DIGITAL SIGNAL PROCESSING LAB MANUAL

Subject Code: ECE 3161

For

5th Semester B. Tech. ECE

DEPARTMENT OF
ELECTRONICS & COMMUNICATION ENGINNEERING

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Prepared by **Approved by**

Staff, ECE Department, M.I.T., Manipal.

Date: 25-07-2022

List of Experiments:

Instructions to the Students:

1. All the students are required to come prepared for the experiments to be done in the lab.
2. Students should try to analyze and understand the solved problems and then try to solve the unsolved problems of the experiment in the lab.
3. Maintaining an observation copy is compulsory for all, where the programs (codes) and results of the unsolved-problems (exercise) are to be noted down.
4. Students have to get their results verified and observation copies checked by the instructor before leaving the lab.
5. Maintain a separate folder in the computer you use, where you save all the programs you do in the lab.
6. Use of external storage media during lab is not allowed.
7. Maintain the timings and the discipline of the lab.
8. Students will be evaluated in every lab based on individual performance, observation copy and involvement in the lab.

Evaluation plan

- In-semester Assessment Marks : 60% (60 Marks)
  - ✓ Continuous evaluation component (lab sessions 1 to 6): 10 marks each.
  - ✓ Assessment is based on conduction of each experiment, exercise problems, answering the questions related to the experiment.
- End semester assessment: 40 % (40 marks)
  - ✓ write up (coding, analysis): 20 marks
  - ✓ Concept/Theory: 10 marks
  - ✓ Viva-Voce: 10 marks

Experiment No. 1: Introduction to Signal processing using MATLAB

MATLAB (**MAT**rix **LAB**oratory) is an interactive program for scientific and engineering numeric calculation, which has application in almost all the areas. MATLAB is one of the few languages in which each variable is a matrix. Moreover, the fundamental operators (e.g. addition, multiplication) are programmed to deal with matrices when required. In the MATLAB environment, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. Unlike languages such as C and C++, there is no much syntax in the MATLAB. Anyone having knowledge of simple programming language can work with the MATLAB. So learning MATLAB programming is made easier using certain applications. MATLAB is very rich with toolboxes, which consist of various functions depending on the applications. These functions can be used as a command to perform particular operation. Let us begin with starting MATLAB from the desktop. You can find a MATLAB icon on the desktop or from the start menu. By double clicking the icon you can invoke the MATLAB command window, which is shown in the figure 1.1

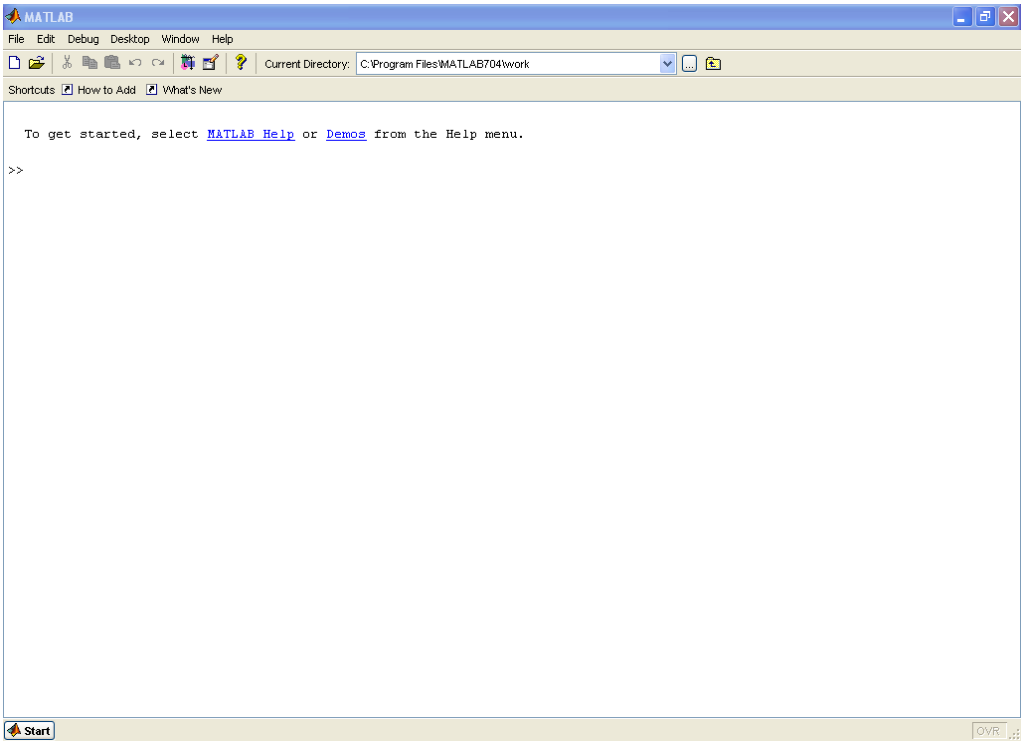


Figure 1.2: MATLAB Command Window

Command window consists of prompt for entering the command for the execution of the particular function. Suppose we want to create a variable with a vector we can enter the following at the command prompt and press ENTER key.

>> a = [1 2 3 4 5]

a =

1 2 3 4 5

This will create a vector with variable name ‘a’. If the command which is entered at prompt is terminated

by the semicolon, then the values corresponding to the variable is not shown in the command window when you press the ENTER key. Similarly, the command, which does the particular function, can be entered at the prompt, which is shown as follows.

>> **a=ones (2, 2)**

a =

1 1

1 1

The above shown command will create a matrix of size 2x2 with all ones. There are many commands like this are available in the MATLAB, which are given in various tool boxes. In this lab students will get acquainted with the usage of these commands using some examples. The function rand will generate uniformly distributed random elements and *randn* will generate normally distributed random elements.

R = randn(4,4)

R =

0.6353 0.0860 -0.3210 -1.2316

-0.6014 -2.0046 1.2366 1.0556

0.5512 -0.4931 -0.6313 -0.1132

-1.0998 0.4620 -2.3252 0.3792

The data set can be saved in a file such with an extension of **.mat, .dat** and **.txt** using save command and same can be retrieved using load function. The load function reads binary files containing matrices generated by earlier MATLAB sessions, or reads text files containing numeric data. The text file should be organized as a rectangular table of numbers, separated by blanks, with one row per line, and an equal number of elements in each row. For example, outside of MATLAB, create a text file containing these four lines:

16.0 3.0 2.0 13.0

5.0 10.0 11.0 8.0

9.0 6.0 7.0 12.0

4.0 15.0 14.0 1.0

Save the file as myfile.dat in the current directory. The statement load myfile.dat reads the file and creates a variable, containing the example matrix.

M-Files

You can create your own matrices using M-files, which are text files containing MATLAB code. Use the MATLAB Editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends in .m.

Let us learn this by writing a simple program for generating a sine wave and plotting it using plot command.

In **File** menu select the **New** option in and then select **M-file.**

Matlab editor window will open. Then enter the following program.

```
clc;                % CLEARS THE COMMAND WINDOW
clear all;          % CLEARS ALL VARIABLE
close all;          % ALL THE EXISTING FIGURE WINDOWS WILL BE CLOSED
Fs = 1000;          % SAMPLING FREQUENCY
f = 200;            % SIGNAL FREQUENCY
t = 0:1/Fs:(2/f);       % GENERATION OF THE TIME SEQUENCE FOR SINEWAVE
x = 5*sin(2*pi*f*t);    % SINE WAVE GENERATION USING SINE FUNCTION
plot(t,x);          % THE SINE WAVE IS DISPLAYED IN A SEPERATE PLOT WINDOW
```

The above program and editor window is shown in figure 1.2

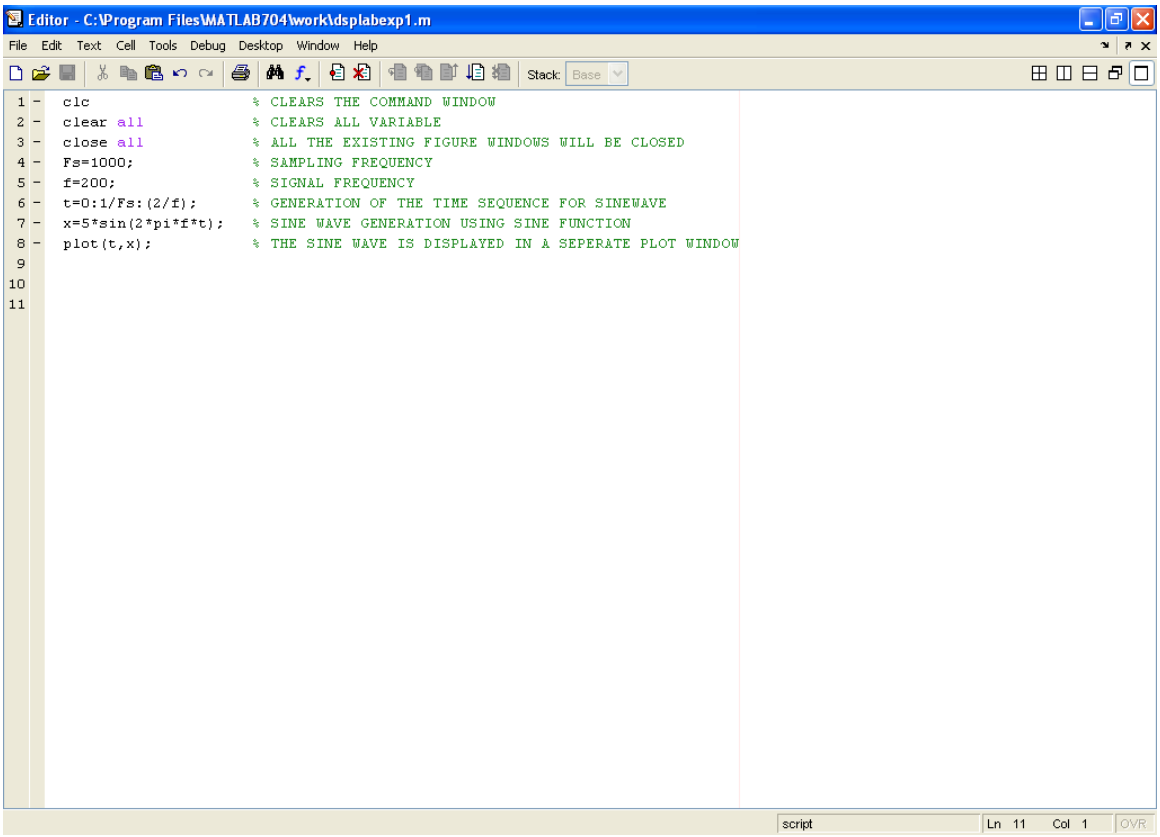


Figure Figure 1. 1 Matlab editor window with a program

Save the program from save menu with any meaningful file name. ***Strictly don't use file name, which is already a MATLAB command***. This will overwrite the existing command file which is a built in function. The program can be executed by selecting the **Run** option from the **Debug** menu. The output is plotted in separate window which is shown in figure 1.3

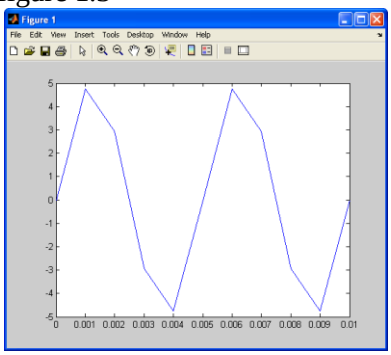


Figure 1.2 Plot of the execution (sine wave)

Always use commands such as *clc*, *close al*l and *clear all*, before starting any program in the MATLAB editor.

Help for the MATLAB commands are available from the help menu or by typing help followed by the command name in the command window.

Important Note: Students can refer to the tutorial documents available at *c:\dsplab\Matlab\gettingstartedMATLAB7.pdf*. This tutorial gives all the information about the MATLAB usage.

Activity Questions

Students must complete the execution of activity questions in the lab and write down the programs or answers in the observation book.

1. Study the following command function through the help menu and check it in the command window with suitable illustrations:

i) fliplr	vii) plot	xiii) disp	xix) angle
ii) zeros	viii) title	xiv) eye	xx) abs
iii) linspace	ix) subplot	xv) num2str	xxi) log
iv) sin	x) who	xvi) floor	xxii) double
v) cos	xi) whos	xvii) sqrt	xxiii) save
vi) figure	xii) input	xviii) exp	xxiv) load

2. Assume array c is defined as shown, and determine the contents of the following sub-arrays.

$$c = \begin{bmatrix} 1.1 & -3.2 & 3.4 & 0.6 \\ 0.6 & 1.2 & -0.6 & 3.1 \\ 1.3 & 0.6 & 5.5 & 0.0 \end{bmatrix}$$

i) c(2,3)	iv) c(1:2,2:end)	vii) c(1:2,2:4)
ii) c(2,:)	v) c(6)	viii) c([1 3],2)
iii) c(:,end)	vi) c(4:end)	ix) c([2 2],[3 3])

3. Assume a, b, c and d are defined as follows

$$a = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}, b = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix}, c = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, d = 5$$

What is the result of the following expressions?

i) a+b	iv) a .* b	vii) a.*c
ii) a+c	v) a * b	viii) a .* d
iii) a+d	vi) a * c	ix) a * d

4. Evaluate the following MATLAB expressions yourself before checking the answers in MATLAB:

i) 11 / 5 + 6
ii) (11 / 5) + 6
iii) 11 / (5 + 6)
iv) round(-11 / 5) + 6
v) ceil(-11 / 5) + 6
vi) floor(-11 / 5) + 6
vii) 3 ^ 2 ^ 3
viii) 3 ^ (2 ^ 3)
ix) (3 ^ 2) ^ 3

5. Given $a = 3$, $b = 2$, $c = 5$, and $d = 3$. Use MATLAB to evaluate the following:

i) a * b + c * d

ii) a*(b+c)*d

iii) (a*b)+(c*d)

Experiment No. 2: Analysis of signals in time domain

Objective:

To study signals and systems in time-domain and understand the concepts of sampling, convolution, correlation and impulse response.

Signal generation and sampling

The signals we use in the real world (such as voice signal) are "analog" signals. To process these signals in computing devices, we need to convert them in to "digital" form. While an analog signal is continuous in both time and amplitude, a digital signal is discrete in both time and amplitude. To convert a signal from continuous time to discrete time, sampling is used. The value of the signal is taken at certain intervals in time to get discrete samples of the signal.

The Sampling Theorem states that a signal can be exactly reproduced if it is sampled at a frequency F_s, where F_s is equal or greater than twice the maximum frequency in the signal (Nyquist rate). The sampling frequency or sampling rate, F_s, is the average number of samples obtained in one second (*samples per second*), thus $F_s = 1/T$. As the sampling frequency decreases, the signal separation also decreases. When the sampling frequency drops below the Nyquist rate, the frequencies will crossover and cause aliasing. In MATLAB signals can be simulated using functions where the duration and the sampling time can be specified.

Example-1: Generate and plot the following signals

a) Unit-step function
b) Delayed step function with a delay of $n_0 = 5$.
c) Step function with amplitude m= 5
d) Rectangular window of length 5;
e) Ramp signal

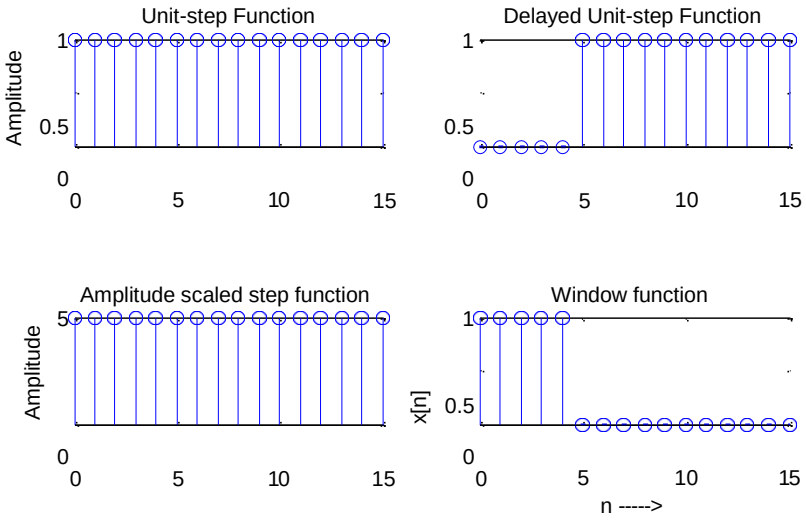
Solution:

```
clc;                % clears the command window
clear all;          % clear all variables
close all;          % all the existing figure windows will be closed
n = 0:15;           % time index for the discrete signal from 0 to 15 with
unity sampling
                    % time
u = [(n)>=0];       % Unit Step
u5 = [(n-5)>=0];    % Delayed step
a5 = 5* u;          % Scaled unit step
xn = u-u5;          % rectangular window x[n]= u[n]- u[n-5]
xr = n.*u;          % Ramp signal

% v(1:16) = ones(1,16) also defines the unit step vector v of length 16 or a
rectangular
%window of size 16.

subplot(321), stem(n,u), title('Unit-step Function'); ylabel('Amplitude ');
subplot(322), stem(n,u5), title('Delayed Unit-step Function');
subplot(323), stem(n,a5), title('Amplitude scaled step function
');ylabel('Amplitude');
subplot(324), stem(n,xn), title('Window function ');ylabel('x[n]');xlabel('n --
--->')
subplot(325), stem(n,xr), title('Ramp
Function');ylabel('Amplitude');xlabel('n ---- >');
```

Output:


Unit-step Function
Amplitude
1
0.5
0
0
5
10
15
Delayed Unit-step Function
1
0.5
0
0
5
10
15
Amplitude scaled step function
Amplitude
5
0
0
5
10
15
Window function
x[n]
1
0.5
0
0
5
10
15
n ----->
0

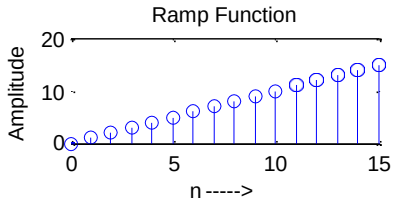


Figure 2.1 Results of example-1

Example-2: MATLAB code to generate sine wave at different sampling frequencies.

```
clc;
clear all;
close all;
Fs = 1000;                % sampling frequency
f = 200;                  % signal frequency
T = 0:1/Fs:(1/f);         % generation of the time sequence at sampling
                          % frequency fs for 1 period duration.
x = 5*sin(2*pi*f*t);      %sine wave of freq. Of f hz and amplitude 5;
sampled at fs
subplot(3,1,1); stem(t,x);
title('Sine wave sampled at 1000 Hz 1 period duration'); ylabel('Amplitude')

Fs1 =10000;               % Different sampling frequency
f = 200;
t = 0:1/Fs1:(1/f);
x = 5*sin(2*pi*f*t);
subplot(3,1,2);stem(t,x);
title('Sine wave sampled at 10000 Hz for 1 period duration ');
ylabel('Amplitude')

Fs1 =10000;               % another sampling frequency
T = 0:1/Fs1:(2/f);        % generation of the time sequence at sampling
frequency fs
                          %for 2 cycle duration
x = 5*sin(2*pi*f*t);
subplot(3,1,3);
```

```
stem(t,x);
title('Sine wave sampled at 10000 Hz for 2 cycle duration');
ylabel('Amplitude')
xlabel('n----->');
```

Output:

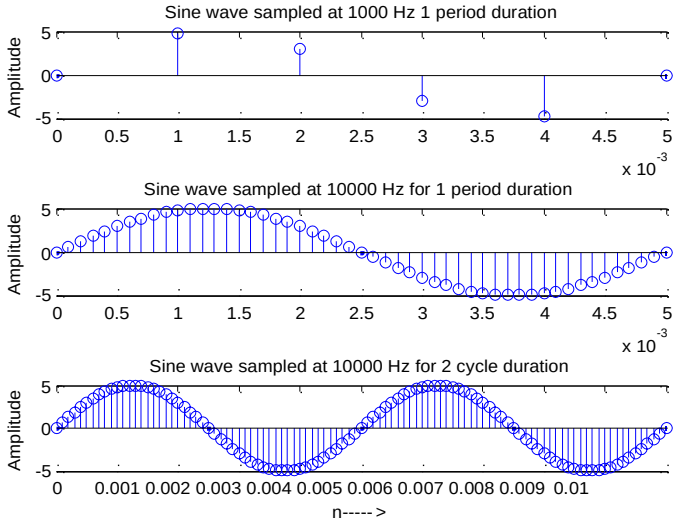


Figure 2.2 Results of example-2

Example-3: Write a MATLAB program to find and plot the convolution between the two signals x = u - u5 and y = [1 2 3 4 5] where u is unit step function in the range 0:15.

Solution:

Create the signals x (Refer example-1) and y.

Use conv(x, y) MATLAB function to perform the convolution and plot the signals

Example-4: Given a discrete-time sequence $x[n] = \{3, 11, 7, 0, -1, 4, 2\}$ and $y[n] = x[n-2] + w[n]$, where $w[n]$ is a Gaussian sequence with zero mean and unit variance. Compute the cross- correlation between $x[n]$ and $y[n]$.

Solution:

```
clc;
close all;
clear all;
x = [3,11,7,0,-1,4,2];                  % Given sequence x[n]
                ↑
nx = -3:3;
ny = nx + 2;                            % Shifted x[n] ; delay of 2
units
y1 = x;
w = randn(1, length(y1));               % Random sequence
generation
y = y1 + w;
nyb = ny(1)+nx(1);                       % Starting point
nye = ny(length(y1))+nx(length(x));  % Ending point
ny = nyb:nye;
xcr = xcorr(x,y); % cross-correlation
plot(ny,xcr);
title('Cross-correlation example');
xlabel('Time, n');
```

Output:

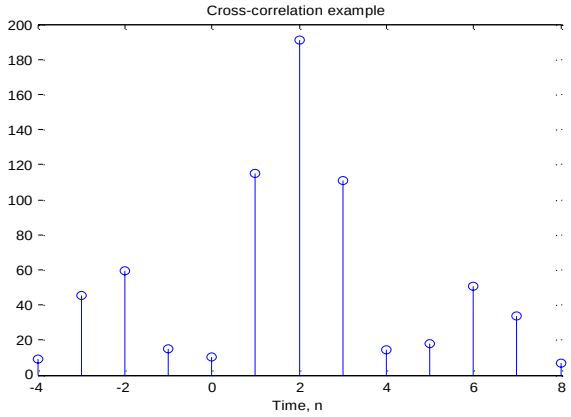


Figure 2.3 Result of example-4

Note: The signal x and its delayed signal y have maximum correlation at time n = 2. Hence, the correlation plot shows a maximum peak at n = 2. Such signal processing is used in Radar applications.

Example-5: Determine the UNIT impulse response h[n] of a system described by the difference equation $y[n] - 0.6y[n-1] - 0.16y[n-2] = 5x[n]$

Solution:

```
clc;
close all;
clear all;
N = 20;
b = [5 0 0];
a = [1 -0.6 -0.16];
f = [1, zeros(1,N-1)]; % generation of unit impulse δ[n]
h = filter(b, a, f); % input to the system is δ[n]; output is impulse
response h[n]
n = 0: 1: N-1;
```

```
stem(n, h);
title('Unit Impulse Response, h[n]'), xlabel('n'), ylabel('Amplitude');
```

Output:

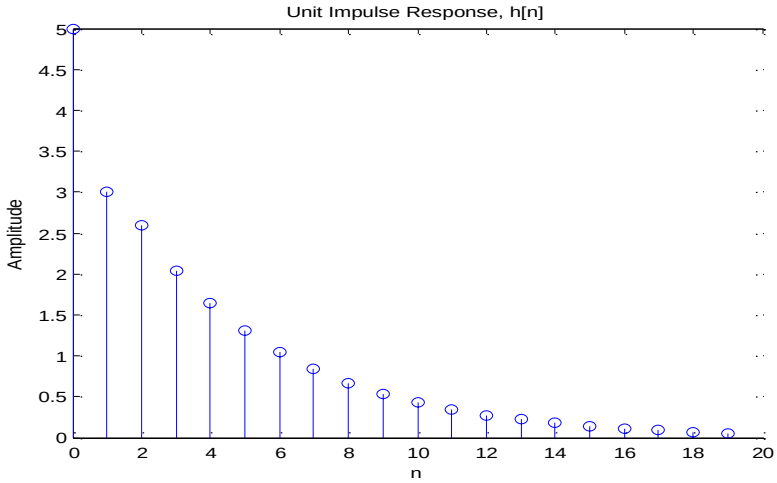


Figure 2.4 Result of example-5

Note: Refer MATLAB help for details of *filter* function

Exercise:

1. Generate the following signals
   a) $(-1/2)^k$, $(2)^k$ and $(-2)^k$, assuming k = 0 to 15
   b) Square wave with duty cycle 25%, 50% and 75%. (Use *square* function)
   c) Triangular wave. (Use *sawtooth* function)
   d) A multi-tone signal having the frequencies 10Hz, 30Hz and 60Hz.

2. Write a MATLAB program to sketch the following discrete-time signals in the time range of -10 ≤ n ≤ 10. If the sequence is complex, plot the magnitude and angle separately.

   a) x[n] =u [n] − u[n − 3]

   b) x[n] = sin ($\frac{n\pi}{3}$) u[n]

   c)x [n] = $0.5^{n}e^{j\pi n/2}$

   Note: Study the difference between plot and stem functions of MATLAB

3. Find the system output y[n]; 0 ≤ n ≤ 10 of an LTI system with impulse response

   h[n] = $(0.5)^{n}$\{u[n] – u[n – 10]) when the inputs are

   a) x[n] = $(0.8)^{n}$(u[n]– u[n − 5])

   b) x[n] = δ[n] + 3δ[n – 1] + 4δ[n – 3]

4. A simple digital differentiator is given by y[n] = x[n] − x[n − 1]. Implement this differentiator and test with the following sequences. Plot the results and comment on the appropriateness of this differentiator.

   a) x[n] = 5(u[n] − u[n − 20])

   b) x[n] = (u[n] − u[n − 10]) + (20 − n)(u[n − 10] − u[n − 20])

   c) x[n] = sin($\frac{\pi n}{25}$)(u[n] − u[n − 100]))

5. Write a MATLAB program to generate a pure sine wave of frequency 80Hz and add additive white Gaussian noise with SNR of 10dB. (Use the function *awgn.*) Find and plot the autocorrelation of the resulting sequence.

Experiment No. 3: Analysis in frequency domain

Objectives:
To study the frequency-domain representation of signals and systems.

Theory: The Discrete Time Fourier Transform $H(e^{j\omega})$ of the impulse response is called the frequency response.

For infinite-duration impulse response (IIR) system

$$H(e^{j\omega}) = \frac{B(e^{j\omega})}{A(e^{j\omega})} = \frac{b_0 + b_1 e^{-j\omega} + b_2 e^{-j\omega} \ldots}{1 + a_0 + a_1 e^{-j\omega} + a_2 e^{-j\omega} \ldots}$$

For finite-duration impulse response (FIR) systems

$$H(e^{j\omega}) = b_0 + b_1 e^{-j\omega} + b_2 e^{-j\omega} \ldots$$

Example-1: Find Discrete-time Fourier transform (DTFT) of the given signal and plot the magnitude and the phase.

$$h[n] = 0.5^n u[n]$$

$$H(e^{j\omega}) = \frac{1}{1 - 0.5e^{-j\omega}} = \frac{e^{j\omega}}{e^{j\omega} - 0.5}$$

```
clc;
clear all;
close all;
w = [0:1:500]*pi/500;          % defining 501 values of radian frequency ω
in the range 0 to π
x = exp(j*w)./(exp(j*w)-0.5);
subplot(121)
plot(w, abs(x)), title('Magnitude response'), xlabel('w'), ylabel('Amplitude');

subplot(122), plot(w,angle(x));
title('Phase Response'), xlabel('w(radians)'), ylabel('Angle');
```

Output:

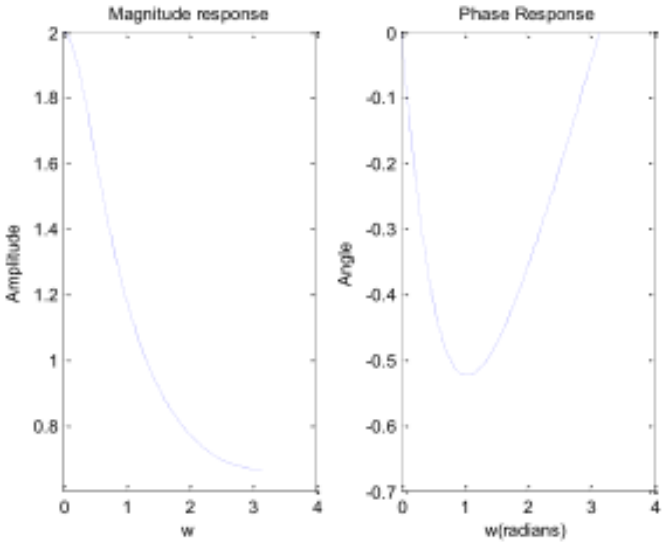


Figure 3.1 Results of example-1

Example-2: Verify the symmetry property of DTFT for the impulse response given below:

$$h[n] = a^n u[n], |a| < 1$$

Solution:

```
clc;
clear all;
close all;

b = [1 0 0];
a = [1 -0.5 0];            % assuming a = 0.5
N = 512;
[h w] = freqz(b,a,N,'whole');
w = w-pi;
subplot(121), plot(w,abs(h));
title('Magnitude Response'), xlabel('w(radians)'), ylabel('Amplitude');
subplot(122), plot(w,angle(h));
title('Phase Response'), xlabel('w(radians)'), ylabel('Angle');
```

Output:

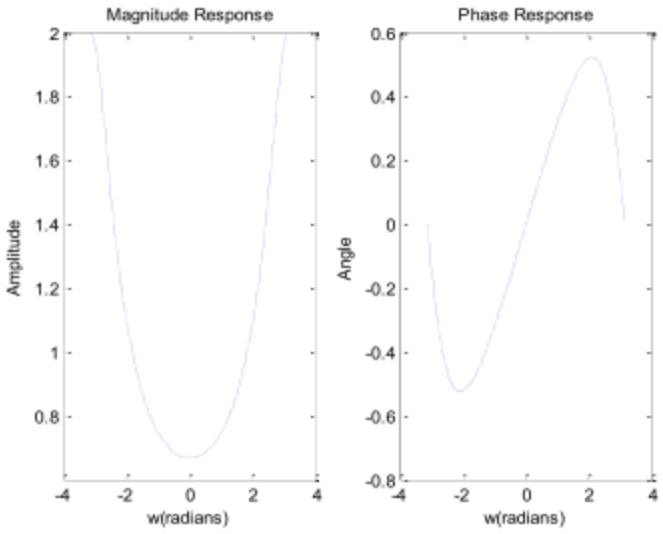


Figure 3.2 Results of example-2

Discrete Fourier Transform

The DTFT spectrum is continuous and hence it is difficult to evaluate on a computer. So to make the evaluation of the DTFT possible on a computer,

we choose a finite number of frequency points. This is equivalent to sampling the DTFT at a certain number of points. This is called the Discrete Fourier Transform (DFT). '*fft'* function in MATLAB is used to compute DFT of finite duration signal. DFT length should be at least equal to the length of the signal or more. By default function *fft* has length equal to the length of the signal.

Example-3: Find the Discrete Fourier Transform (DFT) of a windowed sine wave of frequency 30Hz and plot the magnitude and phase. (Use *fft* function)

Solution:

```
clc; clear all; close all;
frq = 30;
Fs = 1000;
t = 0:1/Fs:1;          %window length is 0 to 1
x = sin(2*pi*frq*t);
X = fft(x);            % fft(x, M) can be used to control the DFT length
l = length(X)/2;
f = (0:(l-1))*Fs/(2*l);
subplot(311), plot(t, x);
title('Sine wave of frequency 30Hz');
xlabel('Time in sec');ylabel('Amplitude');
subplot(312), plot(f,abs(X(1:l)));zoom on;
title('Magnitude Response'); ylabel('Amplitude');
ang=(angle(X(1:l)) * (180/pi));
subplot(313), plot(f, ang);
title('Phase Response');xlabel('Frequency in Hz'); ylabel('Angle in Degrees');
```

Output:

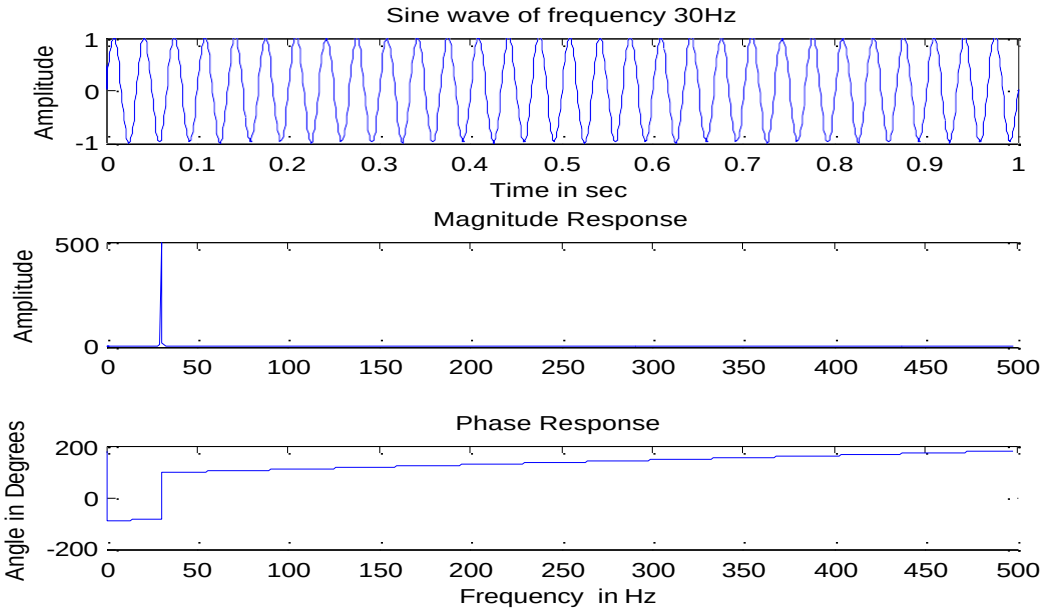


Figure 3.3 Results of example-3

NOTE: Refer MATLAB help for *fft* function

Exercise:

1. Let x[n] = (0.9 $e^{j\frac{\pi}{3}n}$) , 0≤ n ≤ 10. Determine X($e^{j\omega}$) and verify its periodicity property.

2. Determine the frequency response of the filters, which are represented by the difference equations given below. Also, state what type of filter each defines by observing the response (like HPF,LPF etc..).

a) $y[n] + 0.13y[n-1] + 0.52y[n-2] + 0.3y[n-3] = 0.16x[n] - 0.48x[n-1] + 0.48x(n-2) - 0.16x(n-3]$

b) $y[n] - 0.268y[n-2] = 0.634x[n] - 0.634x[n-2]$

c) $y[n] + 0.268y[n-2] = 0.634x[n] + 0.634x[n-2]$

d) $10y[n] - 5y[n-1] + y[n-2] = x[n] - 5x[n-1] + 10x[n-2]$

3. Find and plot the frequency response of a causal 3-point moving averager. Generate first 100 samples of $0.9^n u[n]$ and corrupt it using a random signal. Pass the corrupted signal through the moving point averager and plot the output.

4. Find the DFT of mutitone sine wave with 30Hz, 80Hz, 120Hz frequency components and plot the magnitude Vs. frequency in Hz.

5. Write a MATLAB program to compute the circular convolution of two N-length sequences using DFT/IDFT computations. Use it to determine the circular convolution of the following pair of sequences:
   a) g[n] = [1, -3, 4, 2, 0, -2] and h[n] = [3, 0, 1, -1, 2, 1]
   b) $x[n] = \cos(\frac{n\pi}{2})$, $0 \le n \le 5$ and $y[n] = 3^n$

   Also demonstrate how linear convolution is achieved through circular convolution.

Experiment No. 4: Analysis in Z-domain

Objective:

To study the Z-domain analysis of signals and systems.

Introduction:

Z-transform maps a signal in the time domain to a power series in the complex (frequency) domain. There are many advantageous to working with z-transformed signals. The algebra of system analysis becomes greatly simplified in the z- domain. The response y[n] of an LTI system with impulse response h[n] to input x[n] is given by the convolution x[n] *h[n], which becomes product in Z-domain. The ratio H(z) = Y(z)/X(z) is called system function. In MATLAB *residuez* function is used to convert from rational z-transform to partial fraction form and vice-versa. In using *residuez* function, the z- transfer function is expressed as a rational function with ascending powers of z^{-1}.

Example-1: Consider a causal LTI system described by the difference equation given by y(n) – 1.81y(n - 1) + 0.81y(n − 2) = x(n) + 0.5 x(n − 1). Write MATLAB program to obtain the following

a) System function H(z) in the partial fraction form
b) Pole – zero plot
c) Frequency response of the system
d) Unit impulse response of the system
e) Output of the system for the given input $x[n] = 3\cos(\frac{n\pi}{3})u[n]$

Solution:

```
clc;
clear all;
close all;

b = [1 0.5];              % Coefficients of numerator polynomials
a = [1 -1.8 0.81];        % Coefficients of denominator
polynomials

% To obtain system function in the partial fraction form

[r p k]=residuez(b,a); % finds the residues, poles and direct
terms of the partial-%fraction expansion of the system function.
disp(['r = ' num2str(r')]);
disp(['p = ' num2str(p')]);
disp(['k = ' num2str(k')]);


% To obtain The pole – zero plot

[z p k] = tf2zp(b,a); % To obtain zeros , ploes and gain constant
of the system.
disp('Zeros are at'); disp(z);
disp('poles are at'); disp(p);
disp('Gain Constant is'); disp(k);
zplane(b,a);              % Gives the pole – zero plot



% To obtain frequency response of the system

N = 512;              % Frequency resolution assumed.
[h w] = freqz(b,a,N);% returns the N-point complex frequency
response
   %vector h and the N-point frequency vector w in radians.
figure;
subplot(211), plot(w,abs(h)), title('Magnitude Response');
```

```
xlabel('Frequency in radians'), ylabel('Amplitude');
subplot(212), plot(w, angle(h)), title('Phase Response');
xlabel('Frequency in radians'), ylabel('Phase in radians');
```

```
% To Obtain the unit impulse response of the system

L = 30;                 % First 15 values of the impulse response
[y n] = impz(b,a,L);
figure;
stem(y), title('Impulse Response, h[n]');
grid on;
xlabel('Sample Number'), ylabel('Amplitude');
```

```
% To find the output of the system for the given input

n = 0:100;
u = [n>=0];
x = 3*cos(n*pi/3).*u;        % input signal
y = filter(b, a, x); % o/p signal
figure;
subplot(211), plot(n,x);
subplot(212), plot(n,y);
```

Output:

```
r = -0.55556      1.5556

p = 0.9         0.9

k = 0
```

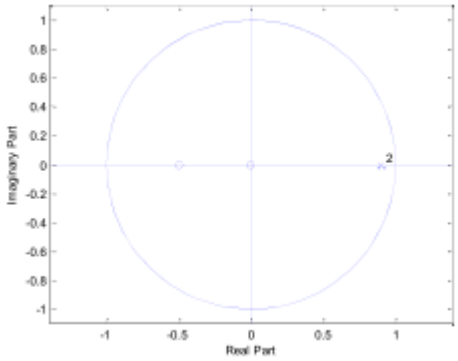


Figure 4.1 Pole zero plot

Zeros are at -0.5000. Poles are at 0.9000 and 0.9000. Gain Constant is 1

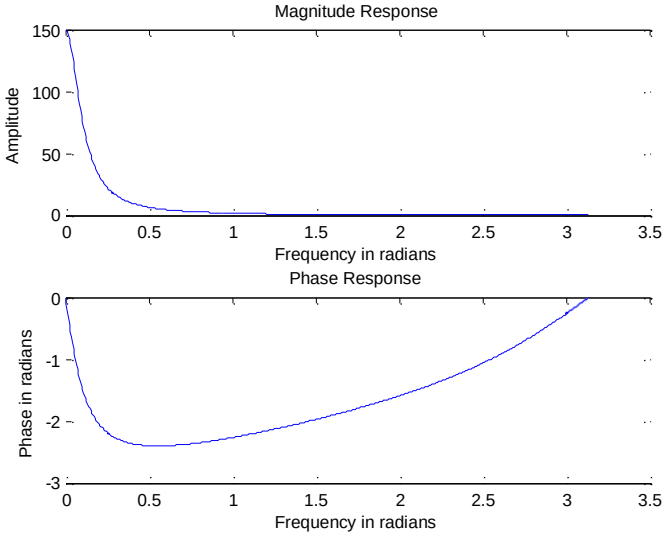


Figure 4.2 Frequency response

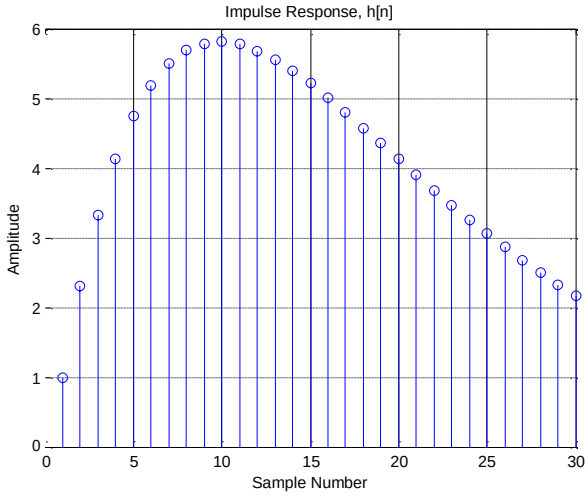


Figure 4.3 Impulse response

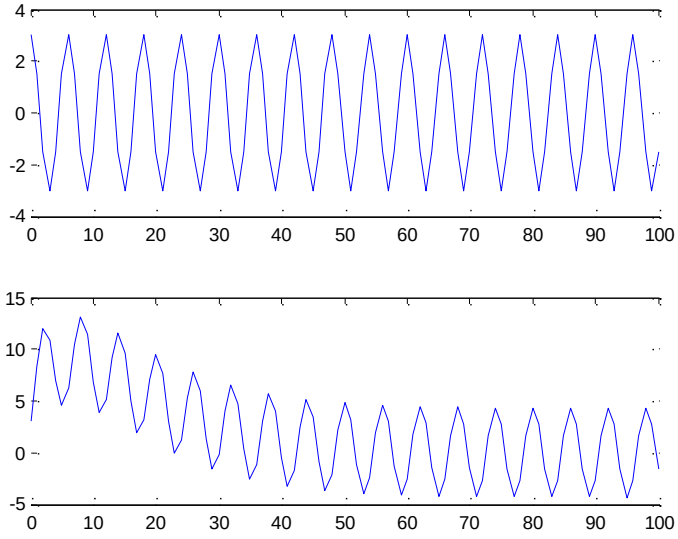


Figure 4.4 Input and output of the system

Example-2: Consider a causal LTI system described by the difference equation $y(n) = \frac{1}{3}[x(n) + x(n-1) + x(n-2)] + 0.95y(n-1) - 0.9025y(n-2)$ with initial conditions y(−1) = −2, y(−2) = −3; x(−1) = 1, x(−2) = 1. Write a MATLAB program to determine the response of the system for the input $x(n) = \cos(\frac{\pi}{3}n)u(n)$.

Note: Analytical Solution has the following steps.

Step1: Compute one sided z-transform of the difference equation

$$Y^+(z) = \frac{1}{3}[X^+(z) + x(-1) + z^{-1}X^+(z) + x(-2) + z^{-1}x(-1) + z^{-2}X^+(z)]$$
$$+ 0.95[y(-1) + z^{-1}Y^+(z)] - 0.9025[y(-2) + z^{-1}y(-1) + z^{-2}Y^+(z)$$

Step2: Substitute the initial condition.

$$Y^+(z) = \frac{\frac{1}{3} + \frac{1}{3}z^{-1} + \frac{1}{3}z^{-2}}{1 - 0.95z^{-1} + 0.9025z^{-2}}X^+(z) + \frac{1.4742 + 2.1383z^{-1}}{1 - 0.95z^{-1} + 0.9025z^{-2}}$$

Step3: Compute $X^+(z) = \frac{1 - 0.5z^{-1}}{1 - z^{-1} + z^{-2}}$

and simplifying, we will obtain $Y^+(z)$ as a rational function.

Step4:

$$Y^+(z) = \frac{0.0584 + j3.9468}{1 - e^{\frac{-j\pi}{3}}z^{-1}} + \frac{0.0584 - j3.9468}{1 - e^{\frac{j\pi}{3}}z^{-1}} + \frac{0.8453 + j2.0311}{1 - 0.95e^{\frac{j\pi}{3}}z^{-1}} + \frac{0.8453 - j2.0311}{1 - 0.95e^{\frac{-j\pi}{3}}z^{-1}}$$

Step5: By applying the inverse z- transform

$$y(n) = (0.0584 + j3.9486)e^{\frac{-j\pi n}{3}} + (0.0584 - j3.9486)e^{\frac{j\pi n}{3}} +$$

$$(0.8453 + j2.031)(095)^n e^{\frac{j\pi n}{3}} + (0.8453.\, j2.031)(095)^n e^{\frac{-j\pi n}{3}}$$

MATLAB solution:

```
clc;
clear all;
close all;
a = [1 -0.95 0.9025]; % Denominator coefficient as in step 2
b = [1 1 1]/3;        % Numerator coefficient as in step 2
y = [-2 -3];          % Initial conditions array
x = [1 1];            % Initial conditions array
xic = filtic(b,a,y,x); % converts past input x and output y  into
                       initial %conditions which is required for the
                       computation of %Y(z). Refer Step 2
ax = [1 -1 1]; bx = [1 -0.5]; % X(z) transform coeffs
ay = conv(a, ax)      % Gives denominator polynomial coefficients
of Y(z)
by = conv(b, bx) + conv(xic,ax)% Gives numerator polynomial
coefficients of Y(z)
[R, p, C] = residuez(by, ay);
disp('Residues '), disp( R), disp('Ploes '), disp( p), disp('Direct Term
    '), disp(C);
```

Output:

Denominator and numerator coefficients of respectively are

ay = 1.0000 -1.9500 2.8525 -1.8525 0. $Y^+(z)$ 9025

by = 1.8075 0.8308 -0.4975 1.9717

The $Y^+(z)$ in the partial fraction form has residues, pole values as shown below.

Residues

0.0584 - 3.9468i

0.0584 + 3.9468i

0.8453 + 2.0311i

0.8453 - 2.0311i

Ploes

0.5000 + 0.8660i

0.5000 - 0.8660i

0.4750 + 0.8227i

0.4750 - 0.8227i

From the above we can write the $Y^+(z)$ as

$$Y^+(z) = \frac{0.0584+ j3.9468}{1 - e^{\frac{-j\pi}{3}} z^{-1}} + \frac{0.0584- j3.9468}{1 - e^{\frac{j\pi}{3}} z^{-1}} + \frac{0.8453+ j2.0311}{1 - 0.95e^{\frac{j\pi}{3}} z^{-1}} + \frac{0.8453- j2.0311}{1- 0.95e^{\frac{-j\pi}{3}} z^{-1}}$$

And by taking the inverse z- transform we get

$$y(n) = (0.0584+ j3.9486)e^{\frac{-j\pi n}{3}} + (0.0584- j3.9486)e^{\frac{j\pi n}{3}} +$$

$$(0.8453+ j2.031)(095)^n e^{\frac{j\pi n}{3}} + (0.8453. j2.031)(095)^n e^{\frac{-j\pi n}{3}} ,$$

$n \geq 0$.

Exercise:

1. An anti-causal LTI system is described by the system function $H(z) = \frac{(z^2 - 1)}{(z-3)^2}$. Determine the following analytically and using MATLAB
   a) impulse response
   b) difference equation representation
   c) pole-zero plot
   d) output y[n] if the input is $x(n) = 3\sin(\frac{\pi}{3} n)\, u(n)$

2. Write a MATLAB program to determine the system function of a system for which the location of zeros and poles, and the value of the gain factor are specified. Test this program for various specifications.

3. Consider a causal LTI system described by the difference equation $y]n] = x[n] - x[n-1] + 0.81y[n-2]$ with initial conditions y(−1) =

   y(−2) = 2 Write a MATLAB program to determine the response of the system for the input $x(n) = 0.7^n u[n+1]$.

Experiment No. 5: IIR Filter Design – 1

Objective: To design and study analog infinite-duration impulse response (IIR) filters.

Theory: Digital IIR filters are mainly designed from analog filters (classical techniques). The widely used IIR filters are Butterworth, Chebyshev l (Type I and II), and Elliptic.

Butterworth filters:

The magnitude response low pass Butterworth filter is $|H(j\Omega)| = \frac{1}{\sqrt{1+(\frac{\Omega}{\Omega_c})^{2N}}}$

Here N is the order of the filter and Ω_c = 3dB cut-off frequency (rad/sec).

A typical magnitude response plot is shown below

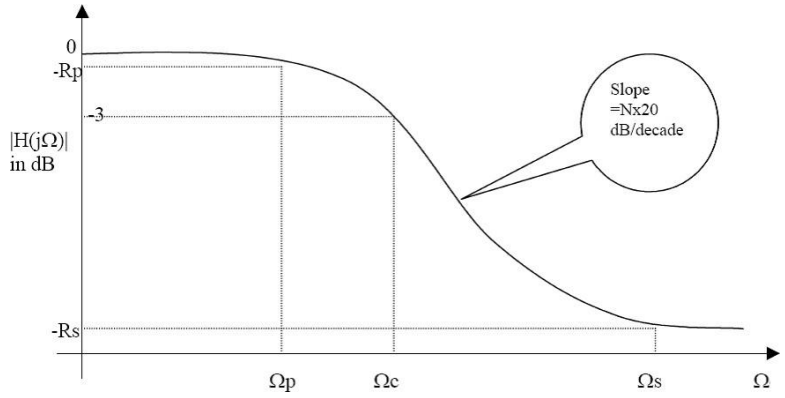


Figure 5.1 Magnitude response of Butterworth LPF

In the above figure, R_p and R_s represent the pass band and stop band attenuations, respectively, and Ω_p and Ω_s are the corresponding edge frequencies in rad/sec.

Chebyshev: A Chebyshev filter with ripples in pass-band is categorized as type-1, whereas, the one with ripples in the stop-band comes under type-2. The figures shown below are examples of type-I filters.

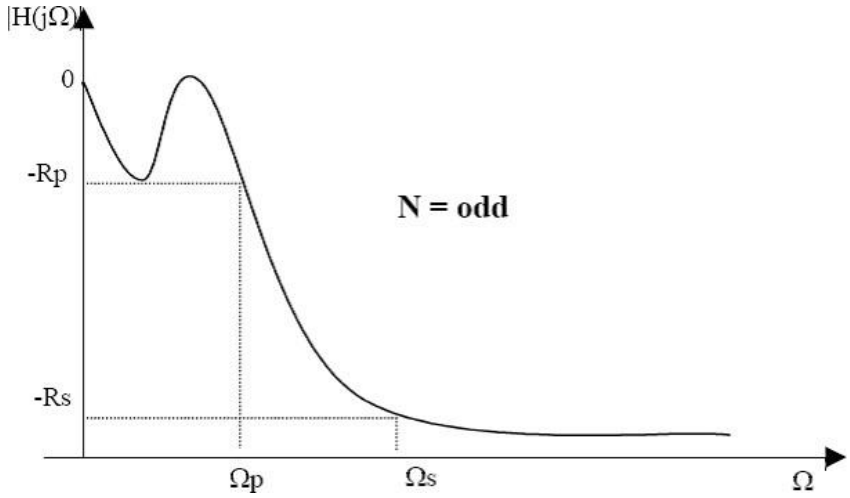


Figure 5.2 Magnitude response of type-1 Chebyshev filter with N odd

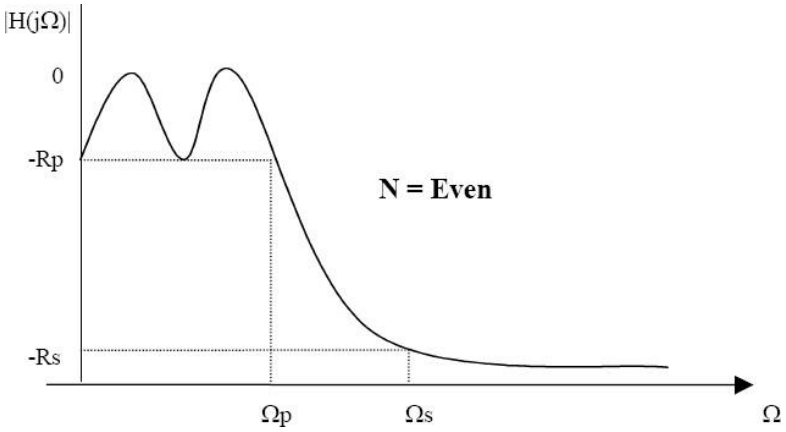


Figure 5.3 Magnitude response of type-1 Chebyshev filter with N even

Elliptic: They have ripples in both pass band and stop-band.

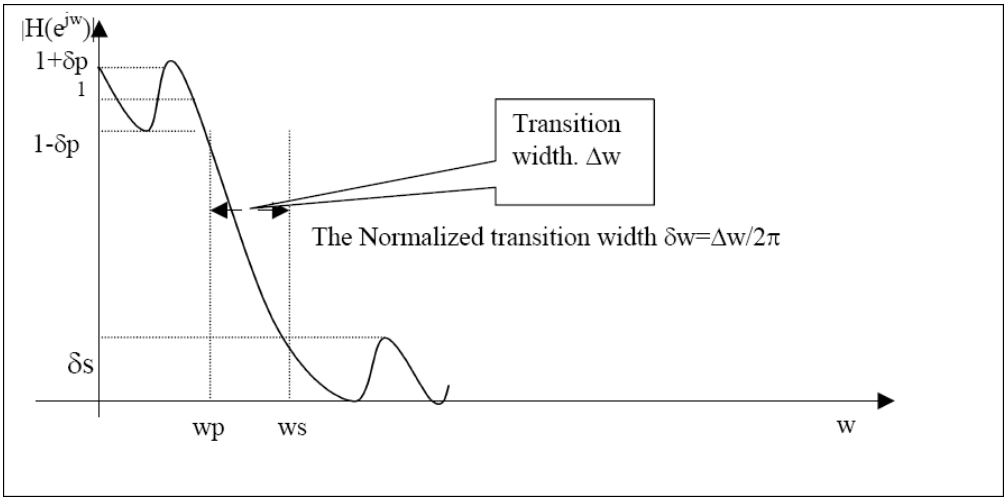


Figure 5.4 Magnitude response of elliptic filter

Implementation in MATLAB:

The following functions are available in MATLAB to design an analog low pass filters.

1. **[N,Wc] = buttord(wp,ws,Rp,Rs,'s')** finds the minimum order N and cutoff frequency Wc for an analog low pass Butterworth filter where wp and ws are the edge frequencies in radians per second.
2. **[b,a] = butter(N,Wc,'s'**) returns the transfer function coefficients of an N^{th} order analog lowpass Butterworth filter with cutoff angular frequency Wc.
3. **buttap(N)** designs a unity cutoff LPF (these are referred to as prototype filters)
4. Similarly, **cheb1ord/cheby1**, **cheb2ord/cheby2**, **ellipord/ellip**

functions can be used to design Chebyshev lowpass (Type 1 and 2), and Elliptic lowpass respectively.

Note: Refer to MATLAB documentation for more information.

Example-1: Design an **analog** Butterworth low-pass prototype filter of order 5. Use it to get an un-normalized Butterworth filter with cutoff frequency of 0.2π rad/sec.

Solution:

```
clc; clear all; close all;
N = 5;                  % given order of the filter
omegac = 0.2*pi;        % Given cutoff frequency
[z, p, k ] = buttap(N);          % Getting the prototype
p1 = p* omegac;
k1 = k* omegac ^N;
B = real(poly(z));
b0 = k1;
b = k1*B;
a = real(poly(p1));
```

Solution:

```
b =       0.0979
a =       1.0000    2.033    2.0671    1.2988    0.5044    0.0979
```

$$H_a(s) = \frac{0.0979}{s^5 + 2.033s^4 + 2.0671s^3 + 1.2988s^2 + 0.5044s + 0.0979}$$

Example-2: Design an **analog** Butterworth low-pass filter, given the following specifications and also plot the frequency response.

R_p = 1dB, R_s = 40dB, f_p = 100Hz and f_s = 500Hz

Solution:

```
clc;
clear all;
close all;
rp = 1;                 %Pass-band attenuation
rs = 40;                %Stop-band attenuation
wp = 2*pi*100;          %Frequency should be in rad/sec
ws = 2*pi*500;
[N, wc] = buttord(wp,ws,rp,rs,'s'); %Getting the order N and the
3dB cut-off freq
[b, a] = butter(N,wc,'s');      %Getting the numerator &
denominator polynomial coeffs
[H, W] = freqs(b,a);            %Complex freq.response of analog
filter
plot(W/(2*pi),20*log10(abs(H))), grid on;
title('Frequency Response');
xlabel(' Frequency in Hz '),ylabel(' Magnitude in dB ');
disp(' the order and cut off frequency in Hz are '),disp( N),disp(
wc/(2*pi));
```

Solution:

Order: 4

Cut-off frequency(Hz): 158.119

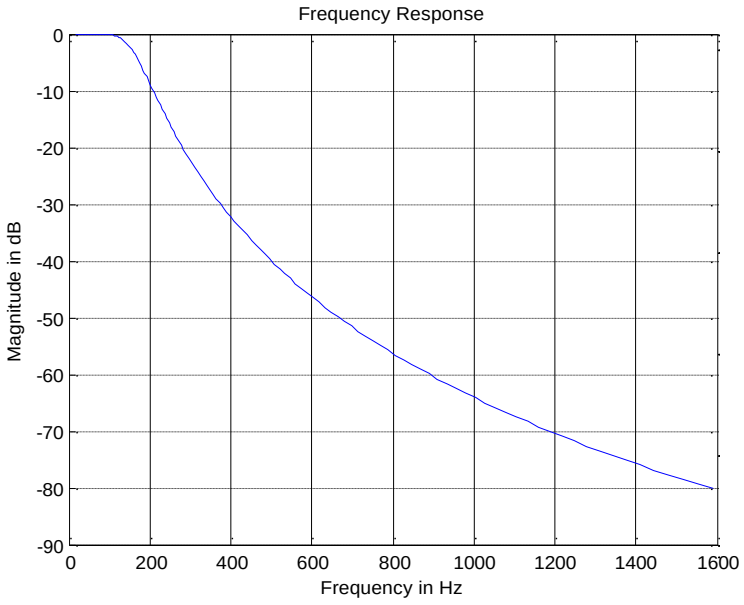


Figure 5.5 Frequency response

Exercise

1. Given the following filter specifications: Passband edge: 200Hz; Rp= 1dB; Stopband edge: 700Hz; Rs= 50 dB.
    a. Design an analog lowpass Butterworth filter
    b. Design an analog lowpass Chebyshev Type 1 filter
    c. Design an analog lowpass Chebyshev Type 2 filter
    d. Design an analog lowpass Elliptic filter

   Plot the frequency response for each case and compare them.

2. Given the following filter specifications: Passband edge: 1kHz; Rp= 1dB; Stopband edge: 200Hz; Rs= 50 dB.
    a. Design an analog high pass Butterworth filter
    b. Design an analog high pass Chebyshev Type 1 filter

   Note: Refer “butter” function to design high pass and other filters.

Experiment No. 6: IIR Filter Design – 2

Objective: To study analog-to-digital filter transformation methods for IIR Filter design.

Theory: The analog filters designed in the previous experiment can be transformed into digital filter using impulse invariance or bilinear transformation (BLT) methods.

Implementation details: The following functions are available in MATLAB:

1. **[bz, az] = impinvar(b, a, Fs)** creates a digital filter with numerator and denominator coefficients bz and az, respectively, whose impulse response is equal to the impulse response of the analog filter with coefficients b and a, scaled by 1/Fs (Fs = sampling frequency).
2. **[numd, dend] = bilinear(num, den, Fs)** convert an *s*-domain transfer function given by num and den to a discrete equivalent using BLT method.

Refer to MATLAB documentation for more information.

Note: The functions *buttord* and *butter* can also be used to design a digital filter directly. For example, the following function returns the order N and the cutoff frequency of digital filter with specifications, wp, ws, rp, rs.

[N, wc] = buttord(wp, ws, rp, rs);

Here rp and rs must be in dB & wp and ws must be normalized to π or half the sampling frequency. (e.g. If fp is 100Hz, then wp = 2*fp/Fs). Therefore, the range for wp and ws is always 0 to 1.

3. **[b, a]=butter(N, wc) -** By default the function butter(N, wc) uses bilinear transformation with frequency prewarping.
4. Similarly, **cheb1ord/cheby1**, **cheb2ord/cheby2**, **ellipord/ellip**

functions can be used to design Chebyshev lowpass (Type 1 and 2), and Elliptic digital IIR filters respectively.

Refer to MATLAB documentation for more information.

Example-1: Design a low-pass **digital** filter using an analog Butterworth filter to satisfy $\omega_p = 0.2\pi\ rad$, $\omega_s = 0.3\pi\ rad$, $R_p = 1\ dB$, $A_s = 15\ dB$. Use the impulse invariance technique to digitize (use *impinvar* function).

Solution:

```
clc, clear all , close all;
% Digital Filter specification
wp = 0.2*pi;
ws = 0.3*pi;
rp = 1;
rs = 15;

% Analog design
T = 1;                      % assumed
wa_p = wp/T;
wa_s = ws/T;

% Butterworth filter design
[Nb, wcb] = buttord(wa_p,wa_s, rp, rs, 's');
[bb_s,ab_s] = butter(Nb, wcb, 's');
disp('For Butterworth, the order and cut off frequency are '),
disp( Nb),disp(wcb);
[Hb,wb] = freqs(bb_s,ab_s);             % Analog freq. response

% Transformation from analog to digital
[bb_z, ab_z] = impinvar(bb_s,ab_s,1/T);
[Hbd, wbd] = freqz(bb_z,ab_z);        % Digital freq. response
figure(1); plot(wbd/pi,20*log10(abs(Hbd))),grid on;
ylabel('Magnitude in dB' ) ;
xlabel('Frequency in pi units') ;
```

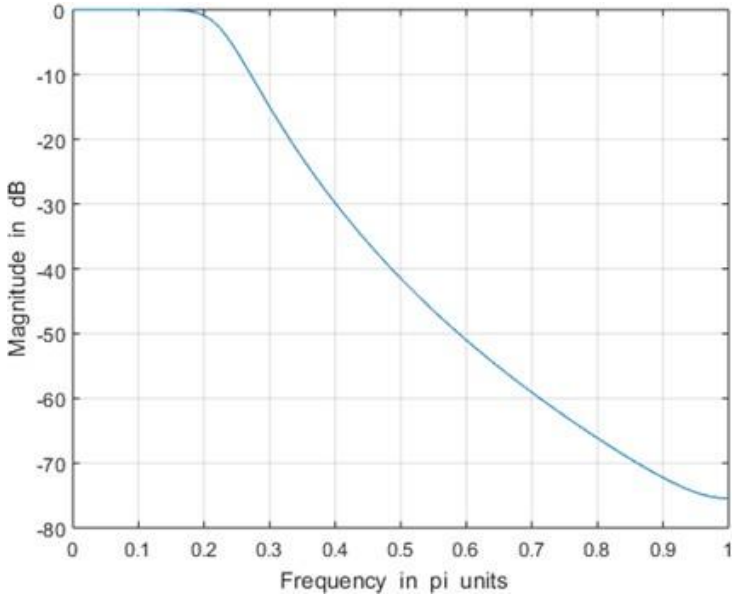


Figure 6.1 Frequency response

Example-2: Simulate a signal with 50Hz and 200 Hz frequency. Filter the signal through a 3rd order digital Butterworth LPF with 3-dB cutoff frequency 100 Hz. Plot the spectrum of the input and the filtered signal.

Solution:

```
clear all; clc ; close all
Fs = 1000; % Sampling rate
t = 0:1/(Fs):0.5;
wc = 2*100/Fs                    %Normalized 3 dB cutoff freq. for the
filter
x = sin(2*pi*t*50) + sin(2*pi*t*200);
[b,a] = butter(3,wc);            %Getting the numerator &
denominator polynomial coeffs
[H,W] = freqz(b,a,Fs);           %Complex freq. response of analog
filter
figure(1);
```

```
k=W*Fs/(2*pi);                  % defining frequency vector for
plotting
plot(k,20*log10(abs(H))), grid on;
title('Frequency Response');
xlabel(' Frequency in Hz '),ylabel(' Magnitude in dB ');
% Filter testing
yb = filter(b,a,x) ;
% Getting spectrum and plot
N = 512;
w  =  [0:N/2 - 1]* (Fs/N) ;      %defining frequency vector for
plotting
X = fft(x,N);
Yb = fft(yb,N);
figure(2);
subplot(211), plot(w,abs(X(1:N/2)));
title('Spect rum of input signal ' ) ;
subplot(212), plot(w,abs(Yb(1:N/2)));
title('Spectrum of filtered signal ' ) ;
```

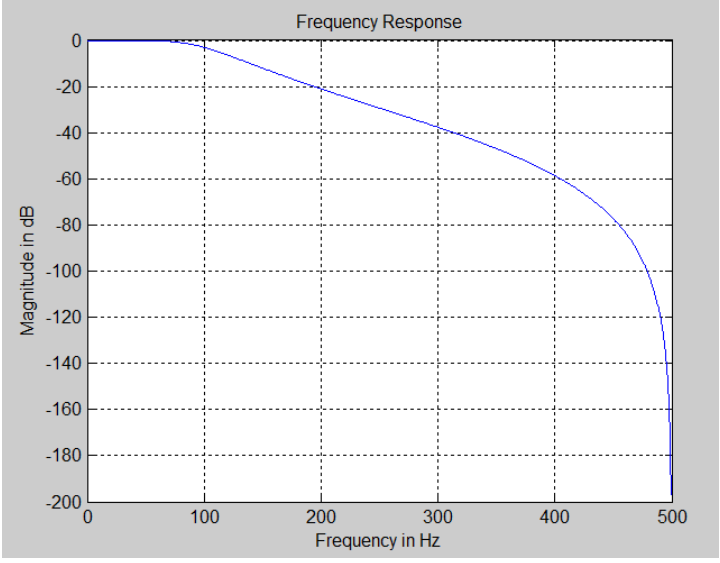


Figure 6.2 Frequency response

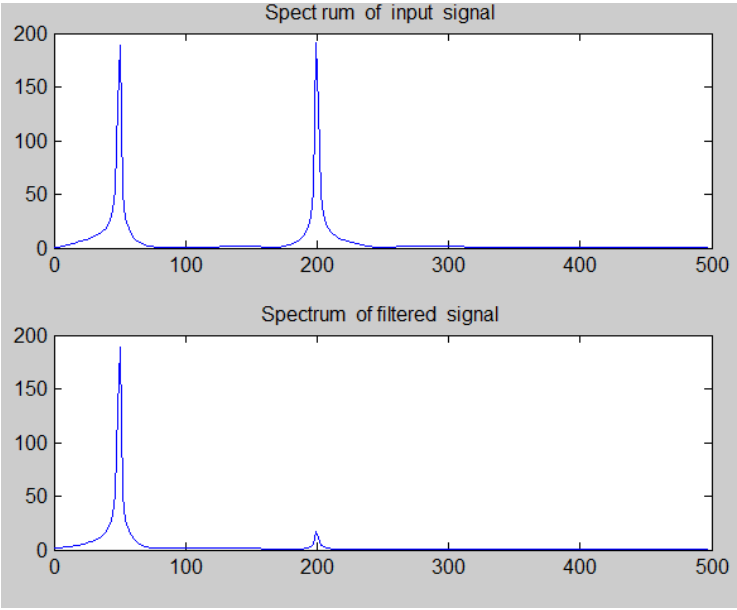


Figure 6.3 Input and output for example-2

Exercise:

1. Transform

$$H_a(s) = \frac{s+1}{s^2+5s+6}$$

into a **digital** filter H(z) using the impulse invariance technique in which T = 0.1s. First find the solution analytically and then verify your result with the help of a MATLAB program. Also plot the magnitude responses of the analog and the resulting digital filter and comment on the result.

2. Design a low-pass **digital** filter using a
   i) Chebyshev-I analog filter
   ii) Chebyshev-II analog filter
   iii) Elliptic analog filter

   to satisfy $\omega_p = 0.2\pi\ rad, \omega_s = 0.3\pi\ rad, R_p = 1\ dB, A_s = 15\ dB$. Use the impulse invariance technique.

3. Transform the system function given in Question 1 above using the bilinear transformation assuming T = 1s.

4. Repeat Question 2 using bilinear transformation.

5. Certain **digital** low-pass filter has the following specifications:
   Pass-band frequency = 0.2π rad, Pass-band ripple = 1dB

   Stop-band frequency = 0.3π rad, Stop-band attenuation = 20dB.

   Write a MATLAB program to design this filter using Butterworth, Chebyshev and elliptic methods. Plot the frequency response in each case and compare the performance (Use functions like butter, cheby1…etc).

6. Design using MATLAB, an analog Butterworth low-pass filter that has maximum of 1dB attenuation at 30rad/sec and at least 30dB attenuation at 40rad/sec. Digitize this using bilinear and impulse invariant transformation. Use suitable sampling rate. Plot the frequency response in each case.

Note: Refer MATLAB functions to design high pass and other filters.

Experiment No. 7: FIR Filter Design

Objective: To design and study finite-duration impulse response (FIR) filters.

Theory: A typical FIR (low-pass) filter response is as shown below Pass

band ripple in dB is $R_p = -20\log_{10}[(1-\delta_p)/(1+\delta_p)]$

Stop band attenuation in dB is $A_s = -20\log_{10}\delta_s$

Pass band deviation in dB = $-20\log_{10}(1+\delta_p)$

For an N^{th} order filter, there are N+1 coefficients or filter length is N+1.

Important design methods are window based design and frequency sampling design.

Window based Design: The commonly used windows are hamming window and Kaiser Window.

Hamming window, $\mathbf{w[n]} = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{N-1}),\ 0 \le n \le N-1 \\ 0,\ otherwise \end{cases}$

The normalized transition width is δ_w, (transition width divided by sampling frequency or (w_s-w_p) divided by 2π) and filter length N are related by $\delta_w = 3.3/N$, and the designed filter will have a stop band attenuation of 53 dB.

Kaiser window, $\mathbf{w[n]} = \frac{I_0[\beta\sqrt{1-(1-\frac{2n}{N-1})^2}]}{I_0(\beta)}$

Where $I_0[.]$ is modified zero order Bessel function and β is a parameter that depends on N.

The filter length M is given by $M = [\frac{A_s - 7.95}{14.36\delta w}] + 1$

where $\beta = \begin{cases} 0.1102(A_s - 8.7),\ A_s \ge 50 \\ 0.5842(A_s-21)^{0.4} + 0.07886(A_s-21),\ 21 < A_s < 50 \end{cases}$

The other windows are rectangular, hanning, Bartlett, Blackman and raised cosine.

Frequency Sampling Design: The desired frequency response is sampled at discrete frequency points 2πk/N to get frequency domain points from which filter taps are obtained.

For linear phase filter, $H[k] = H_r[k]e^{j\angle H[k]}$

where, $H_r[k]$ = sampled magnitude value

and $\angle H[k] = \{ \begin{matrix} -\alpha\left(\frac{2\pi k}{N}\right), k = 0,1, \ldots \frac{N-1}{2} \\ \alpha\left\{\frac{2\pi(N-k)}{N}\right\}, k = \frac{N-1}{2}+1, \ldots, N+1 \end{matrix}$

where, $\alpha = \frac{N-1}{2}$, and if N is even, lower integer of $\frac{N-1}{2}$ is taken for k.

To get filter coefficients, N-point inverse FFT is taken.

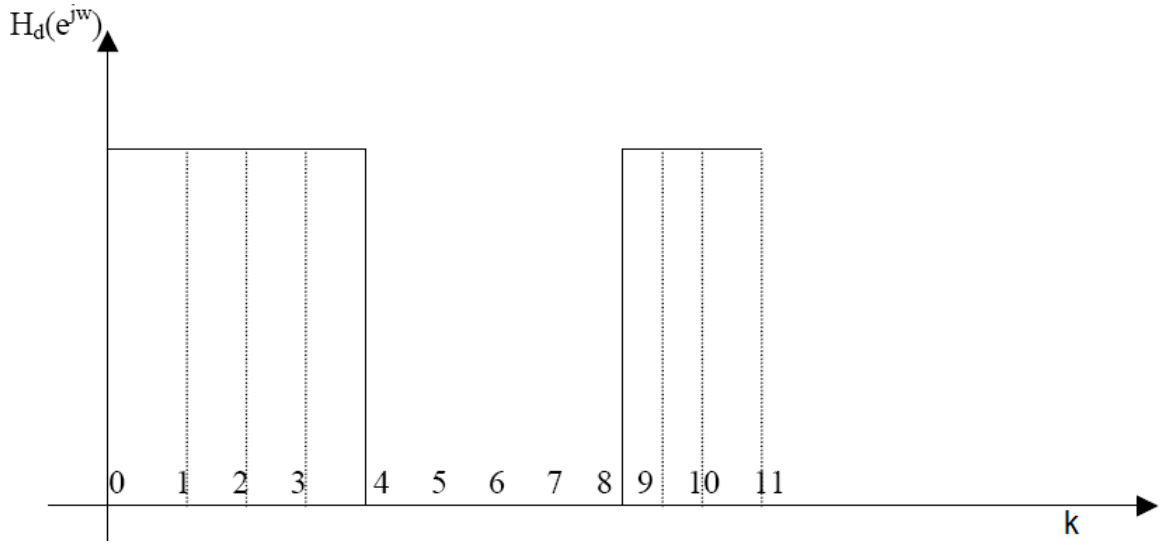


Figure 7.1 Frequency Sampling Design example

Note: The length M can be estimated as follows

For LPF/HPF:

$M = \frac{D}{\delta_f} - F\delta_f$, where D and F are functions

$$D = [a_1(\log_{10}\delta_p)^2 + a_2\log_{10}\delta_p + a_3]\log_{10}\delta_s + [a_4(\log_{10}\delta_p)^2 + a_5\log_{10}\delta_p + a_6]$$

$$F = 11.01217 + 0.51244\,(\log_{10}\delta_p - \log_{10}\delta_s)$$

$a_1 = 5.309 \times 10^{-3}$, $a_2 = 7.114 \times 10^{-2}$, $a_3 = -4.761 \times 10^{-1}$,

$a_4 = -2.66 \times 10^{-3}$, $a_5 = -0.5941$, $a_6 = -0.4278$.

and δ_f is the normalized transition width.

For BPF:

$M = \frac{C}{\delta_f} - G\delta_f$, where C and G are functions of δ_p and δ_s given by

$$C = [b_1(\log_{10}\delta_p)^2 + b_2\log_{10}\delta_p + b_3]\log_{10}\delta_s + [b_4(\log_{10}\delta_p)^2 + b_5\log_{10}\delta_p + b_6]$$

$$G = -14.61\log_{10}(\delta_p/\delta_s) - 16.9$$

$b_1 = 0.01201$, $b_2 = 0.09664$, $b_3 = -0.51325$, $b_4 = 0.00203$, $b_5 = -0.5705$, $b_6 = -0.44314$.

Example-1: Design, using window based approach, a low-pass **digital** FIR filter whose desired specifications are: $w_p = 0.2\pi$ rad, $w_s = 0.3\pi$ rad, $R_p = 0.2$dB, $A_s = 50$dB.

a) Hamming window

Solution:

```
clc, clear all, close all;
wp = 0.2*pi;
ws = 0.3*pi;
N = ceil(6.6*pi/(ws-wp));      % estimates order of the filter
b = fir1(N, 0.2);
[H,W] = freqz(b,1,512);
```

```
plot(W/pi,20*log10(abs(H))), grid on, title('FIR filter using Hamming window');
xlabel('Normalized Frequency'), ylabel('Gain in dB');

% this filter provides about 53dB attenuation in stop band
% and pass band ripples are less than 0.25dB as expected.
```

Solution:

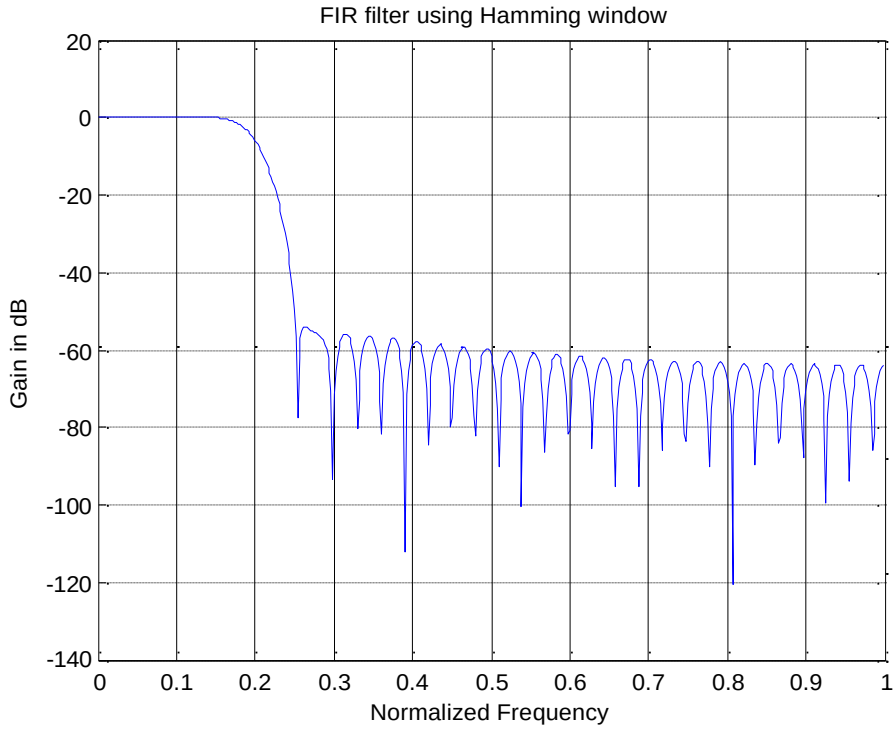


Figure 7.2 FIR filter frequency response

Example-2: Design the filter specified in problem no.1 using frequency sampling design technique assuming linear phase.

Solution:

```
clc, clear all, close all;
M = 20;          % assumed
alpha = (M-1)/2;
Hrs = [1,1,1,zeros(1,15),1,1];
```

```
k1 = 0: floor((M-1)/2);
k2 = floor((M-1)/2)+1:M-1;
angH = [-alpha*(2*pi)*k1/M, alpha*(2*pi)*(M-k2)/M];
H = Hrs.*exp(j*angH);
h = real(ifft(H,M));
[H,W]=freqz([h],1,1024);
plot(W/pi,20*log10(abs(H))), zoom on, grid on;
title('FIR LPF using frequency sampling design');
xlabel('Normalized Frequency'), ylabel('Gain in dB');

% Note that stop band attenuation is not very good, is about 15dB only
```

Solution:

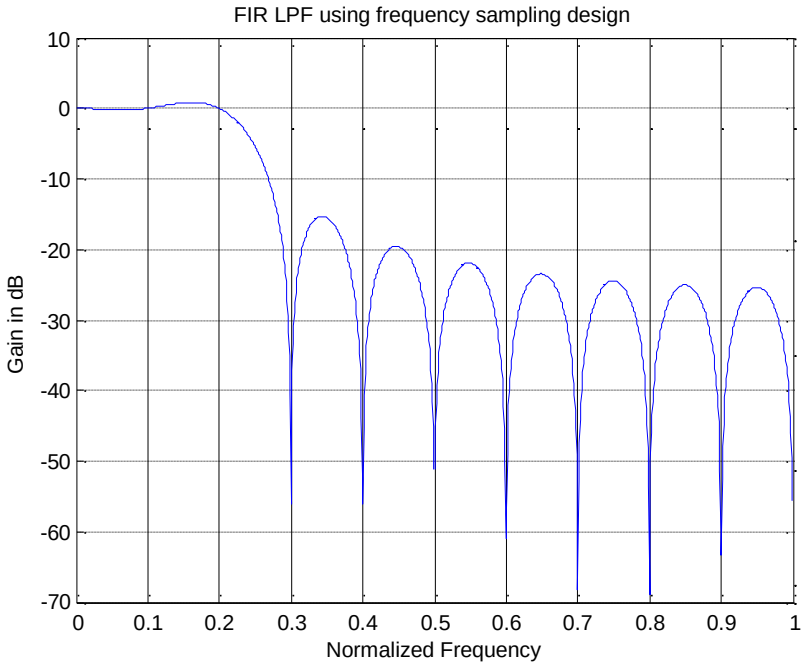


Figure 7.2 FIR filter frequency response

Exercise:

1. Design, using Kaiser window based approach, a low-pass **digital** FIR filter whose desired specifications are: $w_p = 0.2\pi$ rad, $w_s = 0.3\pi$ rad, $R_p = 0.2$dB, $A_s = 50$dB.

2. Design and plot the frequency response of a linear phase band-pass filter whose

specifications are as follows: Lower pass band edge = 400Hz, upper pass-band edge = 500Hz, lower stop- band edge = 300Hz and upper stop-band edge = 600Hz, R_p = 0.5dB and A_s = 40dB. The sampling frequency is 2 kHz.

a. using hamming window
b. using Kaiser window

3. Design the same filter in sample problem 2 with M = 25 and compare them.

4. The attenuation in stop band was observed to be not adequate in the above problem. To improve, a few non-zero points are considered in the transition band. These values are normally estimated by optimization techniques. In the above problem, use Hrs(k) = [1, 1, 1, 0.5, 0.1 , zeros(1,11), 0.1,0.5,1,1] Compare the results and find out whether there is any improvement. If there is, then at what cost?
5. Design the filter specified in problem 2 using frequency sampling designing technique.
6. Certain signal has 10Hz, 30Hz and 50Hz components. Simulate this signal. Design suitable FIR filter using the following methods to select only 30Hz component. Filter the signal using the filters, and plot the spectrum of the input and the filtered signal. Hence compare the performance of the filters.
   a) Window method
   b) Frequency sampling designing

Experiment No. 8

DSP Applications Using MATLAB

Objective:

1. To study the processing of digital images using MATLAB.
   a)
2. To study how speech signals are processed using MATLAB.
3. To study power spectrum estimation methods using MATLAB.

1. For any image file available on your disk, perform the following tasks:
   (a) Read and display the image.
   (b) Get the negative of the image.
   (c) Rotate and display the image.

   Solution:

```
clc;
close all;
clear all;
% a)
a = imread('image.bmp','bmp');
subplot(221),imshow(a), title('a) Image');
% b)
b = double(a);
b = 256-b;
b =uint8(b);
subplot(222), imshow(b), title('b) Negative of the image');
% c)
c = a';
subplot(223), imshow(c), title('c) Image rotated anti-clockwise');
```

2. Write a MATLAB program to improve the contrast of an image by using histogram equalization. Also display the images and their respective histograms before and after equalization. (Use *histeq* and *imhist* functions.)

3. Convolve the two matrices given below:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 8 & 6 \\ 5 & 3 & 1 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 3 & 5 \end{bmatrix}$$

4. Compute the DFT of an image and plot the response with centered low frequencies. (Use fft2 and fftshift functions.)

5. Write a MATLAB program to detect the sharp edges in an image in the horizontal direction.

6. Write a MATLAB program to add 'salt and pepper' type of noise to an image, and then filter it. Show all the three images on the same plot.

7. Write a MATLAB program to compress an image using 2D-DCT and then try reconstructing the image by using only 50% of the DCT coefficients.
   Solution:

```
clc;
close all;
clear all;
a = imread('image.bmp','bmp');
subplot(221), imshow(a);
title('Given image');
a1 = dct2(a);
a2 = log(abs(a1));
a3 = mat2gray(a2);
subplot(222),imshow(a3);
title('DCT of the image');
a4 = triu(a1);% Taking only upper triangle of the DCT matrix
a5 = idct2(a4);
a6 = mat2gray(abs(a5));
subplot(223), imshow(a6);
title('Reconstructed image after compression');
```

8. Write a program to read an image and highlight only specified region. (Use *iroipoly* function.)

9. Re-size an image to twice its size. Add salt and pepper noise with a density of 0.25. Filter the noisy image with median filtering using window sizes 3x3, 5x5 and 7x7. More than one repetition may be done. Display and compare the resulting images. Which one is most suitable for further processing? (Use imresize, imnoise and medflt2functions.)

10. Compute the Fourier Transform of an image .Rotate the original image by 45 degrees and compute the FT of the rotated image. Move the origin to the center of the FT. Display the logarithm of magnitude of the FT before and after rotation. What happens to the FT after rotation? (Use fft2, *imrotate, fftshift, abs, log* and *imagesc* functions.)

Speech Processing

Theory: Speech is an acoustic signal, which is produced when air is forced from the human lungs through the vocal chords (glottis) and along the vocal tract. The human voice conveys information which is received by our ears and decoded by the brain according to the sounds used in the language used. Speech signal is a highly non-stationary signal, having many frequency components. It is required to find out which frequency is present at what time period. This is called time resolution of frequency. Speech signal is actually slowly time-varying signal, i.e., when observed over a sufficiently short duration of time, it is fairly stationary. Therefore, we take short-time Fourier transform (STFT) of speech signal to find its frequency content.

1. Write a MATLAB program to read and plot a given speech file, find its frequency content and also find which frequencies are present at what time.
   Solution:

```
clc;
close all;
clear all;
[a,fs] = wavread('test.wav');
subplot(311), plot(a);
title('Speech signal plot'), xlabel('Time');
%sound(a,fs);
a1 = abs(fft(a));
len = length(a);
t = 0:1:len-1;
t = t.*fs;
subplot(312), plot(t,a1), grid on;
title('Frequency components'), xlabel('Frequency');
subplot(313), spectrogram(a,hamming(64),32,64,fs);
title('STFT of the speech sample');
```

2. Write a MATLAB program to find the linear prediction coefficients (LPC) of given speech sample. Plot them and use to reconstruct the speech sample. (Use *lpc* and *filter* functions. Use 14th order linear predictor FIR filter.)

3. Write a MATLAB program to estimate the fundamental frequency (pitch) of a speech signal from its spectrum. (Cepstrum method of pitch estimation.)

Power Spectrum Estimation

Theory: Some of the important methods of power spectrum estimation are:

1. Periodogram method
2. Welch's method

3. Co-variance method
4. Yule-Walker AR method

Exercise: In MATLAB, direct functions for the above mentioned methods are available, namely, *periodogram*, *pwelch*, *pcov* and *pyulear*.
Study these functions from the MATLAB help. Simulate suitable signals of your own, and obtain and plot their power spectrum. You can also use the speech signals used in the previous section.

Experiment No. 9: Introduction to Code Composer Studio (CCS)

Objective:

1. To get familiarized with code composer studio.
2. To implement basic DSP programs and verify the simulation.

Introduction: Code composer is the DSP industry's first fully integrated development environment (IDE) with DSP-specific functionality. With a familiar environment liked MS-based C++TM, Code Composer lets you edit, build, debug, profile and manage project from a single unified environment. Other unique features include graphical signal analysis, injection/extraction of data signals via I/O, multi-processor debugging, automated testing and customization via a C-interpretive scripting language and much more.

Code Composer Studio (CCS) Features:

1. IDE
2. Debug IDE
3. Advanced watch windows
4. Integrated editor
5. File I/O, Probe points, and graphical algorithm scope probes.
6. Advanced graphical signal analysis
7. Interactive profiling
8. Automated testing and customization via scripting
9. Visual project management system
10. Compile in the background while editing and debugging
11. Multi-processor debugging
12. Help on the target DSP

Note:
Documents for Reference:
Spru509→ Code Composer Studio getting started guide.
Spru189→ TMS320C6000 CPU and Instruction set guide.
Spru189→ TMS320C6000 Peripherals guide.
Slws106d→ codec (TLV320AIC23) data manual.
Spru402→ Programmer's reference guide.
Sprs186j→ TMS320C6713 DSP

Procedure to work on Code Composer Studio (CCS version 3.1/3.3) –Simulation

Note: Run "***setup CCStudio V3.3***" from the desktop. From the 'available factory boards' tab choose 'family' as 67XX, 'platform' as simulator and 'endianness' as little. Now select 'C6713 device cycle Accurate simulator' and save & quit to start the CCS.

Follow the below shown steps to create project on CCS and run the simulation.

To create the new project:
Project→ New (Give the file name, ex: myproj.pjt)

To create a source file:
File→ New→Type the c-code in the new window and save it as "filename.c" file (Ex: source.c)

To add source file to the project:
Project→ Add files to project→ select the source file saved in the previous step. (source.c)

To add library file to the project
Project→ Add files to project→ select the "filename.lib" file from the following path and add to the project.
Path: c:\CCStudio_v3.1\c6000\cgtools\lib\rts6700.lib

Compile the project:
Project→ compile (Any compilation errors, if any will be displayed)

Build the project:
Project→ Build (Any build errors, if any will be displayed)

Load and Run the program:

File→ Load program→ myproj.out (This file is available in the "myproj\debug\" folder). Now the machine program is loaded on to the memory.
Debug→ Run, Now program will be executed.

Example-1: C-program to print a message 'Hello World'

```
# include<stdio.h>
main()
{
        int i=0;
        i++;
        printf("Hello World");
        printf("\n%d",i);
}
```

Example-2: C-program to implement linear convolution

```
// Linear convolution
# include<stdio.h>
int y[10]={1,2,3,4,0,0,0,0}; //global variable for output
main()
{
        int m=4;
        int  n=4;
        int i=0,j;
        int x[10]={1,2,3,4,0,0,0,0};
        int h[10]={1,2,3,4,0,0,0,0};
        for(i=0;i<(m+n-1);i++)
          {
          y[i]=0;
          for(j=0;j<=i;j++)
          y[i]=y[i]+x[j]*h[i-j];
          }
   for(i=0;i<(m+n-1);i++)
        printf("%d\n",y[i]);
}
```

Output:

1
4
10
20
25
24
16

Note: To obtain the graphical view of the result, choose view→time-frequency plot. The following window will appear. Set the parameters as shown in the window and click 'ok'.

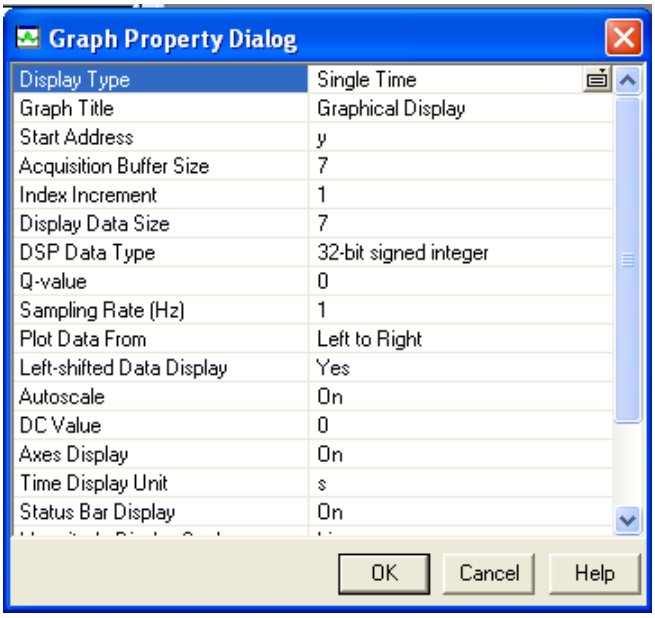


Figure 8.1 Graph property dialog

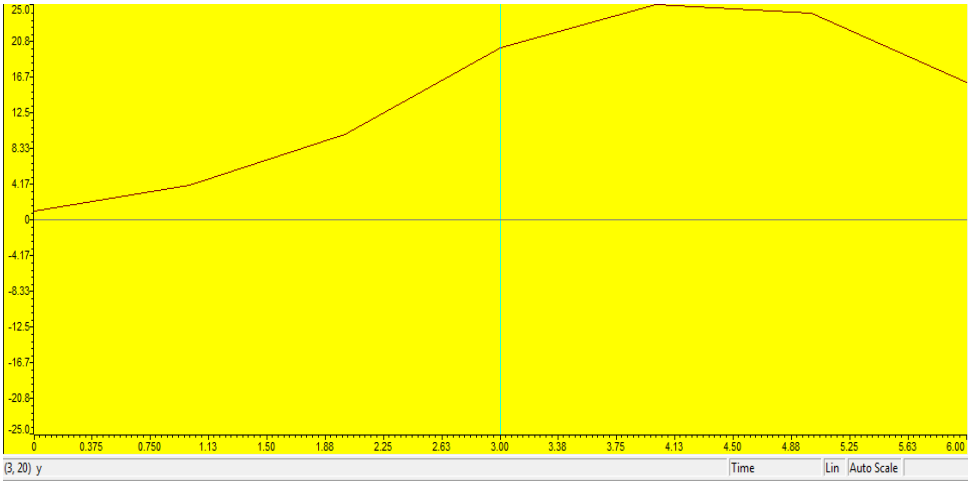


Figure 9.2 Graphical view of the result

Example-3: C-program to implement circular convolution

```
// cicular convolution
# include<stdio.h>
int m,n,x[30],h[30],y[30],i,j,temp[30],k,x2[30],a[30];
main()
```

```
{
        printf("Enter the legth of input sequence\n");
        scanf("%d",&m);
        printf("Enter the legth of impulse response\n");
        scanf("%d",&n);
        printf("Enter the input sequence x[n]\n");
        for(i=0;i<m;i++)
        scanf("%d",&x[i]);
        printf("Enter the impulse reponse sequence h[n]\n");
        for(i=0;i<m;i++)
        scanf("%d",&h[i]);
        if((m-n)!=0) // if length the wto sequences not equal
         {
         if(m>n) // pad smaller sequence with zeros
          {
          for(i=n;i<m;i++)
          h[i]=0;
          n=m;
          }
          for(i=m;i<n;i++)
          x[i]=0;
          m=n;
        }
        y[0]=0;
        a[0]=h[0];
        for(j=1;j<n;j++)
        a[j]=h[n-j];
        // Circular convolution
        for(i=0;i<n;i++)
        y[0]=y[0]+x[i]*a[i];
        for(k=1;k<n;k++)
        {
         y[k]=0;
         // Circular shift
         for(j=1;j<n;j++)
         x2[j]=a[j-1];
         x2[0]=a[n-1];
         for(i=0;i<n;i++)
          {
          a[i]=x2[i];
          y[k]=y[k]+x[i]*x2[i];
          }
   }
        // Displaying the result
        printf("Circular convolution is: \n");
        for(i=0;i<n;i++)
        printf("%d\t",y[i]);
}
```

Output:

```
Enter the legth of input sequence
4
Enter the legth of impulse response
4
Enter the input sequence x[n]
1
2
1
1
Enter the impulse reponse sequence h[n]
1
2
3
4
Circular convolution is:
      14     11        12          13
```

Example-4: C-program to generate stair case waveform

```
#include<stdio.h>
float b=0,a[24];
//a[24]={1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4};
main()
{
int k=0,i=0,j=0;
for(j=0;j<6;j++)
{
for(i=0;i<4;i++)
{

a[k]=b;
k++;
}
b=b+1;
}
}
```

Output:

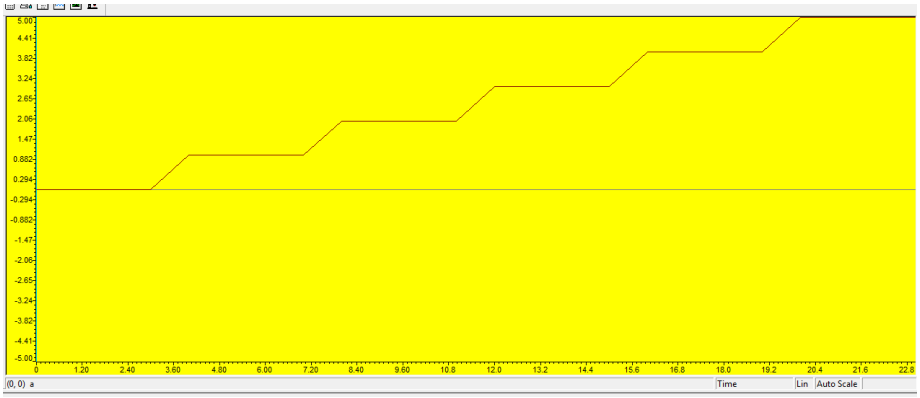


Figure 9.3 Stair case waveform

Example-5: C-program to generate triangular waveform

```
#include<stdio.h>
int res[32]={1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5};
main()
{
int i=0,on[7]={0},off[2]={0},j,k=0,l,m,n;
for(j=0;j<4;j++)
{
m=0;
n=3;
for(l=0;l<8;l++)
{
if(l<5)
{
res[k]=m++;
printf("%d\n",res[k]);
k++;
}

else
{
res[k]=n--;
```

```
printf("%d\n",res[l]);
k++;
}
}
}
k++;
}
```

Output:

Figure 9.4 Triangular waveform

Exercise:

1. Write c-program to generate sine wave of frequency 500Hz.
2. Write c-program to generate multi-tone waveform containing 3-frequency components 500Hz, 750Hz and 1000Hz.
3. Write c-program to find the auto correlation and cross correlation between two input sequences.
4. Write c-programs to generate square waveform of 1 kHz frequency.

Experiment No. 10: Filter Implementation using TMS320C6713 DSK

Objective:

1. To configure the codec in TMS320C6713-DSK for a talk through program using the board support library.
2. To implement filters (FIR and IIR) on the DSK and verify their working.

Required components:

TMS320C6713 DSP starter kit, PC with code composer studio (CCS version 3.1), CRO and function generator.

Note:

1. Run "***setup CCStudio V3.1***" from the desktop. From the 'available factory boards' tab choose 'family' as 67XX, 'platform' as dsk and 'endianness' as little. Now select 'C6713 DSK' and save & quit to start the CCS.
2. After opening the CCS application, connect the board. Choose **debug→connect.** (We can see that board is "connected")

Example 1: To configure the codec TLV320AIC23 using board support library and verify its working.

Procedure:

Note:

- All the real-time implementations covered in the implementations module follow code configuration using board support library (BSL).
- The BSL is a collection of functions, macros, and symbols used to configure and control on-chip peripherals.
- The goal is ease use of peripherals, shortened development time, portability, hardware abstraction, and some level of standardization and compatibility among TI devices.
- BSL is a fully scalable component of DSP/BIOS. It does not require the use of other DSP/BIOS components to operate.

Follow the below shown steps to configure and verify the codec:

1. Connect CRO to the socket provided for LINE OUT.
2. Connect a signal generator to the LINE IN socket.

3. Switch ON the signal generator with a sine wave of amplitude 5Volts (peak- peak) and frequency 500Hz.
4. Now switch ON the DSK and bring up code composer studio on PC.
5. Create a new project (say "myproj.pjt"). Project→ New. (Set project name, location and project type—executable (.out), target—TMS320C67XX, and click on "Finish")
6. Write source program and save it with ".c" extension (say "codec.c"). File→ New→ Source file. (Type the source program and save it as codec.c).
7. Add source file to the project. (Project→ Add files to project→ codec.c)
8. Add configuration files: File→New→DSP/BIOS configuration. (Here select "dsk6713.cdb" file and save it as "config1.cdb"). Now add the saved 'config1.cdb' to the project: Project→Add files to project→ (select "config1.cdb"). Note that 3 files are now added to the "Generated files" folder in the project.
9. View the contents of "config1cfg_c.c" (which is there under the "Generated files") and copy the line containing "#include "config1cfg.h"' to the source file 'codec.c'.
10. Add the generated file "config1cfg.cmd" to the project.
11. Add library file to the project: Project→Add files to project→"dsk6713bsl.lib". (This file is available in path: c:\CCStudio_v3.1\c6000\dsk6713\lib\dsk6713bsl.lib).
12. Set the path for pre-processor: Project→Build options→preprocessor. (Here set the path under include to: c:\CCstudio_v3.1\c6000\dsk6713\include\).
13. Now, build, load and run the program.

C-program to configure and test the onboard codec.

Codec.c

```
//#include "filtercfg.h"

#include "dsk6713.h"
#include "dsk6713_aic23.h"

// codec configuration setting
DSK6713_AIC23_Config config={\
0x0017, //0. DSK6713_AIC23_LEFTINVOl left line input channel volume
0x0017, //1. DSK6713_AIC23_RIGHTINVOl right line input channel volume
0x00d8, //2. DSK6713_AIC23_LEFTHPVOL left channel headphone volume
0x00d8, //3. DSK6713_AIC23_RIGHTHPVOL right channel headphone volume
0x0011, //4. DSK6713_AIC23_ANAPATH Analog audio path control
0x0000, //5. DSK6713_AIC23_DIGPATH Digital audio path control
0x0000, //6. DSK6713_AIC23_POWERDOWN Power down control
```

```
0x0043, //7. DSK6713_AIC23_DIGIF Digital audio interface
0x0081, //8. DSK6713_AIC23_SAMPLERATE Sample rate control
0x0001, //9. DSK6713_AIC23_DIGACT Digital interface activation
};

// Initializes BSL and generates tone

void main()
{
 DSK6713_AIC23_CodecHandle hCodec;
 int l_input, r_input, l_output, r_output;
 //initializes the board support library

DSK6713_init();
// start the codec
hCodec=DSK6713_AIC23_openCodec(0,&config);
//set codec sampling frequency
DSK6713_AIC23_setFreq(hCodec,3);
while(1)
{
// read a sample to the left channel
while(!DSK6713_AIC23_read(hCodec,&l_input));
// read a sample to the right channel
while(!DSK6713_AIC23_read(hCodec,&r_input));
// send a sample to the left channel
while(!DSK6713_AIC23_write(hCodec,l_input));
// send a sample to the right channel
while(!DSK6713_AIC23_write(hCodec,r_input));
}

// Close the codec
DSK6713_AIC23_closeCodec(hCodec);
}
```

Example 2: To design and implement FIR filter on DSK6713 board.

Procedure:

1. Connect CRO to the socket provided for LINE OUT.
2. Connect a signal generator to the LINE IN socket.
3. Switch ON the signal generator and set it to generate sine wave. (Amplitude range, 2 to 10Volts (peak-peak) and frequency range, 100Hz to 10KHz)
4. Now switch ON the DSK and bring up code composer studio on PC.
5. Create a new project (say "fir.pjt"). Project→ New. (Set project name, location and project type—executable (.out), target—TMS320C67XX, and click on

"Finish")

6. Write source program and save it with ".c" extension (say "myfir.c"). File→ New→ Source file. (Type the source program and save it as myfir.c).
7. Add source file to the project. (Project→ Add files to project→ myfir.c)
8. Add configuration files: File→New→DSP/BIOS configuration. (Here select "dsk6713.cdb" file and save it as "config1.cdb"). Now add the saved 'config1.cdb' to the project: Project→Add files to project→ (select "config1.cdb"). Note that 3 files are now added to the "Generated files" folder in the project.
9. View the contents of "config1cfg_c.c" (which is there under the "Generated files") and copy the line containing "#include "config1cfg.h"' to the source file 'myfir.c'.
10. Add the generated file "config1cfg.cmd" to the project.
11. Add library file to the project: Project→Add files to project→"dsk6713bsl.lib". (This file is available in path: c:\CCStudio_v3.1\c6000\dsk6713\lib\dsk6713bsl.lib).
12. Set the path for pre-processor: Project→Build options→preprocessor. (Here set the path under include to: c:\CCstudio_v3.1\c6000\dsk6713\include\).
13. Now, build, load and run the program.

C-program to implement FIR filter

myfir.c

```
#include "filtercfg.h"

#include "dsk6713.h"
#include "dsk6713_aic23.h"

float filter_Coeff[]={-0.015752,-0.023869,-0.018176,0.000000,0.021481,0.033416,0.026254,-0.000000,-
0.033755,-0.055693,
-0.047257,0.000000,0.078762,0.167080,0.236286,0.262448,0.236286,0.167080,0.078762,0.000000,
-0.047257,-0.055693,-0.033755,-0.000000,0.026254,0.033416,0.021481,0.000000,-0.018176,-0.023869,
-0.015752}; //LPF having cutoff freq.=1000Hz, rectangular window (these
coefficients are from MATLAB program)


static short in_buffer[100];
DSK6713_AIC23_Config config={\
0x0017, //0. DSK6713_AIC23_LEFTINVOl left line input channel volume
0x0017, //1. DSK6713_AIC23_RIGHTINVOl right line input channel volume
0x00d8, //2. DSK6713_AIC23_LEFTHPVOL left channel headphone volume
0x00d8, //3. DSK6713_AIC23_RIGHTHPVOL right channel headphone volume
0x0011, //4. DSK6713_AIC23_ANAPATH Analog audio path control
```

```
0x0000, //5. DSK6713_AIC23_DIGPATH Digital audio path control
0x0000, //6. DSK6713_AIC23_POWERDOWN Power down control
0x0043, //7. DSK6713_AIC23_DIGIF Digital audio interface
0x0081, //8. DSK6713_AIC23_SAMPLERATE Sample rate control
0x0001, //9. DSK6713_AIC23_DIGACT Digital interface activation
};
void main()
{
 DSK6713_AIC23_CodecHandle hCodec;
 uint32 l_input, l_output, r_input, r_output;
 // initialize the board support library
 DSK6713_init();
 //start the codec
 hCodec = DSK6713_AIC23_openCodec(0,&config);
 DSK6713_AIC23_setFreq(hCodec,1);
 while(1)
 {
   /* read a sampleto the left channel*/
        while(!DSK6713_AIC23_read(hCodec, &l_input));
          /* read a sample to the right channel*/
         while(!DSK6713_AIC23_read(hCodec,&r_input));
         l_output = (int16) FIR_FILTER(&filter_Coeff, l_input);
         r_output = l_output;

         while(!DSK6713_AIC23_write(hCodec, l_output));
           while(!DSK6713_AIC23_write(hCodec,r_output));
        }
        DSK6713_AIC23_closeCodec(hCodec);
        }
        signed int FIR_FILTER(float *h, signed int x)
        {
           int i=0;
                signed long output=0;
                in_buffer[0]=x; // new input at buffer[0]
                for(i=30;i>0;i--)
                in_buffer[i]=in_buffer[i-1]; //shuffle the buffer
                for(i=0;i<32;i++)
                output=output+h[i]*in_buffer[i];
                return(output);
        }
```

Frequency response observation:

1. Set the input signal to 2 V and 1 KHz, measure the output voltage and calculate the gain.
2. Vary the frequency of the input signal from 100 Hz to 1 MHz. Measure the output voltages. Calculate gain for each reading.

3. Plot the frequency response curve on a semilog graph paper with gain on the vertical axis and frequency on the horizontal axis.
4. From the frequency response curve, determine -3 dB cut off frequency and verify the result.

Frequency (Hz)	Output Voltage (V_{out})	Voltage gain $A(dB)=20\log_{10}(\lvert V_{out}/V_{in}\rvert)$

Note: The above c-program is designed to implement an FIR low-pass filter having cut-off frequency of 1000Hz. This contains the coefficients for FIR LPF which are obtained from MATLAB using fir1 function using rectangular window based design.

Example 3: To design and implement IIR filter on DSK6713 board.

Procedure:

1. Connect CRO to the socket provided for LINE OUT.
2. Connect a signal generator to the LINE IN socket.
3. Switch ON the signal generator and set it to generate sine wave. (Amplitude range, 2 to 10Volts (peak-peak) and frequency range, 100Hz to 10KHz)
4. Now switch ON the DSK and bring up code composer studio on PC.
5. Create a new project (say "iir.pjt"). Project→ New. (Set project name, location and project type—executable (.out), target—TMS320C67XX, and click on "Finish")
6. Write source program and save it with ".c" extension (say "myiir.c"). File→ New→ Source file. (Type the source program and save it as myiir.c).
7. Add source file to the project. (Project→ Add files to project→ myiir.c)
8. Add configuration files: File→New→DSP/BIOS configuration. (Here select "dsk6713.cdb" file and save it as "config1.cdb"). Now add the saved 'config1.cdb' to the project: Project→Add files to project→ (select "config1.cdb"). Note that 3 files are now added to the "Generated files" folder in the project.

9. View the contents of "config1cfg_c.c" (which is there under the "Generated files") and copy the line containing "#include "config1cfg.h"' to the source file 'myiir.c'.
10. Add the generated file "config1cfg.cmd" to the project.
11. Add library file to the project: Project→Add files to project→"dsk6713bsl.lib". (This file is available in path: c:\CCStudio_v3.1\c6000\dsk6713\lib\dsk6713bsl.lib).
12. Set the path for pre-processor: Project→Build options→preprocessor. (Here set the path under include to: c:\CCstudio_v3.1\c6000\dsk6713\include\).
13. Now, build, load and run the program.

C-program to implement IIR filter

myiir.c

```
#include "filtercfg.h"

#include "dsk6713.h"
#include "dsk6713_aic23.h"


const signed int filter_Coeff[]=
{ // 12730, -12730,12730,2767,-18324,21137 //HP 2500
 // 312,312,312, 32767,-27943,24367 // LP 800
 // 1455, 1455, 1455,32767,-23140,21735  // LP 2500
 // 9268, -9268,9268, 32767,-7395,18367  // HP 4000
  7215, -7215,7215, 32767,5039,6171     // HP 7000
};
```

// These coefficients are from MATLAB program

```
// Codec configuration settings

DSK6713_AIC23_Config config={\
0x0017, //0. DSK6713_AIC23_LEFTINVOl left line input channel volume
0x0017, //1. DSK6713_AIC23_RIGHTINVOl right line input channel volume
0x00d8, //2. DSK6713_AIC23_LEFTHPVOL left channel headphone volume
0x00d8, //3. DSK6713_AIC23_RIGHTHPVOL right channel headphone volume
0x0011, //4. DSK6713_AIC23_ANAPATH Analog audio path control
0x0000, //5. DSK6713_AIC23_DIGPATH Digital audio path control
0x0000, //6. DSK6713_AIC23_POWERDOWN Power down control
0x0043, //7. DSK6713_AIC23_DIGIF Digital audio interface
0x0081, //8. DSK6713_AIC23_SAMPLERATE Sample rate control
0x0001, //9. DSK6713_AIC23_DIGACT Digital interface activation
};
void main()
{
  DSK6713_AIC23_CodecHandle hCodec;
```

```
int l_input, l_output, r_input, r_output;
//initialize board support library
DSK6713_init();
hCodec = DSK6713_AIC23_openCodec(0,&config);
DSK6713_AIC23_setFreq(hCodec,3);
while(1)
{
  /* read a sampleto the left channel*/
        while(!DSK6713_AIC23_read(hCodec, &l_input));
          /* read a sample to the right channel*/
          while(!DSK6713_AIC23_read(hCodec,&r_input));
          l_output = IIR_FILTER(&filter_Coeff, l_input);
          r_output = l_output;
         // send sample to the left channel
         while(!DSK6713_AIC23_write(hCodec, l_output));
         // send sample to the left channel
         while(!DSK6713_AIC23_write(hCodec,r_output));
        }
// Close the codec
         DSK6713_AIC23_closeCodec(hCodec);
}
        signed int IIR_FILTER(const signed *h, signed int x1)
        {
                static signed int x[6]={0,0,0,0,0,0}; //x[n], x[n-1], x[n-2]
                static signed int y[6]={0,0,0,0,0,0}; // y[n], y[n-1], y[n-2]
                int temp=0;
                temp=(short int)x1; // copy input to temp
                x[0]=(signed int)temp; // copy input to x[stages][0]
                temp=((int)h[0]*x[0]); // B0*x[n]
                temp=temp+((int)h[1]*x[1]); //B1/2 * x[n-1]
                temp=temp+((int)h[1]*x[1]); //B1/2 * x[n-1]
                temp=temp+((int)h[2]*x[2]); //B2 * x[n-2]

                temp=temp-((int)h[4]*y[1]); //A1/2 * y[n-1]
                temp=temp-((int)h[4]*y[1]); //A1/2 * y[n-1]
                temp=temp-((int)h[5]*y[2]); //A2 * y[n-2]

                // dividing temp by coefficient A0
                temp>>=15;
                if(temp>32767)
                {
                        temp=32767;
                }
                else if(temp<-32767)
                {
                        temp=-32767;
                }
        y[0]=temp;
```

```
// Shuffle values along one place for next time
y[2]=y[1]; // y[n-2]=y[n-1]
y[1]=y[0]; // y[n-1]=y[n]

x[2]=x[1]; // x[n-2]=x[n-1]
x[1]=x[0]; // x[n-1]=x[n]
// temp is used as input next time through
return(temp<<2);
```

Plot the frequency response graph for the above program and verify the results.

Note:

1. The above c-program is designed to implement an IIR Butterworth high-pass filter having cut-off frequency of 7000Hz. Here the IIR HPF is written using MATLAB and the coefficients are first converted to its fixed-point equivalent and then copied.
2. To convert the floating point coefficients (obtained from MATLAB program) into its equivalent 16-bit fixed point representation, multiply coefficients by a factor = $2^{15}=32768$

Exercise:

1. Write c-program to generate four (choose any four frequency components below 1000Hz) different tones at the speaker out, which are controlled by the onboard DIP switch (SW3).
2. Design suitable filter to separate "Bass" component from a given music file. Implement the c-program and demonstrate the working on the DSK 6713 hardware.

Reference

1. Sanjit K. Mitra ,"Digital Signal Processing Laboratory Using Matlab", Mcgraw-Hill College, 2005.
2. D.G. Manolakis and Vinay K. Ingle, "Applied Digital Signal Processing", Cambridge University Press, 2012.
3. MATLAB help
4. CCStudio manual.