# Fourier Transform

By Chris Hyorok Lee

**Fourier Transform** lies at the core of Audio Signal Processing, and it is widely used in applications such as but not limited to: Speech Recognition, Music Information Retrieval, Audio Effects, and much more.

In this paper, I will explore the definition and the meaning behind Fourier Transform, and look at some of its application in the context of Audio Signal Processing.

The questions I am going to try to answer are:

1. How is calculus used in Fourier Transform?
2. How are different forms of Fourier Transform used in Audio Signal Processing?

# Fourier Transform

Fourier Transform is a powerful equation that can change audio signals into frequency. In music specifically, fourier transforms is the key component that changes an analog music signal, which is continuous, into a discrete frequency that can be processed in computers.
But first, I want to start by looking at the definition of the Fourier Transform formula and decipher different parts of the formula.

## Complex Fourier Transform

In Complex Fourier Transform, the equation is **integrating over time** and the output is both magnitude spectrum and phase spectrum. However, many times, only the magnitude spectrum is used for analysis and visualizations (Lerch, 2022). Essentially, the Fourier Transform changes wave form in the **time domain** to its frequency form in the **frequency domain.**
Because time and frequency is continuous in the real world, the Complex Fourier Transform integrates infinitely, and also contains the imaginery numer within its equation.

$$\hat{g}(f) = \int_{-\infty}^{\infty} g(t) \cdot e^{-i2\pi f t}\, dt$$

where $f = frequency$ and $t = time$
$e^{-i2\pi f t}$ represents the pure tone, and it represents a sinusoidal signal(sine wave) with a frequency of $f$. In the complex plane, it traces a unit circle in the clockwise direction as time $t$ increases. Imaginary number is crucial here, because it indicates rotation in the complex plane.
$g(t)$ represents the original signal in this case, so the original signal $g(t)$ is being multiplied to the pure tone at each point in time $t$.
The integration from negative infinity to positive infinity is basically adding up all the contributions over a continuous amount of time.
Calculus is at the center of this equation, and the integral plays the key role of adding a continuous element to Fourier Transform.

## Inverse Fourier Transform

$$g(t) = \int_{-\infty}^{\infty} cf \cdot e^{i2\pi f t}\, df$$

where $f = frequency$ and $t = time$

In the Inverse Fourier Transform, the equation is integrating across frequency, and adding up all the sinusoids.Then we can go back to the original signal.
The Complex Fourier Transform and the Inverse Fourier Transform can be performed interchangeably, as many times as needed. This way, we can find the frequency from a certain waveform over time, or find the original signal/waveform over time from a given frequency as much as we would like.

## Discrete Fourier Transform (DFT)

Although we can find frequency over time using Complex Fourier Transform, computers are not able to process complex numbers. This is where the Discrete Fourier Transform comes in.

In Continuous Fourier Transform, the output $g(t)$ represents a continuous analog signal. But in Discrete Fourier Transform (DFT), the output becomes $x(n)$ which represents a discrete digital sample at a point in time.

$$x(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{i2\pi n \frac{k}{N}}$$

where $f = frequency$ and $n = discrete\ points$

Another main difference between FT and DFT is the use of summation $\sum_{n=0}^{N-1}$ as opposed to an integral $\int_{-\infty}^{\infty}$. This is because in DFT, we are taking snapchats of a continuous signal and approximating the area under the signal rather than using integrals for infinite calculations.

$n$ represents number of different points that are taken from the continuous signal, and you can see a visualization below on Figure 1.

$k/N$ represents the frequency, or the frequency index relative to total number of samples in the signal.

```
import numpy as np
import matplotlib.pyplot as plt


#sampling information
fs = 200;delta = 1.0/fs; oversample = 1000
#signal freqency
f = 5.01; omega = 2*np.pi*f
#define time data sampled
t = np.arange(0, 1, delta)
#quasi continuous time series for smooth curve plot
ts = np.arange(0, 1, delta/oversample)
#define some voltages
Vs = np.sin(omega*t)
Vss = np.sin(omega*ts)
#Plot the sampled data and the smooth curve on a graph with axes labels
plt.plot(t, Vs, '.') #plot the sampled values
plt.plot(ts, Vss, 'r') #plot the quasi smooth values
plt.xlim(0, 1) #set plotting range
plt.xlabel('Time/s')
plt.ylabel('Amplitude/V')
```

```
Text(0, 0.5, 'Amplitude/V')
```

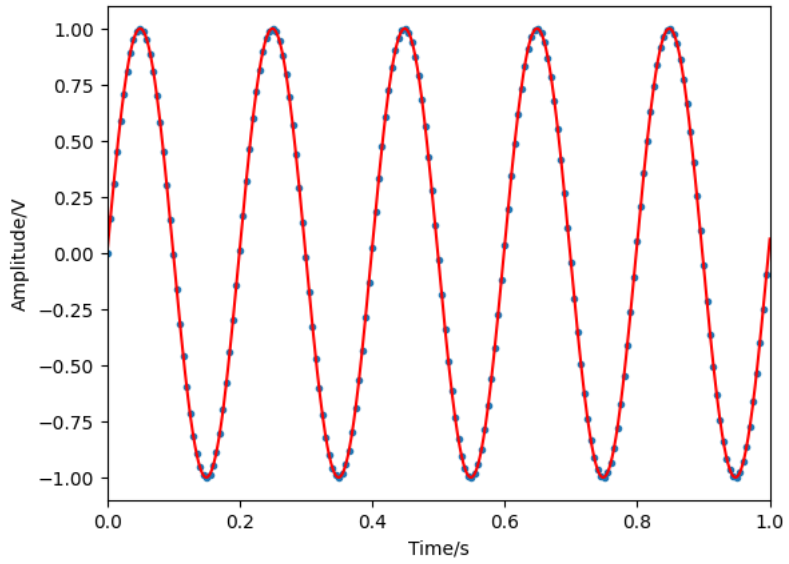


Figure 1. Visualization of discrete number of points taken from a continuous signal or a sine wave.

```
def f(x):
    return np.sin(x)
x_min = 0
x_max = 2 * np.pi
num_intervals = 50
dx = (x_max - x_min) / num_intervals
x = np.linspace(x_min, x_max, 1000)
y = f(x)
plt.figure(figsize=(8, 6))
plt.plot(x, y, label='f(x) = sin(x)', color='blue')
riemann_sum = 0
for i in range(num_intervals):
    x_left = x_min + i * dx
    x_right = x_min + (i + 1) * dx
    area = dx * f((x_left + x_right) / 2)
    riemann_sum += area
    plt.fill_between([x_left, x_right], [f((x_left + x_right) / 2), f((x_left + x_right) / 2)], alpha=0.1)
plt.title('Riemann Sum Approximation')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()
```

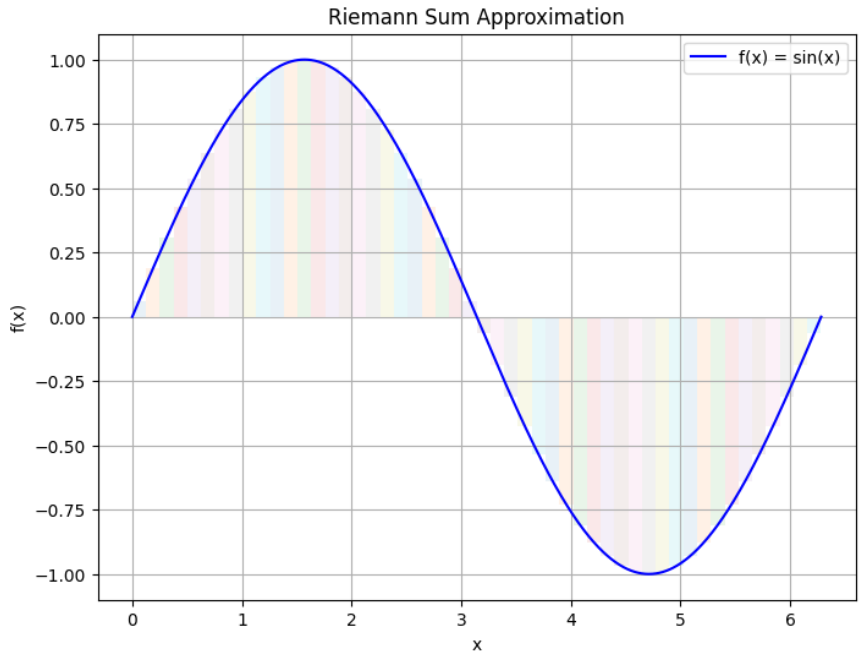


Figure 2. Diagram of approximating area under sine wave.

As shown in Figure 2, conceptually, DFT and Riemann Sum share many similarities, in that they both find the area under a graph.

## Fast Fourier Transform (FFT)

Fast Fourier Transform(FFT) is essentially an optimized implementation of DFT. FFT works when the N (number of samples) is a power of 2, and it exploits redundancies across sinusoids.
The mathematical formula of FFT is the same as DFT, with a key difference that the FFT algorithm uses a much more computationally efficient way of exploiting the symmetries and properties of the DFT.

## Short Time Fourier Transform (STFT)

STFT is one of the most crucial formula for working with audio using computers. STFT enables us to create spectrograms, which is a key component in Audio Signal Processing and AI music.
While FFT only allows us to create a "still image" of all the average frequency of the entire duration, STFT allows us to get a "video" of the frequency over the entire time domain.

STFT is done by applying FFT to small segments - frames of the signal across the time domain. This is done by adding other components like windowing and adding hop size.
**Windowing** is using multiple frames in an entire duration of the sound, to apply FFTs over overlapping frames.
**Hop size** tells us how many samples we slide over to the right for a new frame.

$$S(m,k) = \sum_{N-1}^{n=0} x(n+mH) \cdot w(n) \cdot e^{i2\pi n \frac{k}{N}}$$

This STFT formula has many similarities to a DFT. The key differences between the two formulas include:

- $S(m,k)$ is the output frequency with two parameters - $m$ being the frame number and $k$ being the frequency.
- $x(n+mH)$ indicates the signal in the current frame. $mH$ is the starting sample of the current frame where $H$ is the hop size.
- $w(n)$ is the windowing function

## Applications of Fourier Transform in Audio Signal Processing

In a larger sense, FT(Fourier Transform) "decomposes a complex sound into its frequency components" (Velardo, 2020). FT is able to do this by **comparing its original signal with sinusoids(sine waves)** of various frequencies, and for each frequency, we can get a magnitude and a phase. High magnitude shows high similarity between the signal and a sinuoid.
To demonstrate the DFT in use, I am going use both the numPy and librosa library to change a waveform of an audio file into a frequency graph and a spectrogram.

```
!pip install librosa
```

```
from google.colab import drive
from google.colab import files
from IPython.display import Audio
import librosa
# Mount Google Drive
drive.mount('/content/drive')
# Download the file using its ID from Google Drive
file_id = '1BpmXI5b9FOIN62CB1V7-aaBAw4c2xfun'  # Replace this with your file ID
file_name = 'audio_file.wav'  # Rename the file as required
# Download the file
!gdown --id $file_id -O $file_name
y, sr = librosa.load(file_name)
Audio(data=y, rate=sr)
```

```
Mounted at /content/drive
/usr/local/lib/python3.10/dist-packages/gdown/cli.py:121: FutureWarning: Opti
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=1BpmXI5b9FOIN62CB1V7-aaBAw4c2xfun
To: /content/audio_file.wav
100% 1.06M/1.06M [00:00<00:00, 128MB/s]
```

0:00 / 0:04

```
import librosa.display
# Load audio from the downloaded file
y, sr = librosa.load(file_name)
# Get the time indices for the samples
time = np.arange(0, len(y)) / sr
# Plot the waveform
plt.figure(figsize=(10, 4))
plt.plot(time, y, color='blue')
plt.title('Audio Waveform')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
```

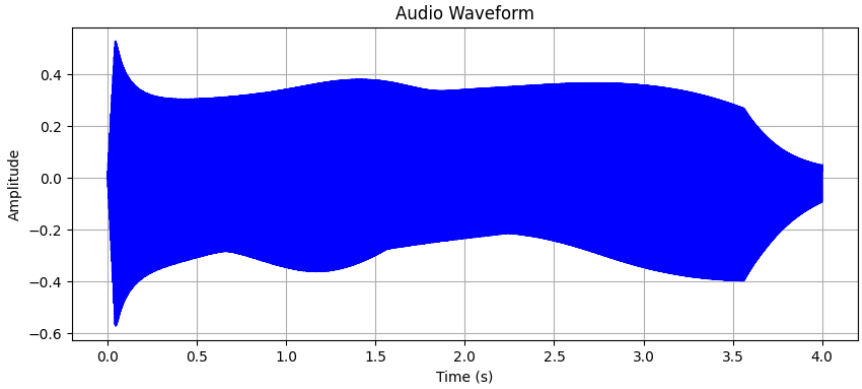


Figure 3. Waveform of a basic sine wave synthesizer playing a C3 note.

This figure shows a basic amplitude over time, a waveform, of a basic sine wave synth playing one a single note for around the duration of 4 seconds.
You can clearly see the attack and a relatively long sustain of the synth and a slight decay of the synth towards the end.

```
# Calculate the FFT
fft = np.fft.fft(y)
magnitude = np.abs(fft)
frequency = np.linspace(0, sr, len(magnitude))
# Plot the FFT
plt.figure(figsize=(10, 4))
plt.plot(frequency[:len(frequency)//2], magnitude[:len(magnitude)//2])
plt.title('Fast Fourier Transform (FFT)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()
```

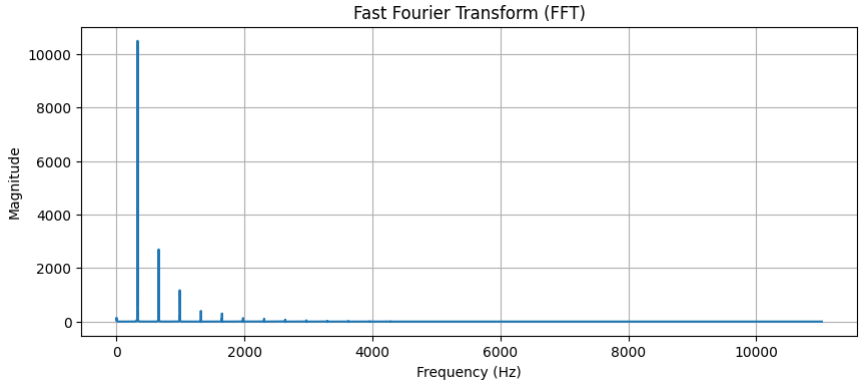


Figure 4. Magnitude over Frequency graph of sine wave synthesizer playing a C3 note.

Here you can see the highest magnitude on its fundamental frequency of 130.81 Hz, which is C3 note. To the right of the fundamental frequency, you can see small magnitudes of its higher overtones.
This is done by using FFT, or the Fast Fourier Transform under the hood. The downside of this graph done by FFT, is that we only get the average frequency of the entire duration, and cannot see the gradual change of frequency over time.

```
# Calculate the spectrogram
spec = np.abs(librosa.stft(y))
# Convert to dB scale
spec_db = librosa.amplitude_to_db(spec, ref=np.max)
# Display the spectrogram
plt.figure(figsize=(10, 6))
librosa.display.specshow(spec_db, sr=sr, x_axis='time', y_axis='linear')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.show()
```

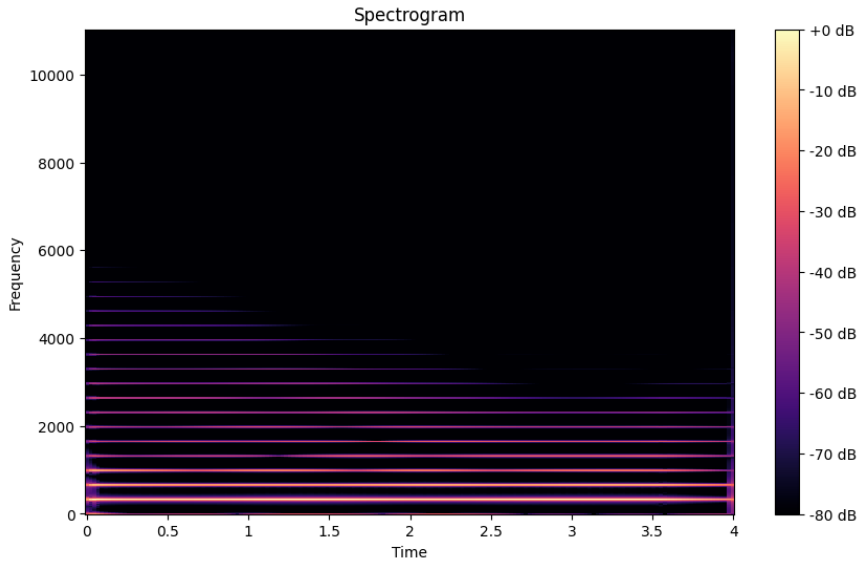


Figure 5. Spectrogram of a sine wave synthesizer playing a C3 note.

Here you can see a detailed view of the overtones when the C3 note is played on the sine wave synethesizer. You can see that the fundamental frequency of 130.81 (C3) is the brightest and the most prevalent through the entire graph, while its upper overtones get weaker as the frequencies are higher.
This spectrogram gives the frequency over time, where we can visualize all the different frequencies and timbre across the entire duration of the recording, as opposed to averaging the entire frequency like the magnitude over frequency graph.
This is done by STFT, or the Short Time Fourier Transform by librosa.

## Conclusion

In this paper, I looked at Fourier Transforms and its different versions like DFT, FFT, and STFT. The Calculus behind all these formulas is the idea of integration and summing up areas under the graph.
Fourier Transform is at the core of Audio Signal Processing and the field of Music Information Retrieval, where groundbreaking research is done largely using FFT or STFT.
Looking ahead, I want to take the knowledge that I have gathered through Calculus and its applications in Audio Signal Processing, and conduct further research in the field of Music Information Retrieval and Music Generation using Machine Learning and Deep Learning.

# Citations


1. Music of AI series by Valerio Velardo
   Velardo, Valerio. Audio Signal Processing for Machine Learning, 2020. YouTube, https://www.youtube.com/playlist?list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0.
2. 17 Equations That Changed the World by Ian Stewart
   Stewart, Ian. In Pursuit of the Unknown : 17 Equations That Changed the World, Basic Books, 2012. ProQuest Ebook Central, https://ebookcentral-proquest-com.libproxy.newschool.edu/lib/newschool/detail.action?docID=876406.
3. An Introduction to Audio Content Analysis by Alexander Lerch
   Lerch, Alexander. An Introduction to Audio Content Analysis, John Wiley & Sons, 2022.

4. Basic of Fourier Analysis of Time Series Data: A practical guide to use of the the Fourier transform in an industrial setting.
   Tipton, Carl.Basic of Fourier Analysis of Time Series Data: A practical guide to use of the the Fourier transform in an industrial setting, Johnson Matthey Technology Review, 2022.
5. Fourier Transform by Wolfram Mathworld.
   Weisstein, Eric W. "Fourier Transform." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/FourierTransform.html


Google Colab online version of the paper: https://colab.research.google.com/drive/11qUiNejPN7ctgPMtsOlUDuQVh3VP9O5d?usp=sharing