Computational Physics Project Report

Submitted to Dr. Subhashish Basak and Dr. Rittik Deb

Sumegha M.T (1911180)

Academic Year: 2024

Acknowledgement

I would like to express my deep and sincere gratitude to Dr. Rittik Deb and Dr. Subhashish Basak for giving me the opportunity to work on this project. I thoroughly enjoyed learning about the Fast Fourier Transform and Fourier Transform and applying it to the data provided by Dr. Rittik Deb's Master's and Doctoral students. The discussions I had with Dr. Rittik Deb, Shwetha Sasindran, N. Devanand, Souradeep Bhaiyya were motivating. I will always cherish and try to emulate the hospitality of Dr. Rittik Deb's lab.

Sumegha M T
5th Year Integrated M.Sc.
School of Physical Sciences
NISER, Bhubaneswar

1 Aim

To study and understand Fast Fourier Transform (FFT) algorithm and inverse FFT and apply it to a audio signals consisting of cricket chirps and generate the frequency spectrum of the signals.

2 Theoretical Background

The Fourier transform is widely employed across various disciplines in engineering and science by professionals such as circuit designers, spectroscopists, crystallographers, signal processing and communications experts, as well as those involved in imaging. Originally, Fourier analysis focused on the representation and analysis of periodic phenomena using Fourier series. Later, it expanded to encompass the application of those principles to non-periodic phenomena through the Fourier transform.

2.1 Fourier Series

Given a periodic function $f(t)$ of period T, that is,

$$f(t+T) = f(t) \tag{1}$$

we can write it as

$$f(t) = a_0 + a_1 cos(\omega_0 t) + b_1 sin(\omega_0 t) + a_2 cos(2\omega_0 t) + b_2 sin(2\omega_0 t) + ... + a_k cos(k\omega_0 t) + b_k sin(k\omega_0 t)$$

where

$$\omega_0 = \frac{2\pi}{T}$$

is the frequency of the signal and $k = 0, 1, 2....$
In other words, a fourier series of a periodic function of frequency ω_0 is

$$f(t) = a_0 + \sum_{k=1}^{\infty} [a_k cos(k\omega_0 t) + b_k sin(k\omega_0 t)] \tag{2}$$

Here

$$a_0 = \frac{1}{T} \int_0^T f(t)\, dt \tag{3}$$

$$a_k = \frac{2}{T} \int_0^T f(t) cos(k\omega_0 t)\, dt \tag{4}$$

$$b_k = \frac{2}{T} \int_0^T f(t) sin(k\omega_0 t)\, dt \tag{5}$$

We can write fourier series in terms of Euler's identity

$$f(t) = \sum_{k=-\infty}^{\infty} e^{ik\omega_0 t} \tilde{C}_k \tag{6}$$

where

$$\tilde{C_k} = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-ik\omega_0 t}\, dt \tag{7}$$

$\tilde{C_k}$ is known as fourier coefficient. It is a complex number. The squared magnitude of fourier coefficients, $|\tilde{C_k}|^2$ are identified as the energy of the (positive and negative) harmonics$e^{\pm k\omega_0 t}$. A frequency spectrum is a plot between $|\tilde{C_k}|$ and $k\omega_0$. The plot of $|\tilde{C_k}|^2$ is knows as power spectrum or the energy spectrum. These plots give an idea of how the fourier coefficients increase or decrease. Given a signal, the frequency spectrum offers the information on how much each frequency component of the signal contribute to the overall signal.

2.2 Fourier Transform

We can extend fourier series to non-periodic function by letting the period $T \to \infty$.
In fourier series, Using equation 7 in equation 6, we get

$$f(t) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \left(\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-ik\omega_0 t}\, dt \right) e^{ik\omega_0 t} \tag{8}$$

We define

$$n_k = \omega_0 k = \frac{2\pi}{T} k \tag{9}$$

where k is an integer going from $-\infty$ to $+\infty$. n_k depends on T.
Then $\Delta n = \frac{2\pi}{T}$
The equation 8 then becomes

$$f(t) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \left(\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-in_k t}\, dt \right) e^{in_k t} \Delta n \tag{10}$$

This can be written as

$$f(t) = \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} C(n_k) e^{in_k t} \Delta n \tag{11}$$

where

$$C(n) = \frac{1}{\sqrt{2\pi}} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-int}\, dt \tag{12}$$

This is basically a continuum generalisation of the fourier coefficients in the fourier series.
Taking $T \to \infty$

$$C(n) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-int}\, dt \tag{13}$$

Under the same limit, $\Delta n \to 0$. Hence,

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} C(n) e^{int}\, dt \tag{14}$$

where n is a real number close to zero.
Definition Let $f : \mathbb{R} \to \mathbb{C}$. The Fourier transform of f is denoted by $F[f] = \hat{f}$ where

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx} dx. \tag{15}$$

The plot of magnitude of $\hat{f}k$ is the frequency spectrum and that of its square is the power spectrum.
Similarly, the inverse Fourier transform of f is denoted by $F^{-1}[\hat{f}] = f$ where

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(k) e^{ikx} \tag{16}$$

When a periodic signal is decomposed to its fourier series and its frequency spectrum is plotted, each frequency component of the frequency spectrum is indexed by k, which is an integer. The frequency spectrum in this case consists of points that are not close to each other. However, when a non-periodic function is undergoes fourier transform, the frequency components in the resulting spectrum is indexed by n, which, by definition, becomes smaller as the period $T \to \infty$ and the values of n gets infinitesimally closer. Thus, the frequency spectrum of a non-periodic function is continuous in nature.

2.3 Discrete fourier transform

Discrete fourier transform (DFT) is the real life application of fourier transform. In practical situations, all signals are sampled at discrete points in its domain. The discrete version of a function $f(t)$ is a list of sampled values $f(t_0), f(t_1), ..., f(t_{N-1})$. Suppose a signal is given and it lies in an interval from $[0, T]$ sampled at equidistant points $t_n = n\Delta t$. Here

$$\Delta t = \frac{T}{N-1}$$

where N is the total number of points and n goes rom 1 to $N-1$.
Equation 15 needs to be modified to be applicable in practical cases. For that, we discretize the integral and arrive at

$$\hat{f}(\omega) = \Delta t \sum_{n=0}^{N-1} f(t_n) e^{-i\omega t_n} \tag{17}$$

Since $f(t)$ itself is discrete, the fourier transform is just samples of $\hat{f}(\omega)$ in the frequency domain. In other words, frequency components are also discretized.
The frequencies are sampled over the interval from 0 to $\frac{1}{\Delta t}$ with a sampling rate of $\frac{1}{(N-1)\Delta t}$. Hence, the discrete frequencies are s_0=0,s_1=$\frac{1}{(N-1)\Delta t}$,
s_2=$\frac{2}{(N-1)\Delta t}$, ..., $s_{(N-1)} = \frac{1}{\Delta t}$.
Thus there are equal number of sample points in the time and frequency domain. Note that

$$\omega_k = 2\pi s_k \tag{18}$$

Therefore

$$\hat{f}(\omega_k) = \Delta t \sum_{n=0}^{N-1} f_n e^{-i\frac{2\pi}{N}kn} \tag{19}$$

Here $f_n = f(t_n)$ We can write this as

$$F_k = \frac{\hat{f}(\omega_k)}{\Delta t} = \sum_{n=0}^{N-1} f_n e^{-i\frac{2\pi}{N}kn} \tag{20}$$

Period of F_k is N, ie, $F_{k+N} = F_k$. F_k is a complex quantity whose magnitude gives the frequency spectrum and squared magnitude gives the power spectrum.
Similarly, the inverse DFT is given by

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{i\frac{2\pi}{N}kn} \tag{21}$$

From the about equations 20 and 21, we can see that for each frequency index k which ranges from 0 to $N-1$, upto $N-1$ summation is required for the evaluation of fourier transform, resulting in a total of $O(N^2)$ operations. Practically, this is cumbersome because a well sampled signal of duration 1 second contains as much as 100,000 samples and to find the fourier transform of such a short signal, over 10^{10} operations must be performed!

2.4 Fast Fourier Transform

The FFT algorithm drastically reduces the computational burden of computing the DFT, bringing the complexity down to $O(Nlog_2N)$. This efficiency makes it feasible to analyze signals with tens of thousands or even millions of samples in a reasonable amount of time. The Fast Fourier Transform (FFT) algorithm is a divide-and-conquer approach to computing the Discrete Fourier Transform (DFT) more efficiently.
Recall equation 20. Discrete Fourier Transform of an array A of size N can also be written as

$$Y[j] = \sum_{k=0}^{N-1} A[k] W_N^{kj} \tag{22}$$

where $W_N = e^{-\frac{i2\pi}{N}}$ is the N^{th} complex root of unity. k and n are indices of output array Y and input array A respectively. k and n runs from $0, 1, 2, .., N-1$.
A straightforward calculation of the above would require $O(N^2)$ operations.
Now let $N = r_1 r_2$ and let $j = j_0 + j_1 r_1$ and $k = k_0 + k_1 r_2$ where

$$j_0 = 1, 2, 3, ..., r_1 - 1$$

$$k_0 = 1, 2, 3, ..., r_2 - 1$$

$$j_1 = 1, 2, 3, ..., r_2 - 1$$

$$k_1 = 1, 2, 3, ..., r_1 - 1$$

Then

$$Y[(j_1, j_0] = \sum_{k_0=0}^{r_2-1} \sum_{k_1=0}^{r_1-1} A[(k_1, k_0)] W_N^{jk_0} W_N^{jk_1 r_2} \tag{23}$$

By the property of cube root of unity, we have

$$W_N^{jk_1 r_2} = W_N^{j_0 k_1 r_2} \tag{24}$$

Let

$$A_1[(k_0, j_0)] = \sum_{k_1=0}^{r_1-1} A[(k_1, k_0)] W_{j_0}^{k_1 r_2} \tag{25}$$

This takes r_1 operations. Since the array A_1 contains N values, $N \times r_1$ operations are required.
Now,

$$Y[(j_1, j_0] = \sum_{k_0=0}^{r_2-1} A_1[(k_0, j_0)] W_N^{(j_1 r_1 + j_0) k_0} \tag{26}$$

This takes r_2 operations and in total requires Nr_2 operations for all the N values in the Y array. Hence, the total number of operations is

$$T = N(r_1 + r_2)$$

operations.
Now, if $N = r^m$, then $T = N(mr)$. Also

$$log_2 N = m log_2 r$$

Hence,

$$\frac{T}{N} = mr$$

and

$$\frac{T}{N log_2 N} = \frac{r}{log_2 r}$$

For a given N, $\frac{r}{log_2 r}$ is the lowest for $r = 3$. However, $r = 2$ or $r = 4$ is chosen as they offer advantages for computers with binary arithmetic.
With $r = 2$ and considering the binary number system; we can write integers j and k as

$$j = j_{m-1} 2^{m-1} + ... + j_1 2 + j_0$$

$$k = k_{m-1} 2^{m-1} + ... + k_1 2 + k_0$$

where j_ν and k_ν are equal to 0 or 1 for the given index ν (bit position) in the binary system representation of the integer j and k respectively.
Representing the input and output arrays as a function of their bit positions

$$Y[(j_{m-1}, ..., j_0)] = \sum_{k_0} \sum_{k_1} ... \sum_{k_{m-1}} A[(k_{m-1}, ..., k_0)] W_N^{j(k_{m-1} 2^{m-1} + k_{m-2} 2^{m-2} + ... + k_1 2^1 + k_0)} \tag{27}$$

where the summation is over $k_\nu = 0, 1$. By the properties of cube root of unity (see equation 24)

$$W_N^{j(k_{m-1}2^{m-1})} = W_N^{j_0(k_{m-1}2^{m-1})}$$

we evaluate the innermost sum of equation 27

$$A_1[(j_0, k_{m-2}, ..., k_0)] = sum_{k_{m-1}} A[(k_{m-1}, ..., k_0)] W_N^{j_0(k_{m-1}2^{m-1})} \tag{28}$$

Proceeding to the other innermost summation terms using

$$W_N^{j(k_{m-l}2^{m-l})} = W_N^{(j_{l-1}2^{l-1}+...+j_0)(k_{m-l}2^{m-l})} \tag{29}$$

we obtain successive arrays

$$A_l[(j_0, ..., j_{l-1}, k_{m-l-1}, ..., k_0)] = \sum_{k_{m-l}} A_{l-1}[(j_0, ..., j_{l-2}, k_{m-l}, ..., k_0)] W_N^{(j_{l-1}2^{l-1}+...+j_0)(k_{m-l}2^{m-l})} \tag{30}$$

for $l = 1, 2, 3, ..., m$
Writing out the sum, we get

$$\begin{aligned} A_l[(j_0, .., j_{l-1}, k_{m-l-1}, .., k_0)] &= A_{l-1}[(j_0, .., j_{l-2}, k_{m-l-1}, .., k_0)] + \\ (-1)^{j_{l-1}} i^{j_{l-2}} A_{l-1}[(j_0, .., j_{l-2}, k_{m-l-1}, .., k_0)] &W_N^{(j_{l-3}2^{l-3}+..+j_0)2^{m-l}} \end{aligned} \tag{31}$$

where $j_{l-1} = 0, 1$
This array is stored in the location whose index is

$$j_0 2^{m-1} + .. + j_{l-1} 2^{m-l} + k_{m-l-1} 2^{m-l-1} + .. + k_0)$$

3 Data and methodology

Code for fast fourier transform and inverse fast fourier transform were written in python and were applied on about five seconds (2^{19} samples) of five different audio clips that contained chirps of different cricket species. A representation of the a sample audio signal is given below

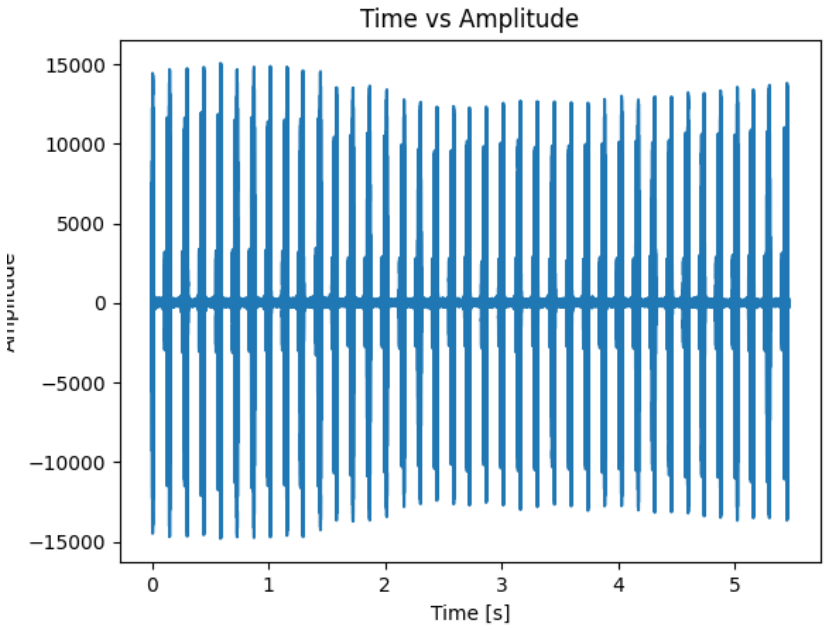


Figure 1: The waveform of about five seconds (2^{19} samples)of chirp of grylloides sigilitus. Here the amplitude measured is relative amplitude

3.1 Features of the dataset

The dataset comprised of five .wav audio files of cricket chirps. They are available in the github repository of the project. These are provided by Dr. Rittik Deb's lab in the School Of Biological Science at NISER. The audio is trimmed so that the number of samples is a power of 2, suitable for processing with the FFT algorithm.
The sampling rate of the audio file is 96 KHz.

3.2 Methodology

The algorithm for FFT works efficiently for 2^M sample points where M is a natural number. For the base case of the algorithm, ie, there is only one sample, the algorithm returns the sample itself, without any processing.
For the recursive case, that is, if the number of samples is greater than one, it divides the input data sample sequence into two subsequences: even - containing the elements at even indices; odd - containing the elements at odd indices. It then recursively applies the FFT algorithm to both even and odd subsequences.

After obtaining the FFT of the even and odd subsequences, it combines the results to compute the overall FFT of the input sequence. The algorithm computes the "twiddle factor"

$$T_k = e^{\frac{-i2\pi k}{N}} \tag{32}$$

for $k = 0$ to$\frac{N}{2} - 1$. Twiddle factor adjusts the phase of each frequency component during the FFT calculation. It represents the rotation in the complex plane corresponding to the frequency index k. Twiddle factor is multiplied with the FFT of the odd subsequence and the FFT of even subsequence is added to this to form the FFT of the input data sequence.
Performing FFT on input data sequence returns a sequence of complex numbers. Frequency spectrum is obtained by plotting the magnitude of this sequence of complex numbers against frequency.
To test the validity of the FFT algorithm as well as to understand better, Inverse FFT is performed on the FFT of the input data sequence. Inverse FFT returns a sequence of list of complex numbers representing the signal in the time domain, reconstructed from its frequency domain representation.It is then compared with the original input audio.

4 Results

FFT is performed on the input audio and IFFT is performed on the FFT of the input audio. It is discerned that FFT is indeed a fast algorithm.

4.1 Frequency spectrum of the input audio

After performing FFT on 2^{19} samples of each audio file, their frequency spectrum is plotted and the carrier frequency is identified in each case.

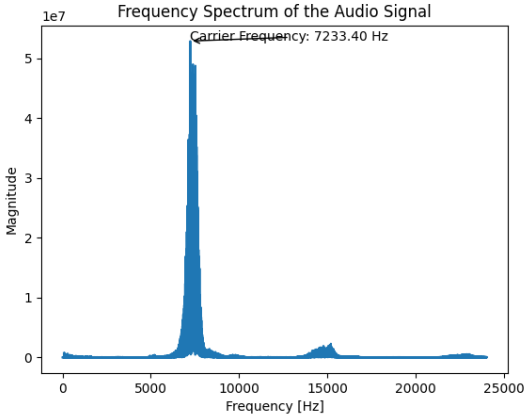


Figure 2: Frequency spectrum obtained using FFT of (2^{19} samples) of an audio clip containing cricket chirp. Carrier frequency = 7233.4 Hz

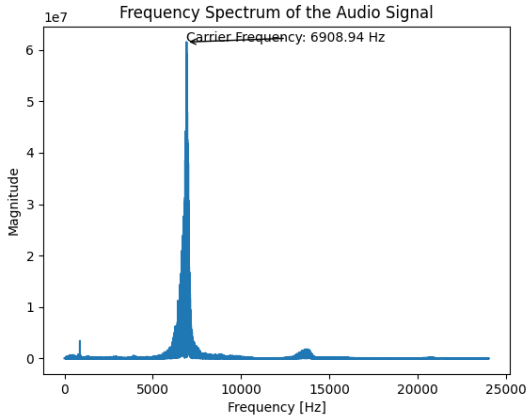


Figure 3: Frequency spectrum obtained using FFT of (2^{19} samples) of an audio clip containing cricket chirp of gryllodes sigillatus.Carrier frequency = 6908.94 Hz

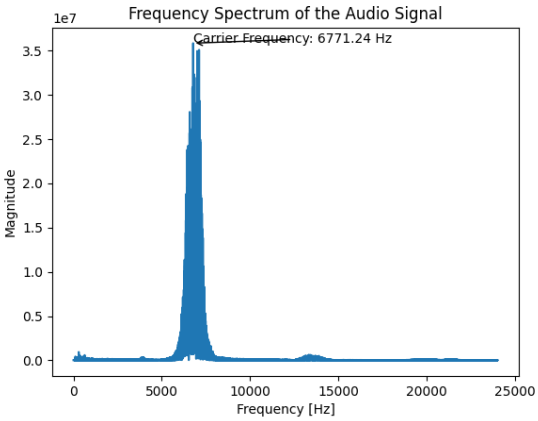


Figure 4: Frequency spectrum obtained using FFT of (2^{19} samples) of an audio clip containing cricket chirp of another gryllodes sigillatus.Carrier frequency = 6771.24 Hz

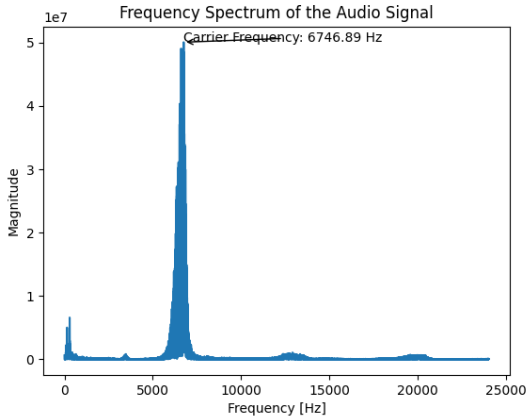


Figure 5: Frequency spectrum obtained using FFT of (2^{19} samples) of an audio clip containing cricket chirp of another gryllodes sigillatus.Carrier frequency = 6746.89 Hz

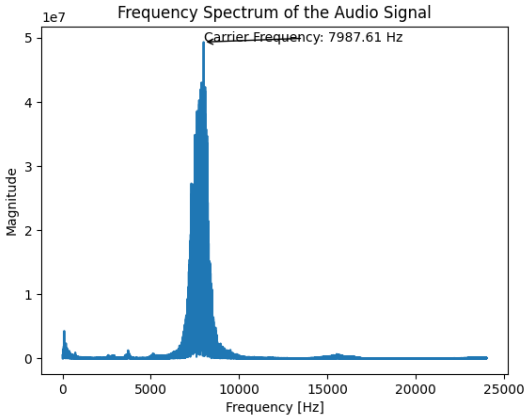
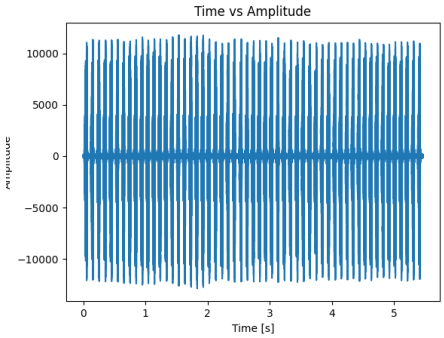


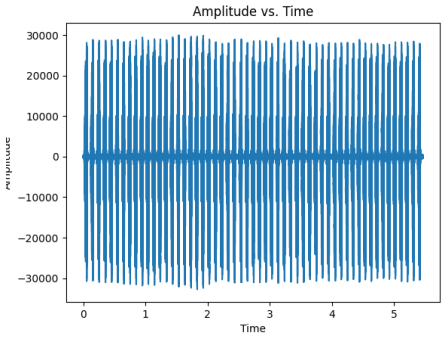
Figure 6: Frequency spectrum obtained using FFT of (2^{19} samples) of an audio clip containing cricket chirp of another gryllodes sigillatus. Carrier frequency = 7987.61 Hz

5 Comparing the IFFT with the input audio

The audio files obtained after performing Inverse FFT on the input data is found to be similar upon qualitative comparison.

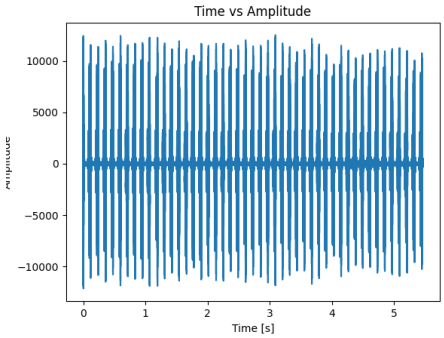


(a) Input audio waveform

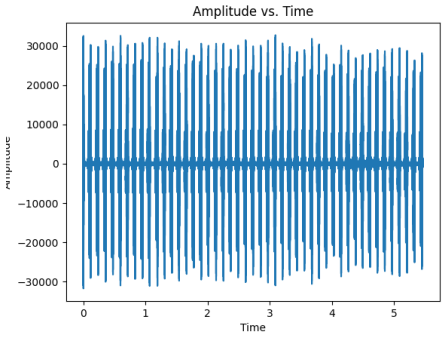


(b) Inverse FFT output audio waveform

Figure 7: Comparing the waveforms

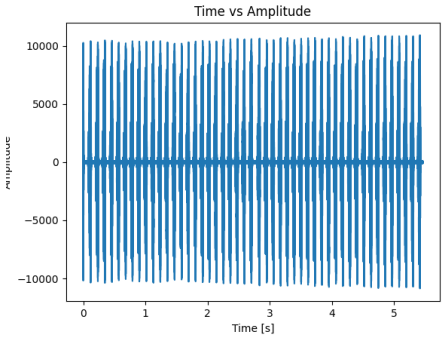


(a) Input audio waveform

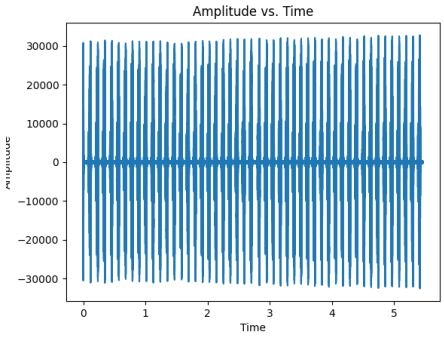


(b) Inverse FFT output audio waveform

Figure 8: Comparing the waveforms

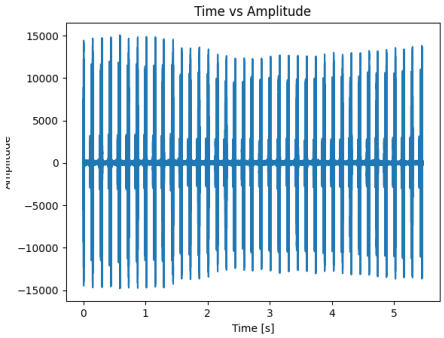


(a) Input audio waveform

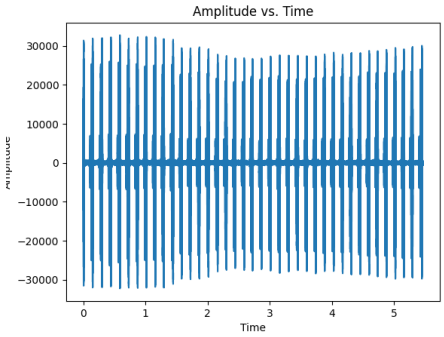


(b) Inverse FFT output audio waveform

Figure 9: Comparing the waveforms

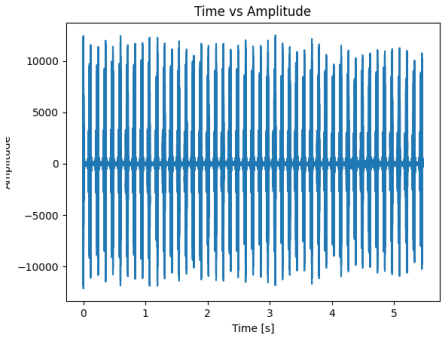


(a) Input audio waveform

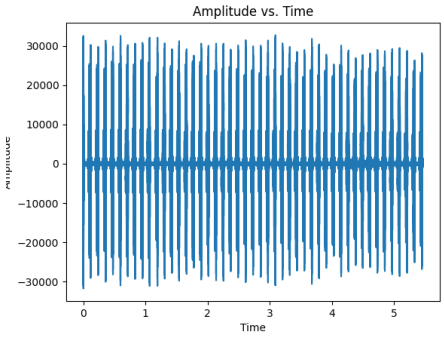


(b) Inverse FFT output audio waveform

Figure 10: Comparing the waveforms

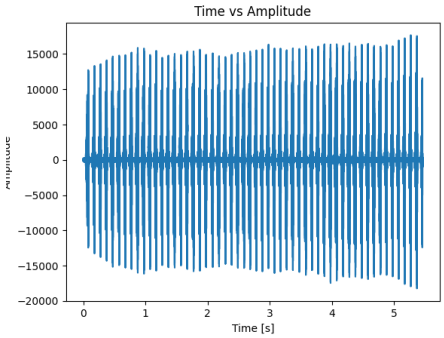


(a) Input audio waveform

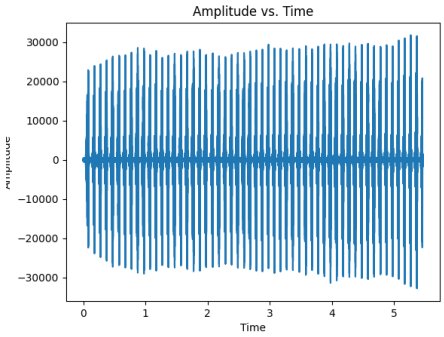


(b) Inverse FFT output audio waveform

Figure 11: Comparing the waveforms



(a) Input audio waveform



(b) Inverse FFT output audio waveform

Figure 12: Comparing the waveforms

It can be concluded that the waveforms are almost similar and hence the algorithm works well.

6 Conclusion

FFT algorithm is understood and is applied on a time series data. It is a very efficient algorithm.

References

[1] Brad Osgood. “Lecture notes for EE 261 the fourier transform and its applications”. In: 2002.