Bangladesh University of Engineering and Technology

Department of Electrical & Electronic Engineering

Course No.: EEE 312
Course Title: Digital Signal Processing I Laboratory

Experiment No.: 02

Assignment 4:

Frequency domain analysis of DT signals and systems

Date of Performance: Jan 01, 2023

Date of Submission: Jan 09, 2023

Prepared by:

Tasmin Khan
Student ID: 1906055

Partner: Tasnimun Razin
Student ID: 1906044

Section: A2
Level:3, Term:1

Problem 1

Question:

1. Consider the filters given below. For each of them, the poles and zeroes have been pointed out. Comment on whether the filters are **low-pass, high-pass, bandpass, or notch**.

   i. $H_1(z)$: poles $[-0.25 \pm j0.74, -0.18 \pm j0.4, -0.16 \pm j0.13]$,
   zeros $[1]^{\times 6}$.

   ii. $H_2(z)$: poles $\left[0.86e^{\pm j22.66^\circ},\ 0.67e^{\pm j10.48^\circ}\right]$,
   zeros $\left[1e^{\pm j76.35^\circ},\ 1e^{\pm j124.45^\circ}\right]$.

   iii. $H_3(z)$: poles $[0.16 \pm j0.96,\ 0.35 \pm j0.81,\ 0.57 \pm j0.8]$,
   zeros $[\pm 1,\ 0.08 \pm j0.99,\ 0.64 \pm j0.77\ ]$.

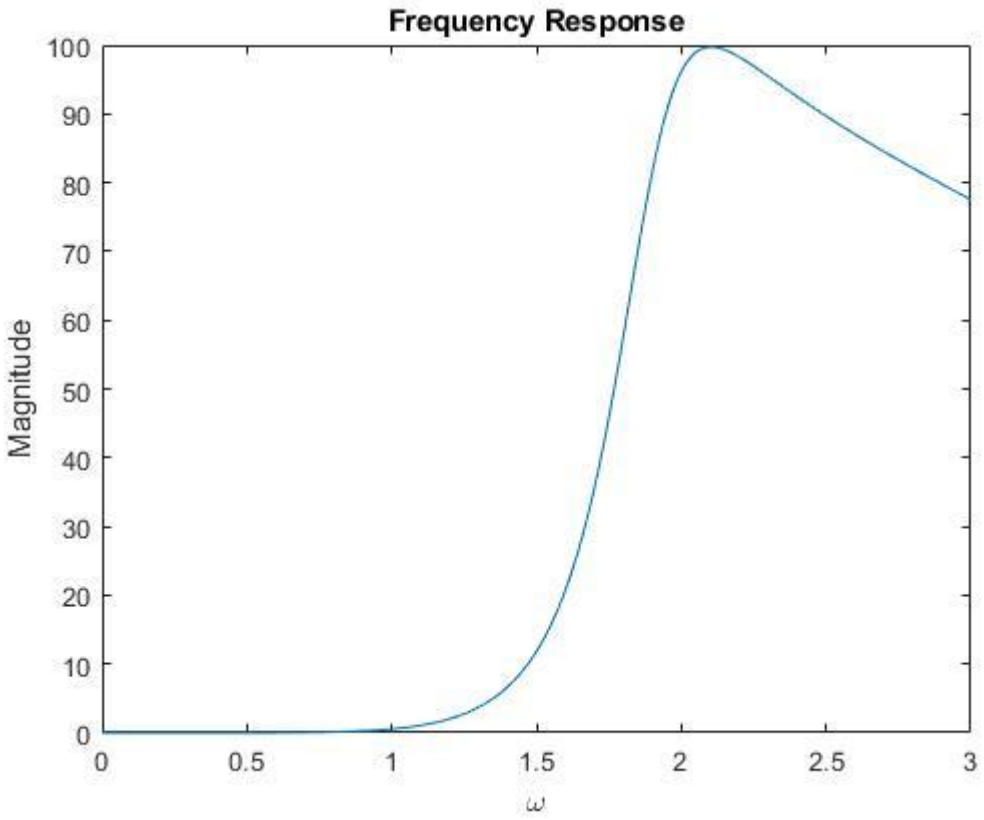
   iv. $H_4(z)$: poles $\left[0.98e^{\pm j35^\circ}\ \right]$, zeros $\left[1e^{\pm j35^\circ}\right]$.

Code:

```
clc;
clear all;
close all;

%Poles
p1 = -0.25 + 0.74*i;
p2 = conj(p1);
p3 = -0.18 + 0.4*i;
p4 = conj(p3);
p5 = -0.16 + 0.13*i;
p6 = conj(p5);

%Zeros
z1 = 1;

num1 = conv([1, -z1], [1, -z1]);
num2 = conv(num1, [1, -z1]);
num3 = conv(num2, [1, -z1]);
num4 = conv(num3, [1, -z1]);
num = conv(num4, [1, -z1]);

den1 = conv([1, -p1], [1, -p2]);
den2 = conv(den1, [1, -p3]);
den3 = conv(den2, [1, -p4]);
den4 = conv(den2, [1, -p5]);
den = conv(den2, [1, -p6]);
```

```
[h,w] = freqz(num, den);
plot(w, abs(h));
title('Frequency Response');
xlabel('\omega');
ylabel('Magnitude');
xlim([0 3]);
```

Plot for problem 1(i):



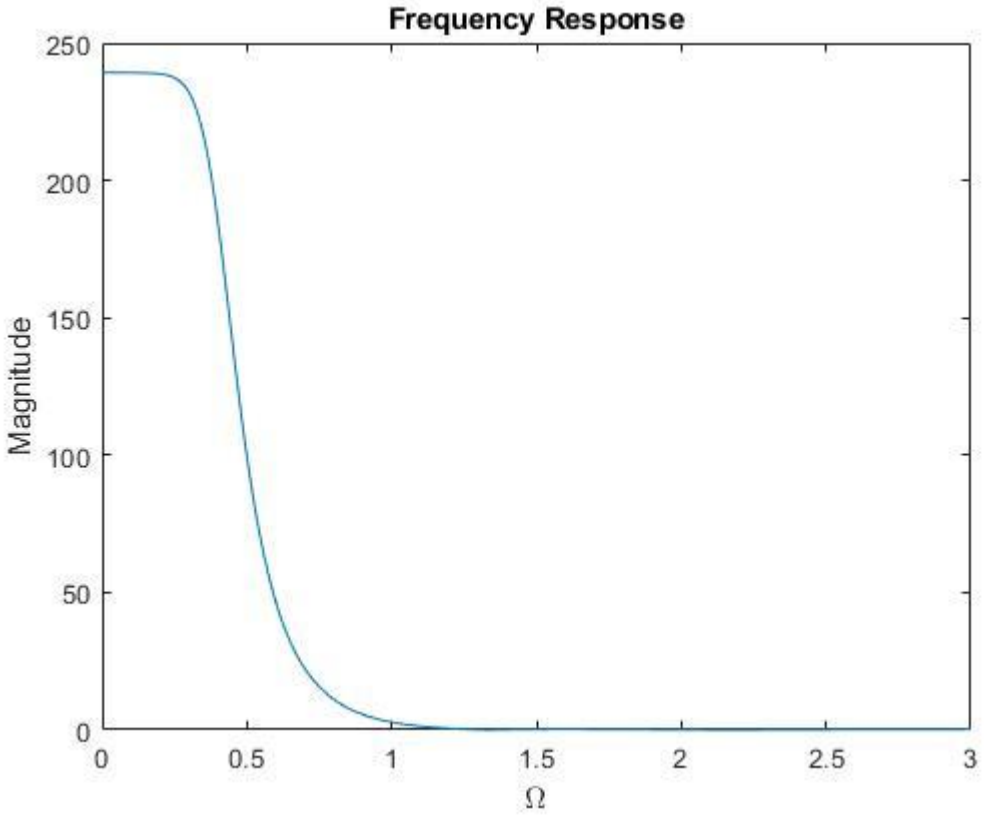
As we can see from the plot, it is a high pass filter but the response at higher frequencies are quite attenuating.

The code is same for rest of the problems in this question. We just need to change the poles and zeros according to the given functions.

For problem 1(ii)

```
%Poles
p1 = 0.86*exp(deg2rad(22.66)*i);
p2 = 0.86*exp(-deg2rad(22.66)*i);
p3 = 0.67*exp(deg2rad(10.48)*i);
p4 = 0.67*exp(-deg2rad(10.48)*i);

%Zeros
z1 = exp(deg2rad(76.35)*i);
z2 = exp(-deg2rad(76.35)*i);
z3 = exp(deg2rad(124.45)*i);
z4 = exp(-deg2rad(124.45)*i);
```

Plot for problem 1(ii):

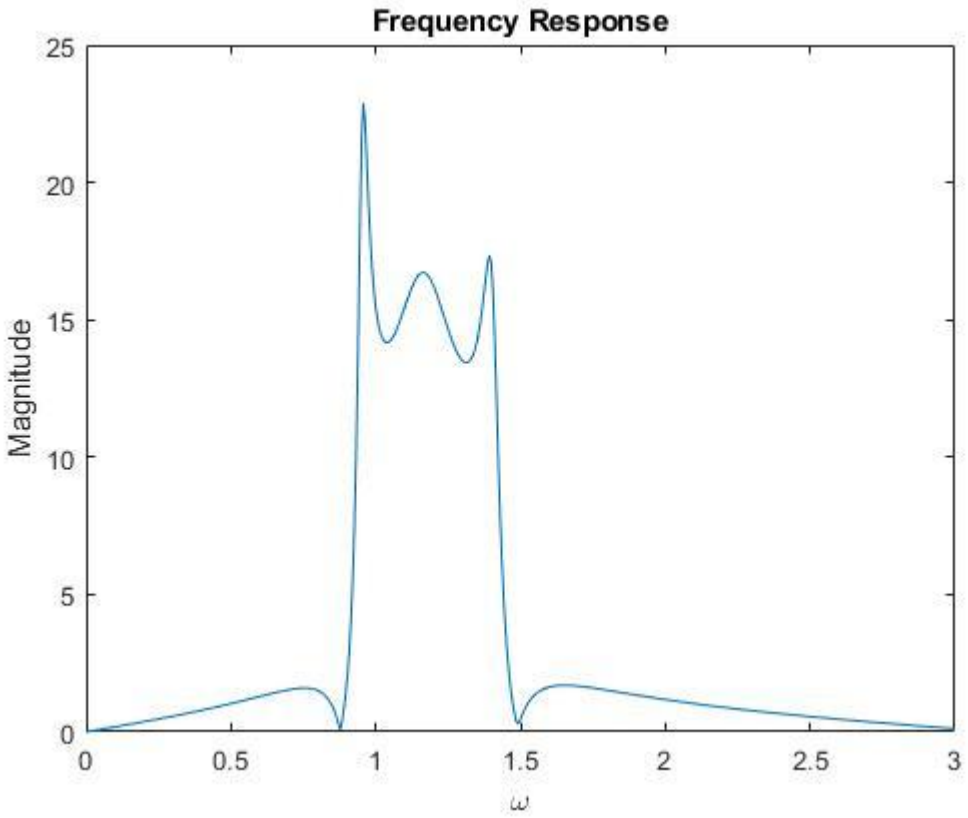


As we can see from the plot, it is a low pass filter with a sharp roll off.

For problem 1(iii)

```
%Poles
p1 = 0.16 + 0.96*i;
p2 = conj(p1);
p3 = 0.35 + 0.81*i;
p4 = conj(p3);
p5 = 0.57 + 0.8*i;
p6 = conj(p5);

%Zeros
z1 = 1;
z2 = -1;
z3 = 0.08 + 0.99*i;
z4 = conj(z3);
z5 = 0.64 + 0.77*i;
z6 = conj(z5);
```

Plot for problem 1(iii):

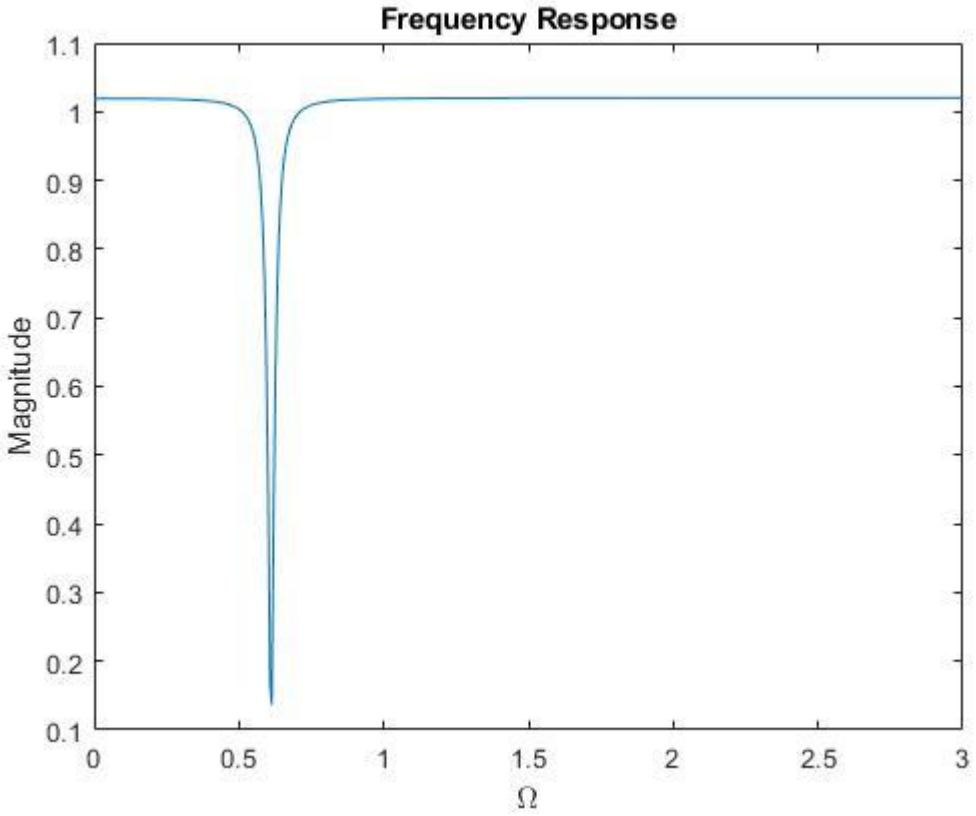


As we can see from the plot, it is a bandpass filter. But the gain in the passband is not uniform. It is maximum for the omega value of 1.

For problem 1(iv)

```
%Poles
p1 = 0.98*exp(deg2rad(35)*i);
p2 = 0.98*exp(-deg2rad(35)*i);

%Zeros
z1 = exp(deg2rad(35)*i);
z2 = exp(-deg2rad(35)*i);
```

Plot for problem 1(iv):



As we can see from the plot, it is a notch filter with very sharp roll off. It stops the gain for the angular frequency of around 0.6

Discussion:

Here we established a function with the poles and zeroes by using the convolution for multiplying different polynomials. Then we used the freqz function to get the frequency response and identifies the filters from the plots.

Problem 2

Question:

2. Take two examples of a signal $x(n) \xleftrightarrow{DFT,\ N\ point} X(k)$ from your choice and show that **Parseval's Theorem** holds true:

$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X(k)|^2$$

Signal-1: x = 4*sin(240*pi*t)

Code:

```
clc;
clear all;
close all;

fs = 2000;
ts = 1 / fs;
t = 0:ts:10;

%LHS
x = 4*sin(240*pi*t);
Et = sum(x.^2)

%RHS => FFT
X = fft(x, length(t)); %N-point FFT
Ef = sum(X.*conj(X)) / length(t)
```

Command window:

```
Et =

   1.6000e+05

Ef =

   1.6000e+05
```

Signal-2: x = 4*sin(240*pi*t) + 6*cos(700*pi*t)

The code is same except for the function input. We used sample frequency Fs = 5000

Command window:

```
Et =

   1.3000e+06

Ef =

   1.3000e+06
```

Discussion:

Here, we used the fft(x,n) function for n point Fast Fourier Transform. The signal values are padded with zeros if n is greater than signal length. On the other hand, signal values are truncated if n is smaller than signal length.

We first calculated the square sum of the n domain signal in LHS. Then, we averaged the square sum of the FFT signal. As we can see in the command window, both are equal. So Parseval's theorem was proved for both the signals of our choice.

Problem 3

Question:

3. Two continuous-time signals are given:

$$p(t) = 4\sin(100\pi t) + 3\cos(300\pi t) + \sin(800\pi t)$$

$$q(t) = 3\cos(300\pi t) + \sin(800\pi t)$$

Use a sampling frequency of $2000\ Hz$. For each mentioned signal, iteratively find the **lowest number of DFT points** that will satisfy the two conditions below:

(a) Only the main frequency representative peaks [three for $p(t)$ and two for $q(t)$] are visible within the $[0, \pi]$ frequency range.

(b) Each peak is located within 1.5% error of its actual representative frequency.

Code for p(t):

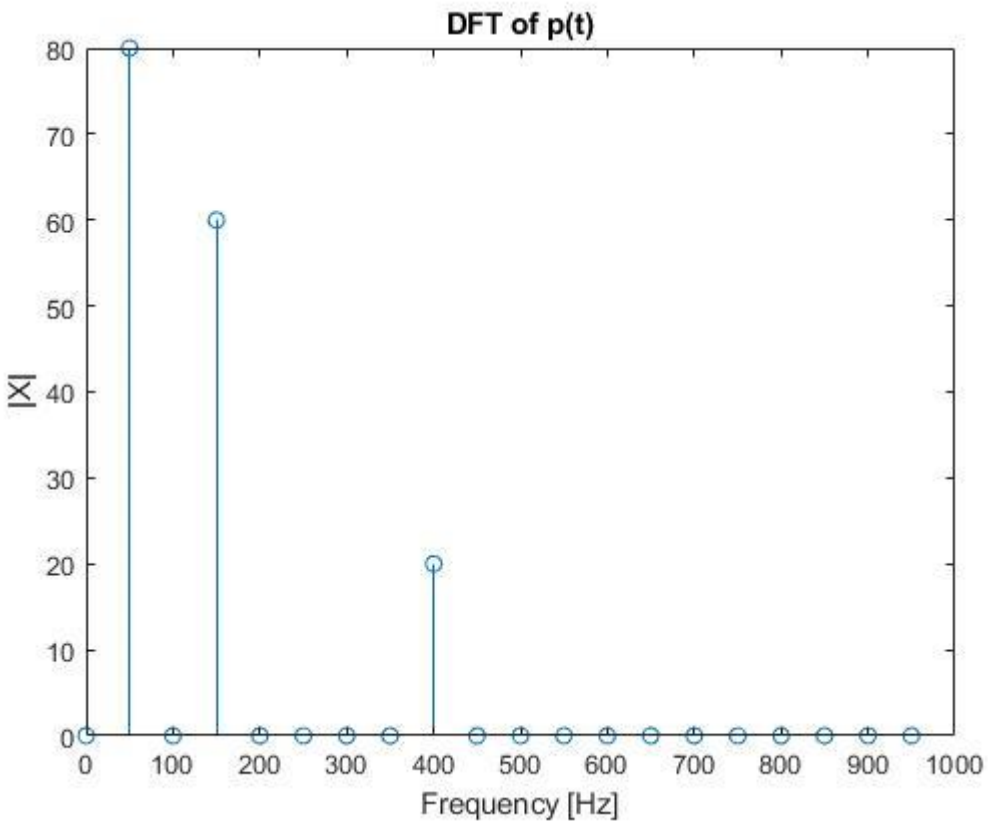
```
clc;
clear all;
close all;

fp1 = 50;
fp2 = 150;
fp3 = 400;
fp = [fp3 fp2 fp1];

fs = 2000;
ts = 1/fs;
n = (0:(120-1))*ts;

fig_no = 1;
for N_p=1:fs
    del_p = fs/N_p;

    if(mod(fp1,del_p)==0 && mod(fp2,del_p)==0 && mod(fp3,del_p)==0)
        p = 4*sin(2*pi*fp1*n) + 3*cos(2*pi*fp2*n) + sin(2*pi*fp3*n);
        P = abs(fftshift(fft(p,N_p)));
        w_p = (0:length(P)-1)/(length(P)*ts)-1/(2*ts) ;

        figure(fig_no)
        stem(w_p,P);
        xlabel('Frequency [Hz]');
        ylabel('|X|');
        title('DFT of p(t)');
        xlim([0 1000]);

        fig_no = fig_no + 1;
```

```
        [peaks_p,locs_p] = findpeaks(P,'Threshold',1);


        for i_p=1:length(peaks_p)/2
            error_p(i_p) = ((abs(w_p(locs_p(i_p)))-fp(i_p))/fp(i_p))*100;
        end

        if max(abs(error_p))<1.5
            break;
        end
    end
end

fprintf('The lowest number of DFT points required for p(t) = N_p = %d',N_p);
```

Command window:

The lowest number of DFT points required for p(t) = N_p = 40

Plot for p(t):



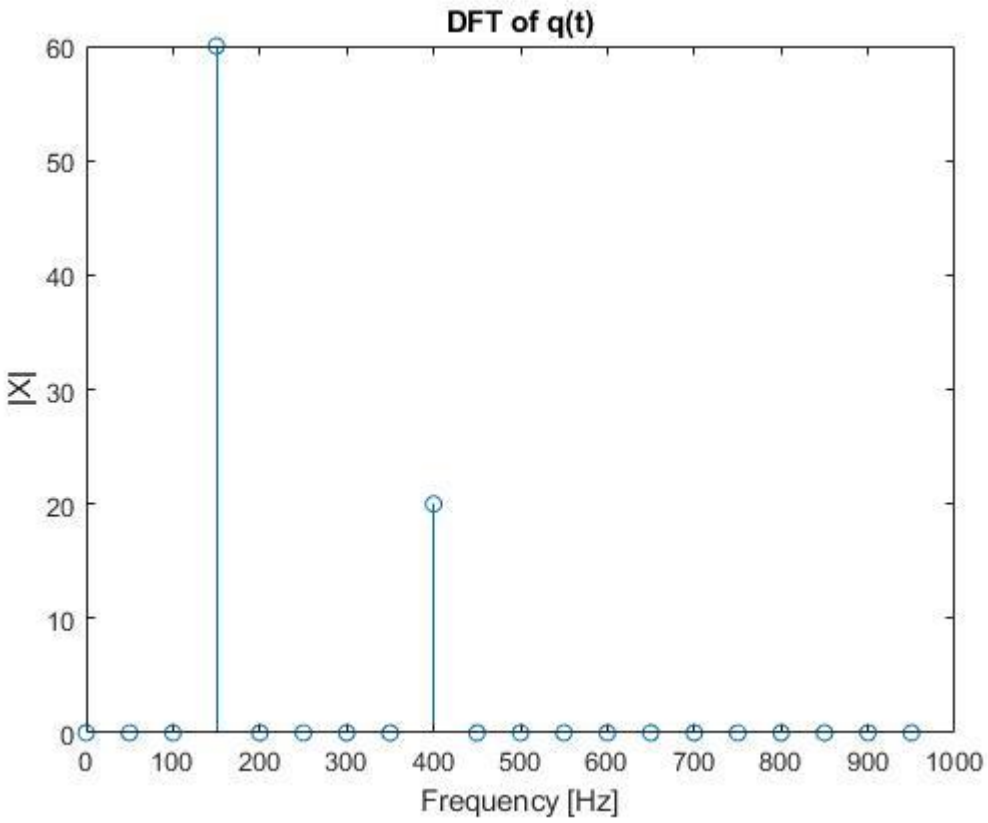
Code for q(t):

```
clc;
clear all;
close all;

fq1 = 150;
fq2 = 400;
fq = [fq2 fq1];

fs = 2000;
ts = 1/fs;
n = (0:(120-1))*ts;

fig_no = 1;
for N_q = 1:fs
     del_q = fs/N_q;

     if(mod(fq1,del_q)==0 && mod(fq2,del_q)==0 )
         q = 3*cos(2*pi*fq1*n) + sin(2*pi*fq2*n);
         Q = abs(fftshift(fft(q,N_q)));
         w_q = (0:length(Q)-1)/(length(Q)*ts)-1/(2*ts) ;

         figure(fig_no);
         stem(w_q,Q);
         xlabel('Frequency [Hz]');
         ylabel('|X|');
         title('DFT of q(t)');
         xlim([0 1000]);

         fig_no = fig_no + 1;

         [peaks_q,locs_q] = findpeaks(Q,'Threshold',1);
         for i_q=1:length(peaks_q)/2
               error_q(i_q)=((abs(w_q(locs_q(i_q)))-fq(i_q))/fq(i_q))*100;
         end

         if max(abs(error_q))<1.5
             break;
         end
     end
 end

 fprintf('The lowest number of DFT points required for q(t) = N_q = %d',N_q);
```

Command window:

The lowest number of DFT points required for q(t) = N_q = 40

Plot for q(t):



Discussion:

Here we did the Fourier transform of p(n) and q(n) and sorted for the peaks at the main frequency components. We ran the loop and found errors in each loop. We made sure the error came down below 1.5%

We checked for three frequency components in p and two frequency components in q.

In both cases, minimum DFT points are determined to be 40

Problem 4

Question:

4. Record your voice signal, $s(n)$ and corrupt is with coloured noise, $w(n)$:

$$y(n) = s(n) + w(n)$$

Compare the power spectral density (**PSD**) of the corrupted signal, $|Y(k)|^2$, with the original voice signal's PSD, $|S(k)|^2$. Consider the coloured noise signal to be each of these: (i) Car noise, (ii) Classroom/Office Noise, (iii) Ceiling fan noise, (iv) Traffic noise. [Source of the noise can be found in the internet or you may record them separately]. From the obtained PSDs, compare $r_{yy}(m)$ and $r_{ss}(m)$ as well.

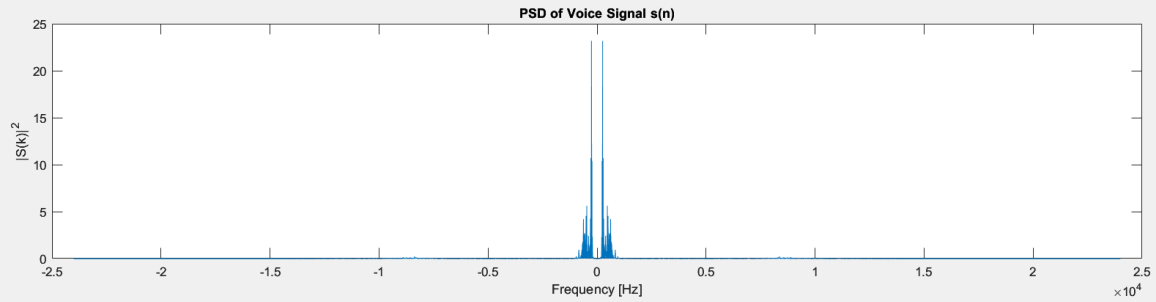
The code is same for all the problems in this question except for the noise input signal.

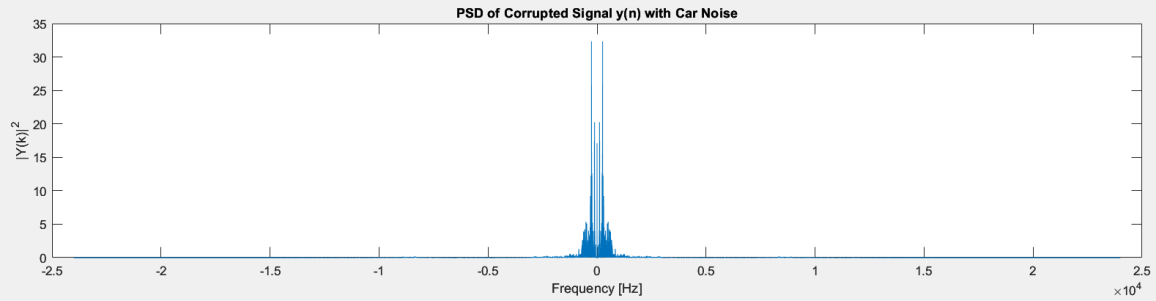
Code:

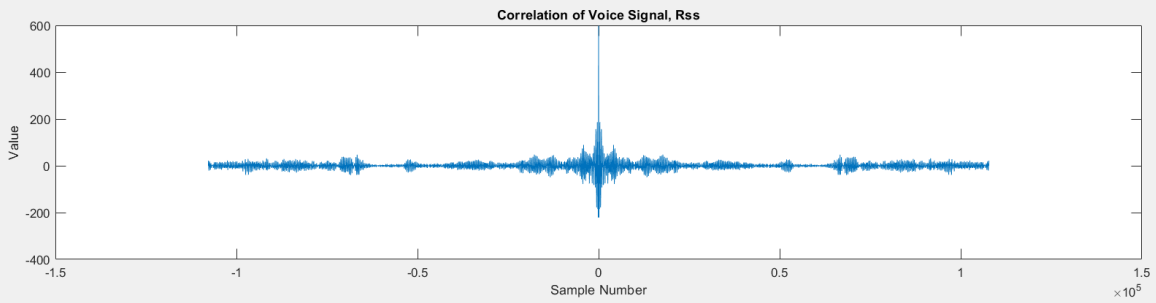
```
clc;
clear all;
close all;

%Voice Signal
[voice,fs] = audioread('1906044_DSP.mp3');
voice = voice(1:length(voice),1);
n1 = 1:length(voice);

%Car Noise
[noise,fsw] = audioread("Car.wav");
noise_car = resample(noise,fs,fsw);
noise_car = noise_car(1:length(voice),1); %Matching length with voice signal
n2 = 1:length(noise_car);

n = 1:max(length(n1),length(n2));
s = zeros(1,length(n));
w = s;
s(find(n>=n1(1) & n<=n1(end))) = voice;
w(find(n>=n2(1) & n<=n2(end))) = noise_car;

%PSD of Voice Signal
S = fft(s);
PSD_s = S.*conj(S);
N1 = length(n);
f = (0:1/N1:1-1/N1)*fs-(fs)/2;
```

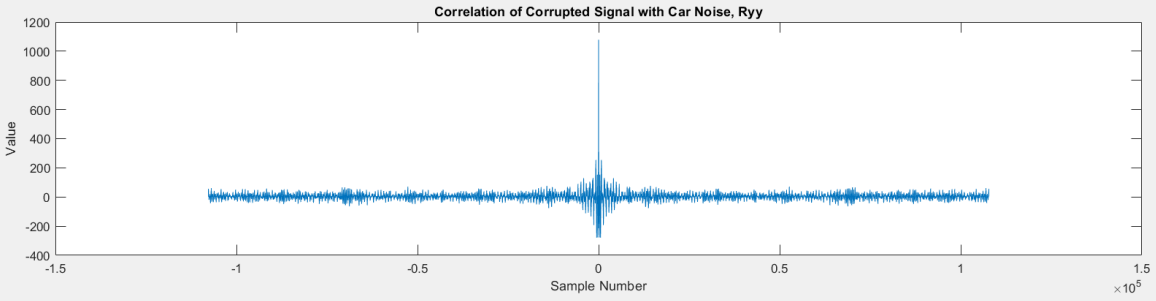
```
figure(1)
subplot(211)
plot(f,fftshift(abs(PSD_s/fs)));
xlabel("Frequency [Hz]");
ylabel("|S(k)|^2");
title("PSD of Voice Signal s(n)");

%PSD of corrupted signal
y = s + w;
sound(y,fs);
Y = fft(y);
PSD_y = Y.*conj(Y);

subplot(212)
plot(f,fftshift(abs(PSD_y/fs)));
xlabel("Frequency [Hz]");
ylabel("|Y(k)|^2");
title("PSD of Corrupted Signal y(n) with Car Noise");


%Correlation from PSD
s_corr = fftshift(ifft(PSD_s,length(PSD_s)*2));
y_corr = fftshift(ifft(PSD_y,length(PSD_y)*2));
n_corr = (0:length(s_corr)-1)-length(s_corr)/2;

figure(2)
subplot(211)
plot(n_corr,s_corr);
xlabel("Sample Number");
ylabel("Value");
title("Correlation of Voice Signal, Rss");

subplot(212)
plot(n_corr,y_corr);
xlabel("Sample Number");
ylabel("Value");
title("Correlation of Corrupted Signal with Car Noise, Ryy");
```

(i) Plot for car noise:

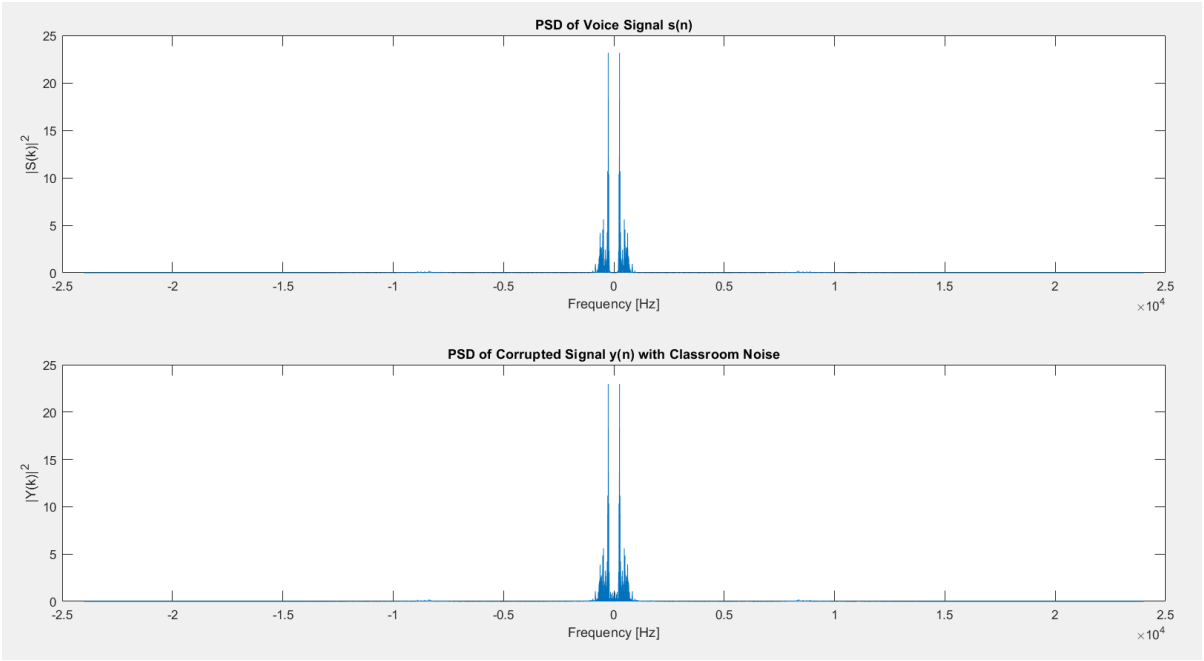


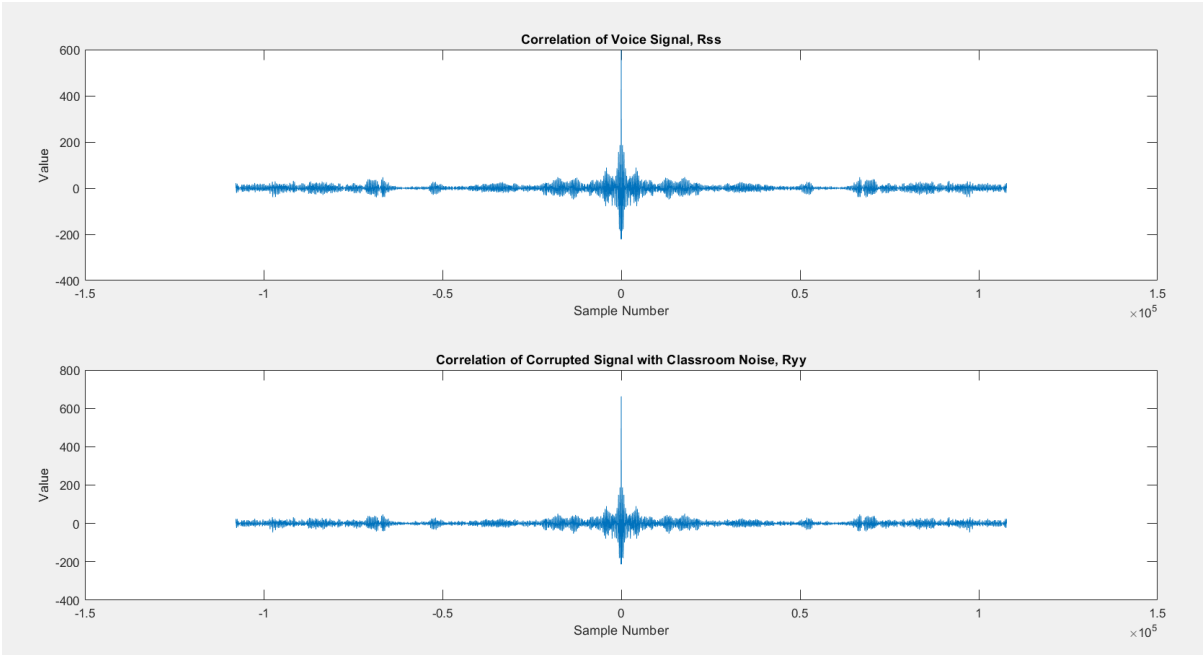




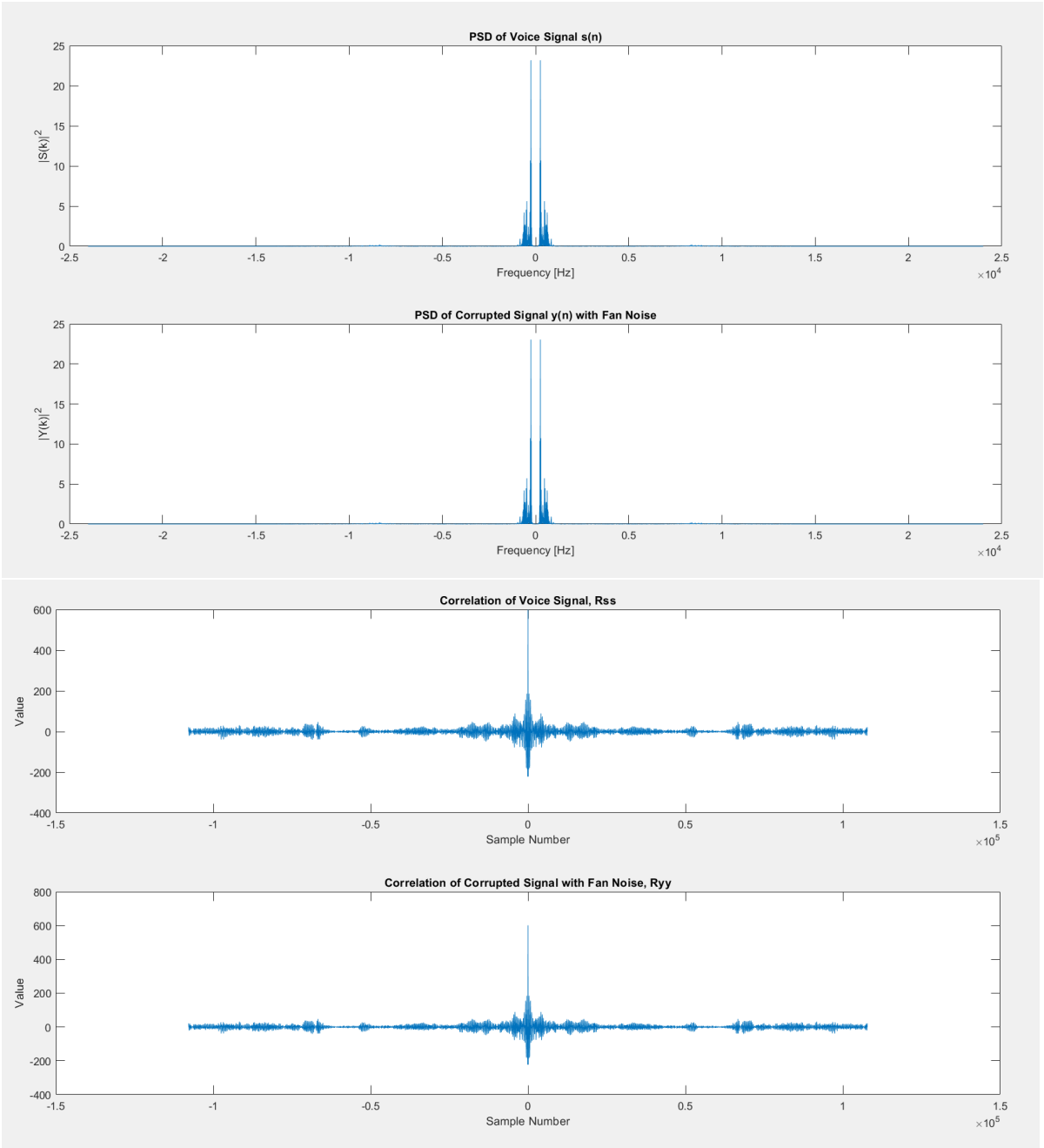


(ii) Plot for classroom/office noise:

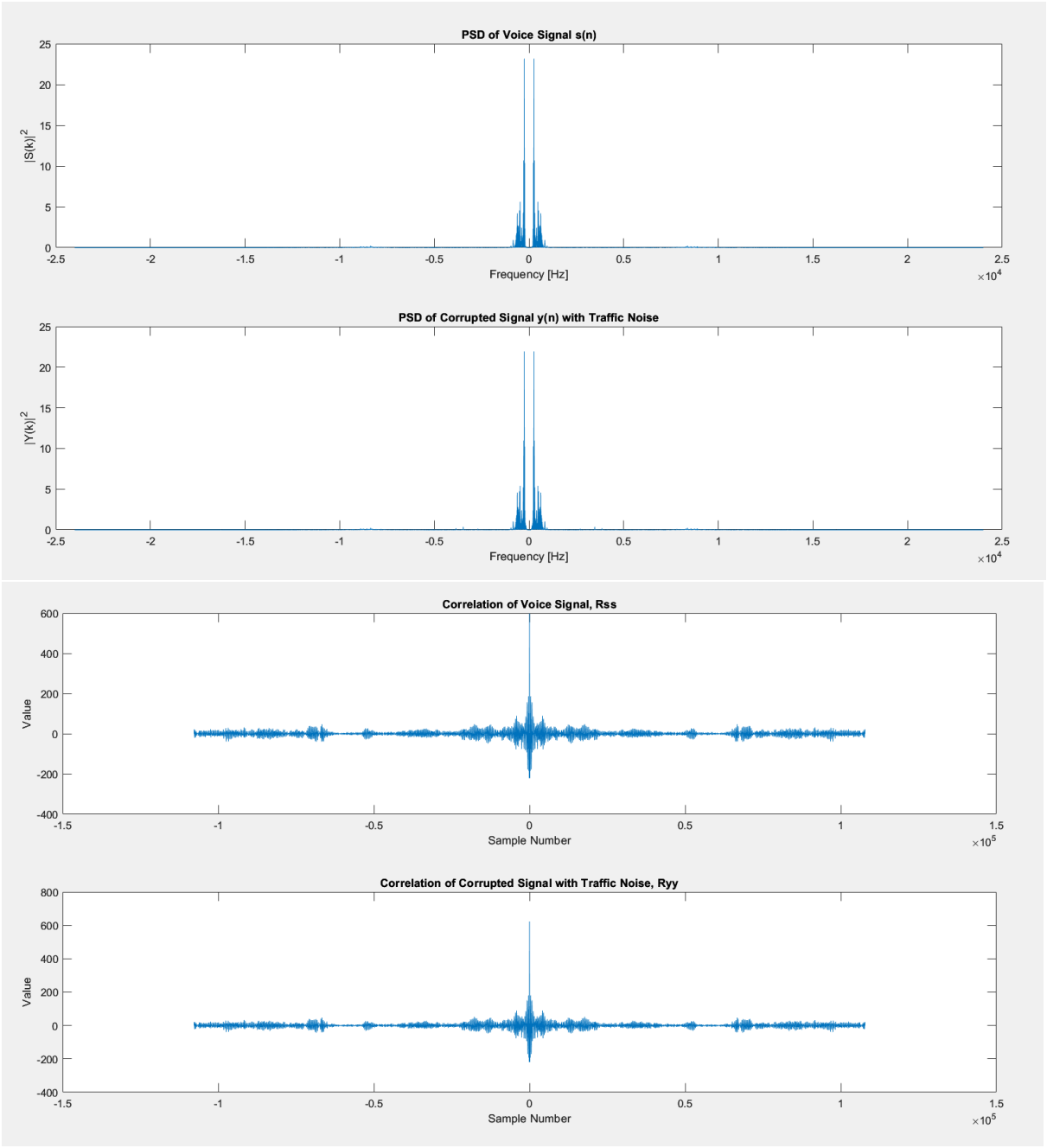




(iii) **Plot for ceiling fan noise:**



(iv) Plot for traffic noise:



Discussion:

To get the PSD signal we multiply the main signal fft and its conjugate. To get the DTFT of autocorrelation, we use PSD of a signal since both of them are equal. So we can find the correlation from the PSD.

From the plots we can notice that the frequency components rise as noise is added. The PSD of the corrupted signal hence is thicker than the original signal. The auto correlation of the noisy signal has zero lag. This happens since the added noise is random in values.

Whereas auto correlation of the original signal is more wide spread. The local peaks in this plot help to determine the pitch.

Problem 5

Question:

5. Implement an **FDM** system with the following protocol:

(i) $m_1(t)$ and $m_2(t)$ are voice recorded signals from you and your partner respectively (both saying the same sentence).

(ii) The sampling frequency found by "`audioread()`" will be too high for MATLAB to perform modulation. So you need to use a lower sampling frequency to accommodate for storage, which is effectively down sampling the audio. [The process for down sampling is depicted afterwards.]

(iii) $c_1(t)$ and $c_2(t)$ are carriers of $5kHz$ and $10kHz$.

(iv) Use a sampling frequency of $30kHz$ or $40kHz$. (This is the new F_s for down sampling. This also need to be greater than the carriers).

(v) As channel characteristics, add some **5~10 dB SNR** based gaussian noise after summing the two modulated signals.

(vi) Visualize each step of constructing the FDM signal. Use proper bandpass and lowpass **rectangular** filters to finally extract $m_1(t)$ and $m_2(t)$.

(vii) Listen to the extracted signals $m_1(t)$ and $m_2(t)$, as well as compare their sounds to the original versions. You will need `ifft()` to convert fourier transformed signals back to time domain signals.

(viii) For down sampling, study the code below and take code segments which are necessary for your implementation:

```
clc
close all
clear all

[m,fs] = audioread("Test0.mp3") ;   % sampling freqency from ".mp3"/".wav"
m = reshape(m,[1 length(m)]);
fs_lower = 40000;                   % lower sampling frequency: downsampling
to=length(m)/fs;
t_lower = [0:1/fs_lower:to-1/fs_lower];
t=[0:1/fs:to-1/fs];
m1 = interp1(t,m,t_lower);          % new audio clip

sound(m1,fs_lower)                  % sounds the same

fc = 5000;
c=cos(2*pi*fc*t_lower);
u=m1.*c;    % DSB -AM modulated Signal

N=2^15;
M=fft(m1,N);
M=M/fs_lower;%scaling

U=fft(u,N);
U=U/fs_lower;%scaling
fn = [0:1/N:1-1/N]*fs_lower-fs_lower/2;
subplot(211),plot(fn,abs(fftshift(M)))
subplot(212),plot(fn,abs(fftshift(U)))
```

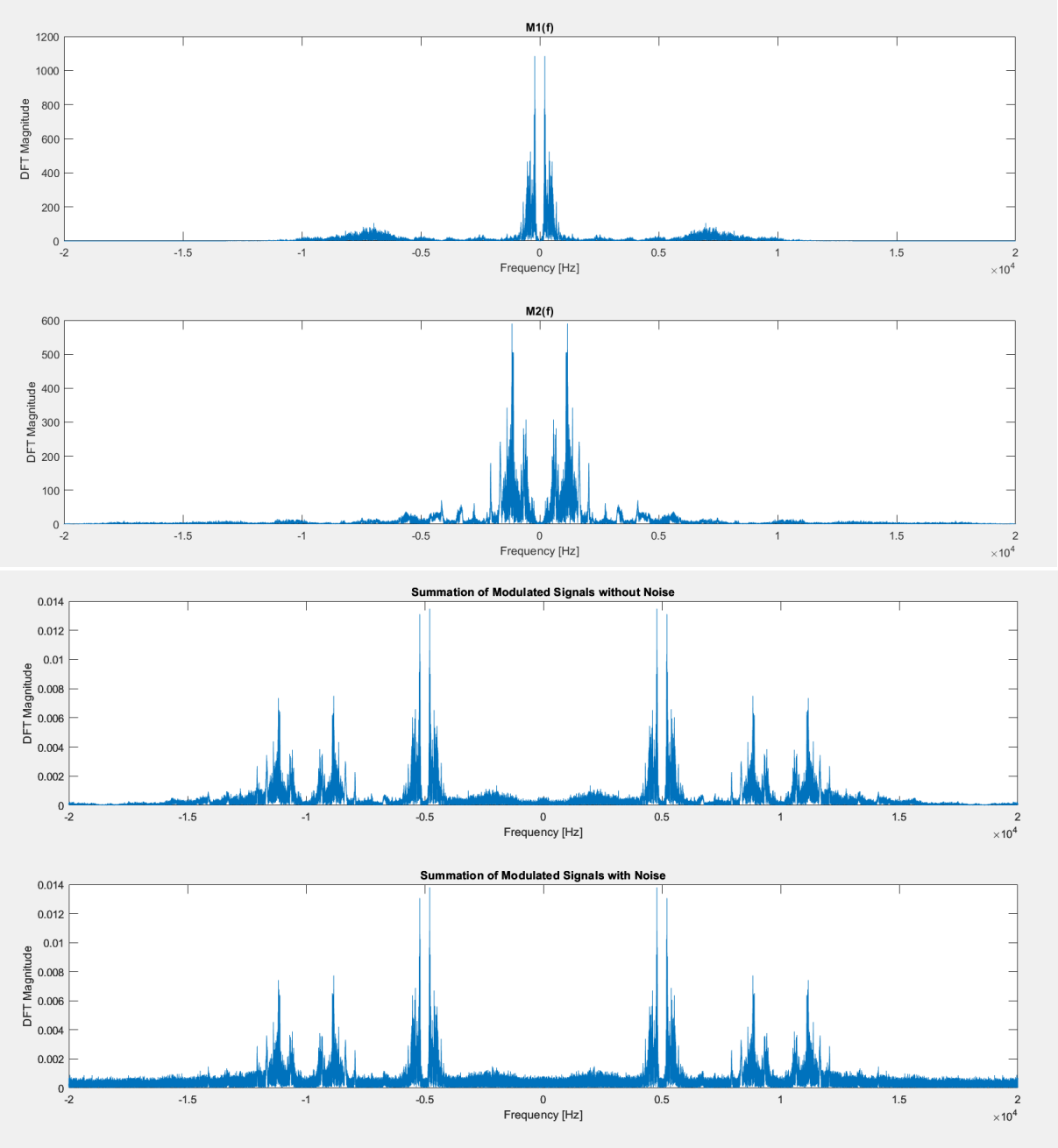
Code:

```
clc;
clear all;
close all;

[m_voice1,fs_1] = audioread("1906044.aac");
[m_voice2,fs_2] = audioread("1906055.ogg");

%Converting 2D arrays of voice signals to 1D arrays
m_voice1 = m_voice1(1:length(m_voice1),1);
m_voice2 = m_voice2(1:length(m_voice2),1);


%Downsampling the Audio Signals
m_voice1 = reshape(m_voice1,[1 length(m_voice1)]);
m_voice2 = reshape(m_voice2,[1 length(m_voice2)]);

n_m1 = 1:length(m_voice1);
n_m2 = 1:length(m_voice2);
n = 1:max(n_m1(end),n_m2(end));

m_1_1 = zeros(1,length(n));
m_2_2 = m_1_1;

m_1_1(find((n>=n_m1(1)) & (n<=n_m1(end)))) = m_voice1;
m_2_2(find((n>=n_m2(1)) & (n<=n_m2(end)))) = m_voice2;

fs_lower = 40000;
to = length(m_1_1)/fs_lower;
t_lower = [0:1/fs_lower:to-1/fs_lower];
t = 0:1/fs_lower:to-1/fs_lower;

m_1 = interp1(t,m_1_1,t_lower);
m_2 = interp1(t,m_2_2,t_lower);


%Modulation
fc_1 = 5000;
fc_2 = 10000;
c_1 = cos(2*pi*fc_1*t_lower);
c_2 = cos(2*pi*fc_2*t_lower);
N = length(t);
f = [0:1/N:1-1/N]*fs_lower-fs_lower/2;

u_1 = m_1.*c_1;
u_2 = m_2.*c_2;
u = u_1 + u_2;

figure(1)

subplot(211);
plot(f,abs(fftshift(fft(m_1,N))));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('M1(f)');
```

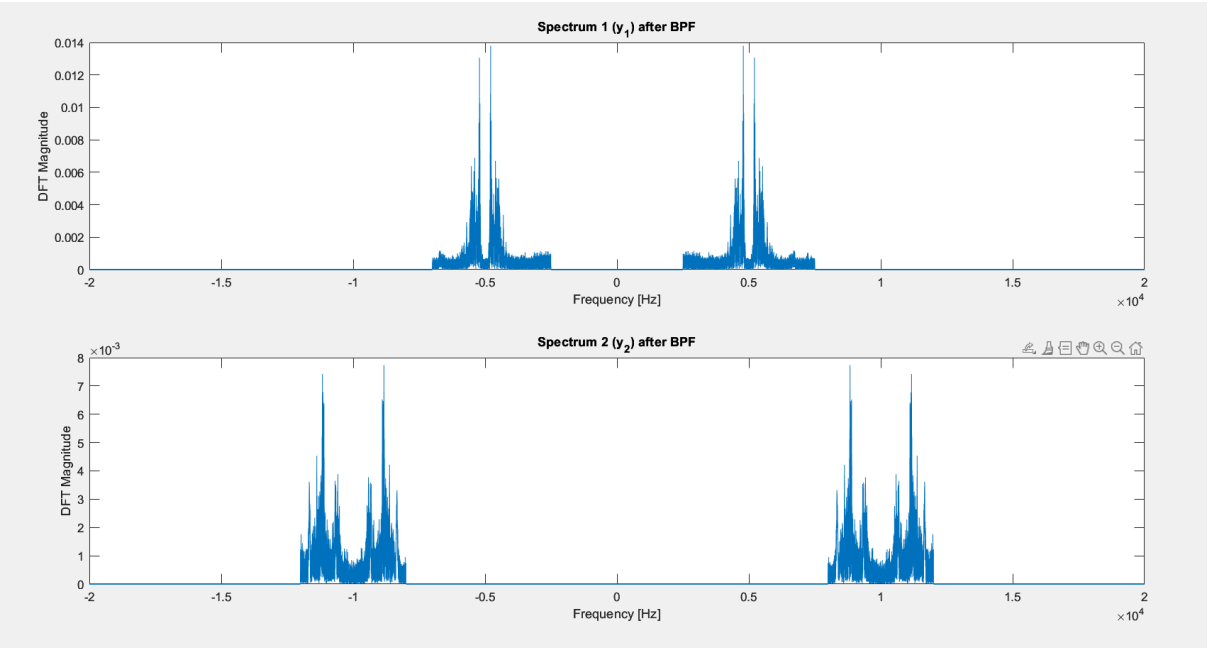
```
subplot(212);
plot(f,abs(fftshift(fft(m_2,N))));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('M2(f)');

y = abs(fftshift(fft(u)));

figure(2)

subplot(211);
plot(f,abs(y/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Summation of Modulated Signals without Noise');


SNR = 10; %in dB
P_x = sum(u.^2) / length(u);
P_y = P_x/10^(SNR/10);
n = sqrt(P_y) * randn(1,length(u));

u_noise = u + n;
y_noise = abs(fftshift(fft(u_noise)));

subplot(212)
plot(f,abs(y_noise/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Summation of Modulated Signals with Noise');


%Receiving end
BPF_1 = zeros(1,N);
BPF_2 = BPF_1;
BPF_1(f>2500 & f<7500) = 1;
BPF_1(f>-7000 & f<-2500) = 1;
BPF_2(f>8000 & f<12000) = 1;
BPF_2(f>-12000 & f<-8000) = 1;

y_1 = y_noise.*BPF_1;
y_2 = y_noise.*BPF_2;

figure(3)

subplot(211);
plot(f,abs(y_1/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Spectrum 1 (y_1) after BPF');

subplot(212)
plot(f,abs(y_2/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Spectrum 2 (y_2) after BPF');
```

```
%Demodulation
y_1_1 = ifft(y_1);
y_2_2 = ifft(y_2);

r_1 = y_1_1.*c_1;
r_2 = y_2_2.*c_2;

y1_f = fft(r_1);
y2_f = fft(r_2);

figure(4)

subplot(211);
plot(f,abs(y1_f/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Spectrum 1 (y1) before LPF');

subplot(212);
plot(f,abs(y2_f/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Spectrum 2 (y2) before LPF');


%LPF Design
LPF_1 = zeros(1,N);
LPF_2 = LPF_1;
LPF_1(abs(f)<1000) = 1;
LPF_2(abs(f)<2000) = 1;


%Message Recovery
m1_r = y1_f.*LPF_1;
m2_r = y2_f.*LPF_2;

figure(5)

subplot(211);
plot(f,abs(m1_r/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Reconstructed Signal 1');

subplot(212);
plot(f,abs(m2_r/fs_lower));
xlabel('Frequency [Hz]');
ylabel('DFT Magnitude');
title('Reconstructed signal 2');

sound(abs(ifft(ifftshift(m1_r))),fs_lower);
%sound(abs(ifft(ifftshift(m2_r))),fs_lower);
```

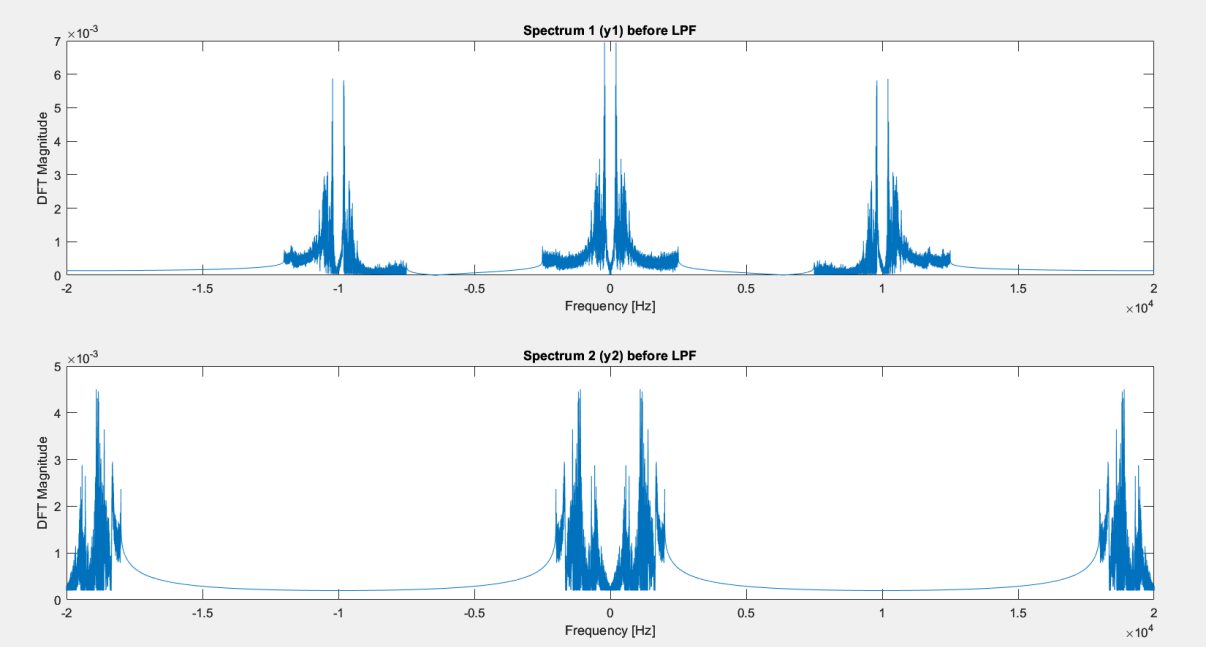
DFT amplitude plots:

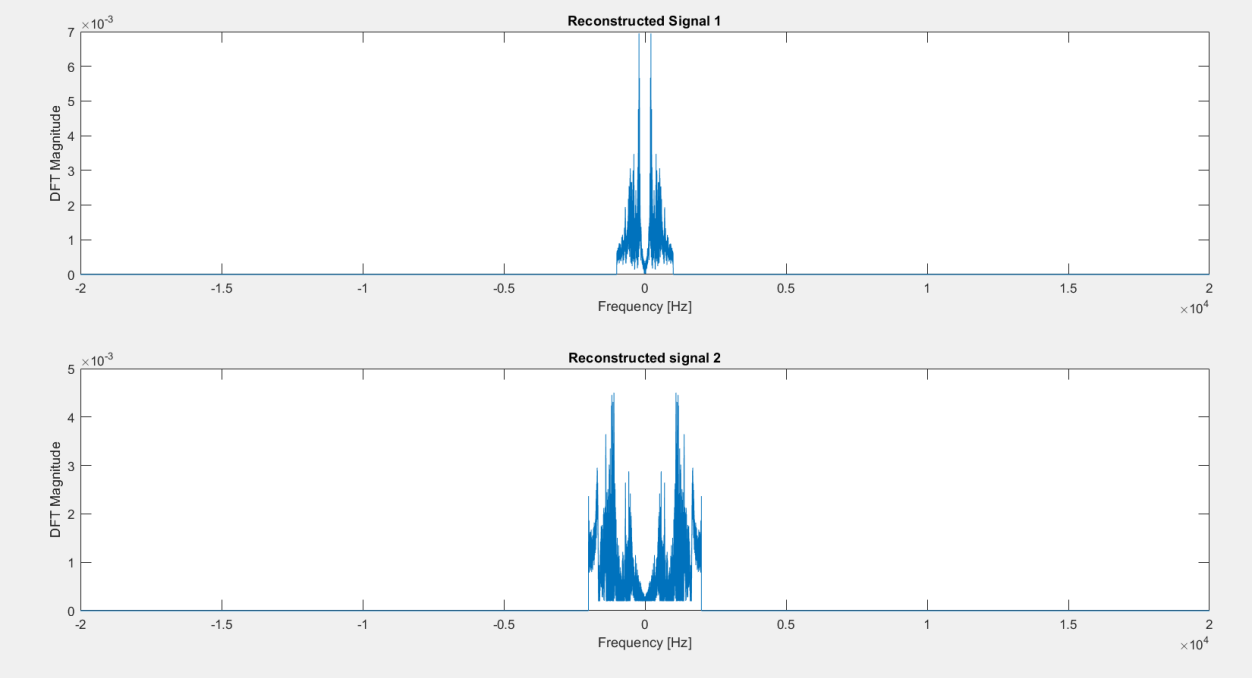


As we can see, both the signals had a spectra shift in the frequency domain. Message one was shifted to $\pm$5000HZ and message two was shifted to $\pm$10000Hz as they were modulated with respective carrier signals. Adding noise has increased randomly all the frequency components.



Here the Bandpass filter clearly separates the spectra contents of the signals allowing only one to pass at a time.





Finally, we multiply the signals again with respective carrier signals to demodulate. But this creates replicas in frequencies other than the message signal. So, we pass through a LPF to get the final output signal. In the frequency domain, the edge frequency components are truncated slightly.

Discussion:

As we finally hear the reconstructed signal, we can hear the message voice but a deeper version and with a lot of noise. It is almost hard to hear the signal uninterrupted. It is due to the presence of channel noise and cross talk. It is also from the fact that some of the frequency components are omitted during BPF filtering phase as well as LPF phase.

Conclusion:

We did Discrete Fourier analysis to determine different characteristics here and analyze the given systems. We also used PSD and correlations to have more insights into the signals. The experiment thus helped to understand frequency domain analysis in discrete scenario better.