UNIVERSITY OF MORATUWA, SRI LANKA

Faculty of Engineering

Department of Electronic and Telecommunication Engineering

Semester 5 (Intake 2020)

EN3551 - Digital Signal Processing

Assignment 1: Detecting Harmonics in Noisy Data and Signal Interpolation using Discrete Fourier Transform

Pramuditha A. A. H. – 200476P

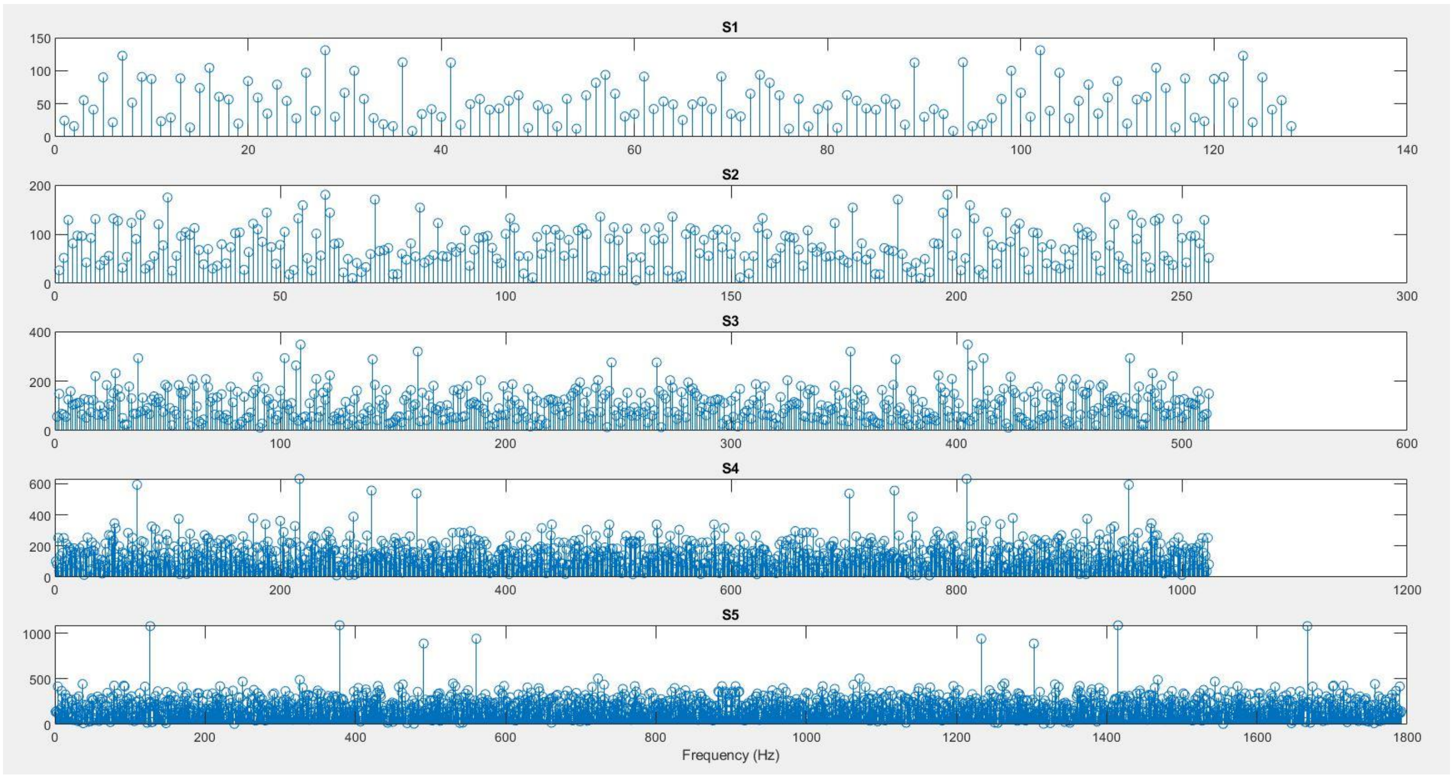
This report is submitted as a partial fulfillment for module EN3551 - Digital Signal Processing, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

1. Harmonic Detection

A noise corrupted signal of 1793 samples correspond to the index number: 200476P, was downloaded for this task. Mainly, the signal consists of 4 harmonics which is less than 64 Hz and severe white Gaussian noise.

First, 5 subsets of signal were made by extracting the first 128, 256, 512, 1024 and 1792 samples from the original sequence {x[n]} in time domain and then DFT was applied to each subset of samples and plotted the magnitude of the resulting DFT sequences to identify harmonics.

The magnitude plots of the DFTs of each sample can be viewed as follows:



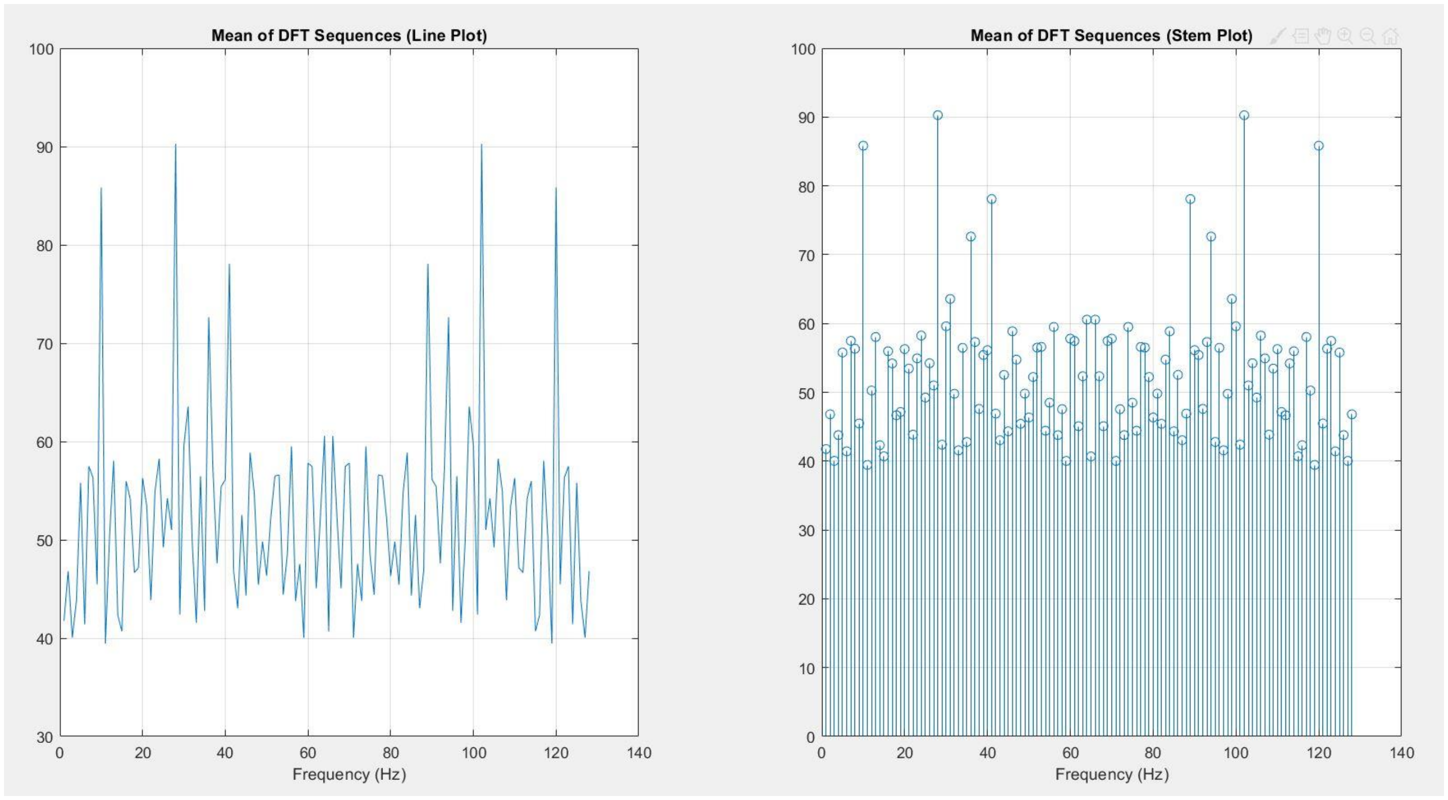
DFT coefficients of each subset were matched to the frequencies by using the following formula:

$$f_k = \frac{K * F_S}{N}$$

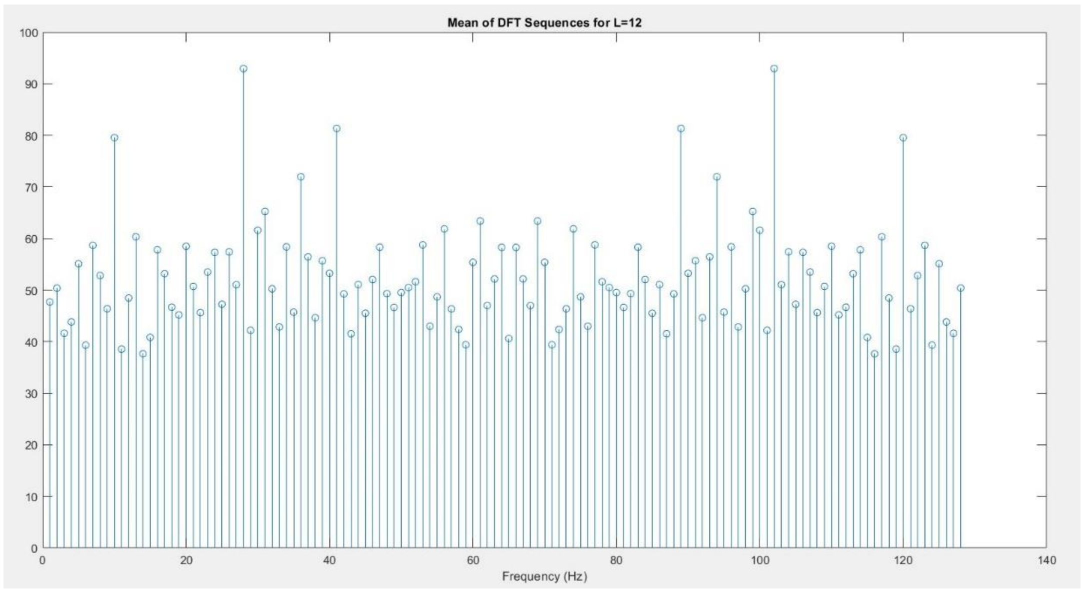
It can be clearly seen that, apart from the 4 harmonic components in the original signal, there are many frequency components in the stem plots. Also, when the length of the subset is increased with more time domain samples, the frequency components are also increased in each DFT subset. We can assume that these frequency components have been generated due to the severe white Gaussian noise mixed in the original signal. So, the desired harmonics cannot be extracted by this implementation.

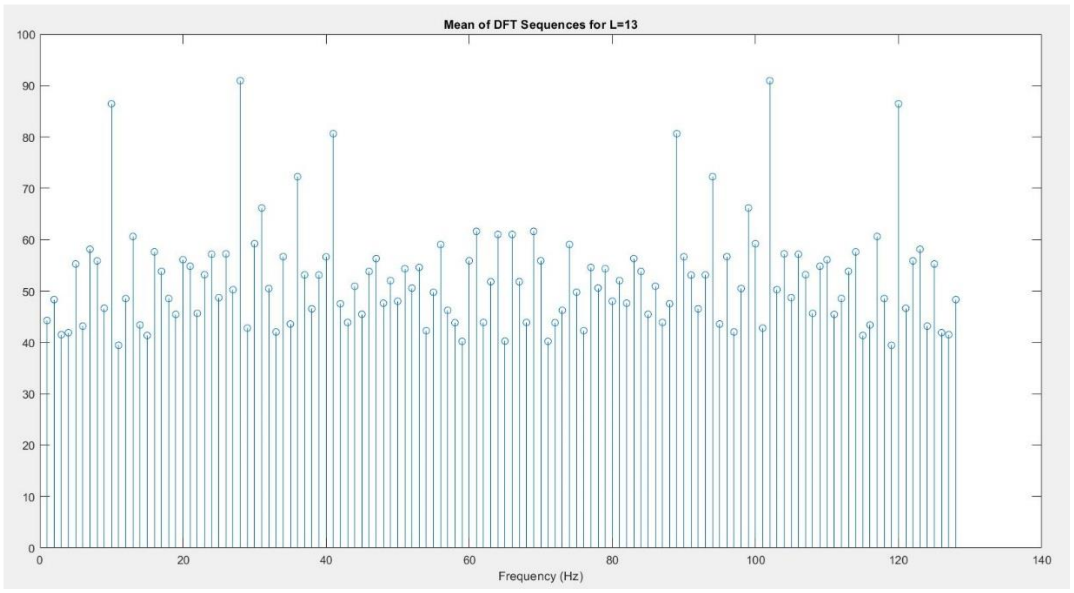
Based on the above equation, the four harmonics can be extracted at 8.9 Hz, 26.9 Hz, 34.98 Hz and 39.98 Hz frequency points. It is obvious that when we increase the number of samples in a subset, the harmonic events can be extracted easily from the noise flow (in S5 signal subset).

After applying DFT to variable length subsets composed with time domain components of the original signal, the whole signal set was divided into 14 different subsets including 128 samples from the original signal set in each. Then, DFT was applied to each subset and calculated the DFT coefficients. The average DFT was obtained by adding all the coefficients together and dividing the total by the number of subsets. The resulted DFT sequence was plotted which can be viewed as follows:

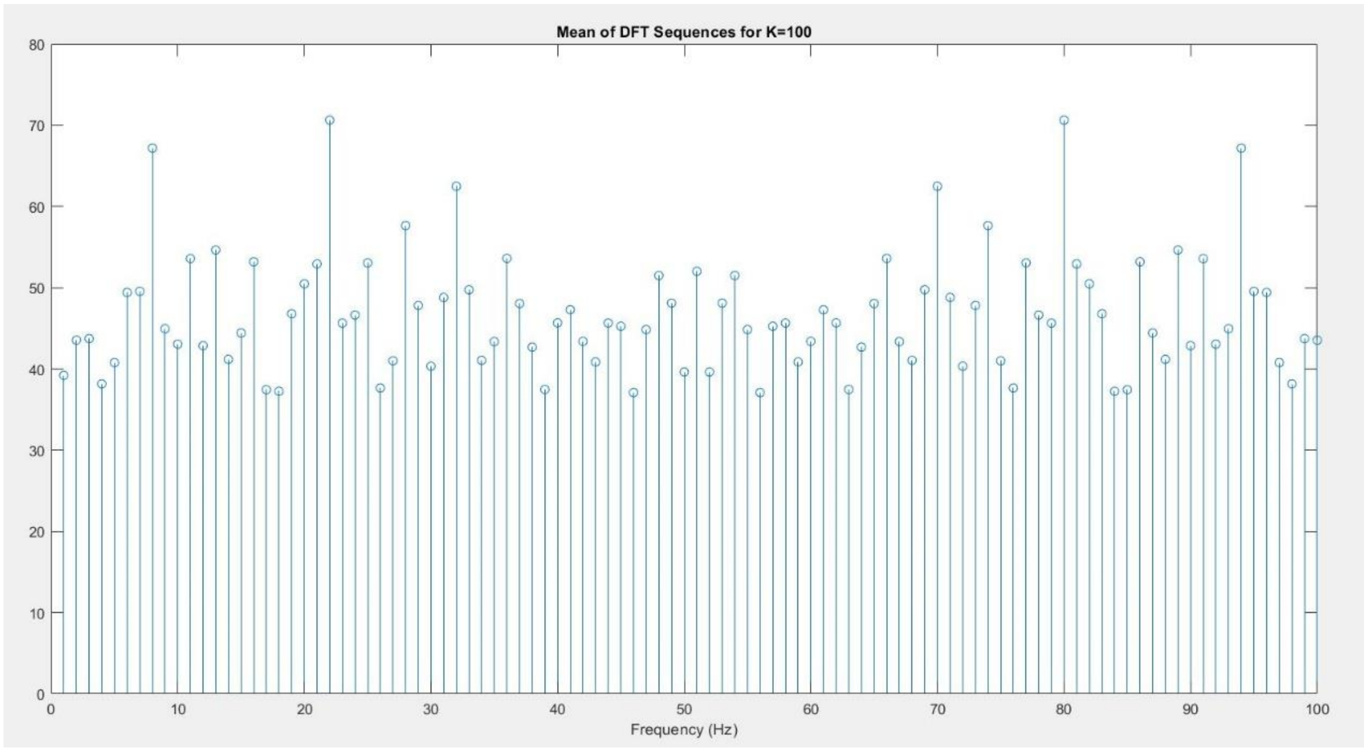


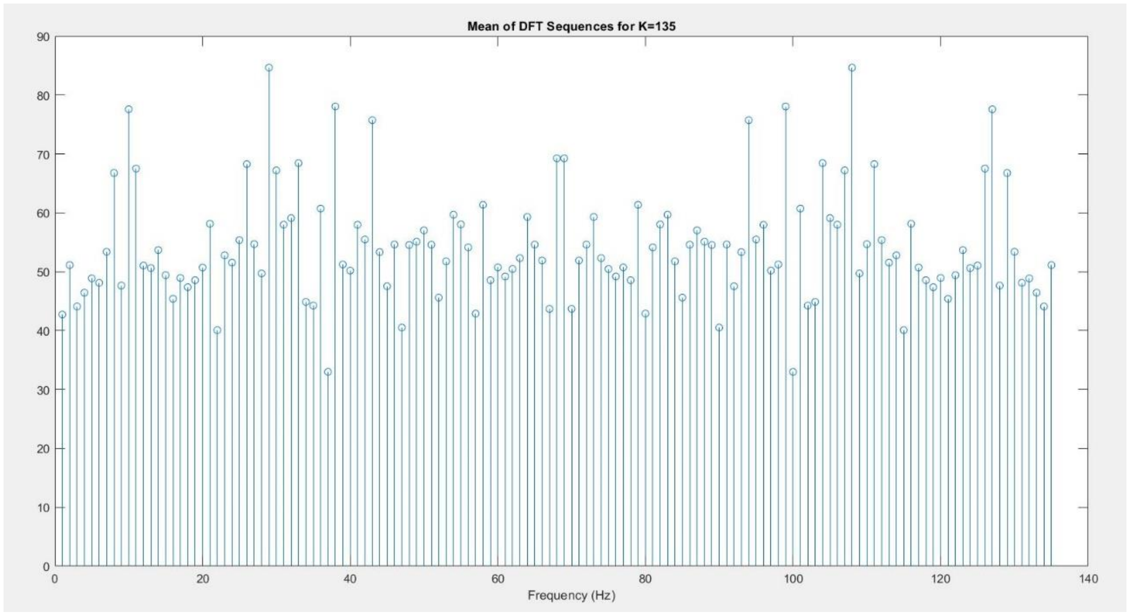
In task 05, this method was iterated for different number of L values with the same subset length to obtain the smallest L value, which will present the 4 harmonics more distinctively. It seems that when L=12 or L=13, these harmonics can be differentiated with other spikes easily.

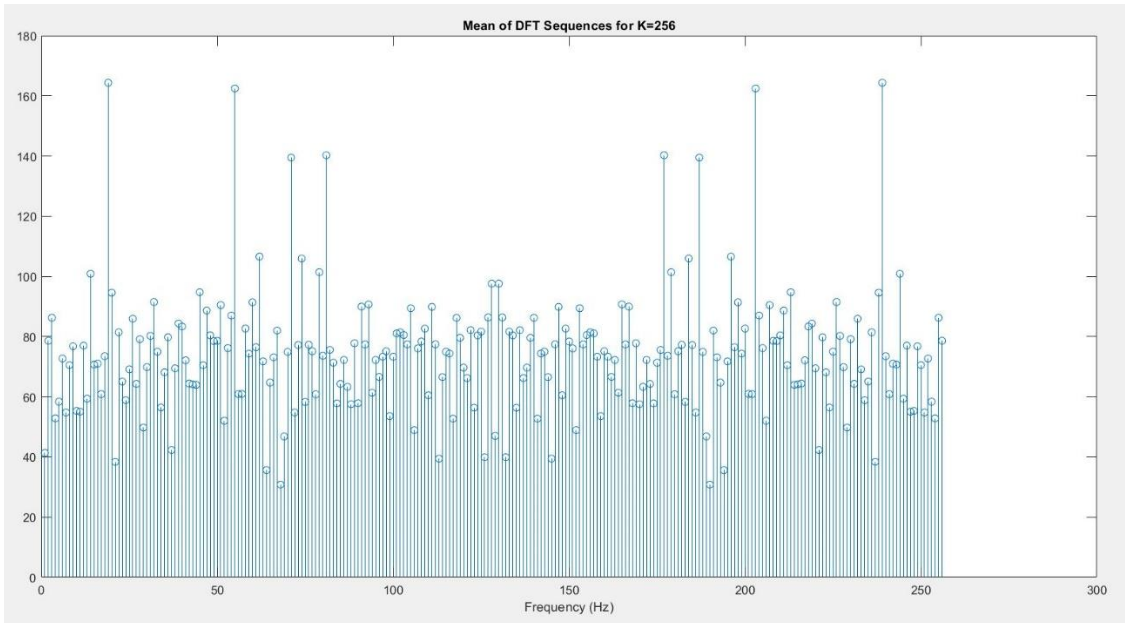


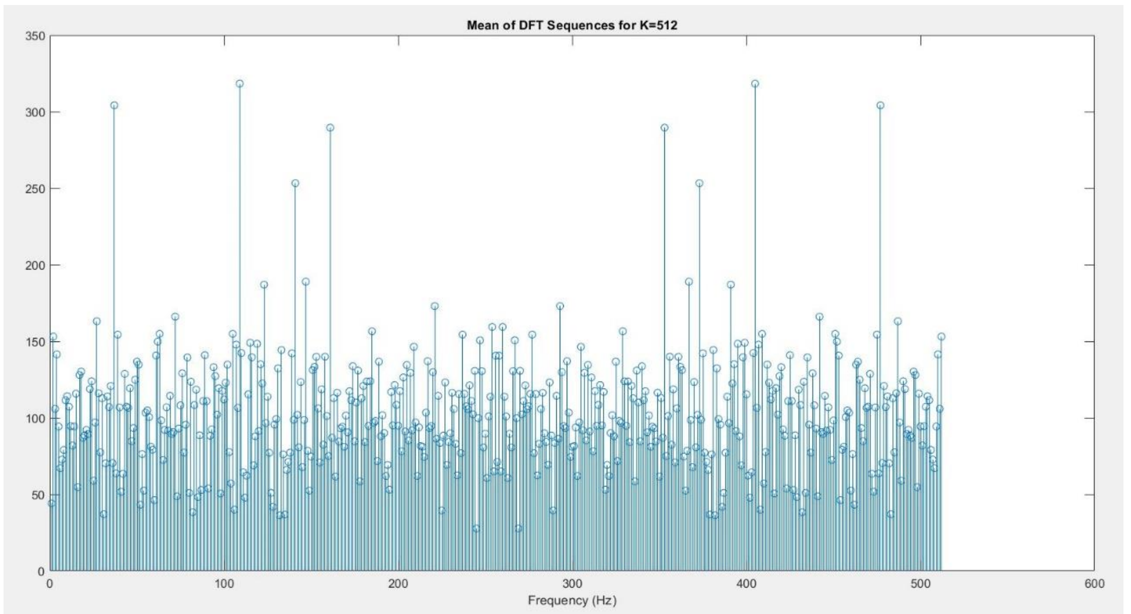


In task 06, If we try changing the value of K, such that K=100 or K=135 with the number of subsets nearly adjusted, the spikes cannot be identified like when K=128 because when we try to generate subsets with equal length (using floor function), some signal components may not be added. Also, decreasing the subset length will cause poor frequency resolution which would be a barrier to detect harmonics properly. Instead of those values we can use K=256, K=512, since under those values we can put all the signal samples into subsets and detect harmonics properly. But for higher K values, frequency coverage will be lower.



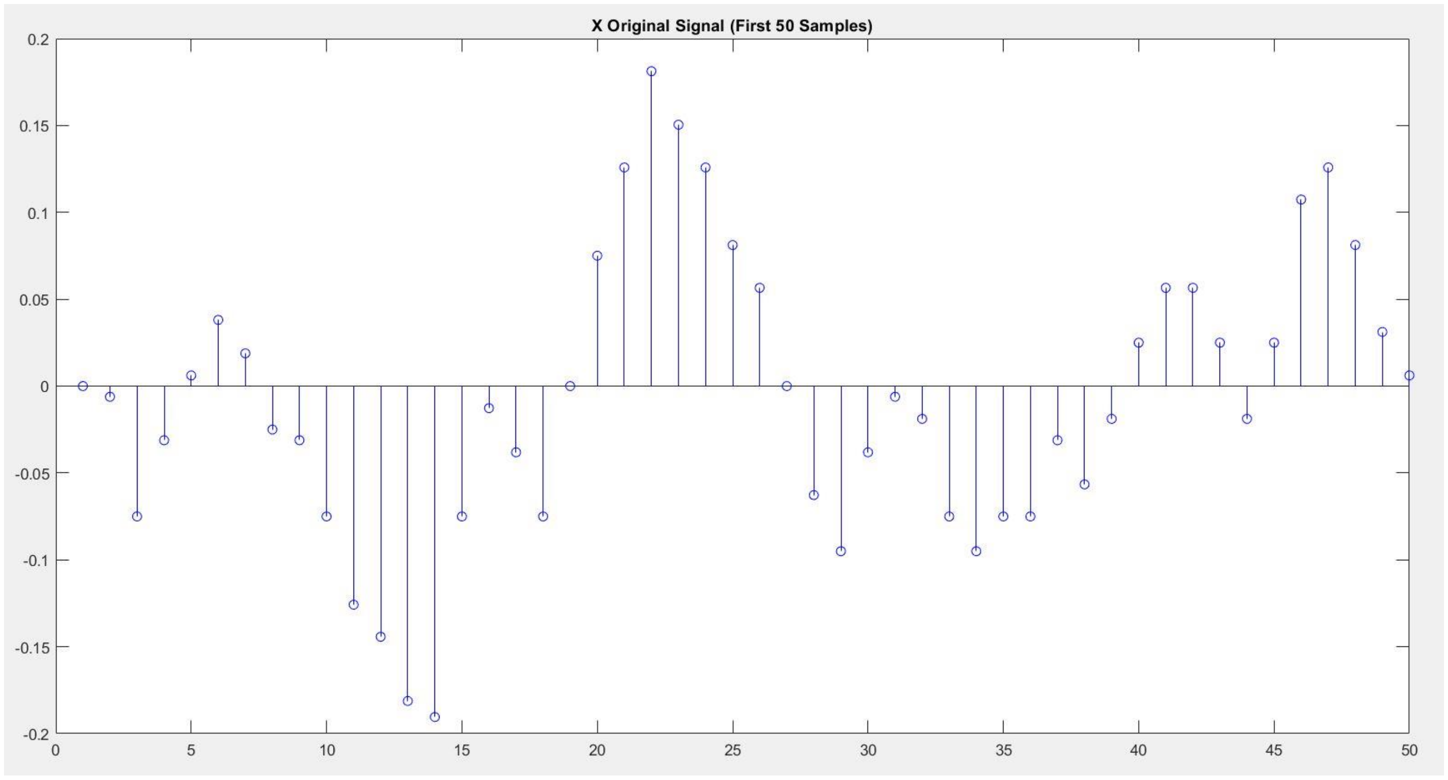

Mean of DFT Sequences for K=135
Frequency (Hz)
0
10
20
30
40
50
60
70
80
90
0
20
40
60
80
100
120
140


Mean of DFT Sequences for K=256
Frequency (Hz)
0
20
40
60
80
100
120
140
160
180
0
50
100
150
200
250
300


Mean of DFT Sequences for K=512
Frequency (Hz)
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

2. Interpolation

In this section, different signal interpolation methods were implemented to reconstruct the original signals by using Discrete Fourier Transform. To apply this technique, an audio signal with 20,000 audio samples in time domain was loaded to the MATLAB workflow as x[n].



Then several signal sets were generated based on the original signal y[n] as follows:

N = 20000

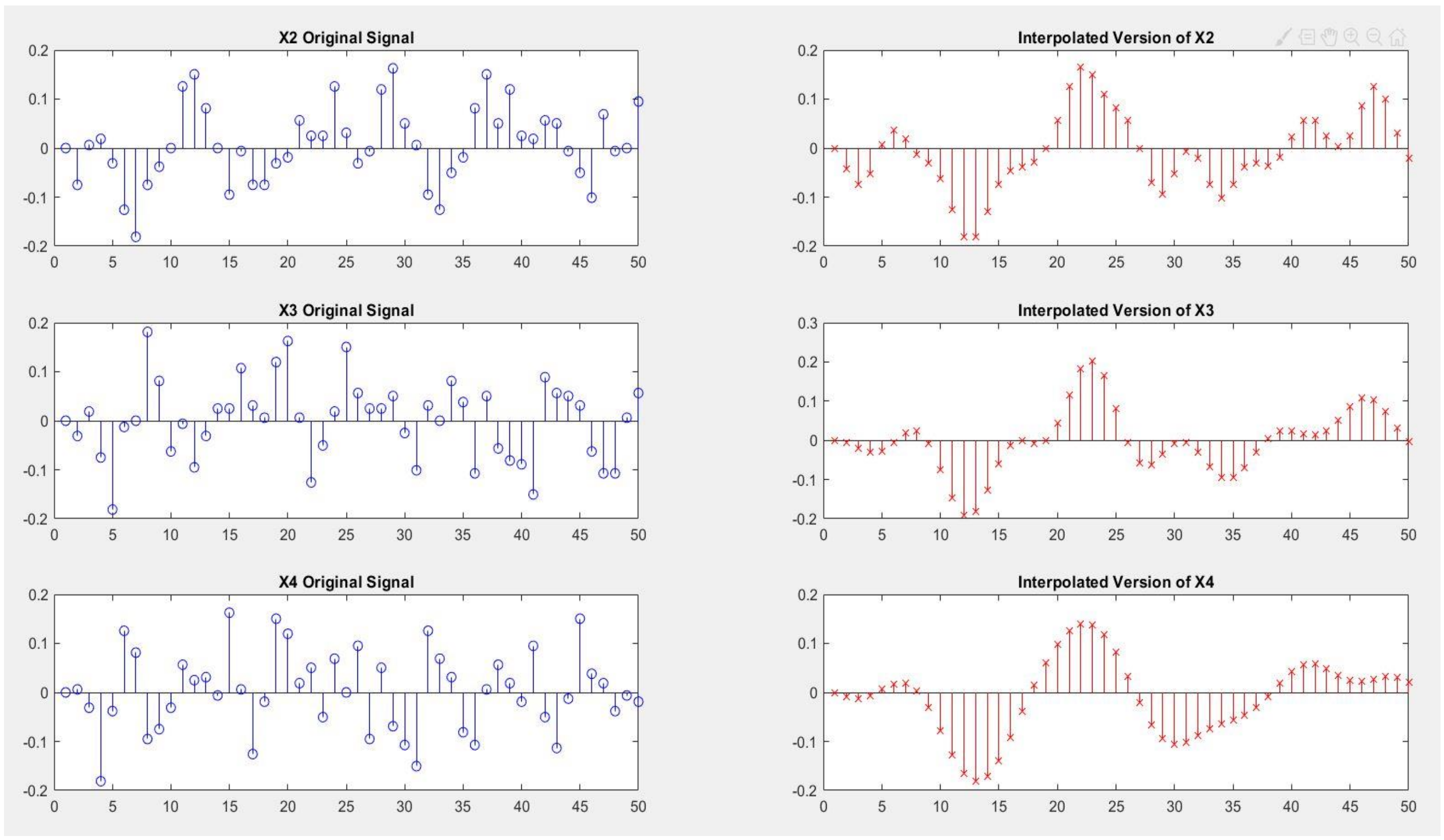
- X = y(1 : N)
- X2 = X(1 : 2 : N)
- X3 = X(1 : 3 : N)
- X4 = X(1 : 4 : N)

As the first step, X2 signal is interpolated with K=1. Here, there are two interpolation methods that can be used to reconstruct the original signal. They are,

1. Time Domain Interpolation by Zero Insertion
2. Frequency Domain Interpolation by Zero Insertion

Since the time domain interpolation method isn't practical and can be realized only with an ideal low pass filter, computationally it is a highly inefficient method. So, the frequency domain interpolation method has been chosen to do the task.

This method was repeated with the other signals and obtained the original and interpolated versions of the signals. They can be viewed as follows:



Here X2, X3 and X4 signals were interpolated in frequency domain with K=1, K=2 and K=3 respectively and obtained the interpolated graphs with respect to their original signals.

Based on the above graphs, it can be clearly seen that, when we increase the zero-insertion gap (K value) for the original signal in frequency domain and do the interpolation, the interpolated signal becomes much smoother and much more accurate. But, when we calculate the 2-norm of the difference between the original signal and the interpolated version of the signal, the 2-norm is increased with the increasing K value.

When we apply more and more zeros to the original signal, it will add more frequency components to the original signal spectrum, which causes more energy in the frequency spectrum. Also, this will change the frequency resolution of the DFT which results in coarser frequency resolution. This will make challenging to represent the original signal's frequency components precisely. So, the gap between the original and interpolated signal frequencies becomes much higher.

```
The difference between X2 and the original signal x in 2-norm: 6.144750
 The difference between X3 and the original signal x in 2-norm: 8.365213
 The difference between X4 and the original signal x in 2-norm: 23.499839
```

APPENDIX – MATLAB code

```matlab
%% Assignment 1 - Detecting Harmonics in Noisy Data and Signal
Interpolation using DFT
% Name          : Pramuditha A.A.H.
% Index No.     : 200476P


%% 3.1 Harmonic Detection

clc;
clear;
close all;

Fs = 128;               % Sampling rate
start_time = 0;         % Starting time of the sampling
end_time = 14;          % Ending time of the sampling

% Create vector of time range in sampling
t_range = (start_time:1/Fs:end_time);

% Load the signal file
signal_file = load('signal476.mat');
signal_data = signal_file.xn_test;

% Assiging signal values to subsets
S1 = signal_data(1:128);
S2 = signal_data(1:256);
S3 = signal_data(1:512);
S4 = signal_data(1:1024);
S5 = signal_data(1:1792);

signal_subsets = {S1, S2, S3, S4, S5};
dft_magnitudes = {};

% Loop through ech subset to compute the DFT magnitudes
for i = 1:5
    % Compute the DFT of the subset
    dft_transform = fft(signal_subsets{i});

    % Calculate the magnitude of the DFT coefficients
    magnitude = abs(dft_transform);

    dft_magnitudes{i} = magnitude;

    % Plot the magnitude for the current subset
    subplot(length(signal_subsets), 1, i);
    frequencies = (0:length(signal_subsets{i})-1) * Fs /
length(signal_data);
    stem(frequencies, magnitude);
    title(['S', num2str(i)]);

end
```

```
% Label the x-axis for the last subplot
xlabel('Frequency (Hz)');




%% 3.14 Apply DFT Averaging Method

% Initialize the variables/ parameters
total_samples = length(signal_data);
L = 14;
K = floor(total_samples / L);
i = 1; j = K;
mean_dft = 0;

for n = 1:L
    subset = signal_data(i:j);
    dft_magnitude = abs(fft(subset, K));
    mean_dft = mean_dft + dft_magnitude;

    i = j+1;
    j = j+K;
end

mean_dft = mean_dft / L;

% Plot the magnitude for the current subset
subplot(1, 2, 1);
plot(mean_dft);
title('Mean of DFT Sequences (Line Plot)');
xlabel('Frequency (Hz)');
grid on;
subplot(1, 2, 2);
stem(mean_dft);
title('Mean of DFT Sequences (Stem Plot)');
xlabel('Frequency (Hz)');
grid on;

% The least value of L which gives 4 distinguishable harmonics which are
% less than 64Hz can be considered as 12 or 13.

% Implement DFT averaging for different L values for the same subset
length
for L = 1:14
    x = 1;
    y = 128;
    dft_mean_new = 0;

    for n = 1:L
        subset = signal_data(x:y);
        dft_magnitude = abs(fft(subset, 128));
        dft_mean_new = dft_mean_new + dft_magnitude;

        x = y+1;
        y = y+128;
    end

    dft_mean_new = dft_mean_new / L;

    % Plot the average DFTs for different L values with the same subset
    % length.
    figure;
    subplot(1, 1, 1);
    stem(dft_mean_new);
    title(['Mean of DFT Sequences for L=',num2str(L)]);
    xlabel('Frequency (Hz)');
end


% Implementing DFT averaging method for different K values.
K = [100, 135, 512, 256];

for i = 1:length(K)
    x = 1;
    y = K(i);
    dft_mean = 0;
    subset_count = floor(length(signal_data)/K(i));

    for j = 1:subset_count
        subset = signal_data(x:y);
        dft_magnitude = abs(fft(subset, K(i)));
        dft_mean = dft_mean + dft_magnitude;

        x = y+1;
        y = y+K(i);
    end

    dft_mean = dft_mean / subset_count;

    % Plot the average DFTs for different L values with the same subset
    % length.
    figure;
    subplot(1, 1, 1);
    stem(dft_mean);
    title(['Mean of DFT Sequences for K=',num2str(K(i))]);
    xlabel('Frequency (Hz)');
end
```

```
%% 3.2 Interpolation

clc;
clear;
close all;

% Load the signal and store it in variable "y"
load handel;

% Get first 20,000 sample values
N = 20000;                      % Number of samples needed
signal_subset = y(1:N);

% Define various signals based on the obtained samples
x = y(1:N);
x2 = x(1:2:N);
x3 = x(1:3:N);
x4 = x(1:4:N);

% Plot original signal
figure;
subplot(1, 1, 1);
stem(x(1:50), 'b', 'Marker', 'o');
title('X Original Signal (First 50 Samples)');

% Signal interpolation for x2 by adding zeros into the middle of the
signal sample
% in ferquency domain.
K1 = 1;
X2 = fft(x2);
N1 = length(X2);
if mod(N1, 2) == 0
    N = N1/2;
    zero_padded_X2 = [X2(1:N); X2(N+1)/2; zeros((K1*N1)-1, 1); X2(N+1)/2;
X2((N+2):N1)];
else
    N = (N1+1)/2;
    zero_padded_X2 = [X2(1:N); zeros(K1*N1, 1); X2((N+1):N1)];
end

interpolated_signal_x2 = (K1+1)*ifft(zero_padded_X2);
edited_x2 = interpolated_signal_x2(1:((K1+1)*(N1-1))+2);
difference_2 = norm(edited_x2 - x);

% Plot the first 50 samples of both X2 and interpolated version of X2
% using stem plots
figure;
subplot(3, 2, 1);
stem(x2(1:50), 'b', 'Marker', 'o');
title('X2 Original Signal');
subplot(3, 2, 2)
stem(edited_x2(1:50), 'r', 'Marker', 'x');
```

```
title('Interpolated Version of X2');


% Signal interpolation for x3 by adding zeros into the middle of the
signal sample
% in ferquency domain.
K2 = 2;
X3 = fft(x3);
N2 = length(X3);
if mod(N2, 2) == 0
    N = N2/2;
    zero_padded_X3 = [X3(1:N); X3(N+1)/2; zeros((K2*N2)-1, 1); X3(N+1)/2;
X3((N+2):N2)];
else
    N = (N2+1)/2;
    zero_padded_X3 = [X3(1:N); zeros(K2*N2, 1); X3((N+1):N2)];
end

interpolated_signal_x3 = (K2+1)*ifft(zero_padded_X3);
edited_x3 = interpolated_signal_x3(1:((K2+1)*(N2-1))+2);
difference_3 = norm(edited_x3 - x);

% Plot the first 50 samples of both original and interpolated versions
% using stem plots
subplot(3, 2, 3);
stem(x3(1:50), 'b', 'Marker', 'o');
title('X3 Original Signal');
subplot(3, 2, 4)
stem(edited_x3(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X3');


% Signal interpolation for x4 by adding zeros into the middle of the
signal sample
% in ferquency domain.
K3 = 3;
X4 = fft(x4);
N3 = length(X4);
if mod(N3, 2) == 0
    N = N3/2;
    zero_padded_X4 = [X4(1:N); X4(N+1)/2; zeros((K3*N3)-1, 1); X4(N+1)/2;
X4((N+2):N3)];
else
    N = (N3+1)/2;
    zero_padded_X4 = [X4(1:N); zeros(K3*N3, 1); X4((N+1):N3)];
end

interpolated_signal_x4 = (K3+1)*ifft(zero_padded_X4);
edited_x4 = interpolated_signal_x4(1:((K3+1)*(N3-1))+4);
difference_4 = norm(edited_x4 - x);

% Plot the first 50 samples of both original and interpolated versions
% using stem plots
subplot(3, 2, 5);
stem(x4(1:50), 'b', 'Marker', 'o');
title('X4 Original Signal');
subplot(3, 2, 6)
stem(edited_x4(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X4');

% Print the 2-norm difference values between original signal and
% interpolated signal.
fprintf('The difference between X2 and the original signal x in 2-norm:
%f\n ', difference_2);
fprintf('The difference between X3 and the original signal x in 2-norm:
%f\n ', difference_3);
fprintf('The difference between X4 and the original signal x in 2-norm:
%f\n ', difference_4);
```

End of the Assignment