
भारतीय प्रौद्योगिकी संस्थान धारवाड

**Indian Institute of Technology Dharwad**

# Indian Institute Of Technology Dharwad

## Group: 13

## CS-427: Project Report

---

**Fast Fourier Transform and its Application**

---

*Submitted To:*
Prof. Dr. Prabhuchandran K.J.
TA: Vikas Kumar & Puneet Kumar

*Submitted By:*
Vikas Jain - 220120029
Adarsh Gupta - 220120001
Aman Pushkar - 220120004

# Contents

# 1 Introduction

The Fourier transform is a foundational mathematical tool in signal processing that enables us to represent signals in terms of their frequency components. It transforms a time-domain signal into its frequency-domain representation, revealing the periodic behaviors and frequency patterns in the data. This transformation is particularly valuable for the analysis, filtering, and compression of signals in various fields, including audio processing, image analysis, and communication.

In practical applications for analysis, we often use the Discrete Fourier Transform (DFT), which transforms discrete data points into a sum of sines and cosines functions, effectively decomposing the signal into its frequency components. However, DFT comes with a computational complexity of $O(N^2)$, making it inefficient for large data sets or real-time applications.

To overcome this limitation, the Fast Fourier Transform (FFT) was developed, reducing the computational complexity to $O(N \log N)$. This breakthrough allowed Fourier analysis to be used on a much larger scale, enabling real-time processing in fields like digital signal processing. FFT's efficiency has made it a standard tool in applications such as audio compression formats (e.g., MP3) and image compression (e.g., JPEG).

This report explores the theoretical foundations of the Fourier series, DFT, and FFT, and demonstrates their applications in real-world scenarios like frequency domain analysis, image compression and audio compression . Using FFT, we can perform frequency analysis on large datasets, aiding in tasks like noise filtering in audio signals and frequency-based image compression. This investigation aims to highlight the power of FFT and its essential role in modern signal processing.

# 2 Motivation

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT), which is crucial for analyzing signals in the frequency domain. By decomposing signals into their frequency components, FFT helps identify patterns, periodic behaviors, and underlying structures that are not easily visible in the time domain.

In real-world applications, FFT and DFT are widely used in fields like audio processing, communications, medical imaging, and financial analysis.

- **Audio Processing**: FFT is used in audio compression formats like MP3 and AAC, where it helps break down audio signals into their frequency components, enabling efficient storage and transmission. In real-time audio processing, such as live sound filtering or noise cancellation, FFT allows for rapid analysis and modification of the signal's frequency spectrum.

- **Image Processing**: In digital image compression (e.g., JPEG), FFT is applied to convert spatial data into frequency data, allowing efficient data compression by removing less important frequencies. It is also used in image filtering and edge detection.
- **Communications**: In telecommunications, FFT is essential for the efficient processing of signals in systems such as orthogonal frequency division multiplexing (OFDM), which is used in technologies such as 4G/5G networks and Wi-Fi. FFT enables the fast analysis and manipulation of signals in these high-speed systems.
- **Medical Imaging**: Techniques like magnetic resonance imaging (MRI) and CT scans rely on Fourier transforms to process raw data from sensors and convert them into usable images. FFT reduces computation time, making these imaging techniques more efficient.
- **Financial Analysis**: In financial markets, FFT is applied to detect patterns in stock prices and other time series data. It helps identify cyclical trends or anomalies that can guide investment strategies.

The efficiency of FFT has had a transformative impact in these fields, enabling faster processing, better data compression, and more accurate analysis. By reducing computational burden, FFT has made frequency analysis accessible in real-time applications, improving everything from daily communications to advanced medical diagnostics.

# 3 Problem Statement

Fourier analysis enables the transformation of signals from the time domain to the frequency domain, revealing patterns and frequencies that are not immediately apparent in the original form. However, calculating the discrete Fourier transform (DFT) for a sequence of $N$ points requires significant computation with a complexity of $O(N^2)$. This high computational cost can be prohibitive for large datasets or real-time applications, where quick results are essential.

The Fast Fourier Transform (FFT) solves this problem by reducing the computational complexity to $O(N \log N)$, making frequency analysis much faster and more efficient. This reduction in computation time has made FFT an ideal tool for applications that require rapid data processing and analysis. In this report, we investigate the properties of FFT and demonstrate its practical applications, with a focus on two main areas: audio processing, where it enables the analysis and compression of the frequency spectrum, and image processing, where it facilitates spatial frequency

filtering and compression. Through these applications, our aim is to show how FFT can optimize data compression.

# 4 Introduction to Fourier Series

A fundamental result in Fourier analysis is that if $f(x)$ is periodic and piecewise smooth, then it can be expressed as a Fourier series, which is an infinite sum of cosines and sines of increasing frequency. Specifically, if $f(x)$ is $2\pi$-periodic, it may be written as:

$$f(x) = a_0 + \sum_{k=1}^{\infty} (a_k \cos(kx) + b_k \sin(kx))$$

where the coefficients $a_k$ and $b_k$ are determined by:

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, dx$$

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx)\, dx$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(kx)\, dx$$

The Fourier series can also be represented using complex exponentials, using Euler's formula $e^{ikx} = \cos(kx) + i \sin(kx)$:

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

Where the complex Fourier coefficients $c_k$ are defined as:

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-ikx}\, dx$$

This comprehensive formulation not only allows the representation of $2\pi$-periodic functions in terms of their sine and cosine components but also aids in a variety of applications in signal processing, differential equations, and the analysis of periodic functions.

# 5 Approximating the Hat Function using Fourier Series

A typical hat function $f(x)$ is defined as:

$$f(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

This is a symmetric, piecewise linear function with period 2.

The Fourier series representation of a periodic function $f(x)$ with period 2 is given by:

$$f(x) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(n\pi x) + b_n \sin(n\pi x)]$$

where the Fourier coefficients $a_0$, $a_n$, and $b_n$ are calculated by integrals over one period.

## 5.1 Calculating the Coefficients

**1. Zeroth Coefficient** $a_0$: This represents the average value of the function over one period:

$$a_0 = \frac{1}{3}$$

**2. Cosine Coefficients** $a_n$: The Fourier cosine coefficients are given by:

$$a_n = \frac{1}{2} \int_{-1}^{1} (1 - |x|) \cos(n\pi x)\, dx$$

The coefficients $a_n$ decay as $1/n^2$.

**3. Sine Coefficients** $b_n$: Since the hat function is symmetric (even function), all sine coefficients are zero:

$$b_n = 0$$

## 5.2 Fourier Series Approximation

The Fourier series of the hat function is:

$$f(x) \approx \frac{1}{3} + \sum_{n=1}^{N} a_n \cos(n\pi x)$$

As more terms (higher $N$) are included, the approximation becomes more accurate. However, there will be oscillations near the edges of the function, which can be reduced by including more terms.

In summary, the hat function is approximated by summing the Fourier series, where the coefficients $a_0$ and $a_n$ are calculated for the function. The result improves as more terms are included in the sum.

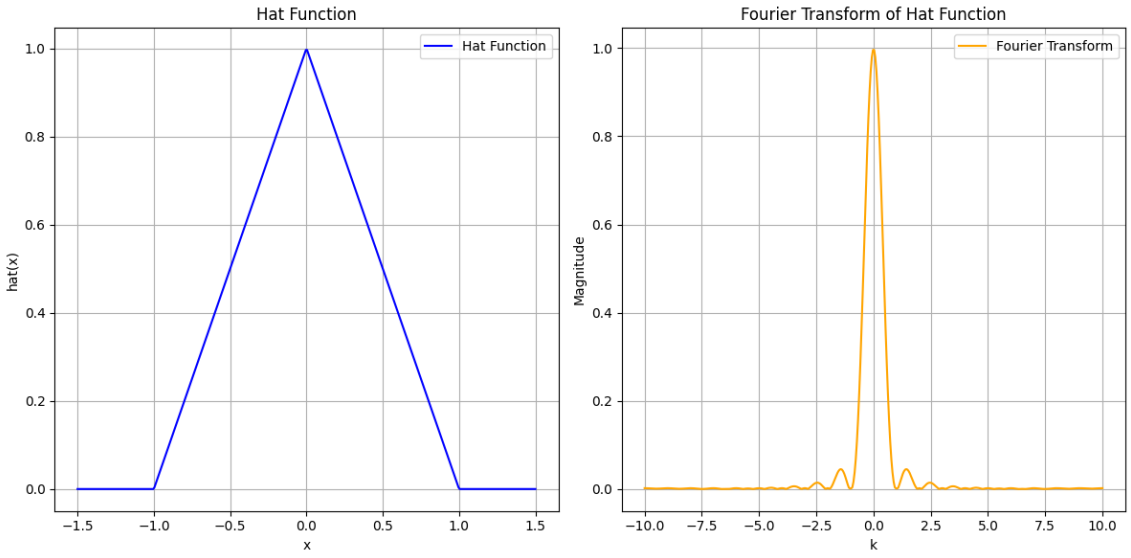


Figure 1: Hat and approximate fourier transform

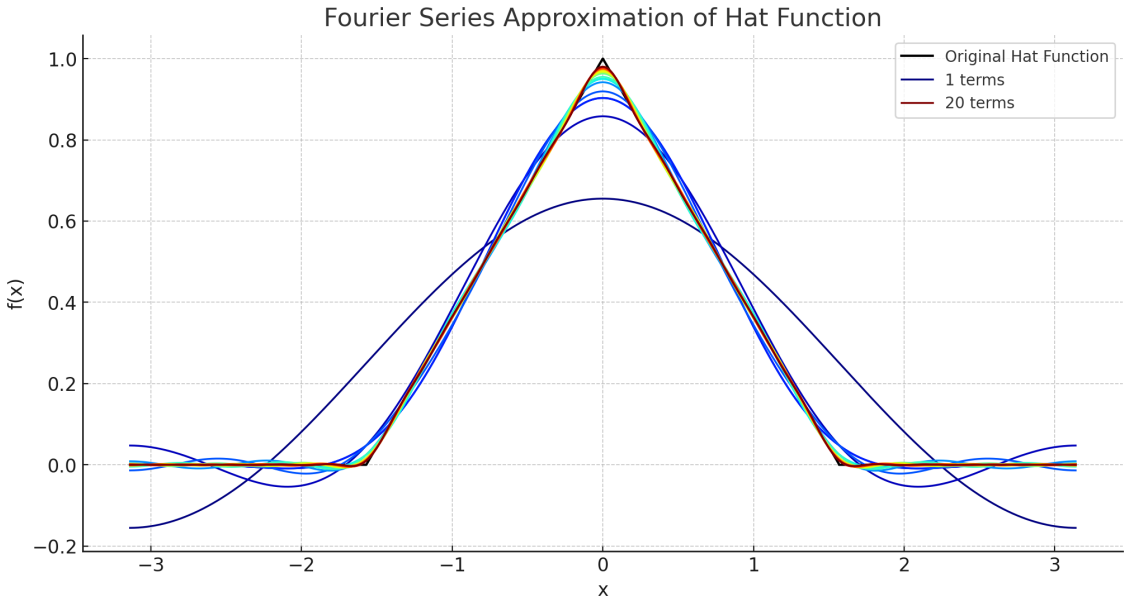


Figure 2: Hat function and different sines+cosines waves

# 6 Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

## 6.1 Introduction to Discrete Fourier Transform (DFT)

While Fourier series and Fourier transforms are commonly applied to continuous functions $f(x)$, real-world applications typically work with discrete data.

The **Discrete Fourier transform (DFT)** is a method of transforming a sequence of discrete data points into a sum of sines and cosines, effectively moving data from the time domain to the frequency domain. This transformation allows for the analysis and manipulation of data based on its frequency components.

The DFT for a sequence of $n$ data points $\{f_0, f_1, \ldots, f_{n-1}\}$ is given by:

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j e^{-i2\pi jk/n}, \quad k = 0, 1, \ldots, n-1$$

The **inverse DFT (iDFT)**, which reconstructs the original data from its frequency components, is defined as:

$$f_j = \frac{1}{n} \sum_{k=0}^{n-1} \hat{f}_k e^{i2\pi jk/n}, \quad j = 0, 1, \ldots, n-1$$

In matrix form, the DFT can be represented as a multiplication by a unitary matrix $F$, known as the DFT matrix:

$$\hat{f} = Ff$$

The DFT can be computed as a matrix-vector multiplication, where the **DFT matrix** $F$ is a unitary matrix with entries based on the complex exponentials. For an input vector $f = [f_0, f_1, \ldots, f_{n-1}]$, the DFT maps $f$ to the frequency domain vector $\hat{f} = [\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{n-1}]$ as follows:

$$\hat{f} = Ff$$

where $F$ is defined by the matrix:

$$F = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)} \end{bmatrix}$$

with $\omega_n = e^{-2\pi i/n}$, which is the primitive root $n$ of unity.

Each element $F_{jk}$ of the matrix is defined as:

$$F_{jk} = \omega_n^{jk} = e^{-2\pi ijk/n}$$

This structure allows $F$ to efficiently transform the time domain vector $f$ into its frequency components when used in the FFT algorithm.

where $F$ is a complex-valued Vandermonde matrix with a structure that includes both magnitude and phase information. The DFT matrix makes it possible to move data from the time domain to the frequency domain and is crucial for applications in signal processing.

## 6.2 Usage of DFT in frequency Domain Analysis

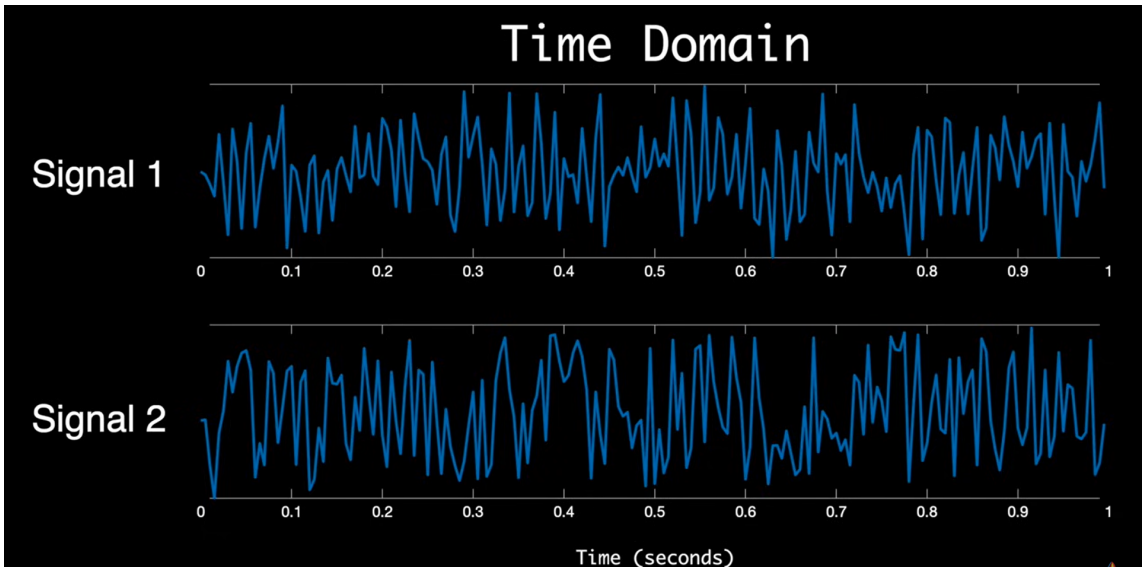


Figure 3: Time domain of signals

For example, here we have two time domain signals that are sampled at 200 hertz. And the question I have for you is, which of these has a significant 60 hertz component? It's not terribly obvious, right? They both look pretty similar. But if we transform them into the frequency domain, it's much easier to answer

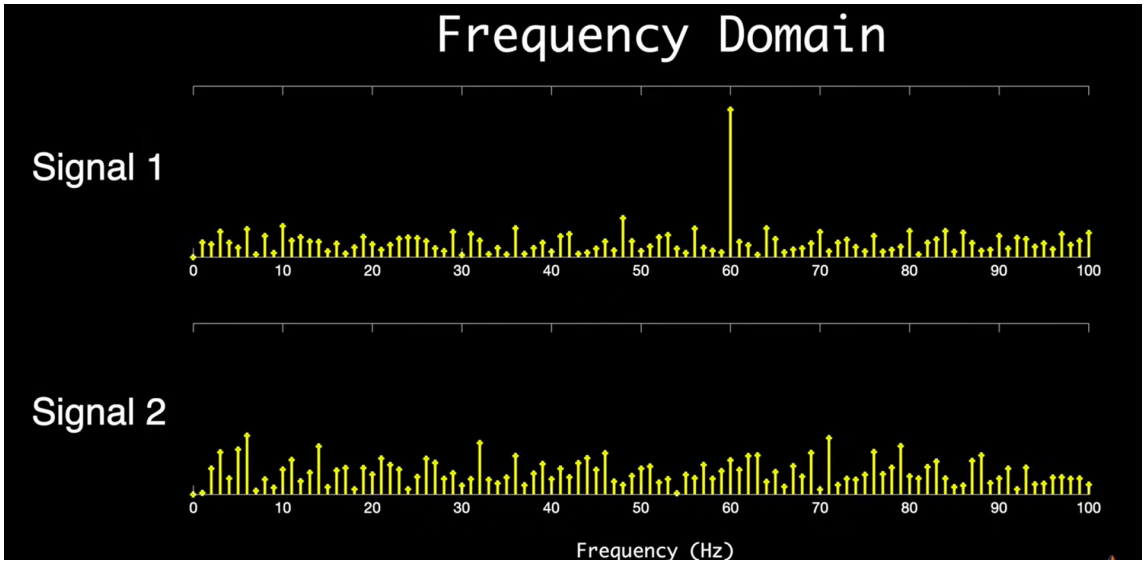


Figure 4: Frequency domain of signals

Clearly the first signal because there's this large 60hz peak. So we can use the frequency domain to understand the frequency makeup of a signal.

## 6.3 Limitations of DFT

The **Discrete Fourier Transform (DFT)** is a powerful tool that transforms a signal from the time domain into the frequency domain, which is extremely useful for analyzing and processing digital signals. However, the traditional way of computing the DFT can be very resource-intensive, especially when working with large datasets. The reason for this is that the computational complexity of the DFT is $O(N^2)$, meaning the time required to perform the DFT grows dramatically as the size of the data increases.

To put this into perspective, consider a 10-second audio clip sampled at 44.1 kHz, which results in $N = 4.41 \times 10^5$ samples. Calculating the DFT of this audio signal would require roughly $2 \times 10^{11}$ multiplications. If we assume that a processor can handle one million (i.e., $10^6$) multiplications per second, the DFT computation would take about **200,000 seconds**, which is equivalent to more than **55 hours**!

In real-world applications, such long computation times make the DFT impractical for many purposes, such as real-time audio processing or live video streaming. Even when processors can perform operations at faster rates, the exponential increase in computation time with larger data sizes makes DFT inefficient for high-resolution images, audio, or video. This inefficiency has restricted the applicability of DFT to only those tasks where computational speed is not a primary concern, such as offline processing or small data sets.

## 6.4 Introduction to FFT

The **Fast Fourier Transform (FFT)** algorithm addresses the inefficiencies of the Discrete Fourier Transform (DFT) by significantly reducing its computational complexity, from $O(N^2)$ to $O(N \log(N))$. For large datasets, this reduction in complexity can lead to a drastic decrease in computation time.

To illustrate this, let's revisit the example of a 10-second audio clip with $N = 4.41 \times 10^5$ samples. In this case, the FFT would require around $6 \times 10^6$ multiplications, compared to the DFT's $2 \times 10^{11}$. Assuming the processor can perform one million multiplications per second, the FFT would finish in just **6 seconds**, which is more than **30,000 times faster** than the DFT!

This remarkable efficiency has made FFT a practical solution for real-time applications and has played a crucial role in enabling a variety of digital technologies, such as audio compression (like MP3), image compression (like JPG), and telecommunications. Given its speed and versatility, FFT has largely replaced the DFT

in most practical applications, becoming a fundamental algorithm built into almost every device and operating system that handles digital signal processing.

# 7 Mathematical Working of FFT in Image Compression

Image compression using FFT involves transforming the image from the spatial domain to the frequency domain. This transformation reveals the frequency components of the image, allowing for effective compression by discarding less significant frequencies. Here's how it works mathematically:

**1. Transformation**: The image, represented as a matrix of pixel intensities $I(x, y)$, is transformed using the 2D Fourier Transform:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x, y) \cdot e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

where $F(u, v)$ represents the Fourier coefficients, and $M$ and $N$ are the dimensions of the image.

**2. Frequency Domain Representation**: The resulting matrix $F(u, v)$ contains both amplitude and phase information of the frequency components. The low-frequency components (located at the center of the transformed matrix) represent the essential information of the image, while high-frequency components (edges and fine details) are often less significant.

**3. Compression via Coefficient Thresholding**: To compress the image, we apply a threshold that keeps only a fraction of the coefficients. This is often implemented using a mask that retains only the significant low-frequency coefficients, effectively zeroing out the high-frequency coefficients:

$$F'(u, v) = \begin{cases} F(u, v) & \text{if } |F(u, v)| > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

**4. Inverse Transformation**: After applying the threshold, we perform the inverse Fourier Transform to reconstruct the compressed image:

$$I'(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F'(u, v) \cdot e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

**5. Compression Ratio Calculation**: The effectiveness of the compression can be evaluated by calculating the compression ratio, which is defined as:

$$\text{Compression Ratio} = \frac{\text{Size of Original Data}}{\text{Size of Compressed Data}}$$

The FFT-based approach to image compression is highly effective because it allows significant reductions in data size while retaining the essential visual characteristics of the original image. The ability to discard less important high-frequency information while preserving low-frequency details is key to achieving efficient compression.

The FFT has become synonymous with the DFT in modern applications, with nearly all devices performing digital signal processing employing FFT-based computations. Its computational advantages have made it a critical component in a wide range of applications.

## 7.1 Results on Image Dataset

We have implemented the fast Fourier transform on the image dataset by converting it to the grey scale and are showing the compression results using fft, and the working of the same has been shown above.


Figure 5: Original image


Figure 6: Compressed image-1


Figure 7: Original image


Figure 8: Compressed image-1


Figure 9: Original image


Figure 10: Compressed image-1


Figure 11: Original image


Figure 12: Compressed image-1

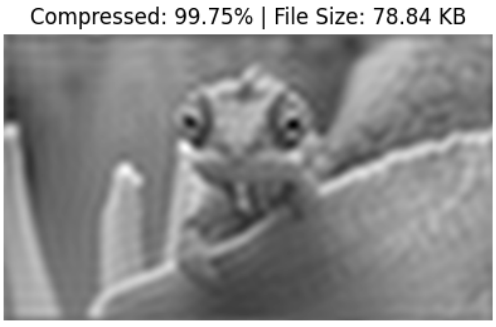


Figure 13: Original image

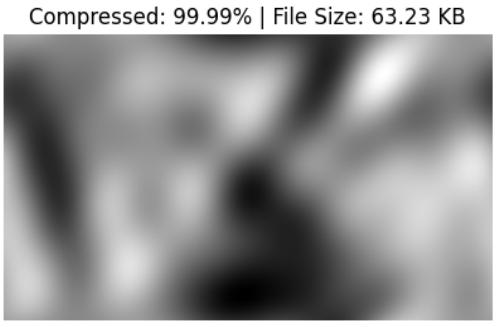


Figure 14: Compressed image-1

## 7.2 Results and Code links and for Image files

**Link to the image results folder**

https://colab.research.google.com/drive/1P_tx2RKvGn_3IjoklvB3pJwnWD6HYl5I?usp=drive_link

**Link to the google colab file**
https://colab.research.google.com/drive/1P_tx2RKvGn_3IjoklvB3pJwnWD6HYl5I?usp=drive_link

# 8 Mathematical working of Audio Compression Using FFT

The Fast Fourier Transform (FFT) is a powerful algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, which is instrumental in audio signal processing. The DFT converts a time-domain signal into its frequency-domain representation, allowing us to analyze the frequency components of the audio signal.

## 8.1 Discrete Fourier Transform (DFT)

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

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

where: - $X[k]$ is the DFT output (frequency spectrum), - $x[n]$ is the input time-domain signal, - $j$ is the imaginary unit.

The FFT algorithm reduces the computational complexity of the DFT from $O(N^2)$ to $O(N \log N)$, making it feasible for real-time applications.

## 8.2 Compression Process

In audio compression, the FFT allows us to retain only the most significant frequency components. The basic steps involved in the compression process are:

**1. Transform the Signal**: Compute the FFT of the audio signal $x[n]$:

$$F = \text{FFT}(x[n]) \tag{2}$$

**2. Frequency Domain Masking**: To achieve compression, we apply a mask to retain only a fraction of the lowest frequency components. Let keep_fraction be the desired fraction of frequencies to retain:

$$\text{mask}[k] = \begin{cases} 1 & \text{if } k < r_{\text{keep}} \text{ or } k > N - r_{\text{keep}} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $r_{\text{keep}} = \lfloor N \cdot \text{keep\_fraction} \rfloor$. **3. Apply the Mask**: The masked frequency spectrum is obtained as:

$$F_{\text{compressed}}[k] = F[k] \cdot \text{mask}[k] \tag{4}$$

**4. Inverse Transform:** The compressed audio signal is reconstructed by applying the inverse FFT:

$$\hat{x}[n] = \text{IFFT}(F_{\text{compressed}}[k]) \tag{5}$$

where $\hat{x}[n]$ is the reconstructed time-domain signal.

## 8.3 Compression Ratio

The compression ratio can be calculated based on the size of the original and compressed audio data. If $S_{\text{original}}$ is the size of the original audio and $S_{\text{compressed}}$ is the size after compression, the compression ratio $R$ is given by:

$$R = \left(1 - \frac{S_{\text{compressed}}}{S_{\text{original}}}\right) \times 100\% \tag{6}$$

This mathematical framework enables efficient audio compression while preserving the essential characteristics of the original signal.

## 8.4 Implementation and Efficiency of FFT

The primary advantage of FFT over DFT lies in its ability to split the computation into recursive halves if $n$ is a power of two. For example, if $n = 1024$, the DFT matrix $F_{1024}$ can be represented by the smaller matrices $F_{512}, F_{256}, \ldots, F_2$, recursively breaking down the computation. If $n$ is not a power of two, the vector can be padded with zeros to achieve this property, further enhancing the efficiency of the FFT.

| Compression Ratio | File Size |
|---|---|
| 50% | 79.23 KB |
| 30% | 65.13 KB |
| 20% | 54.75 KB |
| 10% | 45.58 KB |

Figure 15: Compression ratio and File Size

## 8.5 Results and Code Links for audio files

**Link to the audio compressed files**

https://drive.google.com/drive/folders/1q_jDcXhodx8QlNuRGtSitowqgAXxeFO0?usp=drive_link

**Link to the google colab file**

https://colab.research.google.com/drive/1P_tx2RKvGn_3IjoklvB3pJwnWD6HYl5I?usp=drive_link

# 9 Using Fast Fourier Transform (FFT) for Noise Removal

The Fast Fourier Transform (FFT) is an efficient algorithm that transforms a signal from the time domain to the frequency domain, enabling the analysis of its frequency components. In this section, we discuss how FFT is used to remove high-pitched noise from an audio signal.

## 9.1 Conceptual Overview

Audio signals consist of multiple frequencies, with high-pitched noise typically appearing in the higher frequency range. By applying FFT, we convert the signal into its frequency components, where we can identify and filter out the high-frequency noise.

The process involves:

1. **Transforming to the Frequency Domain:** FFT is applied to break the audio signal into its constituent frequencies, represented by both amplitude and phase.
2. **Identifying and Filtering Noise:** A cutoff frequency is chosen above which all components are considered noise. Frequencies above this threshold are set to zero, effectively removing the unwanted high-pitched disturbance.
3. **Reconstructing the Audio Signal:** The filtered signal is transformed back to the time domain using the inverse FFT (IFFT), reconstructing the audio without the noise.
4. **Normalization:** The audio is then normalized to ensure the amplitude remains within the valid playback range.

## 9.2 Filtering High-Frequency Noise

High-frequency noise is removed by defining a cutoff frequency. Components above this threshold are set to zero, leaving the desired low- and mid-frequency components

intact. This approach effectively eliminates noise while preserving the important audio content.

## 9.3 Frequency Spectrum Analysis

The effectiveness of the filtering can be visualized by comparing the frequency spectrum before and after filtering. Initially, the spectrum includes both the signal and noise in the high-frequency range. After filtering, the high-frequency noise is reduced, resulting in a cleaner signal.

## 9.4 Results for Noise Removal

Here are the frequency vs amplitude plots for the original and filtered audio:

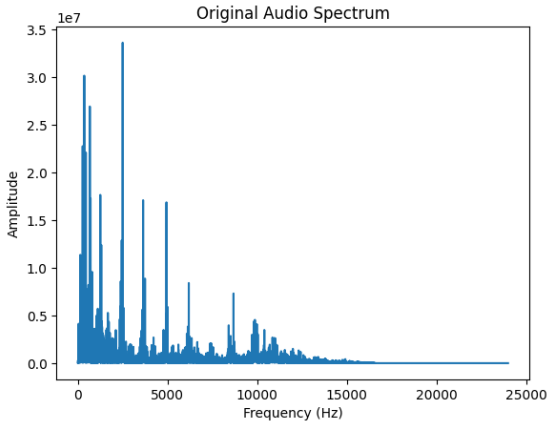


Figure 16: Original Audio

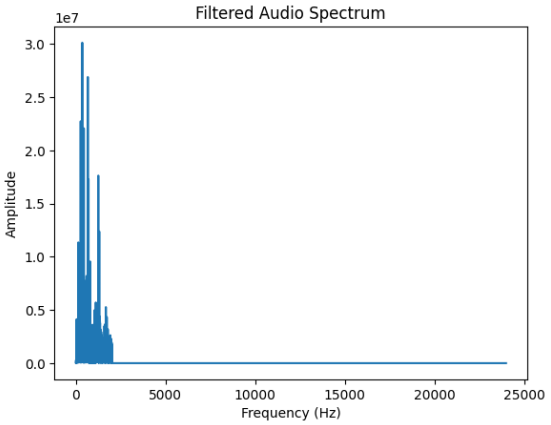


Figure 17: Filtered Audio

**Here is the link for the outputs generated:**
https://drive.google.com/drive/folders/1rd2NvrgUL3_Wvy6b5-PqUPTG1CyuWi1R?usp=sharing

**Link to the google colab file** https://colab.research.google.com/drive/1P_tx2RKvGn_3IjoklvB3pJwnWD6HYl5I?usp=drive_link

# 10 Conclusion and Future Work

In this project, we delved into the Fast Fourier Transform (FFT), understanding its foundational principles and observing its applications in compressing audio and image data. The FFT stands out as a powerful alternative to the Discrete Fourier Transform (DFT) due to its reduced computational complexity, which allows it

to process large datasets efficiently. Our experiments demonstrated how FFT can effectively compress data by preserving critical frequency components and discarding redundant information, with practical results showcased in both audio and image datasets.

Looking ahead, there is a wealth of opportunity to build on this work. One promising direction is to develop adaptive FFT techniques that adjust compression thresholds dynamically based on content type. For example, we could tailor audio compression to emphasize specific frequency ranges for speech or music. Another interesting avenue would be to integrate FFT with machine learning algorithms to enhance real-time signal analysis, which could be particularly useful in streaming applications. The versatility of FFT positions it as a valuable tool not only in multimedia but also in fields like medical imaging and telecommunications. Continued exploration and optimization of FFT will enable it to meet the growing demands for efficiency and quality in these areas.