Michigan Data Science Team

Shazam Clone Week 2: Fourier Transform

Evan Teal, Dennis Farmer

Schedule for today

1: Digital Audio Recap

2: What is a Spectrogram?

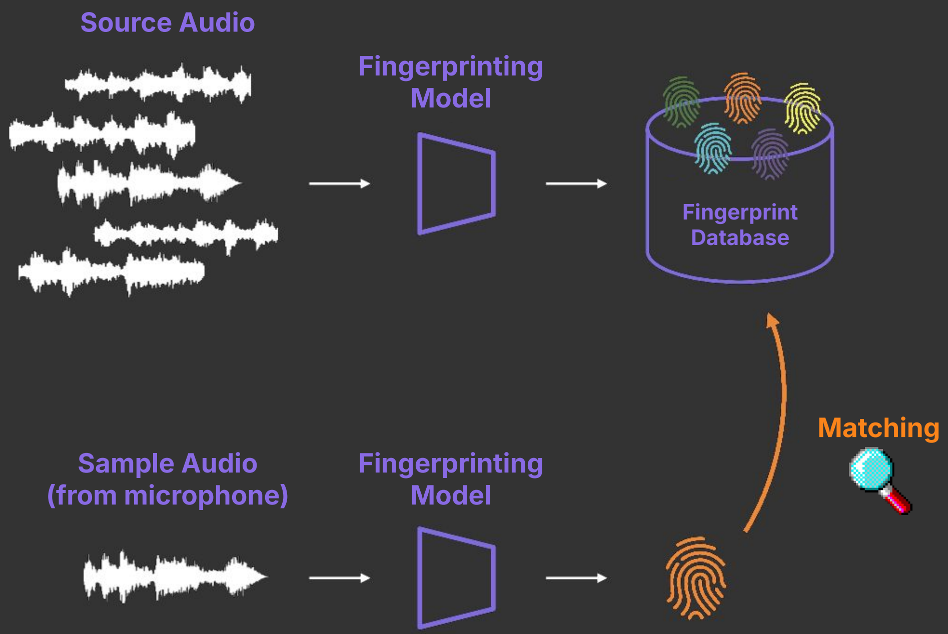
3: Fourier Transforms

4: Jupyter Notebook

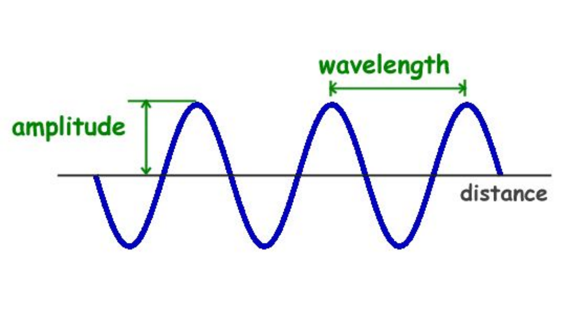
Rough Timeline

Date	Activity
~~Sept 21~~	~~Introduction + Digital Audio 🔊~~
Sept 28	Fourier Transforms, Spectrograms 🧮
Oct 5	Constellation Mapping 🔭
-	Fall Break 🍂
Oct 19	Audio Search, Expo Intro 🔍
Oct 26	Buffer Week, MySQL 💽
Nov 2	Flask endpoint, Expo 🌐
Nov 9	Putting it all together 🔧
Nov 16	Prepare for final presentations 🎉

Recap: Model Overview



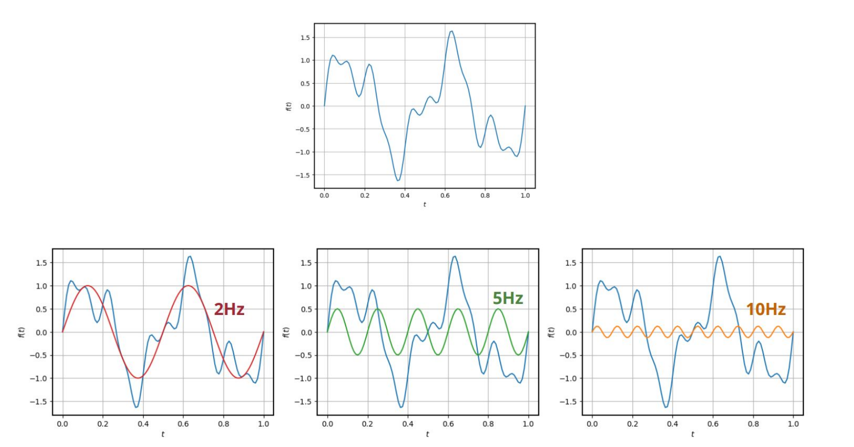
Recap: What is sound?



Sounds are composed of combinations of waves.

Sounds waves have 3 defining factors:

1. Frequency (Hz - cycles/sec)
2. Amplitude (dB - Loudness)
3. Phase (location)



Digital Audio: samples of amplitude at some sampling rate.

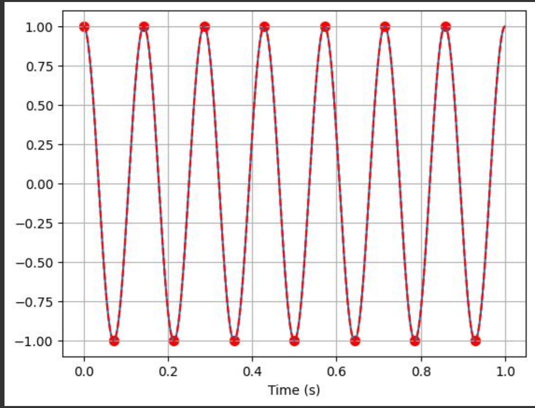
Recap: How many samples do we need?

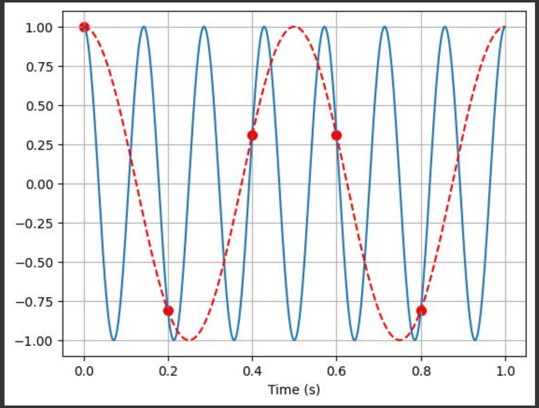
We can perfectly reconstruct a signal with highest contained frequency f_max, if our sampling rate is greater than 2*f_max.

<u>Intuition</u>: at least sample each peak and valley
(real world: sample a bit more)

When downsampling, first remove frequencies > sr/2 with “anti-aliasing” filter.

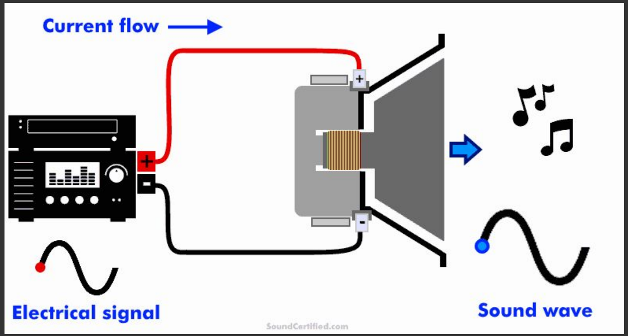
$$sr > 2f_{\max} \equiv f_{\max} < sr/2$$





Recap: Waveform Files (.mp3, .wav)

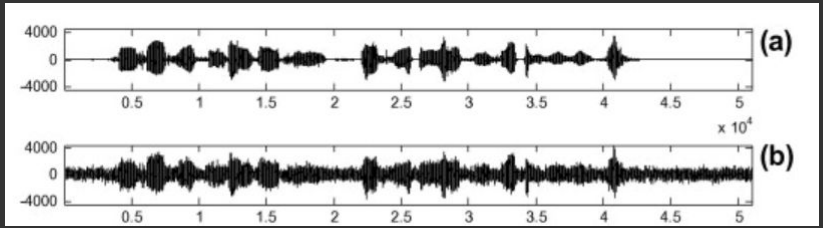
Pressure on Y-axis and time on X-axis



Designed for audio playback but not interpretation.

Why is this bad?

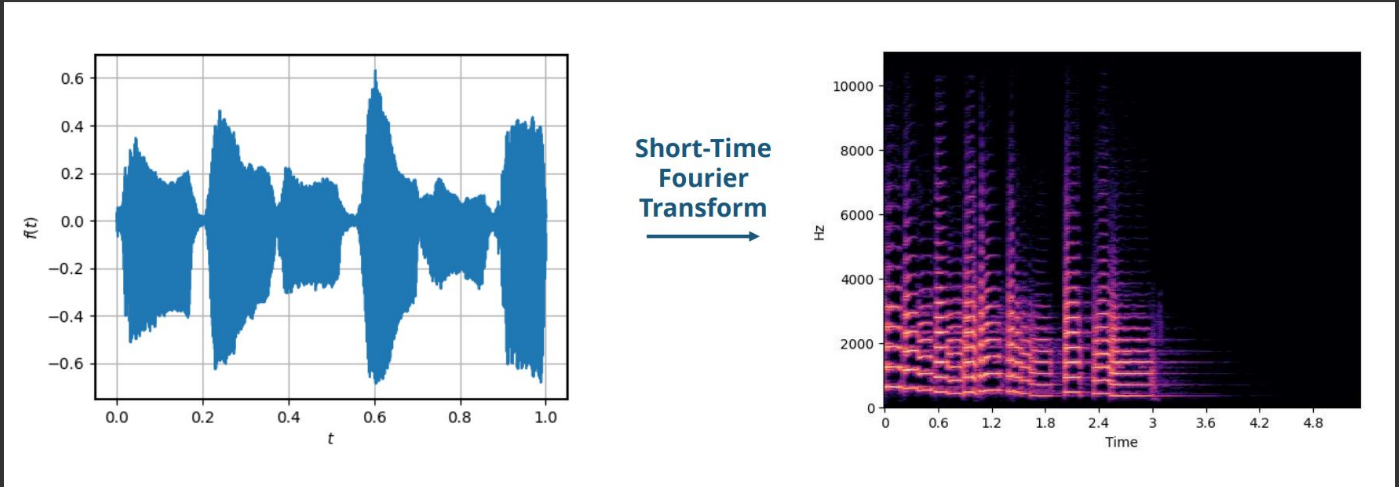
1. We cannot determine underlying frequencies
2. Background noise can't be isolated



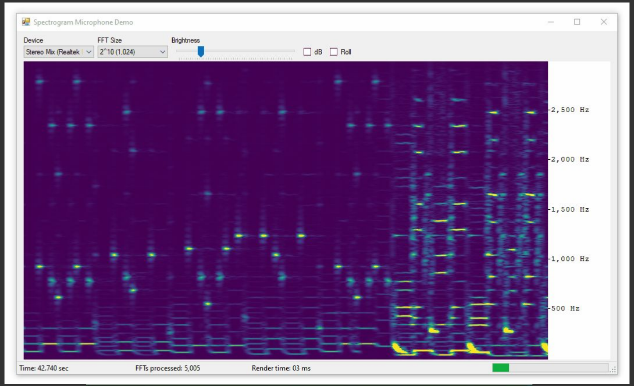
Recap: How do we fix this?

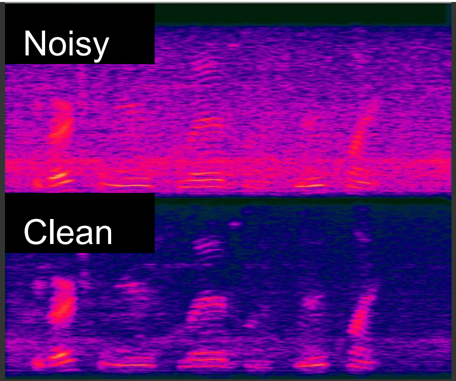
Waveform is not enough, we need frequencies:

Translate to a spectrogram!



Recap: Benefits of a spectrogram





We can see the distribution of frequencies while still maintaining a measure of amplitude and time.

Important frequencies stand out against background noise.

Audio fingerprinting is now possible!!!

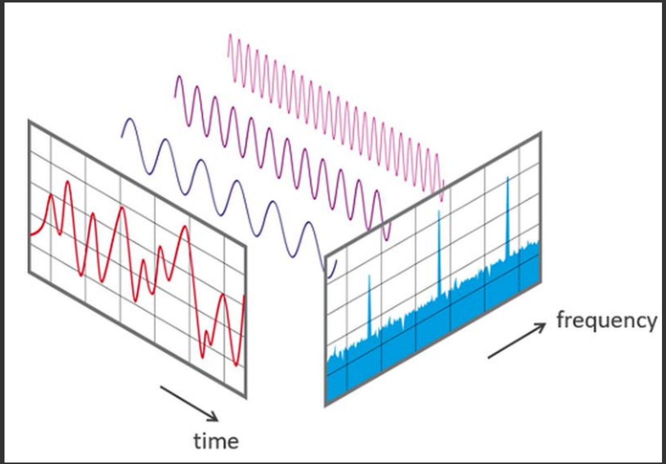
What is a Fourier Transform?

Formula to map functions in the time domain to the frequency domain.

Decipher which frequencies make up a signal via multiplying the original signal by many candidate frequencies.

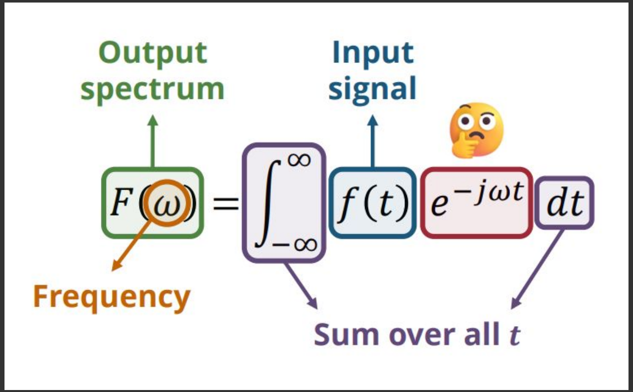
Uses:

Audio Recognition, Noise Cancelation, Speech Recognition, Spectroscopy



Fourier Transform

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t}\, dt$$



Why sine and cosine waves?

Why do we have a complex number?

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t}\, dt$$

Euler's formula

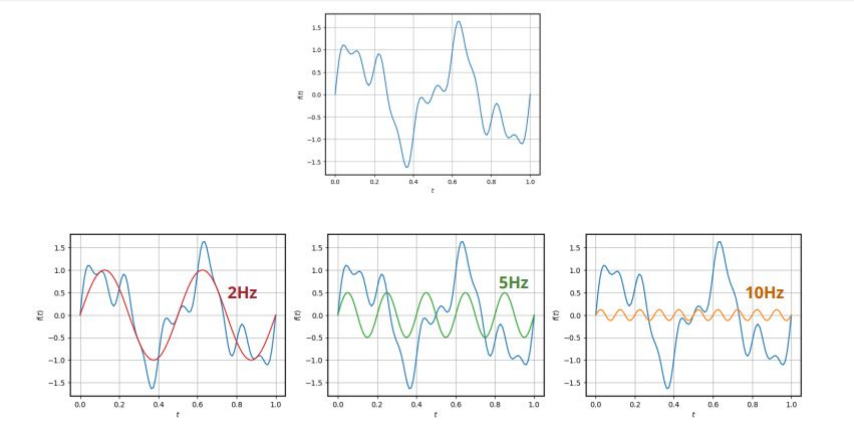
$$e^{j\theta} = \cos\theta + j\sin\theta$$

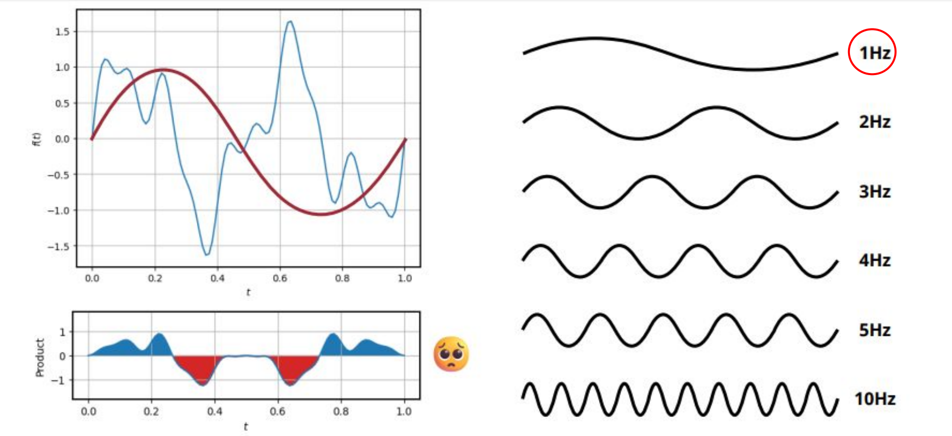
$$F(\omega) = \int_{-\infty}^{\infty} f(t)\cos(-\omega t) + j\, f(t)\sin(-\omega t)\, dt$$

Fourier Transform in Action

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\cos(-\omega t) + j\, f(t) \sin(-\omega t)\, dt$$

Sum over all t

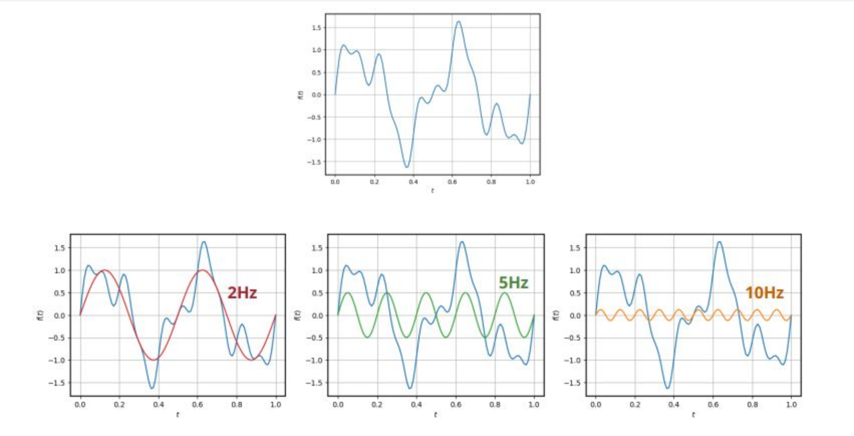


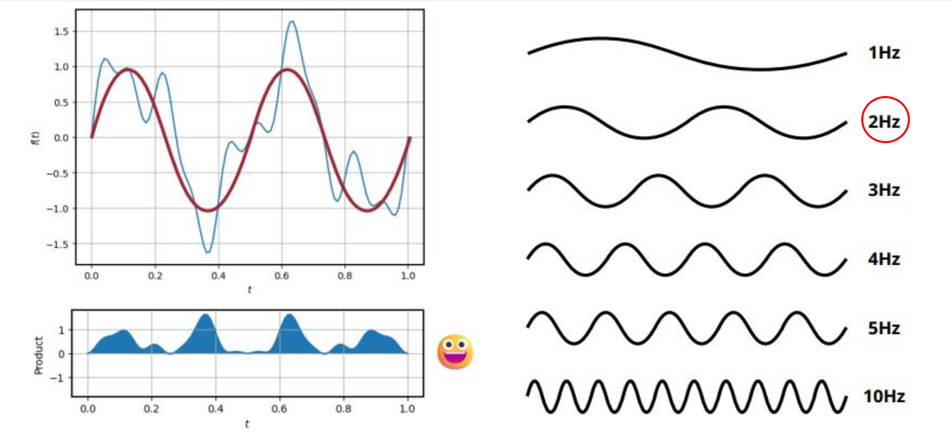


Fourier Transform in Action

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\cos(-\omega t) + j\, f(t)\sin(-\omega t)\, dt$$

Sum over all t

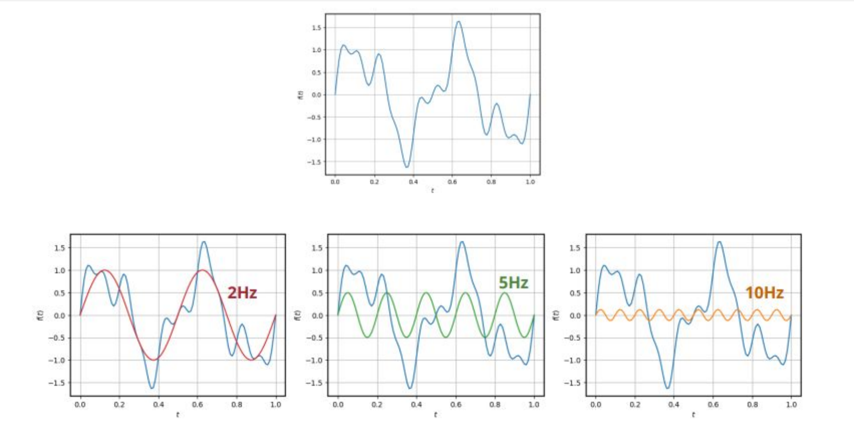


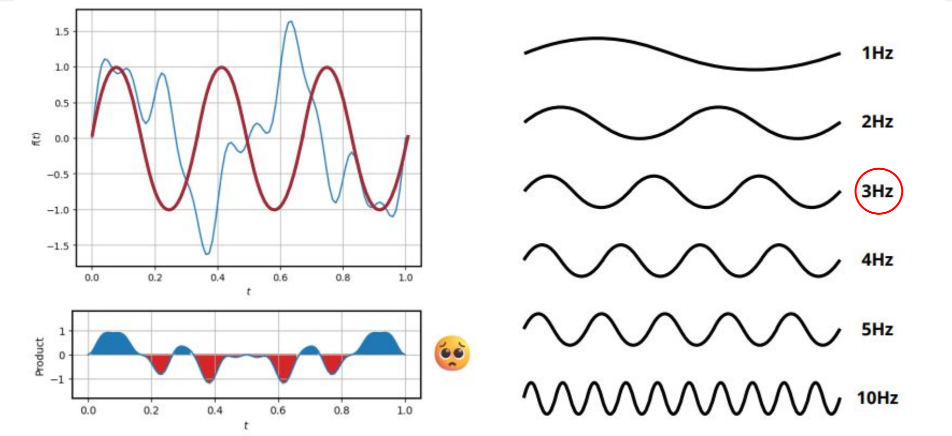


Fourier Transform in Action

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\cos(-\omega t) + j\, f(t)\sin(-\omega t)\, dt$$

Sum over all t

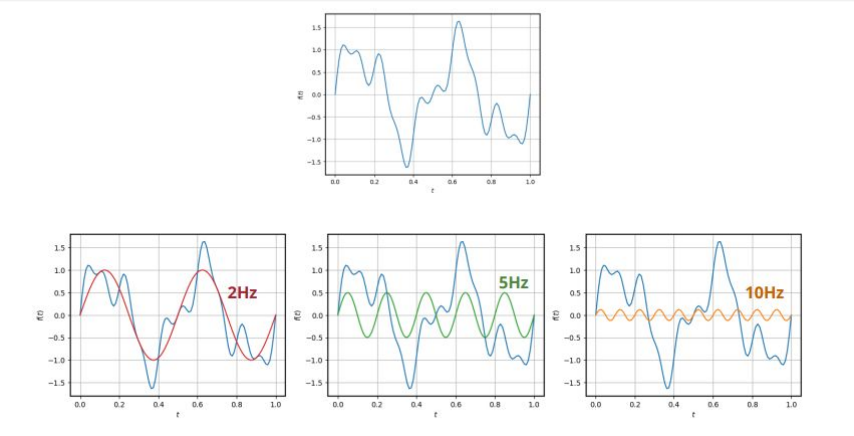


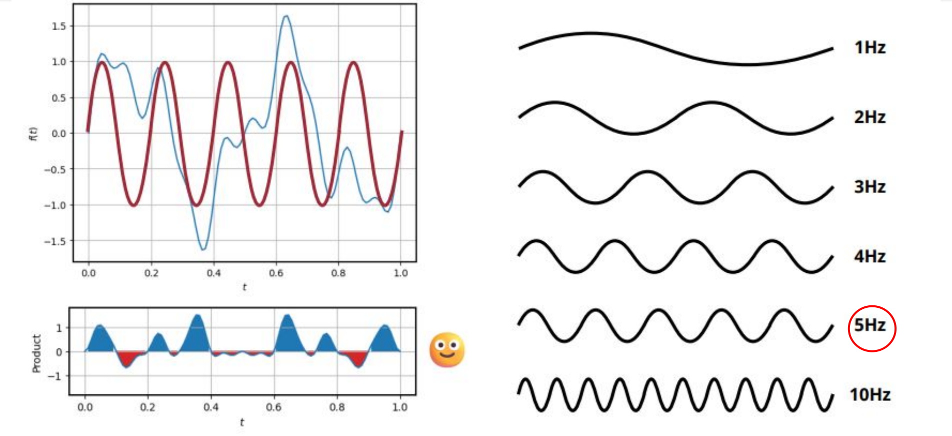


Fourier Transform in Action

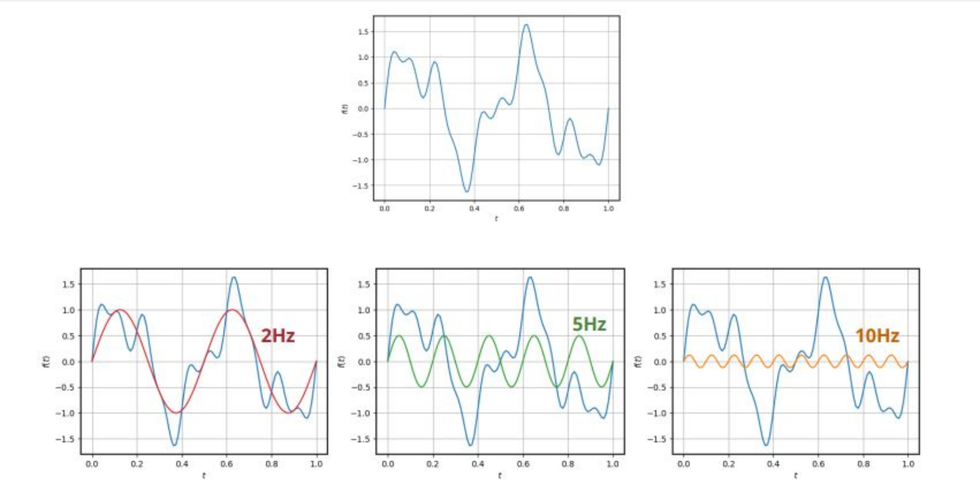
$$F(\omega) = \int_{-\infty}^{\infty} f(t)\cos(-\omega t) + j\, f(t) \sin(-\omega t)\, dt$$

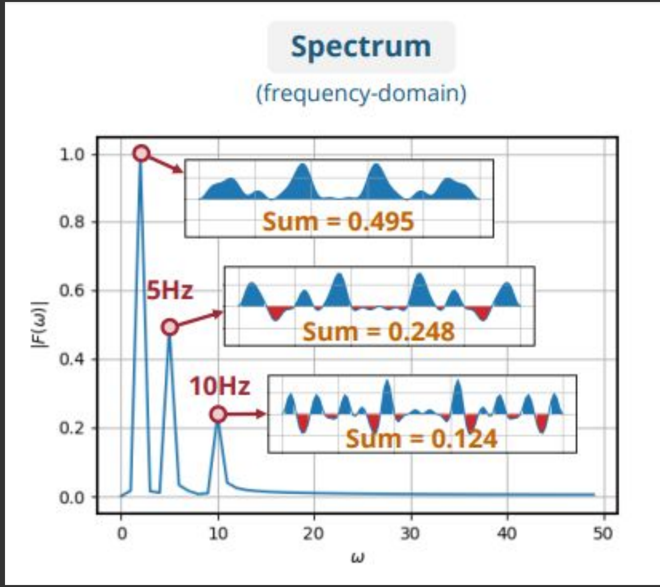
Sum over all t





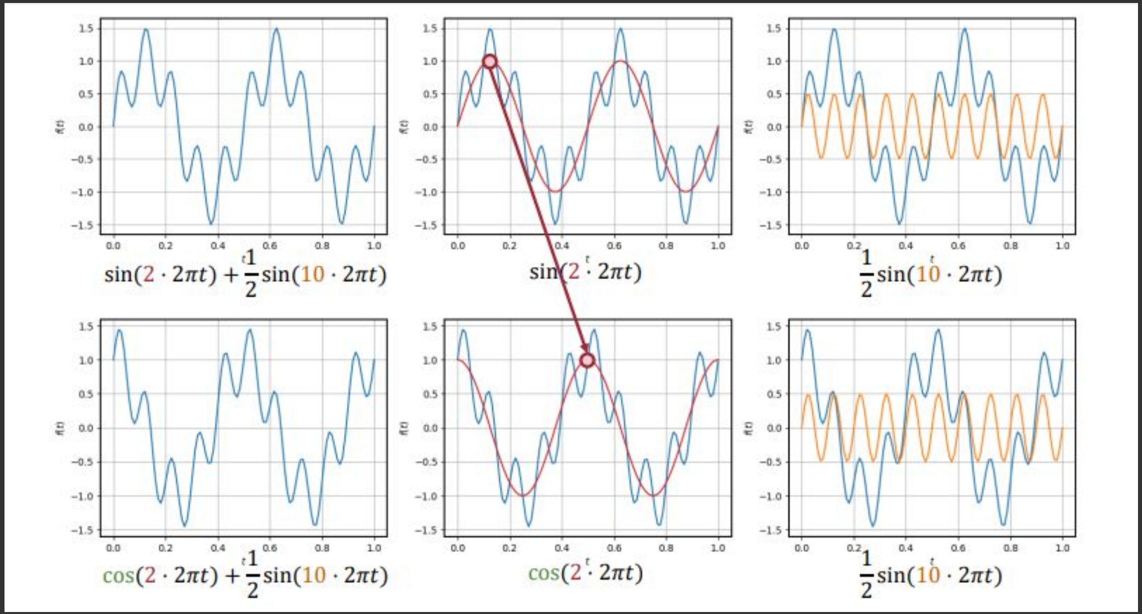
Fourier Transform: Final Product





Robustness Against Phase

What if our underlying frequency does not follow a sine wave (phase shift)?

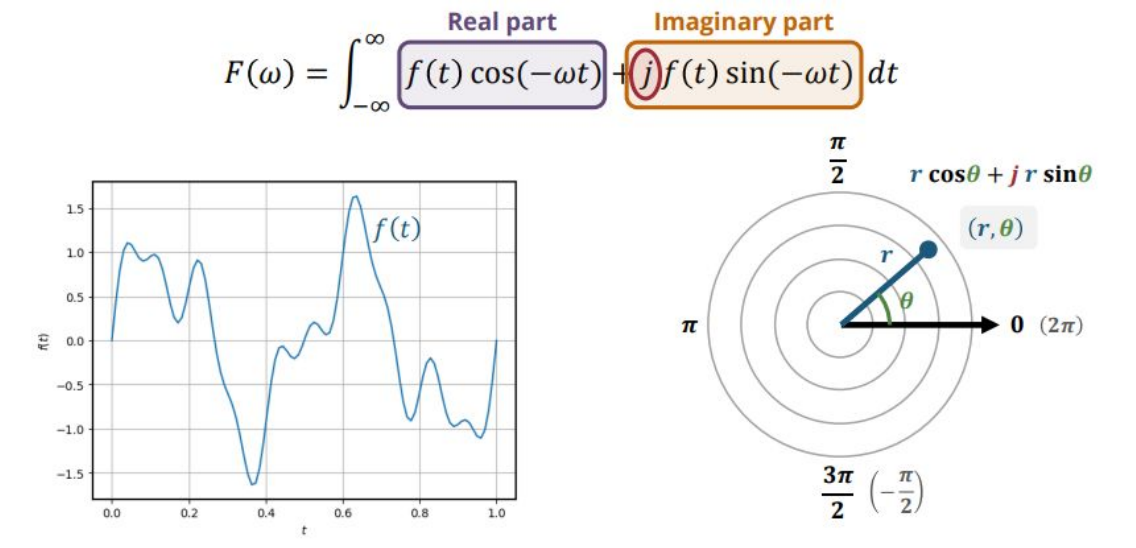


Robustness Against Phase

Radian to Cartesian:

$$x = r\cos(\theta) \quad \text{(real)}$$
$$y = r\sin(\theta) \quad \text{(imaginary)}$$

$$r: \quad \text{Magnitude}$$
$$\theta: \quad \text{Phase}$$



Allows us to explore waveforms at any phase offset.

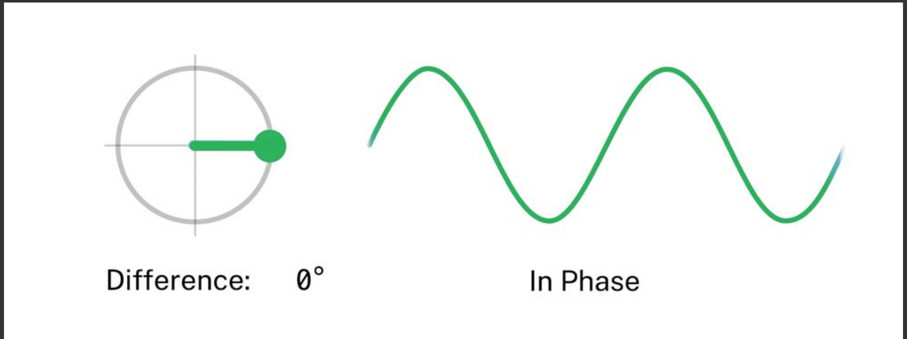
Robustness Against Phase

<u>Main Idea</u>:
independent of phase offset

Real(x): presence of cosine at freq
Im(x): presence of sine at freq

Sine and cosine contributions are treated equally when we calculate magnitude of the output:

$$|\,a + bj\,| = \sqrt{a^2 + b^2}$$



$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t}\, \mathrm{d}t$$
$$= \int_{-\infty}^{\infty} \underbrace{f(t)\cos(-\omega t)}_{\text{real}} + \underbrace{j\, f(t)\sin(-\omega t)}_{\text{imaginary}}\, \mathrm{d}t$$

Discrete Fourier Transform (DFT)

For digital audio:
replace integral with summation
over discrete samples

k: discrete frequency
n: discrete time index
N = number of samples

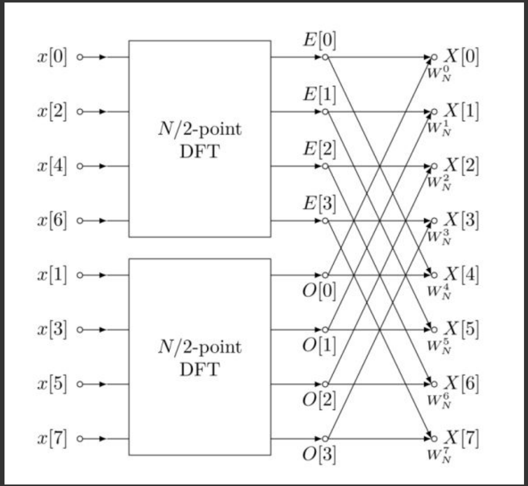
$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t}\, \mathrm{d}t$$
$$= \int_{-\infty}^{\infty} \underbrace{f(t)\cos(-\omega t)}_{\text{real}} + \underbrace{j\, f(t)\sin(-\omega t)}_{\text{imaginary}}\, \mathrm{d}t$$

$$X_k = \sum_{n=0}^{N-1} x_n\, e^{-j2\pi \frac{n}{N} k}$$
$$= \sum_{n=0}^{N-1} \underbrace{x_n \cos(-2\pi \frac{n}{N} k)}_{\text{real}} + \underbrace{j\, x_n \sin(-2\pi \frac{n}{N} k)}_{\text{imaginary}}$$

Fast Fourier Transform (FFT)

Efficient Implementation of DFT: O(n*log(n))

1. Base Case: DFT
2. Split into evens/odds
3. Recursive call on both sides.
4. Do a weighted recombination of even and odd segments



$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of even-indexed part of } x_n} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of odd-indexed part of } x_n} = E_k + e^{-\frac{2\pi i}{N}k} O_k \qquad \text{for } k = 0, \dots, \frac{N}{2}-1.$$

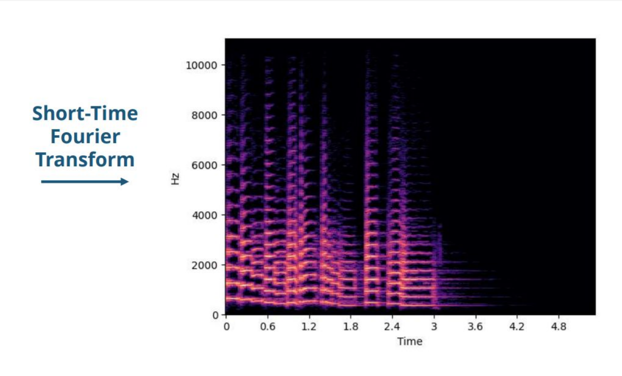
Cooley-Tukey Formula

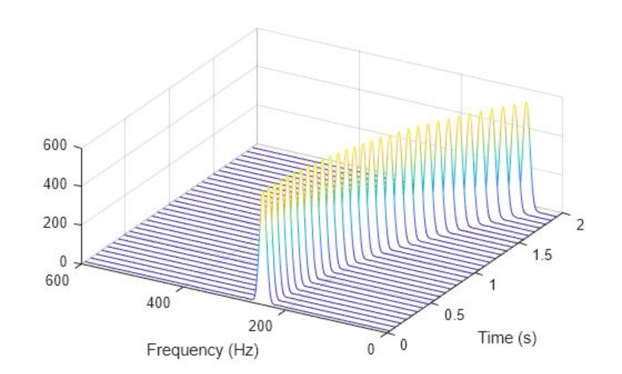
Short-Time Fourier Transform (STFT)

Now we know what frequencies make up our original sound!

How do we incorporate back time?

Take fourier transform at consecutive time "windows" to keep track of frequency distribution across time.

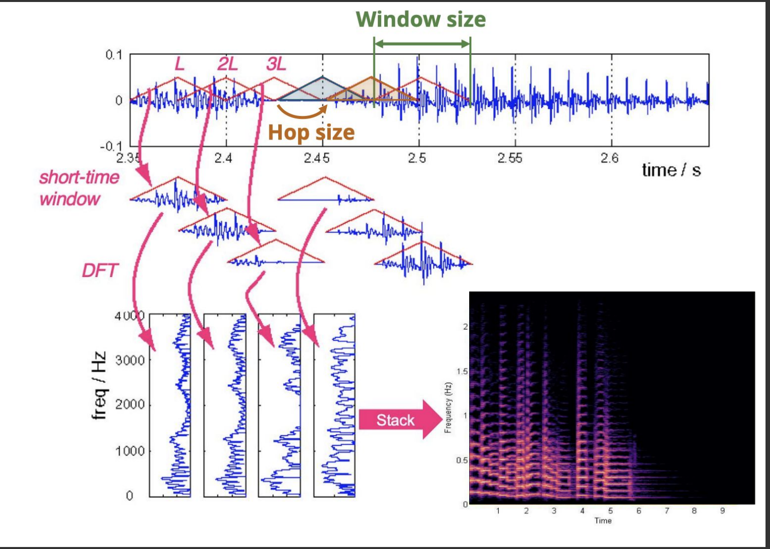


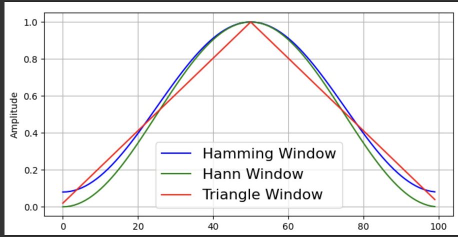


Short-Time Fourier Transform - Window Functions

Key Idea: Take the Fourier Transform of a segment (with some window size), then shift over (by some hop size) and repeat.

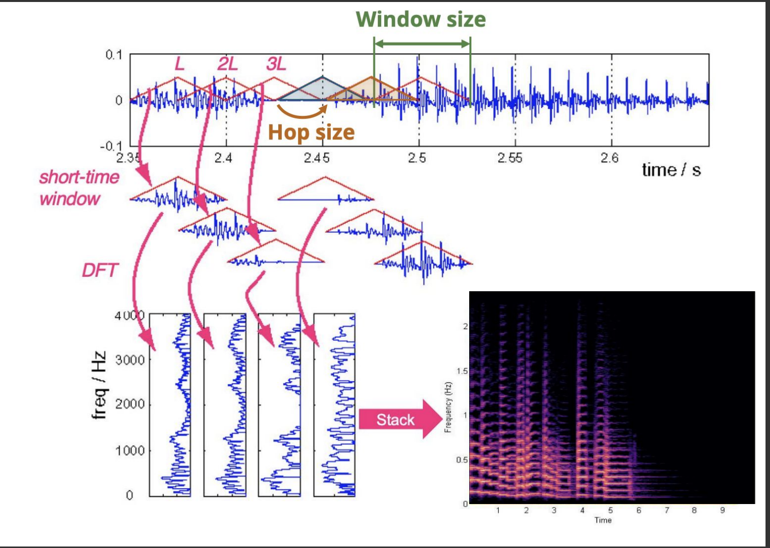
Each window is tapered at the edges by multiplying by a window function to reduce sharpness of transitions





Short Time Fourier Transform - Python

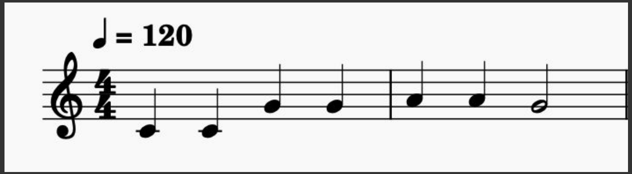
```python
audio_path = "./asset/log_scale_perception.wav"
audio, sr = librosa.load(audio_path, sr=None)

# number of consecutive samples that window is applied to
win_length = 2**11
hop_length = win_length // 4  # 75% overlap
window = scipy.signal.get_window("triang", Nx=win_length)

nperseg = win_length
# Number of points used in the FFT for each windowed segment
nfft = win_length  # common default
# if nfft > win_length,
# zero pad (segment*window) array to be length nfft
# before computing FFT
# effect: increase resolution, no new information

fs=sr
noverlap = nperseg - hop_length

freq_scipy, time_scipy, s_scipy = scipy.signal.stft(
    audio,
    fs=fs, window="hann", nfft=nfft,
    nperseg=nperseg, noverlap=noverlap
)
```



$$n\_rows = 1 + \frac{nfft}{2}$$

$$n\_cols = 1 + \left\lfloor \frac{N - window\_size}{hop\_length} \right\rfloor$$

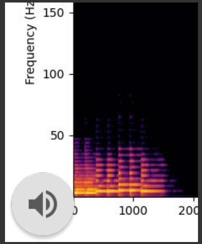
What information do we now have?



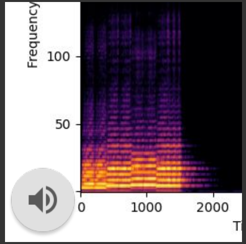
Timbre: tone color of a complex tone, determined by the different partials (pure tones/frequencies) composing the complex tone.

```python
stft_db = librosa.amplitude_to_db(np.abs(stft), ref=np.max)
im = plt.imshow(stft_db, cmap="inferno", aspect="auto", origin="lower")
plt.colorbar(im, format="%+2.0f dB")
plt.xlabel("Time (sec)")
plt.ylabel("Frequency (Hz)")
plt.show()
```

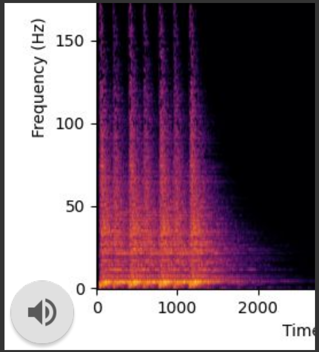
Piano



Flute



Cymbal



Trumpet