between the idealized and the actual filter characteristic over the range of the filter, but with ripples in the passband.Because of the passband ripple inherent in Chebyshev filters, filters which have a smoother response in the passband but a more irregular response in the stopband are preferred for some applications.

These are the most common Chebyshev filters. The gain (or amplitude) response as a function of angular frequency ω of the *n*th order low pass filter is

$$G_n(\omega) = \left| H_n(j\omega) \right| = \frac{1}{\sqrt{\square}}$$

Whereε is the ripple factor, ω_0 is the cutoff frequency and $T_n()$ is a Chebyshev polynomial of the n^{th} order.

INSTRUCTIONS

1. Accept the butterworth filter design parameters for the user (Pass band and stop band cut-off frequencies, and attenuations)
2. For each type of lowpass, bandpass, high pass filter, find the filter order and implement the filter design using inbuilt functions
3. Repeat the same for Chebyshev filter design
4. Plot filter magnitude and phase response

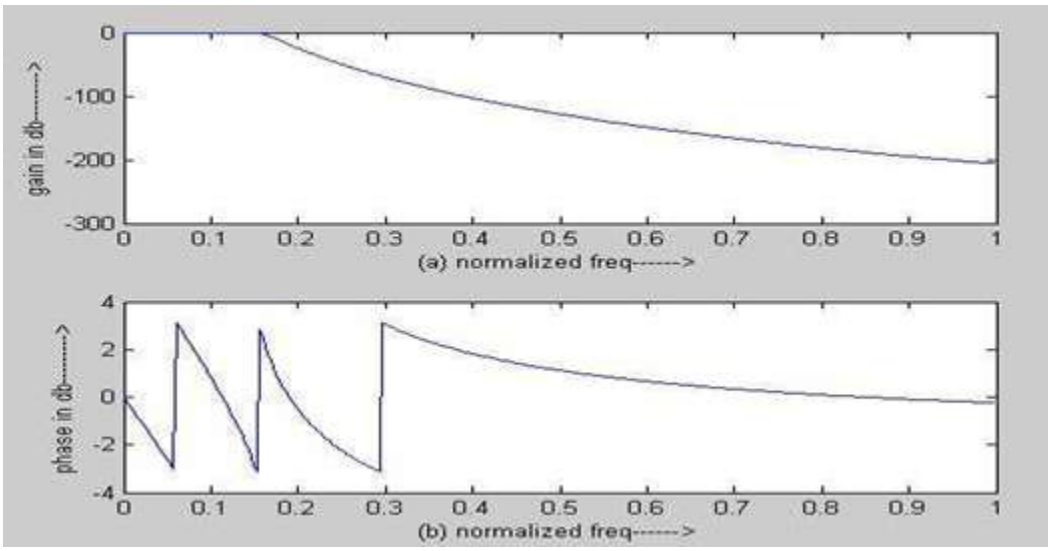
SAMPLES OF EXPECTED OUTPUT:

BUTTERWORTH

a) **Low pass**

The pass band frequency wp = 1500
The stop band frequency ws = 3000
The pass band attenuation rp = .15
The stop band attenuation rs = 60
The sampling frequency fs = 70000

n =8
wn = 0.1151
n2 = 5
wn2 = 0.1000
N = 512



b)High pass

The pass band frequency wp =2000

The stop band frequency ws = 3500

The pass band attenuation rp = 0.2

The stop band attenuation rs = 40

The sampling frequency fs = 80000

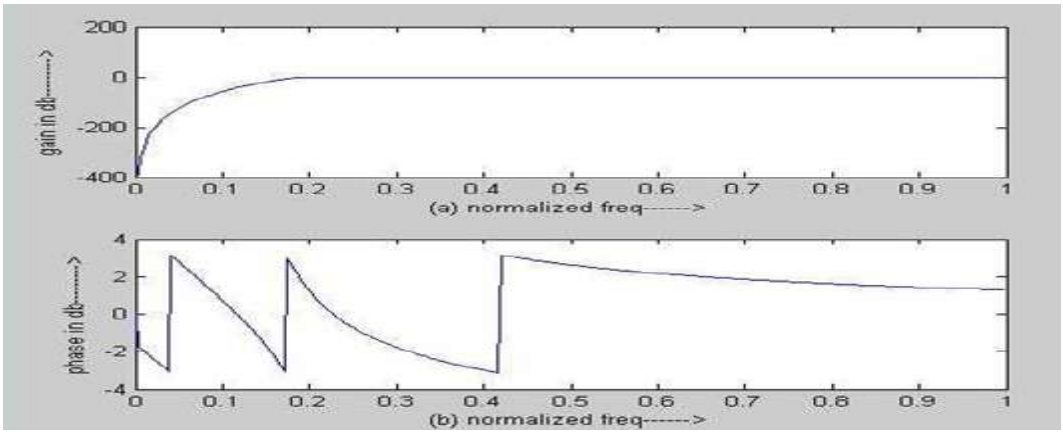
n =8

wn = 0.1151

n2 = 5

wn2 = 0.1000

N = 512



c) Bandpass

The pass band frequency wp =1500

The stop band frequency ws = 2000

The pass band attenuation rp = 0.3

The stop band attenuation rs = 40

The sampling frequency fs = 90000

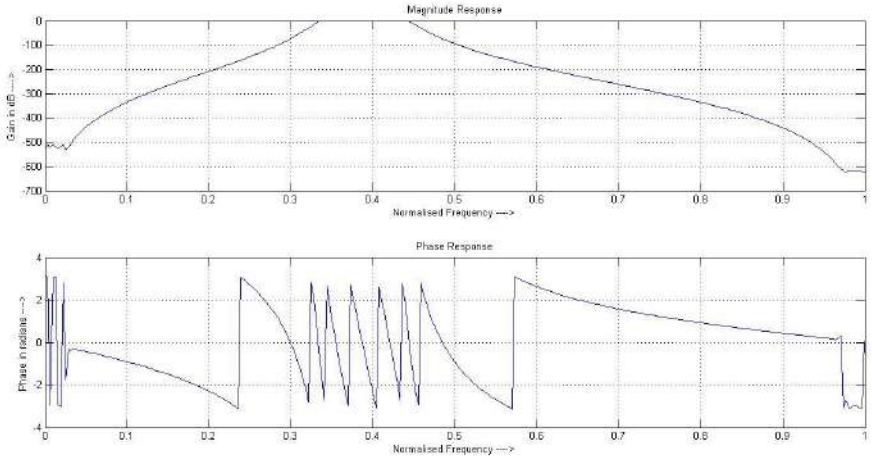
n =8

wn = 0.1151

n2 = 5

wn2 = 0.1000

N = 512



CHEBYSHEV

a)Lowpass

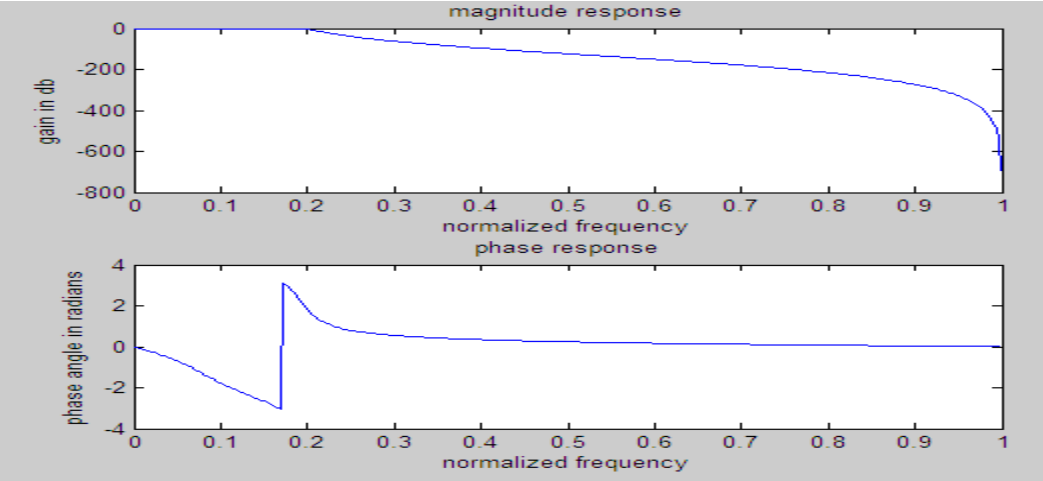
The passband attenuation=2

The stopband attenuation=20

The stopband edge frequency=200

The passband edge frequency=300

The sampling frequency=2000



b)Highpass

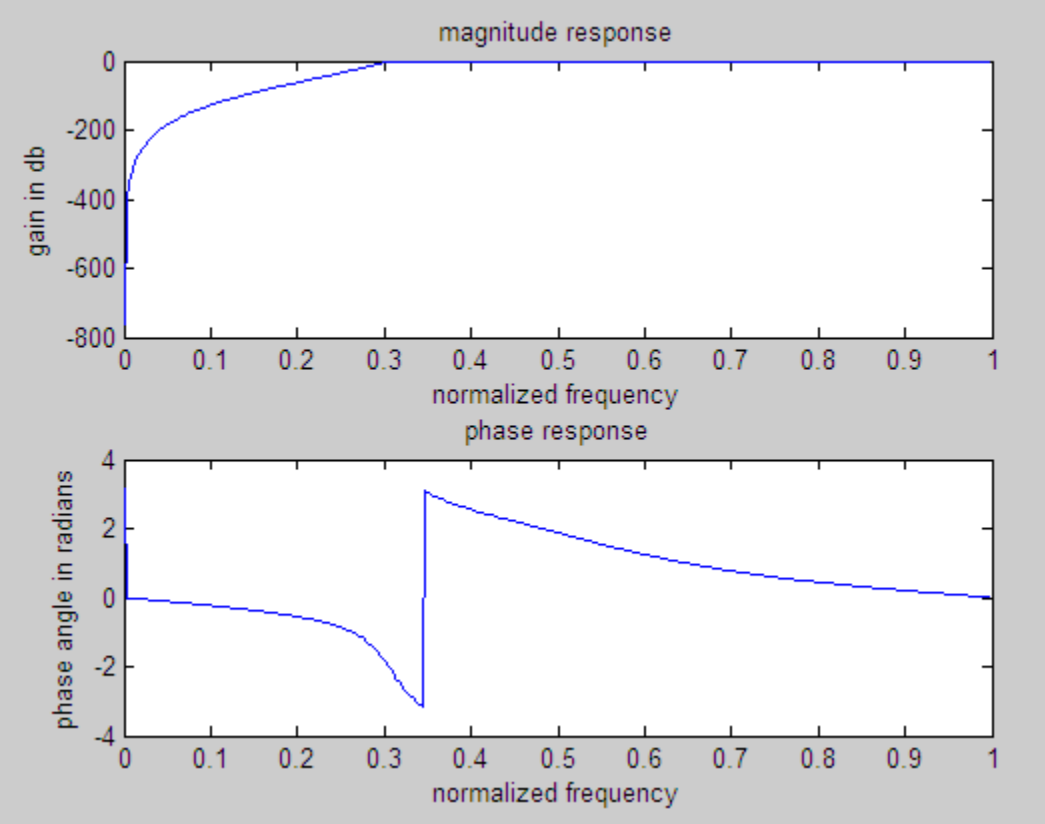
The passband attenuation=2

The stopband attenuation=20

The stopband edge frequency=200

The passband edge frequency=300

The sampling frequency=2000



c)Bandpass

The passband attenuation=2

The stopband attenuation=20

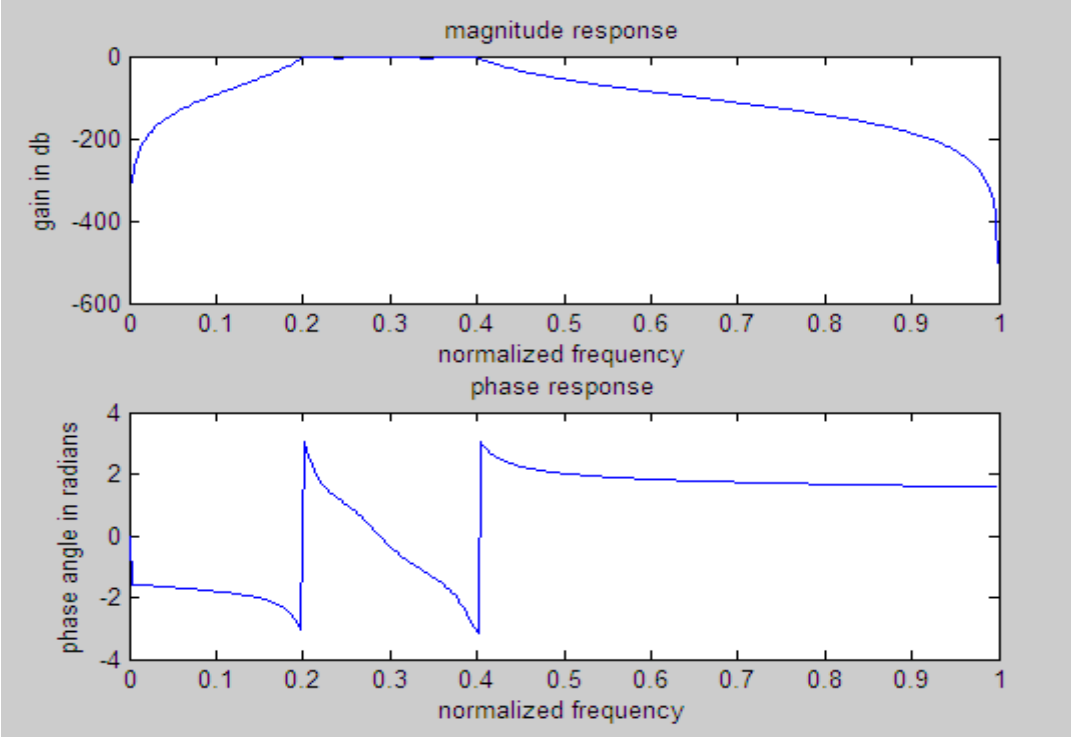
The upper stopband edge frequency=500

The lower stopband edge frequency=100

The upper passband edge frequency=400

The lower passband edge frequency=200

The sampling frequency=2000



RESULT:

Designed Butterworth and Chebyshev Filters.

EXPERIMENT NO 12

GENERATION OF AM, FM & PWM WAVEFORMS AND THEIR SPECTRUM

AIM

a) To generate amplitude modulated wave with and without using standard Matlab functions and demodulate the AM wave obtained..
b) To generate frequency modulated wave with and without using standard Matlab functions. Also demodulate the frequency modulated wave.
c) To generate PWM modulated wave .

THEORY

a) Amplitude Modulation

The process by which the amplitude of a relatively high frequency wave called the carrier wave is changed in accordance with the instantaneous value of amplitude of a low frequency electromagnetic wave called the modulating wave is known as amplitude modulation. If $E_c \cos w_c t$ denotes the carrier signal and $e_m(t) = E_m \cos w_m t$ denotes the message signal, then amplitude modulated signal is given by:

$$\mathbf{AM} = \mathbf{E_c}[1 + Ka * e_m(t)] \cos w_c t$$

Where,m is the modulation index; E_c is the maximum amplitude of the carrier; $_c$ and $_m$ are the angular frequencies of the carrier and message wave.

b) Frequency Modulation

The process by which the frequency of the carrier wave is changed in accordance with the instantaneous value of the modulating wave is known as frequency modulation.Frequency modulated wave can be expressed as

$$\textbf{FM=}\ E_c\ \ \textbf{sin[}\ \ w_c t + \left(\frac{k_f * E_m}{w_m}\right) * \sin w_m t\ \ \textbf{)]}$$

Where E_c is the carrier amplitude; k_f is the frequency conversion factor. w_c, w_m are the angular frequencies of the carrier and message wave.

c) Pulse Width Modulation(PWM)

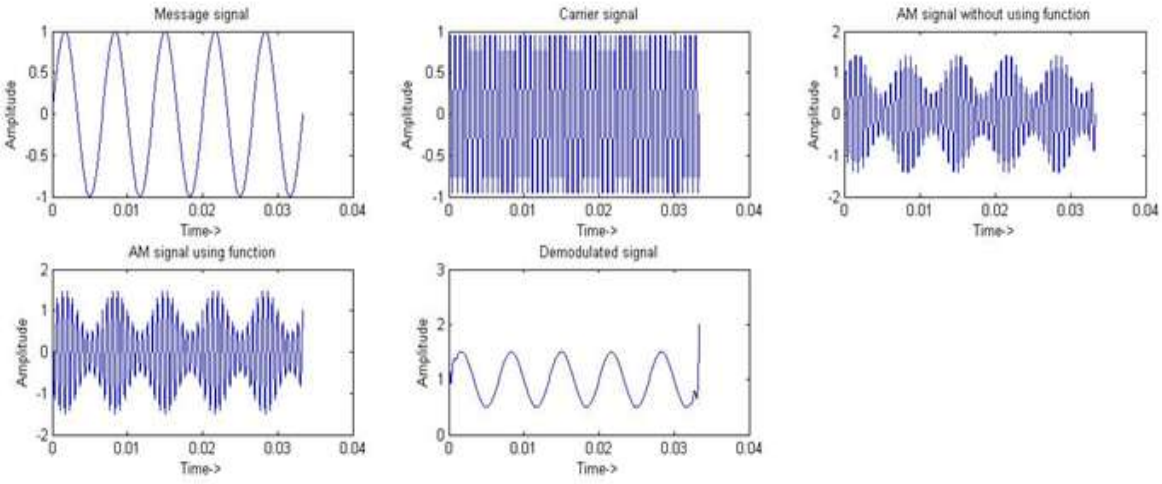
PWM does the modulation of the duty cycle of the pulse. In addition of use in communication systems, PWM is also used in voltage regulators such as Switched Mode Power Supplies (SMPS) to control the power delivered to load.PWM can be generated using a comparator to which the modulating signal and a reference ramp (sawtooth) waveform are fed.

INSTRUCTIONS

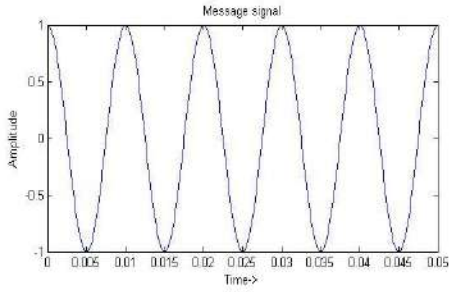
1. Interactively accept amplitude and frequencies of the message and carrier signal along with the modulation index. (Sinusoidal)
2. Implement amplitude modulation, Frequency modulation and PWM.
3. Plot all the input and output sequences on same time scale for each type of modulation. Make sure to plot 2 cycles of message signal.

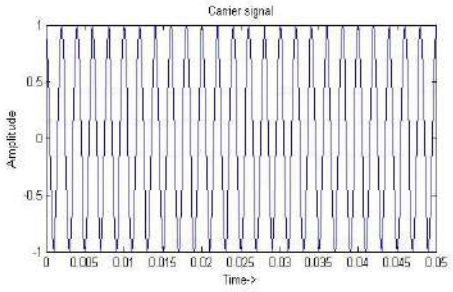
SAMPLES OF EXPECTED OUTPUT

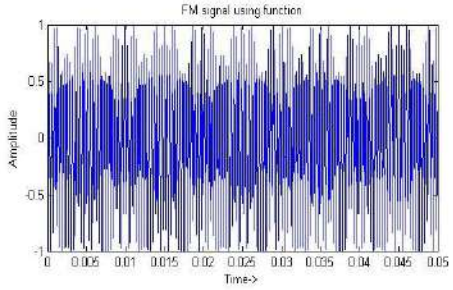
a) Amplitude modulation

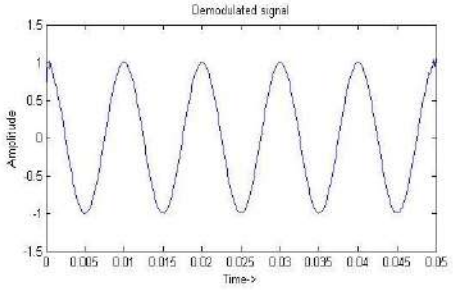


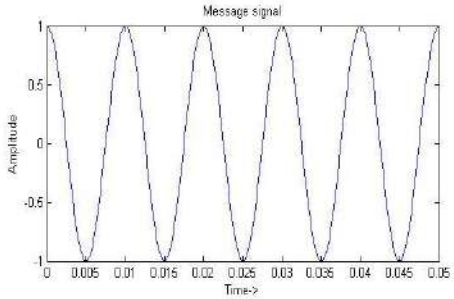
b) Frequency Modulation

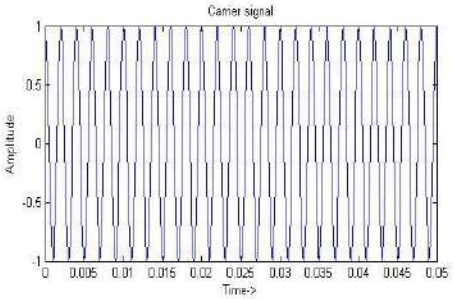

Message signal
Amplitude
Time->
1
0.5
0
-0.5
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05

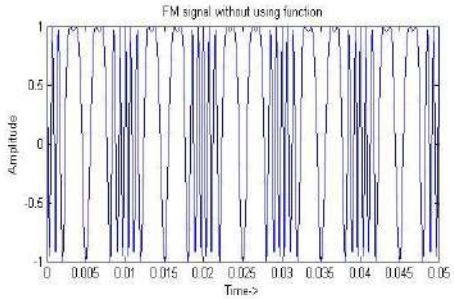

Carrier signal
Amplitude
Time->
1
0.5
0
-0.5
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05


FM signal using function
Amplitude
Time->
1
0.5
0
-0.5
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05

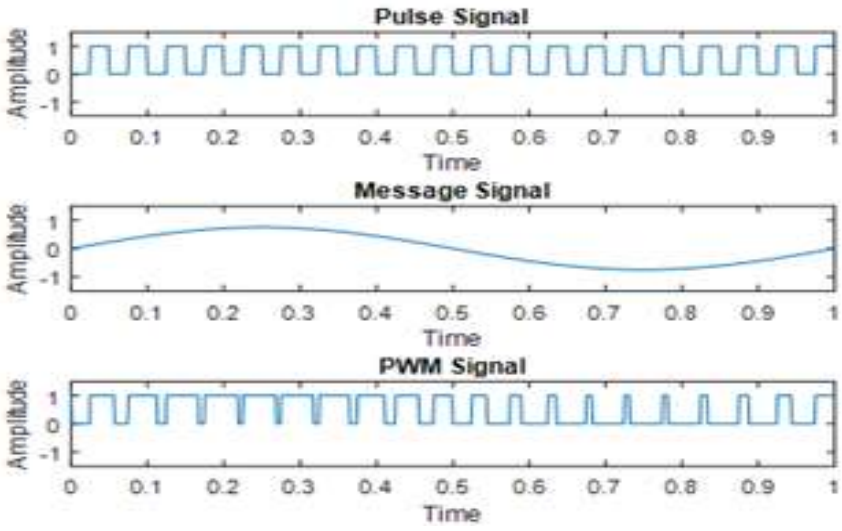

Demodulated signal
Amplitude
Time->
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05


Message signal
Amplitude
Time->
1
0.5
0
-0.5
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05


Carrier signal
Amplitude
Time->
1
0.5
0
-0.5
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05


FM signal without using function
Amplitude
Time->
1
0.5
0
-0.5
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05

c) PWM



RESULT

a) The amplitude modulated wave was created with and without using Matlab functions and was demodulated.

b) The frequency modulated wave was created with and without using Matlab functions and was demodulated.

c) The PWM wave was created.

EXPERIMENT NO 13

GENERATION OF DTMF SIGNAL

AIM

To generate a DTMF signal using matlab code.

THEORY

Dual-tone multi-frequency signaling (**DTMF**) is atelecommunication signalling system using the voice-frequency band over telephone lines betweentelephone equipment and other communications devices and switching centers. DTMF was first developed in the Bell System in the United States, and became known under the trademark **Touch-Tone** for use in push-button telephones supplied to telephone customers, starting in 1963.

The DTMF telephone keypad is laid out as a matrix of push buttons in which each row represents the low frequency component and each column represents the high frequency component of the DTMF signal. The commonly used keypad has four rows and three columns, but a fourth column is present for some applications. Pressing a key sends a combination of the row and column frequencies. For example, the *1* key produces a superimposition of a 697 Hz low tone and a 1209 Hz high tone. The tones are decoded by the switching center to determine the keys pressed by the user.

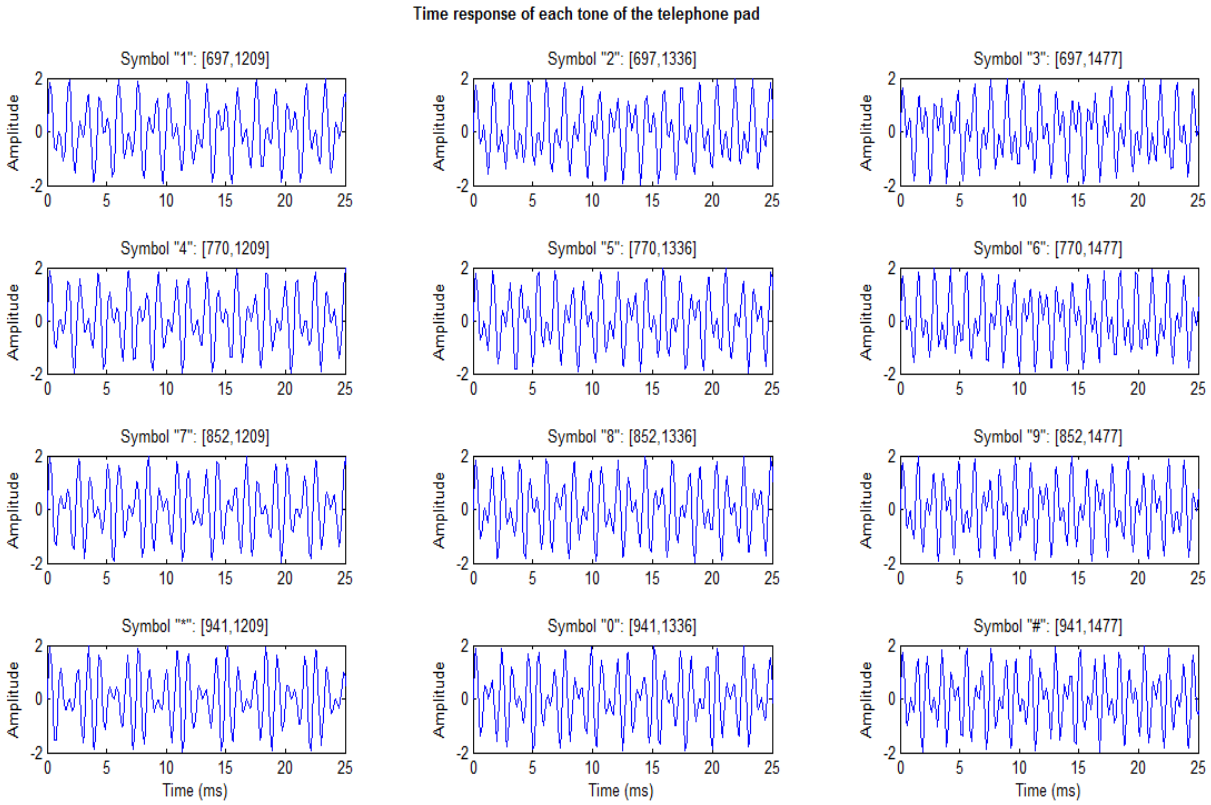
DTMF was originally decoded by tuned filter banks. Now,digital signal processing became the predominant technology for decoding. DTMF decoding algorithms typically use the Goertzel algorithm.

INSTRUCTIONS

1. Designate symbols ranging from 0,1,2---9, *,# for use
2. Designate values of frequencies to rows and columns
3. Depending on the choice of symbol, generate tones by combining appropriate values.

4. Display the tone frequencies

SAMPLE OF EXPECTED OUTPUT



RESULT

Generated a DTMF signal using matlab code.

EXPERIMENT NO 14

STUDY OF SAMPLING RATE CONVERSION

AIM

To study and simulate sampling rate conversion using matlab (decimation, interpolation & rational factor).

THEORY

The process of converting the sampling rate of a signal from one rate to another is called sampling rate conversion (SRC). This technique is encountered in many application areas such as:

- Digital audio
- Communication systems
- Speech processing
- Antenna systems
- Radar systems

Sampling rate may be changed upward or downward. Increasing the sampling rate is called interpolation and decreasing the sampling rate is called decimation. Reducing the sampling rate by a factor of M is achieved by discarding every M-1 samples or equivalently keeping every M^{th} sample.

Increasing the sampling rate by a factor of L(Interpolation by factor L) is achieved by inserting L-1 zeros into the output stream after every sample from the input stream of samples.

INSTRUCTIONS

1. Based on the frequency and amplitude choice of user, generate a discrete sinusoid and display at least 3 cycles
2. Accept upsampling and downsampling rate from user.
3. Apply these functions and display the effects of sampling rate conversion

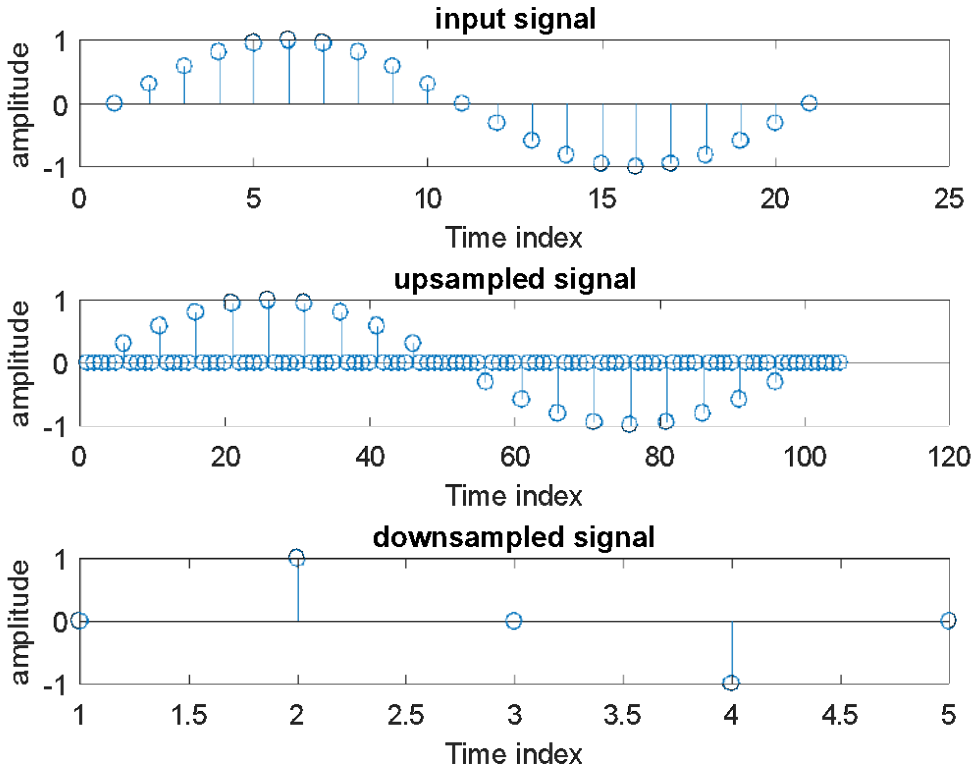
SAMPLES OF EXPECTED OUTPUT

Enter the signal frequency: 1

Enter the sampling rate: 20

Enter the upsampling rate: 5

Enter the downsampling rate: 5



RESULT

Studied sampling rate conversion using matlab (decimation, interpolation & rational factor).

EXPERIMENT NO 15

FILTERING OF NOISY SIGNALS

AIM

To filter noisy signal using matlab.

THEORY

Filtering is a class of signal processing, the defining feature of filters being the complete or partial suppression of some aspect of the signal. Most often, this means removing some frequencies or frequency bands. However, filters do not exclusively act in the frequency domain; especially in the field of image processing many other targets for filtering exist. Correlations can be removed for certain frequency components and not for others without having to act in the frequency domain.

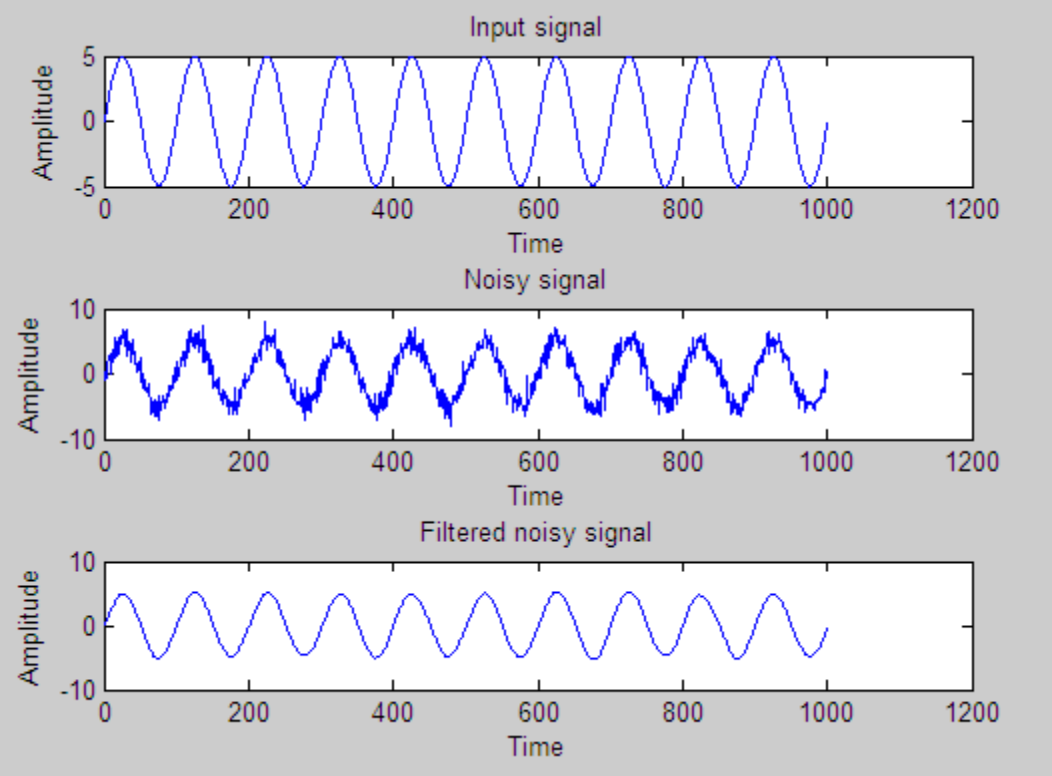
INSTRUCTIONS

1. Prepare a sinusoid continuous signal of users choice
2. Add AWGN noise of user specified SNR
3. Using a filter eg: hpfilter(), remove the noise and display the output

OUTPUT

Enter frequency : 10

Enter SNR : 0.1



RESULT

Filtered noisy signal using matlab.

EXPERIMENT NO 16

IMPLEMENTATION OF SIMPLE ALGORITHMS IN AUDIO PROCESSING

AIM

To implement the algorithms in audio signal processing for delay,reverb and flange.

THEORY

Delay Based Effects

Many useful audio effects can be implemented using a delay structure:

• Sounds reflected of walls – In a cave or large room we here an echo and also reverberation takes place – this is a different effect

If walls are closer together repeated reflections can appear as parallel boundaries and we hear a modification of sound colour instead.

• Vibrato, Flanging, Chorus and Echo are examples of delay effects

Basic Delay Structure

We build basic delay structures out of some very basic FIR and IIR filters

• We use FIR and IIR comb filters

• Combination of FIR and IIR gives the Universal Comb Filter

FIR Comb Filter

This simulates a single delay:

• The input signal is delayed by a given time duration, τ .

• The delayed (processed) signal is added to the input signal some amplitude gain, g

• The difference equation is simply:

$$y(n)=x(n)+gx(n-M) \text{ with } M=\tau/f_s$$

The transfer function is:

$$H(z)=1+gz^{-M}$$

IIR Comb Filter

This simulates a single delay:

• Simulates endless reflections at both ends of cylinder.

• We get an endless series of responses, y(n) to input, x(n).

• The input signal circulates in delay line (delay time τ) that is fed back to the input.

. • Each time it is fed back it is attenuated by g.

• Input sometime scaled by c to compensate for high amplification of the structure.

• The difference equation is simply:

$$y(n)=Cx(n)+gy(n-M) \qquad \text{with } M=\tau/f_s$$

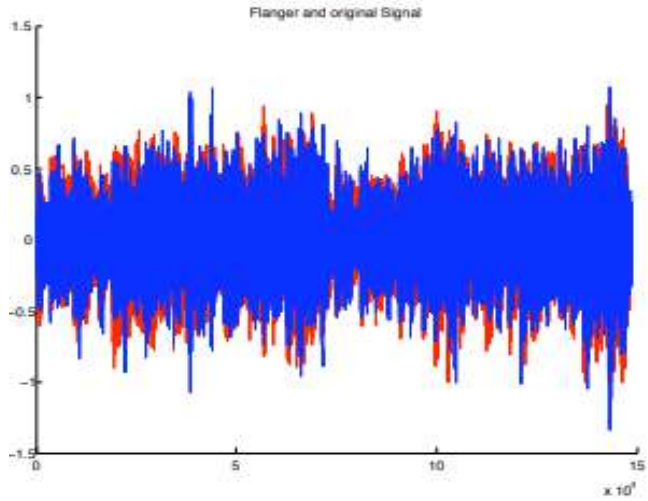
the transfer function is:

$$H(z)=c/(1-gz^{-M})$$

FLANGE

Flanging means continuously varying LFO of delay,

SAMPLES OF EXPECTED OUTPUT
(red plot is original audio)



Flanger and original signal

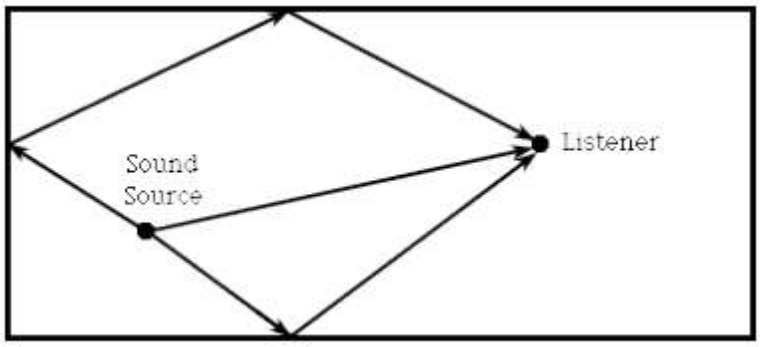
Reverb/Spatial Effects

The final set of effects we look at are effects that change to spatial localization of sound.

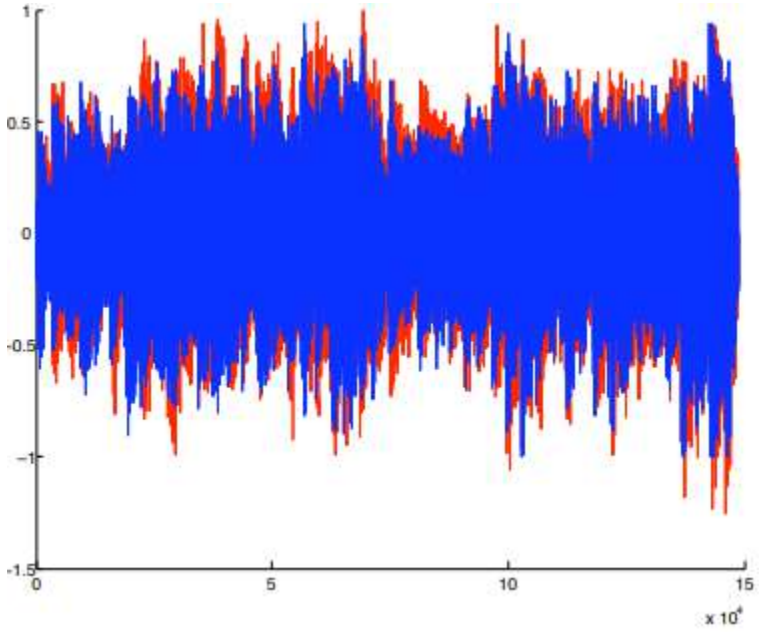
Reverberation (reverb for short) is probably one of the most heavily used effects in music. Reverberation is the result of the many reflections of a sound that occur in a room.

• From any sound source, say a speaker of your stereo, there is a direct path that the sounds covers to reach our ears.

• Sound waves can also take a slightly longer path by reflecting off a wall or the ceiling, before arriving at your ears.



Reverberation



Schroeder Reverberated Signal

RESULT

Implemented the algorithms in audio signal processing for delay, reverb and flange.

EXPERIMENT NO 17

IMPLEMENTATION OF SIMPLE ALGORITHMS IN IMAGE PROCESSING

AIM

To implement simple algorithms in image processing ie, filtering, denoising, detection.

THEORY

FILTERING:

The Canny edge detector is an edge detection operator that uses a multi- stage algorithm to detect a wide range of edges in images. It was developed by John F.
Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

a. Apply Gaussian filter to smooth the image in order to remove the noise
b. Find the intensity gradients of the image
c. Apply non-maximum suppression to get rid of spurious response to edge detection
d. Apply double threshold to determine potential edges
e. Track edges by hypothesis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

DENOISING:

Digital images are prone to various types of noise. Noise is the result of errors in the image acquisition process that result in pixel values that do not reflect the true intensities of the real scene. There are several ways that noise can be introduced into an image, depending on how the image is created. For example:

1. If the image is scanned from a photograph made on film, the film grain is a source of noise. Noise can also be the result of damage to the film, or be introduced by the scanner itself.

2. If the image is acquired directly in a digital format, the mechanism for gathering the data (such as a CCD detector) can introduce noise.
3. Electronic transmission of image data can introduce noise.

We can remove noise using various filtering techniques, for example, linear filtering, average filtering,median filtering.

DETECTION:

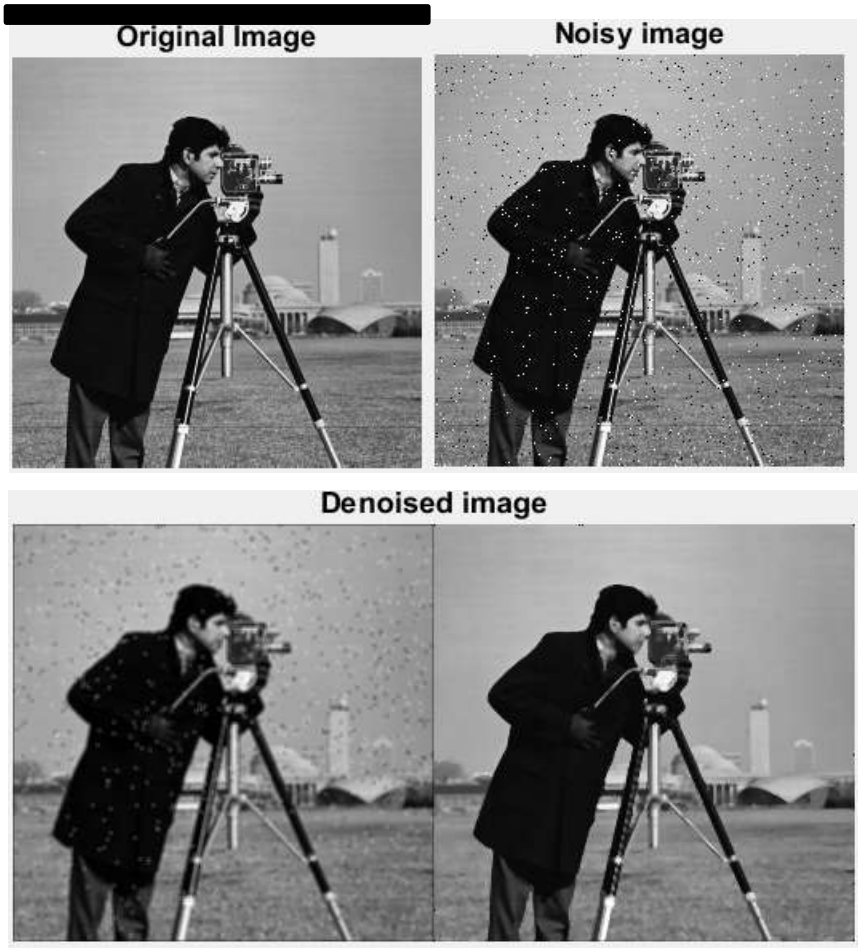
Filtering is a technique for modifying or enhancing an image. Mask or filters will be defined. The general process of convolution and correlation will be introduced via an example. Also smoothing linear filters such as box and weighted average filters will be introduced. In statistic and image processing, to smooth a data set is to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. Smoothing may be used in two important ways that can aid in data analysis by being able to extract more information from the data as long as the assumption of smoothing is reasonable by being able to provide analyses that are both flexible and robust. Different algorithms are used in smoothing.

INSTRUCTIONS

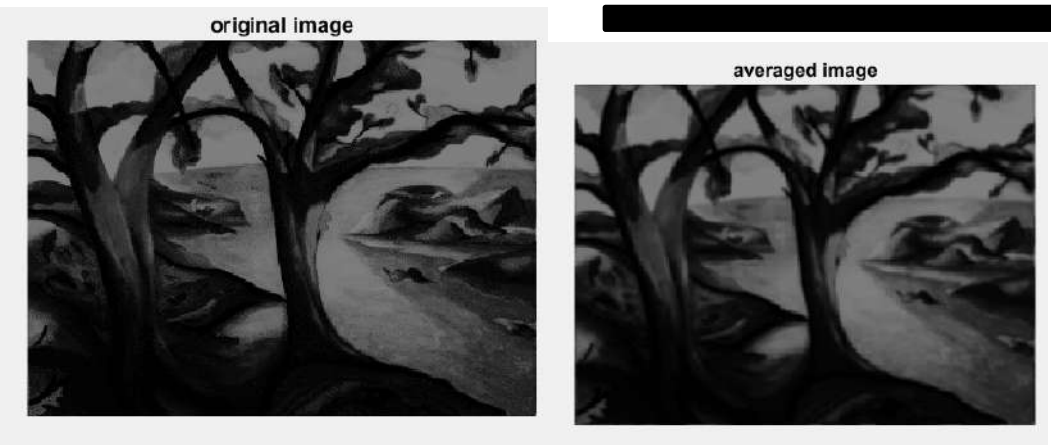
1. Using inbuilt image processing functions perform denoising, edge detection and filtering on Images

SAMPLES OF EXPECTED OUTPUT

DENOISING



FILTERING



DETECTION

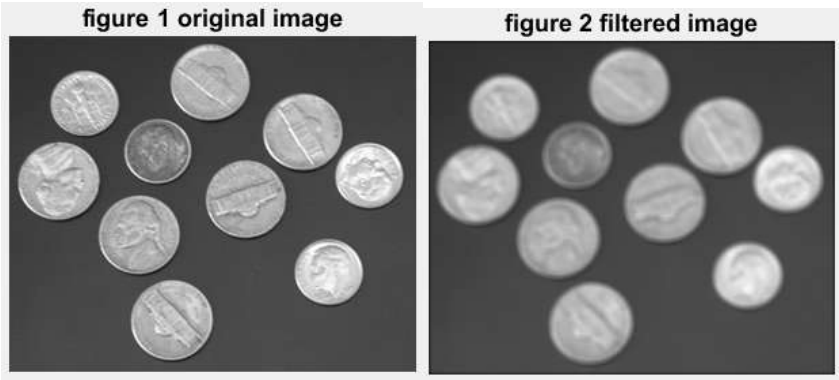


figure 3 edge detected output by sobel operator

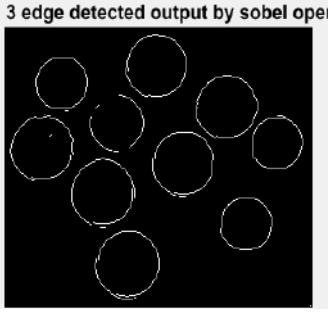


figure 4 edge detected output by prewitt operator

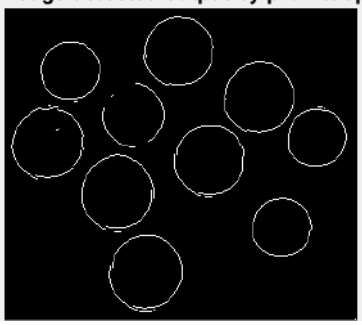


figure 5 edge detected output by robert operator

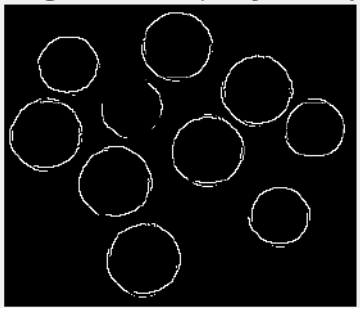
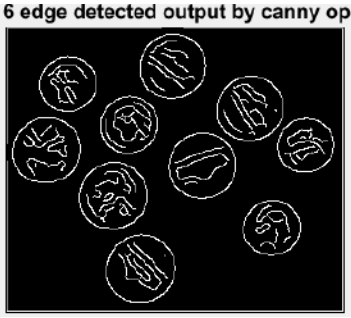


figure 6 edge detected output by canny operator



RESULT

Implemented simple algorithms in image signal processing ie., denoising, filtering and averaging.

Part B

EXPERIMENT NO 01

FAMILIARISATION OF TMS 320C6713 DIGITAL SIGNAL PROCESSOR

AIM

To study the architecture of TMS320C6713 Digital Signal Processor.

THEORY

DSK 6713 is a DSP starter kit based on TMS320C6713 DSP processor. It has 32 bit stereo code TLV 320AIC23 supporting built in A/D and D/A conversion. It has 16 MB external SD RAM. It has a JTAG interface for host computer interface.

FEATURES

The 6713 DSK board includes following hardware:

- C6713 DSP operating at 225 MHz.
- 4 Kbytes memory for L1D data cache.
- 4 Kbytes memory for L1P program cache.
- 256 Kbytes memory for L2 memory.
- 8 Mbytes of onboard SDRAM.
- 512 Kbytes of flash memory.
- 16-bit stereo code AIC23 with sampling frequency of 8KHz to 96KHz.

Belongs to C67XX Family (6701,6711,6713).

Floating point processor.

VLIW (Very Long Instruction Word) Architecture.

Eight 32-bit instruction can be fetched per cycle.

264 K internal memory.

8 KB L1 cache.

6 ALUs and 2 Multiplier units.

-32 bit address bus to address 4GB.

-Two sets of 32-bit general purpose registers.

-Two McBSPs (Multi channel Buffered Serial Ports).

Voltage Regulator

-1.25 V for TMS320C6713 DSP processor.

-3.3 V for memory.

-AIC23 codec using TLV320AIC23.

-1 Line IN and 1 Mic IN port.

-1 Line OUT and 1 Head Phone output.

-ADC and DAC using Sigma-Delta Technology.

-Sampling rate from 8,16,24,32,44,48 and 96 KHz.

PROCEDURE

The basic procedure for DSK713 is as follows.

1) To Setup dsp board

Connect DSP board to USB port

Turn ON power

Programs->Texas Instruments->Code composer Studio 3.1 ->

Setup Code composer Studio 3.1->Select C6713 DSK -> save and quit

2)To Create a new project

Eg: to create a New project myprog1 in *c:\mydsp directory*

1) Start Code Composer Studio

Programs->Texas Instruments->Code composer Studio 3.1 ->

Code composer Studio 3.1

2) Project -> New

Project name – myprog1(project name)

Location c:\mydsp

Project Type – Executable (*.out)

Target – TMS320C67xx

Click Finish

3)Write Your main dsp program

(Here myprog1.c) To do this

File->New->source file

4) Save the above file

File->Save As -> (save file in c:\mydsp\myprog1\myprog1.c)

The file name is projectname.c and should be saved in your

project directory - Here c:\mydsp\myprog1)

5) Add the above file to Your Project (Here myprog1)

project->Add files To Project->

(Select myprog1.c from c:\mydsp\myprog1\myprog1.c)

6) Include Libraries csl6713.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\csl\lib\ and select csl6713.lib

7) Include Libraries dsk6713bsl.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\dsk6713\lib\ and select dsk6713bsl.lib

8) Include Libraries rts6700.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\cgtools\lib\ and select rts6700.lib

9) Add supporting additional support files from Chassaing's Text Book

1) c6713dskinit.c

Project->Add Files to Project

Go to c:\dsk6713 and select c6713dskinit.c

2) Vectors_intr.asm

Project->Add Files to Project

Go to c:\dsk6713 and select Vectors_intr.asm

3) c6713dsk.cmd

Project->Add Files to Project

Go To c:\dsk6713 and select c6713dsk.cmd

10) To include additional dependent files

Project->Scan All File Dependencies

11) Set Compiler Options

Project->BuildOptions->Compiler

-g -q -fr".\Debug" -i"." -i"$(Install_dir)\c6000\dsk6713\include"

-d"_DEBUG" -d"CHIP_6713" -ml3 -mv6710

Click OK

12) To set linker options

Project->BuildOptions->linker

-c -m".\Debug\myprog1.map" -o".\Debug\myprog1.out" -w -x -i"$

(Install_dir)\c6000\dsk6713\lib"

Note: In above configuration change myprog1.map by your projectname.map

And myprog1.out by your projectname,out

Click Ok

13) Now Make Executable

Go Debug -> Build All

(If any error debug your source file)

14) Connect Your Host PC to DSP board

Debug-> Connect

15) Load Your Program

File->Load Program

Debug->Run->Free

16) To stop

Debug ->Stop

RESULT

Familiarized with TMS320C6713 DSP Board.

EXPERIMENT NO 02

GENERATION OF SINE WAVE AND STANDARD TEST SIGNALS

AIM

To generate sine wave and standard test signals using DSK6713.

PROCEDURE

The procedure is same for all the experiments, only the source code changes for the wave generation.

1) To Setup dsp board

Connect DSP board to USB port

Turn ON power

Programs->Texas Instruments->Code composer Studio 3.1 ->

Setup Code composer Studio 3.1->Select C6713 DSK -> save and quit

2) To Create a new project

Eg: to create a New project myprog1 in *c:\mydsp directory*

1) Start Code Composer Studio

Programs->Texas Instruments->Code composer Studio 3.1 ->

Code composer Studio 3.1

2) Project -> New

Project name – myprog1(project name)

Location c:\mydsp

Project Type – Executable (*.out)

Target – TMS320C67xx

Click Finish

3)Write Your main dsp program

(Here myprog1.c) To do this

File->New->source file

4) Save the above file

File->Save As -> (save file in c:\mydsp\myprog1\myprog1.c)

The file name is projectname.c and should be saved in your

project directory - Here c:\mydsp\myprog1)

5) Add the above file to Your Project (Here myprog1)

project->Add files To Project->

(Select myprog1.c from c:\mydsp\myprog1\myprog1.c)

6) Include Libraries csl6713.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\csl\lib\ and select csl6713.lib

7) Include Libraries dsk6713bsl.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\dsk6713\lib\ and select dsk6713bsl.lib

8) Include Libraries rts6700.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\cgtools\lib\ and select rts6700.lib

9) Add supporting additional support files from Chassaing's Text Book

1) c6713dskinit.c

Project->Add Files to Project

Go to c:\dsk6713 and select c6713dskinit.c

2) Vectors_intr.asm

Project->Add Files to Project

Go to c:\dsk6713 and select Vectors_intr.asm

3) c6713dsk.cmd

Project->Add Files to Project

Go To c:\dsk6713 and select c6713dsk.cmd

10) To include additional dependent files

Project->Scan All File Dependencies

11) Set Compiler Options

Project->BuildOptions->Compiler

-g -q -fr".\Debug" -i"." -i"$(Install_dir)\c6000\dsk6713\include"

-d"_DEBUG" -d"CHIP_6713" -ml3 -mv6710

Click OK

12) To set linker options

Project->BuildOptions->linker

-c -m".\Debug\myprog1.map" -o".\Debug\myprog1.out" -w -x -i"$

(Install_dir)\c6000\dsk6713\lib"

Note: In above configuration change myprog1.map by your projectname.map

And myprog1.out by your projectname,out

Click Ok

13) Now Make Executable

Go Debug -> Build All

(If any error debug your source file)

14) Connect Your Host PC to DSP board

Debug-> Connect

15) Load Your Program

File->Load Program

Debug->Run->Free

16) To stop

Debug ->Stop

RESULT

Generated sine wave and standard test signals using DSK6713

EXPERIMENT NO 03

CONVOLUTION: LINEAR AND CIRCULAR

AIM

To perform linear and circular convolution using DSP trainer kit.

PROCEDURE

The procedure is same as given in the experiment no.01. Write C language code for linear and circular convolution and load it to the kit as explained in the procedure..

RESULT

Performed Linear and Circular convolution using DSP processor

EXPERIMENT NO 04

REAL TIME FIR FILTER

AIM

To implement a Real Time FIR Filterby inputting a signal from the signal generator.

PROCEDURE

1) Create a new Project eg; fir.pjt

2) Write Your main dspprogramfir.c

3) Use fdatool of Matlab to get filter coefficients eg: lpcof.h

Obtain order of the filter M and coefficients using Matlab.

Start Matlab and on command prompt type fdatool (filter design and analysis tool)

- fdatool

(Give Specification and design the required filter)

Export Filter Coefficients From fdatool For this

Targets -> Generate C Header ->

Numerator H

Numerator Length N

Export as signed -16 bit Number

>Export to your project directory (c:\mydsp\firlpf\lpcof.h)

Quit Matlab

4) Edit lpcof.h

Remove #include “tmtypes.h”

Replace constint N by #define N

Replace const int_16T by short

Save File as lpcof.cof in your project

/** Typical format of lpcof.h **/

#define N 5

h[N]={100, -200, 100, 300, 400 } ;

5) Include support files

6) compile load and run

7) feed input signal from function generator to line in port of DSK

8) Observe output from line out port of DSK using C.R.O

RESULT

Implemented a Real Time FIR filter using DSP kit.

EXPERIMENT NO 05

REAL TIME IIR FILTER

AIM

To implement a Real Time IIR Filterby inputting a signal from the signal generator.

PROCEDURE

1) Create project iir in mydsp folder

2) Write iir.c and add to project

3) Obtain cof.h using FDATOOL of Matlab

4) Copy cof.h to Project directory eg: c:\\mydsp\\iir

Edit cof.h as

/* remove first 32768 of denominator coefficients **/

/* Eg: cof.h **/

```
#define stages 5            //number of 2nd-order stages
 int a[stages][3]=     {    //numerator coefficients
 {1, -3, 1},             //a10,a11,a12 for 1st stage
{32768, 63298, 32768},       //a20,a21,a22 for 2nd stage
{32768, -51261, 32768},      //a30,a31,a32 for 3rd stage
{32768, 51261, 32768},       //.....
{32768, 20322, 32768}    };   //a50,a51,a52 for 5th stage
int b[stages][2]=  {       //denominator coefficients
```

{-3196, 13135}, //b11,b12 for 1st stage

{3196, 13135}, //b21,b22 for 2nd stage

{16350, 31943},

{600,500}, {600,400}

}; //b181,b182 for 18th stage

RESULT

Implemented a Real Time IIR filter using DSP kit.

EXPERIMENT NO 06

SAMPLING OF ANALOG SIGNAL AND STUDY OF ALIASING

AIM

To sample analog signal and to study aliasing using DSP kit.

PROCEDURE

1) To Setup dsp board

Connect DSP board to USB port

Turn ON power

Programs->Texas Instruments->Code composer Studio 3.1 ->

Setup Code composer Studio 3.1->Select C6713 DSK -> save and quit

2) To Create a new project

Eg: to create a New project myprog1 in *c:\mydsp directory*

1) Start Code Composer Studio

Programs->Texas Instruments->Code composer Studio 3.1 ->

Code composer Studio 3.1

2) Project -> New

Project name – myprog1(project name)

Location c:\mydsp

Project Type – Executable (*.out)

Target – TMS320C67xx

Click Finish

3)Write Your main dsp program

(Here myprog1.c) To do this

File->New->source file

(To observe the effect of aliasing introduced due to choice of incorrect sampling frequency may be demonstrated using a lower value of the same).

4) Save the above file

File->Save As -> (save file in c:\mydsp\myprog1\myprog1.c)

The file name is projectname.c and should be saved in your

project directory - Here c:\mydsp\myprog1)

5) Add the above file to Your Project (Here myprog1)

project->Add files To Project->

(Select myprog1.c from c:\mydsp\myprog1\myprog1.c)

6) Include Libraries csl6713.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\csl\lib\ and select csl6713.lib

7) Include Libraries dsk6713bsl.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\dsk6713\lib\ and select dsk6713bsl.lib

8) Include Libraries rts6700.lib

Project->Add Files to Project

Go to C:\CCStudio_v3.1\C6000\cgtools\lib\ and select rts6700.lib

9) Add supporting additional support files from Chassaing's Text Book

1) c6713dskinit.c

Project->Add Files to Project

Go to c:\dsk6713 and select c6713dskinit.c

2) Vectors_intr.asm

Project->Add Files to Project

Go to c:\dsk6713 and select Vectors_intr.asm

3) c6713dsk.cmd

Project->Add Files to Project

Go To c:\dsk6713 and select c6713dsk.cmd

10) To include additional dependent files

Project->Scan All File Dependencies

11) Set Compiler Options

Project->BuildOptions->Compiler

-g -q -fr".\Debug" -i"." -i"$(Install_dir)\c6000\dsk6713\include"

-d"_DEBUG" -d"CHIP_6713" -ml3 -mv6710

Click OK

12) To set linker options

Project->BuildOptions->linker

-c -m".\Debug\myprog1.map" -o".\Debug\myprog1.out" -w -x -i"$

(Install_dir)\c6000\dsk6713\lib"

Note: In above configuration change myprog1.map by your projectname.map

And myprog1.out by your projectname,out

Click Ok

13) Now Make Executable

Go Debug -> Build All

(If any error debug your source file)

14) Connect Your Host PC to DSP board

Debug-> Connect

15) Load Your Program

File->Load Program

Debug->Run->Free

16) To stop

Debug ->Stop

RESULT

Analog signal sampling is done and studied aliasing using DSP kit.