Department of Electrical Engineering and Computer Science

Howard University

Washington, D.C.

EECE 333 Fundamentals of Signals and Systems

Spring 2025

A logo of a university

Description automatically generated

Final Project Report

Aayush Jha

Bishesh Adhikari

Instructor: Dr. Lashhab

Submitted on: April 24th, 2025

**Abstract**

The Discrete Fourier Transform (DFT) is a fundamental digital signal processing (DSP) technique widely used to analyze the frequency content of discrete-time signals. Due to the high computational cost of directly evaluating the DFT for large datasets, the Fast Fourier Transform (FFT) algorithm is commonly used to compute the DFT efficiently. This project investigates the mathematical properties of the DFT and explores how the FFT algorithm enables practical computation of the DFT in signal processing applications. Python will serve as the primary platform for implementing algorithms, visualizing results, and applying these techniques to real-world signals in audio processing, image processing, and data analysis.

**Introduction**

The Discrete Fourier Transform (DFT) plays an important role in digital signal processing by doing the analysis of frequency content in discrete-time signals. However, directly calculating it is computationally intensive, especially for large datasets, making it impractical for real-time or large-scale applications. To address this, the Fast Fourier Transform (FFT) algorithm offers an efficient way to compute the DFT, reducing the computational complexity from O(n2) to O(nlogn).

This project goes into the mathematical theory and practical implementation of the FFT. By using Python for both algorithm development and data visualization, we explore how FFT can be applied to a wide array of signal and image processing problems. Through six different experiments, we demonstrate the capability and performance benefits of FFT in tasks like polynomial multiplication, signal denoising, spectral filtering, image compression, pitch detection, and custom frequency-domain analysis.

Each module shows specific real-world applications of FFT:

* Comparing FFT-based polynomial multiplication against naive methods to validate theoretical efficiency.
* Removing noise from signals using low-pass and band-pass filters in the frequency domain.
* Compressing images by discarding low-magnitude frequencies while preserving essential visual features.
* Estimating audio pitch via spectral analysis.
* Implementing a custom recursive FFT for deeper understanding and application to 1D, 2D, and time series data.

Together, these modules show the practical value of FFT as a cornerstone technique in modern signal processing.

**Theoretical Background**

1. Discrete Fourier Transform (DFT)  
   The DFT converts a finite sequence of time-domain samples into its frequency-domain representation. Mathematically, for a signal of length , the DFT is defined as:

Inverse DFT:  
To reconstruct the original signal, we have:

The time complexity to calculate this is O(n2) since we have to calculate for each N.

1. Fast Fourier Transform (FFT)

Mathematically, for a signal of length , the DFT is defined as:

Let,

, these are the Nth roots of unity. Then,

Assuming N is a power of 2. If it isn’t, we normally pad x[n] to make N a power of 2.

We split into two sequences:

: even indexed values

: odd indexed values

Then,

Taking out ,

Now,

And,

So,

Then the equation becomes:

Let,

Also, we only need to calculate till k/2, because

corresponds to the even-indexed terms, and

corresponds to the odd-indexed terms

For even part,

For odd part,

So,

Finally,

Time complexity for the FFT:

Let be the number of operations needed to compute an FFT of size N. Since, we split the FFT into two FFTs of size and do O(N) work to combine them:

…. *FFT recursive function*

**Master Theorem:**

For a recurrence of the form:

The solution depends on the comparison between and , where

If for some , then

If , then

If , and the regularity conditions holds, then

For FFT recursive function,

a = 2

b = 2

Then,

Since,

This matches Case 2 of Master theorem. So,

**Applications**

1. Polynomial Multiplication:

File used: fft.py

This module presents an experimental comparison between two polynomial multiplication techniques: the naive method with O(n2) time complexity and the Fast Fourier Transform (FFT)-based method with O(n\*log(n)) complexity. The objective is to evaluate the practical performance difference across varying polynomial sizes and validate the theoretical time complexities.

**Implementation Details**

Two approaches were implemented:

* **Naive Multiplication**: This method directly computes the product of each term in one polynomial with every term in the other. For two polynomials of length n, it requires n2 operations.
* **FFT-Based Multiplication**: This method leverages the Discrete Fourier Transform (DFT) to convert the time-domain polynomial coefficients into frequency domain, perform point-wise multiplication, and then transform the result back using the inverse FFT (IFFT). A recursive Cooley-Tukey FFT algorithm was used for both the forward and inverse transforms.

Both methods were implemented in Python. The FFT-based multiplication includes appropriate zero-padding of input vectors to the next power of two to facilitate efficient recursive decomposition.

**Experimental Setup**

Polynomials of increasing lengths n=23 to 210 (i.e., from 8 to 1024) were randomly generated for the benchmarking. For each polynomial length, the time taken to perform the multiplication was recorded separately for both methods using Python’s time module.

Additionally, two theoretical growth curves—n2 and n\*log(n) —were plotted for comparison, scaled appropriately for visual alignment with the measured runtimes.

**Results**

A graph with a line

AI-generated content may be incorrect.

**Observations**

The plot generated from the benchmarking experiment shows the following:

* **Naive method** exhibits a steep increase in computation time as n grows, consistent with its O(n2) time complexity.
* **FFT-based method** scales significantly better for larger inputs, following the expected O(n\*log(n)) pattern. While the FFT method has more overhead for smaller inputs due to recursive calls and complex number operations, its efficiency becomes apparent as n increases.
* The empirical data closely aligns with the theoretical curves, confirming the expected computational behavior of both algorithms

1. Denoising using low-pass filter

File used: fft\_denoising.py

This module demonstrates the application of the Fast Fourier Transform (FFT) for denoising a signal by removing high-frequency noise components through low-pass filtering.

**Methodology**

A synthetic noisy signal was generated by combining a clean sine wave of 50 Hz with additive Gaussian noise. The signal was sampled over 1 second at 1000 Hz (i.e., 1000 samples).

The process of denoising involves the following steps:

1. FFT Transformation: The time-domain signal was transformed into the frequency domain using the FFT, revealing the spectral content of the signal.
2. Low-Pass Filtering: A simple low-pass filter was applied in the frequency domain by zeroing out all frequency components above a defined cutoff (100 Hz in this case).
3. Inverse FFT: The filtered frequency-domain data was transformed back to the time domain using the Inverse FFT (IFFT), reconstructing the denoised signal.

**Results**

**A graph showing a signal

AI-generated content may be incorrect.**

**Observations**

The resulting plot illustrates the effectiveness of the FFT-based low-pass filter. The

filtered signal closely matches the original sine wave, with much of the high frequency

noise removed while preserving the main frequency component.

1. Band Pass Filter

File: fft\_filtering.py

This module demonstrates the use of the Fast Fourier Transform (FFT) to apply a band-pass filter to a noisy signal.

**Methodology**

A signal was synthesized by combining 6 sine waves at 20 Hz, 50 Hz, 75 Hz, 100 Hz, 150 Hz, and 200 Hz, sampled at 500 Hz for 2 seconds. Random Gaussian noise was added to simulate a noisy environment. The signal was then transformed to the frequency domain using the FFT. A band-pass filter was implemented by zeroing out all frequency components outside the desired range of 40–130 Hz. This retained the primary signal components while discarding low-frequency drift and high-frequency noise. The filtered spectrum was converted back to the time domain using the inverse FFT.

**Results**

**A diagram of a signal

AI-generated content may be incorrect.**

**Observations**

The plot shows that the filtered signal effectively preserves the original 50 Hz, 75Hz and 120 Hz components while significantly reducing the noise and filtering out the 20Hz, 150Hz, and 200Hz signals outside the pass band. This highlights the precision and simplicity of frequency-domain filtering using FFT.

1. Image Compression

File: fft\_image\_compression.py

This module demonstrates image compression by discarding low-magnitude frequency components in the frequency domain using the 2D Fast Fourier Transform (FFT).

**Methodology**

A grayscale version of the standard "astronaut" image was resized to 256×256 pixels for manageable processing. The 2D FFT was applied to transform the image into the frequency domain. The resulting spectrum was shifted to center the low frequencies. To compress the image, only the top 5% of frequency components (by magnitude) were retained, effectively discarding low-energy components that contribute less to visual perception. The filtered spectrum was then transformed back to the spatial domain using the inverse FFT.

**Results**

**A collage of women in space suits

AI-generated content may be incorrect.**

**Observations**   
The output shows a side-by-side comparison of the original and compressed images. Despite significant frequency component reduction, the compressed image retains most of the visual detail, demonstrating how high-magnitude frequencies capture essential image features.

1. Pitch Detection

File: fft\_pitch\_detection.py

This section illustrates how the Fast Fourier Transform (FFT) can be used to estimate the pitch of an audio signal.

**Methodology**

A short segment (4096 samples) was extracted from a recorded piano audio file. If the audio was stereo, only one channel was used. To reduce spectral leakage, a Hann window was applied before computing the FFT.

The FFT transformed the time-domain audio signal into its frequency components. The frequency corresponding to the maximum magnitude in the spectrum was identified as the dominant frequency, representing the pitch.

**Results**

A graph with a line graph

AI-generated content may be incorrect.

The estimated pitch frequency is printed to the console and visualized through a magnitude spectrum plot. The plot shows a distinct peak at the dominant frequency, validating the pitch detection process.

1. Custom FFT Implementation Applications

File: custom\_fft\_applications.py

This section demonstrates the use of a custom recursive FFT implementation for analyzing different types of signals, including 1D synthetic signals, 2D images, and time series data.

1. **Synthetic 1D Signal Analysis**

A synthetic signal was created by combining two sine waves (50 Hz and 120 Hz) and adding noise. To perform FFT, the signal was padded to the next power of two. The custom FFT was then applied to this padded signal, and the frequency spectrum was computed. The magnitude of the FFT was plotted to visualize the frequency content of the signal.

2. **2D FFT on Image**

For image analysis, a portion of the "astronaut" image was selected, converted to grayscale, and padded to the next power of two in both dimensions. The custom 2D FFT was applied, and the magnitude spectrum of the Fourier transform was displayed on a logarithmic scale. This spectrum reveals the distribution of frequency components across the image, where low-frequency components are centered and high-frequency components are spread out.

3. **Time Series Analysis**

A time series signal was generated by combining sinusoidal waves at 2 Hz and 6 Hz, along with added Gaussian noise. As with the 1D signal, the time series was padded to the next power of two, and the custom FFT was used to analyze its frequency content. The resulting frequency spectrum was plotted, highlighting the dominant frequencies present in the signal.

**Results**

A graph of a custom ftt

AI-generated content may be incorrect.

A close-up of a person wearing a suit

AI-generated content may be incorrect.

A graph of a custom ftt

AI-generated content may be incorrect.

The custom FFT implementation demonstrates its versatility in analyzing various types of data. By applying it to 1D signals, 2D images, and time series, we can extract valuable frequency-domain information, which is useful in a wide range of signal processing applications such as noise removal, feature extraction, and compression.

**GITHUB LINK:** [**https://github.com/AayushJha2004/signalsandsystems**](https://github.com/AayushJha2004/signalsandsystems)

**Conclusion:**

This project has thoroughly explored the Discrete Fourier Transform (DFT) and its efficient computation through the Fast Fourier Transform (FFT), showcasing its practical value across diverse signal and image processing applications. Through theoretical analysis, we demonstrated how the FFT algorithm significantly reduces computational complexity from O(n²) to O(n log n), making it suitable for real-time and large-scale problems.

Each experimental module reinforced the importance and versatility of FFT:

1. In polynomial multiplication, the FFT-based approach drastically outperformed the naive method as input size increased.
2. In denoising and band-pass filtering, FFT allowed for precise manipulation of signal frequency content, effectively isolating desired components and removing noise.
3. Image compression illustrated how discarding low-magnitude frequency components can reduce data size while retaining essential visual features.
4. Pitch detection used spectral analysis to estimate dominant frequencies in audio signals, a crucial step in many audio processing pipelines.
5. Finally, the custom FFT implementation proved the algorithm’s adaptability, enabling analysis of 1D signals, 2D images, and time-series data with consistent efficiency.

Together, these experiments demonstrate that FFT is not just a theoretical tool but a cornerstone technique in modern signal processing. Its ability to extract, manipulate, and reconstruct frequency information makes it indispensable in fields ranging from telecommunications to biomedical engineering. By implementing FFT and its applications in Python, this project has laid a solid foundation for further exploration into more advanced topics such as convolution, real-time signal processing, and machine learning-based spectral analysis.