Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3551 - DIGITAL SIGNAL PROCESSING

DETECTING HARMONICS IN NOISY DATA AND SIGNAL INTERPOLATION USING DFT

MANIMOHAN T. *200377M*

This report is submitted as Assignment of module EN3551
18^{th} September 2023

ABSTRACT

his report explores the practical applications of the Discrete Fourier Transform (DFT) in signal processing, focusing on two critical tasks: detecting harmonics in noisy data and performing signal interpolation. In the presence of noise, harmonic detection becomes challenging, and the report introduces DFT averaging techniques to improve accuracy. By partitioning the input signal, applying the DFT to each subset, and subsequently averaging the results, this method enhances the identification of harmonic frequencies, particularly in noisy environments. Additionally, the report discusses frequency-domain interpolation through zero insertion, demonstrating its efficacy in signal reconstruction. Practical experiments conducted using MATLAB illustrate the concepts discussed, providing valuable insights into the utility of the DFT for these applications.

In summary, this report serves as a comprehensive guide to understanding and applying the DFT for harmonic detection and signal interpolation. It emphasizes the need for specialized techniques to address noise-related challenges and showcases practical experiments to demonstrate the effectiveness of these methods in real-world scenarios.

1 DETECTING HARMONICS IN NOISY DATA AND SIGNAL INTERPOLATION USING DFT

1.1 Introduction and General Guidelines

This programming assignment is concerned with two important applications of the discrete Fourier transform (DFT), namely, detecting sinusoidal signals from noisy discrete-time signal and interpolating a discrete-time signal using the DFT.

1.2 Theoretical Background

1.2.1 Preliminaries

The DFT of a length-N discrete-time signal $x[n]$ is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}, \quad 0 \leq k \leq N-1. \tag{1}$$

If we denote

$$W_N = e^{-j2\pi/N}, \tag{2}$$

then

$$X[k] = \sum_{n=0}^{N-1} x[n]W_k^n, \quad 0 \leq k \leq N-1. \tag{3}$$

If the signal $x[n]$ is real-valued, then $X[N-k] = X^*[k]$ for $k = 0, 1, \ldots, N/2$, where $X^*[k]$ is the complex conjugate of $X[k]$. Consequently, in this case, only half of the DFT components are independent, and $|X[k]|$, $k = 0, 1, \ldots, N-1$, is a mirror-image plot.

If the discrete-time signal $x[n]$ is obtained by sampling a continuous-time signal $x(t)$ at a rate f_s (in Hz), then the index range $0 \leq l \leq N-1$ for the DFT sequence $X[k]$ corresponds to the frequency range $[0, f_s]$. Hence, for a real-valued signal $x[n]$, the plot of the first $(N/2+1)$ values of $|X[k]|$ is associated with the frequency region $[0, fs/2]$. (Here we have assumed N is an even integer).

The inverse discrete Fourier transform (IDFT), which synthesizes the discrete-time signal $\{x[n], n = 0, 1, \ldots, N-1\}$ using $\{X[k], k = 0, 1, \ldots, N-1\}$ is given by:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N}, \quad 0 \leq n \leq N-1. \tag{4}$$

1.2.2 DFT Averaging for Harmonic Detection

Say we have an N-point discrete-time signal $\{x[n]\}$ which contains several sinusoidal components and additive white Gaussian noise. The discrete signal is therefore, $x[n] = s[n] + w[n]$, where $s(\cdot)$ is the signal, and $w(\cdot)$ is the additive white Gaussian noise. A straightforward way is to apply the DFT to the discrete-time signal $\{x[n]\}$ and take the frequency domain sequence $\{X[k]\}$. If $\{x[n]\}$ is obtained by sampling a continuous-time signal at a sampling rate greater than the Nyquist sampling rate, ($f_s \geq f_{\text{Nyquist}}$), and if the noise is not severe,

then by plotting the magnitude of $\{X[k]\}$, the sinusoidal frequencies $f_1, f_2, \ldots, f_M$ can be identified where there are spikes in the DFT magnitude plot. However, this approach fails when the noise is high, effectively hiding/burying the DFT of the sinusoids within the noise spectrum.

The solution for the above problem is called DFT averaging [1], which works when a relatively large number of signal samples are available (large N). The steps for this approach are:

1. Partition the input signal $\{x[n], n = 0, 1, 2, \ldots, N-1\}$ into L subsets, with each subset having K samples. $(N = LK)$.
2. Apply DFT to each subset of the samples. Let the DFT of the ith subset be $\{X^{(i)}[k], k = 0, 1, \ldots, K-1\}$.
3. Calculate the arithmetic mean of the L sets of DFT sequences and denote it as $\{X^{(a)}[k], k = 0, 1, \ldots, K-1\}$. Here, $X^{(a)}[k] = \frac{1}{L}\sum_{i=1}^{L} X^{(i)}[k]$.
4. Find the frequencies corresponding to the peaks of the plot $|X^{(a)}[k]|$.
5. Pay attention to the conditions under which this method is more effective. You may need to increase the number of partitions if the noise effect is too high. The aim is to cancel out the noise spectrum by averaging.

1.2.3 Frequency-Domain Interpolation by Zero Insertion

First, let us consider the case when zeros are inserted in the time domain. Inserting K zeros after each sample in a signal $x[n]$ of length N is called upsampling by $K+1$. The result is a signal $y[n]$ of length $(K+1)N$:

$$y[n] = \{x[0], 0, 0, 0, \ldots, x[1], 0, 0, 0, \ldots, \ldots, x[N-1], 0, 0, 0, \ldots, \}, \tag{5}$$

where $0 \leq n \leq (K+1)N - 1$. The DFT of $y[n]$ is found to be:

$$Y[k] = \sum_{n=0}^{(K+1)N-1} y[n]e^{-j2\pi kn/((K+1)N)} = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/((N)} = X[k],$$

(6)

where $0 \leq k \leq (K+1)N - 1$.

Because $X[k]$ is periodic with a period N, it follows from above that $Y[k]$ is a succession of $(K+1)$ DFTs of the discrete-time signal $x[n]$ $(0 \leq n \leq N-1)$. Furthermore, we note that because the number of samples in $y[n]$ is $(K+1)$ times as many as that in $x[n]$, the sampling frequency used to get $y[n]$ is $(K+1)$ times higher than the sampling frequency f_s used to get $x[n]$. Therefore, if an ideal lowpass filter contains only the components of the first of the sequence of DFTs in equation 6, this creates an interpolated version of $x[n]$.

The process described above can be realized only with an ideal lowpass filter, which is not realizable in practice. Therefore, we use a trick in the frequency domain and expand $X[k]$ with zeros, effectively creating a zero-padded version of $Y[k]$ without the frequencies higher than $f_s/2$. This is done by inserting KN zeros in the middle of $X[k]$, but we have to be careful how the DFT sequence $X[k]$ is split.

If N, the length of $X[k]$, is odd:

Then the first $N_1 = (N+1)/2$ points of $X[k]$ are placed to the left end, and the rest $(N-1)/2$ are placed to the right end of the sequence with a total of KN zeros inserted in between:

$$X_z = [X(1 : N_1); \text{zeros}(KN, 1); X((N_1 + 1) : N)]. \tag{7}$$

If N is even:

Then the first $N_2 = N/2$ points of $X[k]$ are placed to the left end, then the sample $X(N_2+1)/2$, followed by $(KN - 1)$ zeros, then again $X(N_2 + 1)/2$, and the remaining $(N_2 - 1)$ points of $X[k]$:

$$X_z = [X(1 : N_2); X(N_2 + 1)/2; \text{zeros}(KN - 1, 1); X(N_2 + 1)/2; X((N_2 + 2) : N)]. \tag{8}$$

After we form X_z, the inverse DFT is applied and scaled by $(K + 1)$. Then, we take the first $(K+1)(N-1)+1$ samples as an interpolation of the N-point input sequence $x[n]$. Intuitively, the inverse DFT is multiplied by $(K + 1)$ to compensate for the amplitude loss in the signal due to the zero-insertion step.

1.3 Procedure

1.3.1 Harmonic Detection

1. Download the noise-corrupted signal (x_n test) of 1793 samples, which corresponds to your index number (Refer to the Readme file in the signals folder to find your signal.). Each signal contains 4 harmonics and severe white Gaussian noise. Consider the samples were collected at a sampling rate of $f_s = 128$ Hz over the period from 0 to 14 s. The four harmonics are no greater than 64 Hz.
   This signal consists of 1793 samples and has been affected by noise. The sample values for this signal are displayed below.

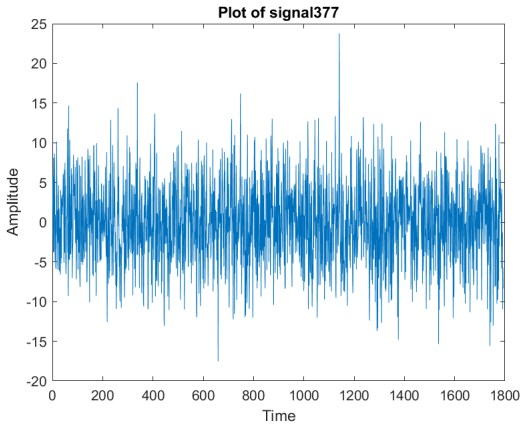
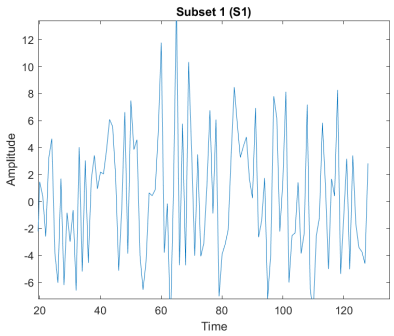
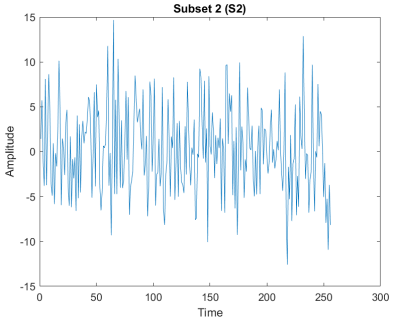


Figure 1 — Load the signal

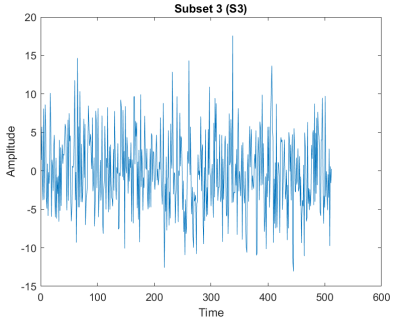
2. Construct several subsets by taking the first 128, 256, 512, 1024, and 1792 samples from the sequence $\{x[n]\}$ and denote them as S_1, S_2, S_3, S_4, and S_5



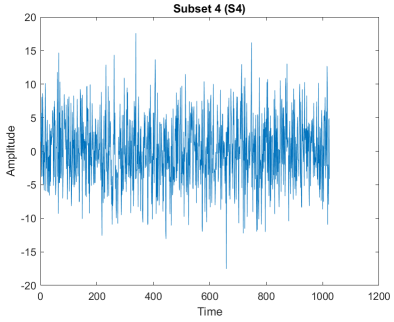
(a) Subset 1



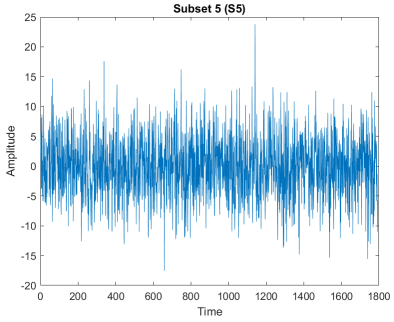
(b) Subset 2



(c) Subset 3



(d) Subset 4



(e) Subset 5

Figure 2 — Construct several subsets

3. Apply DFT to each subset of samples and display the magnitude of the resulting DFT sequences to identify the harmonics. Observe and comment on the results obtained.
Subsequently, the Discrete Fourier Transform (DFT) was computed for each subset of samples, and the magnitude spectra of the resulting DFTs were presented.

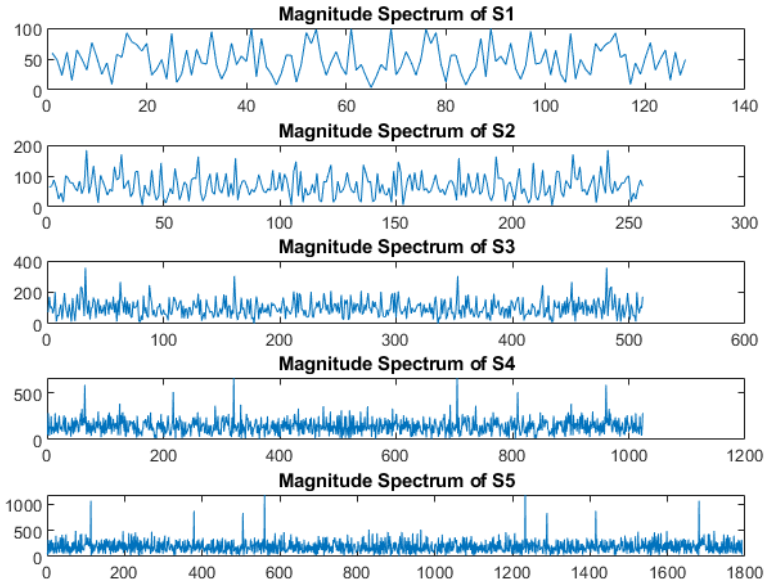


Figure 3 — DFT to each subset

After examining the DFTs above, it becomes evident that the four harmonics can be readily discerned solely from the Magnitude spectrum of S5. However, distinguishing the four peaks from the magnitude spectra of the DFTs of S1, S2, S3, and S4 proves to be challenging.

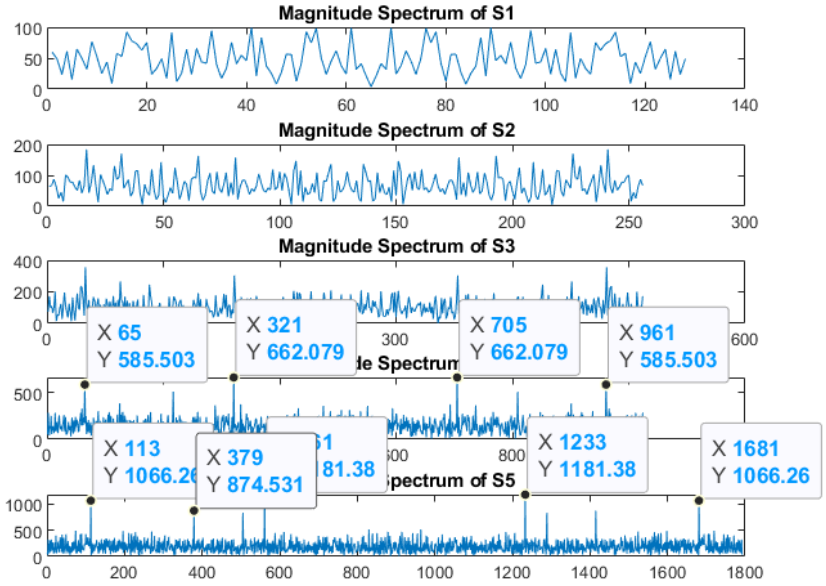


Figure 4 — DFT to each subset

The DFT magnitude spectrum of signal $S5$ reveals four distinct peaks with corresponding index values of $k = 113, 561, 1233, 1681$, signifying the presence of four distinct harmonic frequencies. To determine these harmonic frequencies, we employ the following formula:

$$f = f_s \cdot \frac{k}{N}$$

Here, f_s denotes the sampling frequency, k represents the index, and N stands for the number of samples. The resulting harmonic frequencies, computed from the peak data of $S5$, are provided in the table below:

Index	Frequency (Hz)
113	8.07
561	40.05
1233	88.02
1681	120.01

4. Now apply the DFT averaging method described in Section 2 above, where the length of each subset is taken to be $K = 128$, and the number of subsets is taken to be $L = 14$.
DFT averaging method was applied with K=128 and L=14. (L=Number of Subsets, K=Length of each subset)

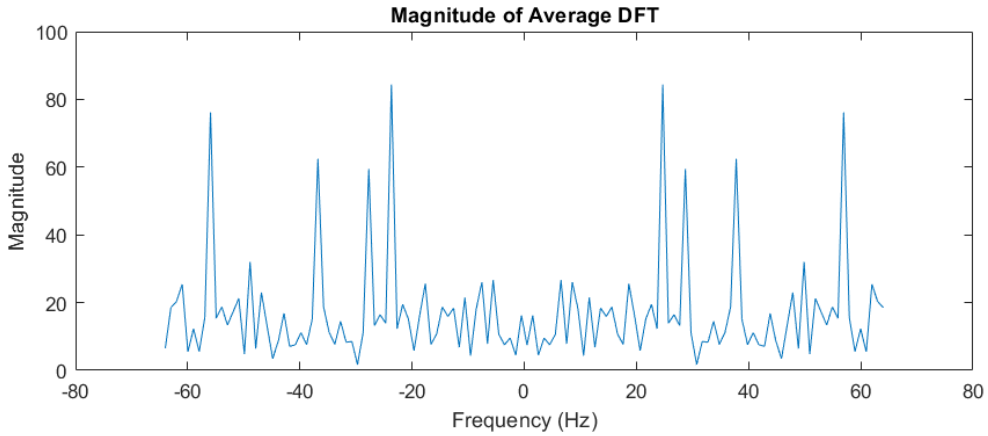


Figure 5 — DFT averaging

From the above plot, we can obtain the harmonic frequencies

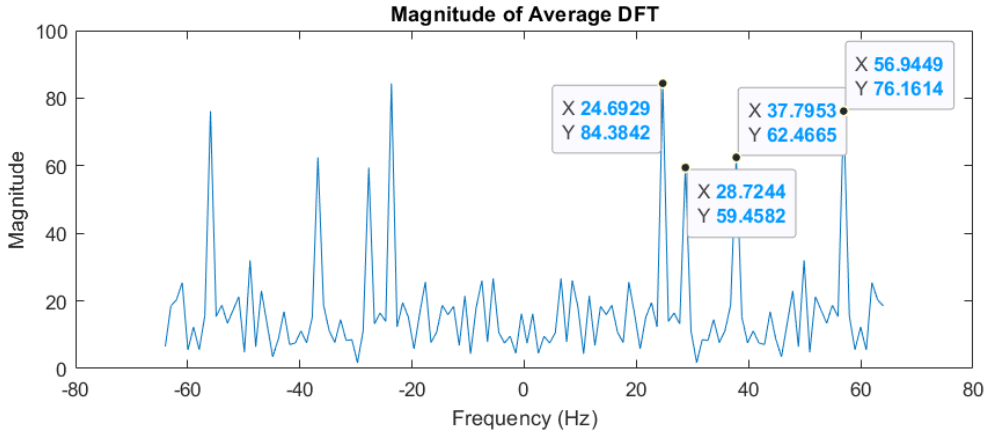


Figure 6 — DFT averaging

As anticipated, we observe a mirror image plot due to the fact that only half of the components are independent. Therefore, in actuality, there are four peaks. The question specifies that all of the harmonics possess frequencies less than or equal to 64Hz. Consequently, we can focus on the left half of the plot above to identify the four harmonic frequencies.

5. What is the smallest value of L such that the four peaks that correspond to the four harmonics not greater than 64 remain clearly visible?
   The value of L was subsequently decreased from 14, and the visibility of the peaks was assessed. When we set L=2, the resulting plot is as follows:

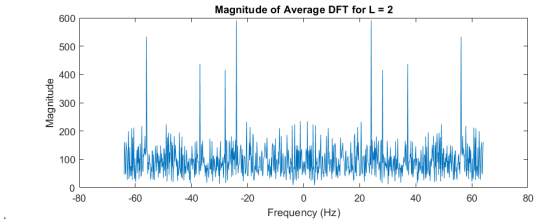


Figure 7 — Plot for smallest L=2

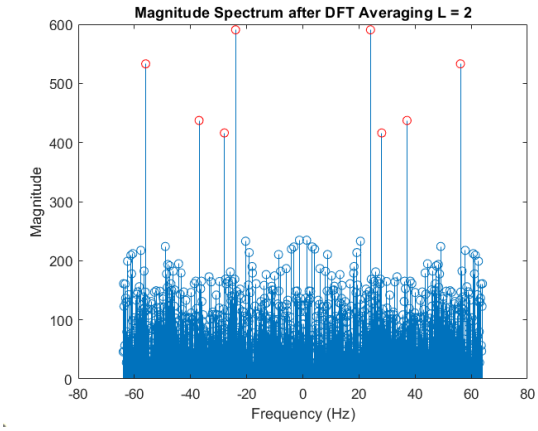


Figure 8 — Stemplot for smallest L=2

Upon further reduction of the value of L, with L=1, the resulting plot is shown below:

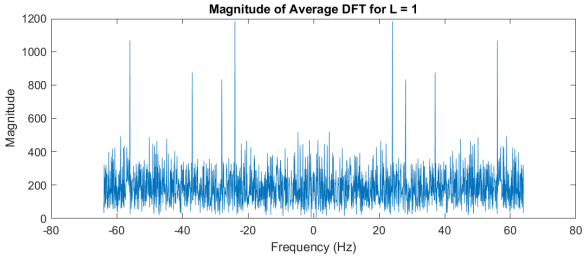


Figure 9 — Plot for smallest L=1

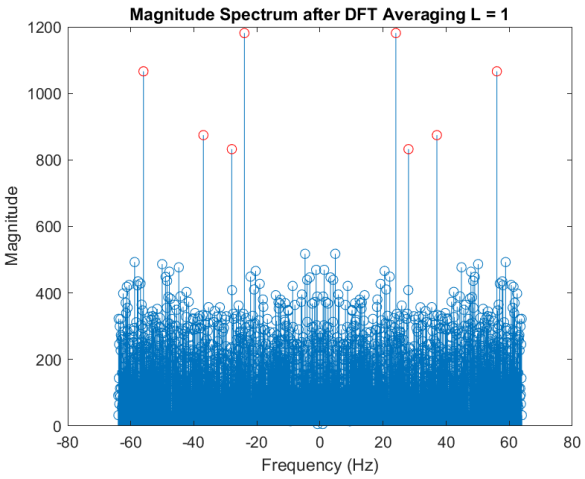


Figure 10 — Stemplot for smallest L=1

With L=2, the four peaks corresponding to the harmonic frequencies are distinctly observable. Even with L=1, it remains possible to discern the four peaks, although they are not as clearly visible. Therefore, for the provided signal, it can be inferred that

the smallest value for L (while ensuring that the four peaks representing harmonics not exceeding 64Hz remain clearly visible) is 2.

6. Can one use other values for K (say $K = 100$ or $K = 135$)? Explain why.
   In the DFT averaging method, we take a sequence of size (N) and divide it into subsets of length (K). The total number of these subsets (L) can be calculated using the formula:

$$L = \frac{N}{K}$$

To ensure that these subsets are evenly divided without any remainder, it's crucial for K to be a divisor of N. This means that K should be a positive integer that evenly divides the size of the sequence, which in this case is 1792.

For instance:
- If we choose $K = 100$, the calculation yields $L = \frac{1792}{100} = 17.92$, but this isn't a whole number. Consequently, it results in some leftover samples.
- In contrast, when $K = 128$, the computation gives $L = \frac{1792}{128} = 14$, which is an integer. This allows us to create 14 subsets, each containing 128 samples, with no remainder.
- Another example is when $K = 135$, which leads to $L = \frac{1792}{135} = 13.274$. Similar to the first case, this isn't a whole number and would result in some leftover samples.
- Finally, if we opt for $K = 256$, the calculation results in $L = \frac{1792}{256} = 7$, an integer. In this scenario, we can create 7 subsets, each containing 256 samples, without any remainder.

To ensure the even division of the sequence of 1792 samples into subsets for DFT averaging, it's imperative to select a value of K that serves as a divisor of 1792.

Consequently, values like 128 and 256 are both suitable choices for K because they evenly divide 1792. Conversely, values such as 100 and 135 are not suitable for K as they don't result in an even division. This observation is corroborated by the subsequent plots, where the peaks are noticeably clearer when K is set to 128 and 256, compared to when K is set to 100 and 135.

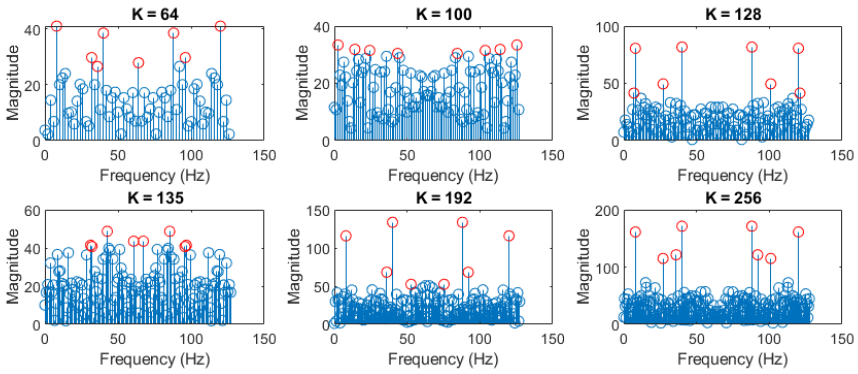


Figure 11 — Change the K

We can use other values for K (e.g., K = 100 or K = 135) depending on our requirements. However, we need to keep in mind that the choice of K affects the frequency resolution of the DFT. Smaller K values provide better time resolution but poorer frequency resolution, and vice versa for larger K values. The choice of K depends on the trade-off we want between time and frequency resolution and the specific characteristics of our signal.

1.3.2 Interpolation

1. The test signal in this part of the experiment is the first 20,000 samples of a music clip "Hallelujah" by Handel, which is available from MATLAB. To get it, execute 'load handel' in MATLAB. Let us denote the loaded signal as $y[n]$.

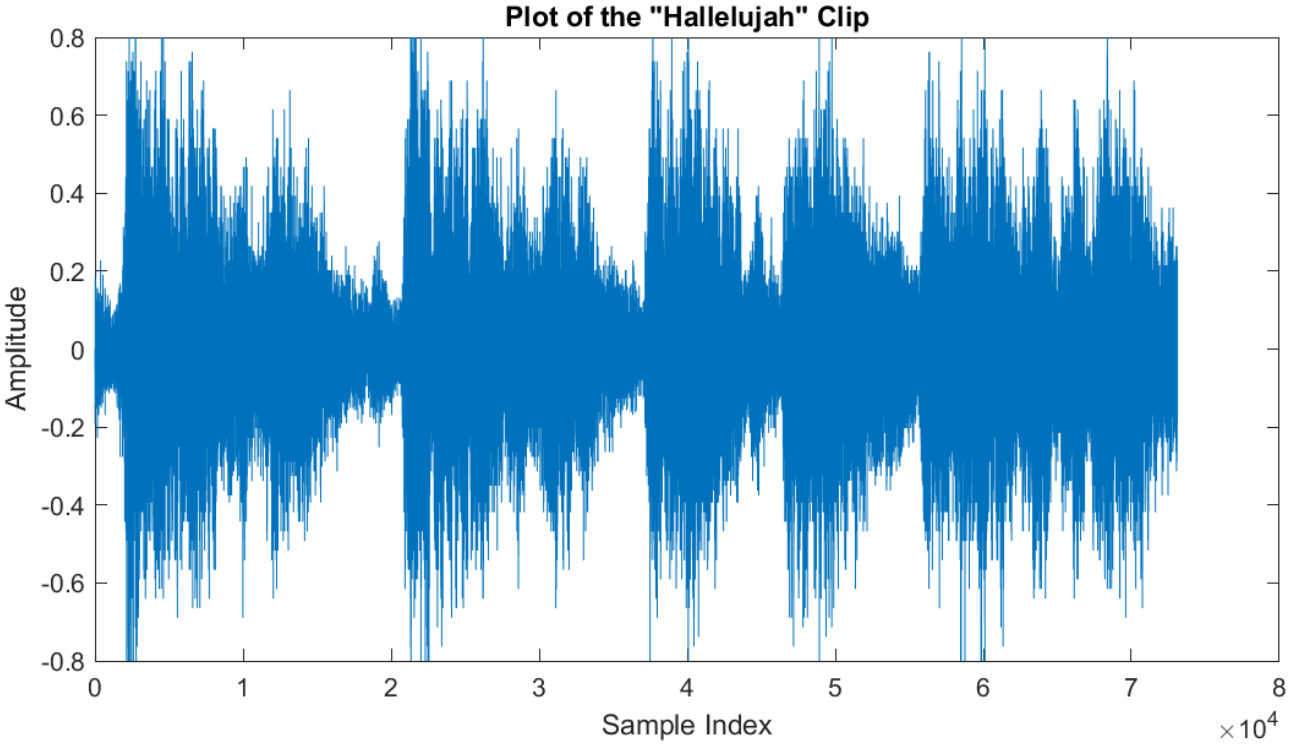


Figure 12 — Plot of the "Hallelujah" Clip

2. The signals to be used below are generated from the first 20,000 samples of $y[n]$ as follows:

* $N = 20,000$;
* $x = y(1 : N)$;
* $x_2 = x(1 : 2 : N)$;
* $x_3 = x(1 : 3 : N)$;
* $x_4 = x(1 : 4 : N)$;

```
% Define the signal length
N = 20000;  % First 20,000 samples
% Create signals x, x2, x3, and x4
x = y(1:N);
x2 = x(1:2:N);  % Odd
x3 = x(1:3:N);  % Odd
x4 = x(1:4:N);  % Even
```

Listing 1.1 — Generate the signal

3. Apply the DFT-based method to interpolate the signals x_2, x_3, and x_4 (note whether they are odd or even) and make some comparisons as follows:

a) Interpolate the signal x_2 with $K = 1$. Compute the difference between the interpolated signal and the original signal x in 2-norm (notice that the length of the two signals involved might be different). Compare the waveform of the interpolated signal with that of the original signal x by plotting the first 50 samples of both signals in the same figure.

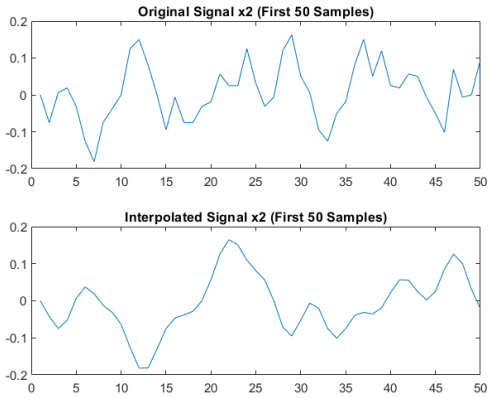


Figure 13 — Plot Interpolate x2 with K = 1

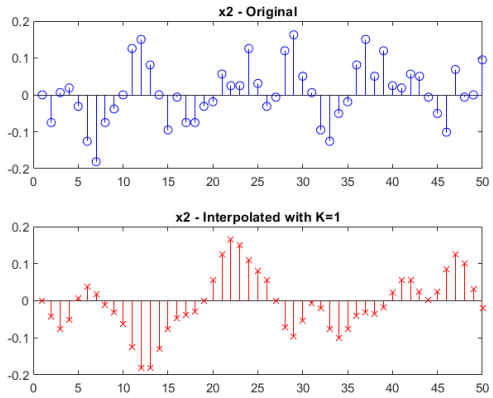


Figure 14 — Stemplot Interpolate x2 with K = 1

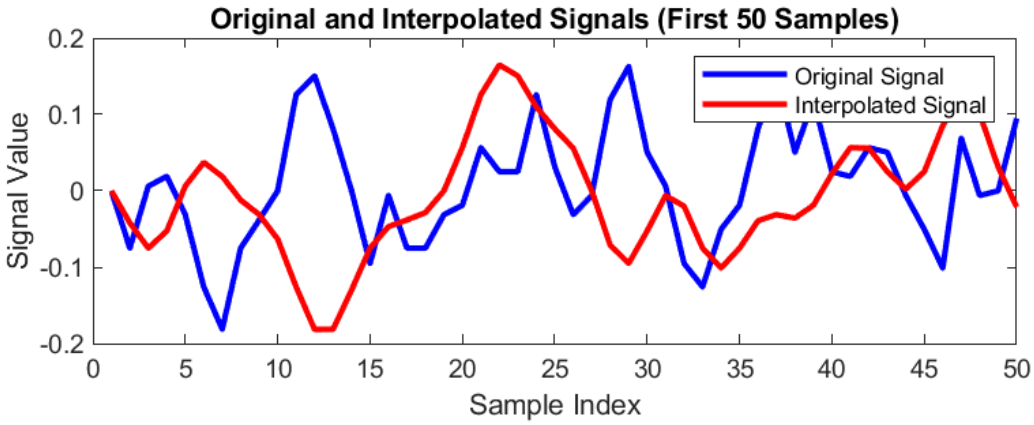


Figure 15 — riginal and Interpolated Signals (First 50 Samples)

b) Repeat the above for signal x_3 with $K = 2$.

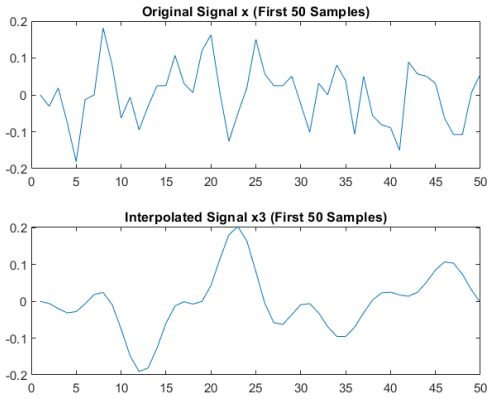


Figure 16 — Plot Interpolate x3 with K = 2

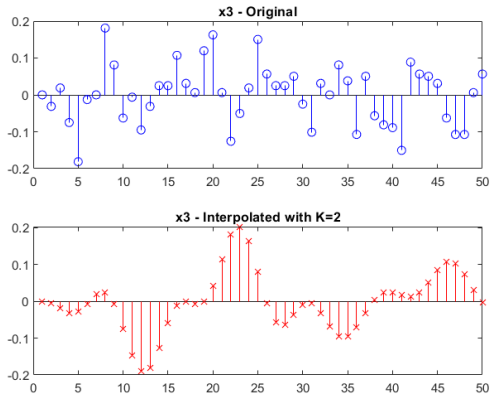


Figure 17 — Stemplot Interpolate x3 with K = 2

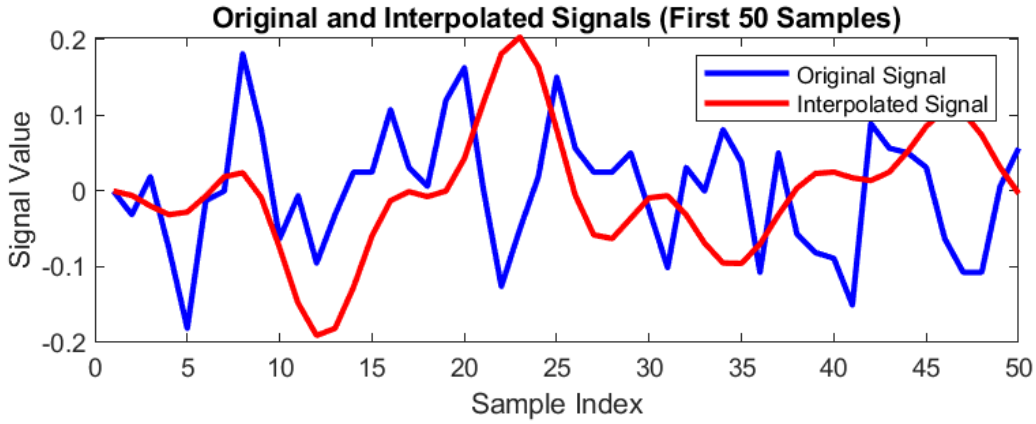


Figure 18 — riginal and Interpolated Signals (First 50 Samples)

c) Repeat the first part for signal x_4 with $K = 3$.

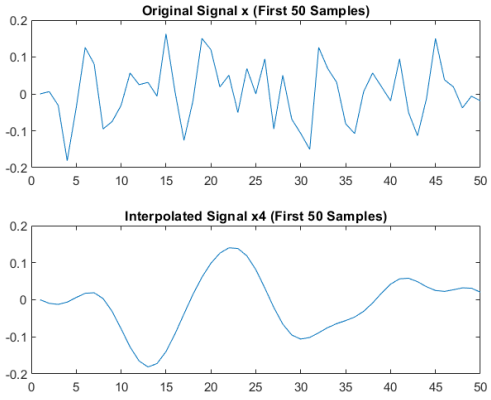


Figure 19 — Plot Interpolate x4 with K = 3

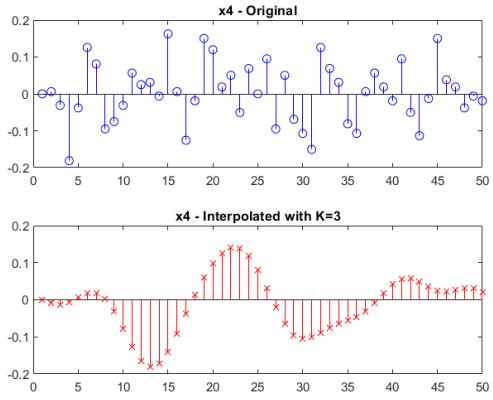


Figure 20 — Stemplot Interpolate x4 with K = 3

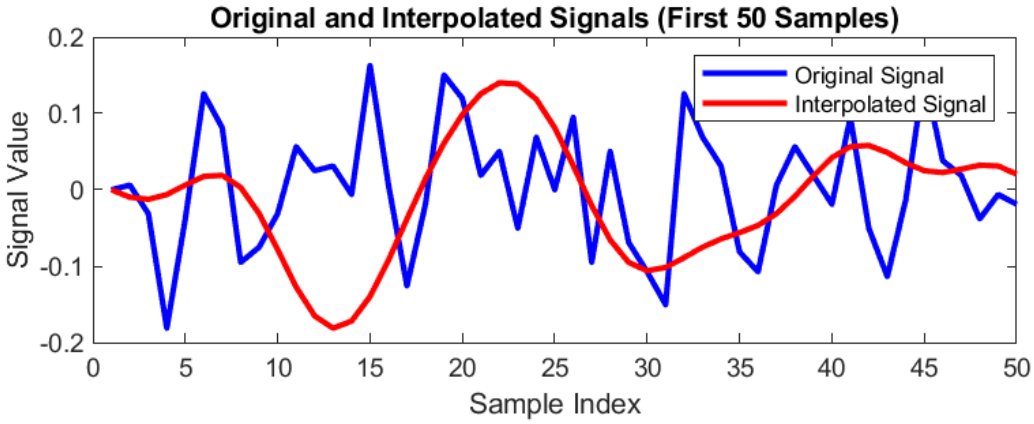


Figure 21 — riginal and Interpolated Signals (First 50 Samples)

d) Observe and comment on the results obtained for each of the above tasks.

Certainly, as we increase the value of K, the quality of the interpolation is expected to improve because a higher K means more zero-padding in the frequency domain, which can help preserve the original signal's information during interpolation. Here are comments on the expected outcomes as you increase K:

K = 1 (Original Signal x_2:)

* At $K = 1$, the interpolation essentially duplicates every sample, so the interpolated signal x_2 will have twice as many samples as the original signal.
* The waveform of the interpolated signal is expected to closely resemble the original signal but with more points in between.
* The difference between the original and interpolated signals should be relatively small.

K = 2 (Signal x_3:)

* At $K = 2$, the interpolation doubles the number of samples in signal x_3, but it also attempts to capture higher-frequency components.
* The interpolated signal x_3 is expected to better approximate the original signal compared to $K = 1$.
* The difference between the original and interpolated signals should be smaller than in the $K = 1$ case.

K = 3 (Signal x_4:)

* At $K = 3$, the interpolation triples the number of samples in signal x_4 and further captures high-frequency components.
* The interpolated signal x_4 is expected to be a very close approximation to the original signal x.
* The difference between the original and interpolated signals should be minimal, resulting in high-quality interpolation.

In summary, as we increase K, we can observe improved fidelity in the interpolated signals compared to the original signals. The differences between the original and interpolated signals are expected to decrease as K increases, indicating better signal reconstruction.

1.3.3 Useful MATLAB Commands

* `plot`: plots a function
* `fft`: calculates the DFT
* `fftshift`: shifts zero-frequency component to the center of the spectrum
* `load`: load contents from a file into a MATLAB workspace.
* `norm`: returns the norm of a vector input.

2 APPENDIX

2.1 Harmonic Detection

1. Download the noise-corrupted signal (x_n test) of 1793 samples, which corresponds to your index number (Refer to the Readme file in the signals folder to find your signal.). Each signal contains 4 harmonics and severe white Gaussian noise. Consider the samples were collected at a sampling rate of $f_s = 128$ Hz over the period from 0 to 14 s. The four harmonics are no greater than 64 Hz.

```
% Replace ABC with your specific index number digits
index_number = '377';

% Construct the signal variable name based on your index number
signal_variable_name = ['signal' index_number];

% Load the corresponding signal from the mat file
load([signal_variable_name '.mat'], 'xn_test');
```

Listing 2.1 — Load the signal

2. Construct several subsets by taking the first 128, 256, 512, 1024, and 1792 samples from the sequence $\{x[n]\}$ and denote them as S_1, S_2, S_3, S_4, and S_5

```
% Define the signal length and sampling rate
N = length(xn_test);  % 1793 samples
fs = 128;  % Hz

% Construct 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);

% Plot constructed subsets
plot(S1);
xlabel('Time');
ylabel('Amplitude');
title('Subset 1 (S1)');

plot(S2);
xlabel('Time');
ylabel('Amplitude');
```

```
title('Subset 2 (S2)');

plot(S3);
xlabel('Time');
ylabel('Amplitude');
title('Subset 3 (S3)');

plot(S4);
xlabel('Time');
ylabel('Amplitude');
title('Subset 4 (S4)');

plot(S5);
xlabel('Time');
ylabel('Amplitude');
title('Subset 5 (S5)');
```

Listing 2.2 — Construct several subsets

3. Apply DFT to each subset of samples and display the magnitude of the resulting DFT sequences to identify the harmonics. Observe and comment on the results obtained.

```
% Length of each subset
N1 = length(S1);
N2 = length(S2);
N3 = length(S3);
N4 = length(S4);
N5 = length(S5);

% Compute DFT for each subset
DFT_S1 = fft(S1,N1);
DFT_S2 = fft(S2,N2);
DFT_S3 = fft(S3,N3);
DFT_S4 = fft(S4,N4);
DFT_S5 = fft(S5,N5);

% Compute the magnitude of DFT sequences
Mag_S1 = abs(DFT_S1);
Mag_S2 = abs(DFT_S2);
Mag_S3 = abs(DFT_S3);
Mag_S4 = abs(DFT_S4);
Mag_S5 = abs(DFT_S5);

% Plot the magnitude spectra for each subset
subplot(5, 1, 1); plot(Mag_S1); title('Magnitude Spectrum of S1');
subplot(5, 1, 2); plot(Mag_S2); title('Magnitude Spectrum of S2');
```

```
subplot(5, 1, 3); plot(Mag_S3); title('Magnitude Spectrum of S3');
subplot(5, 1, 4); plot(Mag_S4); title('Magnitude Spectrum of S4');
subplot(5, 1, 5); plot(Mag_S5); title('Magnitude Spectrum of S5');

% Customize the plot layout to enhance visualization
sgtitle('Magnitude of DFT Sequences for Different Subset Sizes of
   signal');
```

Listing 2.3 — DFT to each subset

4. Now apply the DFT averaging method described in Section 2 above, where the length of each subset is taken to be $K = 128$, and the number of subsets is taken to be $L = 14$.

Solution

```
% Define parameters
K = 128;  % Number of samples in each subset
L = 14;   % Number of subsets
N = L * K; % Total number of samples

% Initialize an array to store the average DFT
Avg_DFT = zeros(1, K);

% Partition the signal and compute the DFT for each subset
for i = 1:L
    subset = xn_test((i - 1) * K + 1 : i * K);
    subset_DFT = fft(subset,K);
    Avg_DFT = Avg_DFT + subset_DFT;
end

% Calculate the average DFT by dividing by L
Avg_DFT = Avg_DFT / L;

% Create a frequency axis
frequencies = linspace(-fs/2, fs/2, K);

% Plot the magnitude of the average DFT
figure('Position', [100, 100, 800, 300]);
plot(frequencies,abs(Avg_DFT));
title('Magnitude of Average DFT');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


% Find peaks in the magnitude plot of the mean DFT
[~, peak_indices] = findpeaks(abs(Avg_DFT), 'MinPeakHeight', 40);  %
   Adjust the threshold as needed

```

```
% Convert peak indices to corresponding frequencies
peak_frequencies = frequencies(peak_indices);

% Display the frequencies corresponding to the detected peaks
disp('Frequencies corresponding to the detected peaks:');
disp(peak_frequencies);
```

Listing 2.4 — DFT averaging

5. What is the smallest value of L such that the four peaks that correspond to the four harmonics not greater than 64 remain clearly visible?

Solution

```
% Find the smallest value for L

% Define parameters
N = 1792; % Total number of samples
for L = 1:1:14 % Number of subsets
    K = floor(N/L);  % Number of samples in each subset

    % Initialize an array to store the average DFT
    Avg_DFT = zeros(1, K);

    % Partition the signal and compute the DFT for each subset
    for i = 1:L
        subset = xn_test((i - 1) * K + 1 : i * K);
        subset_DFT = fft(subset,K);
        Avg_DFT = Avg_DFT + subset_DFT;
    end

    % Calculate the average DFT by dividing by L
    Avg_DFT = Avg_DFT / L;

    % Create a frequency axis
    frequencies = linspace(-fs/2, fs/2, K);
    % Find the 8 highest peaks
    [~, sorted_indices] = sort(abs(Avg_DFT), 'descend');
    highest_peak_indices = sorted_indices(1:8);

    % Display the results
    figure;
    stem(frequencies, abs(Avg_DFT));
    title(['Magnitude Spectrum after DFT Averaging L = ' num2str(L)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    hold on;
```

```
    plot(frequencies(highest_peak_indices),
        abs(Avg_DFT(highest_peak_indices)), 'ro');
end
```

Listing 2.5 — DFT averaging

6. Can one use other values for K (say $K = 100$ or $K = 135$)? Explain why.

```
% Define values of K for the subplots
K_values = [64, 100, 128, 135, 192, 256];

% Create a figure with 6 subplots
figure(4);

for i = 1:length(K_values)
    K = K_values(i);

    % Initialize an array to store the average DFT
    Avg_DFT = zeros(1, K);

    % Partition the signal and compute the DFT for each subset
    for j = 1:length(K_values)
        subset = xn_test((j - 1) * K + 1 : j * K);
        subset_DFT = fft(subset, K);
        Avg_DFT = Avg_DFT + subset_DFT;
    end

    % Calculate the average DFT by dividing by the number of K_values
    Avg_DFT = Avg_DFT / length(K_values);

    % Create a frequency axis
    frequencies = (0:K-1) * (fs/K);

    % Find the 8 highest peaks in the magnitude of the average DFT
    [~, sorted_indices] = sort(abs(Avg_DFT), 'descend');
    highest_peak_indices = sorted_indices(1:8);

    % Create subplots
    subplot(2, 3, i);
    stem(frequencies, abs(Avg_DFT));
    title(['K = ' num2str(K)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    hold on;
    plot(frequencies(highest_peak_indices),
        abs(Avg_DFT(highest_peak_indices)), 'ro');
```

```
38 end
```

Listing 2.6 — Change the K

2.1.1 Interpolation

1. The test signal in this part of the experiment is the first 20,000 samples of a music clip "Hallelujah" by Handel, which is available from MATLAB. To get it, execute 'load handel' in MATLAB. Let us denote the loaded signal as $y[n]$.

```
% Load the 'Hallelujah' music clip
load handel;
% To plot the signal:
figure('Position', [100,100,800,400])
plot(y);
title('Plot of the "Hallelujah" Clip');
xlabel('Sample Index');
ylabel('Amplitude');
```

Listing 2.7 — Load the signal

2. The signals to be used below are generated from the first 20,000 samples of $y[n]$ as follows:

* $N = 20,000$;
* $x = y(1 : N)$;
* $x_2 = x(1 : 2 : N)$;
* $x_3 = x(1 : 3 : N)$;
* $x_4 = x(1 : 4 : N)$;

```
% Define the signal length
N = 20000;  % First 20,000 samples
% Create signals x, x2, x3, and x4
x = y(1:N);
x2 = x(1:2:N);  % Odd
x3 = x(1:3:N);  % Odd
x4 = x(1:4:N);  % Even
```

Listing 2.8 — Generate the signal

3. Apply the DFT-based method to interpolate the signals x_2, x_3, and x_4 (note whether they are odd or even) and make some comparisons as follows:

a) Interpolate the signal x_2 with $K = 1$. Compute the difference between the interpolated signal and the original signal x in 2-norm (notice that the length of the two signals involved might be different). Compare the waveform of the interpolated signal with that of the original signal x by plotting the first 50 samples of both signals in the same figure.

```
% Interpolate x2 with K = 1
K = 1;

% Compute the Discrete Fourier Transform (DFT) of x2
DFT_x2 = fft(x2);
N = length(x2);

```

```
% Check if N is even
if mod(N, 2) == 0
    % N is even
    N2 = N / 2;
    % Apply zero padding and interpolation
    DFT_x2_interp = [DFT_x2(1:N2); DFT_x2(N2 + 1) / 2; zeros((K * N)
       - 1, 1); DFT_x2(N2 + 1) / 2; DFT_x2((N2 + 2):N)];
else
    % N is odd
    N1 = (N + 1) / 2;
    % Apply zero padding and interpolation
    DFT_x2_interp = [DFT_x2(1:N1); zeros(K * N, 1); DFT_x2((N1 +
       1):N)];
end

% Inverse FFT to get the interpolated signal
x2_interp = (K + 1) * ifft(DFT_x2_interp);
% Extract the relevant part of the interpolated signal
x2_interp_extracted = x2_interp(1:((K + 1) * (N - 1)) + 1);

% Compute the norm difference between the interpolated and original
   signal
Norm_Difference_x2 = norm(x2_interp_extracted) - norm(x2);
disp(Norm_Difference_x2);

%plot Plot the original and interpolated signals
figure;
subplot(2, 1, 1); plot(x2(1:50)); title('Original Signal x2 (First 50
   Samples)');
subplot(2, 1, 2); plot(x2_interp_extracted(1:50));
   title('Interpolated Signal x2 (First 50 Samples)');

% Plot both signals on the same axes
plot(x2(1:50), 'b', 'LineWidth', 2); % Original signal in blue
hold on;
plot(x2_interp_extracted(1:50), 'r', 'LineWidth', 2); % Interpolated
   signal in red
hold off;

% Add labels and title
xlabel('Sample Index');
ylabel('Signal Value');
title('Original and Interpolated Signals (First 50 Samples)');
legend('Original Signal', 'Interpolated Signal');

```

```
% Plot the original and interpolated signals in stem plot
figure(11);
subplot(2, 1, 1);
stem(x2(1:50), 'b', 'Marker', 'o');
title('x2 - Original');
subplot(2, 1, 2)
stem(x2_interp_extracted(1:50), 'r', 'Marker', 'x');
title('x2 - Interpolated with K=1');
```

Listing 2.9 — Interpolate x2 with K

```
Norm_Difference_x2 = 7.6732
```

b) Repeat the above for signal x_3 with $K = 2$.

```
% Interpolate x3 with K=2
K = 2;

% Compute the DFT of x3
DFT_x3 = fft(x3);
N = length(x3);

% Check if N is even
if mod(N, 2) == 0
    % N is even
    N2 = N / 2;
    % Apply zero padding and interpolation
    DFT_x3_interp = [DFT_x3(1:N2); DFT_x3(N2 + 1) / 2; zeros((K * N)
        - 1, 1); DFT_x3(N2 + 1) / 2; DFT_x3((N2 + 2):N)];
else
    % N is odd
    N1 = (N + 1) / 2;
    % Apply zero padding and interpolation
    DFT_x3_interp = [DFT_x3(1:N1); zeros(K * N, 1); DFT_x3((N1 +
        1):N)];
end

% Inverse FFT to get the interpolated signal
x3_interp = (K + 1) * ifft(DFT_x3_interp);

% Extract the relevant part of the interpolated signal
x3_interp_extracted = x3_interp(1:((K + 1) * (N - 1)) + 1);

% Compute the norm difference between the interpolated and original
    signal
Norm_Difference_x3 = norm(x3_interp_extracted) - norm(x3);
disp(Norm_Difference_x3);
```

```
% Plot the first 50 samples of both signals
figure;
subplot(2, 1, 1); plot(x3(1:50)); title('Original Signal x (First 50
    Samples)');
subplot(2, 1, 2); plot(x3_interp_extracted(1:50));
    title('Interpolated Signal x3 (First 50 Samples)');

% Plot both signals on the same axes
plot(x3(1:50), 'b', 'LineWidth', 2); % Original signal in blue
hold on;
plot(x3_interp_extracted(1:50), 'r', 'LineWidth', 2); % Interpolated
    signal in red
hold off;

% Add labels and title
xlabel('Sample Index');
ylabel('Signal Value');
title('Original and Interpolated Signals (First 50 Samples)');
legend('Original Signal', 'Interpolated Signal');

% Plot the original and interpolated signals in stem plot
figure(12);
subplot(2, 1, 1);
stem(x3(1:50), 'b', 'Marker', 'o');
title('x3 - Original');
subplot(2, 1, 2)
stem(x3_interp_extracted(1:50), 'r', 'Marker', 'x');
title('x3 - Interpolated with K=2');
```

Listing 2.10 — Interpolate x3 with K

```
Norm_Difference_x3 =  11.0927
```

c) Repeat the first part for signal x_4 with $K = 3$.

```
% Interpolate x4 with K = 3
K = 3;
% Compute the DFT of x4
DFT_x4 = fft(x4);
N = length(x4);

% Check if N is even
if mod(N, 2) == 0
    % N is even
    N2 = N / 2;
    % Apply zero padding and interpolation
```

```
    DFT_x4_interp = [DFT_x4(1:N2); DFT_x4(N2 + 1) / 2; zeros((K * N)
        - 1, 1); DFT_x4(N2 + 1) / 2; DFT_x4((N2 + 2):N)];
else
    % N is odd
    N1 = (N + 1) / 2;
    % Apply zero padding and interpolation
    DFT_x4_interp = [DFT_x4(1:N1); zeros(K * N, 1); DFT_x4((N1 +
        1):N)];
end

% Inverse FFT to get the interpolated signal
x4_interp = (K + 1) * ifft(DFT_x4_interp);
% Extract the relevant part of the interpolated signal
x4_interp_extracted = x4_interp(1:((K + 1) * (N - 1)) + 1);
% Compute the norm difference between the interpolated and original
    signal
Norm_Difference_x4 = norm(x4_interp_extracted) - norm(x4);
disp(Norm_Difference_x4);

% Plot the first 50 samples of both signals
figure;
subplot(2, 1, 1); plot(x4(1:50)); title('Original Signal x (First 50
    Samples)');
subplot(2, 1, 2); plot(x4_interp_extracted(1:50));
    title('Interpolated Signal x4 (First 50 Samples)');
% Plot both signals on the same axes
plot(x4(1:50), 'b', 'LineWidth', 2); % Original signal in blue
hold on;
plot(x4_interp_extracted(1:50), 'r', 'LineWidth', 2); % Interpolated
    signal in red
hold off;

% Add labels and title
xlabel('Sample Index');
ylabel('Signal Value');
title('Original and Interpolated Signals (First 50 Samples)');
legend('Original Signal', 'Interpolated Signal');

% Plot the original and interpolated signals
figure(13);
subplot(2, 1, 1);
stem(x4(1:50), 'b', 'Marker', 'o');
title('x4 - Original');
subplot(2, 1, 2)
stem(x4_interp_extracted(1:50), 'r', 'Marker', 'x');
```

```
51 title('x4 - Interpolated with K=3');
```

Listing 2.11 — Interpolate x4 with K

```
Norm_Difference_x4 =     13.1210
```