Department of Electronic & Telecommunication Engineering

University of Moratuwa

Assignment 01: Detecting Harmonics in Noisy Data and Signal Interpolation using DFT

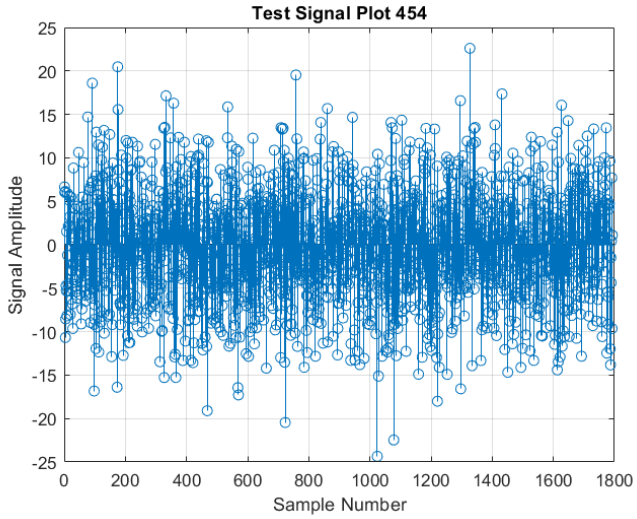
Name: Peiris D.L.C.J.

Index No: 210454G

3.1 Harmonic Detection

1) Loading the given Signal and plot

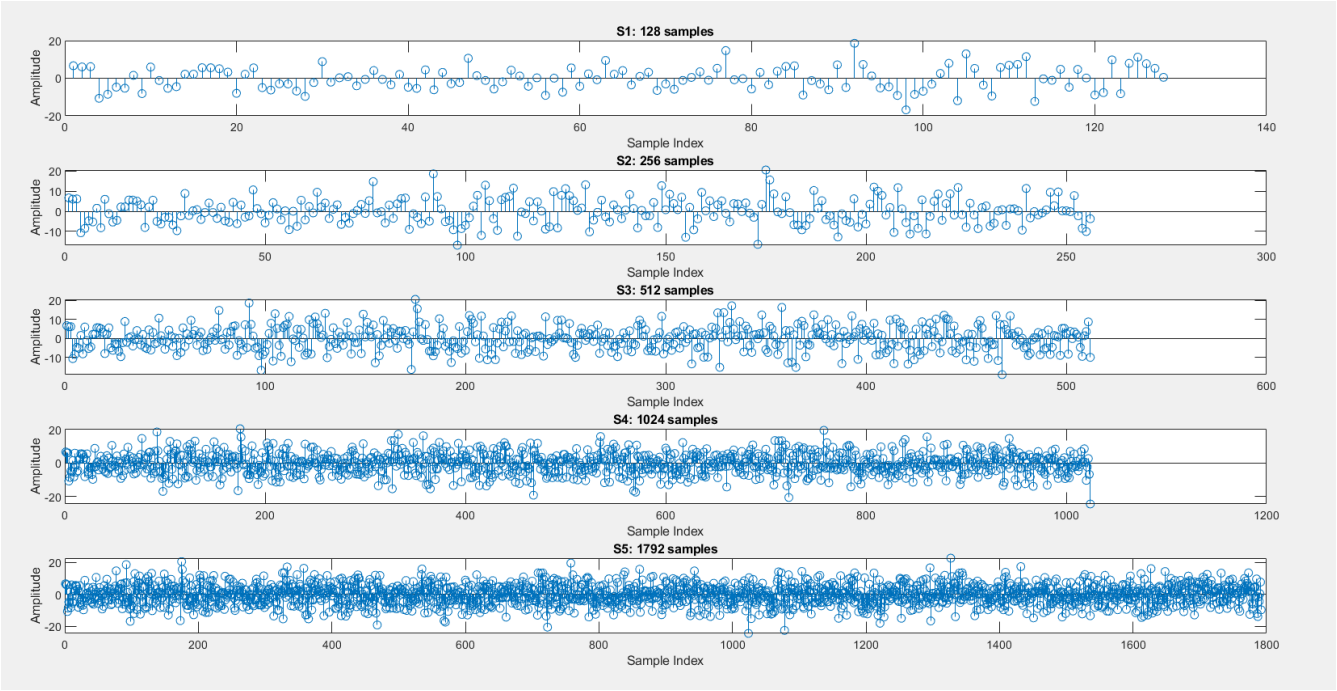
```
%Load the test signal corresponding to index number 210454G
load('signal454.mat', 'xn_test');  % Load the signal from the .mat file

% Plot the discrete signal
figure;
stem(xn_test);
xlabel('Sample Number');
ylabel('Signal Amplitude');
title('Test Signal Plot 454');
grid on;
```



2) Constructing Subsets of the signal and plot

```
fs = 128;           % Sampling rate (Hz)
n_total = 1792;     % Number of samples to consider

% Subsets of the signal
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);

% Create a figure for the subplots
figure;

% Subplot 1: Plot S1 (128 samples)
subplot(5, 1, 1);
stem(S1);
title('S1: 128 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Subplot 2: Plot S2 (256 samples)
subplot(5, 1, 2);
stem(S2);
title('S2: 256 samples');
xlabel('Sample Index');
ylabel('Amplitude');
```

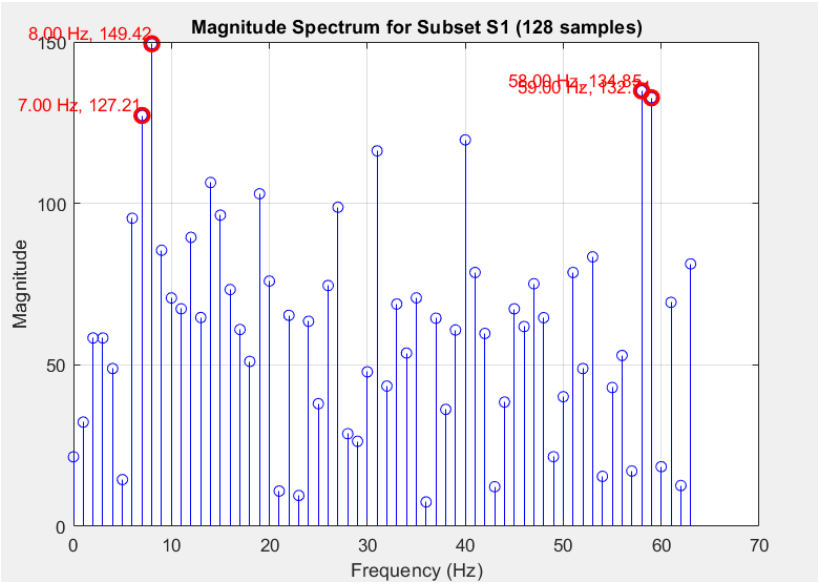
```
% Subplot 3: Plot S3 (512 samples)
subplot(5, 1, 3);
stem(S3);
title('S3: 512 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Subplot 4: Plot S4 (1024 samples)
subplot(5, 1, 4);
stem(S4);
title('S4: 1024 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Subplot 5: Plot S5 (1792 samples)
subplot(5, 1, 5);
stem(S5);
title('S5: 1792 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Add a global title for the entire figure
sgtitle('Plot of Subsets of the Signal');
```



3) Apply DFT to each subset of samples and display the magnitude of the resulting DFT sequences so as to identify the harmonics

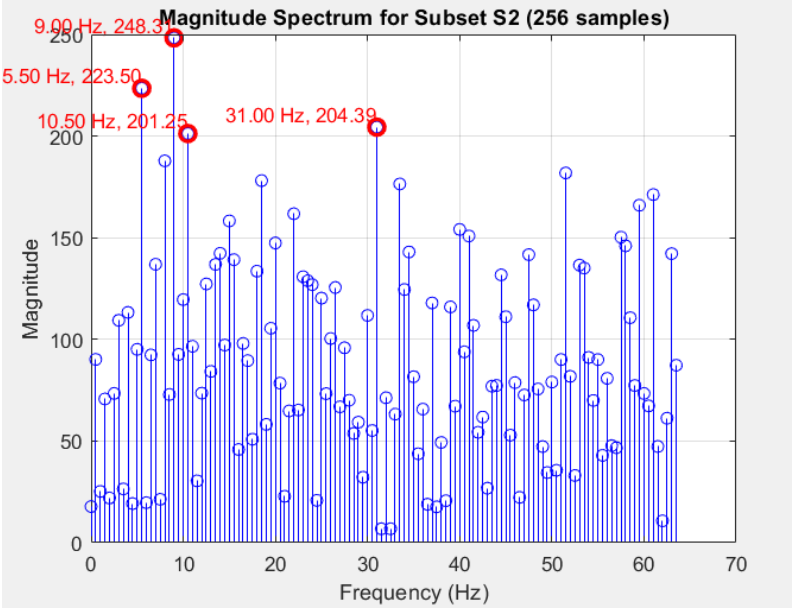
For Subset S1 (1,128)

```
% Create figure for Subset S1
figure;
N1 = length(S1);
Xk1 = fft(S1);               % Compute DFT using FFT
f1 = (0:N1-1)*(fs/N1);       % Frequency vector for plotting
magnitude1 = abs(Xk1);       % Magnitude of DFT

stem(f1(1:N1/2), magnitude1(1:N1/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx1] = sort(magnitude1(1:N1/2), 'descend'); % Sort magnitudes in descending order
top4_idx1 = idx1(1:4); % Indices of top 4 harmonics

plot(f1(top4_idx1), magnitude1(top4_idx1), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f1(top4_idx1(k)), magnitude1(top4_idx1(k)), sprintf('%.2f Hz, %.2f', f1(top4_idx1(k)), magnitude1(top4_idx1(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S1 (128 samples)');
grid on;
hold off;
```



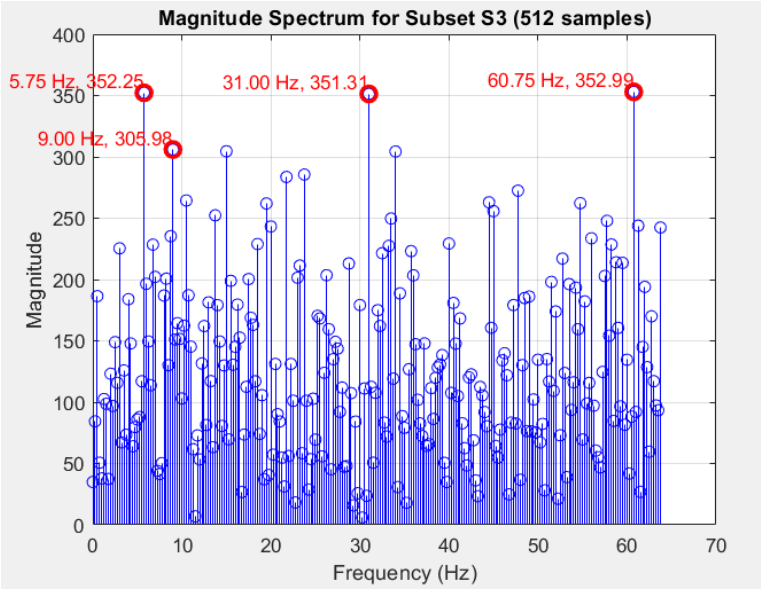
For Subset S2 (1,256)

```
% Create figure for Subset S2
figure;
N2 = length(S2);
Xk2 = fft(S2);               % Compute DFT using FFT
f2 = (0:N2-1)*(fs/N2);       % Frequency vector for plotting
magnitude2 = abs(Xk2);       % Magnitude of DFT

stem(f2(1:N2/2), magnitude2(1:N2/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx2] = sort(magnitude2(1:N2/2), 'descend'); % Sort magnitudes in descending order
top4_idx2 = idx2(1:4); % Indices of top 4 harmonics

plot(f2(top4_idx2), magnitude2(top4_idx2), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f2(top4_idx2(k)), magnitude2(top4_idx2(k)), sprintf('%.2f Hz, %.2f', f2(top4_idx2(k)), magnitude2(top4_idx2(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S2 (256 samples)');
grid on;
hold off;
```



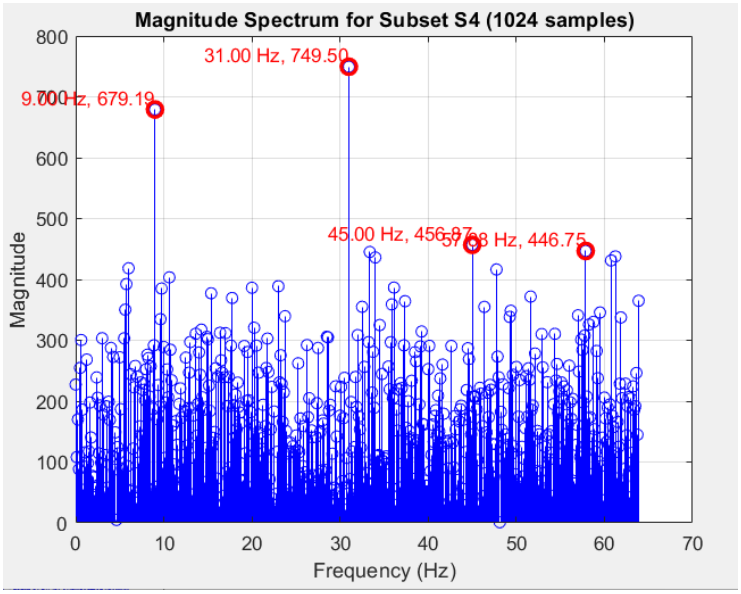
For Subset S3(1,512)

```
% Create figure for Subset S3
figure;
N3 = length(S3);
Xk3 = fft(S3);               % Compute DFT using FFT
f3 = (0:N3-1)*(fs/N3);       % Frequency vector for plotting
magnitude3 = abs(Xk3);       % Magnitude of DFT

stem(f3(1:N3/2), magnitude3(1:N3/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx3] = sort(magnitude3(1:N3/2), 'descend'); % Sort magnitudes in descending order
top4_idx3 = idx3(1:4); % Indices of top 4 harmonics

plot(f3(top4_idx3), magnitude3(top4_idx3), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f3(top4_idx3(k)), magnitude3(top4_idx3(k)), sprintf('%.2f Hz, %.2f', f3(top4_idx3(k)), magnitude3(top4_idx3(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S3 (512 samples)');
grid on;
hold off;
```



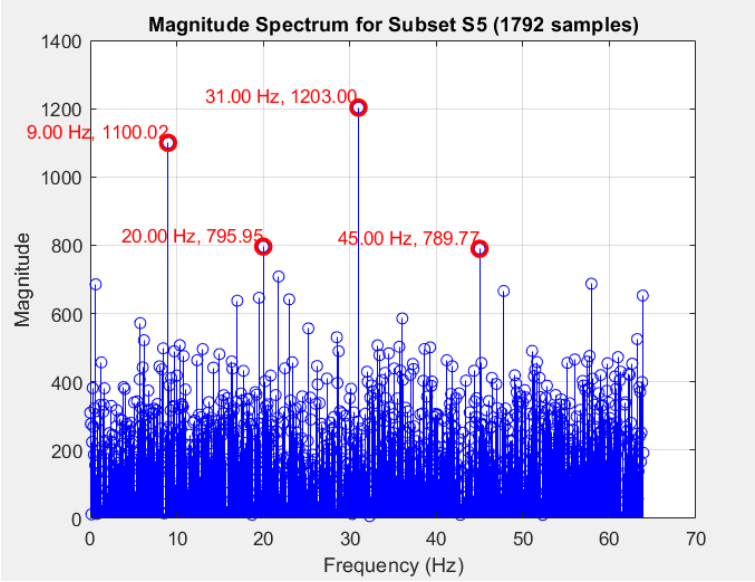
For Subset S4(1,1024)

```
% Create figure for Subset S4
figure;
N4 = length(S4);
Xk4 = fft(S4);               % Compute DFT using FFT
f4 = (0:N4-1)*(fs/N4);       % Frequency vector for plotting
magnitude4 = abs(Xk4);       % Magnitude of DFT

stem(f4(1:N4/2), magnitude4(1:N4/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx4] = sort(magnitude4(1:N4/2), 'descend'); % Sort magnitudes in descending order
top4_idx4 = idx4(1:4); % Indices of top 4 harmonics

plot(f4(top4_idx4), magnitude4(top4_idx4), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f4(top4_idx4(k)), magnitude4(top4_idx4(k)), sprintf('%.2f Hz, %.2f', f4(top4_idx4(k)), magnitude4(top4_idx4(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S4 (1024 samples)');
grid on;
hold off;
```



For Subset S5 (1,1792)

```
% Create figure for Subset S5
figure;
N5 = length(S5);
Xk5 = fft(S5);                % Compute DFT using FFT
f5 = (0:N5-1)*(fs/N5);        % Frequency vector for plotting
magnitude5 = abs(Xk5);        % Magnitude of DFT

stem(f5(1:N5/2), magnitude5(1:N5/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx5] = sort(magnitude5(1:N5/2), 'descend'); % Sort magnitudes in descending order
top4_idx5 = idx5(1:4); % Indices of top 4 harmonics

plot(f5(top4_idx5), magnitude5(top4_idx5), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f5(top4_idx5(k)), magnitude5(top4_idx5(k)), sprintf('%.2f Hz, %.2f', f5(top4_idx5(k)), magnitude5(top4_idx5(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S5 (1792 samples)');
grid on;
hold off;
```



Observation :

As the number of samples increases (from S1 to S5), the resolution of the DFT improves, making it easier to distinguish between closely spaced harmonics. With the addition of more samples, the frequency resolution becomes finer.

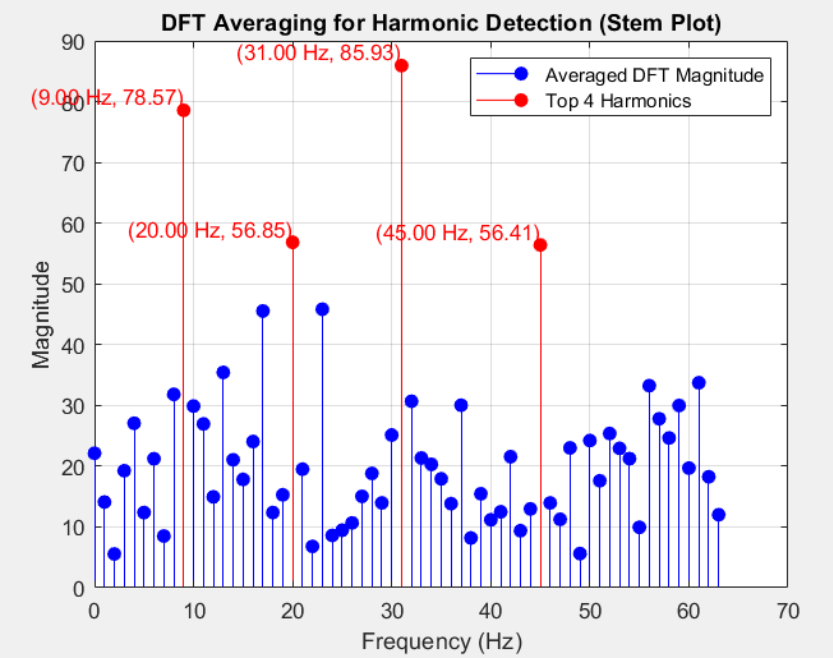
1. **Subset S1 (128 samples):** The frequency resolution is relatively low. Harmonics may not be distinctly visible, and the noise might overshadow some harmonics.
2. **Subset S2 (256 samples):** Slightly better resolution. You might start to see clearer peaks at harmonic frequencies, though some noise is still present.
3. **Subset S3 (512 samples):** Better frequency resolution, and the harmonics should be much more distinguishable. Peaks at distinct harmonic frequencies are more apparent.
4. **Subset S4 (1024 samples):** Even better resolution, with clear harmonic peaks. The impact of noise is more subdued.
5. **Subset S5 (1792 samples):** The best resolution among the subsets, allowing you to clearly identify all harmonics up to 64 Hz. The noise is less dominant compared to the harmonics.

So as move from S1 to S5, the frequency resolution improves, making the harmonic peaks sharper and easier to identify. The higher the number of samples, the more accurate the identification of the harmonics becomes, with noise being less impactful in larger sample sets.

4) Apply DFT averaging method

```
fs = 128;  % Sampling frequency in Hz
K = 128;   % Length of each subset
L = 14;    % Number of subsets

% Ensure the signal length matches the total number of samples required (L * K)
N = L * K;  % Total length of signal for DFT averaging (1792 in this case)
xn_test = xn_test(1:N);  % Truncate or use the first N samples of the signal

% Initialize variables to store the DFT results
X_avg = zeros(1, K);  % Initialize for averaging DFTs

% Loop over each subset and calculate the DFT, then accumulate
for l = 1:L
    subset = xn_test((l-1)*K+1:l*K);  % Get the current subset
    X_subset = fft(subset);           % Compute DFT of the subset
    X_avg = X_avg + X_subset;         % Accumulate the DFTs
end

% Calculate the averaged DFT
X_avg = X_avg / L;

% Frequency vector for plotting
f = (0:K-1)*(fs/K);

% Calculate the magnitude of the averaged DFT
X_avg_mag = abs(X_avg);

% Plot the magnitude of the averaged DFT using a stem plot
figure;
stem(f(1:K/2), X_avg_mag(1:K/2), 'b', 'filled');  % Plot up to Nyquist frequency (fs/2)
hold on;
```

```
% Identify the top 4 harmonics (frequencies with the highest magnitude)
X_half = X_avg_mag(1:K/2);  % Only consider up to the Nyquist frequency
[~, idx] = sort(X_half, 'descend');  % Sort in descending order of magnitude

% Get the indices of the top 4 peaks
top4_indices = idx(1:4);

% Convert these indices to the corresponding frequencies
top4_frequencies = f(top4_indices);
top4_magnitudes = X_half(top4_indices);

% Plot the top 4 harmonics in red and add text annotations
stem(f(top4_indices), X_avg_mag(top4_indices), 'r', 'filled');  % Red for top harmonics
for i = 1:4
    text(f(top4_indices(i)), X_avg_mag(top4_indices(i)), ...
         sprintf('(%.2f Hz, %.2f)', f(top4_indices(i)), X_avg_mag(top4_indices(i))), ...
         'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('DFT Averaging for Harmonic Detection (Stem Plot)');
legend('Averaged DFT Magnitude', 'Top 4 Harmonics');
grid on;
hold off;
```




```
% Display the top 4 harmonic frequencies and their magnitudes
disp('Top 4 harmonic frequencies and their magnitudes (Hz and magnitude):');
```

```
Top 4 harmonic frequencies and their magnitudes (Hz and magnitude):
```

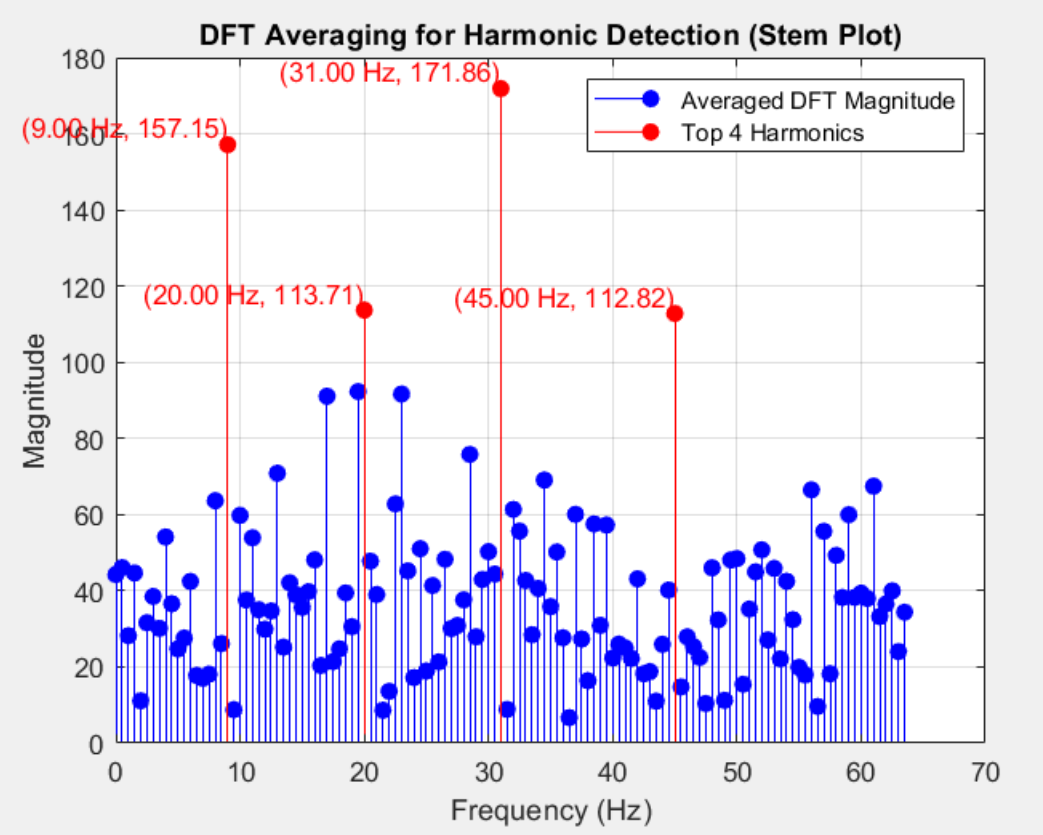
```
for i = 1:4
    fprintf('Frequency: %.2f Hz, Magnitude: %.2f\n', top4_frequencies(i), top4_magnitudes(i));
end
```

```
Frequency: 31.00 Hz, Magnitude: 85.93
Frequency: 9.00 Hz, Magnitude: 78.57
Frequency: 20.00 Hz, Magnitude: 56.85
Frequency: 45.00 Hz, Magnitude: 56.41
```

5) Smallest value of L such that the four peaks that correspond to the four harmonics not greater than 64 remain clearly visible

```
fs = 128;  % Sampling frequency in Hz
K = 256;   % Length of each subset
L = 7;     % Number of subsets
```

Harmonics are clearly visible until L = 7 . So the smallest value of L is 7.



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

No

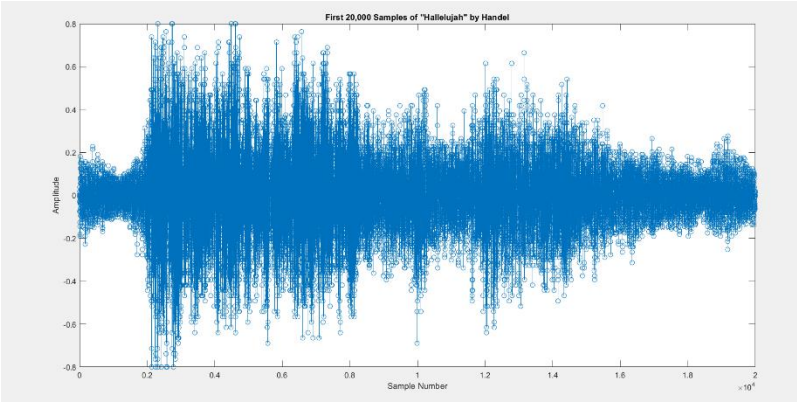
Choosing values like K=100 or K= 135 is not suitable, because they don't divide the total number of samples , N=1792 evenly, This leads to incomplete subsets. This mismatch can result in errors or incomplete data representation, making frequency analysis less reliable. Also , K must be at least 128 to satisfy the Nyquist criterion, which ensures that the DFT can capture all relevant frequency components up to the highest (Nyquist) frequency, thereby preventing aliasing. Also using powers of 2, such as K=128 or K=256 not only ensures proper division of the signal but also significantly boosts computational efficiency. The Fast Fourier Transform (FFT) algorithm is optimized for powers of 2, meaning that selecting values like 128 or 256 allows the algorithm to run faster and more efficiently, especially when handling large datasets. Hence, choosing such values ensures both accuracy in harmonic analysis and optimal performance of the FFT computation.

3.2. Interpolation

1) Load the first 20,000 samples of a music clip" Hallelujah"

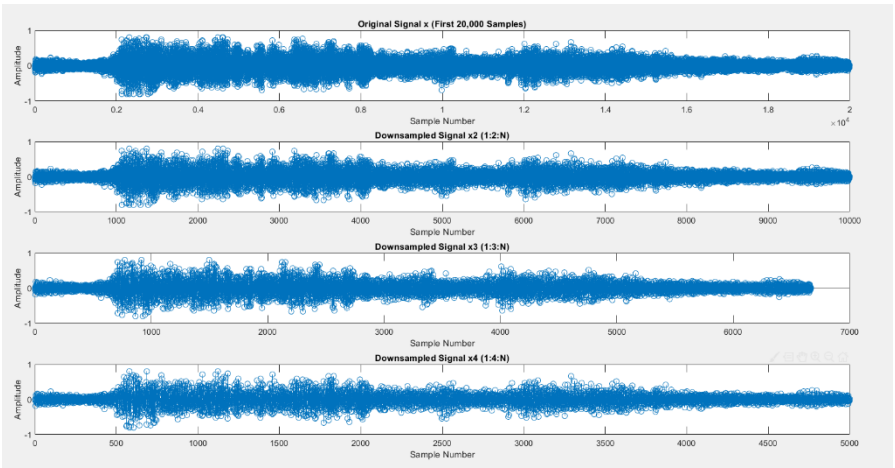
Here the given signal is loaded as following

```
% Load the Handel music clip
load handel

% Extract the first 20,000 samples
y_20000 = y(1:20000);

% Plot the first 20,000 samples
figure;
stem(y_20000);
title('First 20,000 Samples of "Hallelujah" by Handel');
xlabel('Sample Number');
ylabel('Amplitude');
```



2) Generate given signals from the first 20,000 samples

```
% Load the Handel music clip
load handel

% Define the number of samples
N = 20000;

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

% Generate signals with different sampling rates
x2 = x(1:2:N); % Downsample by 2
x3 = x(1:3:N); % Downsample by 3
x4 = x(1:4:N); % Downsample by 4

% Plot the original and downsampled signals
figure;
subplot(4, 1, 1);
stem(x);
title('Original Signal x (First 20,000 Samples)');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(4, 1, 2);
stem(x2);
title('Downsampled Signal x2 (1:2:N)');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(4, 1, 3);
stem(x3);
title('Downsampled Signal x3 (1:3:N)');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(4, 1, 4);
stem(x4);
title('Downsampled Signal x4 (1:4:N)');
xlabel('Sample Number');
ylabel('Amplitude');
```



3. DFT-based method to interpolate the signals x_2, x_3 and x_4

(a) Interpolate the signal x_2 with K = 1.

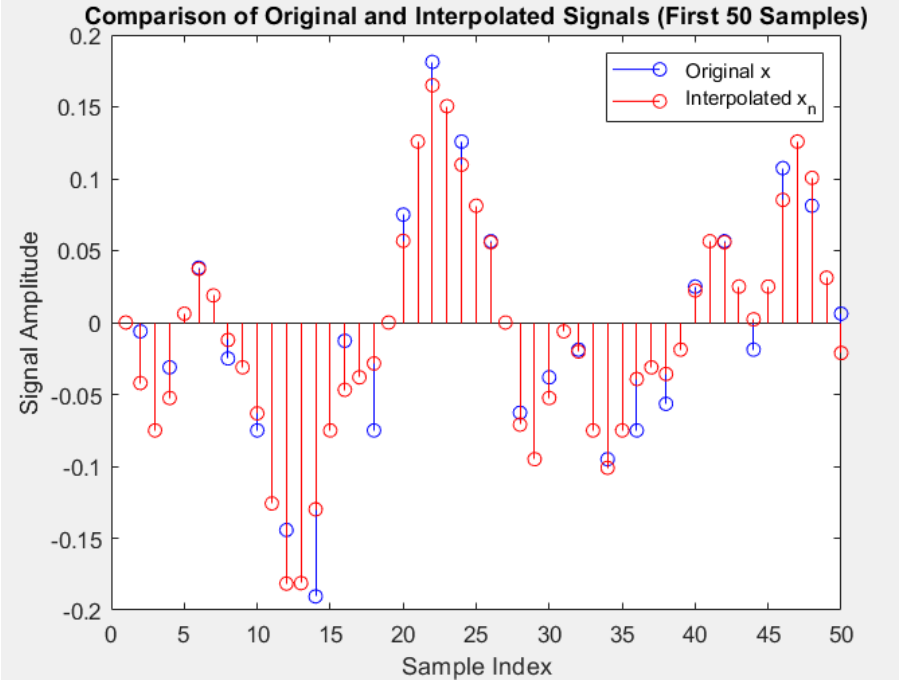
Here To interpolate the signal x_2 using the DFT-based method with K=1, First the signal is transformed into the frequency domain using DFT using the FFT algorithm. The DFT spectrum is split into positive and negative frequency components. Then zero-padding is applied in the frequency domain to increase the resolution of the signal. Inverse FFT is then performed to return the signal to the time domain, where the resulting interpolated signal is adjusted in length to match the original. Here the 2-norm difference is calculated to measure the deviation between the interpolated signal and the original. Finally, both signals are plotted together to visually compare the first 50 samples.

```
% Interpolation process for x2 with K = 1 (frequency-domain zero-padding)
K = 1;            % Upsampling factor (K = 1 means interpolating back to original rate)
% Compute FFT of x2
X2 = fft(x2);
N2 = length(X2);  % Length of the DFT
half_N2 = N2 / 2; % Half the length of the DFT

% Split the DFT into two parts (positive and negative frequencies)
first_half = X2(1:half_N2);        % Positive frequency components
second_half = X2(half_N2+2:end);   % Negative frequency components

% Handle midpoint for even-length signals
MidTerm = X2(half_N2+1) / 2;  % Midpoint term since signal length is even

% Insert zeros for interpolation
zero_list = zeros(K*N2 - 1, 1);    % Zero padding

% Reconstruct spectrum with zero padding
X_Z = [first_half; MidTerm; zero_list; MidTerm; second_half];

% Compute inverse FFT and scale
x_interpolated = real(ifft(X_Z)) * (K+1);

% Adjust the length of the interpolated signal to match original
x_n = x_interpolated(1:(K+1)*(N2-1)+1);  % Select necessary samples

% Calculate the 2-norm difference between interpolated and original signals
len_diff = length(x) - length(x_n);

% Zero-pad the shorter signal if necessary
if len_diff > 0
    x_n = [x_n; zeros(len_diff, 1)];  % Pad interpolated signal
elseif len_diff < 0
    x_n = x_n(1:length(x));           % Truncate interpolated signal
end

% Compute the 2-norm difference
norm_diff = norm(x - x_n);  % Euclidean distance (2-norm)
disp(['2-Norm Difference: ', num2str(norm_diff)]);
```

```
% Plot the first 50 samples of both original and interpolated signals as stem plots
figure;
stem(1:50, x(1:50), 'b', 'DisplayName', 'Original x');  % Plot original signal
hold on;
stem(1:50, x_n(1:50), 'r', 'DisplayName', 'Interpolated x_n');  % Plot interpolated signal
hold off;
legend('show');
title('Comparison of Original and Interpolated Signals (First 50 Samples)');
xlabel('Sample Index');
ylabel('Signal Amplitude');
```

Results:

2-norm difference: 6.144



Observation

Here the interpolated signal (red) follows the original signal (blue) closely, with only small differences between them. This gives a good approximation of the original signal. The small 2-norm difference shows that this method works well when there are not too many missing samples.

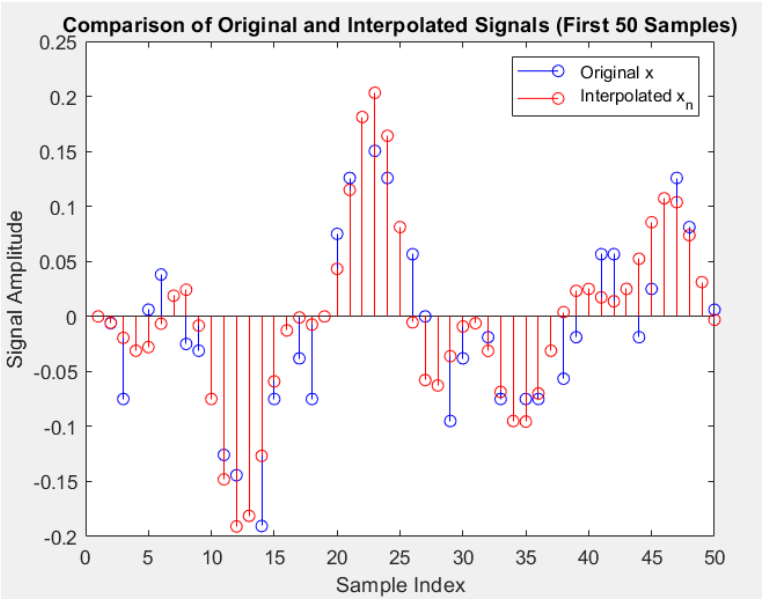
(b) Interpolate the signal x_3 with K = 2.

```
% Interpolation process for x3 with K = 2 (frequency-domain zero-padding)
K = 2;            % Upsampling factor (K = 2 means interpolating back to original rate)

% Compute FFT of x3
X3 = fft(x3);
N3 = length(X3);  % Length of the DFT
half_N3 = floor(N3 / 2);  % Make sure it's an integer (half the length of the DFT)

% Split the DFT into two parts (positive and negative frequencies)
first_half = X3(1:half_N3);        % Positive frequency components
second_half = X3(half_N3+2:end);   % Negative frequency components (ensure valid indices)

% Handle midpoint for even-length signals
MidTerm = X3(half_N3+1) / 2;  % Midpoint term since signal length is even

% Insert zeros for interpolation
zero_list = zeros(K*N3 - 1, 1);    % Zero padding

% Reconstruct spectrum with zero padding
X_Z = [first_half; MidTerm; zero_list; MidTerm; second_half];

% Compute inverse FFT and scale
x_interpolated = real(ifft(X_Z)) * (K+1);

% Adjust the length of the interpolated signal to match original
x_n = x_interpolated(1:(K+1)*(N3-1)+1);  % Select necessary samples

% Calculate the 2-norm difference between interpolated and original signals
len_diff = length(x) - length(x_n);

% Zero-pad the shorter signal if necessary
if len_diff > 0
    x_n = [x_n; zeros(len_diff, 1)];  % Pad interpolated signal
elseif len_diff < 0
    x_n = x_n(1:length(x));           % Truncate interpolated signal
end

% Compute the 2-norm difference
norm_diff = norm(x - x_n);  % Euclidean distance (2-norm)
disp(['2-Norm Difference for x3: ', num2str(norm_diff)]);
```

```
% Plot the first 50 samples of both original and interpolated signals as stem plots
figure;
stem(1:50, x(1:50), 'b', 'DisplayName', 'Original x');  % Plot original signal
hold on;
stem(1:50, x_n(1:50), 'r', 'DisplayName', 'Interpolated x_n');  % Plot interpolated signal
hold off;
legend('show');
title('Comparison of Original and Interpolated Signals (First 50 Samples)');
xlabel('Sample Index');
ylabel('Signal Amplitude');
```

Results:

2-norm difference: 8.3653



Observation

The red plot (interpolated x_3) still follows the original signal's shape, but the differences are more noticeable compared to x_2. Interpolated signal doesn't match the original as closely. Here we can observe that with fewer samples, the interpolation becomes less accurate. So here interpolation starts to lose detail as fewer points are used to reconstruct the signal, but it still captures the overall shape.

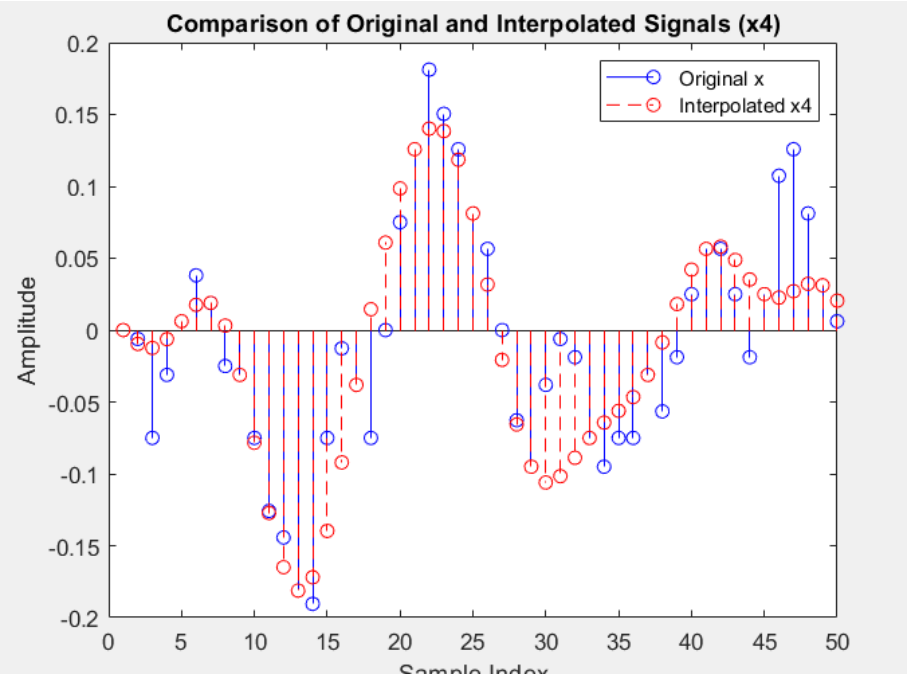
(b) Interpolate the signal x_4 with K = 3

```
% Interpolation process for x4 with K = 3 (frequency-domain zero-padding)
K = 3;            % Upsampling factor (K = 3 means interpolating back to original rate)

% Compute FFT of x4
X4 = fft(x4);
N4 = length(X4);  % Length of the DFT
half_N4 = floor(N4 / 2);  % Make sure it's an integer (half the length of the DFT)

% Split the DFT into two parts (positive and negative frequencies)
first_half = X4(1:half_N4);        % Positive frequency components
second_half = X4(half_N4+2:end);   % Negative frequency components (ensure valid indices)

% Handle midpoint for even-length signals
MidTerm = X4(half_N4+1) / 2;  % Midpoint term since signal length is even

% Insert zeros for interpolation
zero_list = zeros(K*N4 - 1, 1);    % Zero padding

% Reconstruct spectrum with zero padding
X_Z = [first_half; MidTerm; zero_list; MidTerm; second_half];

% Compute inverse FFT and scale
x_interpolated = real(ifft(X_Z)) * (K+1);

% Adjust the length of the interpolated signal to match original
x_n = x_interpolated(1:(K+1)*(N4-1)+1);  % Select necessary samples

% Calculate the 2-norm difference between interpolated and original signals
len_diff = length(x) - length(x_n);

% Zero-pad the shorter signal if necessary
if len_diff > 0
    x_n = [x_n; zeros(len_diff, 1)];  % Pad interpolated signal
elseif len_diff < 0
    x_n = x_n(1:length(x));           % Truncate interpolated signal
end

% Compute the 2-norm difference
norm_diff = norm(x - x_n);  % Euclidean distance (2-norm)
disp(['2-Norm Difference for x4: ', num2str(norm_diff)]);
```

```
% Plot the first 50 samples of both original and interpolated signals
figure;
stem(1:50, x(1:50), 'b', 'DisplayName', 'Original x');  % Plot original signal
hold on;
plot(1:50, x_n(1:50), 'r--', 'DisplayName', 'Interpolated x4');  % Plot interpolated signal
legend;
title('Comparison of Original and Interpolated Signals (x4)');
xlabel('Sample Index');
ylabel('Amplitude');
hold off;
```

Results:

2-norm difference: 23.4998



Observation

The red plot (interpolated x4) significantly drifts away from the original signal. We can see the difference between these two very clearly. Also large 2-norm difference shows that interpolating from every fourth sample leads to much more error. So the interpolated signal no longer captures the fine details of the original, and significant portions of the signal are misaligned. This shows the limits of interpolation when reconstructing a signal from very few samples.

d) Overall Observation

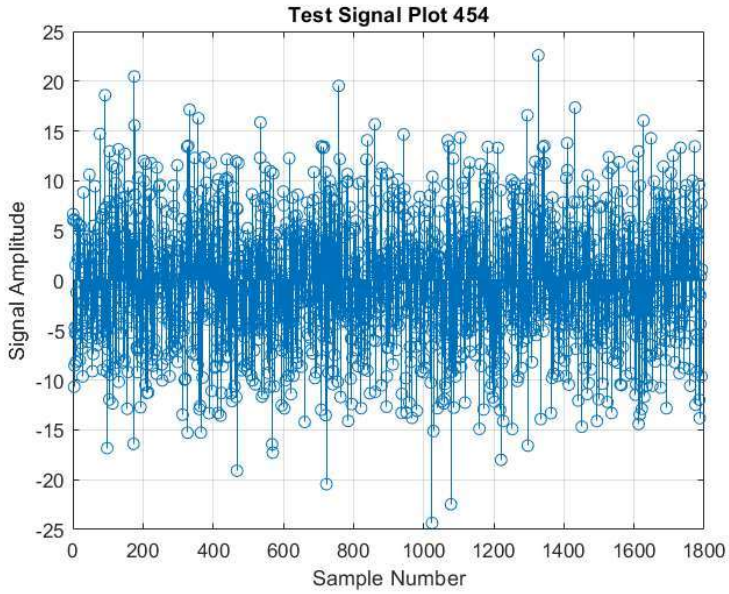
From x_2 to x_4, we can see a clear increase in error and visual differences between the interpolated and original signals. Signals with high frequency parts are harder to interpolate when the sample rate is reduced. This explains why the red plot starts to drift more from the blue plot in x_3 and x_4. x_2 (K = 1) gives the best result, with a small error and very close alignment to the original. This shows that interpolating with a moderate number of samples works well. The accuracy of the interpolation decreases as the number of samples decreases. So, there's a trade-off between how many samples are used and how accurate the interpolation is. More samples lead to better reconstruction, while fewer samples cause the interpolated signal to lose important details.

Appendix

3.1 Harmonic Detection

Part 1 : Loading the given Signal and plot

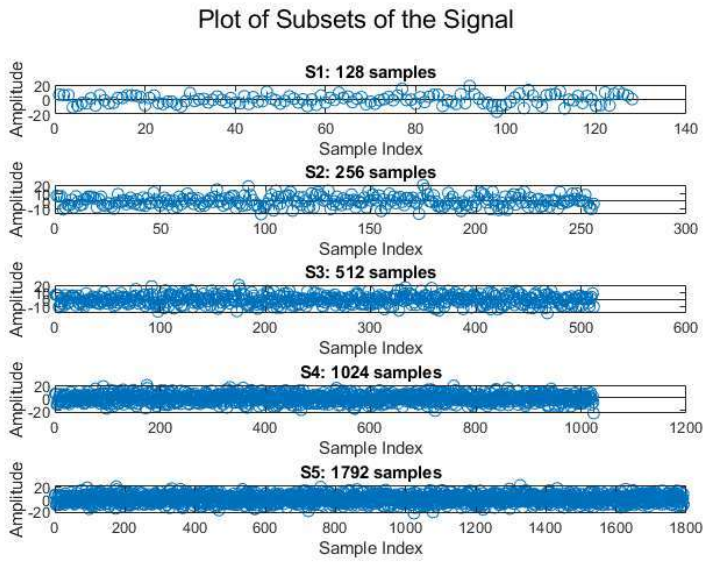
```
%Load the test signal corresponding to index number 210454G
load('signal454.mat', 'xn_test');  % Load the signal from the .mat file

% Plot the discrete signal
figure;
stem(xn_test);
xlabel('Sample Number');
ylabel('Signal Amplitude');
title('Test Signal Plot 454');
grid on;
```



Part 2 : Constructing Subsets of the signal and plot

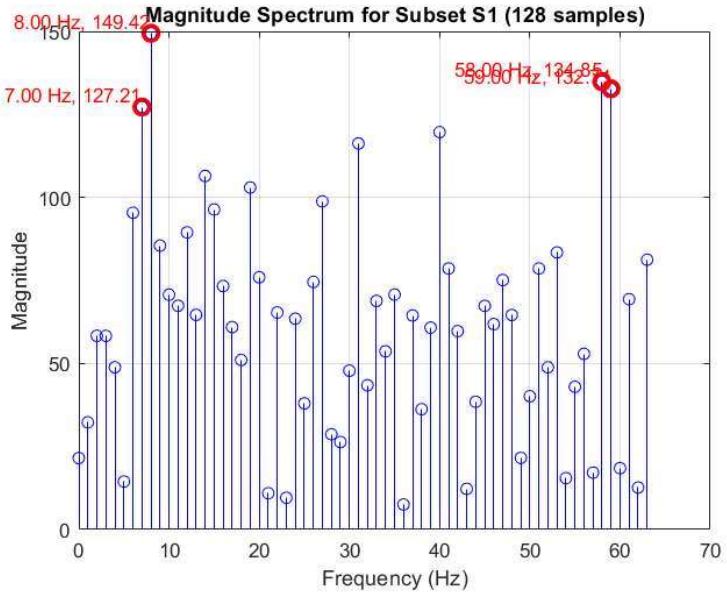
```
fs = 128;           % Sampling rate (Hz)
n_total = 1792;     % Number of samples to consider

% Subsets of the signal
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);

% Create a figure for the subplots
figure;

% Subplot 1: Plot S1 (128 samples)
subplot(5, 1, 1);
stem(S1);
title('S1: 128 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Subplot 2: Plot S2 (256 samples)
subplot(5, 1, 2);
stem(S2);
title('S2: 256 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Subplot 3: Plot S3 (512 samples)
subplot(5, 1, 3);
stem(S3);
title('S3: 512 samples');
xlabel('Sample Index');
```

```
ylabel('Amplitude');

% Subplot 4: Plot S4 (1024 samples)
subplot(5, 1, 4);
stem(S4);
title('S4: 1024 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Subplot 5: Plot S5 (1792 samples)
subplot(5, 1, 5);
stem(S5);
title('S5: 1792 samples');
xlabel('Sample Index');
ylabel('Amplitude');

% Add a global title for the entire figure
sgtitle('Plot of Subsets of the Signal');
```

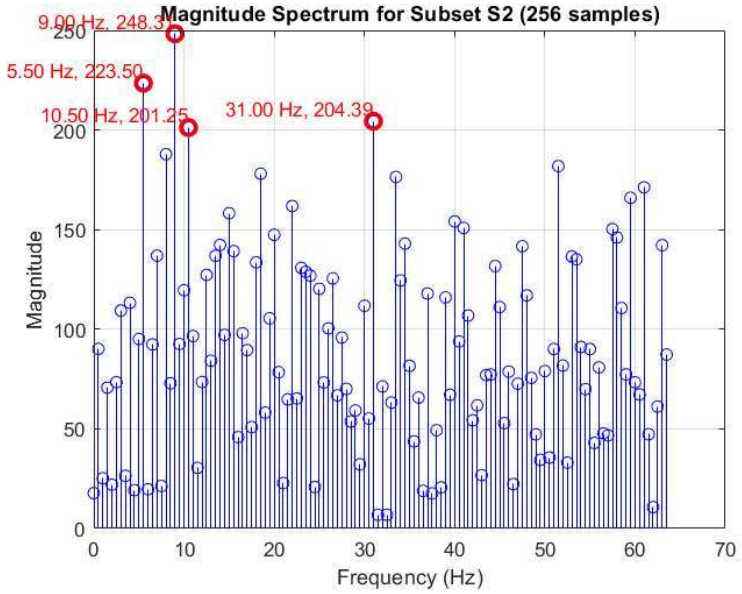


Part 3: Apply DFT to each subset of samples and display the magnitude of the reulting DFT sequences so as to identify the harmonics

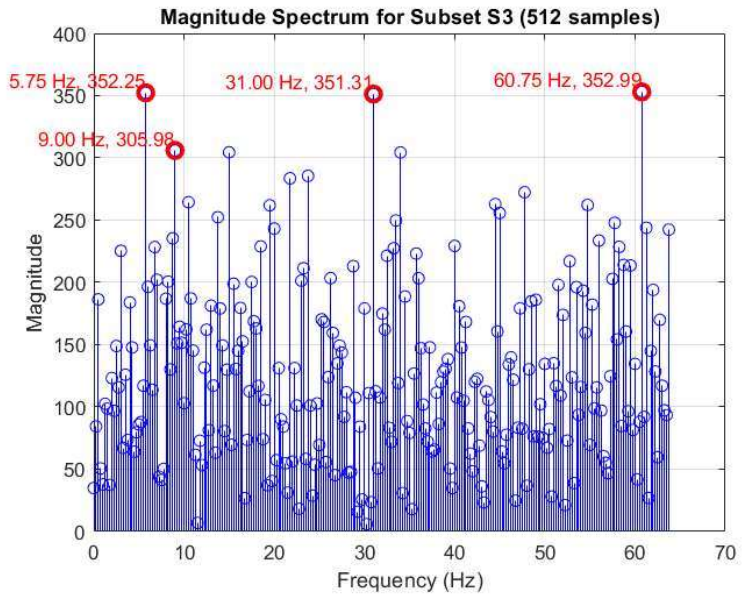
```
% Create figure for Subset S1
figure;
N1 = length(S1);
Xk1 = fft(S1);                % Compute DFT using FFT
f1 = (0:N1-1)*(fs/N1);        % Frequency vector for plotting
magnitude1 = abs(Xk1);        % Magnitude of DFT

stem(f1(1:N1/2), magnitude1(1:N1/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx1] = sort(magnitude1(1:N1/2), 'descend'); % Sort magnitudes in descending order
top4_idx1 = idx1(1:4); % Indices of top 4 harmonics

plot(f1(top4_idx1), magnitude1(top4_idx1), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f1(top4_idx1(k)), magnitude1(top4_idx1(k)), sprintf('%.2f Hz, %.2f', f1(top4_idx1(k)), magnitude1(top4_idx1(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S1 (128 samples)');
grid on;
hold off;
```




```
% Create figure for Subset S2
figure;
N2 = length(S2);
Xk2 = fft(S2);                % Compute DFT using FFT
f2 = (0:N2-1)*(fs/N2);        % Frequency vector for plotting
magnitude2 = abs(Xk2);        % Magnitude of DFT

stem(f2(1:N2/2), magnitude2(1:N2/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx2] = sort(magnitude2(1:N2/2), 'descend'); % Sort magnitudes in descending order
top4_idx2 = idx2(1:4); % Indices of top 4 harmonics

plot(f2(top4_idx2), magnitude2(top4_idx2), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f2(top4_idx2(k)), magnitude2(top4_idx2(k)), sprintf('%.2f Hz, %.2f', f2(top4_idx2(k)), magnitude2(top4_idx2(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S2 (256 samples)');
grid on;
hold off;
```

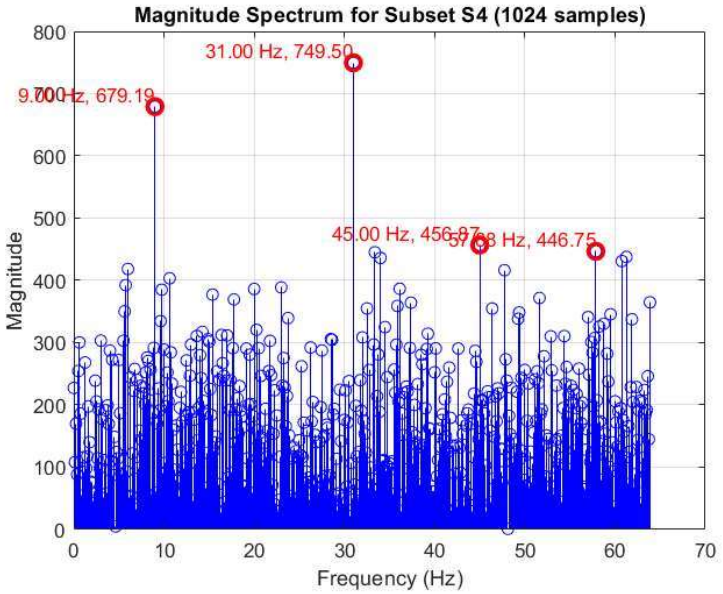


```
% Create figure for Subset S3
figure;
N3 = length(S3);
Xk3 = fft(S3);                % Compute DFT using FFT
f3 = (0:N3-1)*(fs/N3);        % Frequency vector for plotting
magnitude3 = abs(Xk3);        % Magnitude of DFT

stem(f3(1:N3/2), magnitude3(1:N3/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx3] = sort(magnitude3(1:N3/2), 'descend'); % Sort magnitudes in descending order
top4_idx3 = idx3(1:4); % Indices of top 4 harmonics

plot(f3(top4_idx3), magnitude3(top4_idx3), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f3(top4_idx3(k)), magnitude3(top4_idx3(k)), sprintf('%.2f Hz, %.2f', f3(top4_idx3(k)), magnitude3(top4_idx3(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S3 (512 samples)');
grid on;
hold off;
```

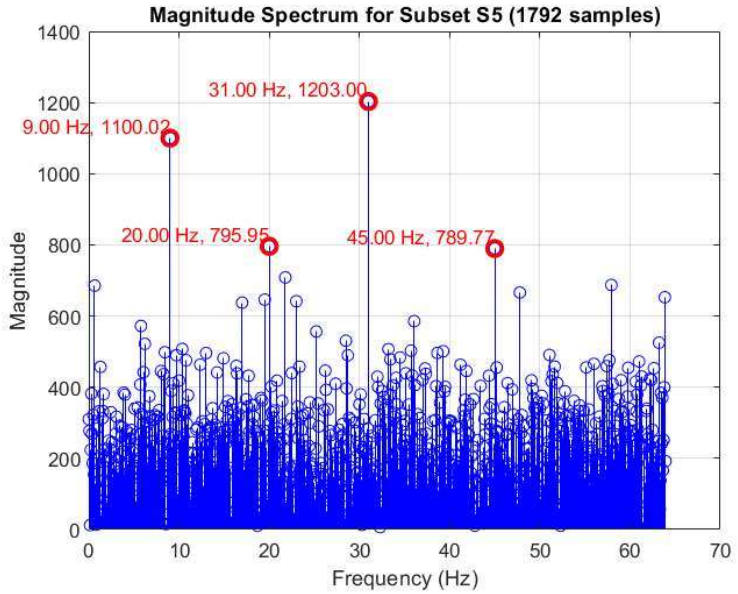



```
% Create figure for Subset S4
figure;
N4 = length(S4);
Xk4 = fft(S4);                % Compute DFT using FFT
f4 = (0:N4-1)*(fs/N4);        % Frequency vector for plotting
magnitude4 = abs(Xk4);        % Magnitude of DFT

stem(f4(1:N4/2), magnitude4(1:N4/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx4] = sort(magnitude4(1:N4/2), 'descend'); % Sort magnitudes in descending order
top4_idx4 = idx4(1:4); % Indices of top 4 harmonics

plot(f4(top4_idx4), magnitude4(top4_idx4), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f4(top4_idx4(k)), magnitude4(top4_idx4(k)), sprintf('%.2f Hz, %.2f', f4(top4_idx4(k)), magnitude4(top4_idx4(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S4 (1024 samples)');
```

```
grid on;
hold off;
```



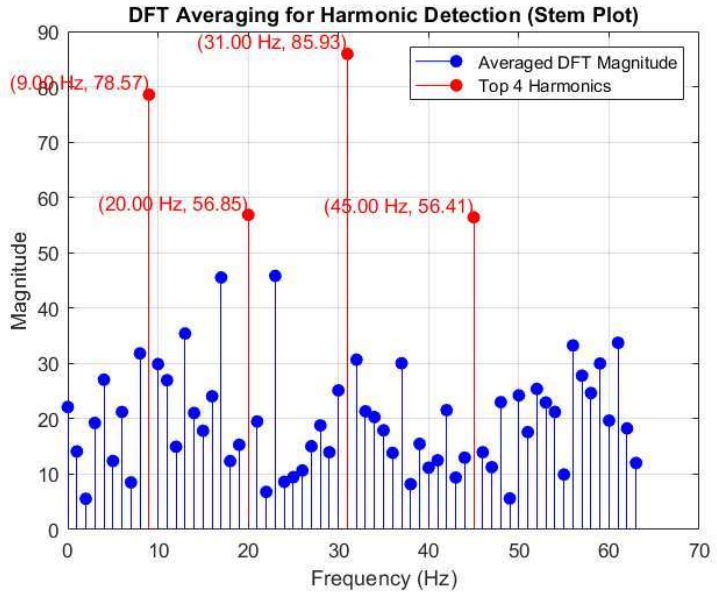

```
% Create figure for Subset S5
figure;
N5 = length(S5);
Xk5 = fft(S5);                % Compute DFT using FFT
f5 = (0:N5-1)*(fs/N5);        % Frequency vector for plotting
magnitude5 = abs(Xk5);        % Magnitude of DFT

stem(f5(1:N5/2), magnitude5(1:N5/2), 'b');  % Plot up to fs/2 (Nyquist frequency)
hold on;

% Find and mark the top 4 harmonics
[~, idx5] = sort(magnitude5(1:N5/2), 'descend'); % Sort magnitudes in descending order
top4_idx5 = idx5(1:4); % Indices of top 4 harmonics

plot(f5(top4_idx5), magnitude5(top4_idx5), 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Mark top 4 harmonics
for k = 1:4
    text(f5(top4_idx5(k)), magnitude5(top4_idx5(k)), sprintf('%.2f Hz, %.2f', f5(top4_idx5(k)), magnitude5(top4_idx5(k))), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum for Subset S5 (1792 samples)');
grid on;
hold off;
```



Part 4: DFT averaing method

```
fs = 128;  % Sampling frequency in Hz
K = 128;   % Length of each subset
L = 14;    % Number of subsets

% Ensure the signal length matches the total number of samples required (L * K)
N = L * K;  % Total length of signal for DFT averaging (1792 in this case)
xn_test = xn_test(1:N);  % Truncate or use the first N samples of the signal

% Initialize variables to store the DFT results
X_avg = zeros(1, K);  % Initialize for averaging DFTs

% Loop over each subset and calculate the DFT, then accumulate
for l = 1:L
    subset = xn_test((l-1)*K+1:l*K);  % Get the current subset
    X_subset = fft(subset);           % Compute DFT of the subset
    X_avg = X_avg + X_subset;         % Accumulate the DFTs
end

% Calculate the averaged DFT
X_avg = X_avg / L;

% Frequency vector for plotting
f = (0:K-1)*(fs/K);

% Calculate the magnitude of the averaged DFT
X_avg_mag = abs(X_avg);

% Plot the magnitude of the averaged DFT using a stem plot
figure;
stem(f(1:K/2), X_avg_mag(1:K/2), 'b', 'filled');  % Plot up to Nyquist frequency (fs/2)
hold on;

% Identify the top 4 harmonics (frequencies with the highest magnitude)
X_half = X_avg_mag(1:K/2);  % Only consider up to the Nyquist frequency
[~, idx] = sort(X_half, 'descend');  % Sort in descending order of magnitude

% Get the indices of the top 4 peaks
top4_indices = idx(1:4);

% Convert these indices to the corresponding frequencies
top4_frequencies = f(top4_indices);
top4_magnitudes = X_half(top4_indices);

% Plot the top 4 harmonics in red and add text annotations
stem(f(top4_indices), X_avg_mag(top4_indices), 'r', 'filled');  % Red for top harmonics
for i = 1:4
    text(f(top4_indices(i)), X_avg_mag(top4_indices(i)), ...
         sprintf('(%.2f Hz, %.2f)', f(top4_indices(i)), X_avg_mag(top4_indices(i))), ...
```

```
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('DFT Averaging for Harmonic Detection (Stem Plot)');
legend('Averaged DFT Magnitude', 'Top 4 Harmonics');
grid on;
hold off;
```




```
% Display the top 4 harmonic frequencies and their magnitudes
disp('Top 4 harmonic frequencies and their magnitudes (Hz and magnitude):');
```

```
Top 4 harmonic frequencies and their magnitudes (Hz and magnitude):
```

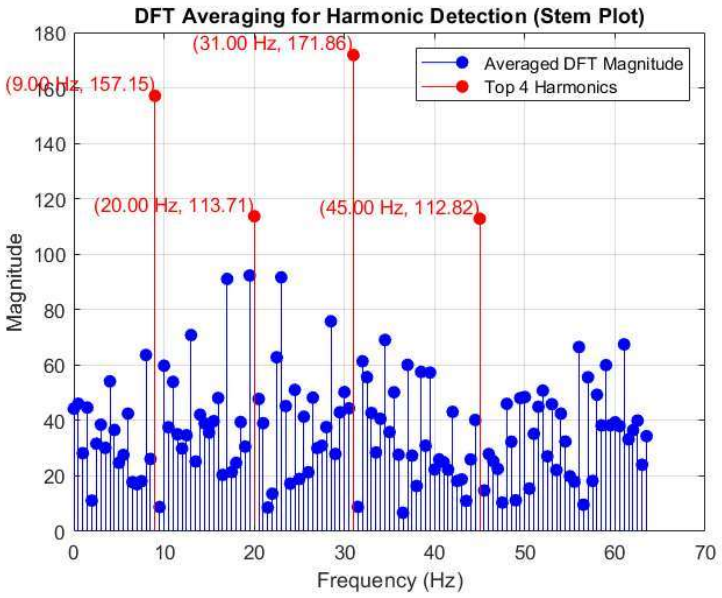
```
for i = 1:4
    fprintf('Frequency: %.2f Hz, Magnitude: %.2f\n', top4_frequencies(i), top4_magnitudes(i));
end
```

```
Frequency: 31.00 Hz, Magnitude: 85.93
Frequency: 9.00 Hz, Magnitude: 78.57
Frequency: 20.00 Hz, Magnitude: 56.85
Frequency: 45.00 Hz, Magnitude: 56.41
```

Part 5 : smallest value of L such that the four peaks that correspond to the four harmonics not greater than 64 remain clearly visible

```
fs = 128;  % Sampling frequency in Hz
K = 256;   % Length of each subset
L = 7;     % Number of subsets

% Ensure the signal length matches the total number of samples required (L * K)
N = L * K;  % Total length of signal for DFT averaging (1792 in this case)
xn_test = xn_test(1:N);  % Truncate or use the first N samples of the signal

% Initialize variables to store the DFT results
X_avg = zeros(1, K);  % Initialize for averaging DFTs

% Loop over each subset and calculate the DFT, then accumulate
for l = 1:L
    subset = xn_test((l-1)*K+1:l*K);  % Get the current subset
    X_subset = fft(subset);           % Compute DFT of the subset
    X_avg = X_avg + X_subset;         % Accumulate the DFTs
end

% Calculate the averaged DFT
X_avg = X_avg / L;

% Frequency vector for plotting
f = (0:K-1)*(fs/K);
```

```
% Calculate the magnitude of the averaged DFT
X_avg_mag = abs(X_avg);

% Plot the magnitude of the averaged DFT using a stem plot
figure;
stem(f(1:K/2), X_avg_mag(1:K/2), 'b', 'filled');  % Plot up to Nyquist frequency (fs/2)
hold on;

% Identify the top 4 harmonics (frequencies with the highest magnitude)
X_half = X_avg_mag(1:K/2);  % Only consider up to the Nyquist frequency
[~, idx] = sort(X_half, 'descend');  % Sort in descending order of magnitude

% Get the indices of the top 4 peaks
top4_indices = idx(1:4);

% Convert these indices to the corresponding frequencies
top4_frequencies = f(top4_indices);
top4_magnitudes = X_half(top4_indices);

% Plot the top 4 harmonics in red and add text annotations
stem(f(top4_indices), X_avg_mag(top4_indices), 'r', 'filled');  % Red for top harmonics
for i = 1:4
    text(f(top4_indices(i)), X_avg_mag(top4_indices(i)), ...
         sprintf('(%.2f Hz, %.2f)', f(top4_indices(i)), X_avg_mag(top4_indices(i))), ...
         'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', 'red');
end

xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('DFT Averaging for Harmonic Detection (Stem Plot)');
legend('Averaged DFT Magnitude', 'Top 4 Harmonics');
grid on;
hold off;
```




```
% Display the top 4 harmonic frequencies and their magnitudes
disp('Top 4 harmonic frequencies and their magnitudes (Hz and magnitude):');
```

```
Top 4 harmonic frequencies and their magnitudes (Hz and magnitude):
```

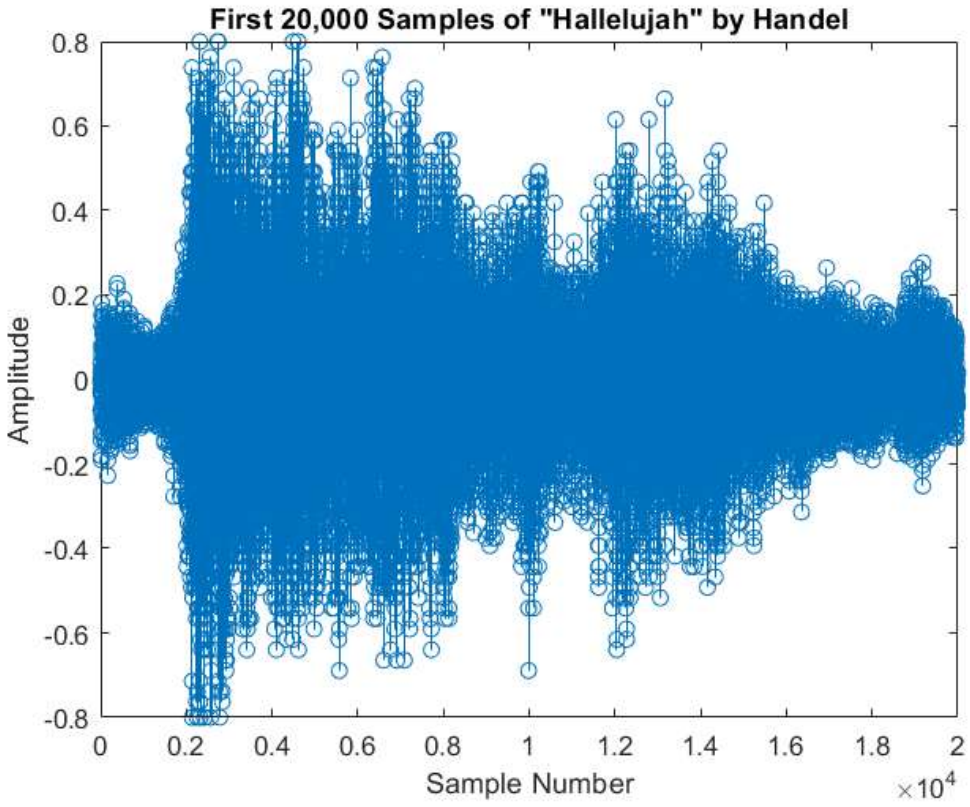
```
for i = 1:4
    fprintf('Frequency: %.2f Hz, Magnitude: %.2f\n', top4_frequencies(i), top4_magnitudes(i));
end
```

```
Frequency: 31.00 Hz, Magnitude: 171.86
Frequency: 9.00 Hz, Magnitude: 157.15
Frequency: 20.00 Hz, Magnitude: 113.71
Frequency: 45.00 Hz, Magnitude: 112.82
```

3.2 Interpolation

Part 1 : Load the first 20,000 samples of a music clip ”Hallelujah”

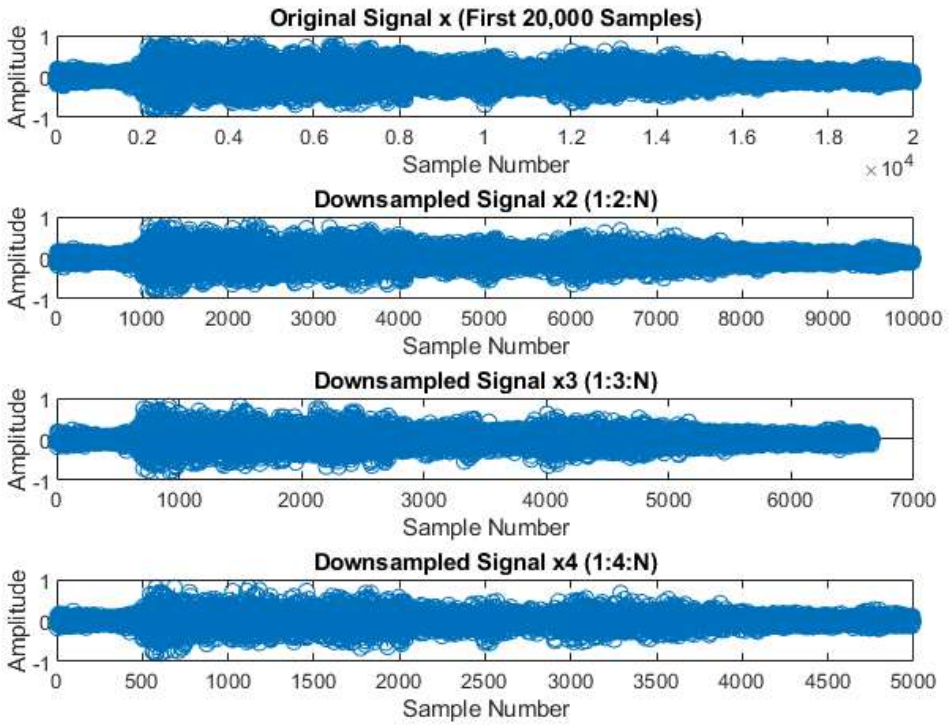
```
% Load the Handel music clip
load handel

% Extract the first 20,000 samples
y_20000 = y(1:20000);

% Plot the first 20,000 samples
figure;
stem(y_20000);
title('First 20,000 Samples of "Hallelujah" by Handel');
xlabel('Sample Number');
ylabel('Amplitude');
```



Part 2: Generate given signals from the frist 20,000 samples

```
% Load the Handel music clip
load handel

% Define the number of samples
N = 20000;

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

% Generate signals with different sampling rates
```

```
x2 = x(1:2:N); % Downsample by 2
x3 = x(1:3:N); % Downsample by 3
x4 = x(1:4:N); % Downsample by 4

% Plot the original and downsampled signals
figure;
subplot(4, 1, 1);
stem(x);
title('Original Signal x (First 20,000 Samples)');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(4, 1, 2);
stem(x2);
title('Downsampled Signal x2 (1:2:N)');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(4, 1, 3);
stem(x3);
title('Downsampled Signal x3 (1:3:N)');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(4, 1, 4);
stem(x4);
title('Downsampled Signal x4 (1:4:N)');
xlabel('Sample Number');
ylabel('Amplitude');
```



Part 3: Apply the DFT-based method to interpolate the signals x2, x3 and x4

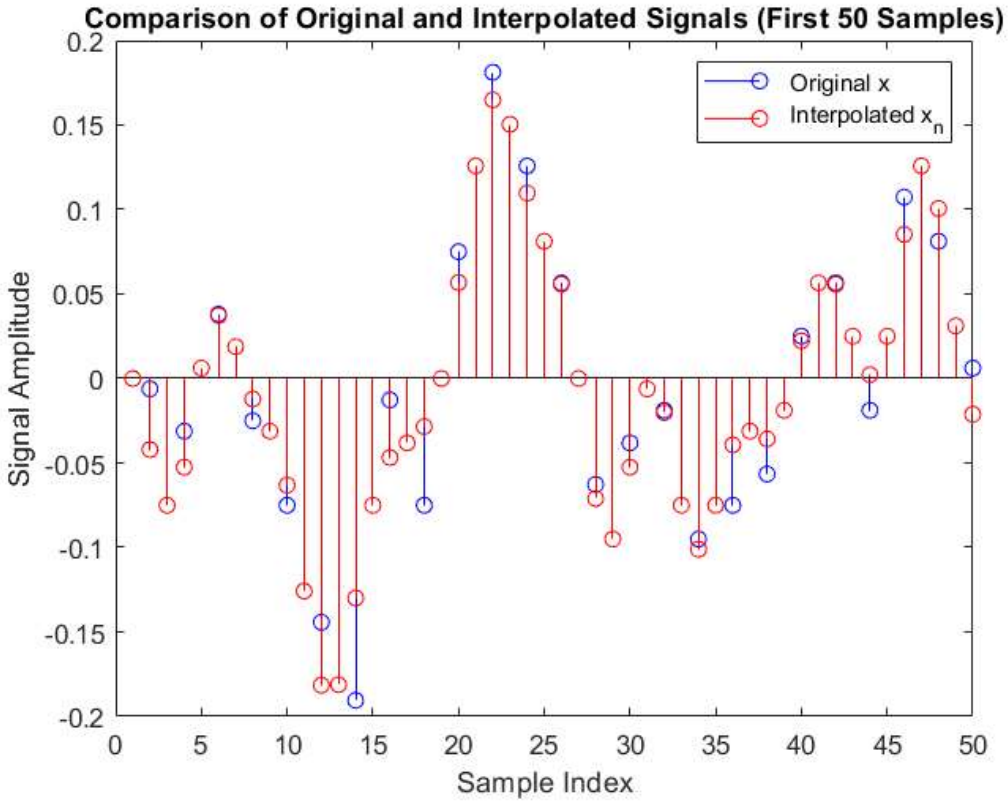
a) Interpolate the signal x2 with K = 1

```
% Parameters
N = 20000;  % Number of samples

% Obtain signals
x = y(1:N);       % Original signal
x2 = x(1:2:N);    % Subsample every 2nd sample (decimated signal)

% Interpolation process for x2 with K = 1 (frequency-domain zero-padding)
K = 1;            % Upsampling factor (K = 1 means interpolating back to original rate)
% Compute FFT of x2
X2 = fft(x2);
N2 = length(X2);  % Length of the DFT
half_N2 = N2 / 2; % Half the length of the DFT

% Split the DFT into two parts (positive and negative frequencies)
first_half = X2(1:half_N2);        % Positive frequency components
second_half = X2(half_N2+2:end);   % Negative frequency components

% Handle midpoint for even-length signals
MidTerm = X2(half_N2+1) / 2;  % Midpoint term since signal length is even

% Insert zeros for interpolation
zero_list = zeros(K*N2 - 1, 1);    % Zero padding

% Reconstruct spectrum with zero padding
X_Z = [first_half; MidTerm; zero_list; MidTerm; second_half];

% Compute inverse FFT and scale
x_interpolated = real(ifft(X_Z)) * (K+1);

% Adjust the length of the interpolated signal to match original
x_n = x_interpolated(1:(K+1)*(N2-1)+1);  % Select necessary samples

% Calculate the 2-norm difference between interpolated and original signals
len_diff = length(x) - length(x_n);

% Zero-pad the shorter signal if necessary
if len_diff > 0
    x_n = [x_n; zeros(len_diff, 1)];  % Pad interpolated signal
elseif len_diff < 0
    x_n = x_n(1:length(x));           % Truncate interpolated signal
end

% Compute the 2-norm difference
norm_diff = norm(x - x_n);  % Euclidean distance (2-norm)
disp(['2-Norm Difference: ', num2str(norm_diff)]);
```

```
2-Norm Difference: 6.1447
```

```
% Plot the first 50 samples of both original and interpolated signals as stem plots
figure;
stem(1:50, x(1:50), 'b', 'DisplayName', 'Original x');  % Plot original signal
```

```
hold on;
stem(1:50, x_n(1:50), 'r', 'DisplayName', 'Interpolated x_n');  % Plot interpolated signal
hold off;
legend('show');
title('Comparison of Original and Interpolated Signals (First 50 Samples)');
xlabel('Sample Index');
ylabel('Signal Amplitude');
```



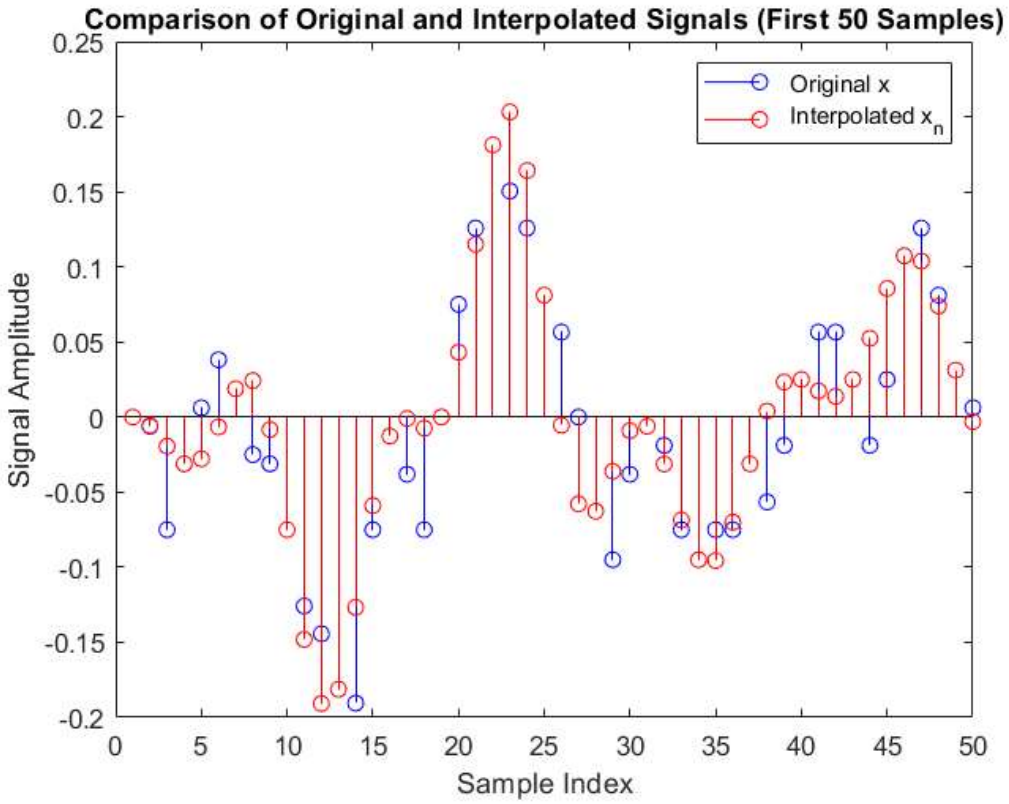
b) Interpolate the signal x3 with K = 2

```
% Parameters
N = 20000;  % Number of samples

% Obtain signals
x = y(1:N);       % Original signal
x3 = x(1:3:N);    % Subsample every 3rd sample (decimated signal)

% Interpolation process for x3 with K = 2 (frequency-domain zero-padding)
K = 2;            % Upsampling factor (K = 2 means interpolating back to original rate)

% Compute FFT of x3
X3 = fft(x3);
N3 = length(X3);  % Length of the DFT
half_N3 = floor(N3 / 2);  % Make sure it's an integer (half the length of the DFT)

% Split the DFT into two parts (positive and negative frequencies)
first_half = X3(1:half_N3);        % Positive frequency components
second_half = X3(half_N3+2:end);   % Negative frequency components (ensure valid indices)

% Handle midpoint for even-length signals
MidTerm = X3(half_N3+1) / 2;  % Midpoint term since signal length is even
```

```
% Insert zeros for interpolation
zero_list = zeros(K*N3 - 1, 1);    % Zero padding

% Reconstruct spectrum with zero padding
X_Z = [first_half; MidTerm; zero_list; MidTerm; second_half];

% Compute inverse FFT and scale
x_interpolated = real(ifft(X_Z)) * (K+1);

% Adjust the length of the interpolated signal to match original
x_n = x_interpolated(1:(K+1)*(N3-1)+1);  % Select necessary samples

% Calculate the 2-norm difference between interpolated and original signals
len_diff = length(x) - length(x_n);

% Zero-pad the shorter signal if necessary
if len_diff > 0
    x_n = [x_n; zeros(len_diff, 1)];  % Pad interpolated signal
elseif len_diff < 0
    x_n = x_n(1:length(x));           % Truncate interpolated signal
end

% Compute the 2-norm difference
norm_diff = norm(x - x_n);  % Euclidean distance (2-norm)
disp(['2-Norm Difference for x3: ', num2str(norm_diff)]);
```

```
2-Norm Difference for x3: 8.3653
```

```
% Plot the first 50 samples of both original and interpolated signals as stem plots
figure;
stem(1:50, x(1:50), 'b', 'DisplayName', 'Original x');  % Plot original signal
hold on;
stem(1:50, x_n(1:50), 'r', 'DisplayName', 'Interpolated x_n');  % Plot interpolated signal
hold off;
legend('show');
title('Comparison of Original and Interpolated Signals (First 50 Samples)');
xlabel('Sample Index');
ylabel('Signal Amplitude');
```



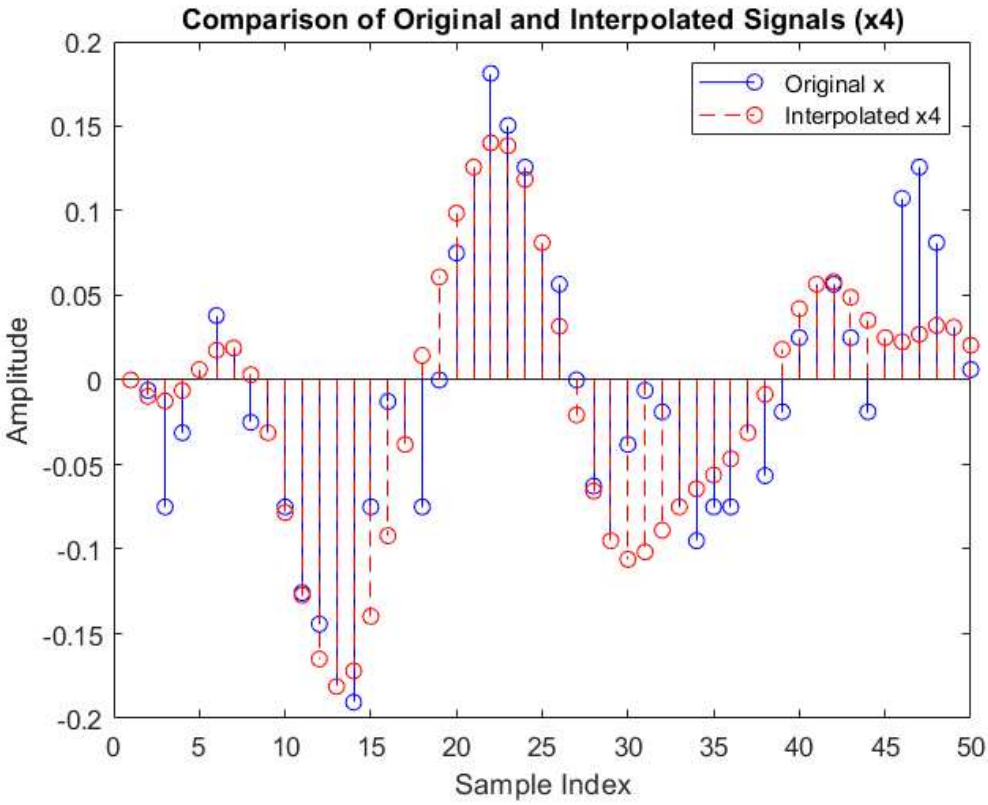
c) Interpolate the signal x4 with K = 3

```
% Parameters
N = 20000;  % Number of samples

% Obtain signals
x = y(1:N);       % Original signal
x4 = x(1:4:N);    % Subsample every 4th sample (decimated signal)

% Interpolation process for x4 with K = 3 (frequency-domain zero-padding)
K = 3;            % Upsampling factor (K = 3 means interpolating back to original rate)

% Compute FFT of x4
X4 = fft(x4);
N4 = length(X4);  % Length of the DFT
half_N4 = floor(N4 / 2);  % Make sure it's an integer (half the length of the DFT)

% Split the DFT into two parts (positive and negative frequencies)
first_half = X4(1:half_N4);        % Positive frequency components
second_half = X4(half_N4+2:end);   % Negative frequency components (ensure valid indices)

% Handle midpoint for even-length signals
MidTerm = X4(half_N4+1) / 2;  % Midpoint term since signal length is even

% Insert zeros for interpolation
zero_list = zeros(K*N4 - 1, 1);    % Zero padding

% Reconstruct spectrum with zero padding
X_Z = [first_half; MidTerm; zero_list; MidTerm; second_half];
```

```
% Compute inverse FFT and scale
x_interpolated = real(ifft(X_Z)) * (K+1);

% Adjust the length of the interpolated signal to match original
x_n = x_interpolated(1:(K+1)*(N4-1)+1);  % Select necessary samples

% Calculate the 2-norm difference between interpolated and original signals
len_diff = length(x) - length(x_n);

% Zero-pad the shorter signal if necessary
if len_diff > 0
    x_n = [x_n; zeros(len_diff, 1)];  % Pad interpolated signal
elseif len_diff < 0
    x_n = x_n(1:length(x));           % Truncate interpolated signal
end

% Compute the 2-norm difference
norm_diff = norm(x - x_n);  % Euclidean distance (2-norm)
disp(['2-Norm Difference for x4: ', num2str(norm_diff)]);
```

```
2-Norm Difference for x4: 23.4998
```

```
% Plot the first 50 samples of both original and interpolated signals
figure;
stem(1:50, x(1:50), 'b', 'DisplayName', 'Original x');  % Plot original signal
hold on;
stem(1:50, x_n(1:50), 'r--', 'DisplayName', 'Interpolated x4');  % Plot interpolated signal
legend;
title('Comparison of Original and Interpolated Signals (x4)');
xlabel('Sample Index');
ylabel('Amplitude');
hold off;
```


Comparison of Original and Interpolated Signals (x4)
Original x
Interpolated x4
Amplitude
Sample Index
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
5
10
15
20
25
30
35
40
45
50