Chapter 10: Fast Fourier Transform: Derivation of the Radix-2 FFT

10.1 Introduction

The **Fast Fourier Transform (FFT)** is one of the most widely used algorithms in digital signal processing for efficiently computing the **Discrete Fourier Transform (DFT)**. The standard DFT, although conceptually simple, requires O(N2)O(N^2) operations, which becomes computationally expensive for large NN. The **Radix-2 FFT** algorithm reduces the complexity of the DFT computation to O(NlogN)O(N \log N), making it highly efficient for practical applications.

In this chapter, we will derive the **Radix-2 FFT** algorithm, which is one of the most commonly used FFT algorithms. We will go step-by-step through the process of deriving the Radix-2 FFT and explain how it optimizes the calculation of the DFT.

10.2 Discrete Fourier Transform (DFT) Recap

The **Discrete Fourier Transform (DFT)** of a sequence x[n]x[n] of length NN is defined as:

X[k]=∑n=0N−1x[n]e−j2πknNfork=0,1,…,N−1X[k] = \sum_{n=0}^{N-1} x[n] e^{-j 2\pi \frac{k n}{N}} \quad \text{for} \quad k = 0, 1, \dots, N-1

Where:

- X[k]X[k] is the DFT of x[n]x[n].
- x[n]x[n] is the time-domain signal of length NN.
- e−j2πknNe^{-j 2\pi \frac{k n}{N}} is the complex exponential factor (the "twiddle factor").
- kk is the frequency index.

This direct computation of the DFT requires O(N2)O(N^2) operations, which becomes inefficient for large NN.

10.3 Radix-2 FFT: Overview

The **Radix-2 FFT** is a divide-and-conquer algorithm that recursively breaks down the DFT into smaller DFTs. The Radix-2 FFT works efficiently when the length of the signal NN is a power of 2, i.e., N=2mN = 2^m. The main idea behind the Radix-2 FFT is to split the DFT into two smaller DFTs of half the size, compute them recursively, and combine the results.

This "divide-and-conquer" approach reduces the number of operations from $O(N^2)$ to $O(N \log N)$, making it much more efficient for large datasets.

10.4 The Radix-2 Cooley-Tukey FFT Algorithm

The **Cooley-Tukey Radix-2 FFT** algorithm is based on decomposing the DFT into two smaller DFTs by exploiting the symmetry in the complex exponentials. Here's the step-by-step derivation of the Radix-2 FFT.

10.4.1 Step 1: Breaking the DFT into Even and Odd Parts

First, observe that the DFT sum involves complex exponentials $e^{-j 2\pi \frac{k n}{N}}$. We can split this sum into two parts: one for the even-indexed terms and one for the odd-indexed terms.

For $N = 2^m$, split the sequence $x[n]$ into two subsequences:

- **Even-indexed terms:** $x[0], x[2], x[4], \dots$
- **Odd-indexed terms:** $x[1], x[3], x[5], \dots$

Let $x_{\text{even}}[n] = x[2n]$ and $x_{\text{odd}}[n] = x[2n+1]$, and rewrite the DFT as:

$$X[k] = \sum_{n=0}^{N/2-1} x[2n] e^{-j 2\pi \frac{k}{N} (2n)} + \sum_{n=0}^{N/2-1} x[2n+1] e^{-j 2\pi \frac{k}{N} (2n+1)}$$

This simplifies to:

$$X[k] = \sum_{n=0}^{N/2-1} x_{\text{even}}[n] e^{-j 2\pi \frac{k}{N} (2n)} + e^{-j 2\pi \frac{k}{N}} \sum_{n=0}^{N/2-1} x_{\text{odd}}[n] e^{-j 2\pi \frac{k}{N} (2n)}$$

By defining $X_{\text{even}}[k]$ and $X_{\text{odd}}[k]$ as the DFTs of the even-indexed and odd-indexed subsequences, respectively, we get the following recursive relation:

$$X[k] = X_{\text{even}}[k] + e^{-j 2\pi \frac{k}{N}} X_{\text{odd}}[k]$$
$$X[k + N/2] = X_{\text{even}}[k] - e^{-j 2\pi \frac{k}{N}} X_{\text{odd}}[k]$$

This is the core of the Radix-2 FFT: it splits the DFT of size N into two DFTs of size $N/2$, one for the even-indexed terms and one for the odd-indexed terms. The results are then combined to compute the DFT of size N.

10.4.2 Step 2: Recursive Computation

The process of splitting the DFT into smaller DFTs continues recursively until we reach the base case, where the DFTs are of size 2 (i.e., two-point DFTs). A two-point DFT is straightforward to compute:

$$X[0] = x[0] + x[1] \quad X[1] = x[0] - x[1]$$

This is the simplest form of the DFT, and it can be computed in constant time.

10.4.3 Step 3: Combining the Results

Once all the smaller DFTs are computed, the results are combined using the recursive formula:

$$X[k] = X_{\text{even}}[k] + e^{-j 2\pi \frac{k}{N}} X_{\text{odd}}[k]$$

$$X[k + N/2] = X_{\text{even}}[k] - e^{-j 2\pi \frac{k}{N}} X_{\text{odd}}[k]$$

The recursion terminates when the DFTs are computed for each pair of data points, and the final result is the DFT of the original sequence.

10.5 Computational Complexity of the Radix-2 FFT

The **computational complexity** of the Radix-2 FFT is significantly reduced compared to the direct computation of the DFT. Let's consider how the number of operations scales with N:

1. At each recursive level, the computation is divided into two smaller DFTs.
2. The number of levels in the recursion is $\log_2 N$.
3. At each level, we perform N operations.

Thus, the total number of operations is proportional to:

$$O(N \log_2 N)$$

This is a dramatic reduction from the $O(N^2)$ operations required for direct computation of the DFT. This makes the FFT algorithm highly efficient, especially for large datasets.

10.6 Implementation of the Radix-2 FFT

Here's an example of how the Radix-2 FFT can be implemented in Python using **NumPy**. While NumPy provides a built-in FFT function, understanding how the algorithm works can be beneficial for custom implementations.

```
import numpy as np

import matplotlib.pyplot as plt



# Generate a sample signal (e.g., a sum of two sinusoids)

fs = 1000  # Sampling frequency

t = np.linspace(0, 1, fs)  # Time vector

signal = np.sin(2 * np.pi * 50 * t) + np.sin(2 * np.pi * 150 * t)  # Sum of 50 Hz and 150 Hz sinusoids



# Compute the FFT of the signal

N = len(signal)  # Length of the signal

fft_signal = np.fft.fft(signal)



# Frequency axis

frequencies = np.fft.fftfreq(N, d=1/fs)



# Plot the FFT result (frequency spectrum)

plt.plot(frequencies[:N//2], np.abs(fft_signal[:N//2]))  # Plot positive frequencies only

plt.title('FFT of the Signal')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()
```

This code generates a signal composed of two sinusoids (50 Hz and 150 Hz), computes the FFT using `np.fft.fft()`, and then plots the magnitude of the frequency components.

10.7 Applications of the FFT

The Radix-2 FFT is a powerful tool with numerous applications:

1. **Signal Analysis:**
   - The FFT is widely used to analyze the frequency content of signals in fields like audio processing, communication, and vibration analysis.
2. **Audio Processing:**
   - In audio systems, FFT is used to perform tasks like equalization, noise reduction, and spectral analysis.
3. **Image Processing:**
   - FFT is used in image compression (e.g., JPEG), image enhancement, and edge detection.
4. **Radar and Sonar:**
   - FFT is employed in radar and sonar systems for detecting and analyzing reflected signals, providing distance and velocity measurements.
5. **Communication Systems:**
   - In digital communication, FFT is used for modulation and demodulation, especially in **OFDM (Orthogonal Frequency Division Multiplexing)** systems like Wi-Fi and LTE.

10.8 Conclusion

The **Radix-2 FFT** is an efficient and widely used algorithm for computing the Discrete Fourier Transform (DFT). By recursively breaking down the DFT computation into smaller DFTs, the

Radix-2 FFT reduces the computational complexity from O(N2)O(N^2) to O(NlogN)O(N \log N), making it feasible to analyze large datasets in real-time applications.

The understanding of how the Radix-2 FFT works, its derivation, and its applications is essential for anyone working in signal processing, especially for tasks like spectral analysis, filtering, and signal compression.