Department of Electronic & Telecommunication Engineering

University of Moratuwa

Sri Lanka

EN3551 - Digital Signal Processing

Assignment 1

Detecting Harmonics in Noisy Data and Signal Interpolation using DFT

210321X Kumarasinghe R D

15/09/2024

Contents

1 Harmonic Detection

This section focuses on the detection of harmonics in a noise-corrupted signal, which consists of 1793 samples. The signal contains four harmonics, corrupted by severe white Gaussian noise. We will identify the harmonics using the Discrete Fourier Transform (DFT) and investigate the effectiveness of the DFT averaging method to detect the harmonics more clearly.

1.1 Problem Description

- The noise-corrupted signal $\{x[n]\}$ contains 1793 samples collected at a sampling rate of $f_s =$ 128 Hz over the period of 14 seconds.
- The signal contains four harmonics, all of which have frequencies no greater than 64 Hz.
- The goal is to analyze the signal using subsets of various lengths and identify the harmonics using DFT.

1.2 Steps Taken

1.2.1 Subsets of the Signal

We first created several subsets from the sequence $\{x[n]\}$ by taking the first 128, 256, 512, 1024, and 1792 samples, which are denoted as S_1, S_2, S_3, S_4, and S_5, respectively.

1.2.2 Applying DFT for subsets

For each subset, the Discrete Fourier Transform (DFT) was applied to obtain the frequency spectrum. The magnitude of each DFT was then plotted to identify the harmonics. The respective frequency for k values is calculated using the below equation,

$$\textbf{frequency} = \frac{\textbf{K} \times \textbf{sampling frequency}}{\textbf{number of samples}}$$

Figure 1: Equation

1.2.3 Observation of Results

As the size of the subsets increases, the harmonics become more distinguishable from the noise. However, for the smaller subsets like S_1 and S_2, S_3,S_4, the harmonics are less visible due to the noise dominating the signal. Four harmonics are observed in signal 5 in **8Hz, 10.35Hz, 28Hz, 45Hz.**

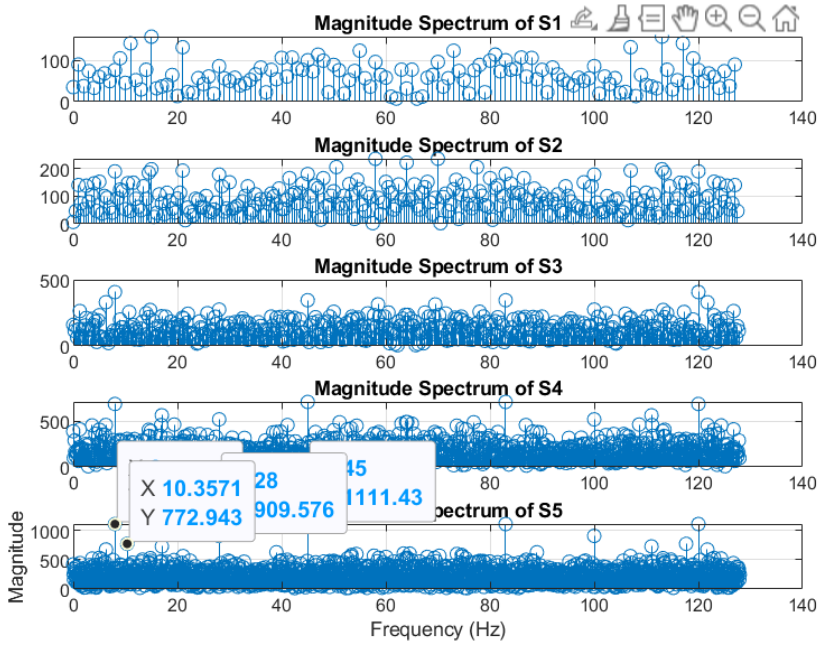


Figure 2: DFT coefficients of each subset

1.3 DFT Averaging Method

We now apply the DFT averaging method, where each subset is taken to be of length $K = 128$, and the number of subsets is $L = 14$.

This averaging reduces the effect of noise and enhances the visibility of the harmonics.

Harmonics were identified at 9Hz, 18Hz, 46Hz, 65Hz.

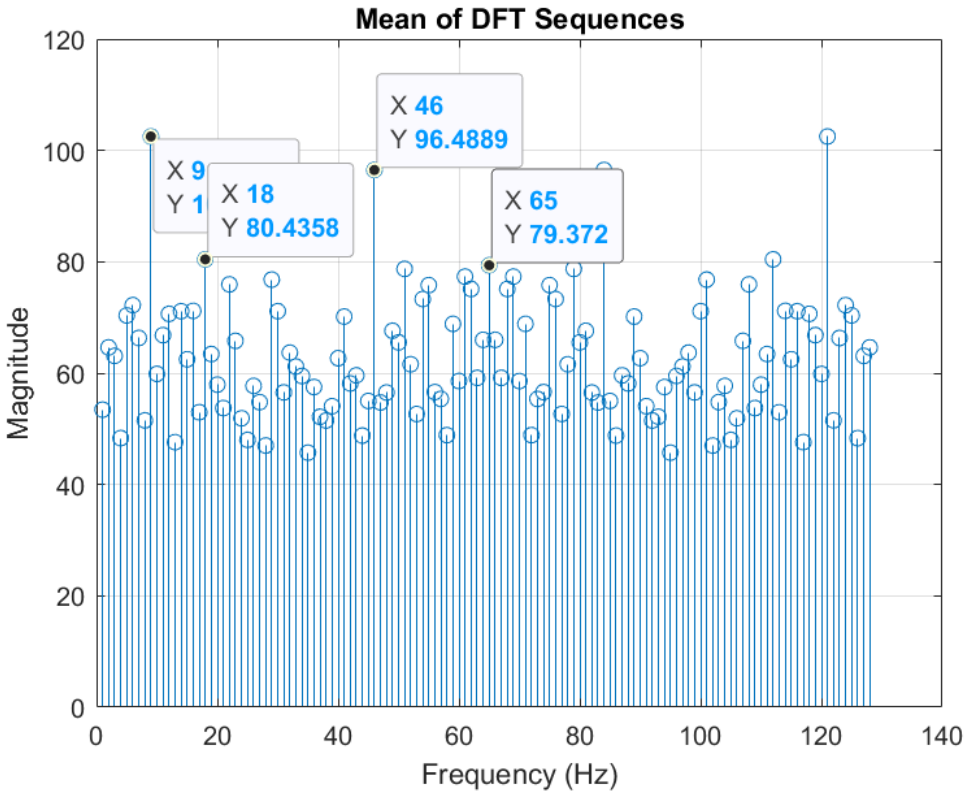


Figure 3: Harmonics from DFT Averaging Method

1.3.1 Smallest Value of L

Next we iterated thorough L = 1 to 14,keeping K = 128 to find out which L value gives out the more visibility of four hramonics.

To determine the smallest value of L such that the four harmonics (below 64 Hz) are clearly visible, we observe that $L = 7$ is sufficient. Averaging with fewer subsets, for example, $L = 5$ or $L = 4$, results in less effective noise reduction, making the harmonics less distinct.

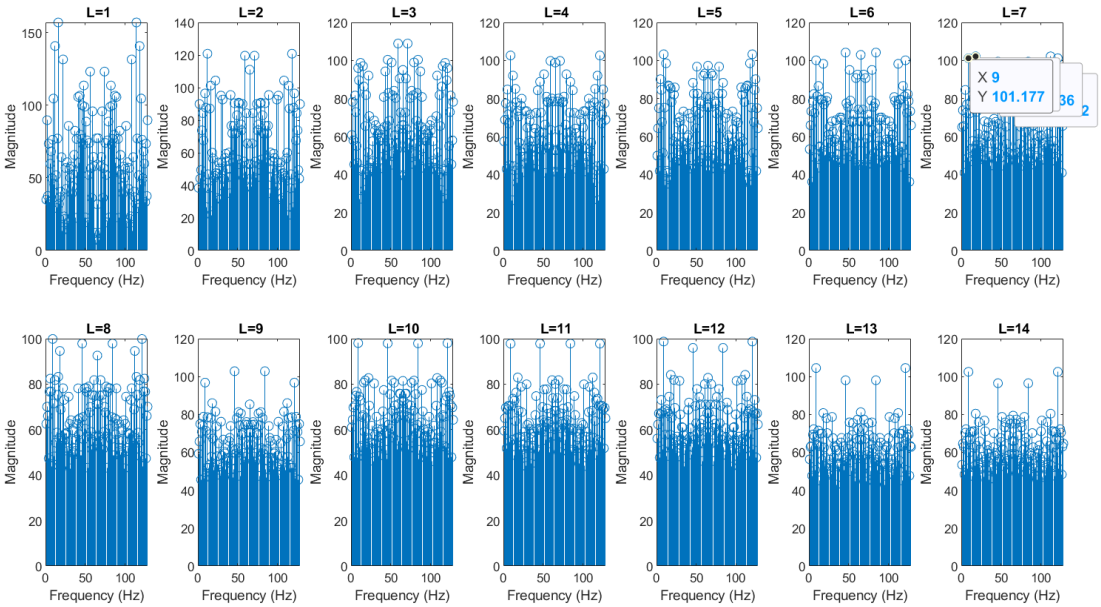


Figure 4: DFT magnitudes for different values of L

1.4 Effect of Other Values for K

The harmonics become harder to observe when the value of K is set to 100 or 135. The subset length, represented by K, remains the same across all subsets and determines the number of samples in each subgroup. When the sampling frequency is 128 Hz and K is set to 128, each subset captures a full cycle of the signal. This ensures that the samples in each subset are aligned and contain the same signal information.

When using other values for K, such as 100 or 135, problems arise. In these cases, subsets may fail to capture complete signal cycles, leading to sample misalignment. This misalignment causes interference between the harmonic contributions of different subsets. As a result, harmonic extraction becomes more challenging during averaging techniques like the DFT. For example, when $K = 100$, the first sample in one subset might not correspond to the same point in the signal cycle as the first sample in another subset, complicating the detection of harmonics.

Therefore, $K = 128$ is a reasonable choice for balancing the trade-off between frequency resolution and the length of the subsets.

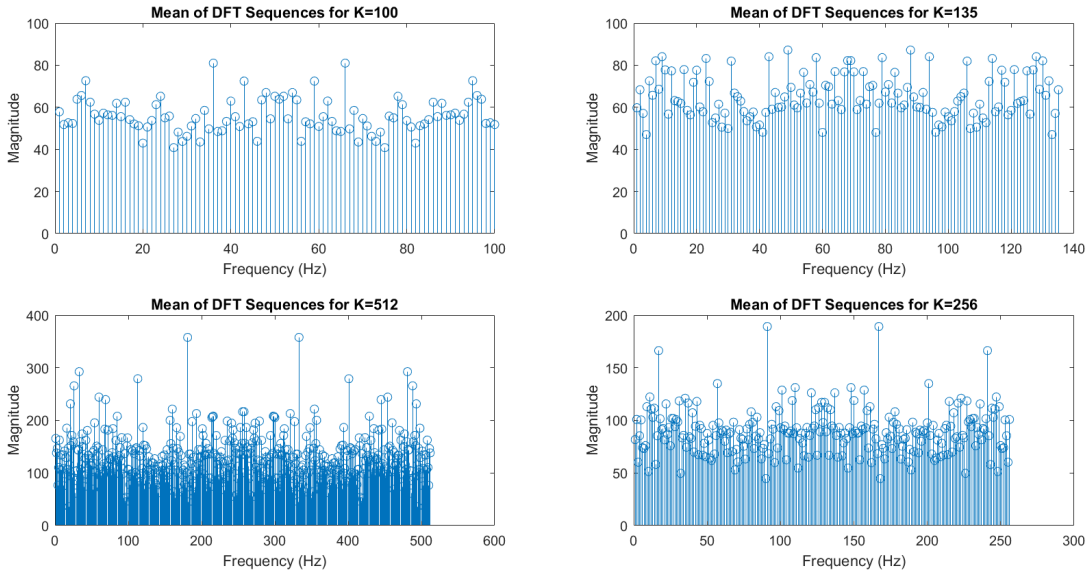


Figure 5: Effect of Other Values for K

2 Interpolation

In this experiment, we work with the first 20,000 samples of the music clip "Hallelujah" by Handel, which is available in MATLAB. The signal is denoted as $y[n]$. We generate several downsampled versions of this signal and apply interpolation techniques to reconstruct the original signal.

There are two primary interpolation methods that can be used to reconstruct the original signal:

- **Time Domain Interpolation by Zero Insertion**: This method involves inserting zeros between the samples of the downsampled signal to increase the signal length. This approach is computationally expensive and requires an optimal low-pass filter to effectively reconstruct the original signal. Due to its impracticality in many real-world applications, this method is less preferred.
- **Frequency Domain Interpolation by Zero Insertion**: This method involves zero-padding the frequency domain representation of the signal and then performing an inverse Fourier transform to reconstruct the signal. This approach is computationally more efficient and commonly used for practical signal processing tasks.

For this experiment, we will use the frequency domain interpolation by zero insertion due to its efficiency and practicality in signal processing. This approach allows us to achieve accurate signal reconstruction without the computational overhead associated with time domain methods.

2.1 Signal Generation

we extracted the first 20000 samples from the audio stream and divided it to signals, x2 , x3 and x4 The signal is defined as follows:

- $N = 20,000$
- $x = y(1 : N)$
- $x_2 = x(1 : 2 : N)$
  (Downsampled signal with a factor of 2)

- $x_3 = x(1:3:N)$
  (Downsampled signal with a factor of 3)
- $x_4 = x(1:4:N)$
  (Downsampled signal with a factor of 4)

2.2 Interpolation and Comparison

2.2.1 Interpolation of x_2 with $K = 1$

For the signal x_2, interpolation is performed with $K = 1$, which aims to reconstruct the signal at its original sampling rate.

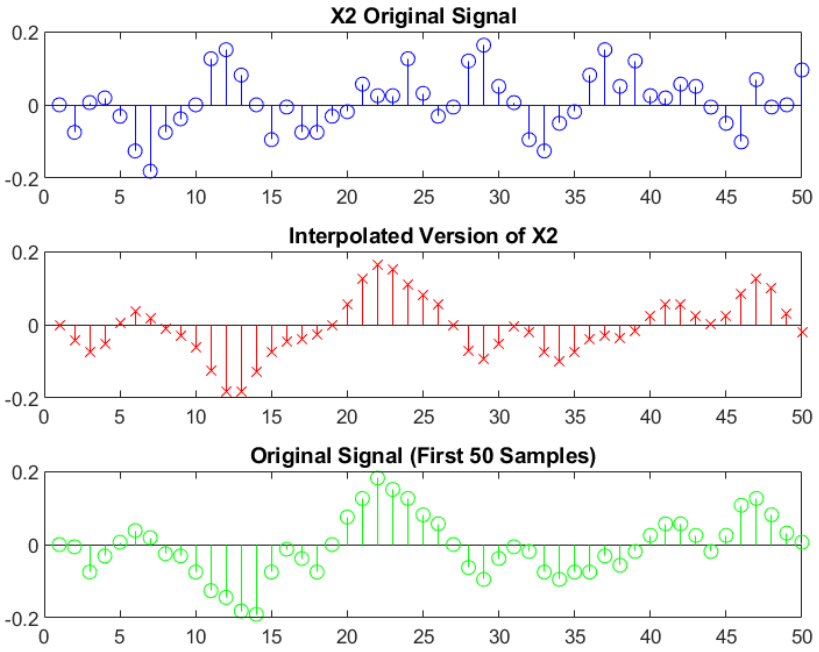


Figure 6: Interpolation of x_2 with $K = 1$

- The difference between the interpolated signal and the original signal x is computed using the 2-norm. The length of the signals involved may differ, so adjustments are made accordingly.
- The first 50 samples of both the interpolated signal and the original signal x are plotted on the same figure. This visual comparison helps to observe the waveform similarity between the interpolated and original signals.

2.2.2 Interpolation of x_3 with $K = 2$

For x_3, interpolation is performed with $K = 2$.

- Compute the 2-norm difference between the interpolated signal and the original signal x. Again, ensure that length mismatches are addressed.
- Plot the first 50 samples of both the interpolated signal and the original signal x to compare their waveforms visually.

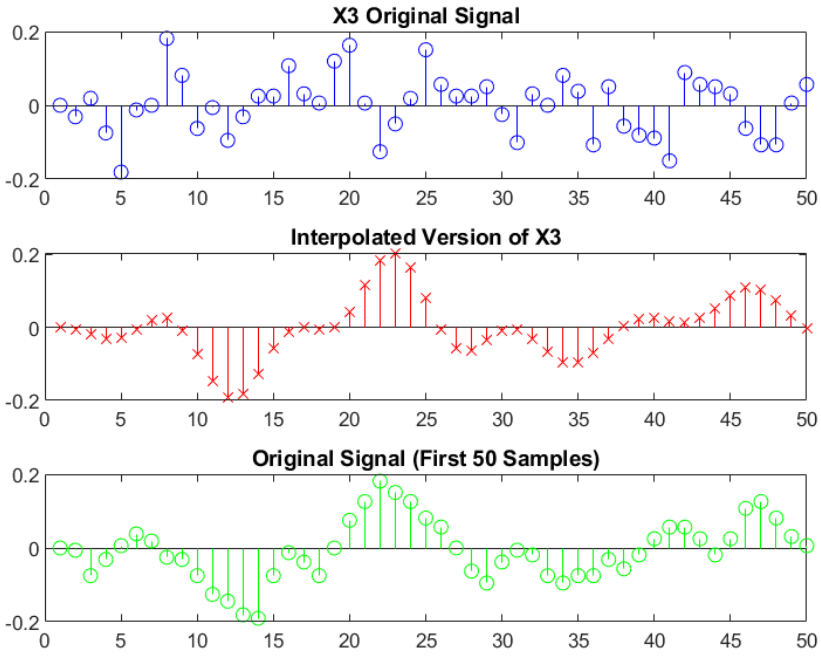


Figure 7: Interpolation of x_3 with $K = 2$

2.2.3 Interpolation of x_4 with $K = 3$

For x_4, interpolation is performed with $K = 3$.

- Compute the 2-norm difference between the interpolated signal and the original signal x, considering the length of the signals.
- Plot the first 50 samples of the interpolated signal and the original signal x to compare their waveforms visually.

2.3 Observations and Comments

After analyzing the graphs, it is evident that increasing the zero-insertion gap (the K value) results in a significantly smoother interpolated signal. However, as the K value increases, the 2-norm of the difference between the original signal and the interpolated signal also increases.

- 2 norm of the difference between x2 and x: 6.144750
- 2 norm of the difference between x3 and x: 8.365213
- 2 norm of the difference between x4 and x: 23.499839

This phenomenon occurs because adding more zeros to the initial signal's spectrum introduces additional frequency components. As a result, the energy within the frequency spectrum increases. Moreover, this additional zero-padding affects the frequency resolution of the Discrete Fourier Transform (DFT), leading to a coarser frequency resolution. Consequently, accurately representing the original signal's frequency components becomes more challenging. This discrepancy causes a noticeable increase in the difference between the frequencies of the interpolated signal and the original signal.

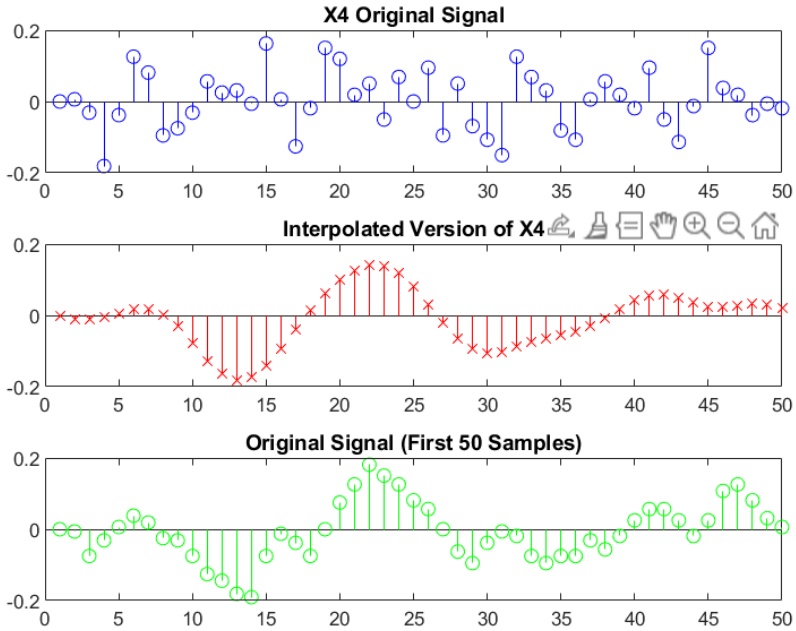


Figure 8: Interpolation of x_4 with $K = 3$

A Matlab codes

A.1 MATLAB Code for section 1

```
% Load the data
load('signal321.mat')
z = xn_test;  % Assign the signal to 'x'
fs = 128;  % Sampling frequency
N_total = length(z);  % Total number of samples

disp(['Total number of samples: ', num2str(N_total)]);

% Subsets
S1 = xn_test(1:128);
S2 = xn_test(1:256);
S3 = xn_test(1:512);
S4 = xn_test(1:1024);
S5 = xn_test(1:1792);

subsets = {S1, S2, S3, S4, S5};

% Plotting DFT magnitudes
for i = 1:5
    subset = subsets{i};
    X = fft(subset);  % Compute DFT
    X_mag = abs(X);
    freq = (0:length(X)-1) * (fs / length(X));  % Frequency vector
    subplot(5, 1, i);
```

```
    stem(freq, X_mag);
    title(['Magnitude Spectrum of S' num2str(i)]);
    grid on;
end
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Plotting DFT magnitude with DFT averaging
clf;
% Initialize parameters
total_samples = length(z);
L = 14;
K = 128;
i = 1; j = 128;
mean_dft = 0;
for n = 1:L
    subset = z(i:j);
    dft_mag = abs(fft(subset, K));
    mean_dft = mean_dft + dft_mag;

    i = j + 1;
    j = j + K;
end
mean_dft = mean_dft / L;
% Plot the magnitude
stem(mean_dft);
title('Mean of DFT Sequences');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Load signal data again
signal_file = load('signal321.mat');
signal_data = signal_file.xn_test;

% Plotting for several values of L
num_plots = 14;
rows = 2;
cols = 7;

figure('Position', [100, 100, 1200, 600]);

% Loop through each value of L
for L = 1:num_plots
    x = 1;
    y = 128;
    dft_mean_new = 0;

    % Compute the DFT magnitude and mean
    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
    subplot(rows, cols, L);
    stem(dft_mean_new);
    title([' L=', num2str(L)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
end

% Plotting for different values of K
K = [100, 135, 512, 256];

figure('Position', [100, 100, 1000, 800]);

% Loop through each value of K
for i = 1:length(K)
    x = 1;
    y = K(i);
    dft_mean = 0;
    subset_count = floor(length(signal_data) / K(i));

    % Compute the DFT magnitude and mean
    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 K values
    subplot(2, 2, i); % 2 rows and 2 columns
    stem(dft_mean);
    title(['Mean of DFT Sequences for K=', num2str(K(i))]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
end
```

A.2 MATLAB Code for section 2

Below is the MATLAB code used for the interpolation tasks described in this report:

```
[style=Matlab]
%loading signal
load handel;  % Load the music clip from MATLAB
N = 20000;    % Number of samples to consider
```

```
z = y(1:N);   % Extract the first 20,000 samples

% Creating sub samples

x2 = z(1:2:N);  % Take every second sample
x3 = z(1:3:N);  % Take every third sample
x4 = z(1:4:N);  % Take every fourth sample


%plottng original signal
figure;
subplot(1, 1, 1);
stem(z(1:50), 'b', 'Marker', 'o');
title('Original Signal (First 50 Samples)');
% Interpolating the Signals in Frequency Domain part a
K1 = 1;
X2 = fft(x2);
N4 = length(X2);
if mod(N4, 2) == 0
 N = N4/2;
 X2_zero_padded = [X2(1:N); X2(N+1)/2; zeros((K1*N4)-1, 1); X2(N+1)/2;
X2((N+2):N4)];
else
 N = (N4+1)/2;
 X2_zero_padded = [X2(1:N); zeros(K1*N4, 1); X2((N+1):N4)];
end
interpolated_x2 = (K1+1)*ifft(X2_zero_padded);
new_x2 = interpolated_x2(1:((K1+1)*(N4-1))+2);
difference_x2 = norm(new_x2 - z);
% Plot the first 50 samples of both X2 and interpolated version of X2
% using stem plots
figure;
subplot(3,1, 1);
stem(x2(1:50), 'b', 'Marker', 'o');
title('X2 Original Signal');
subplot(3, 1, 2)
stem(new_x2(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X2');
subplot(3, 1, 3);
stem(z(1:50), 'g', 'Marker', 'o');
title('Original Signal (First 50 Samples)');

% Interpolating the Signals in Frequency Domain part b

clf;
K2 = 2;
X3 = fft(x3);
N4 = length(X3);
if mod(N4, 2) == 0
 N = N4/2;
 X3_zero_padded = [X3(1:N); X3(N+1)/2; zeros((K2*N4)-1, 1); X3(N+1)/2;
X3((N+2):N4)];
else
 N = (N4+1)/2;
```

```
 X3_zero_padded= [X3(1:N); zeros(K2*N4, 1); X3((N+1):N4)];
end
interpolated_x3 = (K2+1)*ifft(X3_zero_padded);
new_x3 = interpolated_x3(1:((K2+1)*(N4-1))+2);
difference_x3 = norm(new_x3 - z);
% Plot the first 50 samples of both original and interpolated versions
% using stem plots
subplot(3, 1, 1);
stem(x3(1:50), 'b', 'Marker', 'o');
title('X3 Original Signal');
subplot(3, 1, 2)
stem(new_x3(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X3');
subplot(3, 1, 3);
stem(z(1:50), 'g', 'Marker', 'o');
title('Original Signal (First 50 Samples)');

% Interpolating the Signals in Frequency Domain part c

clf;
K3 = 3;
X4 = fft(x4);
N4 = length(X4);
if mod(N4, 2) == 0
 N = N4/2;
 X4_zero_padded = [X4(1:N); X4(N+1)/2; zeros((K3*N4)-1, 1); X4(N+1)/2;
X4((N+2):N4)];
else
 N = (N4+1)/2;
 X4_zero_padded = [X4(1:N); zeros(K3*N4, 1); X4((N+1):N4)];
end
interpolated_x4 = (K3+1)*ifft(X4_zero_padded);
new_x4 = interpolated_x4(1:((K3+1)*(N4-1))+4);
difference_x4 = norm(new_x4 - z);
% Plot the first 50 samples of both original and interpolated versions
% using stem plots
subplot(3, 1, 1);
stem(x4(1:50), 'b', 'Marker', 'o');
title('X4 Original Signal');
subplot(3, 1, 2)
stem(new_x4(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X4');
subplot(3, 1, 3);
stem(z(1:50), 'g', 'Marker', 'o');
title('Original Signal (First 50 Samples)');

fprintf('The difference between X2 and the original signal x in 2-norm:%f\n ', difference_x2);
fprintf('The difference between X3 and the original signal x in 2-norm:%f\n ', difference_x3);
fprintf('The difference between X4 and the original signal x in 2-norm:%f\n ', difference_x4);
```

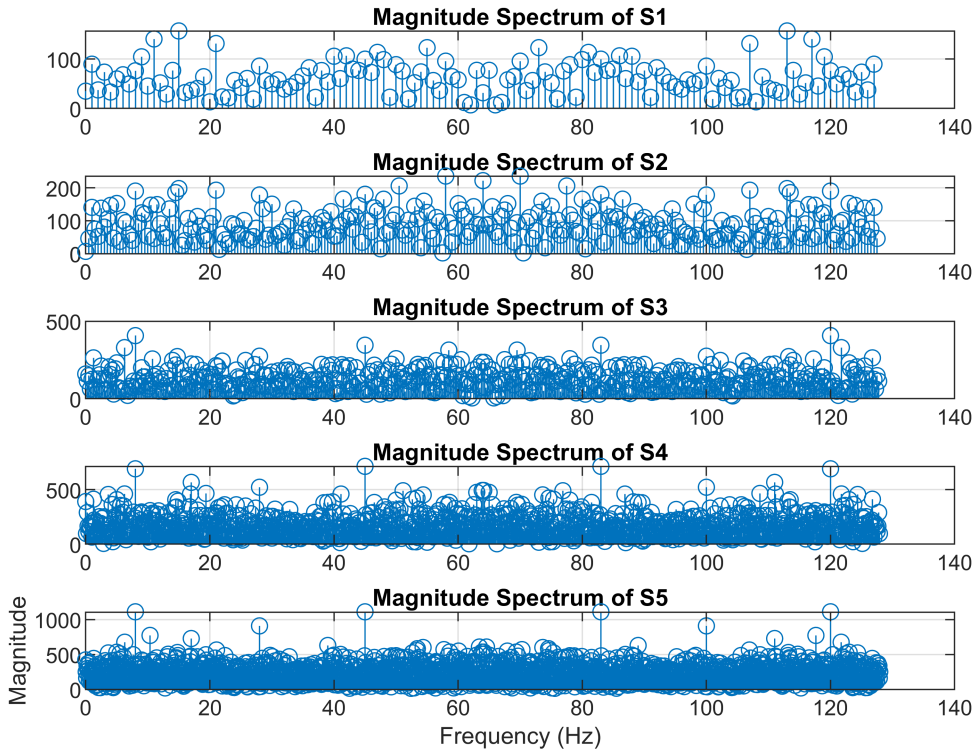
1 Harmonic Detection

```
%loading data
load('signal321.mat')
z = xn_test;  % Assign the signal to 'x'
fs = 128;  % Sampling frequency
N_total = length(z);  % Total number of samples

disp(['Total number of samples: ', num2str(N_total)]);
```

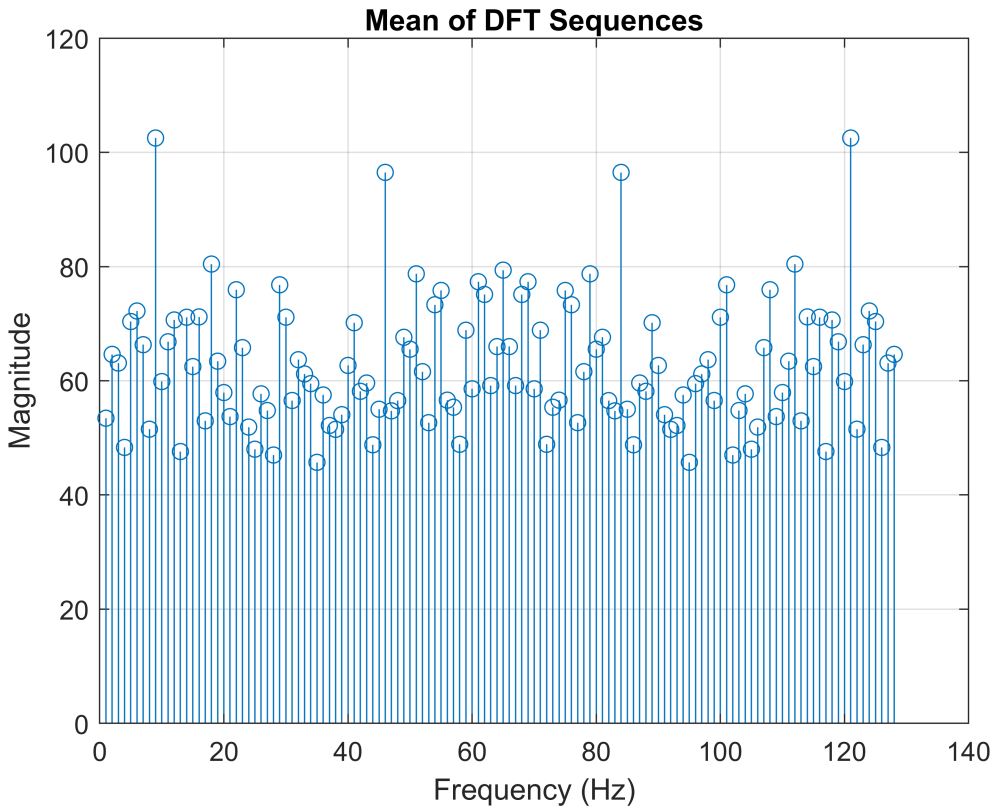
```
Total number of samples: 1793
```

```
%subsets
S1 = xn_test(1:128);
S2 = xn_test(1:256);
S3 = xn_test(1:512);
S4 = xn_test(1:1024);
S5 = xn_test(1:1792);

subsets = {S1, S2, S3, S4, S5};
```

```
%plotting DFT magnitudes
for i = 1:5
    subset = subsets{i};
    X = fft(subset);  % Compute DFT
    X_mag = abs(X);
    freq = (0:length(X)-1) * (fs / length(X));  % Frequency vector
    subplot(5, 1, i);
    stem(freq,X_mag );
    title(['Magnitude Spectrum of S' num2str(i)]);
    grid on;
end
 xlabel('Frequency (Hz)');
    ylabel('Magnitude');
```



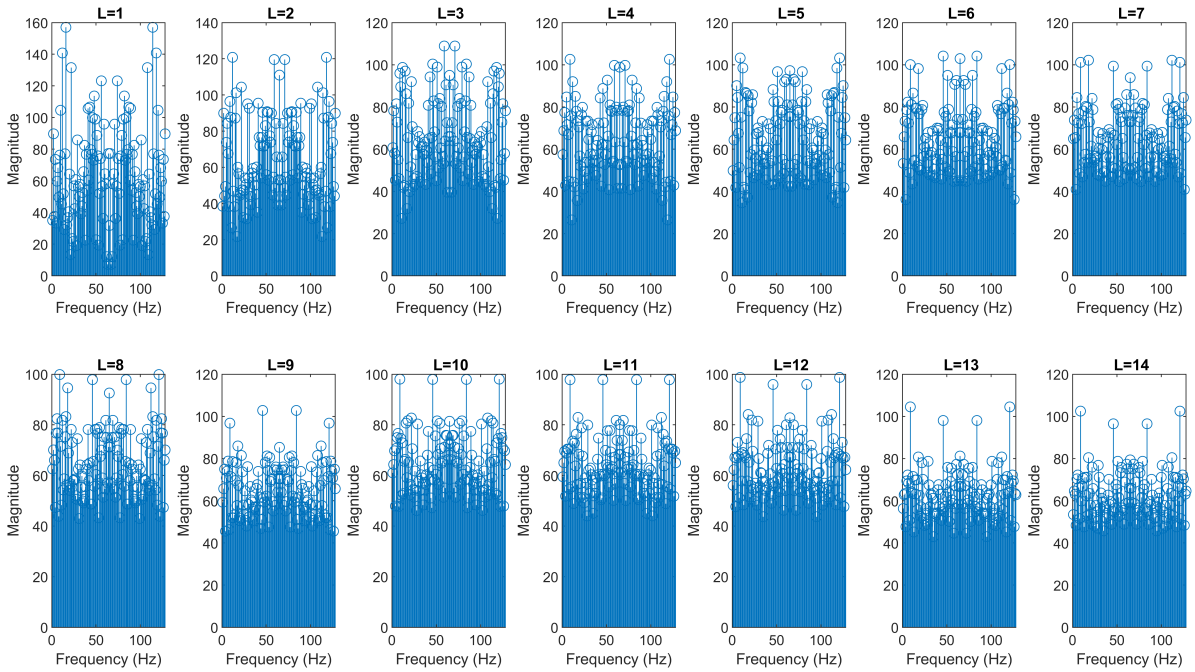

```
%plotting DFT magnitued with DFT averaging
    clf;
% Initialize parameters
total_samples = length(z);
L = 14;
K = 128;
i = 1; j=128;
mean_dft = 0;
for n = 1:L
 subset = z(i:j);
 dft_mag = abs(fft(subset, K));
 mean_dft = mean_dft + dft_mag;

 i = j+1;
 j = j+K;
end
mean_dft = mean_dft / L;
% Plot the magnitude
stem(mean_dft);
title('Mean of DFT Sequences');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on
```




```
signal_file = load('signal321.mat');
signal_data = signal_file.xn_test;
```

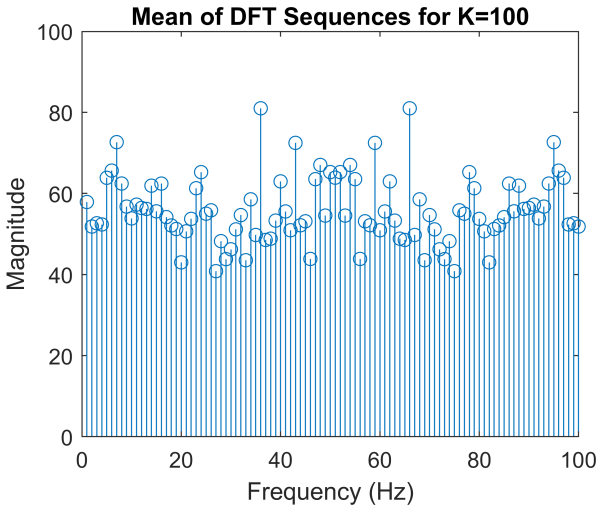
```
%plotting for several values of L
num_plots = 14;
rows = 2;
cols = 7; %


figure('Position', [100, 100, 1200, 600]);

% Loop through each value of L
for L = 1:num_plots
    x = 1;
    y = 128;
    dft_mean_new = 0;

    % Compute the DFT magnitude and mean
    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;
```

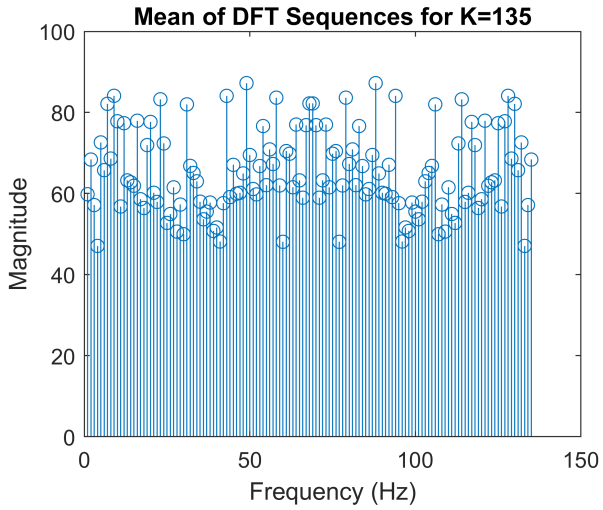
```
    end

    dft_mean_new = dft_mean_new / L;

    % Plot the average DFTs for different L values
    subplot(rows, cols, L);
    stem(dft_mean_new);
    title([' L=', num2str(L)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
end
```

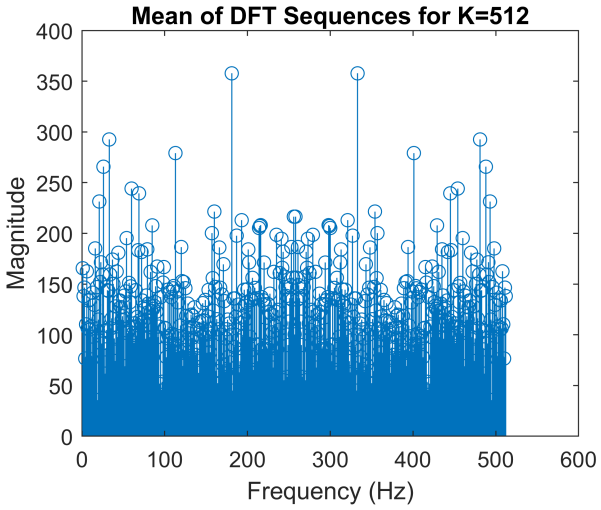


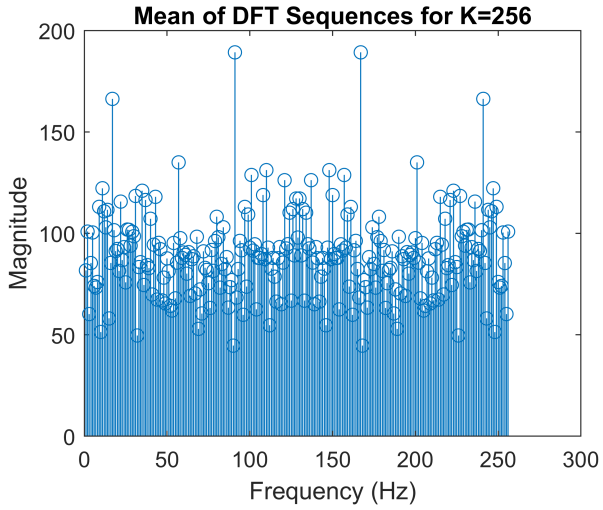

```
%plotting for different values of K
K = [100, 135, 512, 256];


figure('Position', [100, 100, 1000, 800]);

% Loop through each value of K
for i = 1:length(K)
    x = 1;
    y = K(i);
    dft_mean = 0;
    subset_count = floor(length(signal_data) / K(i));

    % Compute the DFT magnitude and mean
    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 K values
    subplot(2, 2, i); % 2 rows and 2 columns
    stem(dft_mean);
    title(['Mean of DFT Sequences for K=', num2str(K(i))]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
end
```







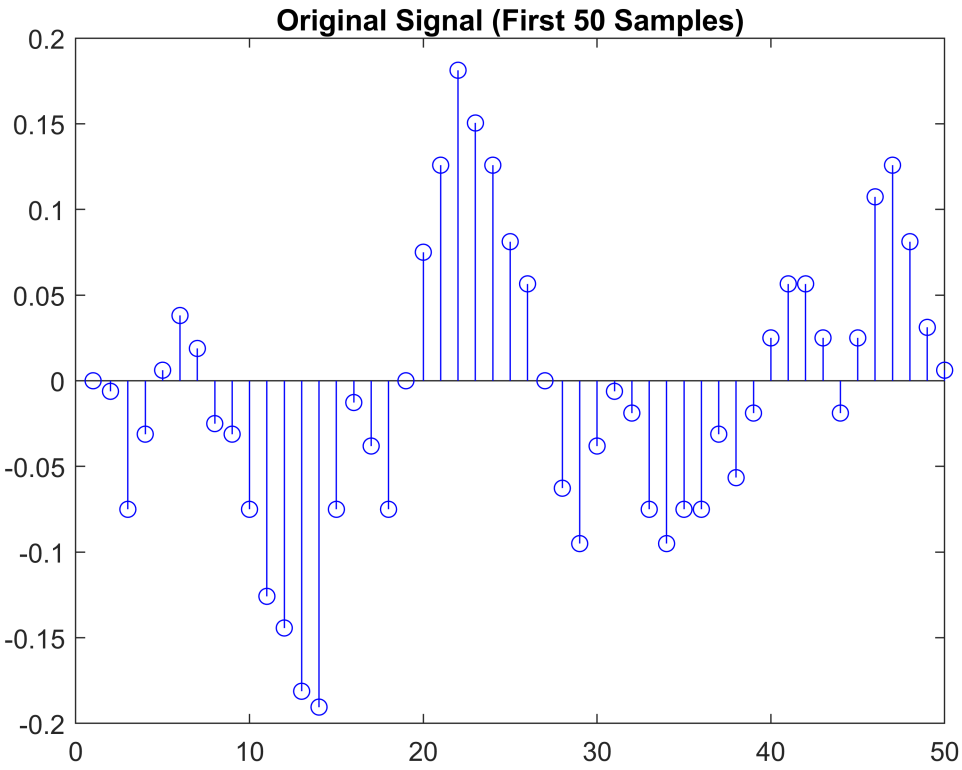


Interpolation

```
load handel;  % Load the music clip from MATLAB
N = 20000;    % Number of samples to consider
z = y(1:N);   % Extract the first 20,000 samples
```

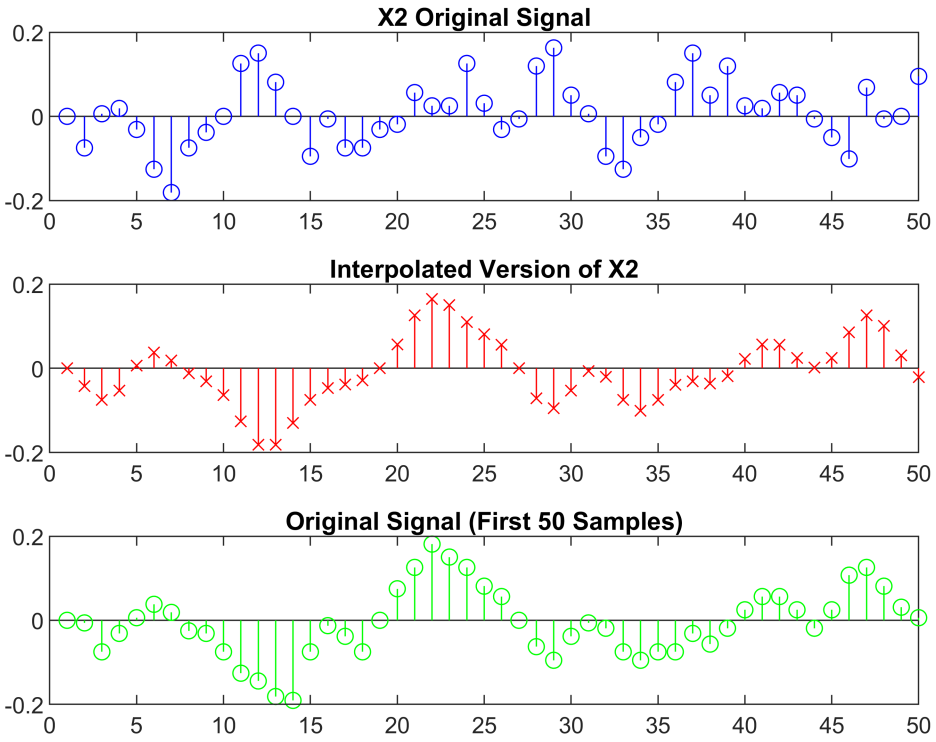
```
x2 = z(1:2:N);  % Take every second sample
x3 = z(1:3:N);  % Take every third sample
x4 = z(1:4:N);  % Take every fourth sample
```

```
figure;
subplot(1, 1, 1);
stem(z(1:50), 'b', 'Marker', 'o');
title('Original Signal (First 50 Samples)');
```



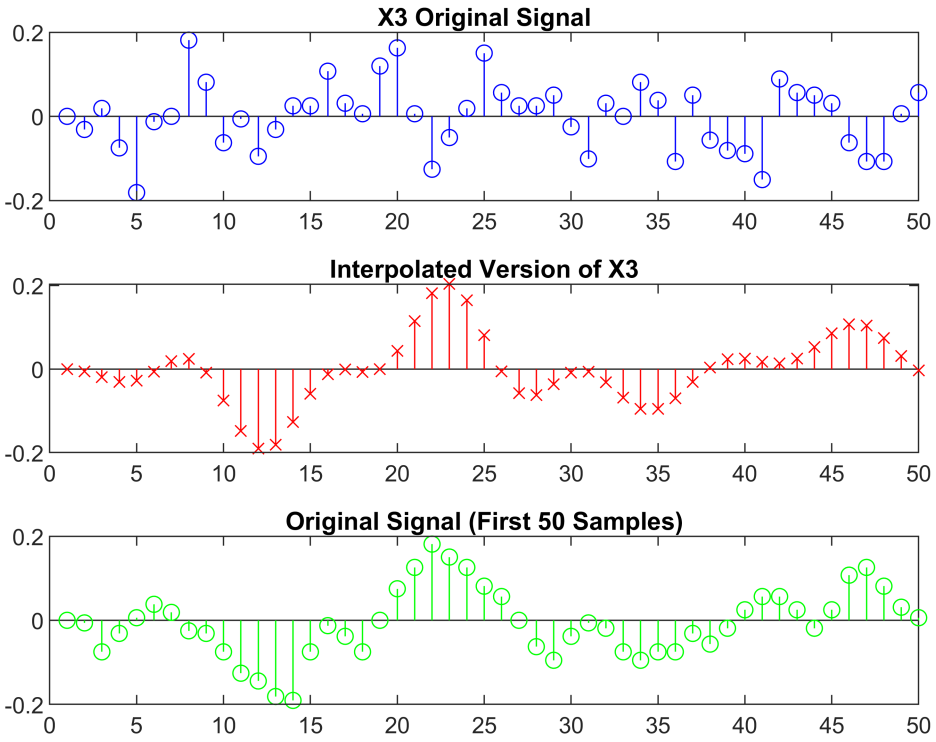

```
K1 = 1;
X2 = fft(x2);
N4 = length(X2);
if mod(N4, 2) == 0
 N = N4/2;
 X2_zero_padded = [X2(1:N); X2(N+1)/2; zeros((K1*N4)-1, 1); X2(N+1)/2;
X2((N+2):N4)];
else
 N = (N4+1)/2;
 X2_zero_padded = [X2(1:N); zeros(K1*N4, 1); X2((N+1):N4)];
end
```

```
interpolated_x2 = (K1+1)*ifft(X2_zero_padded);
new_x2 = interpolated_x2(1:((K1+1)*(N4-1))+2);
difference_x2 = norm(new_x2 - z);
% Plot the first 50 samples of both X2 and interpolated version of X2
% using stem plots
figure;
subplot(3,1, 1);
stem(x2(1:50), 'b', 'Marker', 'o');
title('X2 Original Signal');
subplot(3, 1, 2)
stem(new_x2(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X2');
subplot(3, 1, 3);
stem(z(1:50), 'g', 'Marker', 'o');
title('Original Signal (First 50 Samples)');
```



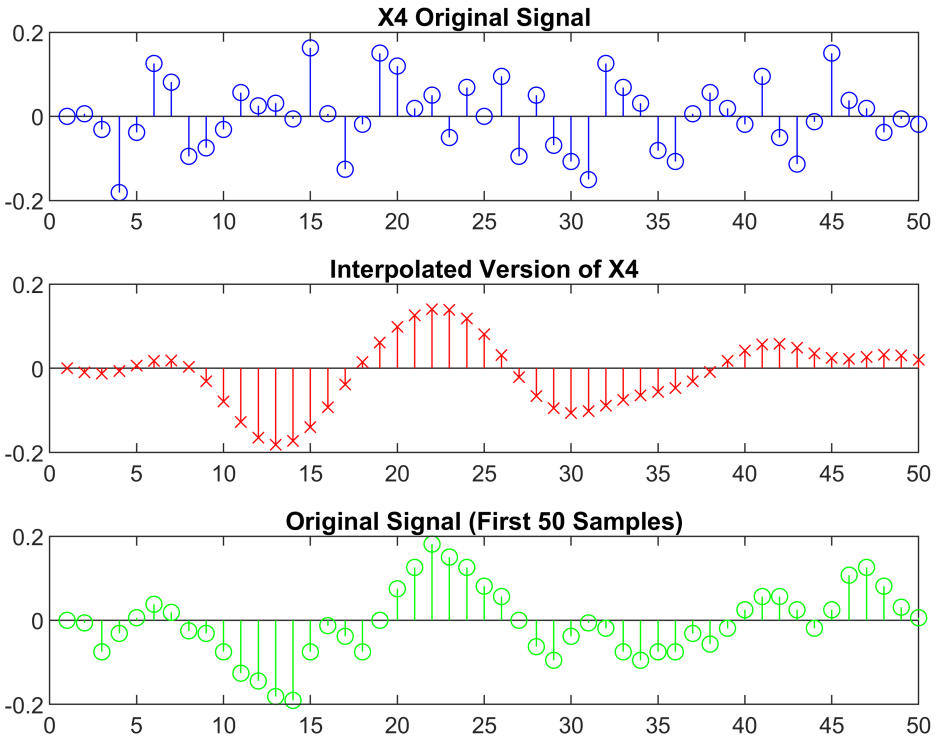

```
clf;
K2 = 2;
X3 = fft(x3);
N4 = length(X3);
if mod(N4, 2) == 0
 N = N4/2;
 X3_zero_padded = [X3(1:N); X3(N+1)/2; zeros((K2*N4)-1, 1); X3(N+1)/2;
X3((N+2):N4)];
else
 N = (N4+1)/2;
 X3_zero_padded= [X3(1:N); zeros(K2*N4, 1); X3((N+1):N4)];
```

```
end
interpolated_x3 = (K2+1)*ifft(X3_zero_padded);
new_x3 = interpolated_x3(1:((K2+1)*(N4-1))+2);
difference_x3 = norm(new_x3 - z);
% Plot the first 50 samples of both original and interpolated versions
% using stem plots
subplot(3, 1, 1);
stem(x3(1:50), 'b', 'Marker', 'o');
title('X3 Original Signal');
subplot(3, 1, 2)
stem(new_x3(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X3');
subplot(3, 1, 3);
stem(z(1:50), 'g', 'Marker', 'o');
title('Original Signal (First 50 Samples)');
```




```
clf;
K3 = 3;
X4 = fft(x4);
N4 = length(X4);
if mod(N4, 2) == 0
 N = N4/2;
 X4_zero_padded = [X4(1:N); X4(N+1)/2; zeros((K3*N4)-1, 1); X4(N+1)/2;
X4((N+2):N4)];
else
 N = (N4+1)/2;
 X4_zero_padded = [X4(1:N); zeros(K3*N4, 1); X4((N+1):N4)];
```

```
end
interpolated_x4 = (K3+1)*ifft(X4_zero_padded);
new_x4 = interpolated_x4(1:((K3+1)*(N4-1))+4);
difference_x4 = norm(new_x4 - z);
% Plot the first 50 samples of both original and interpolated versions
% using stem plots
subplot(3, 1, 1);
stem(x4(1:50), 'b', 'Marker', 'o');
title('X4 Original Signal');
subplot(3, 1, 2)
stem(new_x4(1:50), 'r', 'Marker', 'x');
title('Interpolated Version of X4');
subplot(3, 1, 3);
stem(z(1:50), 'g', 'Marker', 'o');
title('Original Signal (First 50 Samples)');
```




```
fprintf('The difference between X2 and the original signal x in 2-norm:%f\n ', difference_x2);
```

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

```
fprintf('The difference between X3 and the original signal x in 2-norm:%f\n ', difference_x3);
```

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

```
fprintf('The difference between X4 and the original signal x in 2-norm:%f\n ', difference_x4);
```

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