

Cog*Works Week 1 Notes

Capstone: Song Recognition Project

7/12/22

Learning:

- Basic Physics of Sound

Air consists of O₂ and N₂. Each of these molecules is 1.5x10⁻¹⁰m diameter. In a room, these are usually evenly dispersed. This is caused by electron repulsion. The molecules bounce off each other due to electron repulsion which keeps them evenly distributed on average. The spacing between two molecules is roughly 460x its diameter.

Air Pressure is the measure of the average rate and strength of which air molecules collide against a surface, measured per unit area. Air Pressure is measured in Pascals, [P] – Force/Area.

Formula

$$P_h = P_0 e^{\frac{-mgh}{kT}}$$

P_h = pressure at height h

P_0 = sea level pressure

g = acceleration due to gravity

k = Boltzmann's constant (ideal gas constant divided by Avogadro's number)

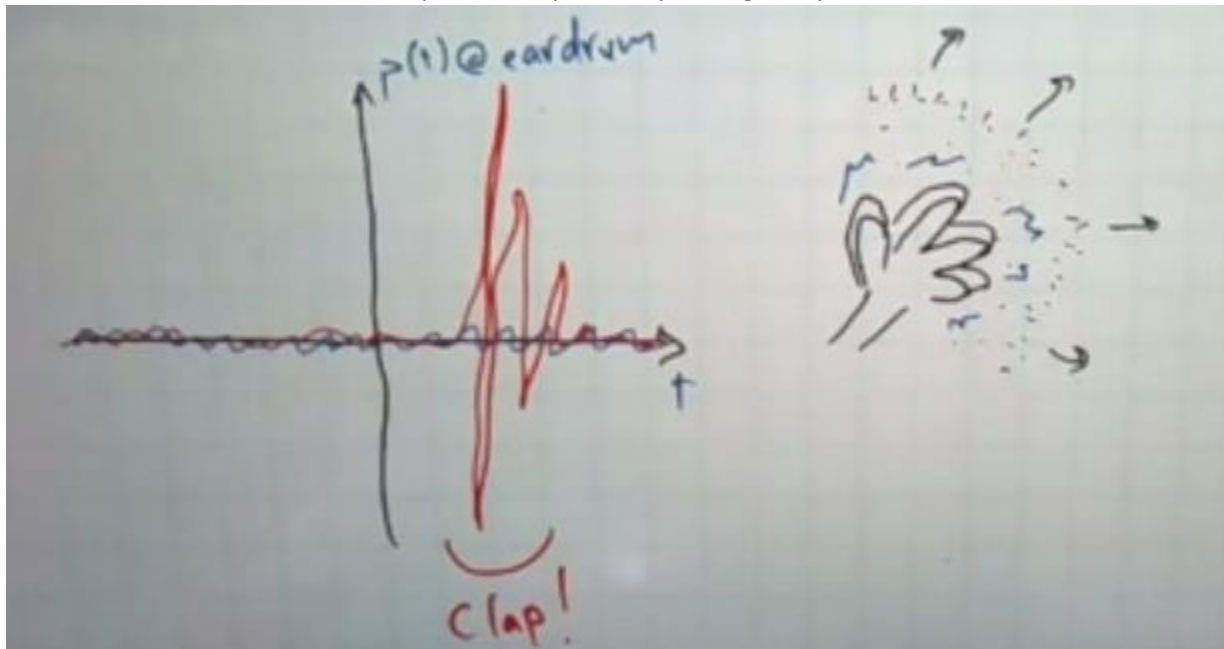
T = absolute temperature

m = mass of one air molecule

We can adjust air pressure by:

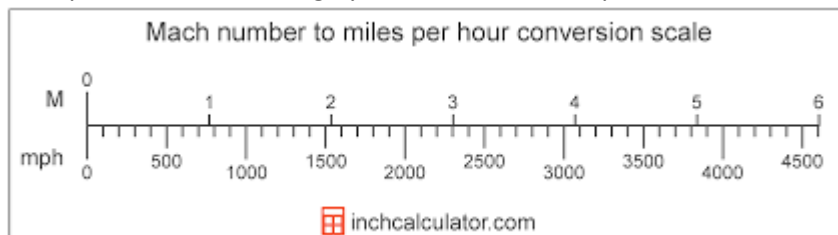
- Increasing the density of the air by changing the number of molecules
- Increasing the temperature to excite the particles
- Shrink the space the particles are in

Our ears can sense fluctuations in pressure. Specifically, changes in pressure over time at our eardrum.



The clap in this example generated a region of compression where the air molecules are densely packed together, and a region of rarefaction where the region is at a lower pressure. This, in a sense, creates a “soundwave” of particles for your ear to process. A soundwave is a perceptible fluctuation in air pressure in time and space. The graph above shows a “temporal waveform”, which is a depiction of this pressure change demonstrated in a graph of time instead of space.

The speed of sound is roughly 343 m/s, or ~761 mph. Here is a Mach scale:

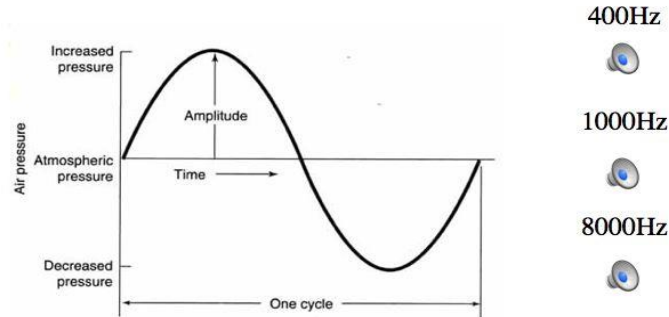


~Continued~

Quantifying Sound

Pure Tone – A pressure wave that oscillates sinusoidally in space and time.

Pure tones



Pure tones can be described by 2 numbers:

Frequency = rate of air pressure modulation (related to pitch)

Amplitude = sound pressure level (related to loudness)

Period of Pure Tone:

$$T = \frac{1}{f}$$

Temporal Waveform of pure tone:

$$P(t) = A \cos(2\pi f t - \varphi)$$

A = Amplitude

- Units in Pascals

F = Frequency of Pure Tone

- Number of repetitions per second
- Units in Hertz

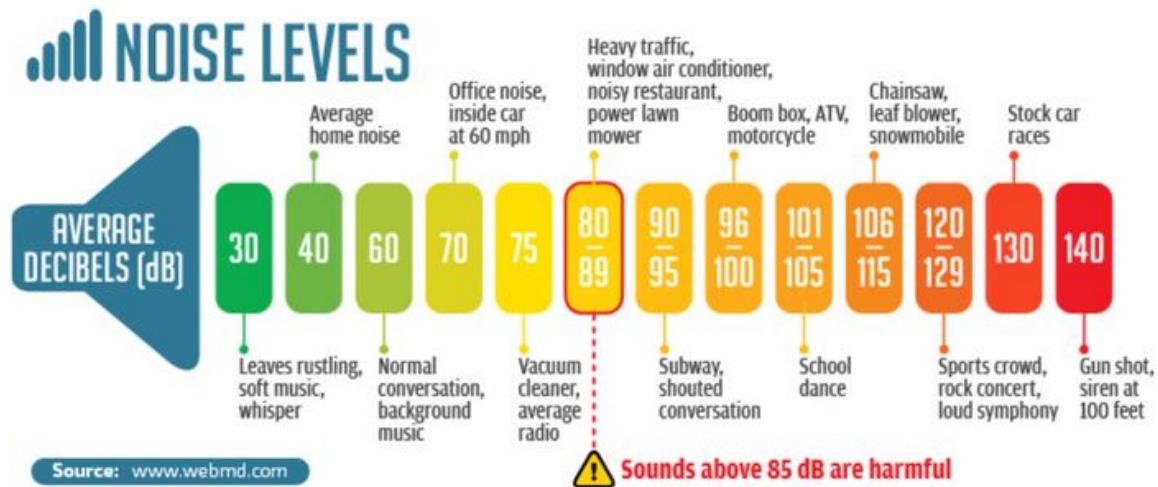
φ = Phase-Shift

- Units in radians
- Not perceptible
- Allows us to describe any pure tone no matter where the pressure starts

Loudness is logarithmic – if you double the amplitude of a pure tone, you do not hear a pure tone twice as loud. You have to apply operations by factors of 10 to the sound to cause effects.

Decibels: \log_{10} amplitudes

50dB -> 100dB: x316 Amplitude



<https://rsokl.github.io/CogWeb/audio.html>

Audible Frequency Ranges

The lowest frequency we can hear is 20 Hz, and the highest frequency is 20,000 Hz.

The wavelength is labelled as λ (lambda).

20,000 Hz -> 1.72cm

20Hz -> 17.2cm

Audio Signal Basics

def pressure(t):

Fs: sample rate – at what regularity do we evaluate the function

Δt is the spacing between our samples

#Td – duration in seconds

To determine how many samples to take of the function, you have to multiply the sampling rate by the duration and take the floor of the number and add 1.

$N_{\text{samples}} = \text{floor}(F_s \cdot T_d) + 1$

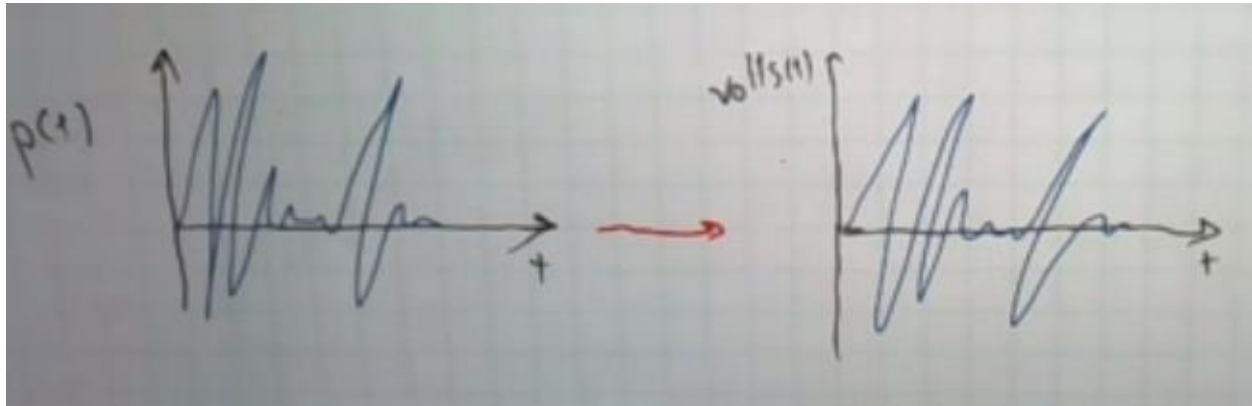
$\text{times} = \text{np.arange}(N_{\text{sample}}) * \Delta t$

Recording Sound with Microphones

- A signal is a phenomenon that carries information about an event.
- An analog signal is specifically a continuously varying quantity. A soundwave is an example of an analog signal.
- A microphone's purpose takes the analog waveform and converts it to an analog electric signal

Sound Waveform in Pascals

Voltage Waveform Generated by Mic



A force on a charged particle is:

$$F = qE$$

If you place a wire in an electric field, the electrons in a wire will create a current. A voltmeter measures how strong the electric field is over a length of wire. The larger the wire, the larger the electric field potential. We can use our voltmeter to generate the voltage waveform by translating the fluctuations in air pressure to the changing in an electric field.

Faradays Law – A changing magnetic field in time will create a winding electric field around it.

A microphone has a membrane that is connected to a wire with a magnet. When air pressure reaches the membrane, it pushes the magnet attached to the coil of wire then read by the voltmeter to generate a voltage, therefore generating the voltage waveform.

- A voltmeter measures the electric field across a wire.

Digitizing an Analog Signal

- Temporal Sampling
- Quantization

Temporal Sampling – how frequently will you record this signal?

Quantization – How precise will you record this signal?

A WAVE file is a time and data file.

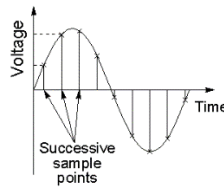
Pulse Code Modulation (PCM): Analog-To-Digital Conversion

- For this method, we need:

- A sampling rate: f_s
 - Measurements per second (Hz)
- Bit-depth: N_d
 - Refers to the number of bits of memory used to record each measurement
 - Corresponds to the precision of each measurement

Example:

If you have a signal of duration T_d , and a sampling rate of F_s , then the spacing between samples (in time) is $\Delta t = \frac{1}{f_s}$



$$N_{\text{samples}} = \text{floor}(T_d * f_s) + 1$$

In numpy, you use the arange function to generate samples spaced out by Δt

$N_{\text{samples}} = \# \text{ samples I need in total}$

$\Delta t = \text{Space between samples}$

$\text{Timevalues} = \text{np.arange}(N_{\text{samples}}) \Delta t$ #gives us the times to sample the audio/data spaced by Δt

Example:

$$\begin{array}{lcl}
 f_s = 0.5 \text{ Hz} & \left. \begin{array}{l} \\ \\ \end{array} \right\} & N_{\text{sample}} ? \\
 T_d = 127.2 \text{ s} & & \lfloor f_s T_d \rfloor + 1 \\
 & & \downarrow \\
 \Delta t = \frac{1}{f_s} = 2 \text{ s} & & \lfloor 63.6 \rfloor + 1 \\
 & & 63 + 1 \\
 & & 64
 \end{array}$$

Bit-Depth

- How precisely we want to record the voltage

1-bit = can record 2 states

2-bit = 4 states

4 bit = 8 states

16bit = 65,536 states/voltages

N-bits = $2^{n\text{-bits}}$ states

In the analog-to-digital notebook, on problem 1.3.2, its asking to make a function to sample a function and you need to use both temporal sampling and quantization.

DAY 3 - 7/13/2022

Fourier Analysis and Fourier Transforms

Fourier Analysis allows us to translate a voltage waveform to a histogram. The histogram allows us to see which frequencies are most prominent in the recording. This histogram is known as a Fourier Spectrum.



Fourier Series – is **a sum that represents a periodic function as a sum of sine and cosine waves**. The frequency of each wave in the sum, or harmonic, is an integer multiple of the periodic function's fundamental frequency. Each harmonic's phase and amplitude can be determined using harmonic analysis.

We will learn how to create a Fourier Spectrum using a DFT (Discrete Fourier Transformation).

You can derive from a Fourier Series what's called a discrete Fourier transformation. This is working with samples from a continuous function (This is having access to times and signals related to those times). The goal is to take the sample data from the continuous function and to generate a Fourier spectrum from that data.

Fourier Series (Continued)

Fourier Series: Represent some function $f(t)$ $t \in [0, T)$ in terms of an infinite sum of sinusoids

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} |a_k| \cos\left(2\pi \frac{k}{T} t - \psi_k'\right)$$

You can represent a function as an infinite sum of sinusoids. We can only find the sum when we restrict ourselves to a certain range.

By breaking down this formula, we can see a representation of the pure tone waveform formula.

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} |a_k| \cos\left(2\pi \frac{k}{T} t - \psi_k'\right) \quad t \in [0, T)$$

Need to find $(|a_k|)_{k=1}^{\infty}$ $(\psi_k')_{k=1}^{\infty}$

$$= \frac{a_0}{2} + |a_1| \cos\left(2\pi \frac{1}{T} t - \psi_1'\right) + |a_2| \cos\left(2\pi \frac{2}{T} t - \psi_2'\right) + |a_k| \cos\left(2\pi \frac{k}{T} t - \psi_k'\right) \rightarrow \text{pure tone}$$

amplitude: $|a_k|$
frequency: $\frac{k}{T}$
phase-shift: ψ_k'

The amplitude is often called a Fourier Coefficient, like the phase-shift. The Fourier Spectrum for $f(t)$ is frequency relative to amplitude. (Frequency X-Axis and Amplitude Y-Axis)

We are now transitioning to describing samples of a function, instead of the continuous function above.

We can describe our sampled data at a finite sum of range (0,N) (N excluded)

$$f(t_n) = y_n = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor} |a_k| \cos(2\pi \frac{k}{N} t_n - \varphi'_k) \quad n \in [0, N)$$

y_n : sample of $f(t)$ at $t = t_n$

Total of N samples at $t_n = \frac{n}{N} T$
where T is duration of signal

$t_0 = \frac{0}{N} T$
 $t_1 = \frac{1}{N} T$
 $t_{N-1} = \frac{N-1}{N} T$

Handwritten notes: "inclusion" and "excluded" with arrows pointing to the interval $n \in [0, N)$

Each of my N samples can be represented as a finite sum of sampled sinusoids.

$N/2 = \text{floor of } N/2$. Like if $N/2 = 3.5$, the floor would be 3.

Each sample has a space of Δt , which is calculated by T/N_{samples}

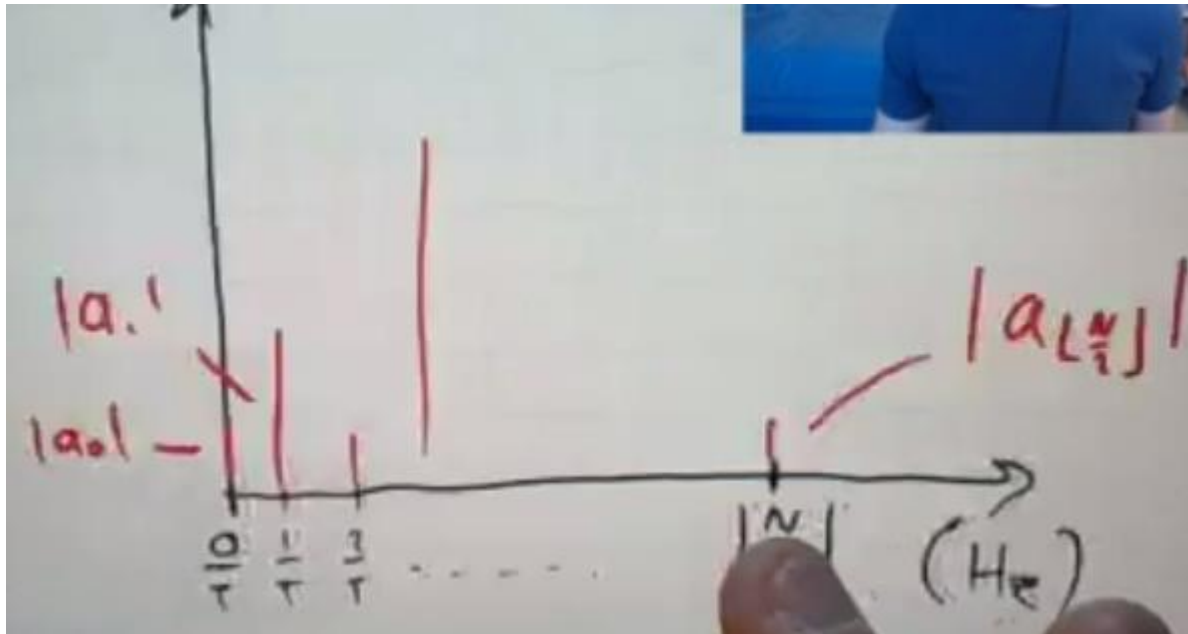
The above formula is known as the inverse discrete Fourier transformation. If you know your amplitudes $a_{\text{sub}k}$ and phase-shifts, you can give back all samples. This is going from your amplitude and phase-shifts to your data.

If you take your data and need to find the amplitudes and phase shifts, you use the discrete Fourier transformation

If you have amplitudes and phase shifts and need your samples, you use inverse discrete Fourier transformation.

$$(y_n)_{n=0}^N \xrightarrow{\text{DFT}} (|a_k|)_{k=0}^{\lfloor \frac{N}{2} \rfloor} (\varphi'_k)_{k=0}^{\lfloor \frac{N}{2} \rfloor}$$

Handwritten note: "IDFT" with a red arrow pointing from the frequency components back to the samples.



Reminder: In python to generate sample times it is:

```
np.arange(Nsamples)*delta-t <-(T/N)
```

Discrete Fourier Transform

Sample points t_n is equal to $\frac{n}{N}T$

$$\frac{k}{T} t_n = \frac{k}{T} \frac{n}{N} T = \frac{k}{N} n$$

Instead of writing $\cos(2\pi k/T t_n - \phi)$ you can write:

$$\begin{aligned} \cos\left(2\pi \frac{k}{T} t_n - \phi'_k\right) \\ \downarrow \\ \cos\left(2\pi \frac{k}{N} n - \phi'_k\right) \end{aligned}$$

Where n is just an integer.

$$e^{ix} = \cos(x) + i\sin(x)$$

$$\cos(x) = (e^{ix} + e^{-ix})/2$$

For each K in $(0, 1, 2, \dots, \text{floor}(N/2))$ the equation for the complex value Fourier coefficient $C_{\text{sub}K}$ is given by:

$$C_k = \sum_{n=0}^{N-1} y_n e^{-i2\pi \frac{k}{N} n}$$

Complex Fourier coeff
 C_k : complex-valued number
 $= R_k + iI_k$
 $C_k \rightarrow |a_k| \phi_k'$

With the complex value coefficients you can find the real values $a_{\text{sub-}K}$ and phase shifts.

We pick a value of K in $(0, 1, 2, \dots, \text{floor}(N/2))$ and have Y_n and pass it into the formula.

You can then take the value of the complex number and then can convert to $a_{\text{sub-}k}$ and $\phi_{\text{sub-}k}$

pick k
have $(y_n)_{n=0}^{N-1}$

$$C_k = \sum_{n=0}^{N-1} y_n e^{-i2\pi \frac{k}{N} n}$$

C_0
 \downarrow
 $|a_0| \phi_0'$

\uparrow
over all N samples

$\rightarrow \text{np.exp}(-2j + \text{np.pi} \times 0 \times 3/4)$

$$|a_k| = \begin{cases} 2 \frac{|C_k|}{N} & 1 \leq k < \frac{N}{2} \\ \frac{|C_k|}{N} & k=0 \quad k = \frac{N_{even}}{2} \end{cases} \quad |C_k| = \sqrt{R_k^2 + I_k^2}$$

The C-sub-k formula is to find the magnitude of the complex number. This is how we find A-sub-k^

To find Phi-sub-k:

$$\text{Phi-sub-k} = \arctan(-I\text{-sub-K}, R\text{-sub-k})$$

The two equations above are in the function `fourier_complex_to_real()`

Takes Complex-subk and returns an array of A-subk and phi-subk values.

Ultimately, we are going from samples from $(n/N)T$ positions and getting t_N and y_N and re-represent this as amplitudes and phase shifts for $K=0, 1, \dots, \text{floor}(N/2)$. The amplitude and phase shifts are amp and phase for pure tone of frequency $V_k = K/T$

$$\begin{array}{ccc} \frac{n}{N}T & n=0, 1, \dots, N-1 & V_k = \frac{k}{T} \\ \downarrow & & k=0, 1, \dots, \lfloor \frac{N}{2} \rfloor \\ t_n, y_n & \xrightarrow{\text{DFT}} & \underbrace{|a_k|, \varphi_k'}_{\text{amp + phase for pure tone of freq } V_k = \frac{k}{T}} \end{array}$$

The inverse of DFT (IDFT) takes amplitude and phi and gives us t_N and y_N .

Applied Example:

$$N=4 \quad y_0 = 1.25 \quad y_1 = 0.75, \quad y_2 = 0.25, \quad y_3 = -1.25$$

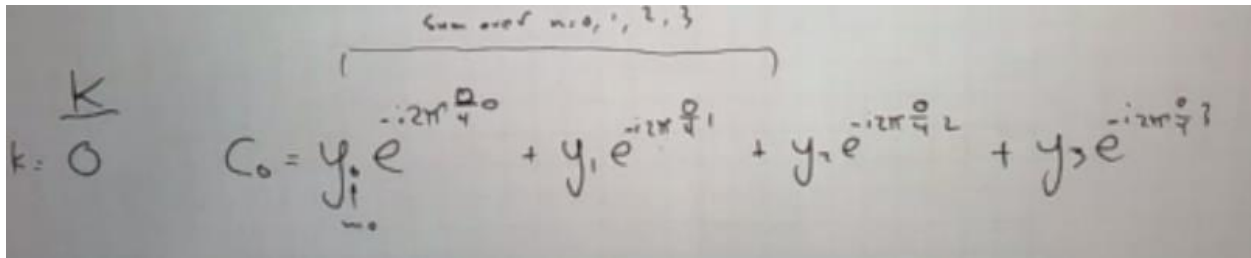
$$T = 2 \text{ seconds} \quad (C_k)_{k=0}^{(4)} = (C_k)_{k=0}^{(4)} \rightarrow C_0, C_1, C_2$$

you compute the 3 Complex value coefficients

First thing to do is pick a value of K and enter it into the complex number formula

K=0 Example:

Nsamples= 4, K=0, And sum over n=0,1,2,3



A handwritten equation on a piece of paper. On the left, it says 'K' over 'k=0'. To the right, there is a sum of four terms: $C_0 = y_0 e^{-i2\pi \frac{0}{4} \cdot 0} + y_1 e^{-i2\pi \frac{0}{4} \cdot 1} + y_2 e^{-i2\pi \frac{0}{4} \cdot 2} + y_3 e^{-i2\pi \frac{0}{4} \cdot 3}$. Above the sum, there is a bracketed note that says 'Sum over n=0, 1, 2, 3'.

So now, $C_{\text{sub}0} = y_0 + y_1 + y_2 + y_3 = 1$

For Csub0, it is always the sum of your data

After doing computations, we get:

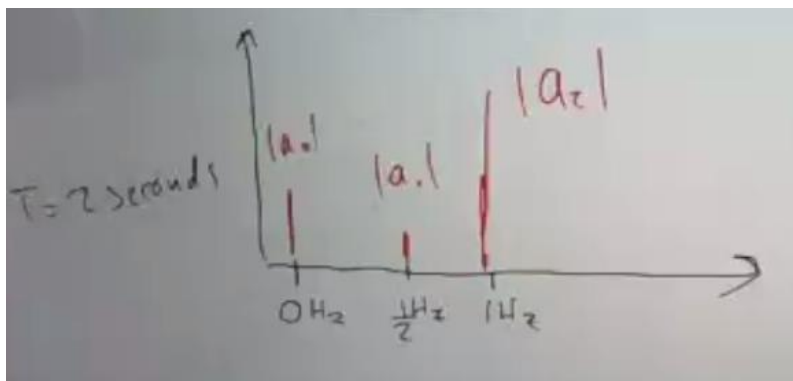
$C_0 = 1, C_1 = 1 - 2i, C_2 = 2$

We can then get the amplitude and phase-shift values for a_0, ϕ_0 , and so on.

When $k = 0, 1, 2$

$T = 2$

We have 0hz, 1/2hz, and 1hz for the frequency X-axis



Pseudocode for calculating K values (K represents the index associated with our different pure tones)
($V_k = K/T$)

```

for k in (0...  $\lfloor \frac{N}{2} \rfloor$ )
    Ck = 0
    for n in (0... N-1)
        Ck += yn np.exp(-j * 2 * pi * fk * n)
    Save Ck to list
return list of Ck values

```

In the above problem, we are iterating over every frequency and little-n corresponds to a time when we sample our function. The runtime efficiency of the above function is $O(n^2)$.

Example:

3min song @ 44100 Hz -> 7.6×10^6 samples to work with

So you have to do $(7.6 \times 10^6)^2$ operations in the above loops

Most processors can do 4×10^9 operations per second

So to do fourier transformation for the 3min song will take about 4 hrs!

$$\text{DFT: } (7.6 \times 10^6)^2 \text{ operations}$$
$$4 \times 10^9 \frac{\text{operations}}{\text{second}}$$
$$\frac{(7.6 \times 10^6)^2 \text{ operations}}{4 \times 10^9 \frac{\text{operations}}{\text{second}}} \approx 4 \text{ hours}$$

So, you need to do a Fast Fourier Transform (FFT) in complexity $O(n \log(n))$. Instead of nested for loops it does a “divide and conquer” method. So the same song would take 0.045s instead of 4 hours.

To compute a FFT you would do:

$\text{Np.fft.rfft}(\text{sample_array} - (y_n))$ -> returns the array C_k coefficients

Np.fft.irfft is the inverse Fourier transform. Takes C_k and gives samples y_n

Representing data using the fourier transform deletes data seeming unnecessary to human hearing.

Recording

Given $f(t)$, N , T

sample @ $[0, 1, \dots, (N-1)] \frac{T}{N}$

$[y_0, y_1, \dots, y_{N-1}]$

DFT

$[C_0, C_1, \dots, C_{N-1}]$

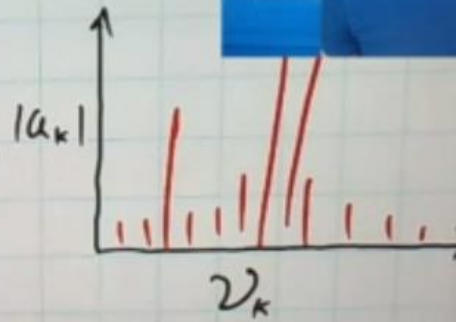
convert

$[|a_0|, |a_1|, \dots, |a_{N-1}|]$
 $[\varphi_0', \varphi_1', \dots, \varphi_{N-1}']$

plot

compute ω_k
for each k

$\Delta t \rightarrow \frac{T}{N}$



$\omega_k = 2\pi \frac{k}{N}$

