**Transforming audio signals to the frequency domain**

In order to analyze audio signals, we need to understand the underlying frequency components. This gives us insights into how to extract meaningful information from this signal. Audio signals are composed of a mixture of sine waves of varying frequencies, phases, and amplitudes.

If we dissect the frequency components, we can identify a lot of characteristics. Any given audio signal is characterized by its distribution in the frequency spectrum. In order to convert a time domain signal into the frequency domain, we need to use a mathematical tool such as the Fourier Transform. If you need a quick refresher on the Fourier Transform, check out this link: [http://www.thefouriertransform.com](http://www.thefouriertransform.com/). Let's see how to transform an audio signal from the time domain to the frequency domain.

**Create a new Python file and import the following packages**:

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

Read the input audio file using the wavefile.read method. It returns two values – sampling frequency and the audio signal:

**# Read the audio file**

sampling\_freq, signal = wavfile.read('spoken\_word.wav')

Normalize the audio signal:

***# Normalize the values***

signal = signal / np.power(2, 15)

Extract the length and half-length of the signal:

***# Extract the length of the audio signal***

len\_signal = len(signal)

***# Extract the half length***

len\_half = np.ceil((len\_signal + 1) / 2.0).astype(np.int)

Apply the Fourier transform to the signal:

**# Apply Fourier transform**

freq\_signal = np.fft.fft(signal)

Normalize the frequency domain signal and take the square:

***# Normalization***

freq\_signal = abs(freq\_signal[0:len\_half]) / len\_signal

*# Take the square*

freq\_signal \*\*= 2

**Adjust the Fourier-transformed signal for even and odd cases:**

***# Extract the length of the frequency transformed signal***

len\_fts = len(freq\_signal)

***# Adjust the signal for even and odd cases***

if len\_signal % 2:

freq\_signal[1:len\_fts] \*= 2

else:

freq\_signal[1:len\_fts-1] \*= 2

**Extract the power signal in dB:**

*# Extract the power value in dB*

signal\_power = 10 \* np.log10(freq\_signal)

Build the *X* axis, which is frequency measured in kHz in this case:

***# Build the X axis***

x\_axis = np.arange(0, len\_half, 1) \* (sampling\_freq / len\_signal) / 1000.0

Plot the figure:

**# Plot the figure**

plt.figure()

plt.plot(x\_axis, signal\_power, color='black')

plt.xlabel('Frequency (kHz)')

plt.ylabel('Signal power (dB)')

plt.show()

The full code is given in the file frequency\_transformer.py. If you run the code, you will see the following screenshot:

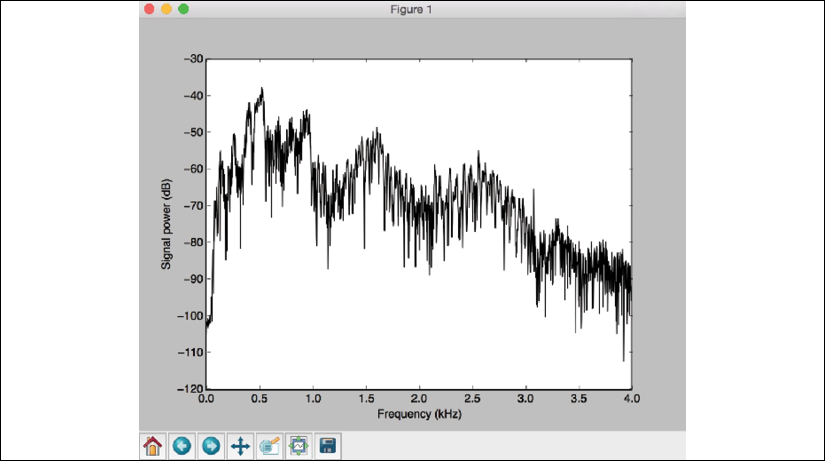


Figure 3: Visualization of audio signal transformation

The preceding screenshot shows how powerful the signal is across the frequency spectrum. In this case, the power of the signal goes down in the higher frequencies.