Published in Towards Data Science

mlearnere Follow
Sep 30, 2020 · 4 min read · ✦ · Listen

Learning from Audio: Fourier Transformations

Breaking down a fundamental equation in signal processing

Related articles:

- Learning from Audio: Wave Forms
- Learning from Audio: Time Domain Features
- Learning from Audio: Spectrograms
- Learning from Audio: The Mel Scale, Mel Spectrograms, and Mel Frequency Cepstral Coefficients
- Learning from Audio: Pitch and Chromagrams

Introduction:

In Wave Forms, we looked at what waves are, how to visualize them, and how to deal with null data.

In this article, I aim to develop an intuition on what the Fourier Transformation is, why it is useful when studying audio, show mathematical proofs to make it computationally efficient, and visualize the results. **The data we are working with in this (and related) articles can be found on my GitHub repository for this series.**

With this in mind, let's begin. We will first initialize our necessary variables and packages below:

Recall from Wave Forms:

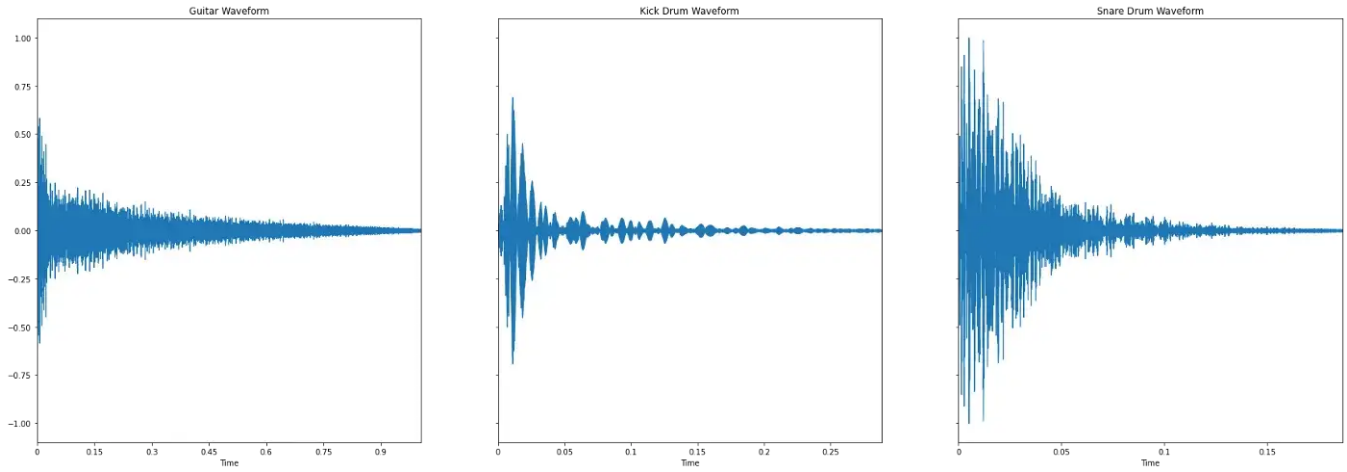

Figure 1 — Image by Author

Fourier Transformation:

What is the Fourier Transformation? Why is it Important?

When looking at the figure above, we can see various spikes of amplitudes creating a unique pattern to the unique sound. This is because all complex sounds — like the sounds of our voices — are really just a sum of many sine and cosine signals. The `gif` below presents a nice visualization of the objective:

By Lucas V. Barbosa — Own work, Public Domain

It is important to note that by using this transformation, we will be translating the audio from the time-domain to the frequency-domain. Here are some key points about the two:

- The time domain looks at the variation of the signal's amplitude over time. This is useful for understanding its physical shape. In order to plot this, we need time on the x-axis and amplitude on the y-axis. The shape gives us a good idea of how loud or quiet the sound will be.
- The frequency domain observes the constituent signals our recording is comprised of. By doing this, we can find a sort of "fingerprint" of the sound. In order to plot this, we need frequency on the x-axis and magnitude on the y-axis. The larger the magnitude, the more important that frequency is. The magnitude is simply the absolute value of our results from the FFT.

Discrete Fourier Transformation (DFT)

The DFT, which is what `numpy` builds off of, is as follows:

$$\hat{x}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega_N^{nk}$$

where

$$\omega_N = \exp\left(\frac{-2\pi i}{N}\right)$$
$$k = 0, 1, \ldots, (N-1)$$

Equations from Derek L. Smith, University of California, Santa Barbara.

When learning about the Fourier Transformation, you may see a continuous version — which utilizes an integral as opposed to a sum — suitably called the Continuous Fourier Transform. The discrete version of the transformation is generally used for computers' operations. In fact, even in the discrete form, most computers still lack the proper computational power to solve for the transformation as the raw equation below. The DFT has some nice properties that allow for computational ease, known as the Fast Fourier Transformation.

Fast Fourier Transformation (FFT)

The FFT is as follows:

$$\hat{x}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{m-1} x_{2n} \omega_m^{nk} + \frac{\omega_N}{\sqrt{N}} \sum_{n=0}^{m-1} x_{2n+1} \omega_m^{nk}$$

Equation from Derek L. Smith, University of California, Santa Barbara.

Do not let the notation worry you. In the Discrete Fourier Transformation, you are dealing with a sum of products x, and omega. This equation splits sum of products into two — one along the odd indices and another along the even. This procedure above can be modeled much more efficiently with computer processes as opposed to the DFT.

Now, let's visualize what the FFT looks like. In our application, we will be using the `numpy.fft.fft` function and apply it to our sound waves.

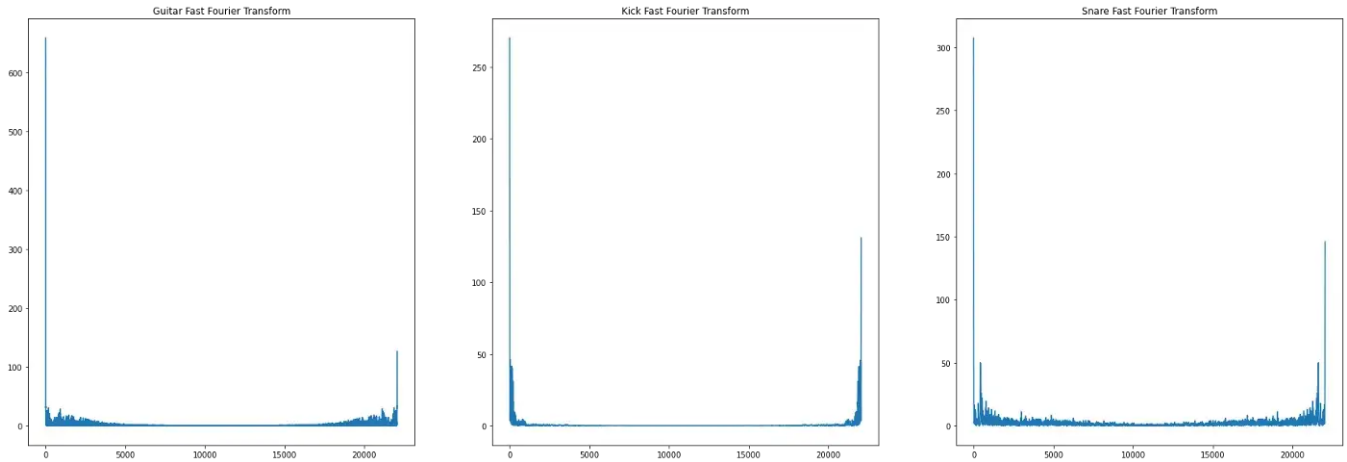

Figure 2 — Image by Author

Again, there are some slight problems with the visualizations. However, this is a quite easy fix as the DFT has a symmetric property. This makes sense as our visualization on the left half mimics the visualizations on the right half. We will slice the arrays in half and visualize from there.

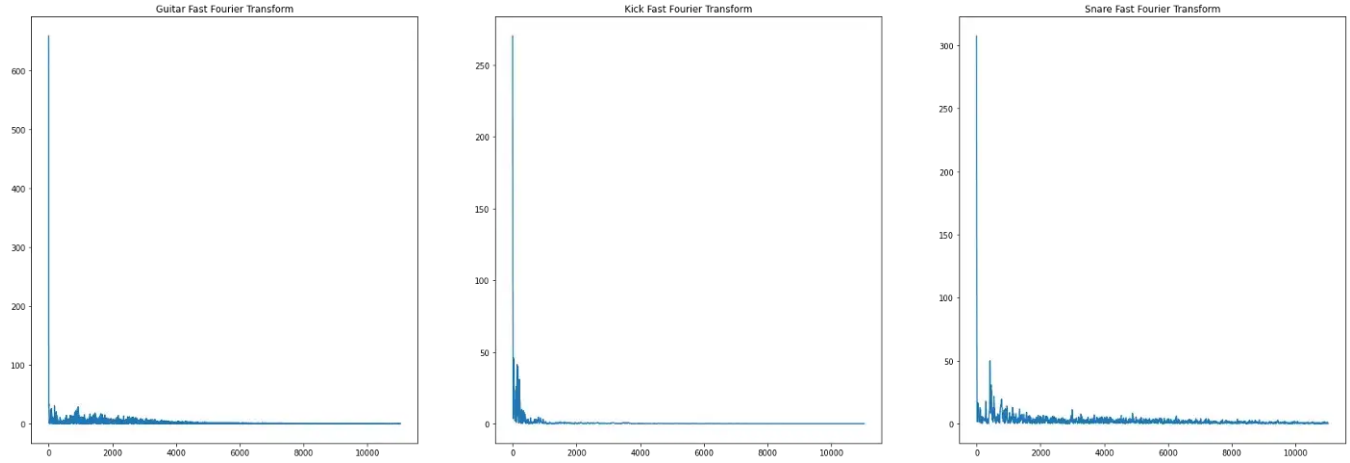

Figure 3 — Image by Author

Conclusion:

In this article, you now should be able to extract the Fourier Transformation of any complex signal of any type and find its unique fingerprint.

Stay tuned for upcoming articles as they will look at deeper processing and visualization techniques that allow us to learn even more from audio.

Thank you for reading.

Signal Processing Data Science Mathematics Engineering Machine Learning