Understanding Data

Digital Audio Processing with Python

Part 2: The Discrete Fourier Transform

Aliasing

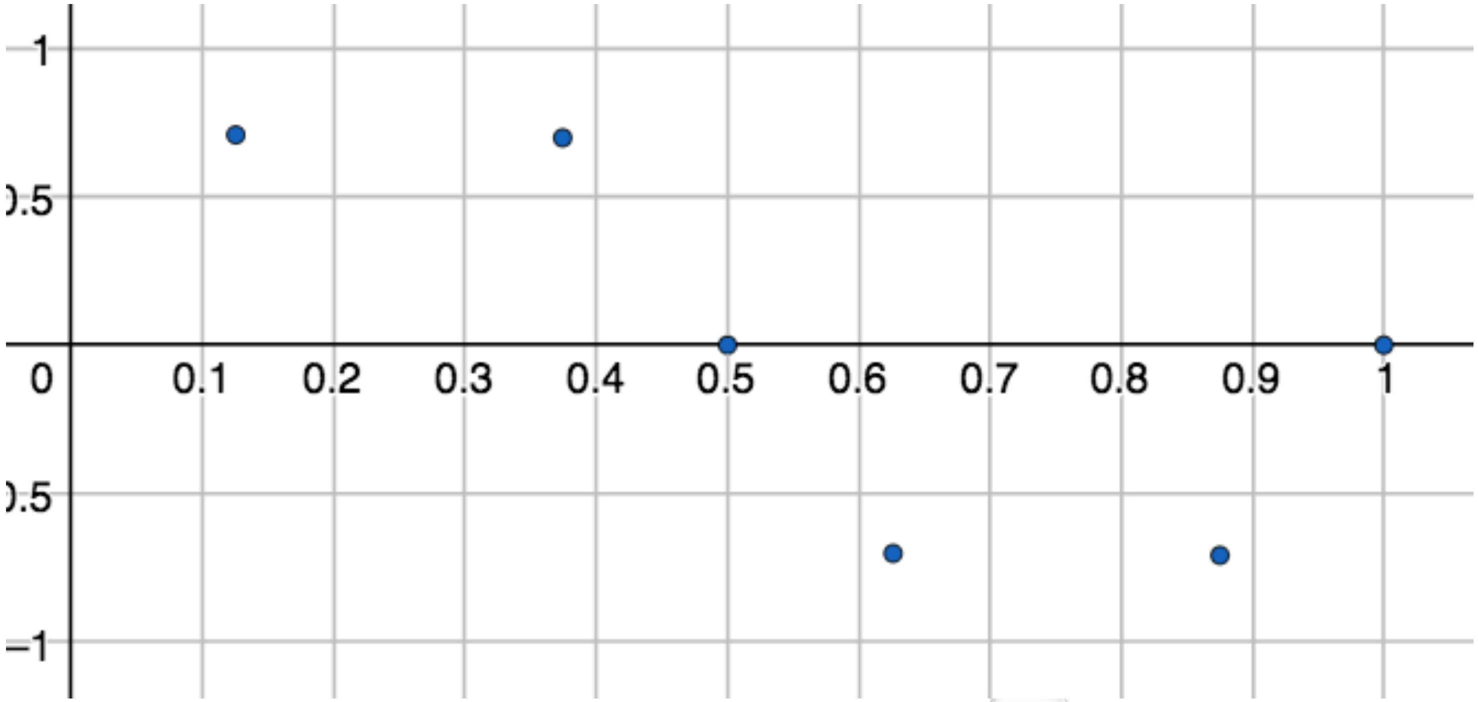
Given a continuous signal. Our discrete, sampled data, is approximate data.

When we only evaluate the signal at discrete times, we lose information on what happened between the samples.

This leads to a frequency ambiguity known as **aliasing**.

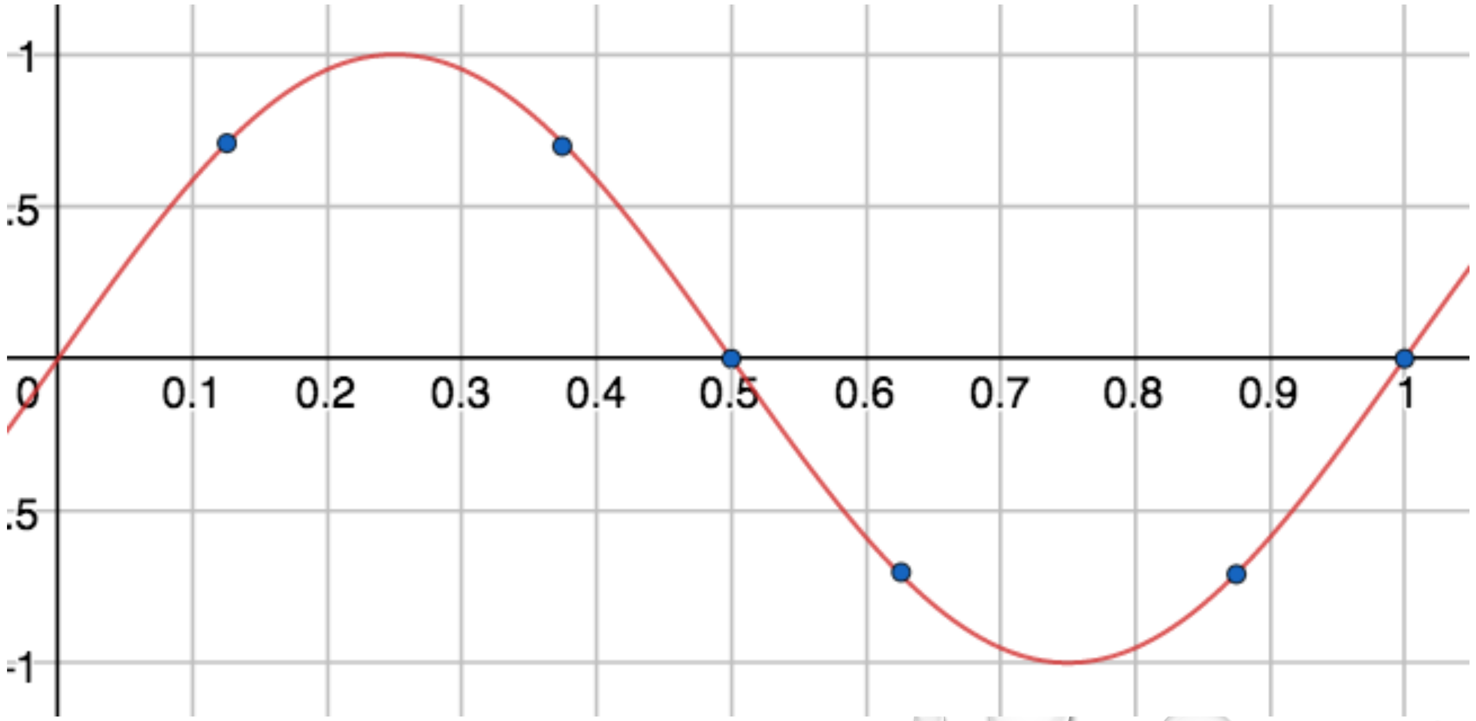
Aliasing

6 samples are taken from some signal.



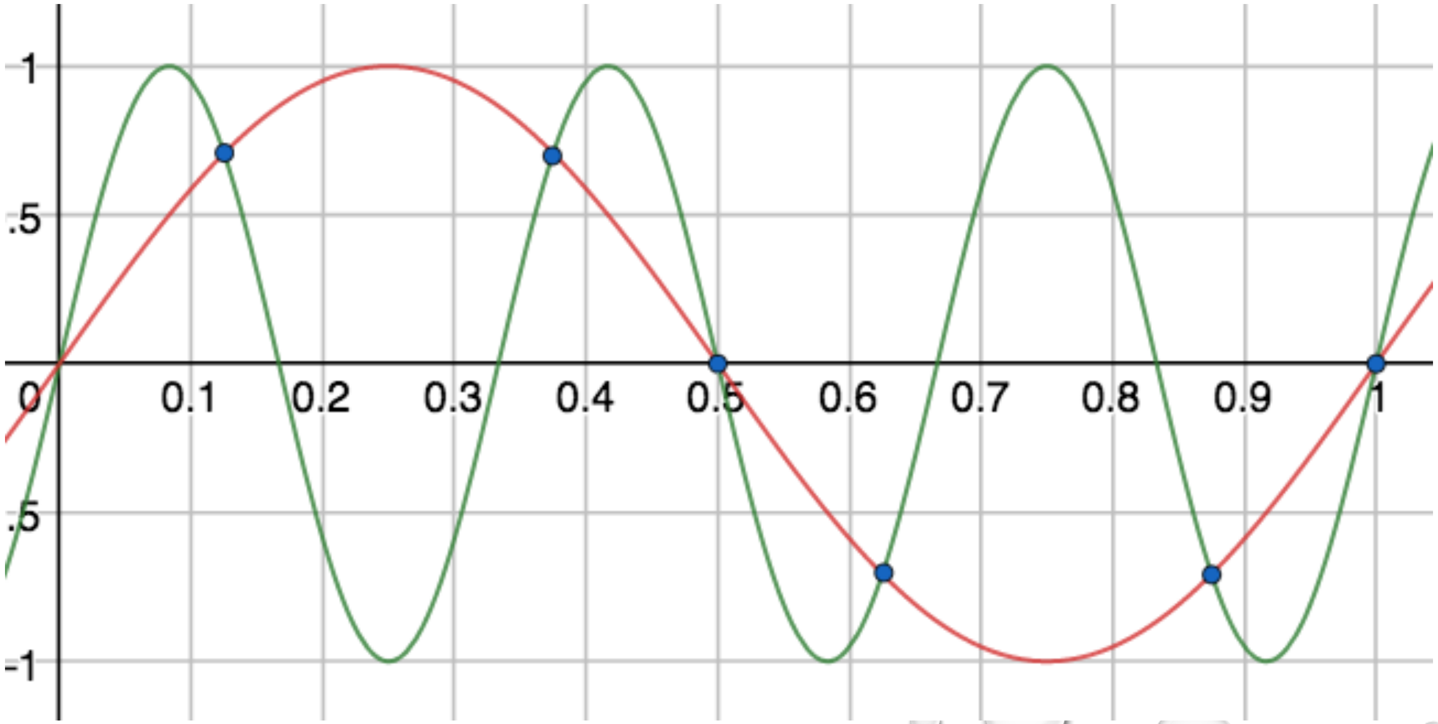
Aliasing

The original signal could be this red signal.



Aliasing

Or this green signal. Aliasing is this ambiguity.



The Sampling Theorem

The Sampling Theorem: A signal $y(t), t \in [0, L]$, can be perfectly reconstructed from its samples taken at the sampling rate f_s provided that signal contains ONLY frequencies less than $f_s/2$.

The frequency $f_s/2$ is called the **Nyquist frequency or the folding frequency.**

Why 44100 Hz?

Range of human hearing is between 20 Hz and 20,000 Hz.

To guarantee that every frequency in this range is properly recorded, the Sampling Theorem states that a sampling rate of at least 40,000 Hz is necessary. CD quality standard sampling rate is 44100 Hz.

Before sampling is taken, a **lowpass-filter** is applied to remove all frequencies outside of the Nyquist frequency range. This prevents aliasing and corruption of the recording. These lowpass-filters are also called **anti-aliasing filters**.

Reconstruction of a Signal

Consider a 1-second signal given by

$$y(t) = \sin(2\pi \cdot 1t) + \sin(2\pi \cdot 3t) + \sin(2\pi \cdot 4t), t \in [0, 1]$$

with frequencies 1Hz, 3 Hz and 4 Hz.

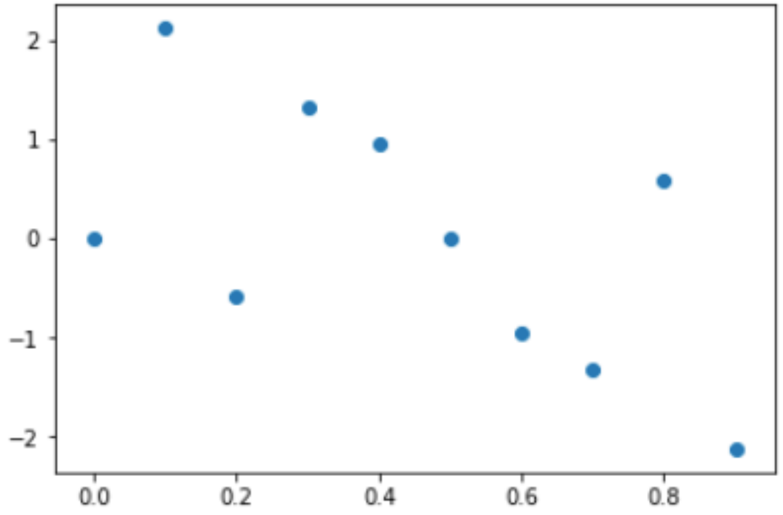
Since the highest frequency component is 4 Hz, the sampling theorem says that this signal can be reconstructed without information loss if we sample, for example, at the sampling rate of 10 Hz.

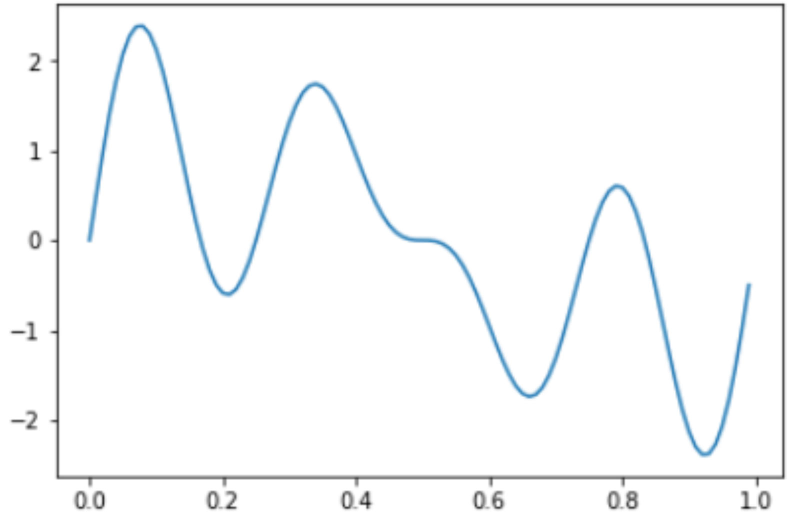
Reconstruction of a Signal

Consider a 1-second signal given by

$$y(t) = \sin(2\pi \cdot 1t) + \sin(2\pi \cdot 3t) + \sin(2\pi \cdot 4t), t \in [0, 1]$$

with frequencies 1Hz, 3 Hz and 4 Hz.





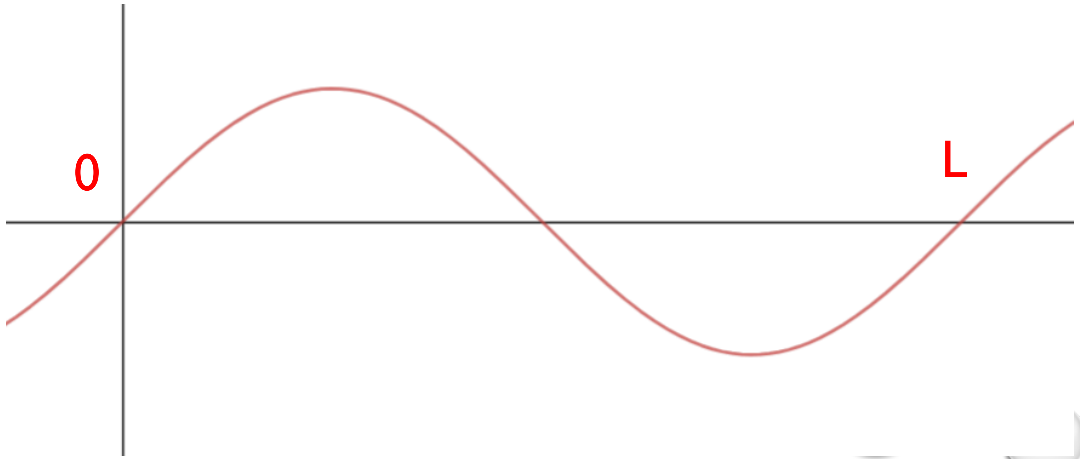
Fundamental Frequency

Consider the interval [0, L] measured in seconds. Define the frequency

$$f_1 = 1/L.$$

This frequency is called the **fundamental frequency** for the interval [0, L]. A sinusoid with frequency f_1 completes one cycle in the interval [0, L]. Note that since $N = f_s L$, we also have

$$f_1 = 1/L = f_s/N.$$



Multiples of the Fundamental Frequency

If we sample a signal $y(t), t \in [0, L]$ and obtain a total of N samples, the Discrete Fourier Transform will give us N coefficients that detects the presence of the following N harmonics:

$$\{0, f_1, 2f_2, \ldots, f_k = kf_1, \ldots, (N-1)f_1\}$$

which are integer multiples of the fundamental frequency f_1.

Spectrum of a Real Signal is Symmetric

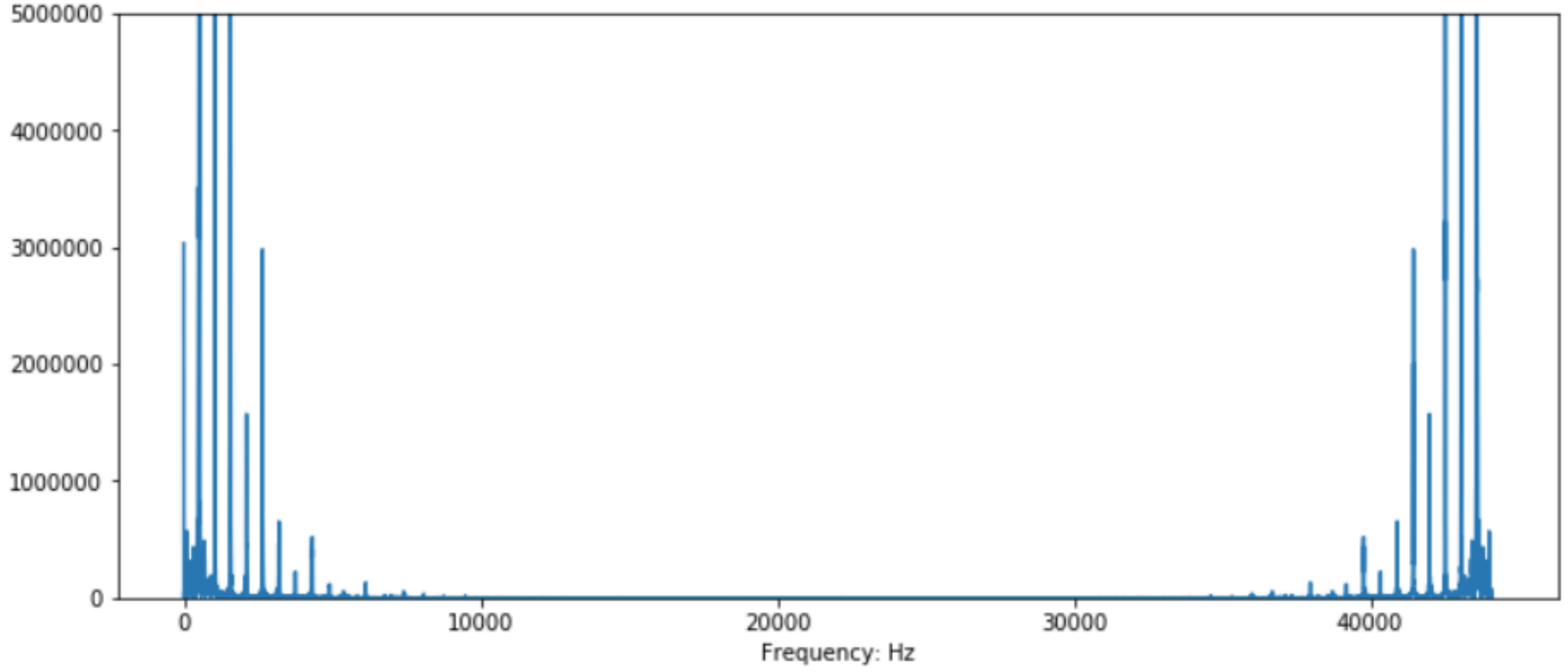
A signal $y(t), t \in [0, L]$ is a **real signal** if y(t) are real values. For example, audio signals are real signals.

The spectrum of every real signal $y(t), t \in [0, L]$ is symmetric around 0 Hz. That is, if a frequency f Hz is present in the signal, so is the frequency -f Hz.

The signal
$y(t) = \sin(2\pi \cdot 1t) + \sin(2\pi \cdot 3t) + \sin(2\pi \cdot 4t), t \in [0, 1]$

contains the frequencies: {-4Hz, -3 Hz, -1 Hz, 1 Hz, 3 Hz, 4 Hz}.

Symmetric Spectrum of a C5 on a Piano



The Discrete Fourier Transform

If we sample a signal at a sampling rate fs, the Discrete Fourier Transform can only detect frequencies between

$-fs/2$ and $f_s/2$.

Because a real signal has a **symmetric spectrum**. The DFT can detect frequencies between

0 and $f_s/2$.

The frequencies outside this range are **redundant** and does not contribute to our analysis of the signal.

Example

Suppose we have an audio signal for 2 seconds,

$$y(t), t \in [0, L] = [0, 2].$$

Sample at this signal at the rate fs = 4 Hz for a total of N = 8 samples.

The fundamental frequency is $f_1 = 1/L = 1/2 \text{ Hz}$.

Note that the 8 harmonics in Hz are:

$$\{f_0 = 0 \text{ Hz}, f_1 = 1/2 \text{ Hz}, f_2 = 1 \text{ Hz}, \ldots, f_7 = 7/2 \text{ Hz}\}.$$

Example

The 8 harmonics are {0 Hz, 0.5, 1, 1.5, 2, 2.5, 3, 3.5 }.

Since fs = 4 Hz, the Nyquist frequency is 2 Hz.

Aliasing: Frequencies above the Nyquist is **folded over(by subtracting the sampling rate fs)**.

{0, 0.5, 1, 1.5, 2, 2.5 - 4, 3 - 4, 3.5 - 4} =

{0, 0.5, 1, 1.5, 2, -1.5, -1, -0.5}

Symmetric Spectrum of a real signal(remove redundant frequencies):

{-1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2} = {0, 0.5, 1, 1.5, 2}

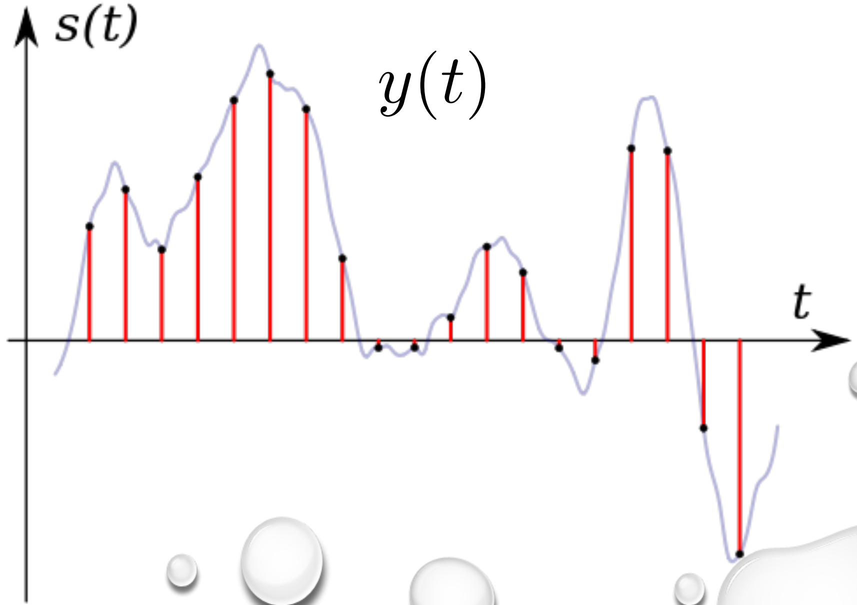
The Discrete Fourier Transform

Sampling An Analog Signal

Given a signal $y(t), t \in [0, L],$ sample the signal at some sampling rate fs for a total of N samples:

$$[y[0], y[1], \ldots, y[N-1]]$$

CD Quality(fs = 4Hz)



The Discrete Fourier Transform

Given an analog signal $y(t), t \in [0, L],$ sample it at the rate of fs to obtain $f_s L = N$ samples

$$[y[0], y[1], \ldots, y[N-1]].$$

The **Discrete Fourier Transform (DFT)** is a frequency detector.

Let the fundamental frequency $f_1 = 1/L = f_s/N$. The DFT detects the presence of the following frequencies in the signal,

$$\{0, f_1, 2f_2, \ldots, f_k = kf_1, \ldots, (N-1)f_1\}$$

which are integer multiples of f_1 .

The Discrete Fourier Transform

The **Discrete Fourier Transform (DFT)** converts the sequence of N samples

$$[y[0], y[1], \ldots, y[N-1]]$$

into another sequence of N complex numbers called **Fourier coefficients**

$$[Y[0], Y[1], \ldots, Y[N-1]]$$

where $Y[k]$ expresses the degree to which the frequency $f_k = kf_1$ is present in the signal.

The Discrete Fourier Transform

The **Discrete Fourier Transform** converts $[y[0], y[1], \ldots, y[N-1]] \rightarrow [Y[0], Y[1], \ldots, Y[N-1]]$ by

$$Y[k] = \sum_{n=0}^{N-1} y[n] e^{-i2\pi kn/N}, k = 0, 1, \ldots, N-1$$

where i is the square root of -1 and $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ and $Y[k]$ expresses the degree to which the frequency $f_k = kf_1$ is present in the signal. `(Python) np.fft.rfft(samples)`

The Discrete Fourier Transform

A simple implementation of the DFT uses a for loop to sum a term-by-term product between the sample and the exponential to compute each Fourier coefficient.

To compute all coefficients, one can use a nested for loop, one for each coefficient.

However, such an implementation is VERY slow. The Fast Fourier Transform is a fast implementation of the computing the DFT.

Because of its many applications, the Fast Fourier Transform is considered one of the Top 10 Algorithms of 20th Century by the IEEE magazine Computing in Science & Engineering.

Numpy's FFT

We will use Numpy's Fast Fourier Transform implementation in our next Jupyter notebook lab to analyze audio clips.

`np.fft.rfft(samples)` returns the Fourier coefficients given the samples. The "r" means that our signal is real and so it only return half of the coefficients ignoring the redundant half.

`np.fft.fft(samples)` returns the full set of Fourier coefficients.

References

1) Müller, Meinard, Fundamentals of Music Processing, Springer 2015.

2) Downey, Allen, ThinkDSP, Green Tea Press 2012.

3) Smith, Julius, The Mathematics of the Discrete Fourier Transform, W3K Publishing 2007.

4) Loy, Gareth, Musimathics, Volumes 1 and 2. The MIT Press 2011.

5) Newman, Mark, Computational Physics, Createspace Independent Publishing Platform 2012.

6) Soklaski, Ryan. MIT Lincoln Lab Researcher. Beaver Works Summer Institute.