Visvesvaraya National Institute of Technology (VNIT), Nagpur

Digital Signal Processing (ECL304)

Lab Report

Submitted by :
Bipasha Parui (BT19ECE019)
Semester 4

Submitted to :
Dr. Deep Gupta and Dr. Ankit A. Bhurane
(Course Instructors)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

1 Assignment -1:
A digital filter is described by
y(n)- 2.56y(n-1) + 2.22y(n-2) - 0.65y(n-3) = x(n) + x(n-3)
where x(n) is the input and y(n) is the output. Assume all zero initial conditions 3

2 Assignment -2: 31

2.1 Q1: Generate a 10-kHz sinusoid sampled at 100 kHz. Plot four cycles of this signal. Also plot the spectrum of this signal in the interval ($-\pi, \pi$). 31

2.2 Q2: Decimate this signal by a factor of 2. Plot the time-domain signal. Also plot the spectrum of the decimated signal in the interval (-,). Comment on your results. What frequency does represent now ? 34

2.3 Q3: Upsample the original sinusoidal sequence by a factor of 3. Plot the time domain signal. Also plot the spectrum of the upsampled signal in the interval (-,). Comment on your results. What frequency does represent now? 36

3 Assignment -3: 39

3.1 Q1):Write a program to compute the N-point DFT of a sequence. Do not use the inbuilt command. Write your own routine of the DFT. 39

3.2 Q2):Use your program to find the 200-point DFT of the following sequences:
(i) x(n) = 2cos(2n/10) + cos(2n/5)
(ii) x(n) = n
Plot the magnitude response of these sequences. The horizontal axis must be scaled appropriately to represent the interval (0,). Also Comment on the validity of your results. 40

3.3 Q3: Write a program to compute the N-point inverse-DFT (IDFT) of a sequence. DO not use the inbuilt command. Test your program on the DFTs of the sequences in the previous part. 45
3.4 Q4: Write a program to compute the circular convolution of two length-N sequences via the DFT approach. Use your DFT and IDFT programs for this purpose.
Use your programs (using calling function) to find the circular convolution of the following pairs of sequences:
(i) x(n) = [1; 3; −2; 4; 7], h(n) = [3; 1; 21; −3]
(ii) x(n) = n; h(n) = (0.5)n, 0 ≤ n ≤ 10
. . . . 50
3.5 Q5: Write a program to perform circular convolution in the time domain. Test your program on the previous sequences given in part (d). 54
4 Assignment -4: 57
4.1 Q1: Implement the equation of Analog Butterworth Low Pass Filter Approximation and observe the effect of order N on the magnitude response. Plot the curves for the Bilinear Transformation and it's inverse so as to compensate the frequency warping effect. Prepare a succinct and original report of your own based these tasks. 57
4.2 Q2 - With reference to the IIR LP filter example discussed in the class, generate two sinusoids one within passband and other out of passband, add them and pass through the filter as designed in the class. Plot the input and output signals. . . 59
4.3 Q3 - Create a generic filter without inbuilt function. 61
4.4 Q4- Implement a BPF given the following specifications
wls = 0.1pi
, wlp = 0.4pi
, whs = 0.9pi ,
whp = 0.6pi 62
4.5 Conclusion: 63

Assignment -1: A digital filter is described by y(n)- 2.56y(n-1) + 2.22y(n-2) - 0.65y(n-3) = x(n) + x(n-3) where x(n) is the input and y(n) is the output. Assume all zero initial conditions

Section a): Generate the input signal x(n), which is a sinusoid of frequency 500 Hz sampled at 6 kHz.

Code:

```
% Code for input x[n] sinusoid creation
clear

F = 500; % Sinusoid frequency ( in Hz -- Cycles per second)
Fs = 6000; % Sampling Frequency ( in Hz --Samples per second )
n = 100;  % no. of samples
% At n= Fs/F= 12(i.e no.of samples required per cycle) we get a ...
    single i/p cycle
t = linspace(0,n/Fs,n);   % n/Fs = no. of seconds per 100 sample

x_n = sin(2 * pi * t * F); % input sinusoid

% coefficients in numerator(input signals in difference equation)
num = [1 0 0 1];
% coefficients in denominator(output signals in difference equation)
den = [1 -2.56 2.22 -0.65];

plot(t, x_n,'LineWidth', 2);
grid on;
xlabel('time in seconds', 'FontSize', 15);
ylabel('Amplitude ', 'FontSize', 15);
```

Observations and Discussions: It can be observed from Fig. 2 and 1, that the time period of the sinusoid is approximately

$$2x10^{-}3s$$

Conclusions: Since Frequency(F) is 500 Hz(cycles per second) and we are sampling(Fs) at 6 kHz(samples per second) , the output sinusoid repeats itself at rate of 12 samples/cycle i.e 6000 / 500 . Now if 6000 Samples take 1 second 12 samples take 12/6000 = 0.002 seconds . I.e time period of input x[n] is 0.002 seconds.

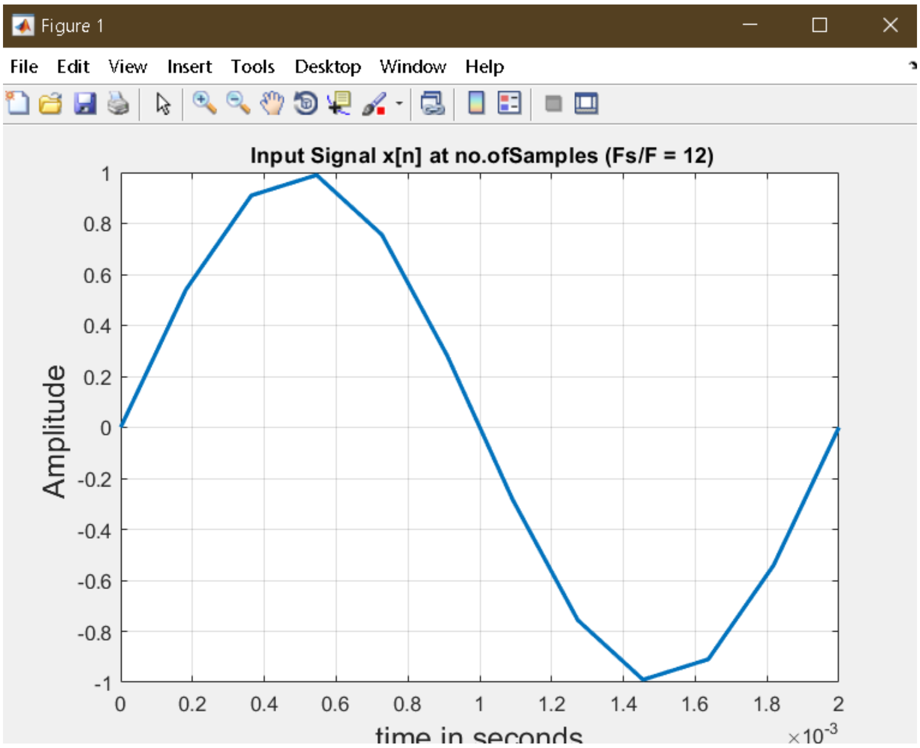

Figure 1: Sampled Sinusoid with 1 Cycle .

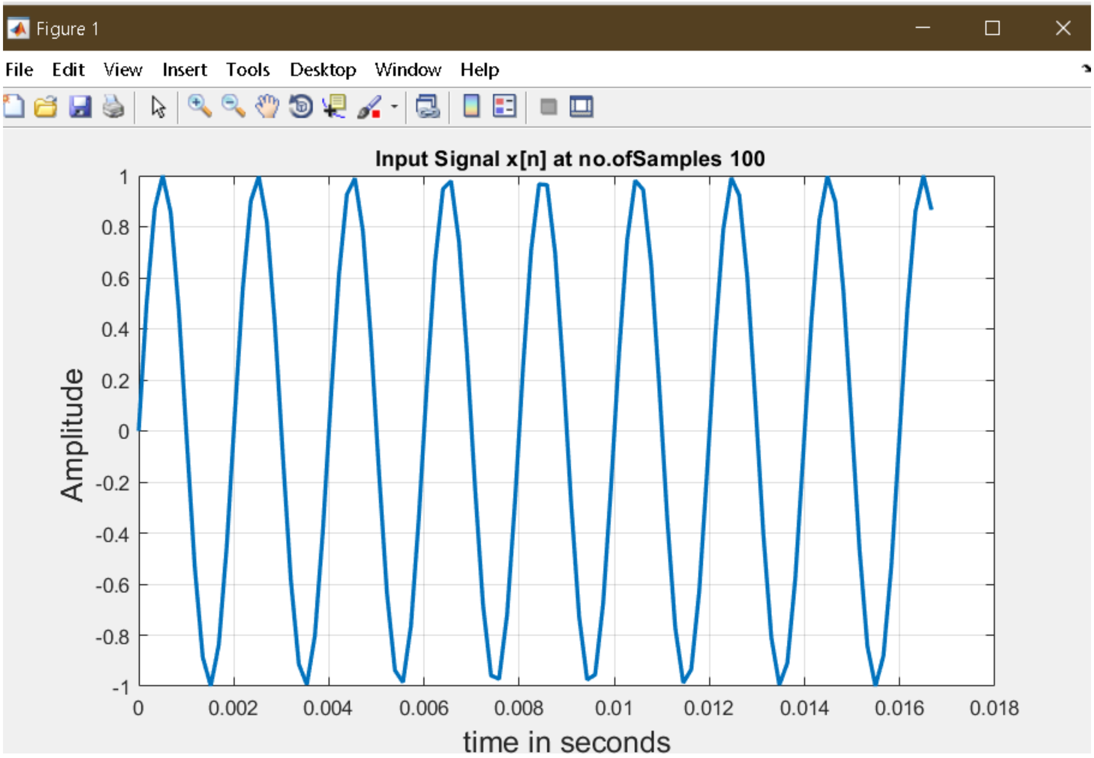

Figure 2: Sampled Sinusoid .

Section b): Compute the first four cycles of the output by directly implementing the above difference equation. Plot the input and output on the same graph.

Code:

```
% Code for output signal from given digital filter using filter()
clear
F = 500;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 6000;        % Sampling Frequency ( Hz -Samples/ second )
n = 48;           % no. of samples
% 4 cycles  =  12X4 = 48 samples
t = linspace(0,n/Fs,n);
x_n = sin(2*pi*t*F);      % input sinusoid
y_n = zeros(1,n);         % Defining Output leny[n] = lenx[n]
y_n(1:3) = [0 0 0];       % Given Zero initial conditions

% using loops to get value for each y[k] k E (4,5..48)
for k = 4:n
    y_n(k) = x_n(k)+ x_n(k-3) + 2.56*y_n(k-1) - 2.22*y_n(k-2) + ...
        0.65*y_n(k-3);
end

plot(t,y_n,'LineWidth',2)
grid on
hold on
plot(t,x_n,'LineWidth',2)
title("Input x[n],Output y[n] without 'Filter' function ...
    (no.ofsamples = 48)");
xlabel("time (t)in seconds");
ylabel("Amplitude");

legend("y[n]","x[n]");
```

Observations and Discussions: 2 points can be observed from Fig. 3
1.Though the input signal is periodic , the output signal is aperiodic.
2.The amplitude of the resultant signal is higher than input signal.

Conclusions: Since it has been established that the input sinusoid x[n] is periodic at a rate of 12samples/cycle. In order to generate a signal with approximate 4 cycle we have used **12x4 = 48 samples**. However since the output is **aperiodic** we cannot define its cycle or time period .
Here the first 3 values of output signal y[n] which forms the initial conditions is 0

$$y[0] = y[1] = y[3] = 0$$

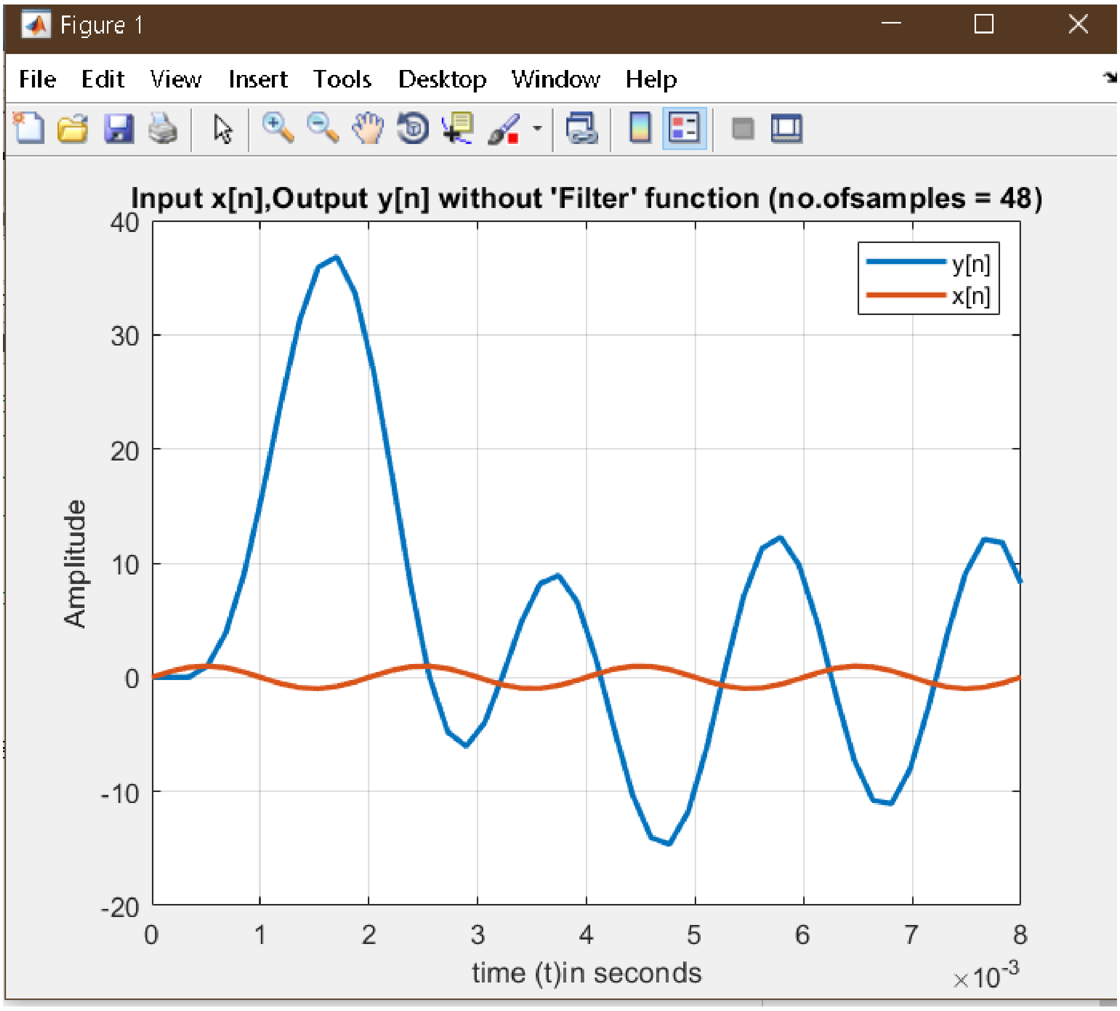

Figure 3: Output Signal from given Sinusoidal Input Signal passed through Digital filter (using loops .

. For the rest values i.e $\mathbf{y[k]}$ $\mathbf{k = 4}$ **to** $\mathbf{48}$ the code iterative implements the difference function.

Section c): Implement the above filter by using MATLAB “filter” function. Compare your results with part(b). Comment on your result.

Code:

```
% Code for output signal from given digital filter using filter()
clear
F = 500;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 6000;        % Sampling Frequency ( Hz -Samples/ second )
n = 48;           % no. of samples
% 4 cycles  =  12X4 = 48 samples
t = linspace(0,n/Fs,n);
x_n = sin(2*pi*t*F);    % input sinusoid

num = [1 0 0 1];                 %Coefficients of x[n-k]
den = [1 -2.56 2.22 -0.65];     %Coefficients of y[n-k]
y_n = filter(num,den,x_n); %INbuilt function to solve difference ...
    equation

plot(t, x_n,'-c','LineWidth', 2);
grid on;
title("x[n],y[n] with Filter function (n=48)");
xlabel('time(t) in seconds', 'FontSize', 15);
ylabel('Amplitude', 'FontSize', 15);
hold on;
plot(t,y_n,'-m','LineWidth',2);
grid on;
legend("x[n]","y[n]");
```

Observations and Discussions: It can be observed from 4 that the output signal obtained from using inbuilt function is **almost** same as observed in Section b) done manually. However Certain deviations can be seen. For example
1. The maximul amplitude in both -ve and +ve y direction is higher (approx 50,-15) than in output signal in previous section (approx 37,-6).
2. Due to initial condition in section B the signals remains at amplitude 0 till 3 points however in section C

Conclusions: Though slight differences the nature and shape of the output in Section b and c are same . Also here the coefficients num and den defined in the code correspond to **”feed-forward”** and **”feed-backward”** coefficients in the form

$$y[n] + \sum_{k=1}^{M} a_k y[n-k] = \sum_{k=0}^{N} b_k x[n-k]$$

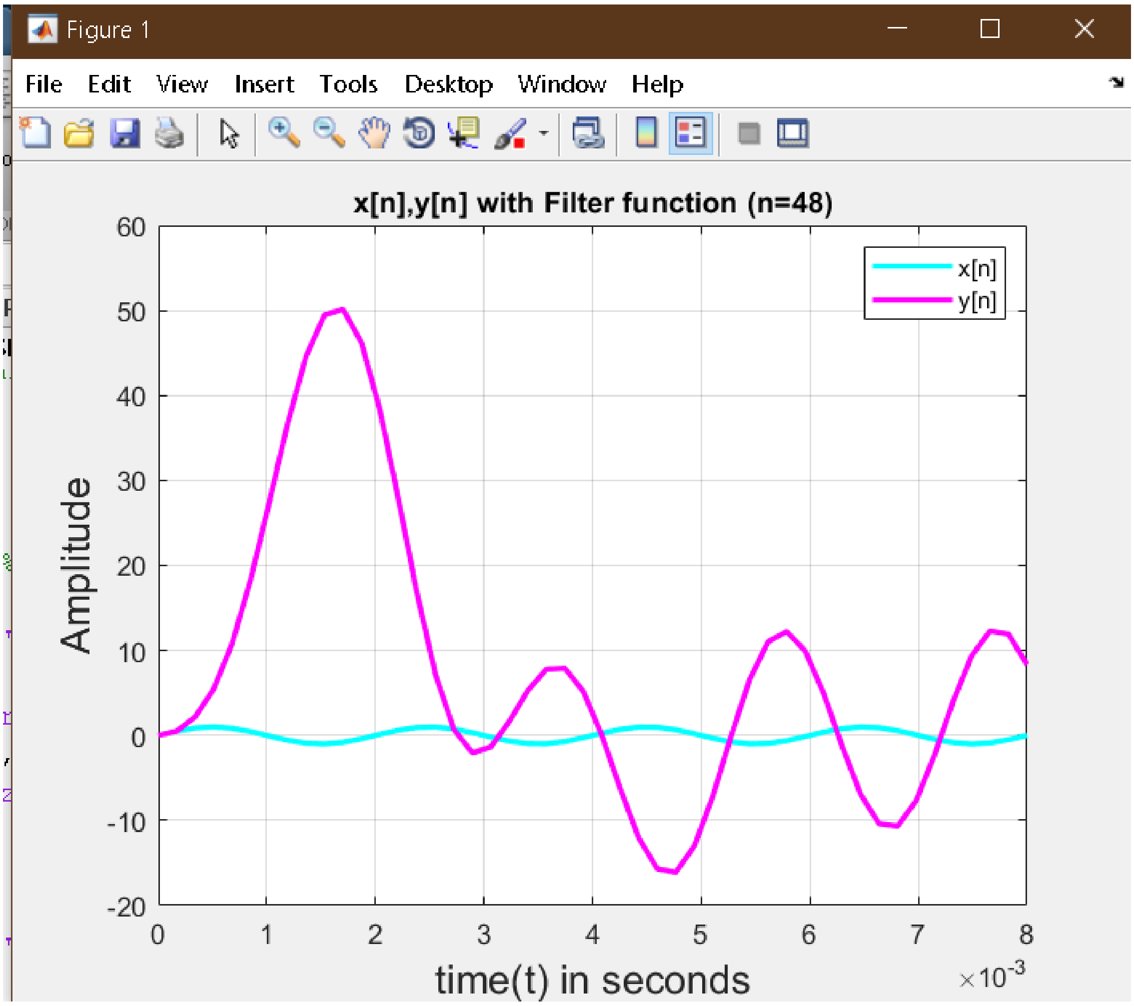

Figure 4: Output Signal from given Sinusoidal Input Signal passed through Digital filter (using MATLAB filter() .

Section d): Plot the impulse response of the filter by using MATLAB "impz" function. Comment on the results.

Code:

```
%Code for Impulse Response
num = [1 0 0 1];              %Feed forward Coefficients
den = [1 -2.56 2.22 -0.65];   %Feed Backward Coefficients

[h,t] = impz(num,den);

plot(h,'LineWidth',2);
grid on
xlabel("Samples");
ylabel("Amplitude");
title("Impulse response of digital filter");
```

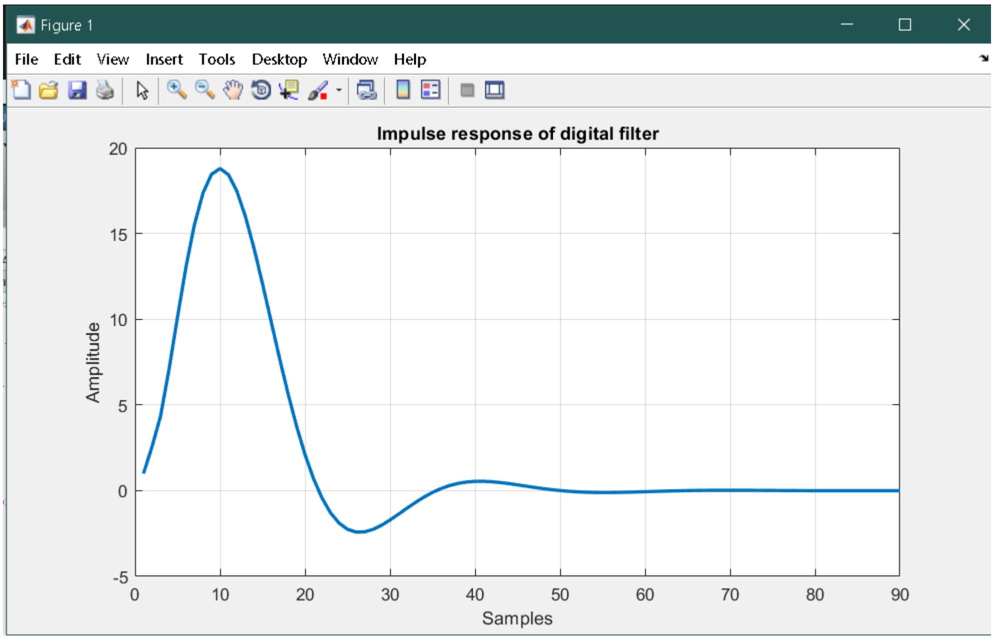

Figure 5: Impulse response of Digital filter with n = 90 samples

Observations and Discussions: We can observe from 5 that the impulse response **seems** its amplitude becomes zero after a certain time

Conclusions: Impulse response is a characterisation of the behaviour of the digital filter.

Even though it looks like a Finite Impulse response(FIR) . It is an Infinite Impulse response(IIR) i. it does not become zero after a point but continues infinitely. This

can be said by looking at the Digital Filter from which the response has been derived
y(n)- 2.56y(n-1) + 2.22y(n-2) - 0.65y(n-3) = x(n) + x(n-3)
this is in the form :

$$\sum_{j=0}^{Q} a_j y[nj] = \sum_{i=0}^{P} b_i x[n - -i]$$

and the z transform as:

$$H(z) = \frac{\sum_{i=0}^{P} b_i z^{-i}}{1 + \sum_{j=1}^{Q} a_j z^{-j}}$$

where

$$a_0/ = 1.$$

This is the general form if IIRs in contrast to FIRs which have feed backward coefficients a1,a2... = 0 and a0 = 1 . Also Clearly,

$$a_j! = 0$$

i.e the poles are not located at the origin of the z-plane indicating its an infinite Impulse Response.

Section e): Find the output signal of the filter. If we truncate the impulse response upto 32 points, Compare your results with part (b) and (c). Comment on the results.

Code:

```
% Code for output with truncated Impulse response
clear
F = 500;          % Signal Frequency (Hz-cycles per second)
n = 48;           % samples (4 cycles)
Fs = 6000;        % Sampling Frequency(Hz-samples per second)
t = linspace(0, n/Fs, n);
x_n = sin(2 * pi * t * F);
num = [1 0 0 1];               %Feed forward Coefficients
den = [1 -2.56 2.22 -0.65]; %Feed Backward Coefficients

h = impz(num,den);
x_0 = conv(x_n,h(1:32));       %IIR*x[n]

figure
subplot(3,1,1)
plot(h(1:32),"LineWidth",2)
title("Truncated Impulse(32samples) ",'FontSize', 15);
xlabel('Samples', 'FontSize', 8);
ylabel('Amplitude', 'FontSize', 8);

subplot(3,1,2)
plot(x_0,'LineWidth',2)
grid on
title("(Impulse Response)*(Input signal-4 cycle) ",'FontSize', 15);
xlabel('Samples', 'FontSize', 8);
ylabel('Amplitude', 'FontSize', 8);


subplot(3,1,3)
plot(t,x_0(1:48),'LineWidth',2)
grid on
title("Same as above but with unitoftime upto 4 input cycle(48 ...
    samples) ",'FontSize', 15);
xlabel('Time(t) in seconds', 'FontSize', 8);
ylabel('Amplitude', 'FontSize', 8);
```

Observations and Discussions: In the 3rd graph it can be observed that the curve is highly similar to that obtained in section b and c where filter wad applied manually and functionally.Here **amplitude(max -apprx 50, min-apprx -15.**

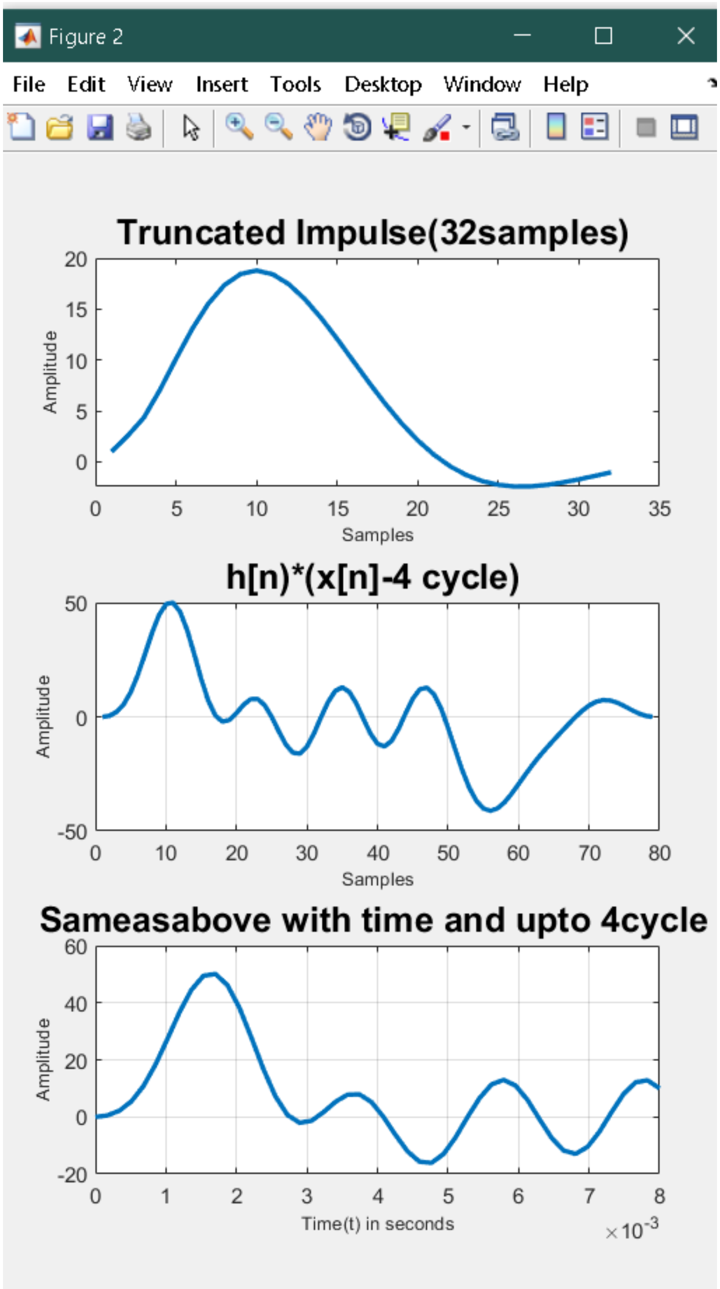

Figure 6: Output filtred with truncated Impulse response

Conclusions: The 3rd graph of 6 is just a zoom in on the first 48 points on the 2nd graph and plotted along with appropriate time units . This striking similarity between **3rd graph and graphs in section b and c** can be explained by noting that the Impulse response is a characterisation of the behaviour of the digital filter. Thus on convolving the impulse response with the input signal similar curve in terms of nature and shape has been observed.

Section f): Generate a new input signal x(n), which is a summation of two sinusoids with frequency 500 Hz and 1500 Hz sampled at 6 kHz, and Repeat part (c) on generated new signal. Comment on your results.

Code:

```
%code
clear
F1 = 500;                    % Sinusoid frequency1 ( Hz))
F2 = 1500;                   % Sinusoid frequency2 ( Hz )
Fs = 6000;                   % Sampling Frequency ( Hz )
n = 100;                     % no. of samples
t = linspace(0,n/Fs,n);

x_n1 = sin(2 * pi * t * F1);  %Sinusoid 1 at F1=500Hz Fs=6kHz
x_n2 = sin(2*pi*t*F2);       %Sinusoid 1 at F1=500Hz Fs=6kHz
x_n = x_n1+x_n2;             % Signal_1 + Signal_2

num = [1 0 0 1];             % Feed Forward Coeffs
den = [1 -2.56 2.22 -0.65];  % Feed Backward Coeffs
y_n = filter(num,den,x_n);   %Digital filterer on New signal

figure
subplot(2,1,1)
plot(t, x_n,'-r','LineWidth', 2);
title("x[n] = x_{'1'}[n] + x_{'2'}[n]",'FontSize', 15);
xlabel("Time in seconds",'FontSize', 15);
ylabel('Amplitude', 'FontSize', 15);
grid on;
subplot(2,1,2)
plot(t,y_n,'LineWidth',2);
title("Output signal y[n]","LineWidth",2);
xlabel('Time in seconds', 'FontSize', 15);
ylabel('Amplituude', 'FontSize',15);
grid on;
```

Observations and Discussions: It is observed that though the input signal has changed due to an addition of two different signals at frequency 500 and 1500 with same sampling frequency of 6kHz, the the output signal obtained after applying the digital filter is similar to that observed in the sections b,c and e .

Conclusions: The signal 1 with frequency 500Hz repeats at 12samples per cycle obtained from $Fs/F1 = 6000/500 = 12$. Similarly signal 2 with frequency 1500 repeats at 4samples per cycle $Fs/F2 = 6000/1500 = 4$. Also $4X3 = 12$ i.e the time period of signal1 is an integral multiple of (3) time period of first signal. Due to this

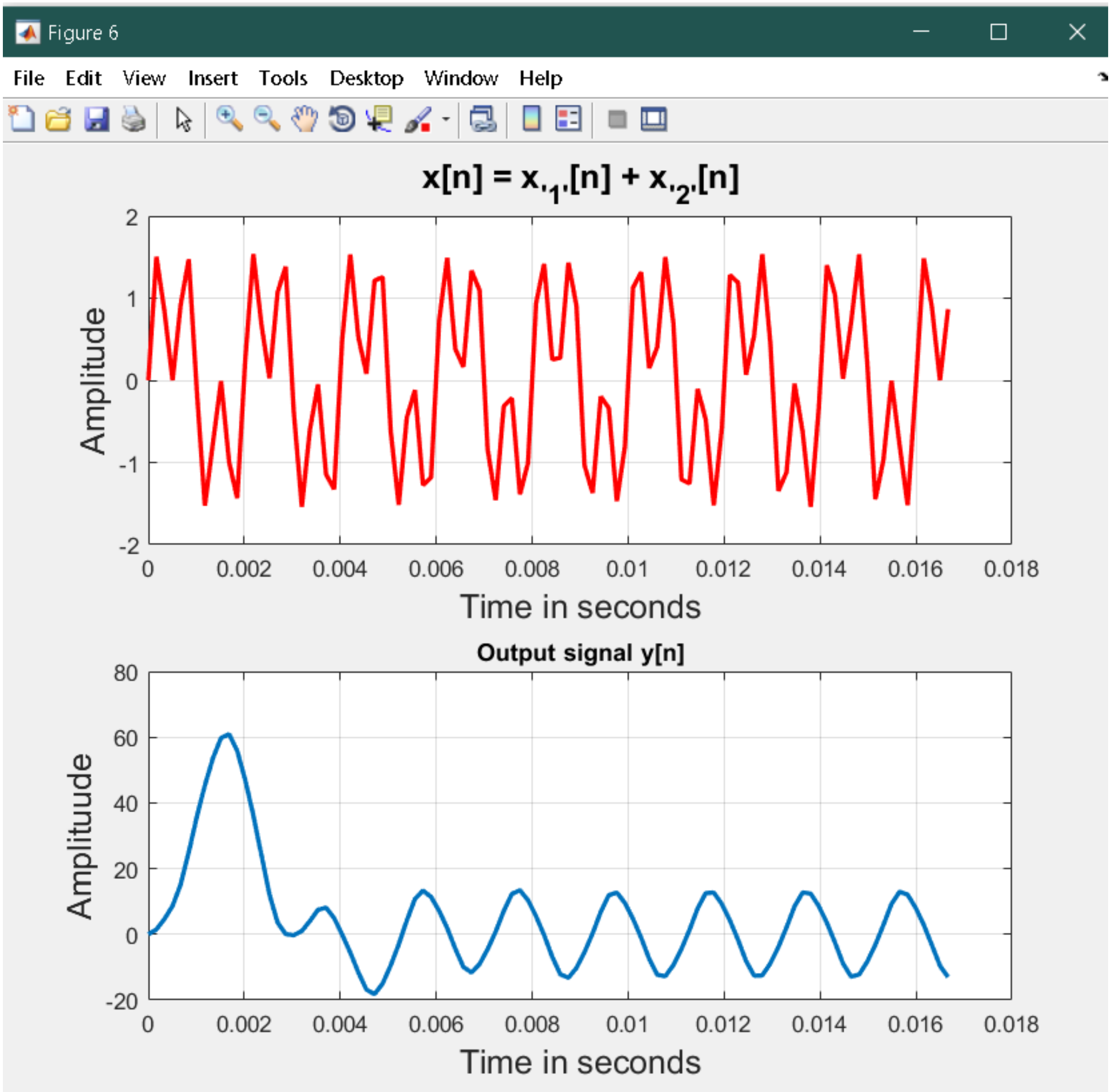

Figure 7: Output signal with Input signal

reason the resultant input signal is periodic and acts a sinusoid signal with uniform noise. Due to this the output signal is similar to previous results.

Section g): Find the frequency response of the filter and plot its magnitude and phase response of the filter.

Code:

```
num = [1 0 0 1];              %Feed Forward Coefficient
den = [1 -2.56 2.22 -0.65];   %Feed Backward Coefficient
[h,w] = freqz(num,den);       %Computing Frequency response
% h is the frequency response and w is the angular frequency (0 ...
   to pi)
figure
subplot(2,1,1)
plot((w/pi),abs(h),'LineWidth',2)
grid on
xlabel("Normalised Frequency (X\pi rad/sample) ",'FontSize',15);
ylabel("Magnitude",'FontSize',15);
title("Magnitude response",'FontSize',15);

subplot(2,1,2)
plot(w/pi,angle(h),'LineWidth',2)
grid on
xlabel("Normalised Frequency (X\pi rad/sample)",'FontSize',15);
ylabel("Phase Angle in rad",'FontSize',15);
title("Phase response",'FontSize',15);
```

Observations and Discussions: the magnitude frequency response of the provide digital filter highly has high gain between frequencies

$$0 - 0.1\pi rad/sample$$

and drastically falls such that it becomes nearly zero after

$$0.2\pi rad/sample$$

. In the phase response the phase angle keeps reducing from 0-0.1 and then suddenly changes its sign from negative to positive ,,,. Then it keeps slowly reducing until around 0.33 after which the phase angle drops and becomes almost constant.

Important: **The frequency response highly resembles frequency response of a low pass filter**

Conclusions: The frequency response is a complex output ..Hence for analysis we need to plot the phase and magnitude parts separately. Also the frequency response of filter is also the fourier transform of the impulse response plotted in section d. One can spot the correlation between the magnitude response and its phase response

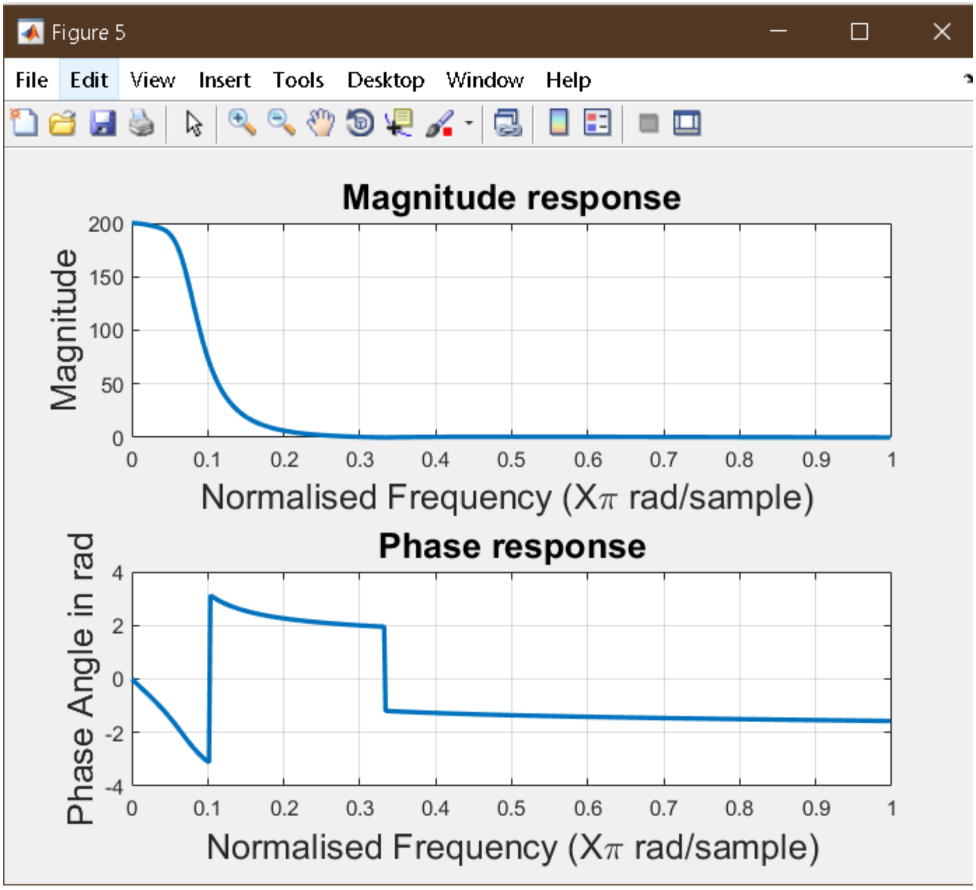

Figure 8: Frequency Response of Digital Filter
Graph a.Magnitude response . Graph b. Phase response

————————Magnitude————————Phase————-
0-0.1————-reduces, concave manner——-linearly reduces(angle -ve)
0.1-0.33——reduces, Convex manner———exponential decrease(angle +ve)
0.33————-almost constant at 0 ——————Almost constant(angle -ve)

i.e when the decrease in magnitude changes from concave to convex , the phase decrease changes its polarity from negative to positive.

Section h): Generate the input signal x(n), which is sinusoid with frequency 0.1 cycles/sample corrupted by a white Gaussian noise of approximately 5 dB.

Code:

```
%Code for White Gaussean Noise addition
clear

F = 100;        % (Cycles/second / Samples/Second) = Cycles/Sample
Fs = 1000;       % F/Fs = 100/1000 = 0.1 Cycles/Sample
n = 100;   % no. of samples| basically 10 cycles
t = linspace(0,n/Fs,n);
x_n = sin(2 * pi * t * F);    %sin(2pi*n*(1/Fs*F))

num = [1 0 0 1];              %Feed Forward Coefficients
den = [1 -2.56 2.22 -0.65];   %Feed Backward Coefficients
op = awgn(x_n,5,'measured');  % Noise induced Signal at SNR = 5dB
SNR = snr(op);
impr = filter(num,den,op);
r_i = snr(impr);

figure
subplot(3,1,1)
plot(x_n,'linewidth',2);
grid on
title("No Noise Input Signal ",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
subplot(3,1,2);
plot(op,'linewidth',2);
grid on
title("Input Signal with White Gaussean Noise(SNR = 5dB) ...
    ",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
subplot(3,1,3)
snr(op,Fs);
```

Observations and Discussions: Graph 1: Since we had taken Frequency at 100Hz and Sampling freqency at 1000 Hz. the Cycles per ratio is $100/1000 = 0.1$. The No noise signal repeats at 10 samples/cycle or in other words it has a time period of 10(samples/cycle)/1000(samples/second) = 0.01 seconds. Graph 2: After adding a noise if 5dB using awgn function the sinusoid has become distorted but we can still roughly make out the 10 noisy cycles that correspond to the 10 noise free cycle in graph 1. Graph 3: In this graph which was obtained by calculating the snr on the obtained noisy signal in graph 2 reaffirms that the SNR remained at 5.064dB.. This

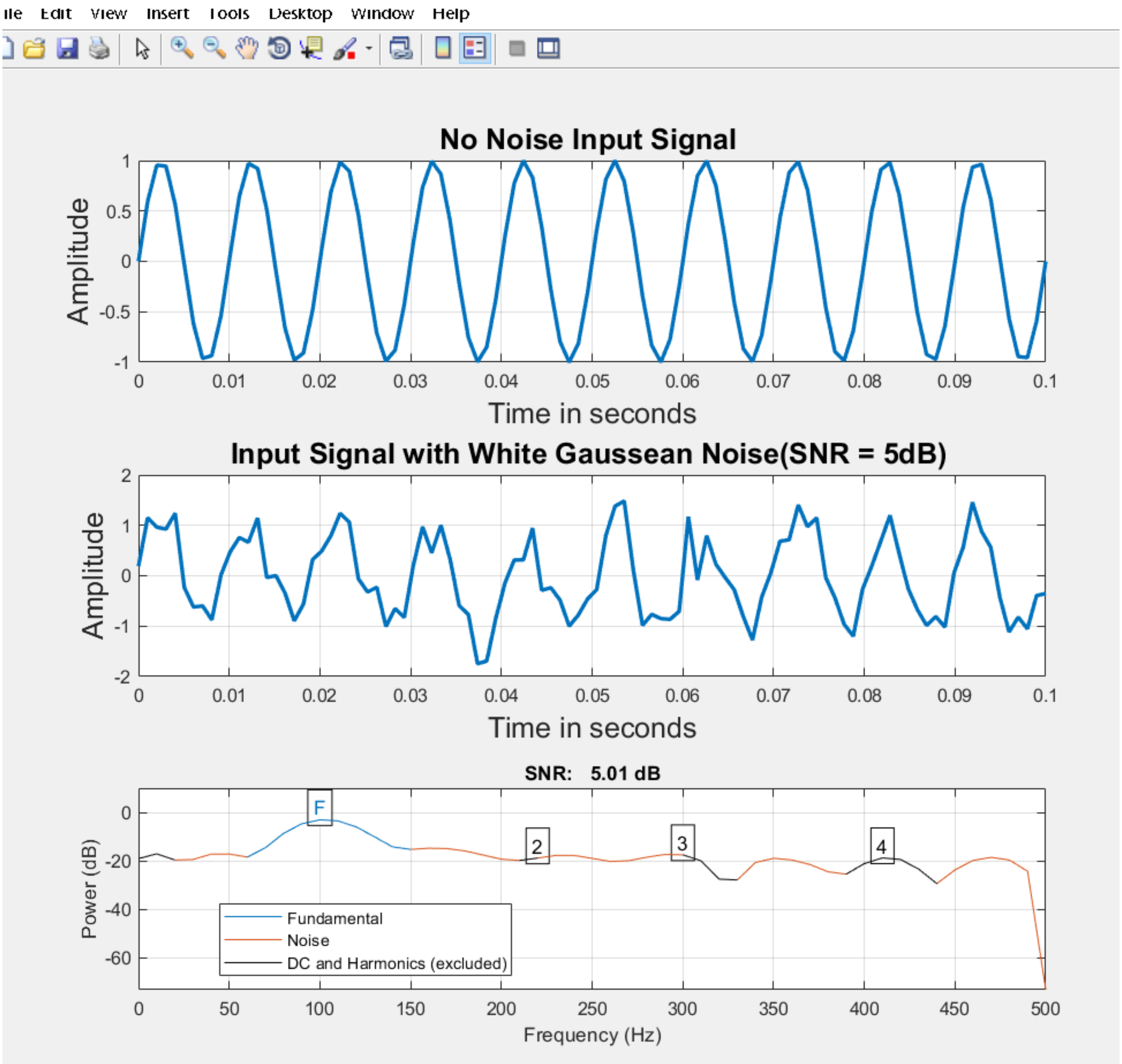

Figure 9: Graph 1. Clean Signal with 0.1 cycles/Sample
Graph 2. White Gaussean Noise added signal at SNR = 5dB.
Graph 3 Noise spectrun

graph is an inbuilt plot by calling snr() function from matlab.

Conclusions: The white gaussean noise that added is a noise which has **'White'** meaning a uniform power throughout the whole frequency band and **'Gaussean'** i.e having normal distribution with mean at 0 for each sample. This ensures that the noise added is uniform and hence though irregular we can make out the peaks and trough i.e cycles of the signal.

Section i): Filter the noisy signal generated in part (h) and compute the improvement in Signal to Noise Ratio. Also Comment on your results.

Code:

```
%Code for Improvement in SNR
clear

F = 100;        % (Cycles/second / Samples/Second) = Cycles/Sample
Fs = 1000;       % F/Fs = 100/1000 = 0.1 Cycles/Sample
n = 100;    % no. of samples| basically 10 cycles
t = linspace(0,n/Fs,n);
x_n = sin(2 * pi * t * F);   %sin(2pi*n*(1/Fs*F))

num = [1 0 0 1];              %Feed Forward Coefficients
den = [1 -2.56 2.22 -0.65];   %Feed Backward Coefficients
op = awgn(x_n,5,'measured');  % Noise induced Signal at SNR = 5dB
SNR = snr(op);
improved = filter(num,den,op);
r_i = snr(improved);

figure
subplot(2,1,1)
plot(t,improved,'linewidth',2);
grid on
title("Filtered White Gaussean Noise signal",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
subplot(2,1,2)
snr(improved,Fs);
```

Observations and Discussions: The noise signal that was filtered had a SNR value of 4.98dB before going through the Filter Obtained in the variable SNR in code. After appying Digital Filter , the new SNR of the signal became 38.17dB . As recorded in variable ri and the noise spectrum plot obtained from its snr() function.

Conclusions: since SNR refers to signal power/Noise power , a higher SNR value would mean a relatively lower noise in the signal . Hence it can be concluded that the obtained graph is an noisy but improved signal. The improvement can be calculated as

Improvement in SNR = SNR after filter - SNR before filter
= 38.17 - 4.98 = 33.19dB

Section i): Filter the noisy signal generated in part (h) and compute the improvement in Signal to Noise Ratio. Also Comment on your results.

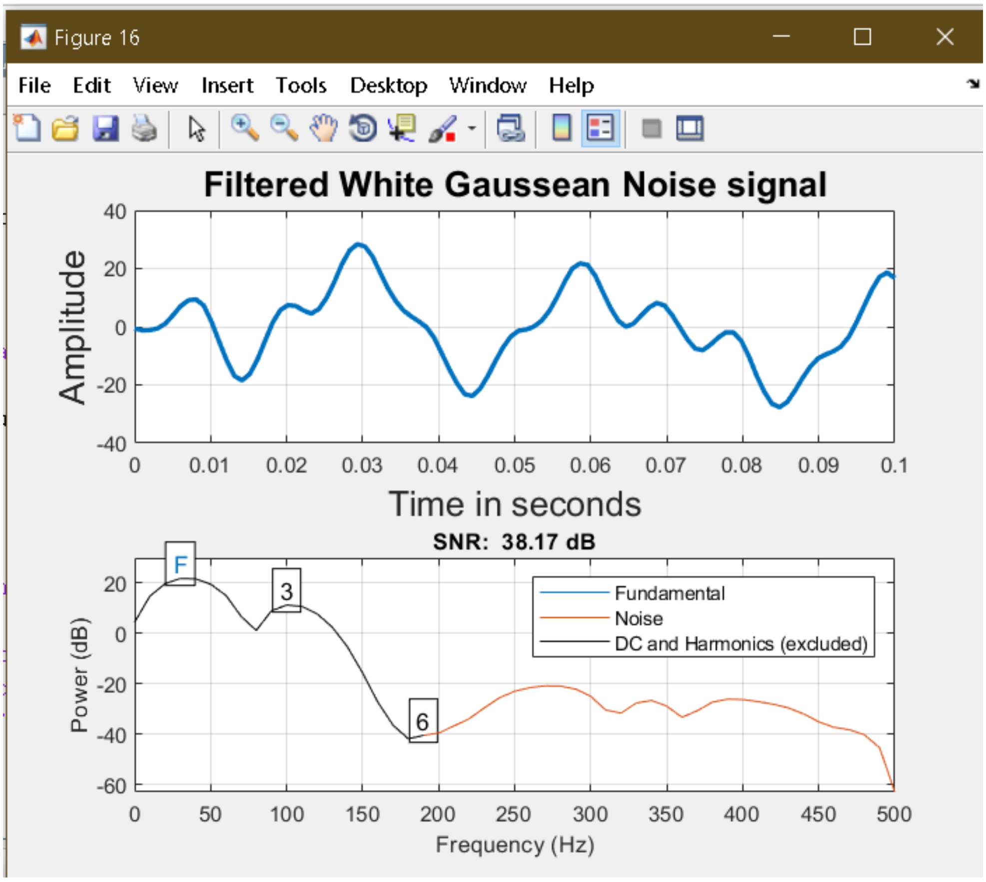

Figure 10: Graph 1. Filtered WGN signalGraph 2. Noise Spectrum

Code:

```
% Code for spectrum of signal (at 4 cycles , perfectly sampled
clear
F = 10000;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 100000;        % Sampling Frequency ( Hz -Samples/ second )
n = 4*Fs/F;           % 4(no. of cycles)* (no. of samples/cycle)
t = 0:n-1;   %time axis ,  1/Fs is the sampling period
x_n = sin(2*pi*t*F*1/Fs);       % sampled signal

fft_x_n = fft(x_n)/length(x_n)*2; %finding the fft, normalising ...
    value for correct amplitude range
fft_x_n = fftshift(fft_x_n);      % shifting fft to make it ...
    symmetric about 0
Freq = ((-n/2:((n/2)-1))/n)*2*pi;  % defining frrquency scale

figure
subplot(3,1,1)
stem(t/Fs,x_n,'linewidth',2)
title("Sampled Sinusoid - 4 cycles",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(t/Fs,x_n,'-g','linewidth',1.5)
grid on
subplot(3,1,2)
stem(Freq,angle(fft_x_n),'linewidth',2);
title("Phase spectrum - Frequency Spectrim of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Phase (in rad)",'FontSize',15);
hold on
plot(Freq,angle(fft_x_n),'linewidth',1.5);
grid on
subplot(3,1,3)
stem(Freq,abs(fft_x_n),'linewidth',2);
title("Amplitude spectrum - Frequency Spectrum of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(Freq,abs(fft_x_n),'linewidth',1.5);
grid on
```

Observations and Discussions: We can observe that the sinusoid has a rate of 10 samples/cycle and hence 4 cycles consist of samples at s = 0 to 39 . We can also see that there is no aliasing happening int he sinusoid . Also the amplitude/Magnitude spectrum of the signal has 2 peaks at around w =-0.6 ans +0.6 i.e symmetricallly

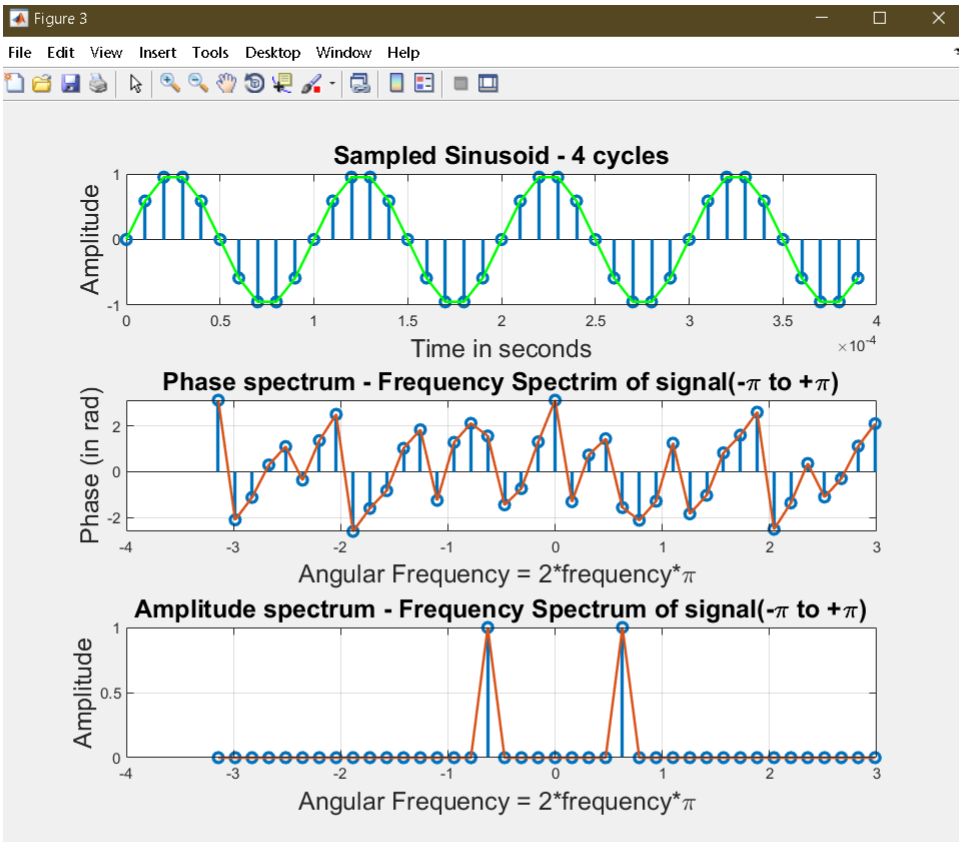

Figure 11: Signal Spectrum with Sampling

about the origin

Conclusions: Here the range -pi to +pi refers to the angular frequencies of the fourier transform of the sampled signal. Also the sampling frequency is adequate enough to represent the sinusoial shape .

Code:

```
% Code for input x[n] sinusoid creation
clear
F = 10000;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 100000;        % Sampling Frequency ( Hz -Samples/ second )
n = 4*Fs/F;         % 4(no. of cycles)* (no. of samples/cycle)  =40

t2 = 0:n-1;         %tiime axis with samples taken at 0,1,...39 = ...
    40 samples
x_n2 = sin(2*pi*t2*F/Fs);   % input sinusoid
x_n2 = downsample(x_n2,2);  % Downsampling Signal by factor of 2
t2 = 0:n/2-1;               %redefining time axis for ...
    downsampled signal
fft_x_n2 = fft(x_n2)/length(x_n2)*2;
%finding the fft, normalising value for correct amplitude range
fft_x_n2 = fftshift(fft_x_n2);
% shifting fft to make it symmetric about 0
Freq = ((-n/4:((n/4)-1))/(n/2))*2*pi;

figure
subplot(3,1,1)
stem(t2/(Fs/2),x_n2,'linewidth',2)
title("Sampled Sinusoid - 4 cycles",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(t2/(Fs/2),x_n2,'-g','linewidth',1.5)
grid on
subplot(3,1,2)
stem(Freq,angle(fft_x_n2),'linewidth',2);
title("Phase spectrum - Frequency Spectrim of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Phase (in rad)",'FontSize',15);
hold on
plot(Freq,angle(fft_x_n2),'linewidth',1.5);
grid on
subplot(3,1,3)
stem(Freq,abs(fft_x_n2),'linewidth',2);
title("Amplitude spectrum - Frequency Spectrum of signal(-\pi to ...
    +\pi)",'FontSize',15);
```

```
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(Freq,abs(fft_x_n2),'linewidth',1.5);
grid on
```

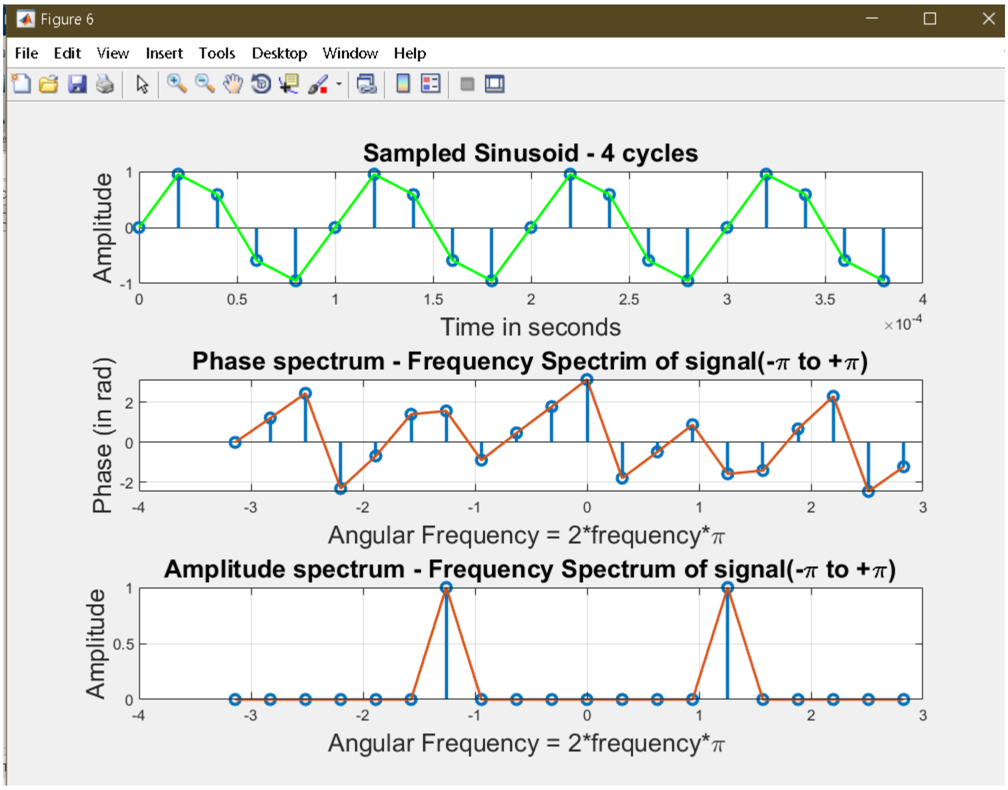

Figure 12: Signal Spectrum with downSampling

Observations and Discussions: It can be observed from Fig. 14, that the sinusoid obtaided by downsampling previous sinusoid gives a graph which is less smooth . In the magnitude specttum we again observe 2 symmetrically occuring peaks. However they are more far apart than than in the first case with Fs = 100000.

Conclusions: We can say that the sampling frequency is not sufficient to correctly describe a smooth sinusoidal signal .Also though the magnitude/amplitude spectrum seems to have been plotted from -pi to less than +pi It can be considered as a frequency spectrum of 4 cycles of sinusoid from -pi to +pi with the value at +pi being the very next point after the last point in magnitude spectrum graph and will have a value same as that at w = -pi i.e 0 . Here for downsampling has been done based on the formula .

Assignment -2:

2.1 Q1: Generate a 10-kHz sinusoid sampled at 100 kHz.Plot four cycles of this signal.

Code:

```
% Code for spectrum of signal (at 4 cycles , perfectly sampled
clear
F = 10000;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 100000;        % Sampling Frequency ( Hz -Samples/ second )
n = 4*Fs/F;           % 4(no. of cycles)* (no. of samples/cycle)
t = 0:n-1;    %time axis ,  1/Fs is the sampling period
x_n = sin(2*pi*t*F*1/Fs);      % sampled signal

fft_x_n = fft(x_n)/length(x_n)*2; %finding the fft, normalising ...
    value for correct amplitude range
fft_x_n = fftshift(fft_x_n);      % shifting fft to make it ...
    symmetric about 0
Freq = ((-n/2:((n/2)-1))/n)*2*pi;  % defining frrquency scale

figure
subplot(3,1,1)
stem(t/Fs,x_n,'linewidth',2)
title("Sampled Sinusoid - 4 cycles",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(t/Fs,x_n,'-g','linewidth',1.5)
grid on
subplot(3,1,2)
stem(Freq,angle(fft_x_n),'linewidth',2);
title("Phase spectrum - Frequency Spectrim of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Phase (in rad)",'FontSize',15);
hold on
plot(Freq,angle(fft_x_n),'linewidth',1.5);
grid on
subplot(3,1,3)
stem(Freq,abs(fft_x_n),'linewidth',2);
title("Amplitude spectrum - Frequency Spectrum of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(Freq,abs(fft_x_n),'linewidth',1.5);
grid on
```

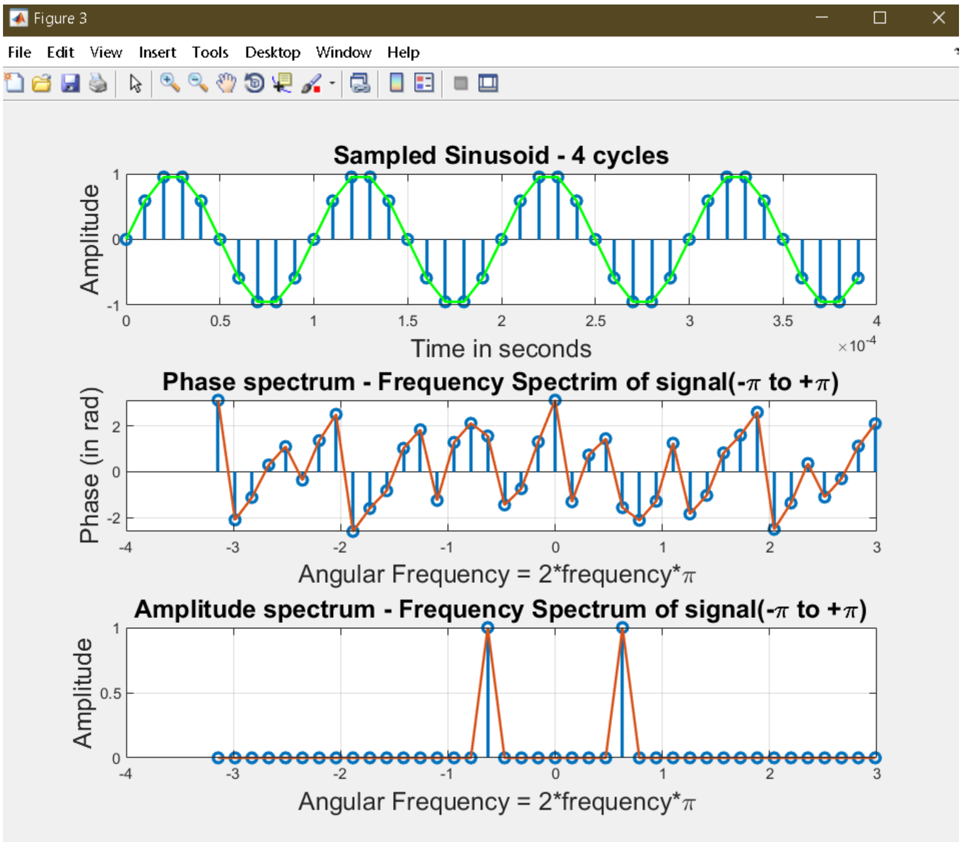

Figure 13: Signal Spectrum with Sampling

<u>Observations and Discussions</u>: We can observe that the sinusoid has a rate of 10 samples/cycle and hence 4 cycles consist of samples at s = 0 to 39 . We can also see that there is no aliasing happening int he sinusoid . Also the amplitude/Magnitude spectrum of the signal has 2 peaks at around w =-0.6 ans +0.6 i.e symmetricallly about the origin

<u>Conclusions</u>: Here the range -pi to +pi refers to the angular frequencies of the fourier transform of the sampled signal. Also the sampling frequency is adequate enough to represent the sinusoial shape .

2.2 Q2: Decimate this signal by a factor of 2. Plot the time-domain signal. Also plot t

Code:

```
% Code for input x[n] sinusoid creation
clear
F = 10000;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 100000;        % Sampling Frequency ( Hz -Samples/ second )
n = 4*Fs/F;         % 4(no. of cycles)* (no. of samples/cycle)  =40

t2 = 0:n-1;         %tiime axis with samples taken at 0,1,...39 = ...
    40 samples
x_n2 = sin(2*pi*t2*F/Fs);   % input sinusoid
x_n2 = downsample(x_n2,2);  % Downsampling Signal by factor of 2
t2 = 0:n/2-1;               %redefining time axis for ...
    downsampled signal
fft_x_n2 = fft(x_n2)/length(x_n2)*2;
%finding the fft, normalising value for correct amplitude range
fft_x_n2 = fftshift(fft_x_n2);
% shifting fft to make it symmetric about 0
Freq = ((-n/4:((n/4)-1))/(n/2))*2*pi;

figure
subplot(3,1,1)
stem(t2/(Fs/2),x_n2,'linewidth',2)
title("Sampled Sinusoid - 4 cycles",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(t2/(Fs/2),x_n2,'-g','linewidth',1.5)
grid on
subplot(3,1,2)
stem(Freq,angle(fft_x_n2),'linewidth',2);
title("Phase spectrum - Frequency Spectrim of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Phase (in rad)",'FontSize',15);
hold on
plot(Freq,angle(fft_x_n2),'linewidth',1.5);
grid on
subplot(3,1,3)
stem(Freq,abs(fft_x_n2),'linewidth',2);
title("Amplitude spectrum - Frequency Spectrum of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(Freq,abs(fft_x_n2),'linewidth',1.5);
grid on
```

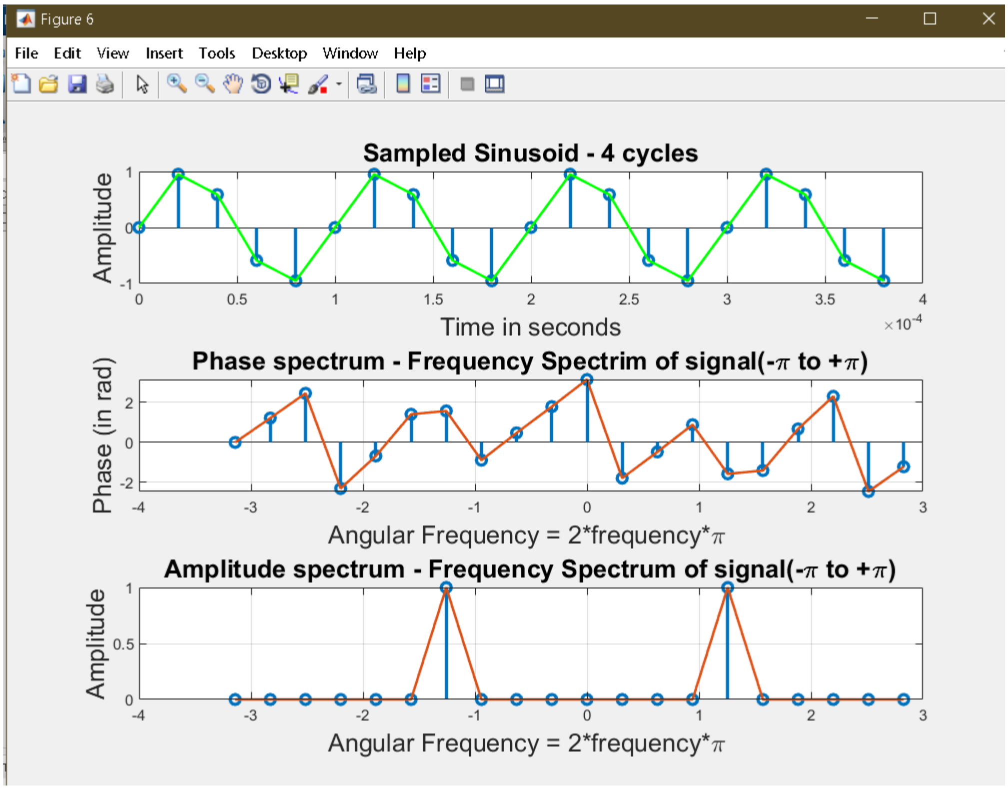

Figure 14: Signal Spectrum with downSampling

Observations and Discussions: It can be observed from Fig. 14, that the sinusoid obtaided by downsampling previous sinusoid gives a graph which is less smooth . In the magnitude specttum we again observe 2 symmetrically occuring peaks. However they are more far apart than than in the first case with Fs = 100000.

Conclusions: We can say that the sampling frequency is not sufficient to correctly describe a smooth sinusoidal signal .Also though the magnitude/amplitude spectrum seems to have been plotted from -pi to less than +pi It can be considered as a frequency spectrum of 4 cycles of sinusoid from -pi to +pi with the value at +pi being the very next point after the last point in magnitude spectrum graph and will have a value same as that at w = -pi i.e 0 . Here for downsampling has been done based on the formula .

2.3 Q3: Upsample the original sinusoidal sequence by a factor of 3. Plot the time dom

Code:

```
% Code for spectrum of upsampled signal
clear
F = 10000;          % Sinusoid frequency ( Hz - Cycles/ second)
Fs = 100000;        % Sampling Frequency ( Hz -Samples/ second )
n = 4*Fs/F;          % 4(no. of cycles)* (no. of samples/cycle)  =40

t3 = 0:n-1;          %tiime axis with samples taken at 0,1,...119 ...
    = 120 samples
x_n3 = sin(2*pi*t3*F/Fs);  % input sinusoid
x_n3 = upsample(x_n3,3);   % upsampling Signal by factor of 3
t3 = 0:n*3-1;              %redefining time axis for upsampled signal
fft_x_n3 = fft(x_n3);
%finding the fft, normalising value for correct amplitude range
fft_x_n3 = fftshift(fft_x_n3);
% shifting fft to make it symmetric about 0
Freq = ((-(3*n)/2:(((n*3)/2)-1))/(n*3))*2*pi;

figure
subplot(3,1,1)
stem(t3/(Fs*3),x_n3,'linewidth',2)
title("Sampled Sinusoid - 4 cycles",'FontSize',15);
xlabel("Time in seconds",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(t3/(Fs*3),x_n3,'-g','linewidth',1.5)
grid on
subplot(3,1,2)
stem(Freq,angle(fft_x_n3),'linewidth',2);
title("Phase spectrum - Frequency Spectrim of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Phase (in rad)",'FontSize',15);
hold on
plot(Freq,angle(fft_x_n3),'linewidth',1.5);
grid on
subplot(3,1,3)
stem(Freq,abs(fft_x_n3),'linewidth',2);
title("Amplitude spectrum - Frequency Spectrum of signal(-\pi to ...
    +\pi)",'FontSize',15);
xlabel("Angular Frequency = 2*frequency*\pi ",'FontSize',15);
ylabel("Amplitude",'FontSize',15);
hold on
plot(Freq,abs(fft_x_n3),'linewidth',1.5);
grid on
```

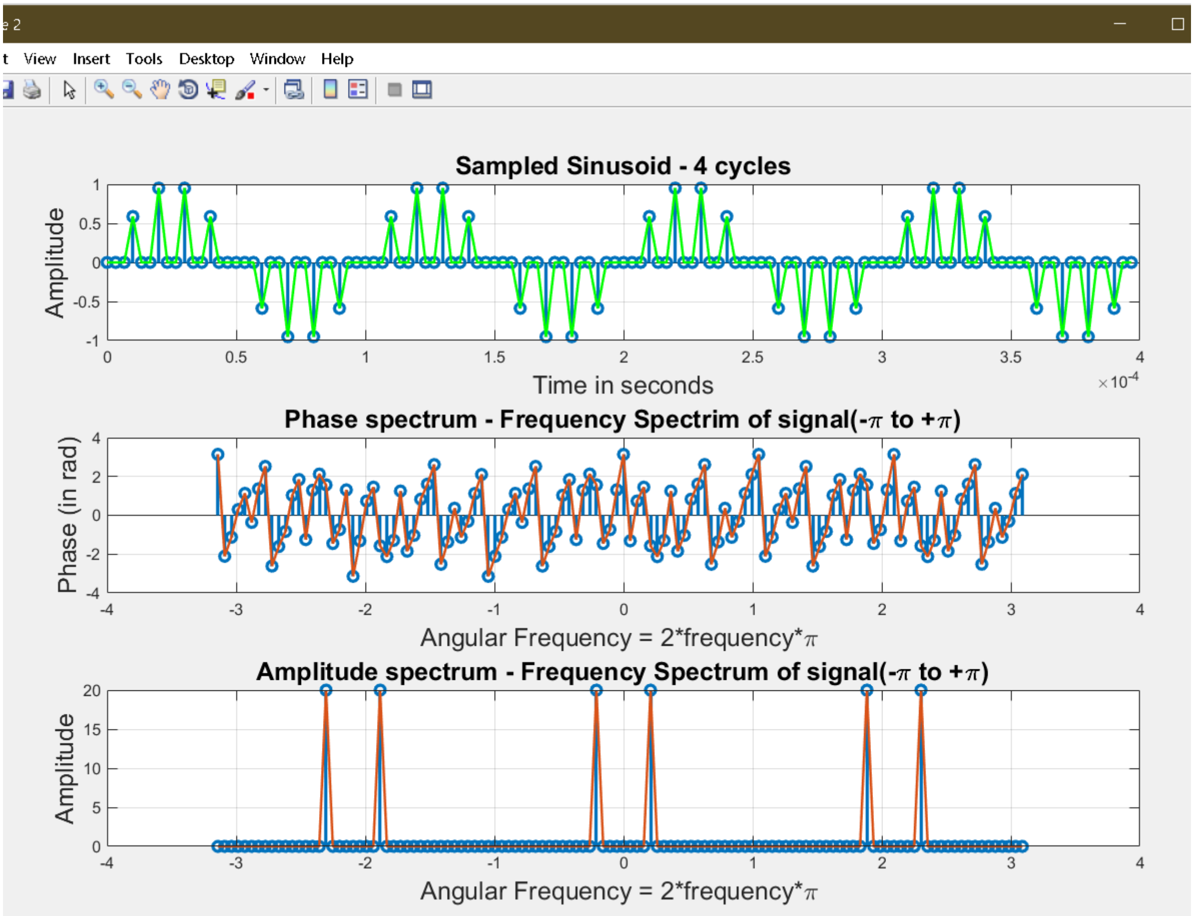

Figure 15: Signal Spectrum with upSampling

<u>Observations and Discussions</u>: It can be observed from Fig. 13, that though the function should represent a sinusoid, Due to the presence of 0 values samples in between the non zero samples, the overall plot of the signal is no longer sinusoid . Also this can be observed in the magnitude spectrum of the signal i.e the magnitude(fourier transform(signal)) that instead of 2 spikes as observed in the previous 2 cases it has 6 spikes place symmetrically across 0 in the range -pi to +pi

<u>Conclusions</u>: The reason for such a sinusoid is overampling. The signal has been very highly oversampled causing unrequired samples and deviating from sinusoidal shape . Here the range -pi to +pi thus refers to the angular frequency range of the fourier transform of a highly oversampled signal.

Assignment -3:

3.1 Q1):Write a program to compute the N-point DFT of a sequence. Do not use the

Code:

```
% Code for n-point DFT function
clc
clear

x_n = [1 2 3 4 ];    %Sample sequence for testing
x_n_nfft = nfft(x_n,length(x_n));  %Calling n point dft function ...
    (nfft)
disp(x_n_nfft);

function [x_k]=nfft(x_n,N)
len = length(x_n);
if N>len
    x_n = [x_n zeros(1,N)];
elseif N<len
    x_n = x_n(1:N);
end
for l = 1:N
    x_k(l) = 0;
    for n = 1:N
        x_k(l) = x_k(l) + x_n(n).*exp((-1j).*2.*pi.*(n-1).*(l-1)/N);
    end
end
end
```

Output: For analyzing the phenomenon,the sequence
x[n] = [1 2 3 4]
for which the output so observed was
10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 - 0.0000i -2.0000 - 2.0000i

Conclusion: The function works correctly , verified by comparing result with the inbuilt function separately

3.2 Q2):Use your program to find the 200-point DFT of the following sequences:(i) x(r

i) x(n) = 2cos(2n/10) + cos(2n/5):

Code:

```
clc
clear

n = 0:199;    % No. of samples
N = 200;      %Length of sequence
xn = 2*cos(2*pi*n/10) + cos(2*pi*n/5);  % defining function
w = pi*n/N;                            % Converting 200 samples ...
    to 0-2pi
xk = nfft(xn,N);                         % Finding 200 point DFT


figure
subplot(2,1,1)
stem(n,xn)
xlabel("samples")
ylabel("Amplitude")
title("x[n] = 2cos(2\pin/10) + cos(2\pin/5)")
grid on
subplot(2,1,2)
stem(w,abs(xk))
hold on
plot(w,abs(xk))
xlabel("omega")
ylabel("Magnitude")
title("x(k) against omega")
grid on
```

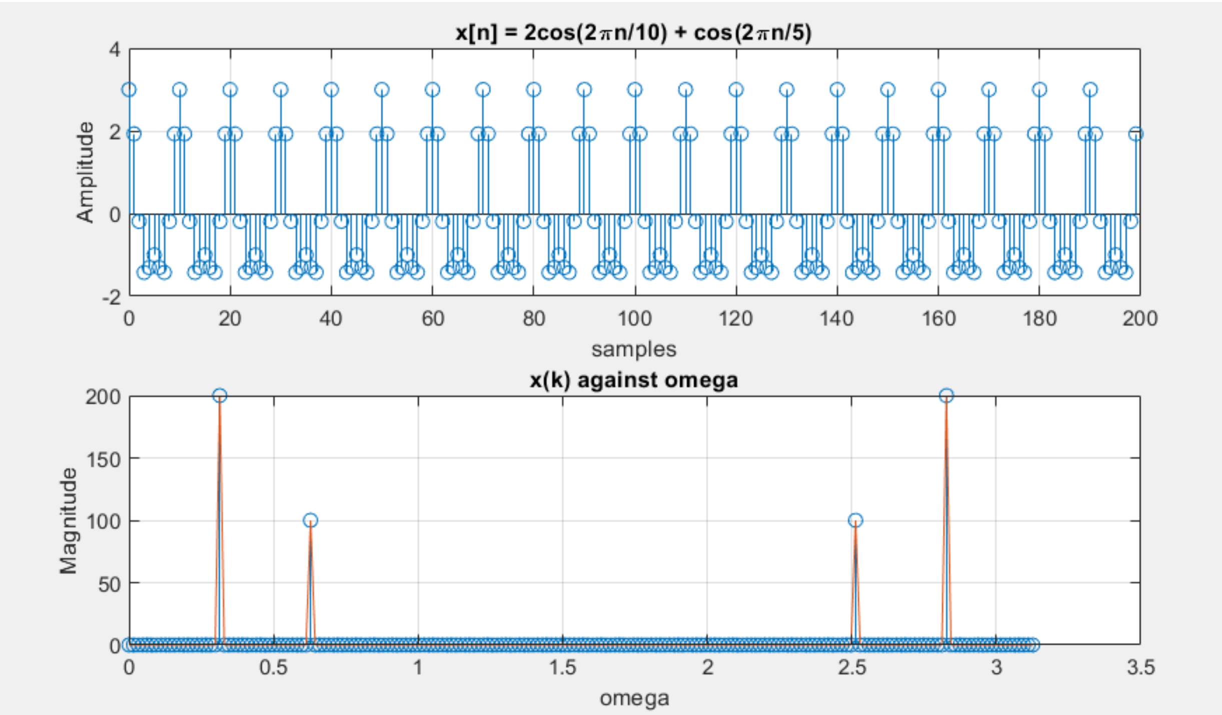

Figure 16: Magnitude response of 200-point DFT of input signal

ii) x(n) = n:

Code:

```
clc
clear

n = 0:199;    % No. of samples
N = 200;      %Length of sequence
xn =n;  % defining function
w = pi*n/N;                               % Converting 200 samples ...
    to 0-2pi
xk = nfft(xn,N);                            % Finding 200 point DFT


figure
subplot(2,1,1)
stem(n,xn,'LineWidth',1)
xlabel("samples")
ylabel("Amplitude")
title("x[n] = n")
grid on
subplot(2,1,2)
stem(w,abs(xk),'LineWidth',1)
xlabel("omega")
ylabel("Magnitude")
title("x(k) against omega")
grid on
```

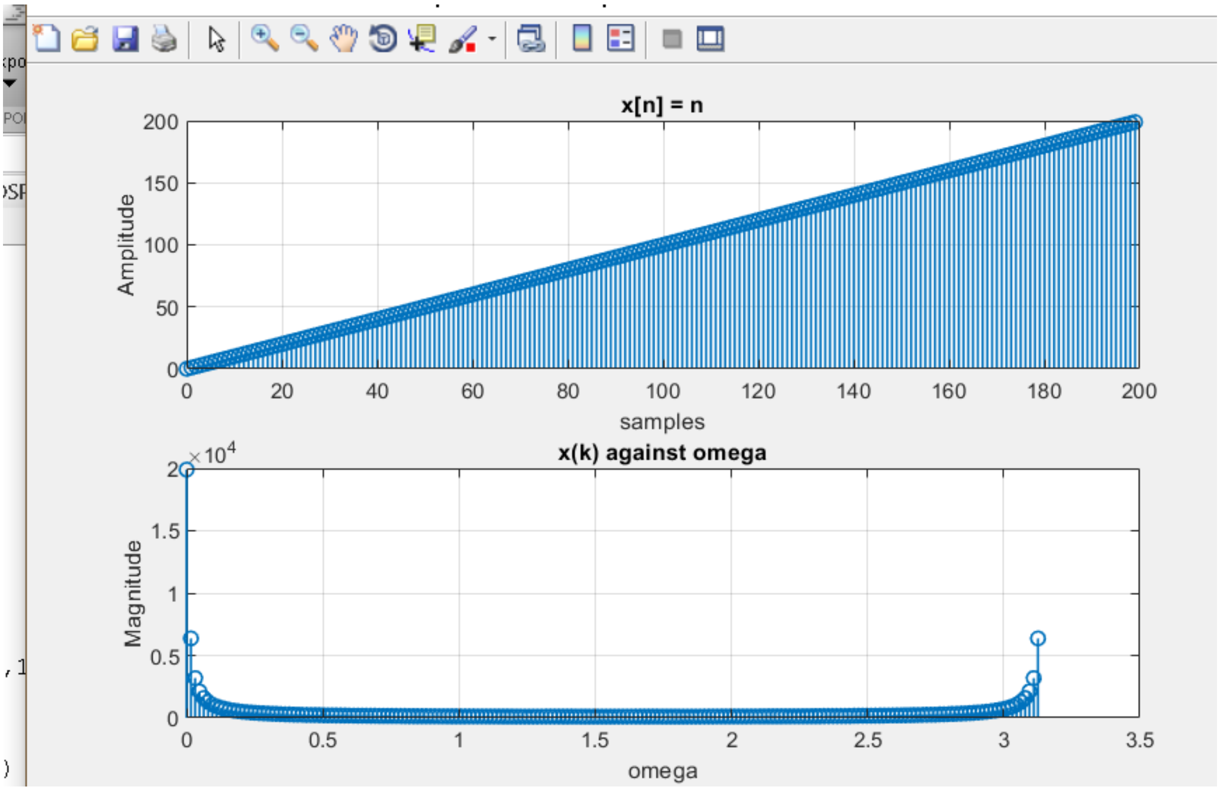

Figure 17: Magnitude response of 200-point DFT of input signal

Conclusion: The function works correctly , verified by comparing results with the result obtained from the inbuilt function separately.
i)The cos function seems to have 2 pair of symmetric peak.
ii)On the other hand the linear function first had the magnitude decreasig at the beginning upto zero and then increasing towards the end samples. However it is non identical.

3.3 Q3: Write a program to compute the N-point inverse-DFT (IDFT) of a sequence.

Inverse DFT function::

Code:

```
% Inverse DFT function
function [x_n]=infft(x_k,N)
len = length(x_k);
if N>len
    x_k = [x_k zeros(1,N)];
elseif N<len
    x_k = x_k(1:N);
end
for k = 1:N
    x_n(k) = 0;
    for n = 1:N
        x_n(k) = x_n(k) + x_k(n).*exp((1j).*2.*pi.*(n-1).*(k-1)/N);
    end
    x_n(k) = x_n(k)/N;
end
end
```

i) x(n) = 2cos(2n/10) + cos(2n/5):

Code:

```

n = 0:199;
N = 200;
x_n = 2*cos(2*pi*n/10) + cos(2*pi*n/5);
x_k = nfft(x_n,N);      % Finding DFT using created function
x_ngen = infft(x_k,N); % Finding original sequence using made ...
    IDFT function
w = pi*n/N;


figure
subplot(3,1,1)
stem(w,abs(x_k),'LineWidth',1)
xlabel("Omega")
ylabel("Magnitude")
title("DFT of x[n] = 2*cos(2*\pi*n/10) + cos(2*\pi*n/5)")
grid on
subplot(3,1,2)
plot(n,x_ngen)
xlabel("Samples")
```

```
ylabel("Amplitude")
title("IDFT of x[k] , Generated from DFT")
grid on
subplot(3,1,3)
plot(n,x_n)
xlabel("Samples")
ylabel("Amplitude")
title("Original function x[n]")
grid on
```

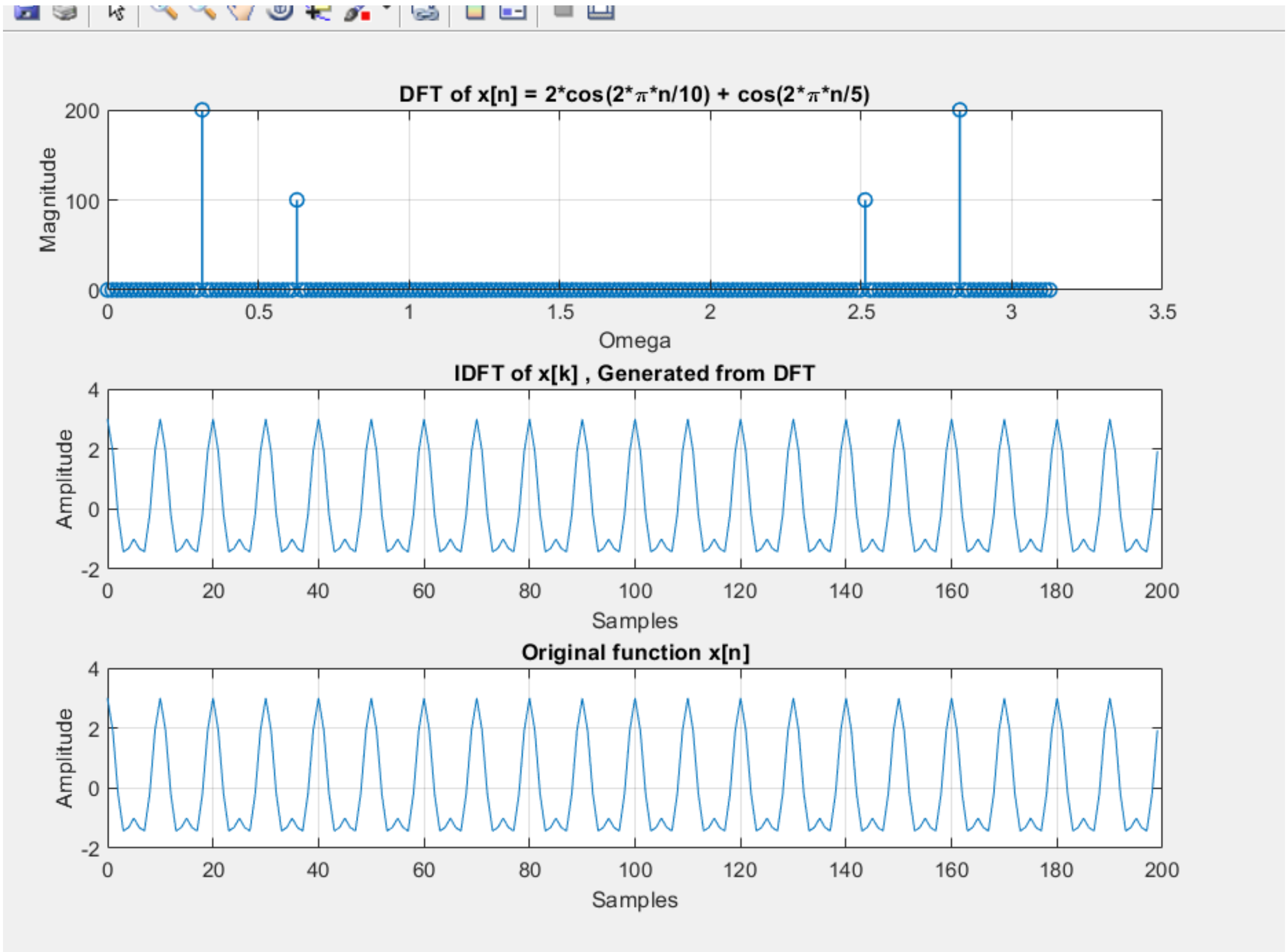

Figure 18: Inverse DFT of signal1

ii) x(n) = n:

Code:

```
clc
clear

n = 0:199;
N = 200;
x_n = n;
x_k = nfft(x_n,N);      % Finding DFT using created function
x_ngen = infft(x_k,N); % Finding original sequence using made ...
    IDFT function
w = pi*n/N;


figure
subplot(3,1,1)
stem(w,abs(x_k),'LineWidth',1)
xlabel("Omega")
ylabel("Magnitude")
title("DFT of x[n] = n")
grid on
subplot(3,1,2)
stem(n,x_ngen)
xlabel("Samples")
ylabel("Amplitude")
title("IDFT of x[k] , Generated from DFT")
grid on
subplot(3,1,3)
stem(n,x_n)
xlabel("Samples")
ylabel("Amplitude")
title("Original function x[n]")
grid on
```

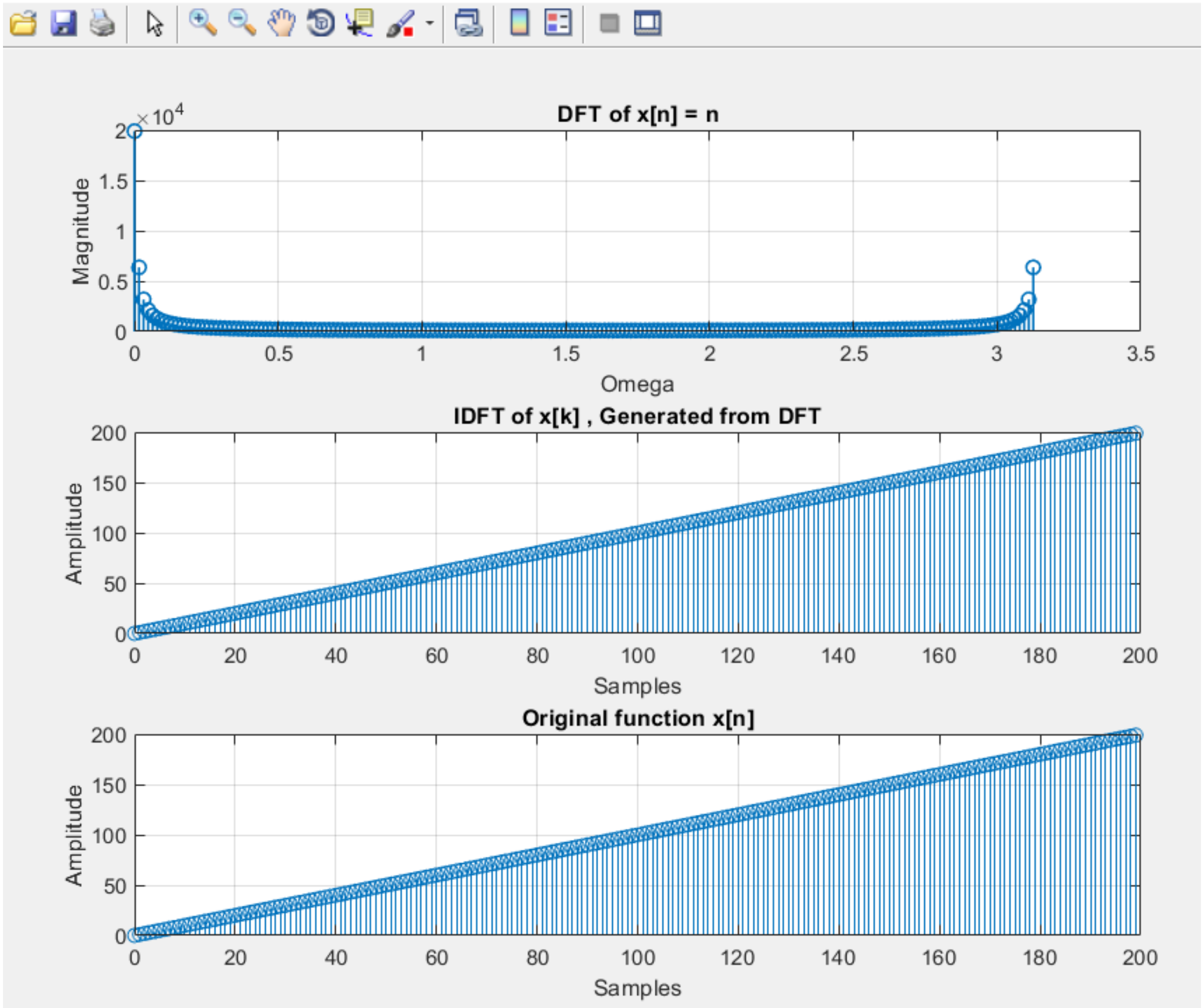

Figure 19: Inverse DFT of signal 2

<u>Conclusion</u>: The function works correctly ,this can be verified by looking at the original and the obtained idft function. They are identical. Also while plotting IDFT of X[k], the imaginary part is ignored since we know the original signal is real. Even if manually real part is not plotted, the matlab does it automatically.

3.4 Q4: Write a program to compute the circular convolution of two length-N sequenc

Circular convolution Using DFT IDFT::

$x(n) = [1; 3; -2; 4; 7]$, $h(n) = [3; 1; 21; -3]$:

Code:

```
clc;
clear;

%given sequences
%y[n]<-->Y[k] = X[k]XH[k]<--> x[n]*h[n](circular)
%('*' Convolution,'X' element wise multiplication)
x_n = [1, 3, -2, 4, 7];
h_n = [3, 1, 21, -3];

N = max(length(x_n),length(h_n));  % Finding length of output ...
    signal y = max(len x,len h)
x_n = [x_n zeros(1,N-length(x_n))];  %Padding x[n] upto N ...
    samples with zero
h_n = [h_n zeros(1,N-length(h_n))];  %Padding h[n] upto N ...
    samples with zero
X = nfft(x_n,N);                     % DFT x[n] = X[k]
H = nfft(h_n,N);                     % DFT h[n] = H[k]
Y_k = X.*H;                          % DFT y[n] = Y[k] = X[k] x H[k]
y_n = infft(Y_k,length(Y_k));        % IDFT Y[k] = y[n]
stem(y_n,'LineWidth',1);
xlabel("Samples")
ylabel("Magnitude")
title("x[n]*h[n]")
hold on
disp(y_n);
```

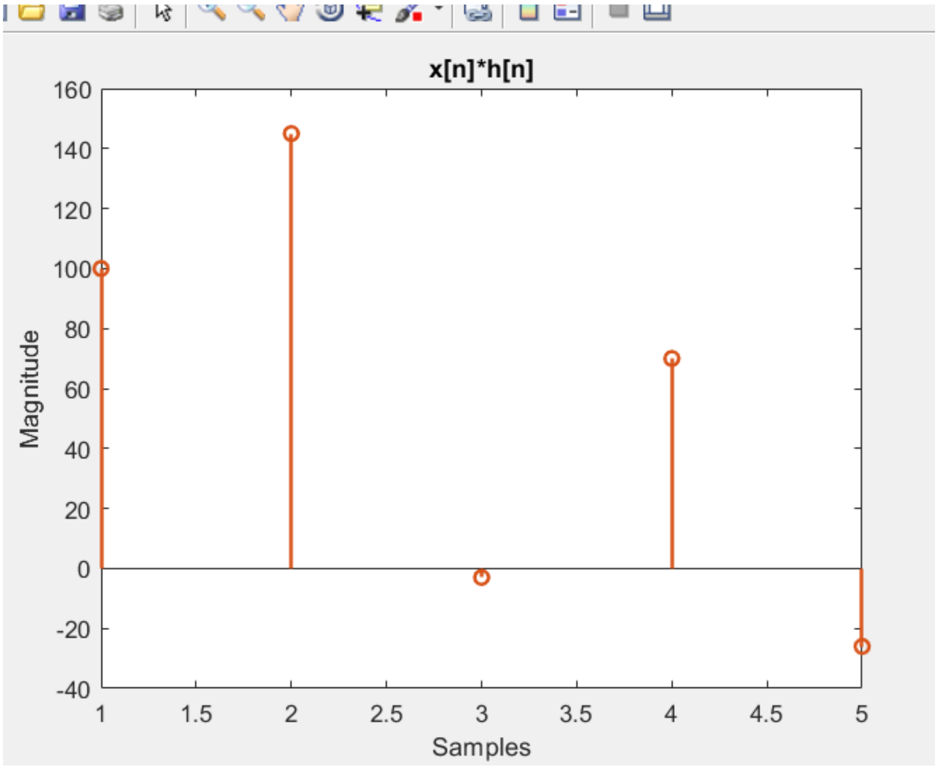

Figure 20: Circular conv using DFT iDFT, signal 1

$\mathbf{x(n) = n; h(n) = (0.5)n, 0 \leq n \leq 10}$:

Code:

```
clc;
clear;

%given sequences
%y[n]<-->Y[k] = X[k]XH[k]<--> x[n]*h[n](circular)
%('*' Convolution,'X' element wise multiplication)
n = 0:10;
x_n = n;
h_n = (0.5).^n;

N = max(length(x_n),length(h_n));  % Finding length of output ...
    signal y = max(len x,len h)
x_n = [x_n zeros(1,N-length(x_n))];  %Padding x[n] upto N ...
    samples with zero
h_n = [h_n zeros(1,N-length(h_n))];  %Padding h[n] upto N ...
    samples with zero
X = nfft(x_n,N);                     % DFT x[n] = X[k]
H = nfft(h_n,N);                     % DFT h[n] = H[k]
Y_k = X.*H;                          % DFT y[n] = Y[k] = X[k] x H[k]
y_n = infft(Y_k,length(Y_k));        % IDFT Y[k] = y[n]
stem(y_n,'LineWidth',1.5);
xlabel("Samples")
ylabel("Magnitude")
title("(x[n] = n)*(h[n] = 0.5^n)")
hold on
disp(y_n);
```

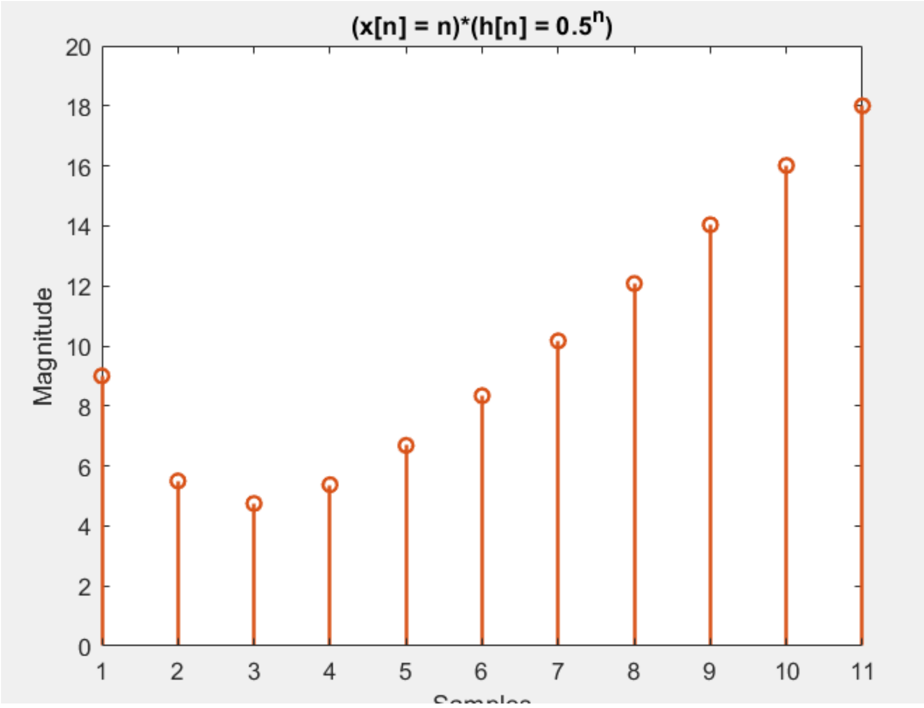

Figure 21: Circular conv using DFT iDFT, signal 2

Conclusion: The function works correctly , verified by comparing result with the inbuilt function cconv separately.

3.5 Q5: Write a program to perform circular convolution in the time domain. Test you

Circular Convolution function::

Code:

```
function [y_n] = circ_conv(x_n,h_n)
N = max(length(x_n),length(h_n));  %length of convolved signal ...
    required
x_n = [x_n zeros(1,N-length(x_n))];
h_n = [h_n zeros(1,N-length(h_n))];
y_n = zeros(1,N);
    for n=1:N
        for i=1:N
            j=n-i+1;
            if(j≤0)
                j=N+j;
            end
            y_n(n)=y_n(n)+(x_n(i)*h_n(j));
        end
    end
end
```

i) x(n) = [1; 3; −2; 4; 7], h(n) = [3; 1; 21; −3]:

Code:

```
clc;
clear;
x_n = [1, 3, -2, 4, 7];
h_n = [3, 1, 21, -3];

y_n = circ_conv(x_n,h_n);
stem(y_n,'LineWidth',1.5)
xlabel("Sample")
ylabel("Amplitude")
title("y[n] = (x[n]=[1, 3, -2, 4, 7])*(h[n] = [3, 1, 21, -3])")
```

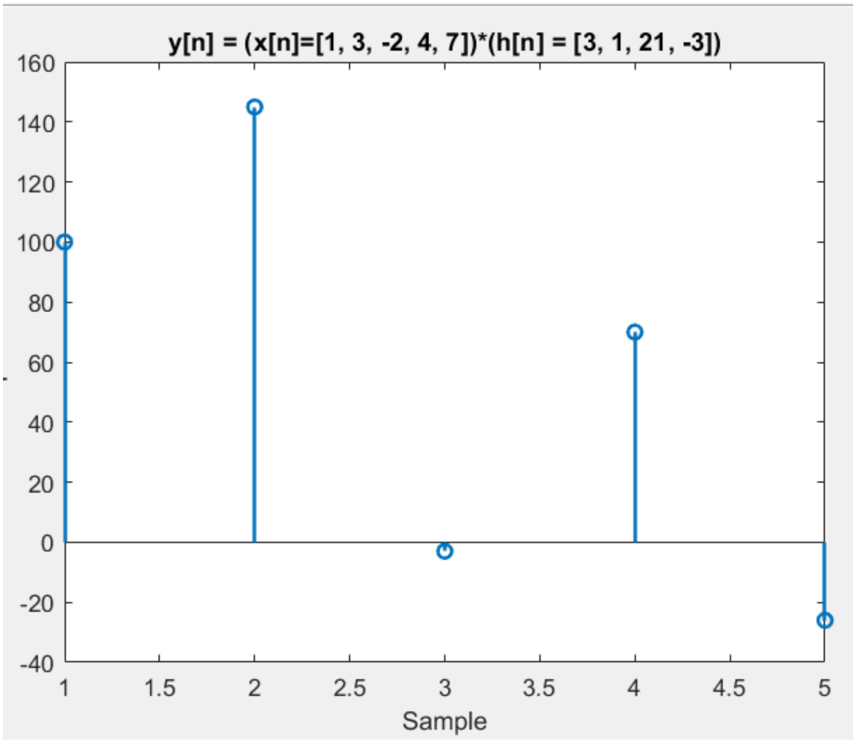

Figure 22: Circular conv signal 1

ii)x(n) = $x_n = n, h_n = (0.5).^n, 0 \leq n \leq 10$:

Code:

```
clc;
clear;
n = 0:10;
x_n = n;
h_n = (0.5).^n;
y_n = circ_conv(x_n,h_n);
stem(y_n,'LineWidth',1.5)
xlabel("Sample")
ylabel("Amplitude")
title("y[n] = (x_n = n)*(h_n = (0.5)^n)")
```

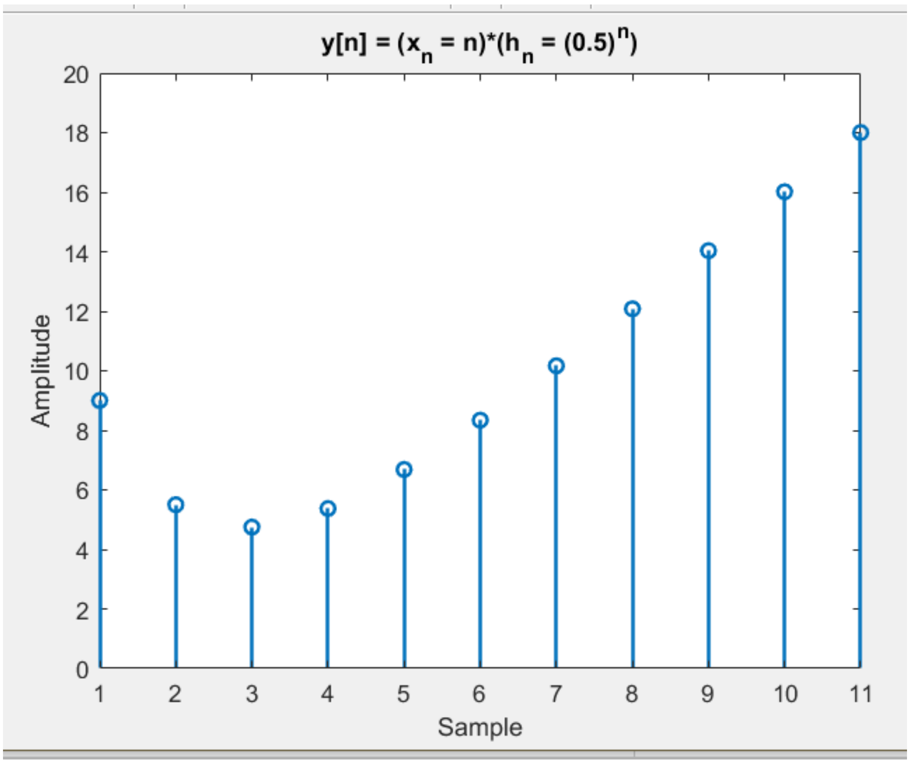

Figure 23: Circular conv signal 2

Conclusion: The function works correctly , verified by comparing result with the inbuilt function cconv separately. Also it is to be noted that though the process of obtaining the convolved output was different , the end result was the same in both the cases.

Assignment -4:

4.1 Q1: Implement the equation of Analog Butterworth Low Pass Filter Approximatio

Code:

```
%Code for task 1
%Given specifications
nn=80
w = 1:nn;
w_stop = 25:60;
w_pass = 20;
As = -25;
Ap = -3;


%Order
a = 10^(-Gs/10)-1;
b = 10^(-Gp/10)-1;
numer = log10((10^(-As/10)-1)/(10^(-Ap/10)-1);
denom = 2*log10(w_stop./w_pass);
n = ...
    ceil((log10((10^(-As/10)-1)/(10^(-Ap/10)-1))./(2*log10(w_stop./w_pass)))
omega_c = w_pass./(10^(-Ap/10)-1).^(1./2*n);

for i=1:numel(n)
    flt = (1+1i.*(w./omega_c(j)).^(2*n(i))).^-1;
    magitude = abs(flt);
    plot(linspace(0,pi,nn),magitude)
    hold on;
end
grid on;

ylabel('Magnitude')
xlabel('Normalized frequency')
```

Observations and Discussions: We can observe that the filtered output is noisy sinusoid

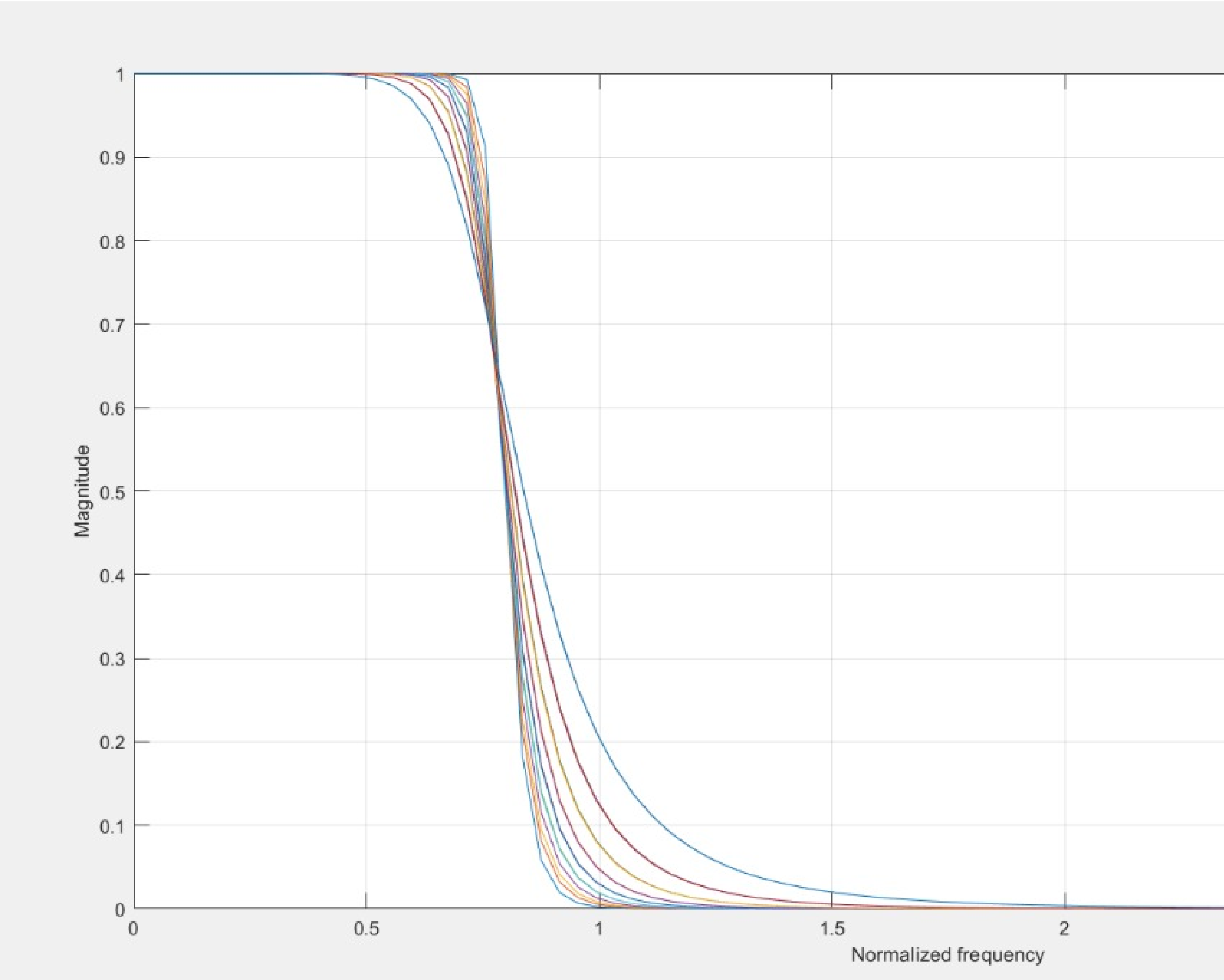

Figure 24: Results

4.2 Q2 - With reference to the IIR LP filter example discussed in the class, generate t

Code:

```
%Code for filter task
coeffy = [1, -0.919, 0.325];
coeffx = [0.102, 0.204, 0.102];


n=0:100;
x_1 = sin(0.1*pi*n);    % Sinusoid within passband
x_2 = sin(0.65*pi*n);   % Sinusoid out of passband

n=0:100;
x_add = x_1+x_2;        % adding them
y = filter( coeffx, coeffy,x_add);     % applying filter given

figure();
subplot(411);
plot(n,x_1);
grid on;
title("x1=sin(0.1\pi n");

subplot(412);
plot(n,x_2);
grid on;
title("x1=sin(0.65\pi n");

subplot(413);
plot(n,x_add);
grid on;
title("x=x1+x2");

subplot(414);
plot(n,y);
grid on;
title("filter output on x");
```

Observations and Discussions: We can observe that the filtered output is looking like noisy sinusoid.

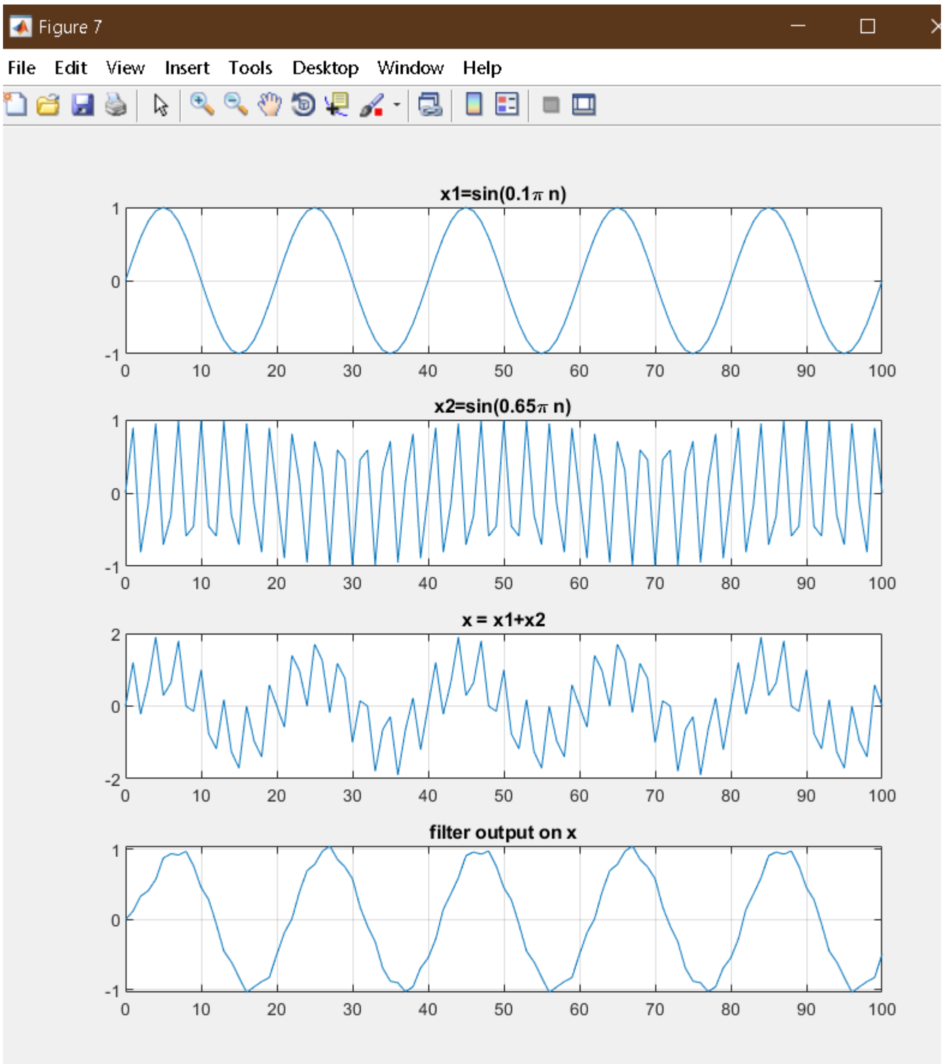

Figure 25: Results

4.3 Q3 - Create a generic filter without inbuilt function.:

Code:

```
%Code for filter task
F = 500;
Fs = 6000;
n = 48;
t = linspace(0,n/Fs,n);
x_n = sin(2*pi*t*F);
y_n = zeros(1,n);
y_n(1:3) = [0,0,0];
for k = 4:n
    y_n(k) = x_n(k) + x_n(k-3) + 2.56*y_n(k-1) - 2.2*y_n(k-2) + ...
        0.65*y_n(k-3);
end
```

Observations and Discussions: It is a self cited code from 2nd assignment

4.4 Q4- Implement a BPF given the following specificationswls = 0.1pi, wlp = 0.4pi, w

Code:

```
% Code for BPF function calculation

%Given
whs= 0.9*pi;
wls = 0.1*pi;
whp = 0.6*pi;
wlp = 0.4*pi;
As = -18;
Ap = -3;

%Doing prewarping to get ohmega values(analog freq)
osl = tan(wls/2);
opl = tan(wlp/2);
opu = tan(whp/2);
ou = tan(whs/2);

%For prototype LPF specs
o_pass= 1;
o_stop = (osu^2-opl*opu)/(osu*(opu-opl));

%Finding N = order
N = ...
    ceil((log10((10^(-As/10)-1)/(10^(-Ap/10)-1))/(2*log10(o_stop/o_pass)));

%According to this
s0 = -1;

%omega c
ohm_c = o_stop/(0^(-As/10)-1)^0.5;

%frequencies cutoff
B = opu - opl;
omega_o = (opu*opl)^0.5;
B = 0.5*((ohm_c*B)^2 + 4*omg_o^2)^0.5;
A = ohm_c*B*0.5;
omega_c1 = -A+B;
omega_c2 = A+B;

s = -1
%getting H_s
s_fin = (s^2 + omega_o^2)/ s(omega_c1 - omega_c2)
H_s = 1/s_fin +1
% Solving H_s
coeff_num = [0.77587, 0, -0.77587]/.(2.77587)
coeff_den = [2.77587,0,1.22413]/.(2.77587)
```

```

freqz(coeff_num , coeff_den)

t = [0:0.05:2*pi]
x = sin(t);
Y = filter(coeff_num,coeff_den,x)
```

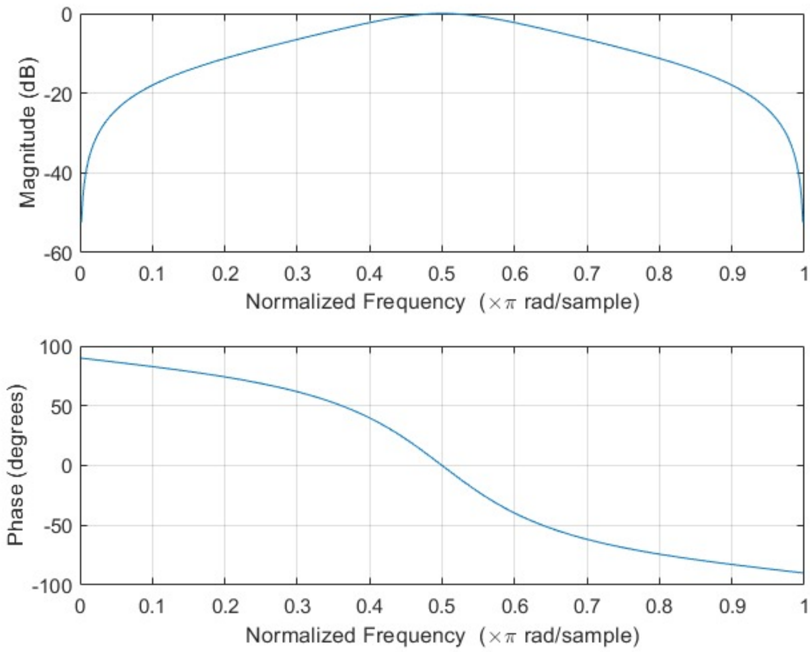

Figure 26: Results

Observations and Discussions: We observe the given frequency response of coefficients.

4.5 Conclusion:: On plotting the frequency response of obtained coefficients of the filter we see that band pass filter is formed successfully