Implementation of N - point FFT and IFFT Algorithm

C

Objectives

1. To understand the computation of Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) for discrete-time sequences in DSP.
2. To analyze the spectral properties of signals using the Power Spectral Density (PSD) and apply filtering techniques to remove noise.
3. To compare the time-domain representation of clean and noisy signals before and after applying the filtering process using FFT and IFFT.

Apparatus

Software: MATLAB R2022b

Hardware: Personal Computer

Theory

Signal transformations between the time and frequency domains are fundamental to signal processing. The Discrete Fourier Transform (DFT) and its efficient variant (FFT) enable the decomposition of signals into their frequency components. These techniques are essential for analyzing the spectral properties of signals. The IFFT allows the recovery of time-domain signals from their frequency representations.

- Discrete Fourier Transform (DFT)

The DFT is a mathematical technique that converts a discrete signal from the time domain to the frequency domain. It breaks down a sequence of values into components of different frequencies, helping to analyze a signal's frequency characteristics.

The DFT of a sequence x[n] with N samples is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad k = 0,1,2,..,N-1$$

In this context, $x[n]$ represents the reconstructed time-domain signal, while $X[k]$ denotes the frequency-domain representation obtained through the FFT. The variable N corresponds to the total number of points in the sequence, and j is the imaginary unit.

The DFT allows us to determine how many of each frequency is present in the signal. In MATLAB, the Discrete Fourier Transform (DFT) is computed using the '`fft`' function, which stands for Fast Fourier Transform. Although MATLAB does not provide a specific function named '`dft`', the '`fft`' function is used to calculate the DFT efficiently. The FFT

is an optimized algorithm that reduces the time complexity from O (N^2) for a standard DFT to O (N log N), making it computationally efficient, especially for large datasets. Therefore, when computing the DFT in MATLAB, the '`fft`' function transforms the time domain to the frequency domain.

- Fast Fourier Transform (FFT)

The FFT is an optimized algorithm for computing the DFT. While the DFT has a time complexity of O (N^2), the FFT reduces this to O (N log N), making it much faster for large datasets. The FFT operates on the same principles as the DFT but uses a divide-and-conquer approach to split the calculation into smaller parts, allowing for more efficient computation.

- Inverse Fast Fourier Transform (IFFT)

The IFFT converts a frequency-domain signal back to its time-domain representation. It is the inverse operation of the FFT and allows us to reconstruct the original time-domain signal from its frequency components. The IFFT is mathematically represented as:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j\frac{2\pi}{N}kn}, \quad n = 0,1,2,..,N-1$$

- Waveforms and Their Harmonic Structure

In periodic signals, harmonics integer multiples of a fundamental frequency contribute to a signal's overall shape. Analyzing the Fourier series of waveforms reveals the presence and amplitude of these harmonics, which play a critical role in defining signal characteristics. Common waveforms such as the sine wave, square wave, sawtooth wave, and triangular wave each have unique harmonic structures. The Fourier series representation of each waveform demonstrates how specific harmonics combine to produce their distinct shape, providing valuable insight into the harmonic composition and behavior of periodic signals.

a) Sine Wave

A sine wave represents the simplest periodic waveform and consists of a single frequency with no harmonics. The equation for a sine wave at frequency f_0 is,

$$x(t) = Asin(2\pi f_0 t + \phi)$$

Where A is amplitude and ϕ is the phase. Only the fundamental frequency component is present, Hence, k = 1.

b) Square Wave

A square wave contains only odd harmonics, with each harmonic's amplitude decreasing inversely with its order. The Fourier series for a square wave is,

$$x_{square}(t) = \frac{8}{\pi^2} \sum_{k=1,3,5,\ldots} \frac{(-1)^{(k-1)/2}}{k^2} \sin(2\pi k f_0 t), \quad k = \text{integer (odd)}$$

c) Sawtooth Wave

A sawtooth wave includes all harmonics (both odd and even) with amplitudes decreasing *1/k*. The Fourier series for a sawtooth wave is,

$$x_{sawtooth}(t) = \frac{2}{\pi} \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \sin(2\pi k f_0 t), \quad k = \text{integer (positive)}$$

d) Triangular Wave

Similar to the square wave, a triangular wave contains only odd harmonics. However, the amplitude decreases more quickly, at a rate of $1/k^2$. Its Fourier series is,

$$x_{triangle}(t) = \frac{8}{\pi^2} \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \sin(2\pi k f_0 t), \quad k = \text{integer (odd)}$$

MATLAB code Overview

1. Signal Generation with Two Frequencies

A continuous-time signal composed of two sine waves with distinct frequencies is chosen for its fundamental role in signal analysis. The signal comprises two components, i.e., a 50 Hz sine wave and a 120 Hz sine wave.

MATLAB code for generating the Continuous-Time Signal is,

```
% Define time vector (0 to 1 second) with a step
of 0.001 seconds

dt = 0.001;  t
= 0:dt:1;

% Generate signal with 50 Hz and 120 Hz
components
x = sin(2*pi*50*t) + sin(2*pi*120*t);
```

2. Noise Addition

Random noise can be added to the continuous signal to simulate real-world conditions.

MATLAB code for Noise Addition is,

```
% Add random noise with a standard deviation of 2.5
y = x + 2.5*randn(size(t));
```

3. Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is applied to the noisy signal to transform it into the frequency domain, allowing the analysis of its frequency components.

MATLAB code for applying Fourier Transform is,

```
% Length of the time vector (number of samples)
N = length(t);

% Apply FFT to the noisy signal
Y = fft(y, N);
```

4. Computation of Power Spectral Density (PSD)

The Power Spectral Density (PSD) is computed to quantify the power distribution of the signal across different frequencies.

MATLAB code for applying Fourier Transform is,

```
% Compute the Power Spectral Density (PSD)
PSD = Y .* conj(Y) / N;

% Generate the frequency axis for the FFT plot
freq = 1/(dt*N) * (0:N-1);
```

5. Filtering Noise via Power Spectral Density (PSD)

The PSD is analyzed to remove low-power frequency components, effectively filtering out noise. The significant frequency components are retained, and the signal is reconstructed using the inverse FFT.

MATLAB code for filtering noise via PSD is,

```
% Select only the first half of the frequencies (positive
frequencies)
L = 1:floor(N/2);
```

```
 % Set a threshold to filter out low-power frequencies
indices = PSD > 60;

% Apply the filter to the Power Spectral Density
 PSD_filtered = PSD .* indices;

% Zero out Fourier coefficients for frequencies below the
 threshold
 Y = Y .* indices;
```

6. Reconstructing the Filtered Signal using Inverse FFT
   The filtered frequency-domain signal is converted back to the time domain using the inverse Fast Fourier Transform (IFFT).

   MATLAB code for filtering noise via PSD is,

```
% Reconstruct the filtered signal using the IFFT
yFilt = ifft(Y);
```

Procedure

1. Open MATLAB and create a new script by navigating to **Home → New → Script**.

2. Implement the MATLAB code in the script to generate a continuous-time sinusoidal wave with 50 Hz and 120 Hz frequencies.

3. Add a Gaussian noise to the clean signal and overlay the noisy signal on the same plot as the clean signal. Label the curves accordingly, as shown in Figure 1.

4. Apply the Fourier Transform to the noisy signal and compute the Power Spectral Density (PSD) by taking the magnitude squared of the FFT result.

5. Generate a frequency axis corresponding to the PSD and plot the power spectrum of the noisy signal, as shown in Figure 2.

6. Set a threshold to filter out low-magnitude frequencies from the power spectrum.

7. Modify the Fourier coefficients of the signal based on the filtering condition to remove noise. Then, plot the power spectrums of the noisy signal and the filtered signal on the same graph, as shown in Figure 3.

8. Compute the inverse FFT to obtain the filtered signal in the time domain.

9. Plot the original clean signal and the noisy signal on the same figure, displaying them in the first row. In the second row of the same figure, plot the clean signal and noisy signal again for a direct comparison, as shown in Figure 4.

10. Save the script with an appropriate name (e.g., fft_and_ifft_algorithm.m).

11. Save the figures in JPEG format by selecting **File → Save As** in the Figure window.
.

Results

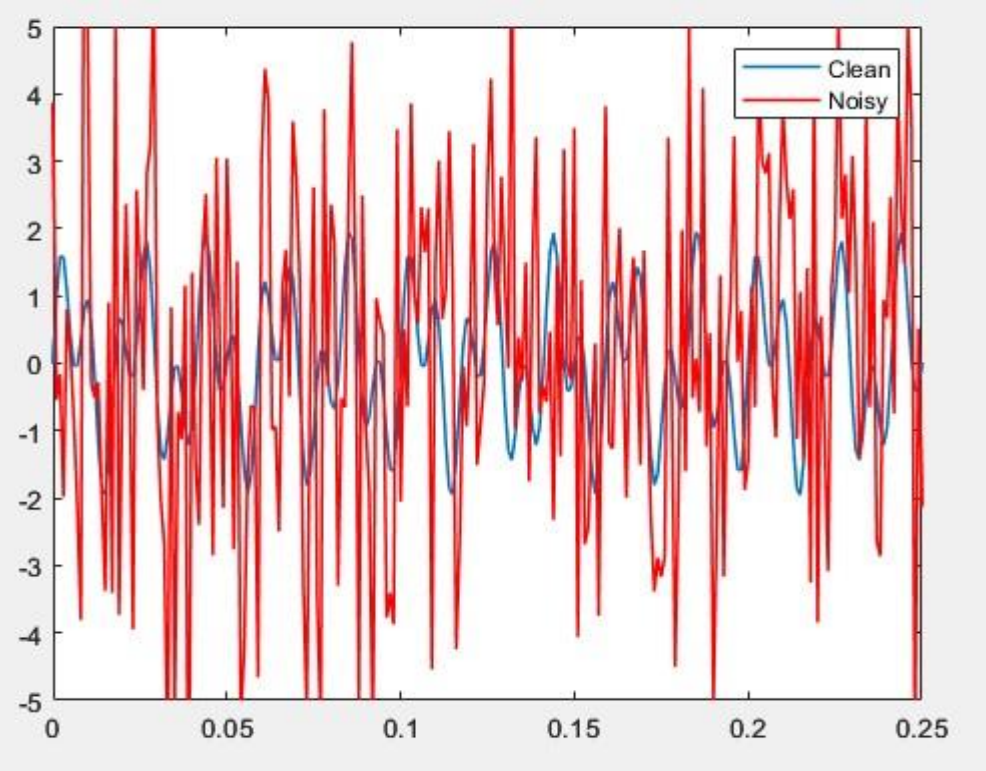

Figure 1. Clean and Noisy Continuous-Time Signal.

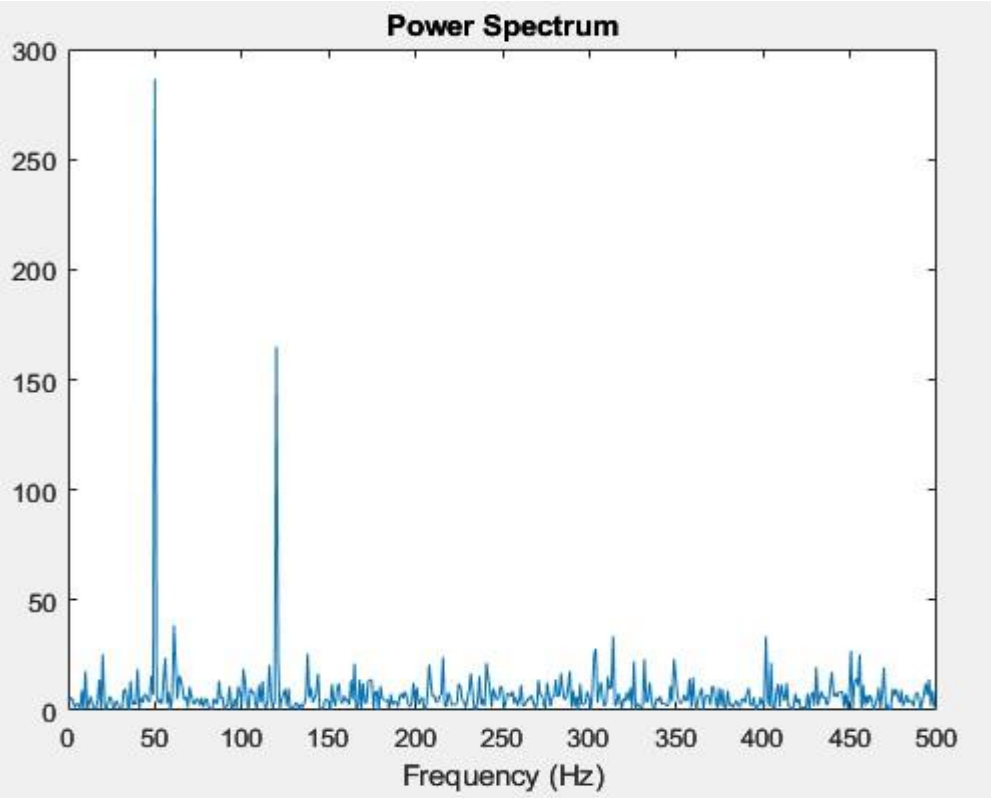

Figure 2. Power Spectrum of the Noisy Signal.

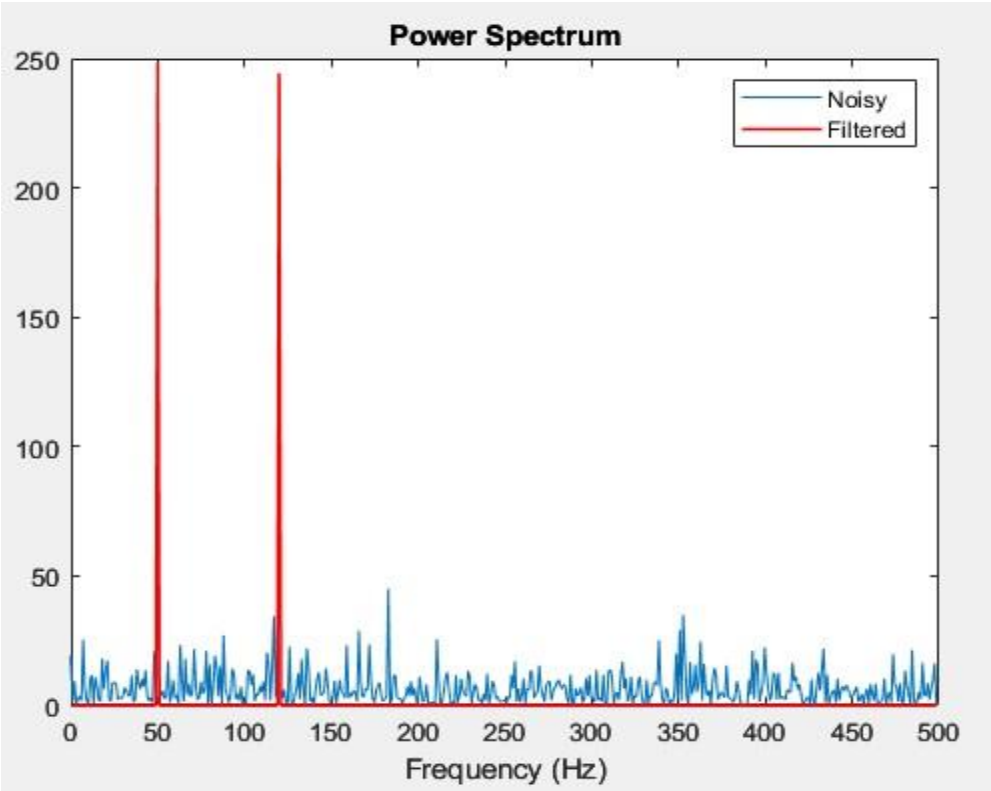

Figure 3. Power Spectrums of Noisy and Filtered Signals after applying filtering.

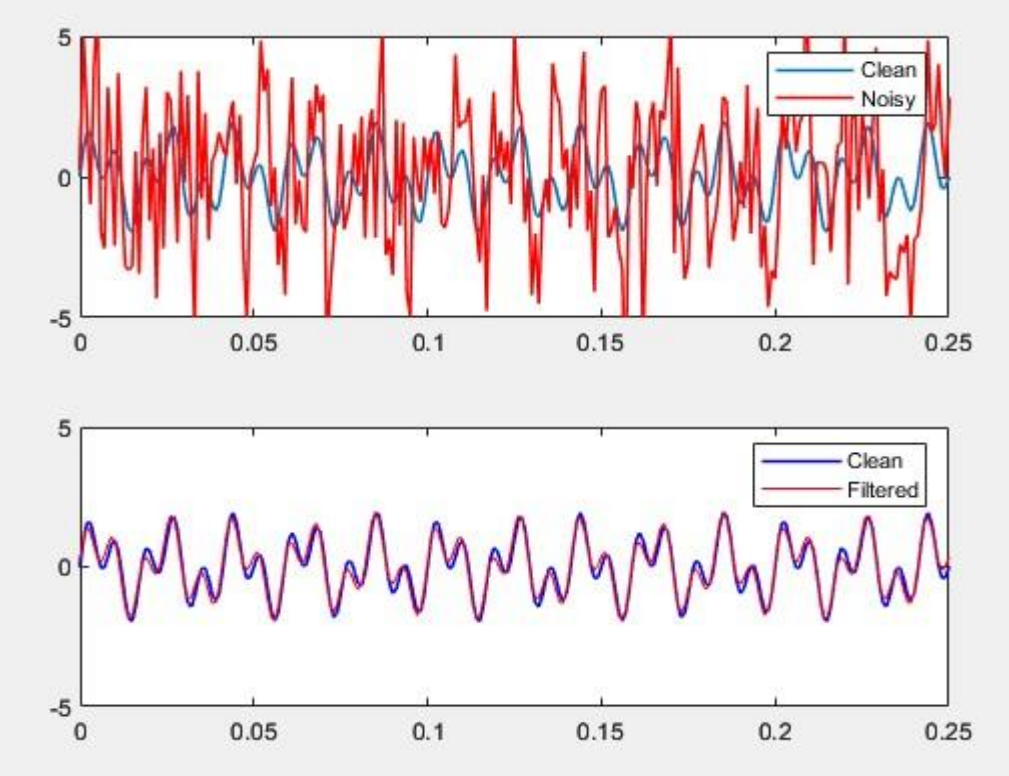

Figure 4. Comparison of Clean and Noisy Signals in Time Domain (before and after filtering).

Exercise

1) a) Write a MATLAB program to generate a signal containing three sinusoidal components at frequencies of 20 Hz, 200 Hz, and 400 Hz, each with unit amplitude. Sample this signal at 900 Hz, which is above the Nyquist rate (800 Hz) for the highest frequency component.

   b) Adjust the sampling rate to 450 Hz, which is below the Nyquist rate for the highest frequency component and observe the effect of aliasing. For each sampling rate (900 Hz, 450 Hz), apply the FFT and plot the frequency spectrum in two subplots within a single figure(sampling rate 900Hz in black, sampling rate 450Hz in blue). Label each subplot with the respective sampling rate and analyze the effects of aliasing on frequency detection when the sampling rate is reduced below the Nyquist rate.

   c) Based on your plots, explain how aliasing affects the detection of frequencies when the sampling rate is below the Nyquist rate. What general rule can you infer about the sampling rate and accurate frequency representation?

2) a) Consider the sequence x[n]={1,1,0,0} and compute its Fast Fourier Transform (FFT) to obtain the frequency-domain representation using MATLAB, and then apply the inverse transform to reconstruct the original sequence. Display the reconstructed sequence in the command window alongside the original for comparison and generate a figure with three

subplots(In red ,black ,yellow colors). In the first subplot, create a stem plot to visualize the original sequence over its discrete time index n. In the second subplot, illustrate the magnitude of the frequency components over the frequency bins. In the third subplot, display a stem plot of the reconstructed sequence to visually confirm it matches the original. Use a distinct color for each plot and add grid lines to all the subplots for better visualization.

b) Verify if the reconstructed sequence matches the original by using a small tolerance level to account for computational precision. Display a message in the command window as follows,

i. If a match output "The reconstructed sequence matches the original sequence".

ii. Otherwise output "The reconstructed sequence does not match the original sequence".

3) a) Consider the sequence x[n]={1,0,1,0} and use MATLAB to analyze its frequency domain representation with the FFT. Compute the FFT of the original sequence, then zero-pad the sequence to lengths of 5 and 10 and compute the FFT for each zero-padded version. For each of the three sequences (original, zero-padded to 5, and zero-padded to 10), display both the magnitude and phase of the FFT to observe the effect of zero-padding on frequency resolution. Reconstruct each sequence by applying the IFFT to each FFT result, then compare each reconstructed signal with its corresponding padded sequence to confirm reconstruction accuracy. Display your findings in a single figure with six subplots as the original sequence in the time domain, the zero-padded sequences in the time domain, the magnitude of the FFT for the original sequence, the magnitude of the FFT for both zero-padded sequences, the phase spectrum of each sequence, and the real part of the reconstructed signals from each IFFT.

> **Note** - Label all graphs appropriately, use a distinct color for each plot, and add grid lines to all subplots for better visualization. When two graphs appear on the same plot, use filled and empty stems to distinguish between them. Include legends in relevant plots for clarity.

b) Print the maximum absolute difference between each reconstructed signal and its corresponding padded sequence in the command window to verify reconstruction accuracy.

c) Explain how zero-padding affects the frequency resolution of the FFT. Why does zero-padding does not change the actual frequency components of the original signal?

4) a) Write a MATLAB program to create a discrete-time square wave signal with a fundamental frequency $f_0 = 8$ Hz, sampled at fs =400 Hz, over a duration of 1 second. Compute the FFT of the square wave and plot the magnitude spectrum, phase spectrum, and both real and imaginary parts of the FFT in four separate subplots(Use black, green,

yellow, blue colors).. Use stem plots for clear visualization. Identify the primary harmonic frequencies, which appear at odd multiples of the fundamental frequency. Set all other FFT components to zero and use the inverse FFT to reconstruct an approximation of the square wave using only its fundamental and first few harmonics (e.g., 3rd, 5th, and 7th).

b) Plot the original square wave signal and the reconstructed signal using selected harmonics on the same graph to visually compare their shapes.

> **Note** - Label all graphs appropriately, use a distinct color for each plot, and add grid lines to all subplots for better visualization. When two graphs appear on the same plot, use solid and dashed lines to distinguish between them. Include legends in relevant plots for clarity.

c) Calculate and display the Mean Squared Error (MSE) between the original and the reconstructed signals as a measure of reconstruction accuracy.

d) Discuss the effect of including more or fewer harmonics on the quality of the reconstructed signal.