|| 7

SCHOOL OF ARTIFICIAL INTELLIGENCE

24MA602 Computational Mathematics for Data Science

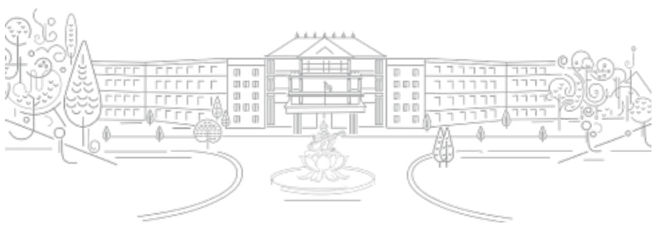
Assignment Set-7

Submitted by

Siju K S
Reg. No. CB.AI.R4CEN24003
Center for Computational Engineering & Networking

Submitted to

Prof. (Dr.) Soman K.P.
Professor & Dean
School of Artificial Intelligence
Amrita Vishwa Vidyapeetham



SEPTEMBER
2024

Contents

1 Assignment 80
Introduction to Discrete-Time Fourier Transform (DTFT) and FIR Filters **9**
1.1 Introduction to Discrete-Time Fourier Transform (DTFT) 9
1.1.1 FIR filter design as an optimization problem 12
1.2 Frequency Response of Filters 14
1.3 Haar Wavelet 16
1.4 Daubechies Wavelet 17
1.4.1 Sparsity-based signal processing and L1 norm optimization 19
1.4.2 Mathematical formulation 20

2 Assignment 81
Applications of Discrete Fourier Transform **23**
2.1 Introduction 23
2.1.1 Mathematical Representation 23
2.1.2 Linear algebra perspective 23
2.2 What does the dft really do? 24
2.2.1 DFT interpretation of a signal 24
2.2.2 Example: DFT basis for signal length 8 24
2.2.3 Why is the central basis vector's wave number ± 4? 25
2.2.4 Geometrical insight of DFT basis 25
2.2.5 Properties of DFT 26
2.3 DFT Spectrum and FFT 27
2.4 Example: DFT of a Simple Signal 28
2.4.1 MATLAB Code Example 28
2.4.2 A fast summary of key points 29
2.5 Applications of DFT 29
2.5.1 Spectral analysis 29
2.5.2 Filter design 29
2.5.3 Image Processing 29
2.5.4 Modulation and demodulation 30
2.5.5 Signal reconstruction 30
2.6 Example: FIR Filter Design Using DFT 30
2.6.1 Problem statement 30
2.6.2 Steps 30
2.7 Designing Optimal Filters for Denoising 31
2.7.1 Mathematical steps for optimal filter design 31
2.8 Denoising Signals Without Ground Truth 33
2.8.1 Mathematical formulation for denoising 34
2.9 Comparison with Other Transform Techniques 36
2.9.1 DFT vs. Fast Fourier Transform (FFT) 36
2.9.2 DFT vs. Wavelet Transform 36
2.10 Tasks 36

3 Assignment 82
Fibonacci Sequence and Its Applications in Optimization **41**
3.1 Introduction 41
3.1.1 Generating the Fibonacci sequence 41
3.1.2 Golden section ratio 41
3.1.3 Matrix representation of the Fibonacci sequence 42
3.1.4 Similarity matrix representation 42
3.1.5 Eigenvalues of the Fibonacci matrix 43
3.1.6 Relationship between ϕ_1 and ϕ_2 43
3.1.7 Binet's formula 44
3.2 Applications of Golden Section Ratio in Optimization, Signal Processing, and Machine Learning 44
3.2.1 Optimization in signal processing 44
3.2.2 Optimization in Machine Learning 45

4 Assignment 83
Computing Derivative Using FFT **47**
4.1 Introduction 47
4.1.1 Sampling range 47
4.1.2 Angular frequency values 47
4.2 Finding the Derivative of sech(x) Using FFT 47
4.3 Tasks 48
4.3.1 A die is thrown 100 times: compute probabilities of getting all possible sums (100 to 600) using FFT 49
4.3.2 A Coin is tossed 100 times: find the probability of getting more than 70 heads using FFT 50
4.3.3 Using FFT show that the magnitude of ft of Gaussian function is again Gaussian 51
4.3.4 A Die is thrown 100 times: compute probability of getting the sum on the die as more than 373 52

List of Tables

1.1 Comparison of FIR, Haar, and Daubechies Filters 19

List of Figures

1.1 Visualization of output from FIR filter and DFT. 12
1.2 Magnitude and Phase of optimized Finite Impulse Response Filter. 14
1.3 Illustration of frequency response of FIR filter 16
1.4 Magnitude of high pass and low pass Haar filter. 17
1.5 Denoising skill of Daubechies Wavelet filter on a noisy sine wave. 20

2.1 DFT of a wave with wave number 8. 26
2.2 Demonstration of a DFT filter 31
2.3 Effect of optimized DFT filtering on a noisy image. 33
2.4 Optimized DFT that denoise a signal without ground truth. 36
2.5 Frequency plot of the triangular wave 37

4.1 Linear convolution of sequence using FFT. 49
4.2 Probabilities for getting sum from 100 to 600. 50
4.3 Magnitude of Gaussian distribution using FFT. 52
4.4 Low pass filtering of an image using FFT. 55

1 | Assignment 80 Introduction to Discrete-Time Fourier Transform (DTFT) and FIR Filters

1.1 Introduction to Discrete-Time Fourier Transform (DTFT)

The **Discrete-Time Fourier Transform (DTFT)** is a mathematical tool used to analyze discrete-time signals in the frequency domain. It transforms a sequence of discrete time samples into a continuous function of frequency, providing insights into the frequency components present in the signal.

Mathematical Definition

For a discrete-time signal $x[n]$, the DTFT is defined as:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

The inverse DTFT allows us to recover the original signal from its frequency-domain representation:

$$x[n] = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} X[k]e^{j\omega n}$$

Intuition

The DTFT provides a way to understand how much of each frequency is present in the signal. By analyzing the magnitude and phase of $X(e^{j\omega})$, we can determine the harmonic content and periodicity of the original signal.

Properties of the DTFT

- **Linearity**: If $x_1[n]$ and $x_2[n]$ are signals, then the DTFT satisfies:

$$ax_1[n] + bx_2[n] \implies aX_1(e^{j\omega}) + bX_2(e^{j\omega})$$

- **Time Shifting**: Shifting the signal in time affects its DTFT:

$$x[n - n_0] \implies X(e^{j\omega})e^{-j\omega n_0}$$

- **Frequency Shifting**: Multiplying a signal by an exponential shifts its frequency content:

$$x[n]e^{j\omega_0 n} \implies X(e^{j(\omega-\omega_0)})$$

- **Convolution**: The DTFT of the convolution of two signals equals the product of their DTFTs:

$$x[n] * h[n] \implies X(e^{j\omega})H(e^{j\omega})$$

Practical Uses in Data Science and AI

- **Signal Modeling**: In applications like speech recognition and audio processing, the DTFT helps model and analyze sound signals, allowing systems to understand frequency characteristics.
- **Noise Removal**: By transforming a noisy signal into the frequency domain, one can easily identify and filter out unwanted frequencies, enhancing the quality of the signal.
- **Feature Extraction**: The frequency components extracted via DTFT can serve as features for machine learning models, aiding in tasks like pattern classification, where frequency-based features may offer more discriminative power.

Introduction to Finite Impulse Response (FIR) Filters

Finite Impulse Response (FIR) filters are digital filters characterized by a finite number of non-zero coefficients in their impulse response. FIR filters are essential tools in signal processing due to their stability and linear phase response.

Mathematical Definition

An FIR filter is defined by its impulse response $h[n]$, which is non-zero for a finite number of samples:

$$h[n] = \{h[0], h[1], \ldots, h[N-1]\}$$

The output $y[n]$ of the FIR filter can be computed using the convolution of the input signal $x[n]$ with the filter coefficients $h[n]$:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k]$$

Intuition

The impulse response $h[n]$ characterizes how the filter affects the input signal over time. The convolution operation blends the input signal with the filter's response, producing a new signal that highlights certain features or removes others.

Properties of FIR Filters

- **Stability**: FIR filters are inherently stable, meaning their output remains bounded for a bounded input, making them reliable for real-time applications.
- **Linear Phase**: FIR filters can be designed to have a linear phase response, which is crucial for applications where maintaining the waveform shape of the signal is important (e.g., audio processing).
- **Realization**: FIR filters can be implemented in various forms (e.g., direct form, cascade form), making them flexible for different applications.

Practical Uses in Data Science and AI

- **Signal Smoothing**: FIR filters are widely used for smoothing time series data, helping to reduce variability and highlight trends, which is particularly useful in financial data analysis.
- **Feature Enhancement**: In image processing, FIR filters can enhance features such as edges and textures, which are essential for tasks like object recognition and segmentation.
- **Noise Reduction**: FIR filters are effective for noise reduction in signals, improving data quality for subsequent analysis or machine learning tasks.

Example of FIR Filter

Consider a simple FIR filter with the following coefficients:

$$h[n] = \{0.2, 0.5, 0.2\} \quad \text{for } n = 0, 1, 2$$

If the input signal is given by:

$$x[n] = \{1, 2, 3, 4\} \quad \text{for } n = 0, 1, 2, 3$$

The output of the FIR filter can be calculated as follows:

$$y[0] = 0.2 \cdot 1 = 0.2$$

$$y[1] = 0.2 \cdot 1 + 0.5 \cdot 2 = 0.2 + 1 = 1.2$$

$$y[2] = 0.2 \cdot 1 + 0.5 \cdot 2 + 0.2 \cdot 3 = 0.2 + 1 + 0.6 = 1.8$$

$$y[3] = 0.2 \cdot 2 + 0.5 \cdot 3 + 0.2 \cdot 4 = 0.4 + 1.5 + 0.8 = 2.7$$

$$y[4] = 0.5 \cdot 3 + 0.2 \cdot 4 = 1.5 + 0.8 = 2.3$$

The resulting output signal is:

$$y[n] = \{0.2, 1.2, 1.8, 2.7, 2.3\}$$

Matlab Code for Visualization

Below is the matlab code to visualize the DTFT and the FIR filter:

```matlab
% Define the input signal
x = [1, 2, 3, 4];

% Define the FIR filter coefficients
h = [0.2, 0.5, 0.2];

% Compute the output of the FIR filter
y = conv(x, h);

% Compute the DTFT of the input signal
N = 512; % Number of points for DTFT
omega = linspace(-pi, pi, N);
X = zeros(1, N);

for k = 1:N
    X(k) = sum(x .* exp(-1j * omega(k) * (0:length(x)-1)));
end

% Plot the results
figure;

subplot(3, 1, 1);
stem(x, 'filled', 'LineWidth', 1.5);
title('Input Signal x[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

```

```
subplot(3, 1, 2);
stem(h, 'filled', 'LineWidth', 1.5);
title('FIR Filter Coefficients h[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

subplot(3, 1, 3);
plot(omega, abs(X), 'LineWidth', 1.5);
title('Magnitude of DTFT |X(e^{j\omega})|');
xlabel('\omega (radians/sample)');
ylabel('|X(e^{j\omega})|');
grid on;
```

Output of the code is shown in Figure 1.1

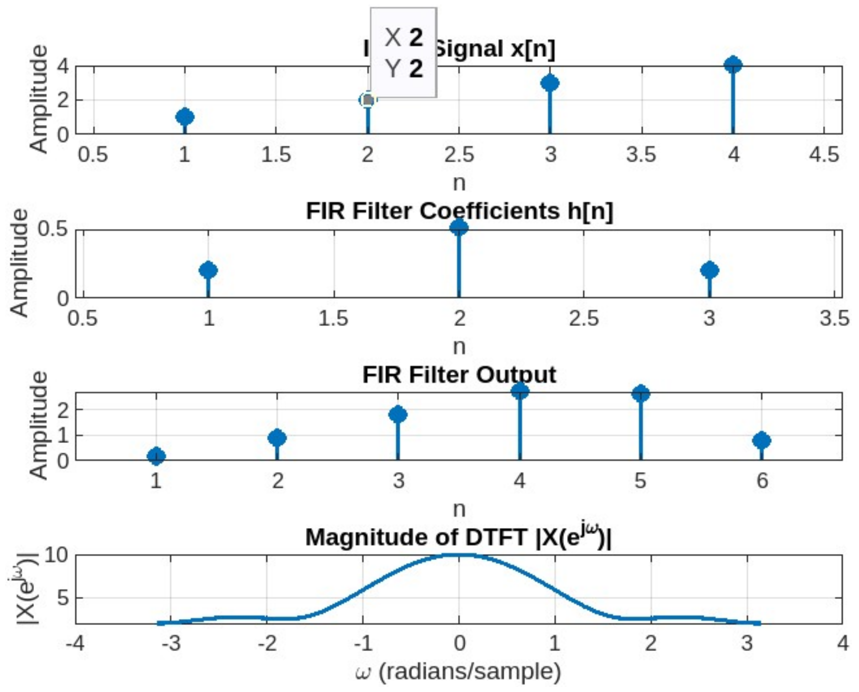


Figure 1.1: Visualization of output from FIR filter and DFT.

Visualization Results

- The **first plot** displays the input signal $x[n]$.
- The **second plot** shows the FIR filter coefficients $h[n]$.
- The **third plot** illustrates the magnitude of the DTFT of the input signal $|X(e^{j\omega})|$.

1.1.1 FIR filter design as an optimization problem

To frame the FIR filter design as an optimization problem, follow these steps: First, define the objective function, which aims to minimize the error between the desired filter response and the actual response of the designed filter. Next, establish the necessary constraints, including passband and stopband requirements based on the desired specifications. Finally, use CVX, a MATLAB package for specifying and solving convex programs, to solve the optimization problem efficiently.

Assume we want to design an FIR low-pass filter that minimizes the error between the desired frequency response and the actual frequency response of the filter while satisfying the passband and stopband constraints.

Below is an example MATLAB code using CVX to optimize the filter coefficients:

```
% Parameters
fs = 1000;                % Sampling frequency (Hz)
N = 20;                   % Filter order
f_pass = 15;              % Passband frequency (Hz)
f_stop = 20;              % Stopband frequency (Hz)
num_points = 512;         % Number of frequency points
f = linspace(0, fs/2, num_points); % Frequency range
H_d = double(f < f_pass);
cvx_begin quiet
    variable h(N+1);
    minimize(norm(abs(H) - H_d', 2))
    subject to
        h >= 0;
        norm(h, 1) <= 1;
cvx_end
h_opt = h;
H = zeros(num_points, 1);
for k = 1:num_points
        H(k) = sum(h_opt .* exp(-1j * 2 * pi * f(k) * (0:N)') / fs)
            ;
end
figure;
freqz(h_opt, 1, num_points);
title('Optimized FIR Filter Frequency Response');
t = 0:1/fs:1-1/fs;
x = sin(2 * pi * 5 * t) + 0.5 * randn(size(t));
y_opt = filter(h_opt, 1, x);

% Visualization of the original and filtered signals
figure;
subplot(2, 1, 1);
plot(t, x, 'b', 'LineWidth', 1.5);
title('Original Signal with Noise');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(2, 1, 2);
plot(t, y_opt, 'r', 'LineWidth', 1.5);
title('Filtered Signal using Optimized FIR Filter');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

Output wave form is shown in Figure 1.2.

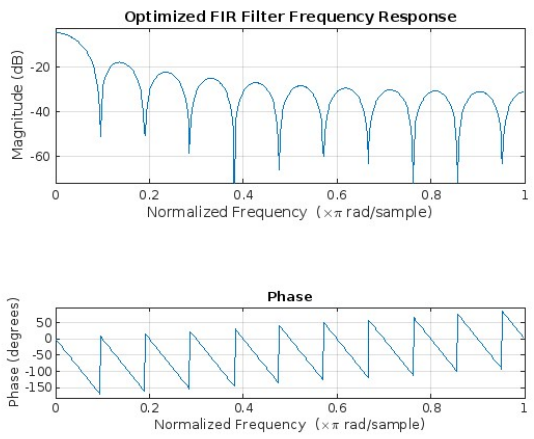


Figure 1.2: Magnitude and Phase of optimized Finite Impulse Response Filter.

1.2 Frequency Response of Filters

Understanding the DTFT and FIR filters is essential for working with digital signals. These concepts form the foundation for more complex signal processing tasks and are widely applicable in various fields, including telecommunications, audio processing, image analysis, and machine learning. By leveraging these concepts, practitioners can effectively model, enhance, and interpret signals to derive meaningful insights from complex data.
The formula

$$H(\omega) = h_0 + h_1 e^{-j\omega} + \cdots + h_N e^{-jN\omega} = R(\omega) e^{j\phi(\omega)}$$

is frequently cited in signal processing textbooks. It represents the *frequency response* of a system, especially for Finite Impulse Response (FIR) filters. Although it may look complex, we can break it down using basic concepts from complex numbers and waves.

Background: Complex Numbers in Signal Processing

1. Complex Numbers: A complex number is written as $z = a + jb$, where a is the real part, b is the imaginary part, and j is the imaginary unit ($j = \sqrt{-1}$). In signal processing, we use complex numbers to represent waves. For example, the complex exponential $e^{j\omega}$ represents a wave of frequency ω.
2. Complex Exponentials and Waves: A complex wave can be expressed as $e^{j\omega t}$, which is a combination of a cosine and sine wave (since $e^{j\omega t} = \cos(\omega t) + j\sin(\omega t)$). This is why complex numbers are so useful in representing signals.

What Does the Formula Mean?

This formula describes a system's *frequency response*. Imagine sending a wave (like a sound wave or a radio signal) through a system that delays and attenuates parts of the wave. The formula tells us how the output wave is modified after passing through the system.

Input Signal

The input signal can be visualized as a sine or cosine wave, represented as a complex exponential $e^{j\omega t}$, where ω is the frequency of the wave.

Delays and Attenuation

The coefficients $h_0, h_1, \ldots, h_N$ represent how the system scales (attenuates) and delays different parts of the input wave. Each term $h_n e^{-jn\omega}$ corresponds to a delayed and scaled version of the input wave.

Sum of Delayed Waves

The system takes several delayed versions of the input wave, adds them together, and modifies the signal in both magnitude and phase. This is what happens in Finite Impulse Response (FIR) filters, where each delayed wave interacts with the others.

Breaking Down the Formula

The frequency response formula:

$$H(\omega) = h_0 + h_1 e^{-j\omega} + \cdots + h_N e^{-jN\omega}$$

can be interpreted as:

- $H(\omega)$: The overall frequency response of the filter.
- h_n: The filter coefficients, which determine how much each delayed version of the signal contributes.
- $e^{-jn\omega}$: The delays, where ω is the frequency of the input signal and n is the amount of delay.

The formula describes how a signal's shape changes when it passes through the system, affecting both its magnitude and phase.

Magnitude and Phase Response

- **Magnitude Response** $R(\omega)$: The magnitude of the frequency response indicates how much the system amplifies or attenuates signals at different frequencies. It is calculated as the absolute value:
$$R(\omega) = |H(\omega)|$$
- **Phase Response** $\phi(\omega)$: The phase response shows how much the system shifts the signal in time for a given frequency. It is the angle (argument) of the complex function $H(\omega)$:
$$\phi(\omega) = \arg(H(\omega))$$

In simpler terms:

- **Magnitude**: How much the filter boosts or reduces the signal at a certain frequency.
- **Phase**: How much the filter shifts the signal in time at a certain frequency.

Application in Signal Processing

When multiple delayed and attenuated versions of a signal are added together, the overall change in the signal's magnitude and phase is given by this formula. This is useful in applications such as:

- **Noise Removal**: Filters can remove unwanted noise by attenuating specific frequencies.
- **Pattern Recognition**: Filters can enhance important features or patterns in signals, which is useful in machine learning.

Visualizing the Concept

The input can be seen as a simple wave, and the output is a combination of multiple delayed versions of this wave. The formula $H(\omega)$ describes how the filter affects the signal.

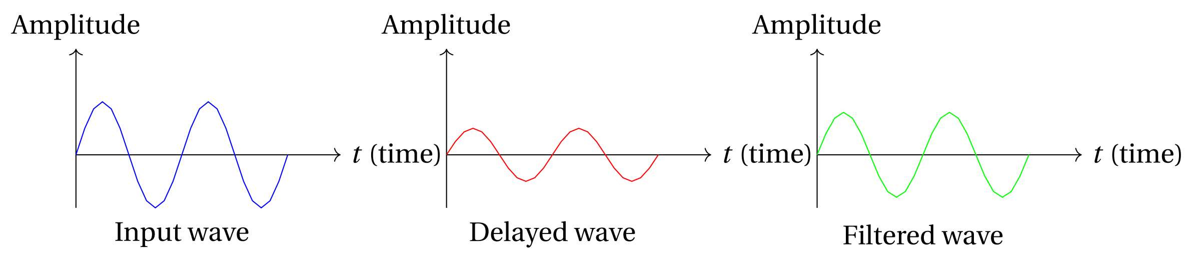


Figure 1.3: Illustration of frequency response of FIR filter

Figure 1.3 shows an input wave and how the filter delays and scales it. The final output is a combination of the delayed waves.

1.3 Haar Wavelet

The Haar wavelet is the simplest wavelet transform and can be thought of as a 2-tap FIR filter. It is highly localized in time but discontinuous, making it suitable for signals with sharp changes.

Mathematical Representation

The Haar scaling function $\phi(t)$ and wavelet function $\psi(t)$ are defined as:

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\psi(t) = \begin{cases} 1 & 0 \le t < \frac{1}{2} \\ -1 & \frac{1}{2} \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

Characteristics

- **Simple computation:** Haar wavelets are computationally efficient.
- **Energy preservation:** The Haar transform preserves the signal's energy.

Applicability

Haar wavelets are often used in:

- **Real-time applications** such as image compression (e.g., JPEG),
- Analyzing signals with sharp transitions, such as **edges in images**.

A sample `matlab` code to create a Haar filter is given below.

```
% Haar filter
close(gcf)
h0=1/sqrt(2);
h1=1/sqrt(2);
omega=0:0.1:pi;
```

```
Hlow=h0+h1*exp(-1*i*omega);
H_mag=abs(Hlow);
plot(omega,H_mag);
hold on
Hhigh=h0-h1*exp(-1*i*omega);H_mag=abs(Hhigh);
plot(omega,H_mag);
```

Output of the magnitude plots of high pass and low pass Haar filter is shown in Figure 1.4.

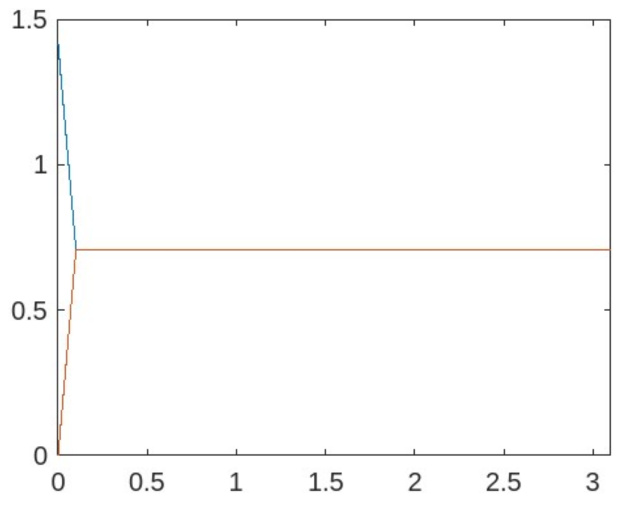


Figure 1.4: Magnitude of high pass and low pass Haar filter.

1.4 Daubechies Wavelet

Daubechies wavelets are a generalization of the Haar wavelet, providing smoother and more compactly supported waveforms. These wavelets are well-suited for analyzing smooth signals.

Mathematical representation

Daubechies wavelets use higher-order vanishing moments. For example, the Daubechies 2 (Db2) wavelet has 4 filter coefficients. The scaling function $\phi(t)$ and wavelet function $\psi(t)$ are derived from orthogonal polynomials.

Characteristics

- **Smoothness:** Daubechies wavelets are more suited for representing smooth signals.
- **Efficient decomposition:** Daubechies wavelets allow for efficient signal decomposition.

Matlab example

Following code shows a simple case of constructing a Daubechies wavelet.

```
% Daubechies filter
close(gcf)
omega=0:0.1:pi;
[h,g]=wfilters('db2');
n=length(h); % here n=4
% hard coded for easy understanding
% you may use a loop. Note that in matlab index start from 1.
```

```
Hlow=h(1)+h(2)*exp(-1*i*omega)+h(3)*exp(-2*i*omega)+h(4)*exp(-3*i*
    omega);
H_mag=abs(Hlow);
plot(omega,H_mag);
hold on
Hhigh=g(1)+g(2)*exp(-1*i*omega)+g(3)*exp(-2*i*omega)+g(4)*exp(-3*i*
    omega);
H_mag=abs(Hhigh);
plot(omega,H_mag);
```

Problem Statement: Denoising noisy data using Daubechies wavelets

In many real-world applications, signals collected from various sensors or measurement instruments often contain noise, obscuring the underlying patterns. This problem can be mathematically represented as:

$$y(t) = s(t) + n(t)$$

where:

- $y(t)$ is the observed noisy signal,
- $s(t)$ is the true underlying signal that we aim to recover, and
- $n(t)$ represents the noise, typically modeled as a random variable with a mean of zero.

The objective is to develop an effective method for denoising the observed signal $y(t)$ using Daubechies wavelet transforms. Daubechies wavelets provide compact support and can capture both high-frequency components (such as noise) and low-frequency components (such as the underlying signal). By applying wavelet decomposition, we can separate the signal from the noise, enabling us to reconstruct a cleaner version of the true signal $s(t)$.

SOLUTION

Given problem can be formulated with a suitable noisy signal. The denoise it using the filter. Matlab code for this task is given below.

```
% Parameters
fs = 1000;
t = 0:1/fs:1-1/fs;
freq = 5;
noise_level = 0.5;

% Create a clean signal
clean_signal = sin(2 * pi * freq * t);

% Create noisy signal
noisy_signal = clean_signal + noise_level * randn(size(t));

% Choose Daubechies wavelet (e.g., db2)
wavelet_name = 'db2';

% Wavelet decomposition
[coeffs, levels] = wavedec(noisy_signal, 5, wavelet_name);

% Thresholding
% Choose a threshold value (can use different strategies)
threshold = sqrt(2 * log(length(noisy_signal))) * median(abs(coeffs
    )) / 0.6745;

% Apply soft thresholding
coeffs_thresh = wthresh(coeffs, 's', threshold);

% Reconstruct the signal from the modified coefficients
denoised_signal = waverec(coeffs_thresh, levels, wavelet_name);

% Plot results
figure;
subplot(3, 1, 1);
plot(t, clean_signal, 'g', 'LineWidth', 1.5);
title('Clean Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3, 1, 2);
plot(t, noisy_signal, 'r', 'LineWidth', 1.5);
title('Noisy Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3, 1, 3);
plot(t, denoised_signal, 'b', 'LineWidth', 1.5);
title('Denoised Signal using Daubechies Wavelet');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

Output of the code is shown in Figure 1.5.
They are commonly used in image processing and signal compression and denoising.
A Comparison of FIR, Haar, and Daubechies Filters is shown in Table 1.1.

Table 1.1: Comparison of FIR, Haar, and Daubechies Filters

Feature	FIR Filter	Haar Wavelet	Daubechies Wavelet
Type	Linear time-invariant	Wavelet transform	Wavelet transform
Phase Response	Can be linear	Not linear	Not linear
Computational Complexity	Moderate	Very low	Moderate
Smoothing	Not optimal for sharp changes	Good for sharp transitions	Good for smooth signals
Energy Preservation	Depends on design	Yes	Yes
Application	General filtering	Real-time, edge detection	Compression, denoising

1.4.1 Sparsity-based signal processing and L1 norm optimization

Sparsity-based signal processing aims to represent signals with a small number of coefficients, leading to efficient storage and processing. **L1 norm optimization** plays a critical role in sparsity-based approaches, particularly in compressed sensing and signal denoising.

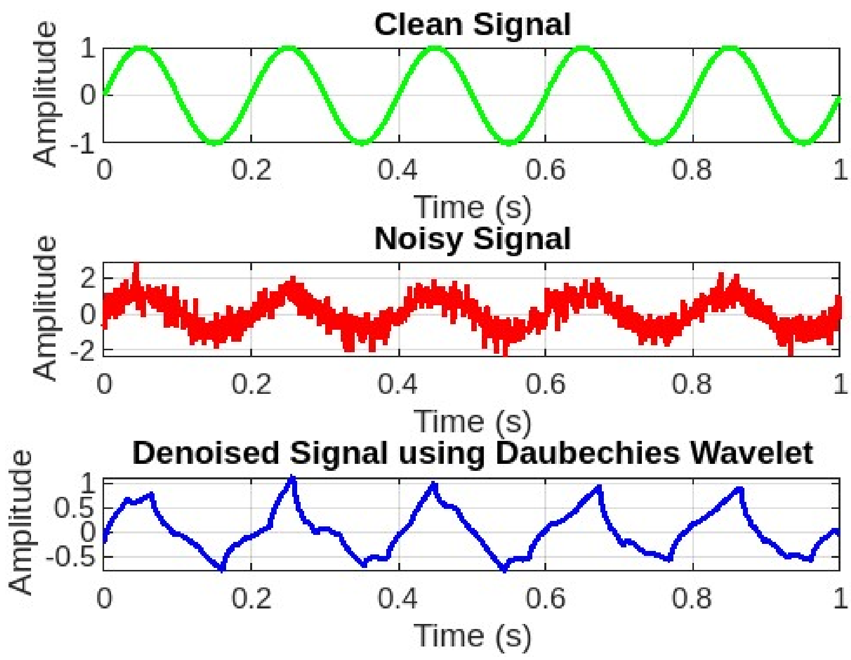


Figure 1.5: Denoising skill of Daubechies Wavelet filter on a noisy sine wave.

1.4.2 Mathematical formulation

Given a signal x and a transformation matrix Ψ, we aim to find a sparse representation α such that:

$$x = \Psi\alpha$$

The sparse vector α is found by solving an L1 norm minimization problem:

$$\min_{\alpha} \|\alpha\|_1 \quad \text{subject to} \quad \|x - \Psi\alpha\|_2 \leq \epsilon$$

Where:

- $\|\alpha\|_1$ is the **L1-norm**, promoting sparsity.
- ϵ is the error tolerance.

Applications in signal processing

- **Compressed Sensing:** L1 optimization is crucial in recovering signals from fewer measurements than typically required.
- **Denoising:** It helps in removing noise while preserving important signal features.
- **Data Science and Machine Learning:** Sparsity-based techniques are used for feature selection and model compression.

RESULTS

1. Basics of signal processing is revisited.
2. Some common filters and their applications are discussed.
3. A simple FIR filter is designed by optimizing the parameters with `CVX` solver.
4. Important characteristics of these filters are:

- FIR filters offer stability and simplicity in general signal processing.
- Haar wavelets are efficient for real-time applications with sharp transitions.
- Daubechies wavelets are better suited for smooth signal analysis and compression.
- Sparsity-based signal processing using L1 optimization opens doors for efficient data representation and has wide applications in modern fields like data science and machine learning.

2 | Assignment 81 Applications of Discrete Fourier Transform

2.1 Introduction

The **Discrete Fourier Transform (DFT)** is a fundamental tool in signal processing that converts a finite sequence of time-domain samples into their frequency-domain representation. DFT helps decompose a discrete signal into its constituent frequency components.
From a **Linear Algebra** point of view, the DFT provides **orthogonal complex exponential bases** with positive and negative integer wave numbers. These wave numbers correspond to the number of full complex sinusoidal waves sampled to create the bases. Each base function is a complex sinusoid representing a specific frequency, and the DFT expresses a signal as a linear combination of these orthogonal basis functions.

2.1.1 Mathematical Representation

For a discrete signal $x[n]$ of length N, the DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi kn/N}, \quad k = 0, 1, 2, \ldots, N-1 \tag{2.1}$$

Where:

- $X[k]$ is the k-th frequency component.
- $x[n]$ is the input signal.
- N is the total number of samples.
- $e^{-j2\pi kn/N}$ is the complex exponential representing rotation in the frequency domain, where k is the wave number.

The **Inverse DFT (IDFT)** is given by:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j2\pi kn/N}, \quad n = 0, 1, 2, \ldots, N-1 \tag{2.2}$$

2.1.2 Linear algebra perspective

From a linear algebra viewpoint:

- The DFT matrix F is an **orthogonal matrix** of complex exponentials (sinusoids). Each row corresponds to a basis vector, which is a complex sinusoid at a specific frequency.

- The orthogonality of these basis functions allows decomposition of a signal into independent frequency components, much like vectors are decomposed into orthogonal bases in Euclidean space.

- The DFT represents the signal as a sum of these orthogonal basis vectors, weighted by their respective coefficients $X[k]$.

2.2 What does the dft really do?

The **Discrete Fourier Transform (DFT)** enables us to transform a signal from the time domain into the frequency domain, where the signal is interpreted in terms of its frequency content. This transformation is accomplished by expressing the signal as a sum of **complex exponential waves**, each corresponding to a different frequency (or wave number).
A complex exponential wave consists of two components:

- A cosine wave: $\cos(\Omega t)$, representing the real part.

- A sine wave: $\sin(\Omega t)$, representing the imaginary part.

Here, Ω is the angular frequency, defined as $\Omega = \frac{2\pi}{T}$, where T is the duration of the signal.

2.2.1 DFT interpretation of a signal

The DFT of a discrete signal allows us to:

1. **Interpret the signal in terms of frequency content**: The DFT decomposes the signal into its constituent frequencies, enabling analysis of how much each frequency contributes to the overall signal.

2. **Perform fast convolution**: By converting signals to the frequency domain, convolution of two sequences (whether 1D, 2D, or nD arrays) can be efficiently performed using the DFT, as convolution in the time domain corresponds to multiplication in the frequency domain.

2.2.2 Example: DFT basis for signal length 8

To better understand how the DFT works, consider a signal of length $N = 8$. The DFT basis matrix consists of N complex exponential waves (or basis vectors), where each vector corresponds to a different frequency or wave number.
The arrangement of these wave numbers in the DFT basis matrix is as follows:

$$[0, 1, 2, 3, \pm 4, -3, -2, -1]$$

These wave numbers correspond to complex exponentials of the form:

$$[e^{0j\Omega t}, e^{1j\Omega t}, e^{2j\Omega t}, e^{3j\Omega t}, e^{\pm 4j\Omega t}, e^{-3j\Omega t}, e^{-2j\Omega t}, e^{-1j\Omega t}]$$

where $\Omega = \frac{2\pi}{T}$ and T is the duration of the signal. The DFT samples these complex exponentials at N equi-spaced points.
The **first basis vector** ($e^{0j\Omega t}$) corresponds to the zero-frequency component, which is simply a vector of ones. This is sometimes called the **DC component**, representing the average value of the signal.
The other basis vectors ($e^{kj\Omega t}$ for $k = 1, 2, 3, \ldots$) correspond to sinusoidal waves with increasing frequency, each representing a different harmonic of the fundamental frequency Ω.

2.2.3 Why is the central basis vector's wave number ± 4?

In the case of an 8-point DFT, the wave numbers range from 0 to $N-1$, but because of the nature of the complex exponentials, the higher frequencies “wrap around" due to the periodicity of the DFT. This explains the symmetry of the wave numbers around zero. Specifically:

- The wave number $+4$ corresponds to a sinusoidal wave that completes exactly four cycles over the duration of the signal.
- The wave number -4 is equivalent to $+4$ because of the periodicity of the DFT. Both represent the same frequency but with opposite phases.

This means that at $k = \pm 4$, the basis vector corresponds to the highest possible frequency that can be represented with a signal of length 8. This is called the **Nyquist frequency**, which is the limit at which any higher frequencies would begin to alias (or overlap) with lower frequencies due to the discrete sampling.

2.2.4 Geometrical insight of DFT basis

Each basis vector in the DFT matrix represents a point on the unit circle in the complex plane. The first basis vector ($k = 0$) represents the center point (constant DC signal), while the other vectors rotate around the unit circle, with the rotation speed determined by the wave number k.
At $k = \pm 4$, the basis vectors rotate the fastest, completing exactly four full cycles over the signal duration. This is why the central basis vector's wave number can be ± 4: it represents the fastest oscillating component, capturing the highest frequency in the signal.
`Matlab` code to do this task is shown below.

```
% Define the signal length
N = 8;
T = 1;  % Duration of the signal (arbitrary units)
t = linspace(0, T, N);  % Time vector with N points

% Create a simple original wave (sum of sine and cosine waves)
f1 = 1;  % Frequency of the first component
f2 = 2;  % Frequency of the second component

% Original wave (sum of a cosine and a sine wave)
original_wave = cos(2*pi*f1*t) + sin(2*pi*f2*t);

% Construct the DFT matrix W_N
n = 0:N-1;
k = n';  % Make k a column vector
W_N = exp(-1j * 2 * pi * k * n / N);  % N x N DFT matrix

% Compute the DFT using matrix multiplication
dft_wave_matrix = W_N * original_wave.';

% Compute frequency axis for plotting the DFT (0 to N-1)
frequencies = (0:N-1)/T;

% Plot the original wave in the time domain
figure;
subplot(2, 1, 1);
stem(t, original_wave, 'filled', 'LineWidth', 1.5);
title('Original Wave in Time Domain');
```

```
xlabel('Time');
ylabel('Amplitude');
grid on;

% Plot the magnitude of the DFT (frequency domain)
subplot(2, 1, 2);
stem(frequencies, abs(dft_wave_matrix), 'filled', 'LineWidth', 1.5)
    ;
title('Magnitude of DFT (Frequency Domain - Matrix Multiplication)'
    );
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
```

Output of this code is shown in Figure 2.1.

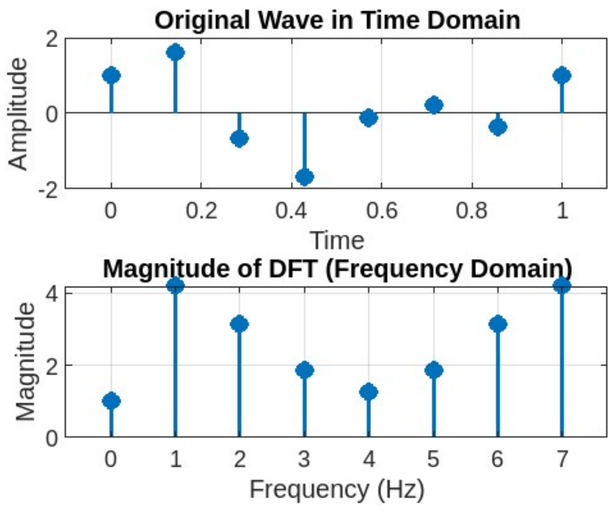


Figure 2.1: DFT of a wave with wave number 8.

2.2.5 Properties of DFT

- **Linearity**: If $x_1[n]$ and $x_2[n]$ have DFTs $X_1[k]$ and $X_2[k]$, then:

$$a\cdot x_1[n]+b\cdot x_2[n]\xrightarrow{\text{DFT}}a\cdot X_1[k]+b\cdot X_2[k]$$

- **Periodicity**: The DFT is periodic with period N. Thus:

$$X[k+N]=X[k]$$

- **Symmetry**: For real-valued signals, the DFT exhibits conjugate symmetry:

$$X[N-k]=\overline{X[k]}$$

- **Circular Convolution**: The DFT of the circular convolution of two signals is the product of their DFTs.

- **Parseval's Theorem**: The total energy in the time domain is equal to the total energy in the frequency domain:

$$\sum_{n=0}^{N-1}|x[n]|^2=\frac{1}{N}\sum_{k=0}^{N-1}|X[k]|^2$$

2.3 DFT Spectrum and FFT

Understanding the DFT spectrum

The Discrete Fourier Transform (DFT) is a method for analyzing the frequency content of a signal by expressing it as a linear combination of sinusoidal basis functions (complex exponentials). The DFT transforms a discrete signal from the time domain into the frequency domain, allowing us to observe how much of each frequency component is present in the signal.
The DFT spectrum refers to the magnitude of the DFT coefficients. When we compute the DFT of a signal, the resulting coefficients are generally complex numbers. These coefficients represent the amplitude and phase of the sinusoidal components that make up the original signal. The magnitude of these coefficients gives us the strength (or amplitude) of each frequency component, while the phase tells us the shift of each component relative to time zero.

Linear combination of DFT basis vectors

In the DFT, we express a signal $x[n]$ as a sum of complex exponentials of different frequencies. These exponentials are the basis vectors of the DFT and are defined as:

$$W_N(k,n) = e^{-j\frac{2\pi kn}{N}}, \quad k = 0,1,2,\dots,N-1$$

Here, N is the number of samples, k is the frequency index, and n is the time index. The DFT formula is written as:

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

The result $X[k]$ is the DFT coefficient at frequency k, and it represents how much of the sinusoidal basis wave $e^{-j\frac{2\pi kn}{N}}$ is present in the signal. These coefficients are complex numbers, and each $X[k]$ contains both amplitude and phase information for that particular frequency component.

Magnitude and phase of DFT coefficients

Each DFT coefficient $X[k]$ is a complex number, and it can be represented as:

$$X[k] = |X[k]|e^{j\theta[k]}$$

where:

- $|X[k]|$ is the **magnitude** of the DFT coefficient, which indicates the strength of the frequency component at k.
- $\theta[k]$ is the **phase** of the DFT coefficient, which indicates the time shift of the corresponding frequency component.

The **DFT spectrum** refers to the plot of $|X[k]|$, the magnitudes of the DFT coefficients, as a function of k. This tells us how much energy or power is present at each frequency.

Symmetry of DFT spectrum

For real-valued signals, the DFT exhibits symmetry. If $x[n]$ is real-valued, the DFT coefficients have the property:

$$X[N-k] = \overline{X[k]}$$

where $\overline{X[k]}$ is the complex conjugate of $X[k]$. This means that the negative frequency components (frequencies beyond $N/2$) are mirror images of the positive frequency components. Therefore, we often only look at the first half of the DFT spectrum, corresponding to positive frequencies.

Why $N = 2^n$?

In many cases, the length of the signal N is chosen to be a power of two, i.e., $N = 2^n$. This is done to allow for the use of the **Fast Fourier Transform (FFT)** algorithm. The FFT is a highly efficient algorithm for computing the DFT, reducing the computational complexity from $O(N^2)$ to $O(N \log N)$. Without FFT, we would need to multiply the signal by the entire DFT matrix, which has a computational complexity of $O(N^2)$. The FFT algorithm exploits the symmetry and periodicity properties of the DFT basis vectors to perform the same calculation in a much faster manner.

Fast Fourier Transform

FFT stands for **Fast Fourier Transform**. It is not a different type of transformation but rather a fast method of calculating the DFT. Specifically, it allows for efficient computation of DFT coefficients, especially when N is a power of two. The FFT takes advantage of the mathematical structure of the DFT and breaks down the problem into smaller, more manageable parts, reducing the number of operations required to compute the DFT.

2.4 Example: DFT of a Simple Signal

Consider a simple example of a discrete signal, such as a sum of two sinusoids:

$$x[n] = \cos(2\pi f_1 n) + \sin(2\pi f_2 n)$$

The DFT will allow us to see the frequency components f_1 and f_2 in the frequency domain. The magnitude of the DFT coefficients $|X[k]|$ will show us how much of each frequency is present in the signal.

2.4.1 MATLAB Code Example

```
% Parameters
N = 8;              % Number of samples
t = 0:N-1;          % Time vector
f1 = 1;             % Frequency of first component (cosine)
f2 = 2;             % Frequency of second component (sine)

% Signal: Sum of a cosine and a sine wave
x = cos(2*pi*f1*t/N) + sin(2*pi*f2*t/N);

% Compute the DFT using FFT
X = fft(x);

% Compute magnitude and phase of DFT coefficients
X_magnitude = abs(X);
X_phase = angle(X);

% Plot the original signal
figure;
subplot(3,1,1);
stem(t, x, 'filled');
title('Original Signal');
xlabel('Sample');
ylabel('Amplitude');

% Plot the magnitude of DFT coefficients
```

```
subplot(3,1,2);
stem(0:N-1, X_magnitude, 'filled');
title('Magnitude of DFT');
xlabel('Frequency Index');
ylabel('Magnitude');

% Plot the phase of DFT coefficients
subplot(3,1,3);
stem(0:N-1, X_phase, 'filled');
title('Phase of DFT');
xlabel('Frequency Index');
ylabel('Phase (radians)');
```

2.4.2 A fast summary of key points

- **DFT Spectrum**: The magnitude of the DFT coefficients represents how much of each frequency is present in the signal.
- **DFT and Basis Vectors**: The DFT expresses the signal as a sum of sinusoidal basis functions, each corresponding to a different frequency.
- **Efficiency with FFT**: By choosing N as a power of two, we can use the FFT algorithm to compute the DFT efficiently, reducing the time complexity significantly.

The DFT and FFT are crucial tools in signal processing, enabling us to analyze signals in the frequency domain, perform fast convolutions, and apply filters. Understanding the DFT spectrum is the first step toward mastering more advanced topics like spectral analysis, filtering, and data compression.

2.5 Applications of DFT

2.5.1 Spectral analysis

The DFT is widely used to analyze the frequency content of discrete signals. By converting a signal from the time domain to the frequency domain, the DFT allows the identification of dominant frequencies present in the signal. This is especially useful in fields like audio processing, communications, and medical imaging.
Example: Analyzing the frequency spectrum of an audio signal to identify specific tones or noise.

2.5.2 Filter design

The DFT is commonly used in designing and analyzing digital filters. The frequency response of a filter can be obtained using the DFT, allowing engineers to inspect how well a filter passes or attenuates certain frequencies.
Example: Designing a low-pass FIR filter using the DFT to minimize the error between desired and actual responses.

2.5.3 Image Processing

In image processing, 2D DFT is used for operations like image filtering, compression, and reconstruction. Applying DFT to images separates frequency components, helping to identify features or remove noise.
Example: JPEG compression utilizes DFT to represent an image in the frequency domain, discarding high-frequency components to reduce file size.

2.5.4 Modulation and demodulation

In communication systems, modulation techniques like OFDM (Orthogonal Frequency Division Multiplexing) rely on the DFT to modulate and demodulate data across multiple frequency channels.
Example: OFDM in Wi-Fi systems handles data transmission across several sub-carriers, increasing robustness against noise.

2.5.5 Signal reconstruction

The **Inverse DFT (IDFT)** reconstructs a signal from its frequency components, which is useful when signals are compressed or transmitted in the frequency domain.
Example: Reconstructing an audio signal after transmission in compressed frequency form.

2.6 Example: FIR Filter Design Using DFT

2.6.1 Problem statement

Design a low-pass FIR filter using the DFT and analyze its performance in the frequency domain.

2.6.2 Steps

1. Define the desired frequency response:

$$H_d(f) = \begin{cases} 1 & \text{for } f \le f_{\text{pass}} \\ 0 & \text{for } f \ge f_{\text{stop}} \end{cases}$$

2. Compute the DFT of the desired response to obtain the filter coefficients.

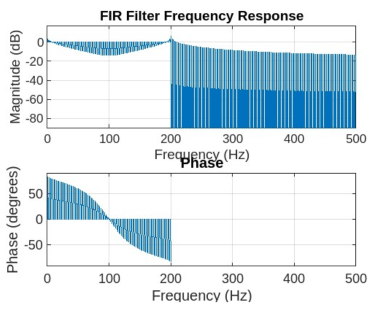
3. Apply the filter to a noisy signal and analyze the results.

`Matlab` code for this task is given below.

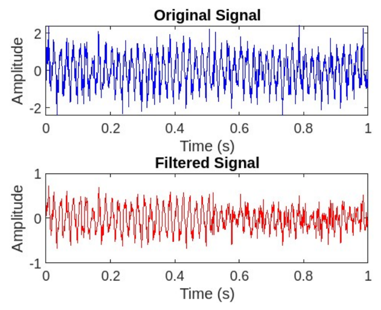
```
% Parameters
fs = 1000;                 % Sampling frequency (Hz)
N = 50;                    % Filter order
f_pass = 100;              % Passband frequency (Hz)
f_stop = 150;              % Stopband frequency (Hz)
num_points = 512;          % Number of frequency points for DFT

% Frequency vector
f = linspace(0, fs/2, num_points);  % Frequency range
H_d = double(f < f_pass);           % Desired frequency response (ideal low-
    pass filter)

% Apply Inverse DFT to get time-domain filter coefficients
h = ifft(H_d);

% Plot frequency response
figure;
freqz(h, 1, num_points, fs);
title('FIR␣Filter␣Frequency␣Response');

% Generate a noisy signal
t = 0:1/fs:1-1/fs;          % Time vector
x = sin(2 * pi * 50 * t) + 0.5 * randn(size(t)); % Signal with noise
```

```
% Filter the signal
y = filter(h, 1, x);

% Plot original and filtered signals
figure;
subplot(2,1,1);
plot(t, x, 'b');
title('Original␣Signal');
xlabel('Time␣(s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, y, 'r');
title('Filtered␣Signal');
xlabel('Time␣(s)');
ylabel('Amplitude');
```

Output of the code is shown in Figure 2.2.



(a) FIR filter action



(b) Filtering with DFT.

Figure 2.2: Demonstration of a DFT filter

2.7 Designing Optimal Filters for Denoising

In the field of signal processing, denoising plays a vital role in recovering true signals obscured by noise. Traditional filtering methods often rely on fixed designs, which can be inadequate in addressing the unique characteristics of varying noise. To tackle this challenge, we propose an innovative approach that utilizes Discrete Fourier Transform (DFT) to design optimal filters tailored specifically for the given noisy signal. By framing the filtering process as an optimization problem, we use tools like the CVX solver to minimize the error between the denoised signal and its true counterpart, ensuring that the resulting filter adapts effectively to the signal's frequency components. This methodology not only enhances denoising capabilities but also sets the stage for further exploration of advanced optimization techniques in signal processing.

2.7.1 Mathematical steps for optimal filter design

To design an optimal filter for denoising using DFT, we follow these key steps:
Given a noisy signal x_{noisy}, we can define an unoptimized filter $H_{\text{non-optimized}}$ as follows:

$$H_{\text{non-optimized}}(f) = \begin{cases} 1 & \text{for } |f| \leq f_{\text{cutoff}} \\ 0 & \text{for } |f| > f_{\text{cutoff}} \end{cases}$$

where f represents the frequency and f_{cutoff} is a predefined cutoff frequency for a low-pass filter.

To create an optimized filter H, we minimize the error in denoising. The optimization problem can be stated as:

$$\text{Minimize} \quad \|H \cdot X_{\text{noisy}} - X_{\text{true}}\|_2$$

where X_{noisy} and X_{true} are the DFTs of the noisy and true signals, respectively.
To ensure the filter remains stable and bounded, we introduce constraints on the filter coefficients:

$$|H(f)| \leq 1 \quad \forall f$$

Using CVX, the optimization problem is formulated as:

```
cvx_begin
    variable H(N) complex  % Frequency domain filter
    minimize( norm(H .* X_noisy - X_true, 2) )  % Objective function
    subject to
        abs(H) <= 1;  % Magnitude constraint
cvx_end
```

Once the optimization is complete, the optimized filter is applied in the frequency domain:

$$X_{\text{denoised}} = H \cdot X_{\text{noisy}}$$

Finally, the denoised signal in the time domain can be obtained using the inverse DFT:

$$x_{\text{denoised}} = \text{ifft}(X_{\text{denoised}})$$

`Matlab` code for this task is given below.

```
% Generate a simple signal (e.g., a sine wave)
n = 0:99;
x_true = sin(2*pi*0.1*n);  % True signal

% Add noise to the signal
noise = 0.3 * randn(size(n));
x_noisy = x_true + noise;
X_noisy = fft(x_noisy);
X_true = fft(x_true);
N = length(X_noisy);
cvx_begin quiet
    variable H(N) complex
    minimize( norm(H .* X_noisy' - X_true', 2) )
    subject to
        abs(H) <= 1;
cvx_end

X_denoised_optimized = H .* X_noisy';
x_denoised_optimized = ifft(X_denoised_optimized);
% Define a cutoff frequency (e.g., low-pass filter)
cutoff = 10;
H_non_optimized = zeros(size(X_noisy));
H_non_optimized(1:cutoff) = 1;  % Keep low frequencies
H_non_optimized(end-cutoff+1:end) = 1;  % Keep symmetric low
    frequencies

X_denoised_non_optimized = H_non_optimized .* X_noisy';
```

```
x_denoised_non_optimized = ifft(X_denoised_non_optimized);
figure;
subplot(3,1,1);
plot(n, x_true, 'g', 'LineWidth', 1.5);
title('True Signal');
xlabel('Sample Index');
ylabel('Amplitude');
grid on;
subplot(3,1,2);
plot(n, x_noisy, 'b', 'LineWidth', 1.5);
title('Noisy Signal');
xlabel('Sample Index');
ylabel('Amplitude');
grid on;
subplot(3,1,3);
hold on;
plot(n, real(x_denoised_non_optimized), 'r--', 'LineWidth', 1.5);
plot(n, real(x_denoised_optimized), 'k-', 'LineWidth', 1.5);
title('Denoised Signals');
xlabel('Sample Index');
ylabel('Amplitude');
legend('Non-Optimized DFT Filter', 'Location', 'northeast');
grid on;
```

Output of the code is shown in Figure 2.3.

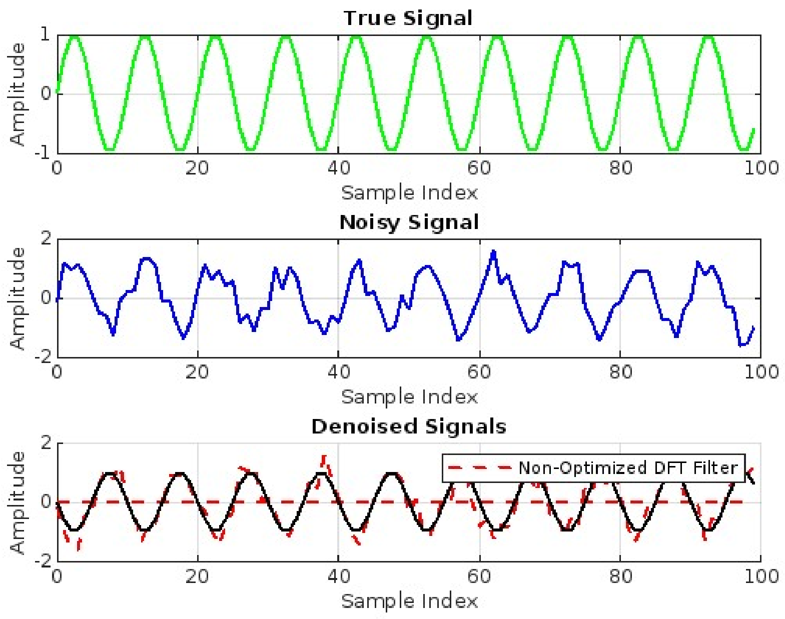


Figure 2.3: Effect of optimized DFT filtering on a noisy image.

2.8 Denoising Signals Without Ground Truth

In many practical scenarios, acquiring the ground truth of a signal can be challenging, particularly in applications such as sensor data processing or communications. This section presents an innovative approach to signal denoising that uses two noisy versions of the same underlying signal, both

contaminated by independent and identically distributed (i.i.d.) noise. By assuming that the noise characteristics are similar across both signals, we can utilize statistical properties to optimize the denoising process. This method demands the need for a reference signal (even a noisy one!), relying instead on the inherent correlations between the noisy inputs to infer a cleaner approximation of the true signal.
The optimization is formulated as a minimization problem, wherein we seek to reduce the error between the denoised output and the expected true signal. By employing techniques such as convex optimization, we can derive a frequency-domain filter that efficiently combines the information from both noisy signals, enhancing the overall denoising effectiveness. This approach demonstrates that it is possible to achieve substantial noise reduction and recover a more accurate representation of the original signal, solely based on statistical properties derived from the available noisy observations. The results underscore the potential of this method as a robust alternative for signal processing tasks where ground truth information is not readily accessible.

2.8.1 Mathematical formulation for denoising

Let $x(t)$ be the true signal we aim to recover, and let $n_1(t)$ and $n_2(t)$ be two independent noise components that contaminate the true signal, resulting in two observed signals:

$$y_1(t) = x(t) + n_1(t)$$

$$y_2(t) = x(t) + n_2(t)$$

Assuming $n_1(t)$ and $n_2(t)$ are i.i.d. noise with zero mean and variance σ^2, our objective is to find a denoised signal $\hat{x}(t)$ that minimizes the expected squared error:

$$\hat{x}(t) = \arg\min_{x(t)} \mathbb{E}\left[(x(t) - \hat{x}(t))^2\right]$$

To achieve this, we define a frequency-domain filter $H(f)$ and optimize it based on the observed noisy signals. The optimization problem can be formulated as:

$$\hat{H} = \arg\min_{H} \|H \cdot Y - X\|_2^2$$

where Y is the DFT of the noisy signals $y_1(t)$ and $y_2(t)$, and X represents the DFT of the true signal $x(t)$. The filter is subject to constraints such as:

$$\left|H(f)\right| \leq 1$$

Finally, the denoised signal is obtained by applying the optimized filter in the frequency domain:

$$\hat{Y} = \hat{H} \cdot Y$$

and transforming back to the time domain:

$$\hat{x}(t) = \text{IFFT}(\hat{Y})$$

`Matlab` code for this task is given below.

```matlab
n = 0:99; % Sample index
x_true = sin(2*pi*0.1*n);
noise1 = 0.3 * randn(size(n));
noise2 = 0.3 * randn(size(n));
x_noisy1 = x_true + noise1;
x_noisy2 = x_true + noise2;
N = length(x_noisy1);
W = exp(-2*pi*1i*(0:N-1)'*(0:N-1)/N) / sqrt(N);
X_noisy1 = W * x_noisy1';
```

```
X_noisy2 = W * x_noisy2
cvx_begin quiet
    variable H(N) complex
    minimize( norm(H .* X_noisy1 - W * x_true', 2) + ...
              norm(H .* X_noisy2 - W * x_true', 2) )
    subject to
        % Ensure filter is bounded in magnitude
        abs(H) <= 1;
cvx_end

X_denoised = H .* X_noisy1;
x_denoised = real(W' * X_denoised);
figure;

subplot(3,1,1);
plot(n, x_noisy1, 'b', 'LineWidth', 1.5);
title('Noisy Signal 1');
xlabel('Sample Index');
ylabel('Amplitude');
grid on;

subplot(3,1,2);
plot(n, x_noisy2, 'r', 'LineWidth', 1.5);
title('Noisy Signal 2');
xlabel('Sample Index');
ylabel('Amplitude');
grid on;

subplot(3,1,3);
plot(n, x_true, 'g', 'LineWidth', 1.5);
hold on;
plot(n, x_denoised, 'k--', 'LineWidth', 1.5);
title('True Signal and Denoised Signal');
xlabel('Sample Index');
ylabel('Amplitude');
legend('True Signal', 'Denoised Signal', 'Location', 'northeast');
grid on;
```

Output of this code is shown in Figure 2.4.

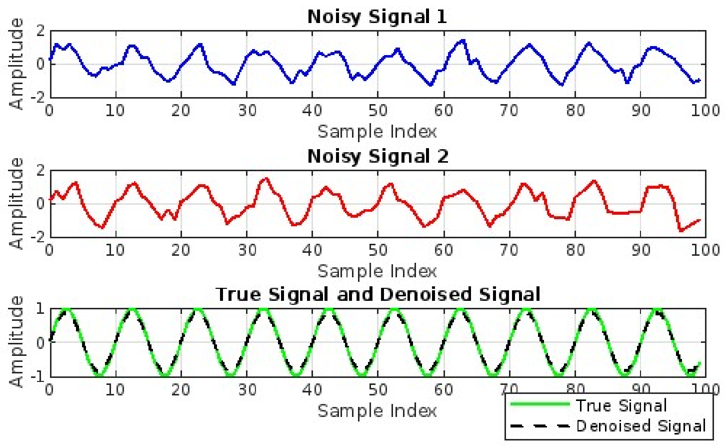


Figure 2.4: Optimized DFT that denoise a signal without ground truth.

2.9 Comparison with Other Transform Techniques

2.9.1 DFT vs. Fast Fourier Transform (FFT)

DFT is the theoretical foundation, while **FFT** is a more efficient algorithm for computing the DFT. The FFT reduces computational complexity from $O(N^2)$ to $O(N \log N)$, making it more practical for large datasets.

2.9.2 DFT vs. Wavelet Transform

While the DFT captures frequency content, the **Wavelet Transform** captures both time and frequency content, making it useful for analyzing non-stationary signals.

2.10 Tasks

1. Form the frequency spectrum of a triangular signal.

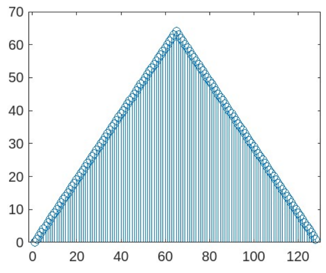
SOLUTION

`Matlab` code for this task is given below.

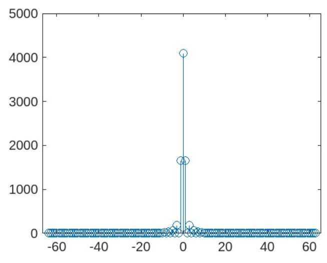
```
close(gcf)
x=[0:64 63:-1:1]'; % signal value increases from 0 to 64 and
    then decreases to 1
% note that signal length is 128
figure
stem(x);
X=fft(x); % find DFT coefficients
X=abs(fftshift(X));
% above line of code make the spectrum centred at origin and
    then find magnitude
ind=[-64:1:0 1:63]; % wave numbers( or frequency) for plotting
    the shifted spectrum
```

```
% ind is used for marking the x-axis. Otherwise it will show 1
    to 128 along x-axis
figure
stem(ind,X);
```

Output of the code is shown in Figure 2.5.



(a) Triangular Wave



(b) Frequency Plots.

Figure 2.5: Frequency plot of the triangular wave

`Matlab` code for this task is given below.

```
close(gcf)
x=[0:64 63:-1:1]'; % signal value increases from 0 to 64 and
    then decreases to 1
% note that signal length is 128
figure
stem(x);
X=fft(x); % find DFT coefficients
X=abs(fftshift(X));
% above line of code make the spectrum centred at origin and
    then find magnitude
ind=[-64:1:0 1:63]; % wave numbers( or frequency) for plotting
    the shifted spectrum
% ind is used for marking the x-axis. Otherwise it will show 1
    to 128 along x-axis
figure
stem(ind,X);
```

2. Convolve sequences 1, 1 with 1 2 1 , the out put is the sequence 1 3 3 1.

SOLUTION

If we linearly convolve two sequences of length m and n, the resulting sequence will have a length of $m+n-1$. FFT and inverse FFT can be employed for convolving two sequences.

If we convolve sequences 1, 1 with 1 2 1 , the out put is the sequence 1 3 3 1 , a sequence of length of 4. To obtain the result using FFT , convert the two input sequence to the same length of the output sequence by appending zeros. So $x = [1100]'$, $y = [1210]'$. The following Matlab code will demonstrate the process.

```
% convolution using fft
x=[ 1 1 0 0]';
y=[1 2 1 0]';
z1=ifft(fft(x).*fft(y));
z2=conv([1 1]',[1 2 1]');
disp('Convolution using FFT:');
disp(z1);
disp('Convolution using built-in function:');
disp(z2);
if z1==z2
    disp('Both the approaches gives the same result:')
else
    disp('Results are different:');
end
```

Output of the code is shown below:

```
Convolution using FFT:
     1
     3
     3
     1
Convolution using built-in function:
     1
     3
     3
     1
Both the approaches gives the same result:
```

3. Convolve sequence $\{1,1\}$ with itself fifteen times to get a sequence of 16 numbers using FFT. Do not use any loop in the program.

SOLUTION

The convolution theorem states that the Discrete Fourier Transform (DFT) of the convolution of two sequences is the element-wise product of their DFTs:

$$\mathscr{DFT}\{x[n]*h[n]\} = \mathscr{DFT}\{x[n]\}\cdot\mathscr{DFT}\{h[n]\}$$

where $*$ denotes convolution, and $\cdot$ represents pointwise multiplication in the frequency domain.

In this problem, we convolve the sequence $\{1,1\}$ with itself 15 times. Instead of performing multiple convolutions in the time domain, we multiply the DFT of the sequence in the frequency domain. Let $X[k]$ be the DFT of the zero-padded sequence $x[n]=\{1,1\}$. Then, the DFT of the 15-fold convolution is:

$$X[k]^{15} = (\mathscr{DFT}\{x[n]\})^{15}$$

Thus, the result in the frequency domain is:

$$\mathscr{DFT}\{y[n]\} = X[k]^{15}$$

To ensure linear convolution using the DFT (which performs circular convolution by default), we apply zero padding. If two sequences of length m and n are convolved, the result has length $m+n-1$.

In this case, the sequence $\{1,1\}$ has length 2, and convolving it 15 times results in a sequence of length 16. To achieve this result, we zero-pad the sequence to a length of 16 before applying the FFT.

$$x[n] = \{1,1\}$$

Zero-padded to length 16:

$$x_{\text{padded}}[n] = \{1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$$

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the DFT with complexity $O(N \log N)$. In this problem, we compute the FFT of the zero-padded sequence:

$$X[k] = \mathscr{FFT}\{x_{\text{padded}}[n]\}$$

Now to convolve the sequence $\{1, 1\}$ with itself 15 times, we use the convolution theorem and raise the DFT of the sequence to the power of 15:

$$Y[k] = X[k]^{15}$$

Finally, the inverse FFT (IFFT) transforms the frequency-domain result back into the time domain:

$$y[n] = \mathscr{IFFT}\{Y[k]\}$$

`Matlab` code for this entire task is given below.

```matlab
x = [1, 1];
n_conv = 15;
N = 2^nextpow2(2 * n_conv);
x_padded = [x, zeros(1, N - length(x
X_fft = fft(x_padded);
X_fft_n_conv = X_fft.^n_conv;
y = ifft(X_fft_n_conv);
y_final = real(y(1:16));
disp('Resulting sequence after 15 convolutions:');
disp(y_final);
```

Output of the code is shown below.

```
Resulting sequence after 15 convolutions:
   1.0e+03 *

0.0010    0.1050
0.4550   1.3650
3.0030    5.0050
6.4350    6.4350
5.0050   3.0030
1.3650    0.4550
0.1050    0.0150
0.0010    0.0150
```

RESULTS

1. Mathematical foundations and properties of DFT is revisited.
2. Skill of FIR and DFT filters are tested.
3. Develop an fine tuned filter by optimizing the filtering parameters with `CVX` solver.
4. A novel approach to denoise a signal without ground truth is discussed and successfully implemented.
5. Various type of filters in signal processing are revisited and compare their characteristics.

3 | Assignment 82
Fibonacci Sequence and Its Applications in Optimization

3.1 Introduction

The Fibonacci sequence is a famous sequence defined as follows:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

The first few terms of the Fibonacci Sequence are $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$. This Sequence has numerous applications in optimization, particularly in algorithms and data structures, such as dynamic programming and greedy algorithms.

3.1.1 Generating the Fibonacci sequence

To generate the Fibonacci Sequence, we can use a simple iterative method or a recursive approach. Here is a concise algorithm using an iterative approach:

```
function fibonacci(n)
    F = zeros(1, n);
    F(1) = 0;
    F(2) = 1;
    for i = 3:n
        F(i) = F(i-1) + F(i-2);
    end
    return F;
end
```

3.1.2 Golden section ratio

The golden section ratio, often denoted by ϕ, can be derived from the Fibonacci Sequence as:

$$\phi = \lim_{n\to\infty} \frac{F(n)}{F(n-1)} \approx 1.6180339887$$

To find the golden section ratio using the shifting property in finite differences, we can express the ratio of consecutive Fibonacci numbers as:

$$\phi_n = \frac{F(n)}{F(n-1)}$$

As n increases, ϕ_n converges to ϕ.

Using the result from the finite difference;

$$\begin{aligned} &F_{n+2} = F_{n+1} + F_n \\ &\implies E^2F_n - EF_n - F_n = 0 \\ &\implies \left(E^2 - E - 1\right)F_n = 0 \end{aligned}$$

where, E is the shift operator defined by $E(y_n) = y_{n+1}$. The auxiliary equation is given by

$$\lambda^2 - \lambda - 1 = 0$$

Solving this quadratic equation, we get

$$\begin{bmatrix}\lambda_1\\ \lambda_2\end{bmatrix} = \begin{bmatrix}\dfrac{1+\sqrt{5}}{2}\\ \dfrac{1-\sqrt{5}}{2}\end{bmatrix} = \begin{bmatrix}\phi_1\\ \phi_2\end{bmatrix}$$

3.1.3 Matrix representation of the Fibonacci sequence

The Fibonacci numbers can be generated using matrix multiplication. The relation can be represented as follows:

$$\begin{bmatrix}F(n)\\ F(n-1)\end{bmatrix} = \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix} \cdot \begin{bmatrix}F(n-1)\\ F(n-2)\end{bmatrix}$$

Defining the transformation matrix A:

$$A = \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$$

We can express the n-th Fibonacci number using matrix exponentiation:

$$\begin{bmatrix}F(n)\\ F(n-1)\end{bmatrix} = A^{n-1} \cdot \begin{bmatrix}F(1)\\ F(0)\end{bmatrix} = A^{n-1} \cdot \begin{bmatrix}1\\ 0\end{bmatrix}$$

3.1.4 Similarity matrix representation

To form a similarity matrix representation, we first need to find the eigenvalues and eigenvectors of the transformation matrix A. The eigenvalues λ of A can be found by solving the characteristic polynomial:

$$\det(A - \lambda I) = 0$$

Calculating the determinant, we get:

$$\det\begin{bmatrix}1-\lambda & 1\\ 1 & -\lambda\end{bmatrix} = (-\lambda)(1-\lambda) - 1 = \lambda^2 - \lambda - 1 = 0$$

The eigenvalues are:

$$\lambda_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

The connection between the two eigenvalues ϕ_1 and ϕ_2 of the Fibonacci transformation matrix A can be understood through their definitions and relationships in the context of the Fibonacci sequence.

3.1.5 Eigenvalues of the Fibonacci matrix

Given the transformation matrix:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

the characteristic polynomial is derived from the determinant:

$$\det(A - \lambda I) = 0$$

This yields the polynomial:

$$\lambda^2 - \lambda - 1 = 0$$

Solving this quadratic equation, we find the eigenvalues:

$$\phi_1 = \frac{1+\sqrt{5}}{2} \quad \text{(the golden ratio)}$$

$$\phi_2 = \frac{1-\sqrt{5}}{2}$$

3.1.6 Relationship between ϕ_1 and ϕ_2

Sum and Product of Eigenvalues: The eigenvalues have a well-defined relationship given by Vieta's formulas: - The sum of the eigenvalues:

$$\phi_1 + \phi_2 = 1$$

- The product of the eigenvalues:

$$\phi_1 \cdot \phi_2 = -1$$

Reciprocal Relationship: The second eigenvalue ϕ_2 is negative and less than zero:

$$\phi_2 = -\frac{1}{\phi_1}$$

Decay of Contributions: In the context of the Fibonacci sequence, while ϕ_1 dominates the growth of the Fibonacci numbers due to its positive nature, ϕ_2 contributes significantly less as n increases. Specifically, because $|\phi_2| < 1$, its contributions diminish to zero in Binet's formula as n becomes large:

$$F_n = \frac{1}{\sqrt{5}}\left(\phi_1^n - \phi_2^n\right) \approx \frac{1}{\sqrt{5}}\phi_1^n \quad \text{for large } n$$

We will prove this at the end of this discussion using Eigen decomposition.
Next, we find the corresponding eigenvectors by solving $(A - \lambda I)v = 0$. For $\lambda_1 = \phi$:

$$(A - \phi I)v = \begin{bmatrix} 1-\phi & 1 \\ 1 & -\phi \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Solving this system gives the eigenvector corresponding to λ_1:

$$v_1 = \begin{bmatrix} 1 \\ \phi - 1 \end{bmatrix}$$

For $\lambda_2 = -\phi^{-1}$:

$$(A + \phi^{-1} I)v = \begin{bmatrix} 1+\phi^{-1} & 1 \\ 1 & -\phi^{-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This gives us another eigenvector.
The transformation matrix can be expressed in terms of its eigenvalues and eigenvectors as:

$$A = PDP^{-1}$$

where P is the matrix of eigenvectors and D is the diagonal matrix of eigenvalues.

3.1.7 Binet's formula

Using the properties of the golden ratio and the similarity matrix representation, we can express the n-th Fibonacci number as:

$$F_n = P \cdot D^{n-1} \cdot P^{-1} \cdot \begin{bmatrix} F(1) \\ F(0) \end{bmatrix}$$

From this, we can derive:

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

As n becomes large, the second term becomes negligible, leading us to Binet's formula:

$$F_n = \text{nearest integer} \left(\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n \right)$$

This formula is derived from the eigenvalue-based matrix representation and reflects the growth rate of Fibonacci numbers in terms of the golden ratio.
The Fibonacci Sequence not only showcases an intriguing numerical pattern but also provides insights and tools for optimization in various fields. Its applications span areas such as algorithm analysis, computational mathematics, and even financial modeling.

3.2 Applications of Golden Section Ratio in Optimization, Signal Processing, and Machine Learning

The Golden Section Ratio, denoted as ϕ and approximately equal to 1.618, is instrumental in optimization techniques across various fields, including engineering, signal processing, and machine learning. The ratio is derived from the equation:

$$\phi = \frac{1+\sqrt{5}}{2}$$

This ratio allows a given interval to be divided in such a way that the ratio of the smaller segment to the larger segment is the same as the ratio of the larger segment to the whole. This property facilitates efficient search methods, making it an invaluable tool in optimization problems.

3.2.1 Optimization in signal processing

In signal processing, the Golden Section Ratio is utilized in filter design and parameter optimization. Consider a digital filter where the goal is to minimize the error between the desired filter response $H_d(\omega)$ and the actual response $H_a(\omega)$. The cost function can be defined as:

$$E(f) = \int |H_d(\omega) - H_a(\omega)|^2 d\omega$$

To find the optimal cutoff frequency f^*, the Golden Section Search method can be applied:

1. Define the interval $[a, b]$ for f.

2. Calculate points using the Golden Section Ratio:

$$c = b - \frac{b-a}{\phi},$$
$$d = a + \frac{b-a}{\phi}.$$

3. Evaluate $E(c)$ and $E(d)$.

4. Narrow the interval based on which point yields a lower error.

Here's a `matlab` code snippet that demonstrates the Golden Section Search method to find the optimal cutoff frequency for a hypothetical digital filter:

```
function [optimal_freq, min_error] = golden_section_search(func, a, b, tol)
    % Golden Section Search to minimize a function 'func' in interval [a, b]

    phi = (1 + sqrt(5)) / 2;  % Golden ratio
    resphi = 2 - phi;  % Inverse of the golden ratio

    % Initialize points
    c = b - resphi * (b - a);
    d = a + resphi * (b - a);

    while abs(b - a) > tol
        if func(c) < func(d)
            b = d;  % Move the upper bound down
        else
            a = c;  % Move the lower bound up
        end
        c = b - resphi * (b - a);  % Recalculate c
        d = a + resphi * (b - a);  % Recalculate d
    end

    optimal_freq = (a + b) / 2;  % Midpoint gives optimal frequency
    min_error = func(optimal_freq);  % Evaluate minimum error
end

cost_function = @(f) (f - 1)^2 + 0.1 * randn;  % Sample cost function with noise
[a, b] = deal(0, 2);  % Interval for search
tolerance = 1e-5;  % Tolerance for convergence

[optimal_freq, min_error] = golden_section_search(cost_function, a, b, tolerance);
fprintf('Optimal Cutoff Frequency: %f\nMinimum Error: %f\n', optimal_freq, min_error);
```

3.2.2 Optimization in Machine Learning

In the context of machine learning, the Golden Section Ratio is also useful for hyperparameter tuning, which is essential for optimizing model performance. Many algorithms require careful selection of hyperparameters, such as learning rates or regularization strengths. The Golden Section Search method can be employed to minimize validation error.
For example, given a model's validation error as a function of hyperparameter λ:

$$E(\lambda) = \text{ValidationError}(\lambda)$$

The same Golden Section Search technique can be applied to find the optimal λ^* that minimizes $E(\lambda)$.
The Golden Section Ratio serves as a powerful optimization tool in signal processing and machine learning, enabling efficient methods that enhance performance while reducing computational complexity. Its applications demonstrate the versatility and effectiveness of this mathematical concept in solving practical engineering problems.

RESULTS

1. The Fibonacci sequence is represented as linear system with the transfer matrix notation.
2. Golden section ratio is viewed as the solution of Auxiliary equation of the Shift operator expression in finite difference and the eigen values of the transfer polynomial.
3. Using similarity matrix approach, general formula to generate Fibonacci sequence $\{F_n\}$ for large n.
4. Some applications of the golden ratio in machine learning and signal processing are discussed.

4 | Assignment 83 Computing Derivative Using FFT

4.1 Introduction

This session discusses the computation of the derivative of a continuous function using Fast Fourier Transform (FFT). The approach leverages the properties of FFT to obtain a more efficient solution compared to traditional numerical differentiation methods.
Length of the Range of Function Considered is $L = 20$ and the number of samples taken is $n = 128$.
The fundamental frequency is defined such that the first base of the Discrete Fourier Transform (DFT) covers one full wave over the length L.

4.1.1 Sampling range

The sample points are taken from the time domain range of $[-10, 10]$ (length $L = 20$).
The maximum positive frequency is 63 and the minimum negative frequency is $-64 \cdot f$.

4.1.2 Angular frequency values

The angular frequencies are given by:

$$\omega = \frac{2\pi}{L}[0, 1, 2, \ldots, 63, -64, -63, \ldots, -1]$$

This setup ensures the reconstruction of the signal if the function is slow-varying.

4.2 Finding the Derivative of sech(x) Using FFT

The function domain is taken as $[-10, 10]$ with $L = 20$. The hyperbolic secant function is defined as:

$$y = \operatorname{sech}(x) = \frac{2}{e^x + e^{-x}}$$

The derivative of y is given by:

$$\frac{dy}{dx} = -\operatorname{sech}(x) \cdot \tanh(x)$$

Matlab code for this task is given below.

```
% Define the range and sample points
L = 20;
n = 128;
x = linspace(-10, 10, n);
y = sech(x); % Original function
yd = -sech(x) .* tanh(x); % True derivative

% Compute FFT of the function
```

```
yhat = fft(y);

% Calculate angular frequencies
w = (2 * pi / L) * [0:(n/2 - 1), -(n/2):(-1)];

% Compute the derivative in frequency domain
ydhat = 1i * w .* yhat; % Multiplying by i * w

% Inverse FFT to get the computed derivative
ydcomputed = ifft(ydhat);

% Compare the true derivative and computed derivative
figure;
subplot(3, 1, 1);
plot(x, y, 'b', 'LineWidth', 1.5);
title('sech(x)');
xlabel('x');
ylabel('Amplitude');
grid on;

subplot(3, 1, 2);
plot(x, yd, 'r', 'LineWidth', 1.5);
title('True Derivative');
xlabel('x');
ylabel('Amplitude');
grid on;

subplot(3, 1, 3);
plot(x, real(ydcomputed), 'k', 'LineWidth', 1.5);
title('Computed Derivative Using FFT');
xlabel('x');
ylabel('Amplitude');
grid on;

legend('sech(x)', 'True Derivative', 'Computed Derivative', '
    Location', 'northeast');
```

4.3 Tasks

1. Compute linear convolution of sequence {1,1} 100 times using FFT

   Mathematical Formulation: The linear convolution of a sequence $x[n]$ with itself can be expressed as:

$$y[n] = x[n] * x[n]$$

   `Matlab` code for this task is given below.

```
% Define the sequence
seq = [1, 1];
N = 100; % Number of repetitions
convolved_seq = conv(seq, ones(1, N));

% Display the result
```

```
figure;
stem(convolved_seq);
title('Convolution of Sequence {1,1} Repeated 100 Times');
xlabel('Sample Index');
ylabel('Amplitude');
```

Output of the code is shown in Figure 4.1.

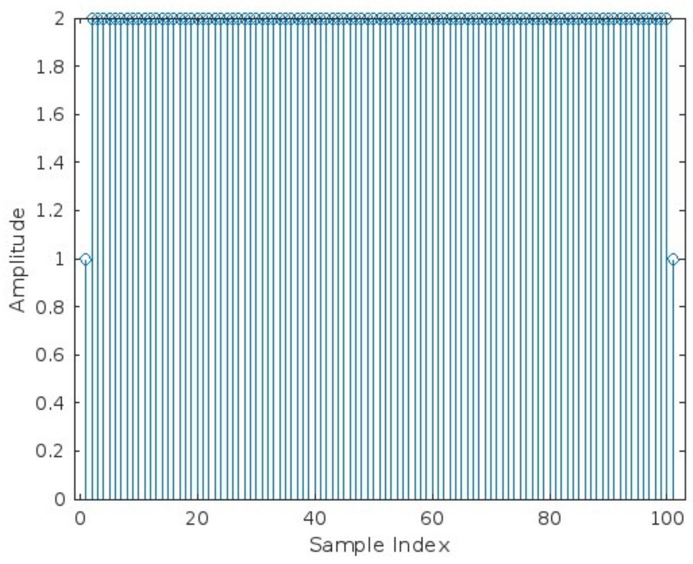


Figure 4.1: Linear convolution of sequence using FFT.

4.3.1 A die is thrown 100 times: compute probabilities of getting all possible sums (100 to 600) using FFT

Mathematical Formulation: The characteristic polynomial for a single die throw can be expressed as:

$$P(z) = \left(\frac{1}{6}z^1 + \frac{1}{6}z^2 + \frac{1}{6}z^3 + \frac{1}{6}z^4 + \frac{1}{6}z^5 + \frac{1}{6}z^6\right)$$

Matlab code for this task is given below.

```
% Probability distribution for a single die
p_die = [1/6, 1/6, 1/6, 1/6, 1/6, 1/6]; % Probability of each
    face (1-6)
N = 100; % Number of throws

% Compute the characteristic polynomial of the die
char_poly = p_die;
for i = 2:N
    char_poly = conv(char_poly, p_die);
end

% Get the probabilities for sums from 100 to 600
sums = 100:length(char_poly) + 99; % Adjust for the number of
    throws
probabilities = char_poly; % Probabilities for sums from 100 to
     600
```

```

% Plot the results
figure;
bar(sums, probabilities);
title('Probabilities of Sums from 100 to 600');
xlabel('Sum');
ylabel('Probability');
```

Output of the code is shown in Figure 4.2.

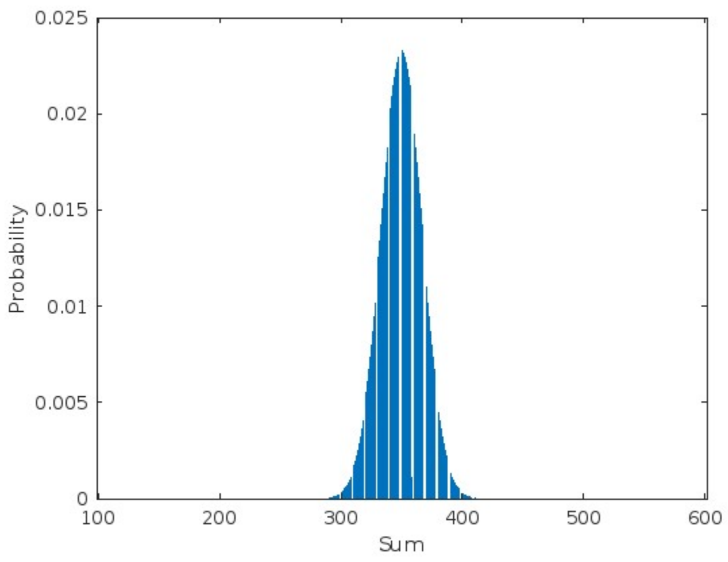


Figure 4.2: Probabilities for getting sum from 100 to 600.

4.3.2 A Coin is tossed 100 times: find the probability of getting more than 70 heads using FFT

Mathematical Formulation: The probability generating function for a single coin toss can be represented as:

$$P(z) = 0.5z + 0.5z^0$$

`Matlab` for this task is given below.

```
% Probability distribution for a single coin toss
p_coin = [0.5, 0.5]; % Probability of heads and tails
N = 100; % Number of tosses

% Compute the characteristic polynomial of the coin toss
char_poly = p_coin;
for i = 2:N
    char_poly = conv(char_poly, p_coin);
end

% Find the probabilities for getting more than 70 heads
prob_more_than_70 = sum(char_poly(71:end)); % Sum probabilities
    from 71 to 100

```

```
disp(['Probability of getting more than 70 heads: ', num2str(
    prob_more_than_70)]);
```

Output of the code is shown below.

```
Probability of getting more than 70 heads: 3.9251e-05
```

4.3.3 Using FFT show that the magnitude of ft of Gaussian function is again Gaussian

Mathematical Formulation: The Fourier transform of a Gaussian function retains its form:

$$\mathscr{F}\{e^{-\frac{x^2}{2\sigma^2}}\} = \sigma\sqrt{2\pi}e^{-\frac{\sigma^2\omega^2}{2}}$$

`Matlab` code for this task is given below.

```
% Define parameters for the Gaussian function
mu = 0; % Mean
sigma = 1; % Standard deviation
x = linspace(-10, 10, 1024);
gaussian = (1/(sigma*sqrt(2*pi))) * exp(-0.5 * ((x - mu)/sigma)
    .^2);

% Compute the FFT of the Gaussian
fft_gaussian = fft(gaussian);
magnitude = abs(fft_gaussian);

% Plot the results
figure;
subplot(2, 1, 1);
plot(x, gaussian);
title('Gaussian Function');
xlabel('x');
ylabel('Amplitude');

subplot(2, 1, 2);
freq = linspace(-512, 511, 1024); % Frequency axis
plot(freq, magnitude);
title('Magnitude of FFT of Gaussian Function');
xlabel('Frequency');
ylabel('Magnitude');
```

Output of the code is shown in Figure 4.3.

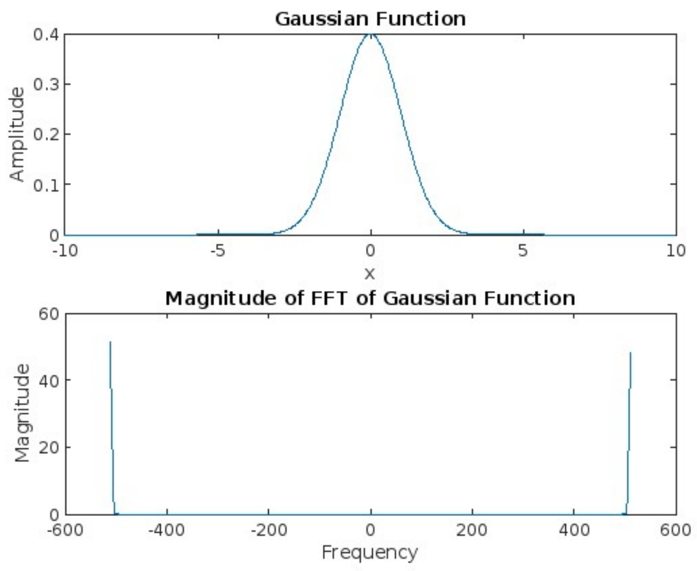


Figure 4.3: Magnitude of Gaussian distribution using FFT.

4.3.4 A Die is thrown 100 times: compute probability of getting the sum on the die as more than 373

`Matlab` code for this task is given below.

```matlab
% Reuse the characteristic polynomial from the earlier problem
% Compute the probabilities for sums from 100 to 600 (already computed)
prob_more_than_373 = sum(char_poly(374:end)); % Sum probabilities from 374 to 600

disp(['Probability of getting a sum more than 373: ', num2str(prob_more_than_373)]);
```

Output of the code is

```
Probability of getting a sum more than 373: 0
```

2. Explain how to convert product of two 3-digit numbers into a convolution followed by a sum of sequences

Given two numbers $a = 123$ and $b = 456$:

$$\text{product}(a, b) = a \cdot b = \sum_{k=0}^{m+n-1} c_k \cdot 10^k$$

Using convolution:

$$c_k = (a * b)[k]$$

To multiply the two numbers 123 and 456 using convolution, we will mimic the traditional place value multiplication process.

(a) Define the Numbers: Represent 123 and 456 as digit sequences:

$$\text{num1} = [1, 2, 3] \quad \text{(representing 123)}$$

$$\text{num2} = [4, 5, 6] \quad \text{(representing 456)}$$

(b) Reverse the Sequences: For convolution to align with traditional multiplication (starting from the rightmost digit):

$$\text{num1} = [3, 2, 1] \quad \text{(reverse of 123)}$$

$$\text{num2} = [6, 5, 4] \quad \text{(reverse of 456)}$$

(c) Perform Convolution:

Using convolution:

$$\text{convolution result} = [3, 2, 1] * [6, 5, 4]$$

The convolution yields:
- Position 0: $3 \times 6 = 18$
- Position 1: $3 \times 5 + 2 \times 6 = 15 + 12 = 27$
- Position 2: $3 \times 4 + 2 \times 5 + 1 \times 6 = 12 + 10 + 6 = 28$
- Position 3: $2 \times 4 + 1 \times 5 = 8 + 5 = 13$
- Position 4: $1 \times 4 = 4$

Thus, the convolution result (without carry-overs) is:

$$\text{product\_digits} = [18, 27, 28, 13, 4]$$

(d) Apply Carry-Over Operations:

- We need to add the carry-over values from each step:

  - Position 0: 18 (write down 8, carry over 1)
  - Position 1: $27 + 1 = 28$ (write down 8, carry over 2)
  - Position 2: $28 + 2 = 30$ (write down 0, carry over 3)
  - Position 3: $13 + 3 = 16$ (write down 6, carry over 1)
  - Position 4: $4 + 1 = 5$ (write down 5)

(e) Final Result:

- Collecting the digits gives us:

$$123 \times 456 = 56088$$

`Matlab` code for this task is given below.

```
% Define the two numbers as sequences
num1 = [1, 2, 3]; % 123
num2 = [4, 5, 6]; % 456

% Reverse the digit sequences for convolution
num1 = fliplr(num1);
num2 = fliplr(num2);

% Perform convolution to get the product
product = conv(num1, num2);

% Handle carry and convert to the correct format
% Initialize the output vector with zeros
result = zeros(1, length(product));

% Add each value in the product to the result
```

```
for i = 1:length(product)
result(i) = product(i);
end

% Carrying over
for i = 1:length(result)-1
if result(i) >= 10
result(i+1) = result(i+1) + floor(result(i) / 10); % Carry to
    next digit
result(i) = mod(result(i), 10); % Keep current digit
end
end

% Display the result
disp(['Product of 123 and 456 is: ', num2str(fliplr(result))]);
```

Output of the code is shown below.

```
Product of 123 and 456 is: 5   6   0   8   8
```

3. Use the 2D Fourier transform for low-pass approximation and interpolation on a given image. The 2D Fourier transform can be used to perform low-pass approximation and interpolation through zero padding. The mathematical formulation for the 2D Fourier transform is:

$$F(u,v) = \iint f(x,y)e^{-2\pi i(ux+vy)}\,dx\,dy$$

The low-pass filtering can be implemented using a Gaussian filter.

Matlab code for this task is given below.

```
% Load an image
name = 'lena'; % Change this to your image path
M = imread(name); % Load the image
M = rgb2gray(M); % Convert to grayscale
M = im2double(M); % Convert to double precision

% Compute the 2D FFT
fft_M = fft2(M);
fft_M_shifted = fftshift(fft_M); % Shift the zero frequency to
    the center

% Create a low-pass filter
[n, m] = size(M);
d0 = 30; % Cut-off frequency
[X, Y] = meshgrid(1:m, 1:n);
D = sqrt((X - m/2).^2 + (Y - n/2).^2);
H = double(D <= d0);

% Apply the low-pass filter
filtered_fft_M = fft_M_shifted .* H;

% Compute the inverse FFT
filtered_image = ifft2(ifftshift(filtered_fft_M));

```

```
% Display the original and filtered images
figure;
subplot(1, 2, 1);
imshow(M, []);
title('Original Image');

subplot(1, 2, 2);
imshow(real(filtered_image), []);
title('Filtered Image with Low-Pass Filter');
```

Output of the code is shown in Figure 4.4.

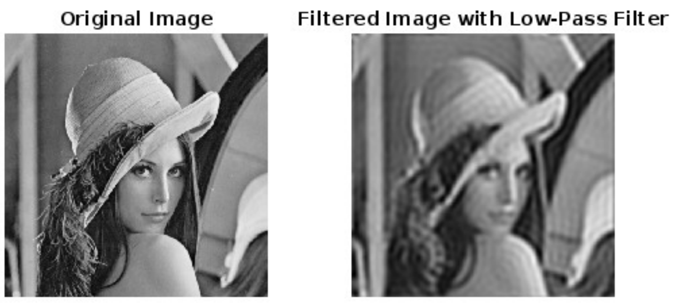


Figure 4.4: Low pass filtering of an image using FFT.

RESULTS

1. Concept and applications of Fast Fourier Transform are revisited.
2. FFT is used to solve many classical problems in probability and algebra.
3. FFT filters are used for denoising applications.