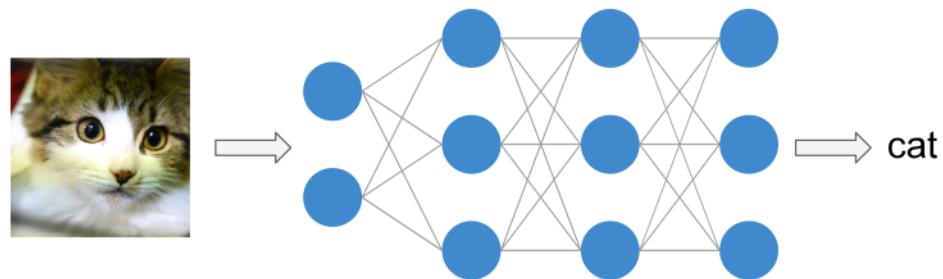


# Audio Signal Processing for Machine Learning

Valerio Velardo

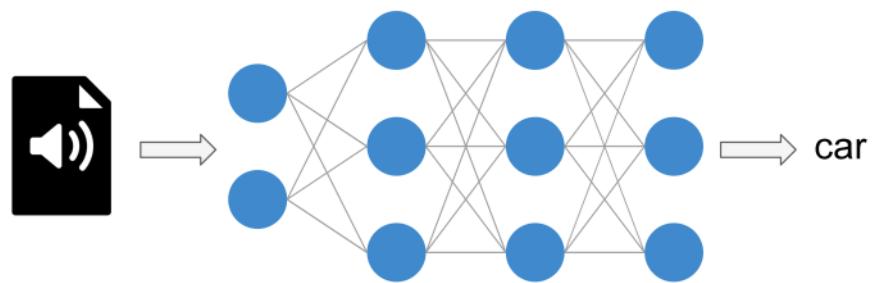
## Problem

---



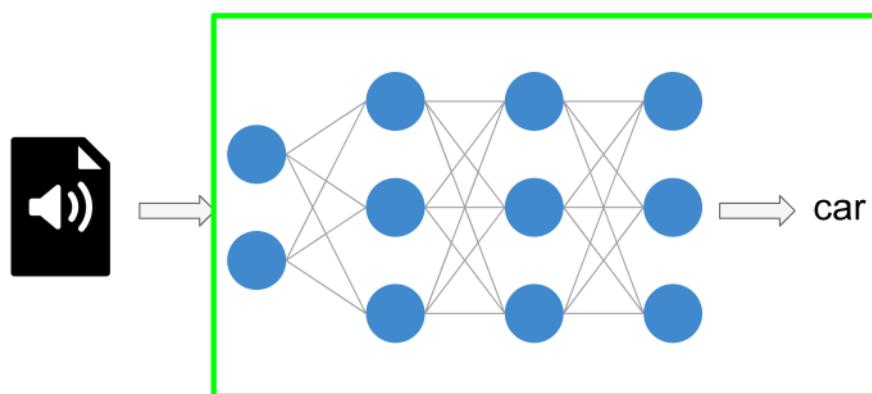
## Problem

---



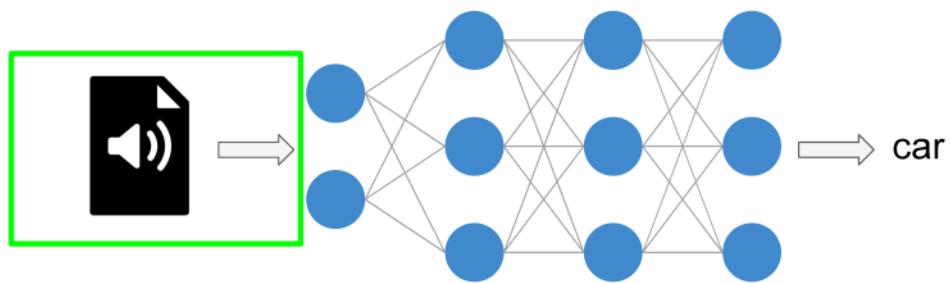
## Problem

---



## Problem

---



## **Applications**

---

- Audio classification
- Speech recognition / speaker verification
- Audio denoising / audio upsampling
- Music Information Retrieval
  - Music Instrument Classification
  - Mood Classification
  - ...
- ...

# Content

---

- Sound waves
- DAC / ADC
- Time- and frequency-domain audio features (e.g., rms, spectral centroid)
- Audio transformations
  - Fourier Transform / STFT
  - Constant-Q Transform
  - Mel Spectrograms
  - Chromograms
- ...

## **What should you expect?**

---

- Theory
- Coding tutorials

Where do you I get the code/slides?



## Technology stack

---



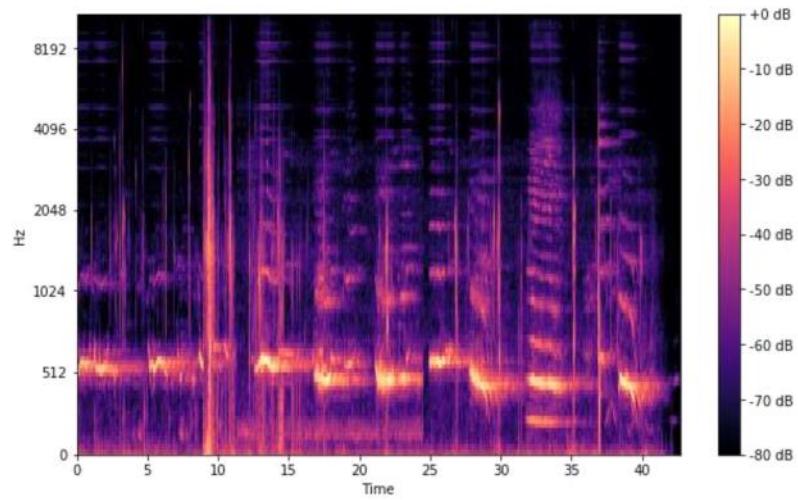
## What you'll learn

---

- Get a deep understanding of audio data
- Familiarise with frequency/time-domain audio features
- Extract features from raw audio
- Recognise what audio features to use for ML applications
- Preprocess audio data for ML
- Understand (some!) math behind audio transformations
- Use *librosa* for your audio projects

**Don't freak out!**

---



## Who's this series for?

---

- ML/DL engineers
- Computer science students
- Software engineers
- Music technologists
- Tech-oriented musicians

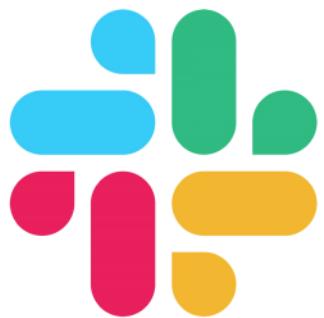
## Prerequisites

---

- Intermediate Python programming

**Join the community!**

---



**[thesoundofai.slack.com](https://thesoundofai.slack.com)**

# Sound and waveforms

Valerio Velardo

## Sound

---

- Produced by vibration of an object
- Vibrations cause air molecules to oscillate
- Change in air pressure creates a wave

## Mechanical wave

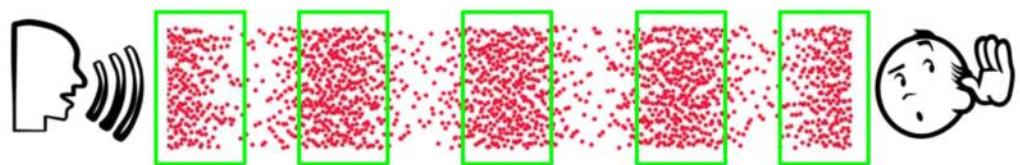
---

- Oscillation that travels through space
- Energy travels from one point to another
- The medium is deformed

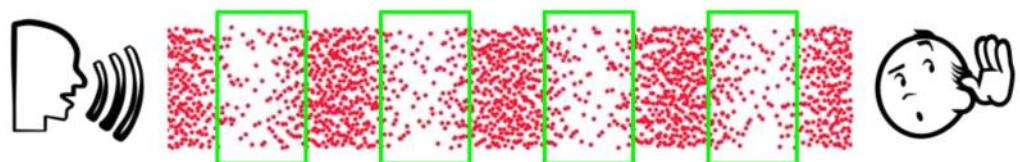
## Sound wave



## Sound wave



## Sound wave

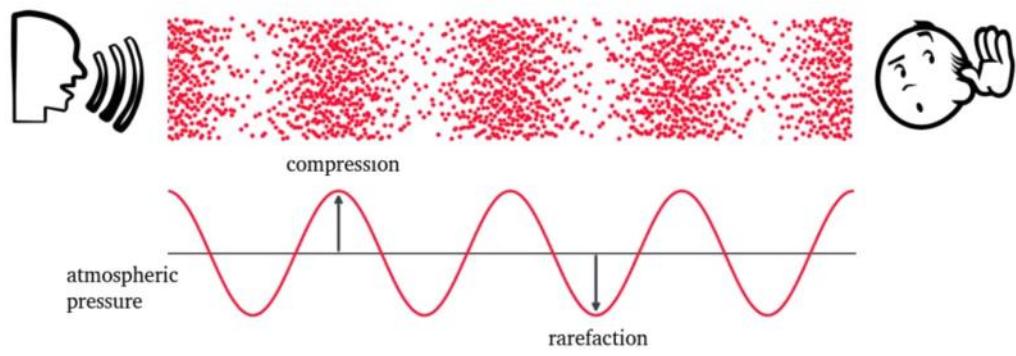


## Sound wave



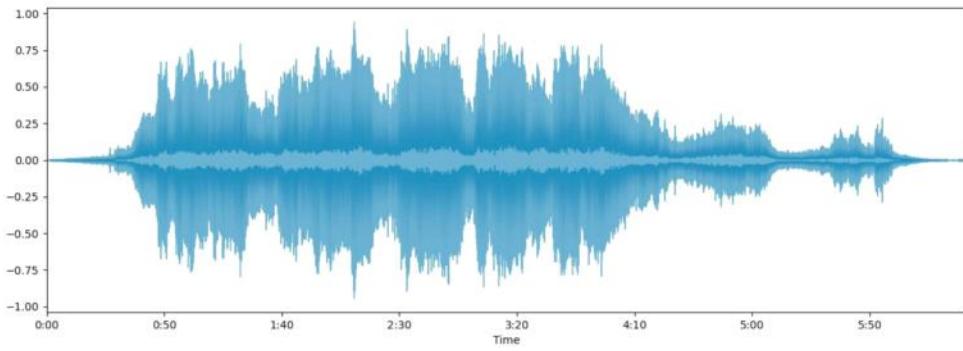
## Sound wave

---



## Waveform

---



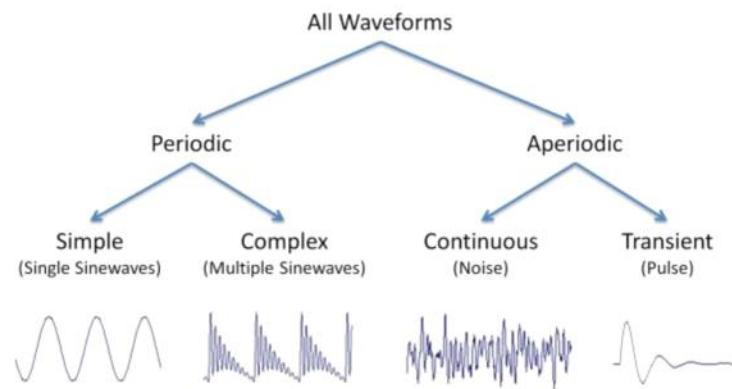
## Waveform

---

- Carries multifactorial information:
  - Frequency
  - Intensity
  - Timbre

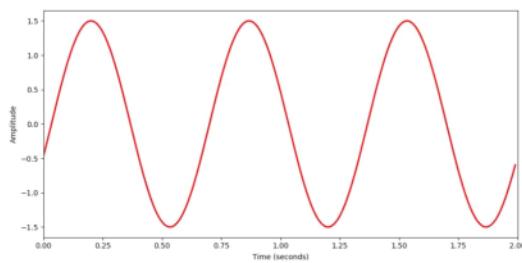
## Periodic and aperiodic sound

---



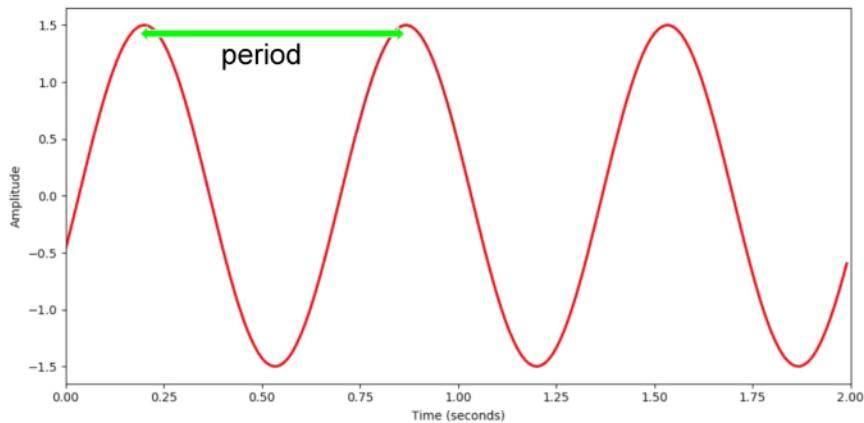
## Waveform

---

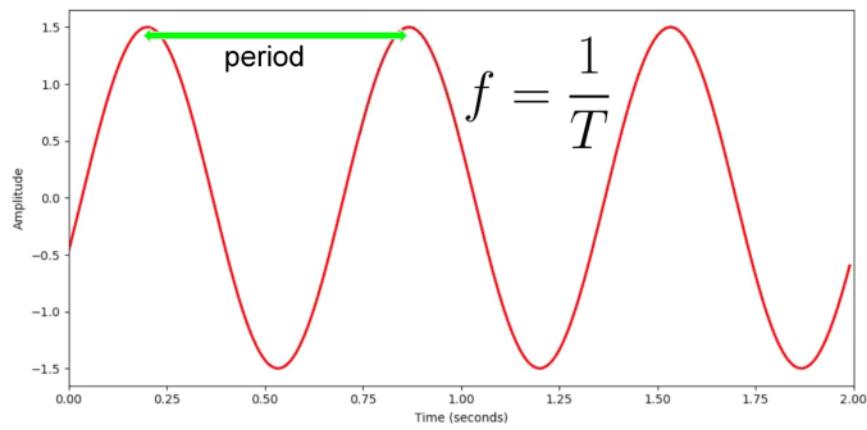


$$y(t) = A \sin(2\pi ft + \varphi)$$

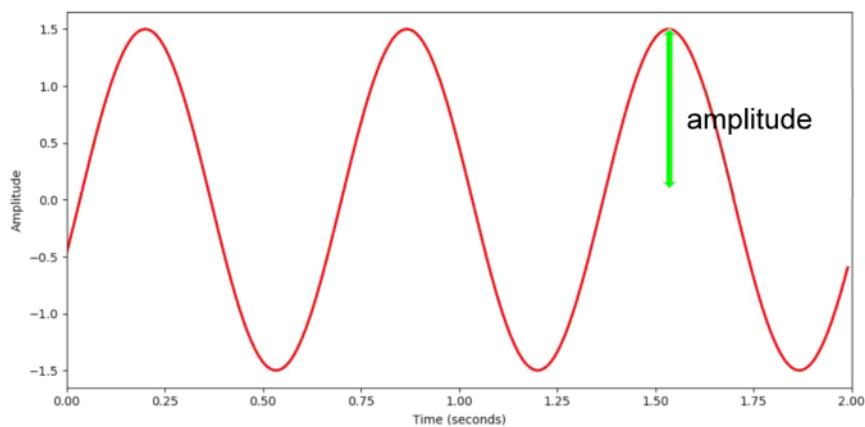
## Frequency



## Frequency

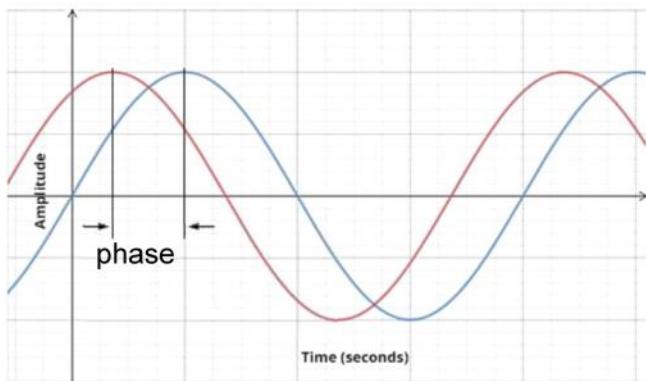


## Amplitude



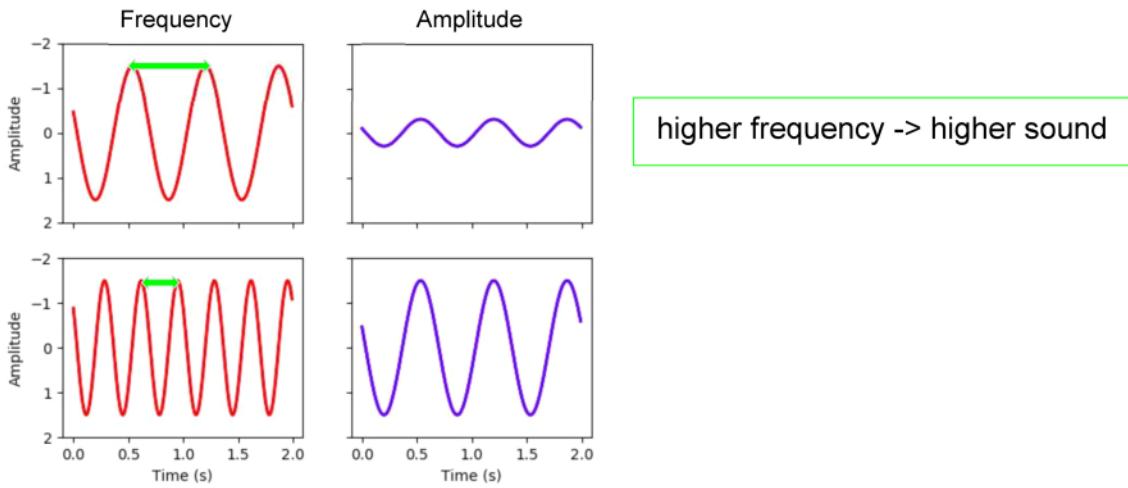
## Phase

---

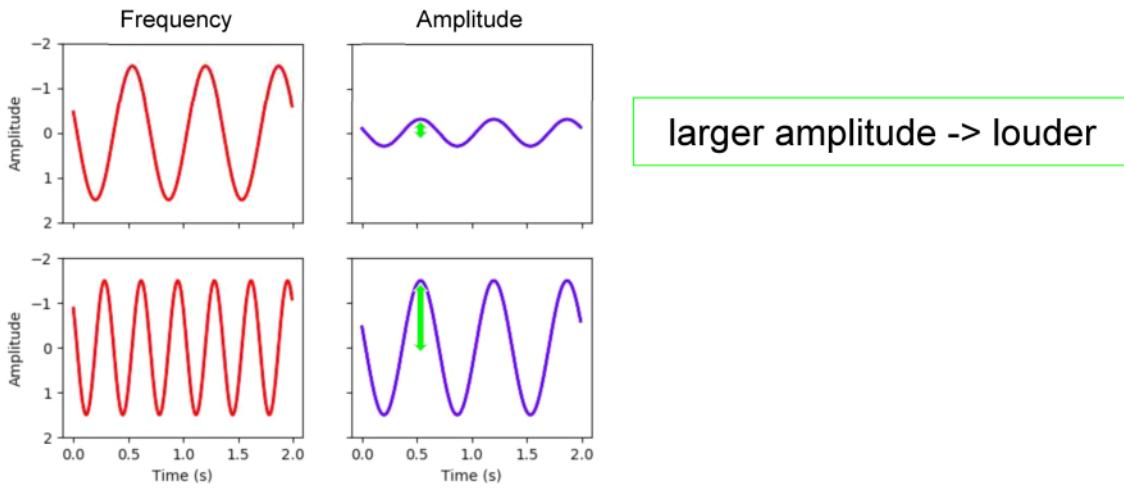


## Frequency and amplitude

---

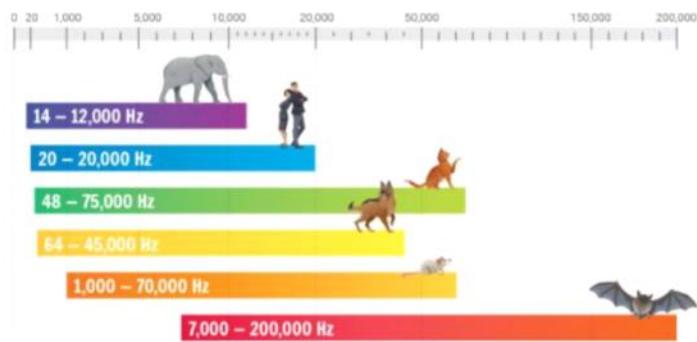


## Frequency and amplitude



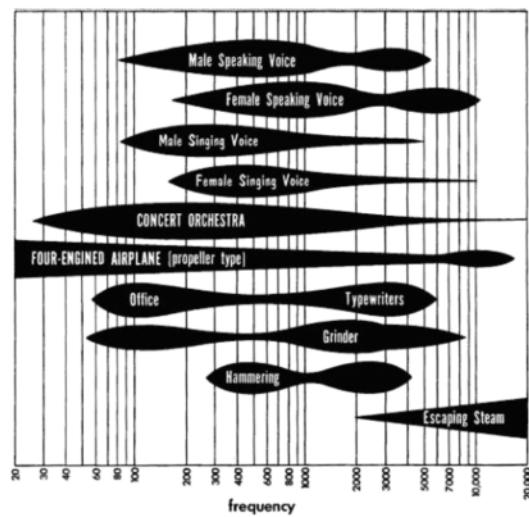
## Hearing range

---



## Hearing range

---



## Pitch

---

- Logarithmic perception 
- 2 frequencies are perceived similarly if they differ by a power of 2

## Midi notes

Note name	A7#	108	C8
Midi number	G7#	106	B8
	F7#	104	A7
	D7#	102	103
	99	101	G7
	98	E7	
	97	D7	
	96	C7	
	94	B6	
	93	A6	
	92	G6	
	91	F6	
	90	E6	
	89	D6	
	88	C6	
	87	B6	
	86	A5#	
	85	G5#	
	84	F5#	
	83	E5	
	82	D5	
	81	C5#	
	80	B5	
	79	A5	
	78	G5	
	77	F5	
	76	E5	
	75	D5	
	74	C5#	
	73	B5	
	72	A4#	
	71	G4#	
	70	F4#	
	69	E4	
	68	D4#	
	67	C4#	
	66	B4	
	65	A3#	
	64	G3#	
	63	F3#	
	62	E3	
	61	D3#	
	60	C3#	
	59	B3	
	58	A3#	
	57	G3#	
	56	F3#	
	55	E3	
	54	D3#	
	53	C3#	
	52	B2#	
	51	A2#	
	50	G2#	
	49	F2#	
	48	E2#	
	47	D2#	
	46	C2#	
	45	B1	
	44	A1#	
	43	G1	
	42	F1#	
	41	E1	
	40	D1#	
	39	C1#	
	38	B0	
	37	A0#	
	36		
	35		
	34		
	33		
	32		
	31		
	30		
	29		
	28		
	27		
	26		
	25		
	24		
	23		
	22		
	21		

## Midi notes

## Midi notes

Note name	A7#	106	107	B7
	G7#	-104		A7
	F7#	102	103	G7
	D7#	99	100	E7
Midi number	C7#	97	98	D7
	A6#	94	95	B6
	G6#	92	93	A6
	F6#	90	91	G6
	E6	89	89	F6
	D6#	87	88	E6
	C6#	85	86	D6
	A5#	82	83	C6
	G5#	80	81	B5
	F5#	78	79	A5
	D5#	75	76	G5
	C5#	73	74	F5#
	A4#	70	71	E5
	G4#	68	69	D5#
	F4#	66	67	C5#
	D4#	63	64	B4
	C4#	61	62	A4#
	B3	59	59	G4#
	A3#	58	59	F4#
	G3#	56	55	E4#
	F3#	54	53	D4#
	D3#	51	52	C4#
	C3#	49	50	B3
	A2#	46	47	A3#
	G2#	44	45	G3#
	F2#	42	43	F3#
	D2#	39	40	D3#
	C2#	37	36	C3#
Note name	A1#	34	35	A2#
	G1#	32	33	F1#
	F1#	30	31	E1
	D1#	27	28	C1#
	C1#	25	26	B0
	A0#	22	23	A0

## Midi notes

Note name	A7#	106	107	C8
	G7#	-104		B7
	F7#	102	103	G7
	D7#	99	100	E7
Midi number	C7#	97	98	D7
	A6#	94	95	B6
	G6#	92	93	A6
	F6#	90	91	G6
	E6#	89	90	F6
	D6#	87	88	E6
	C6#	85	86	D6
	A5#	82	83	C5
	G5#	80	81	B5
	F5#	78	79	G5
	D5#	75	76	F5
	C5#	73	74	E5
	A4#	70	71	D5
	G4#	68	69	C4
	F4#	66	67	B4
	D4#	63	64	E4
	C4#	61	60	D4
	A3#	58	59	C4
	G3#	56	57	B3
	F3#	54	53	A3
	C3#	49	48	F3
	D3#	51	52	E3
	G2#	46	47	D3
	F2#	44	45	C3
	E2#	42	43	B2
	D2#	39	41	F2
	C2#	37	38	D2
	A1#	34	35	C2
	G1#	32	33	B1
	F1#	30	29	A1
	E1#	27	28	G1
	D1#	25	26	F1
	C1#	24	21	E1
	B0	23	21	D1
Note name	A0#	22	23	C1
	A0	21		B0

## Midi notes

Note name	A7#	106	107	C8
	G7#	-104		B7
	F7#	102	103	G7
	D7#	99	100	E7
Midi number	C7#	97	98	D7
	A6#	94	95	B6
	G6#	92	93	A6
	F6#	90	91	G6
	E6#	89	89	F6
	D6#	87	88	D6
	C6#	85	84	C6
	A5#	82	83	B5
	G5#	80	81	A5
	F5#	78	79	G5
	D5#	75	76	E5
	C5#	73	74	D5
		72	72	
Note name	A4#	70	71	B4
	G4#	68	69	A4
	F4#	66	67	G4
	D4#	63	64	E4
	C4#	61	62	D4
		60	60	C4
Note name	A3#	58	59	B3
	G3#	56	57	A3
	F3#	54	55	G3
	D3#	51	52	E3
	C3#	49	48	D3
	A2#	46	47	B2
	G2#	44	45	A2
	F2#	42	43	G2
	D2#	39	41	F2
	C2#	37	38	D2
	A1#	34	35	C2
	G1#	32	33	B1
	F1#	30	31	A1
	D1#	27	28	G1
	C1#	25	26	F1
	A0#	22	23	E1
Midi number		21		B0
				A0

## Midi notes

Note name	A7#	<106	107	C8
	G7#	-104	105	A7
	F7#	-102	103	G7
	D7#	99	100	E7
	C7#	97	98	D7
	A6#	94	95	B6
	G6#	92	93	A6
	F6#	90	91	G6
	E6	89	90	F6
	D6#	87	88	E6
	C6#	85	86	D6
	A5#	82	83	B5
	G5#	80	81	A5
	F5#	78	79	G5
	D5#	75	76	E5
	C5#	73	74	D5
	A4#	70	71	B4
	G4#	68	69	A4
	F4#	66	67	G4
	D4#	63	64	E4
	C4#	61	62	D4
	B3	59	60	C4#
	A3#	58	59	B3
	G3#	56	57	A3#
	F3#	54	55	G3
	D3#	51	52	F3#
	C3#	49	50	E3
	B2	48	49	D3#
	A2#	46	47	C3#
	G2#	44	45	B2
	F2#	42	43	A2#
	E2	41	42	G2#
	D2#	39	40	F2#
	C2#	37	38	E2
	B1	35	36	D2#
	A1#	34	35	C2#
	G1#	32	33	B1
	F1#	30	31	A1#
	E1	29	30	G1#
	D1#	27	28	F1#
	C1#	25	26	E1
	B0	23	24	D1#
	A0	21	22	C1#

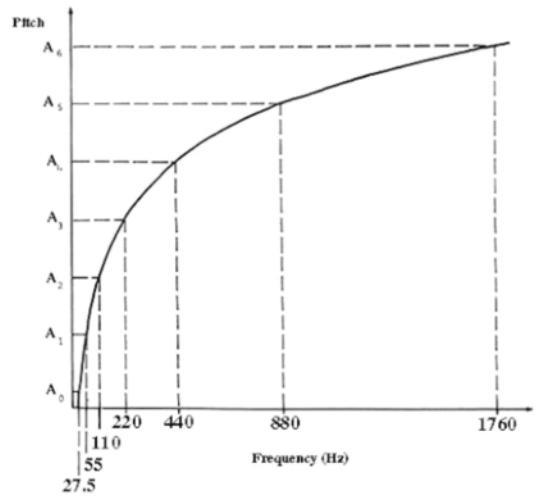
440 Hz

## Midi notes

440 Hz 880 Hz

## Pitch-frequency chart

---



## Mapping pitch to frequency

$$F(p) = 2^{\frac{p-69}{12}} \cdot 440$$

## Mapping pitch to frequency

$$F(60) = 2^{\frac{60-69}{12}} \cdot 440 = 261.6 \text{ Hz}$$

## Mapping pitch to frequency

$$F(p+1)/F(p) = 2^{1/12} = 1.059$$

$\downarrow$   
 $C4 \times 1.059 = D4$  (Freq) ✓  
also in L1C

Op-amp  $\rightarrow 1059 \text{ Hz}$

## Cents

---

- Octave divided in 1200 cents
- 100 cents in a semitone
- Noticeable pitch difference: 10-25 cents

## What's up next?

- Intensity, power, loudness
- Timbre

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

# Intensity, loudness, and timbre

Valerio Velardo

## The power of sound!

---



## Sound power

---

- Rate at which energy is transferred
- Energy per unit of time emitted by a sound source in all directions
- Measured in watt (W)

$$\frac{E}{t}$$

## Sound intensity

---

- Sound power per unit area
- Measured in W/m<sup>2</sup>

$$\frac{P}{A} = \frac{E}{t A}$$

$$A = m^2$$



~ 1 Watt



~ |Watt

1 Watt



= 100 W

## Threshold of hearing

---

- Human can perceive sounds with very small intensities

## Threshold of hearing

---

- Human can perceive sounds with very small intensities

$$TOH = 10^{-12} W/m^2$$

## Threshold of pain

---

$$TOP = 10 \cdot W/m^2$$

## Intensity level

---

- Logarithmic scale
- Measured in decibels (dB)
- Ratio between two intensity values
- Use an intensity of reference (TOH)

Intensity level

$$dB(I) = 10 \cdot \log_{10}\left(\frac{I}{I_{TOH}}\right)$$



Intensity level

$$dB(I_{TOH}) = 10 \cdot \log_{10}\left(\frac{I_{TOH}}{I_{TOH}}\right) = 0$$

## Intensity level

---

$$dB(I_{TOH}) = 10 \cdot \log_{10}\left(\frac{I_{TOH}}{I_{TOH}}\right) = 0$$

**log(1) = 0**

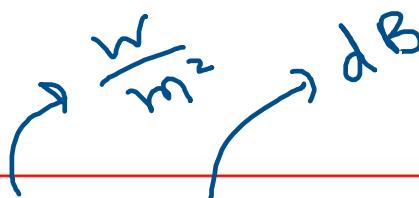
## Intensity level

---

- Every ~3 dBs, intensity doubles

## Intensity level

---



Source	Intensity	Intensity level	$\times$ TOH
Threshold of hearing (TOH)	$10^{-12}$	0 dB	1
Whisper	$10^{-10}$	20 dB	$10^2$
Pianissimo	$10^{-8}$	40 dB	$10^4$
Normal conversation	$10^{-6}$	60 dB	$10^6$
Fortissimo	$10^{-2}$	100 dB	$10^{10}$
Threshold of pain	10	130 dB	$10^{13}$
Jet take-off	$10^2$	140 dB	$10^{14}$
Instant perforation of eardrum	$10^4$	160 dB	$10^{16}$

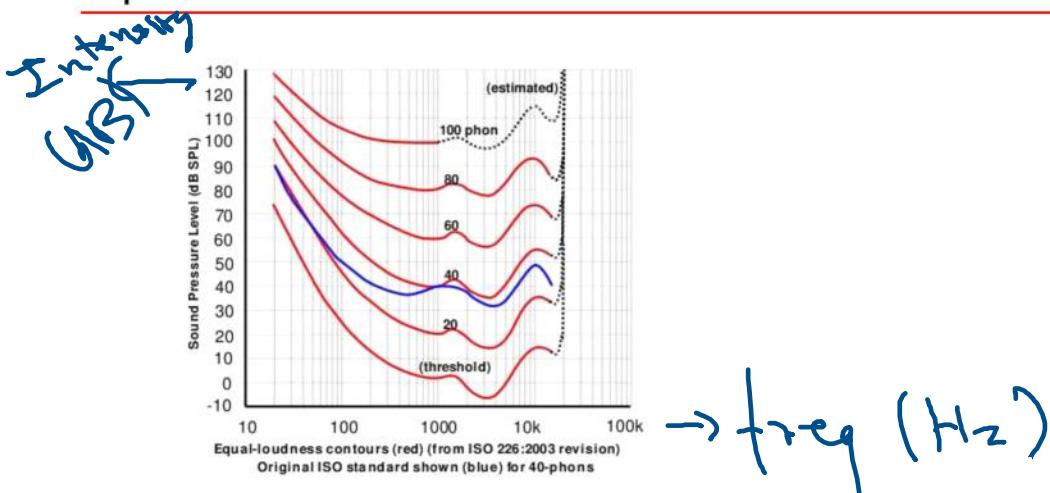
Table 1.1 from [Müller, FMP, Springer 2015]

## Loudness

- Subjective perception of sound intensity
- Depends on duration / frequency of a sound
- Depends on age
- Measured in phons

→ short duration → less loud  
→ more loud  
long duration

## Equal loudness contours



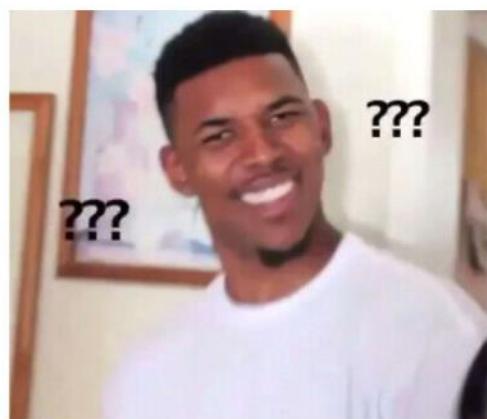
## Timbre

---

No real conclusion def<sup>r</sup> of timbre

## Timbre

---



## Timbre

---

- Colour of sound

## Timbre

---

- Colour of sound
- Diff between two sounds with same intensity, frequency, duration

Violin vs piano

## Timbre

---

- Colour of sound
- Diff between two sounds with same intensity, frequency, duration
- Described with words like: bright, dark, dull, harsh, warm

## What are the features of timbre?

---

- Timbre is multidimensional

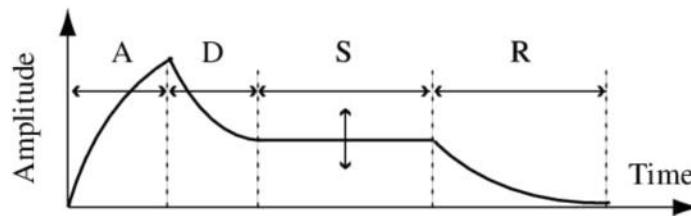
## What are the features of timbre?

---

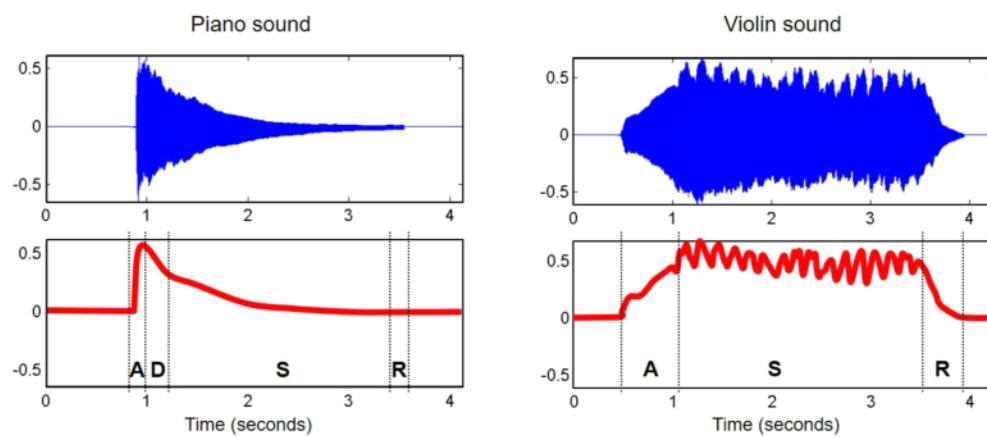
- Timbre is multidimensional
- Sound envelope
- Harmonic content
- Amplitude / frequency modulation

## Sound envelope

- Attack-Decay-Sustain-Release Model



Sound envelope → ADSR → variation of amplitude



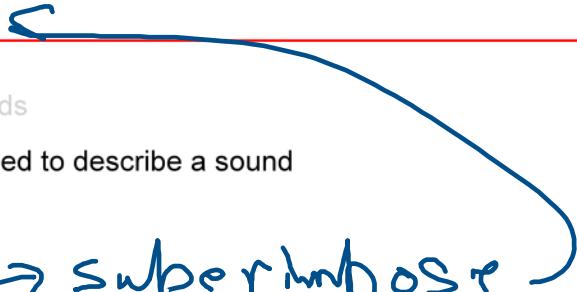
(Muller FMP)

## Complex sound

---

- Superposition of sinusoids

## Complex sound



- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound

partials  $\rightarrow$  Superimpose

$$\text{eg } \sin(2\pi \times 440 \times t) + \sin(2\pi \times 880 \times t) + \dots + \sin(2\pi \times 777 \times t)$$

Harmonic  $\rightarrow$  somehow determining  
Timbre

## Complex sound

- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound
- The lowest partial is called *fundamental frequency*

## Complex sound

---

- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound
- The lowest partial is called *fundamental frequency*
- A harmonic partial is a frequency that's a multiple of the fundamental frequency

## Complex sound

---

- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound
- The lowest partial is called *fundamental frequency*
- A harmonic partial is a frequency that's a multiple of the fundamental frequency

$$f_1 = 440$$

## Complex sound

---

- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound
- The lowest partial is called *fundamental frequency*
- A harmonic partial is a frequency that's a multiple of the fundamental frequency

$$f_1 = 440, f_2 = 2 \cdot 440 = 880$$

## Complex sound

---

- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound
- The lowest partial is called *fundamental frequency*
- A harmonic partial is a frequency that's a multiple of the fundamental frequency

$$f_1 = 440, f_2 = 2 \cdot 440 = 880, f_3 = 3 \cdot 440 = 1320, \dots$$

## Complex sound

---

- Superposition of sinusoids
- A *partial* is a sinusoid used to describe a sound
- The lowest partial is called *fundamental frequency*
- A harmonic partial is a frequency that's a multiple of the fundamental frequency
- **Inharmonicity** indicates a deviation from a harmonic partial

## Harmonic vs inharmonic instruments

---



↓  
harmonics



↑  
per cymbals  
↓  
less harmonics

## Harmonic content

---



distribution of energy among diff  
harmonics

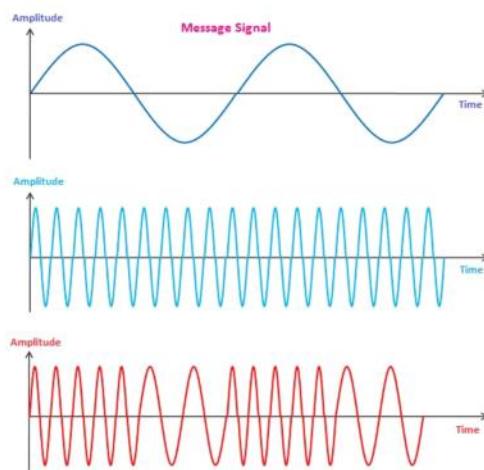
## Frequency modulation

---

- AKA vibrato
- Periodic variation in frequency → can be achieved by FM
- In music, used for expressive purposes

## Frequency modulation

---



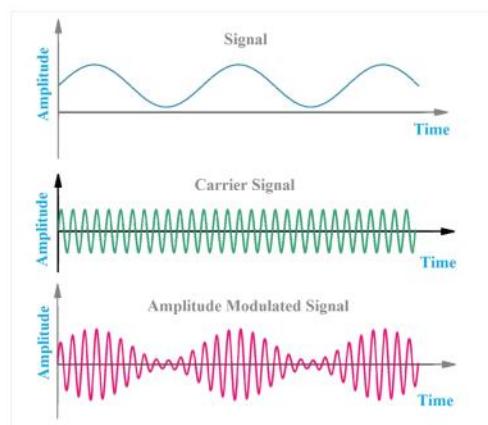
## Amplitude modulation

---

- AKA tremolo
- Periodic variation in amplitude
- In music, used for expressive purposes

## Amplitude modulation

---



## Timbre recap

---

- Multifactorial sound dimension
- Amplitude envelope
- Distribution of energy across partials
- Signal modulation (frequency/amplitude)

## Sound recap

---

- Sound is a wave
- Frequency, intensity, timbre
- Pitch, loudness, timbre → subjective

## What's up next?

---

- Introducing audio signal
- Audio to Digital Conversion (ADC)
- Digital to Audio Conversion (DAC)

Join the community!

---



[thesoundofai.slack.com](https://thesoundofai.slack.com)



intensity\_and\_timbre.ipynb

```
In [1]: pip install matplotlib librosa Python
Requirement already satisfied: matplotlib in c:\users\hp\anaconda3\lib\site-packages (3.2.2)
Requirement already satisfied: librosa in c:\users\hp\anaconda3\lib\site-packages (0.10.2.post1)
Requirement already satisfied: Python in c:\users\hp\anaconda3\lib\site-packages (3.15.0)
Requirement already satisfied: certifi>=0.19.1 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: colorbar>=0.10.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (4.55.0)
Requirement already satisfied: fpdf1olver>=1.0.1 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.20 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (1.24.3)
Requirement already satisfied: pyparsing>=2.2.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (2.4.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyrsistent>=0.1 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.1.7 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: audiotools>=1.9 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (3.0.1)
Requirement already satisfied: librosa>=0.12.0 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (0.12.0)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (1.3.0)
Requirement already satisfied: joblib>=0.14 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (1.2.0)
Requirement already satisfied: decorator>=4.3.0 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (5.1.1)
Requirement already satisfied: numpy>=1.16 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (0.57.1)
Requirement already satisfied: filelock>=3.0.12 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (3.12.3)
Requirement already satisfied: podofo>1.1 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (1.8.2)
Requirement already satisfied: sox>0.12.2 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (0.12.2)
Requirement already satisfied: typing_extensions>=3.6.1 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (4.12.2)
Requirement already satisfied: libtiff>=4.0.2 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (4.1.2)
Requirement already satisfied: msgpack>1.0 in c:\users\hp\anaconda3\lib\site-packages (from librosa) (1.0.3)
Requirement already satisfied: backcall in c:\users\hp\anaconda3\lib\site-packages (from IPython) (0.6.2)
Requirement already satisfied: jedi>0.16 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (0.18.1)
Requirement already satisfied: astroid>2.4.2 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (2.4.2)
Requirement already satisfied: asttokens>=2.1.0 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (2.1.0)
Requirement already satisfied: prompt_toolkit>=3.0.7,!=3.0.10 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (3.0.7)
Requirement already satisfied: pygments>=2.4.0 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (2.15.1)
Requirement already satisfied: traitlets>=4.3.2 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (4.3.2)
Requirement already satisfied: ipykernel>=5.1.0 in c:\users\hp\anaconda3\lib\site-packages (from IPython) (5.1.1)
Requirement already satisfied: colorama in c:\users\hp\anaconda3\lib\site-packages (from IPython) (0.4.6)
Requirement already satisfied: libmklite>=0.41 in c:\users\hp\anaconda3\lib\site-packages (from numba>=0.55.0->librosa) (0.48.0)
Requirement already satisfied: platemodel>=0.5.0 in c:\users\hp\anaconda3\lib\site-packages (from podofo>1.1->librosa) (1.18.0)
Requirement already satisfied: podofo>1.1 in c:\users\hp\anaconda3\lib\site-packages (from podofo>1.1->librosa) (1.2.0)
Requirement already satisfied: six>1.5 in c:\users\hp\anaconda3\lib\site-packages (from prompt_toolkit>=3.0.37,<3.1.0,>=3.0.39->IPython) (0.2.13)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->librosa) (2.2.0)
Requirement already satisfied: certifi>=2.0.0 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.19.0->librosa) (2.15.1)
Requirement already satisfied: futures>=3.0.0 in c:\users\hp\anaconda3\lib\site-packages (from stack-data>=IPython) (4.0.5)
Requirement already satisfied: asttokens in c:\users\hp\anaconda3\lib\site-packages (from stack-data>=IPython) (0.2.5)
Requirement already satisfied: pure_eval in c:\users\hp\anaconda3\lib\site-packages (from stack-data>=IPython) (0.2.2)
Requirement already satisfied: pycparser in c:\users\hp\anaconda3\lib\site-packages (from cffl>1.0->soundfile>=0.12.1->librosa) (2.21)
Requirement already satisfied: character-normalize>=2.0 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.19.0->podofo) (2.24.0)
Requirement already satisfied: iina>v2.5 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.19.0->podofo>1.1->librosa) (3.10)
Requirement already satisfied: urlib3>=1.21.1 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.19.0->podofo>1.1->librosa) (2.24.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.19.0->podofo>1.1->librosa) (2024.0.30)

In [1]: import os
import matplotlib.pyplot as plt
import librosa, librosa.display
import IPython.display.ipd

In [2]: BASE_FOLDER = "C:/Github/Machine_Learning/AudioSignalProcessingForML_Valero_Valerado/audio_resources"
violin_sound_file = "violin_c.wav"
piano_sound_file = "piano_c.wav"
tremolo_sound_file = "tremolo.wav"

In [3]: print(os.path.join(BASE_FOLDER, violin_sound_file))
# Load sounds
violin_c, _ = librosa.load(os.path.join(BASE_FOLDER, violin_sound_file))
piano_c, _ = librosa.load(os.path.join(BASE_FOLDER, piano_sound_file))

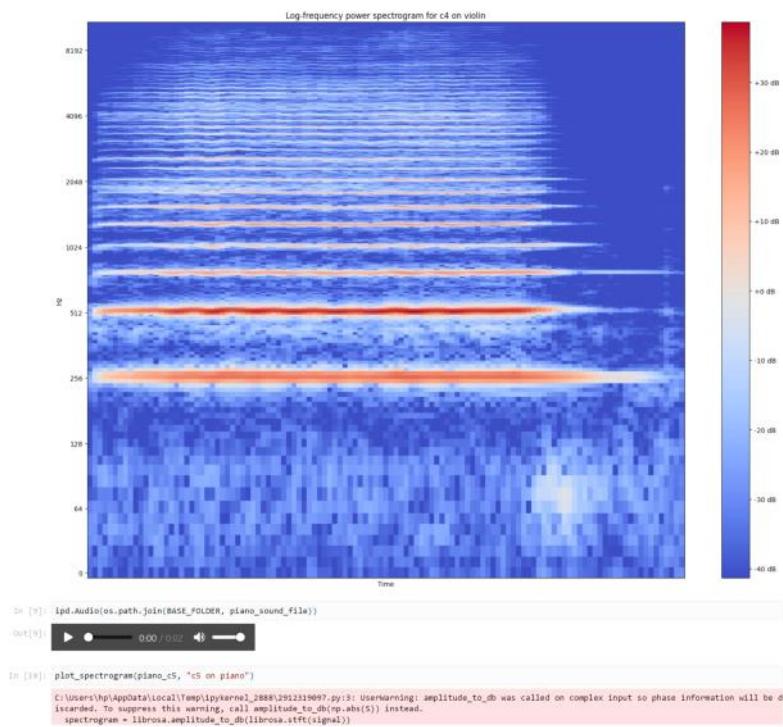
In [4]: def plot_spectrogram(signal, name):
    """Compute power spectrogram with Short-Time Fourier Transform and plot result."""
    spectrogram = librosa.amplitude_to_db(librosa.stft(signal))
    plt.figure(figsize=(10, 10))
    librosa.display.specshow(spectrogram, y_axis="log")
    plt.colorbar(format="%2.0f dB")
    plt.title(f"Log-frequency power spectrogram for {name}")
    plt.xlabel("Time")
    plt.ylabel("Frequency")

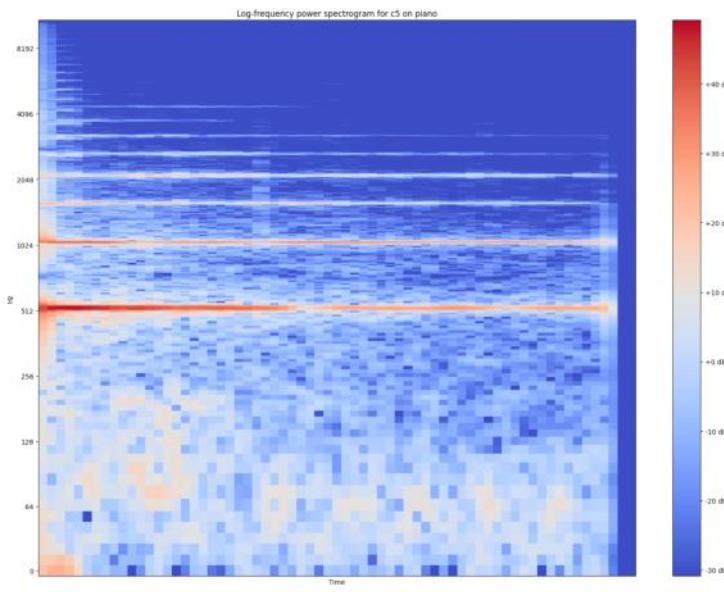
In [5]: ipd.Audio(os.path.join(BASE_FOLDER, violin_sound_file))

Out[5]:
```

In [8]: plot\_spectrogram(violin\_c, "c4 on violin")

```
C:\Users\hp\AppData\Local\Temp\ipykernel_2888\2912319097.py:3: UserWarning: amplitude_to_db was called on complex input so phase information will be discarded. To suppress this warning, call amplitude_to_db(np.abs(s)) instead.
spectrogram = librosa.amplitude_to_db(librosa.stft(signal))
```





```
In [11]: ipd.Audio(os.path.join(BASE_FOLDER, tremolo_sound_file))
```

```
Out[11]:
```

```
In [12]: import numpy as np
```

```
In [13]: X = np.fft.rfft(violin_c4)
```

```
In [14]: X_mag = np.absolute(X)
```

```
f = np.linspace(0, ..., len(X_mag))
```

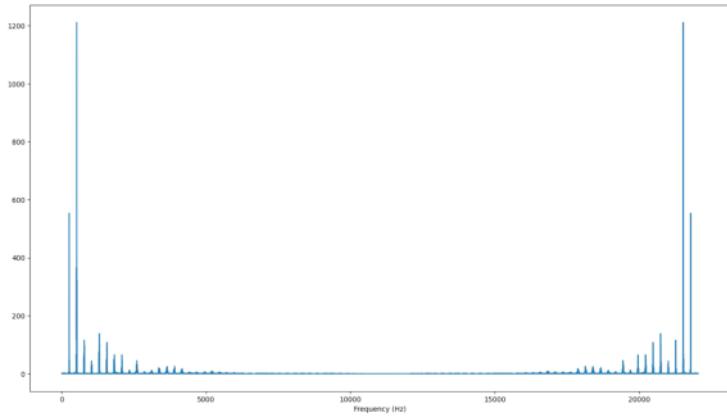
```
In [15]: plt.figure(figsize(18, 10))
```

```
plt.plot(f, X_mag) # magnitude spectrum
```

```
plt.xlabel('frequency (Hz)')
```

```
Text(0.5, 0, 'Frequency (Hz)')
```

```
Out[15]:
```



```
In [16]: len(violin_c4)
```

```
out[16]: 59772
```

```
In [17]:
```



## Understanding audio signals

# Understanding audio signals for ML

Valerio Velardo

## Audio signal

---

- Representation of sound
- Encodes all info we need to reproduce sound

**Houston we have a problem!**

---

Houston we have a problem!

---

Analog



VS

Digital



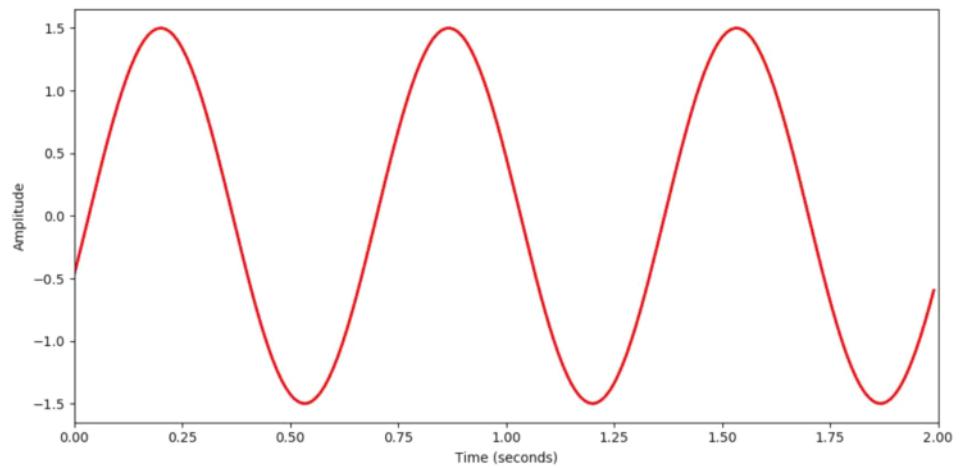
## Analog signal

---

- Continuous values for time
- Continuous values for amplitude

## Analog signal

---



## Digital signal

- Sequence of discrete values
- Data points can only take on a finite number of values

## Analog to digital conversion

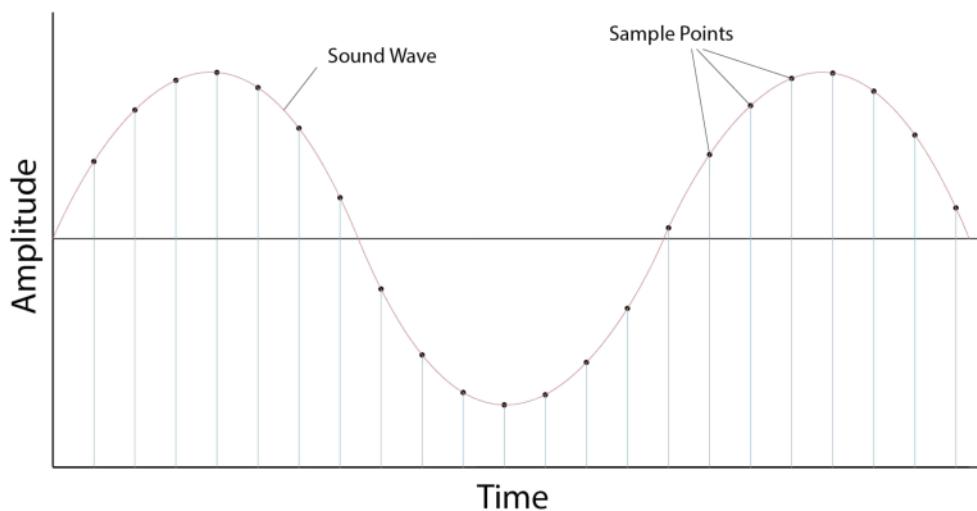
---

- Sampling
- Quantization

# Pulse-code modulation

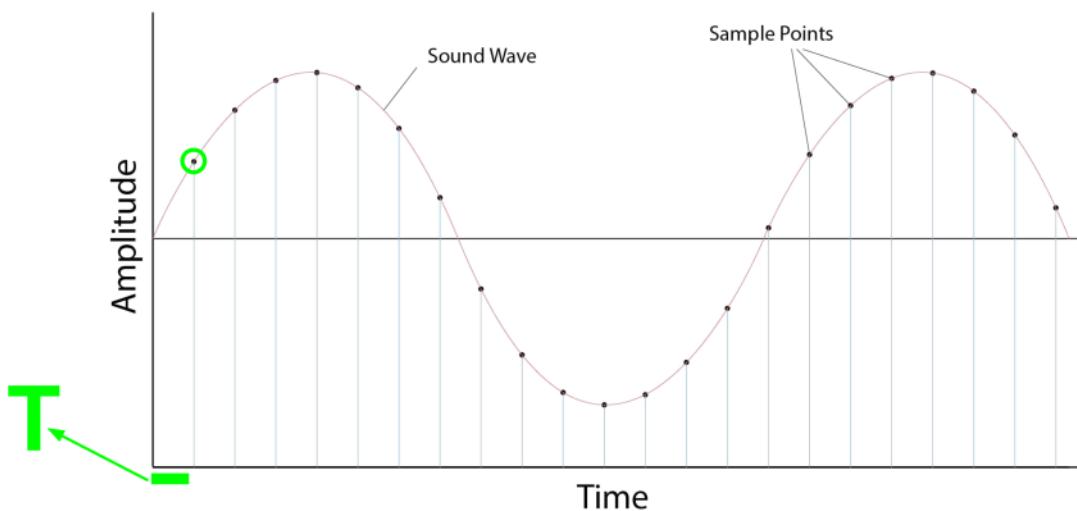
## Sampling

---



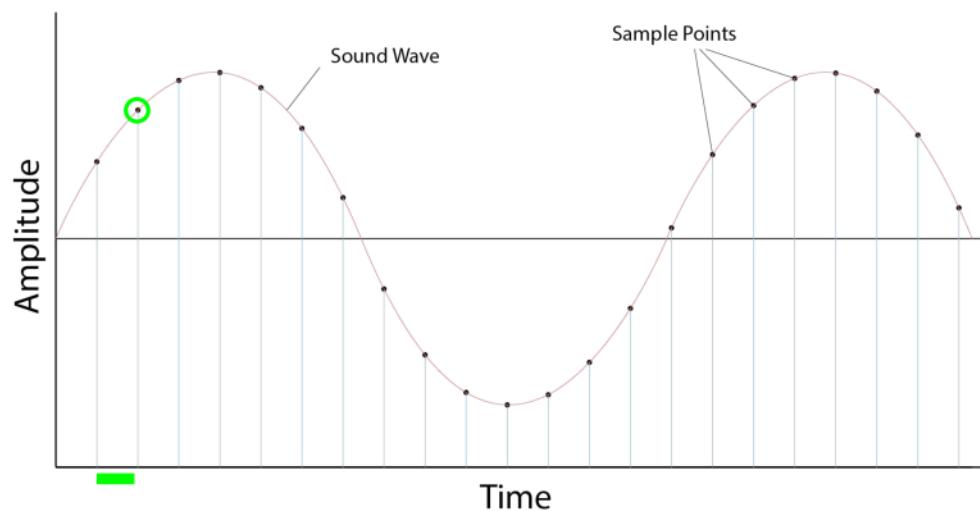
## Sampling period

---



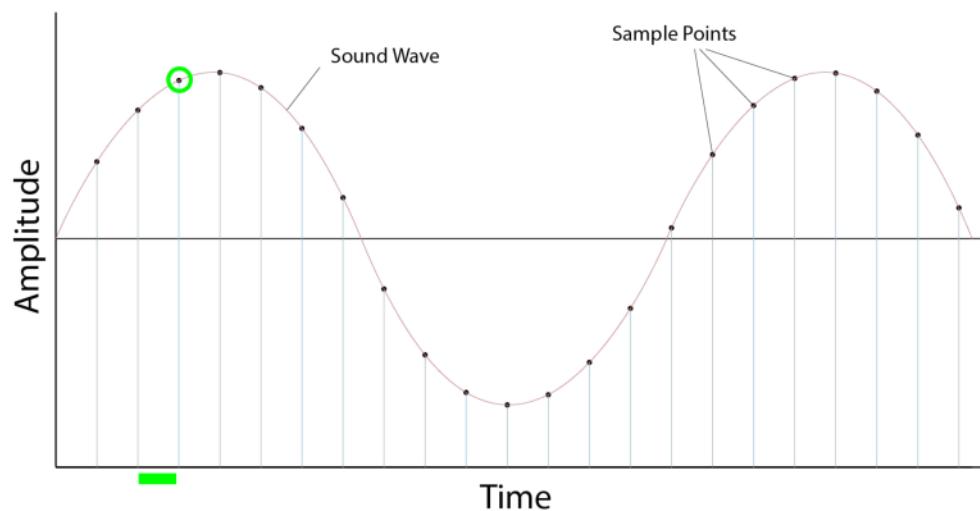
## Sampling period

---



## Sampling period

---



## Locating samples

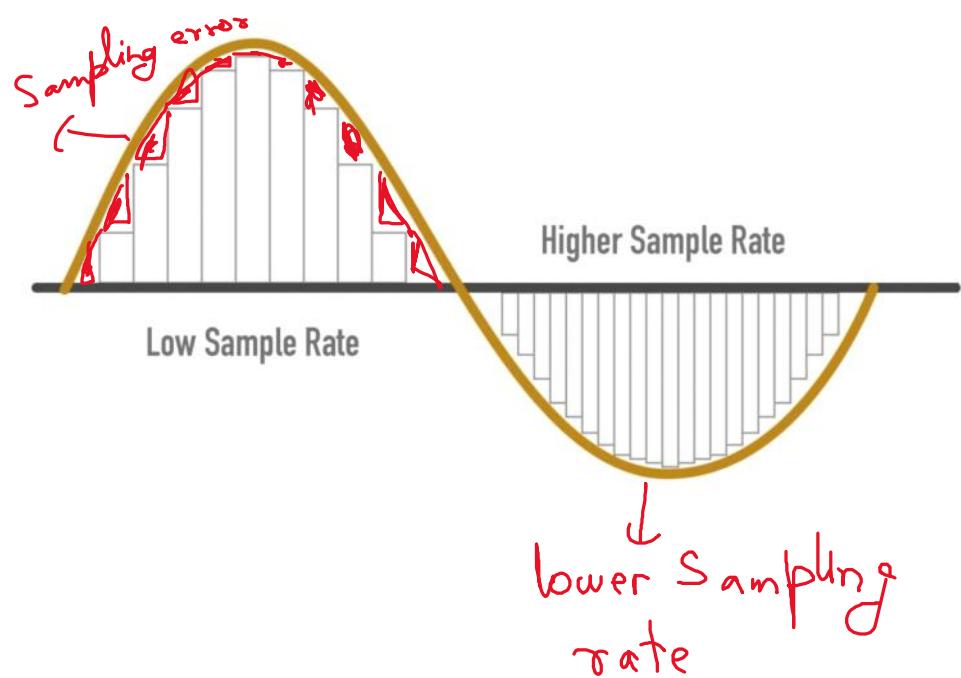
$$t_n = n \cdot T$$

## Sampling rate

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

## Sampling rate

---



Why sampling rate = 44100hz?



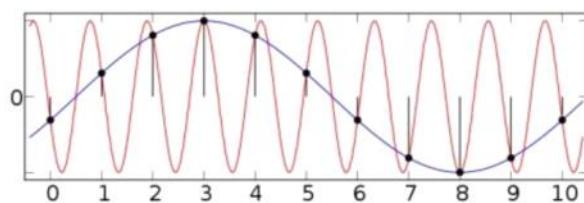
## Nyquist frequency

$$f_N = \frac{s_r}{2}$$

Nyquist frequency for CD

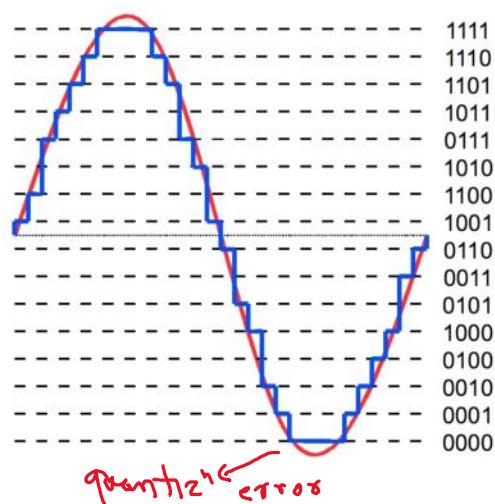
$$f_N = \frac{44100}{2} = 22050$$

## Aliasing



## Quantization

---



## Quantization

---

- Resolution = num. of bits

## Quantization

---

- Resolution = num. of bits
- Bit depth  *synonym*

## Quantization

---

- Resolution = num. of bits
- Bit depth
- **CD resolution = 16 bits**



## Quantization

---

- Resolution = num. of bits
- Bit depth
- CD resolution = 16 bits

$$2^{16} = 65536$$



*1 minute*

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100)/1,048,576)/8) \cdot 60$$

bit to byte  
↓  
to Mega

60 sec  
↓  
1 min

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100)/1,048,576)/8) \cdot 60$$

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100) / 1,048,576) / 8) \cdot 60$$

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100)/1,048,576)/8) \cdot 60$$

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100)/1,048,576)/8) \cdot 60$$

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100)/1,048,576)/8) \cdot 60$$

## Memory for 1' of sound

- Sampling rate = 44100 Hz
- Bit depth = 16 bits

$$(((16 \cdot 44,100)/1,048,576)/8) \cdot 60 = 5.49MB$$

## Dynamic range

- Difference between largest/smallest signal a system can record

## Dynamic range



resolution



dynamic range

## Signal-to-quantization-noise ratio

- Relationship between max signal strength and quantization error
- Correlates with dynamic range

## Signal-to-quantization-noise ratio

- Relationship between max signal strength and quantization error
- Correlates with dynamic range

$$SQNR \approx 6.02 \cdot Q$$

## Signal-to-quantization-noise ratio

- Relationship between max signal strength and quantization error
- Correlates with dynamic range

$$SQNR(16) \approx 96dB$$

**How do we record sound?**

---

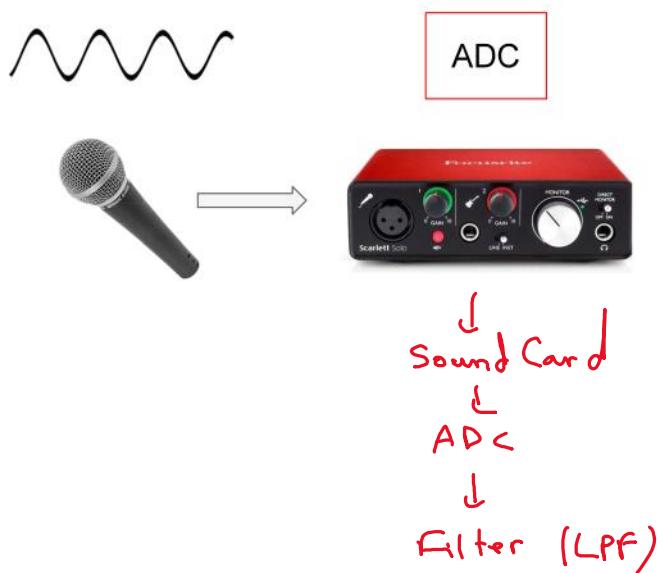
**How do we record sound?**

---



## How do we record sound?

---



## How do we record sound?

---



**How do we reproduce sound?**

---

## How do we reproduce sound?

---



## How do we reproduce sound?

---



## How do we reproduce sound?

---

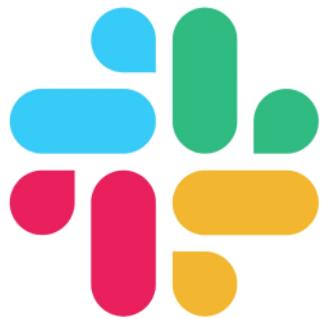


## What's up next?

- Overview of audio features

Join the community!

---



[thesoundofai.slack.com](https://thesoundofai.slack.com)

# Types Audio Features for ML

Valerio Velardo

## Why audio features?

---

- Description of sound

## Why audio features?

---

- Description of sound
- Different features capture different aspects of sound

## Why audio features?

---

- Description of sound
- Different features capture different aspects of sound
- **Build intelligent audio systems**

## Audio feature categorisation

---

- Level of abstraction
- Temporal scope
- Music aspect
- Signal domain
- ML approach

## Level of abstraction

Machine friendly



High-level

Examples: instrumentation, key, chords, melody, rhythm, tempo, lyrics, genre, mood



Mid-level

Examples: pitch- and beat-related descriptors, such as note onsets, fluctuation patterns, MFCCs



Low-level

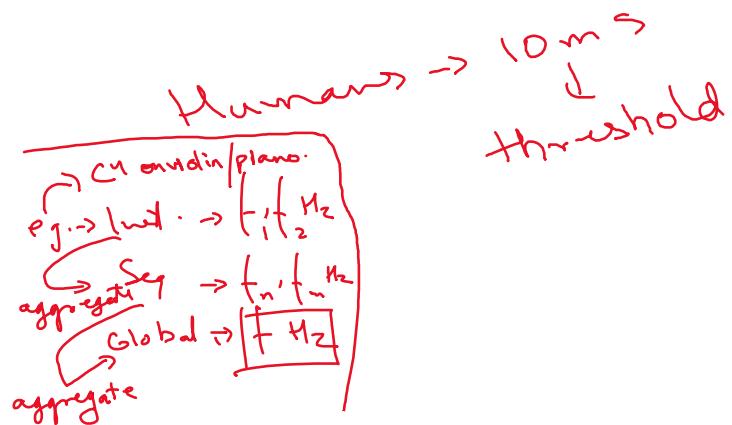
Examples: amplitude envelope, energy, spectral centroid, spectral flux, zero-crossing rate

Human friendly  
or  
abstract

Knees, P., & Schedl, M. (2016). *Music similarity and retrieval: an introduction to audio-and web-based strategies*

## Temporal scope

- Instantaneous ( $\sim 50\text{ms}$ )  $\rightarrow 20\text{-}100 \frac{\text{ms}}{\text{s}}$
- Segment-level (seconds)  $\rightarrow 2\text{-}5, 20 \text{ sec}$
- Global



## Music aspect

---

- Beat
- Timbre
- Pitch
- Harmony
- ...

## **Signal domain**

---

## Signal domain

- Time domain

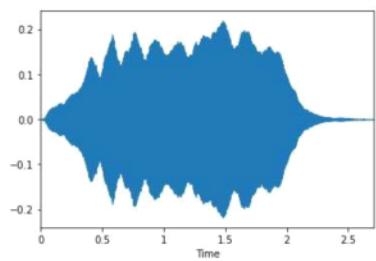
## Signal domain

- Time domain

Amplitude envelope  
Root-mean square energy  
Zero crossing rate

## Signal domain

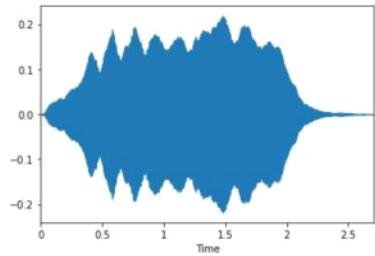
- Time domain



## Signal domain

---

- Time domain
- Frequency domain

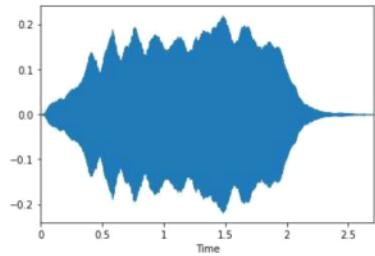


## Signal domain

---

- Time domain
- Frequency domain

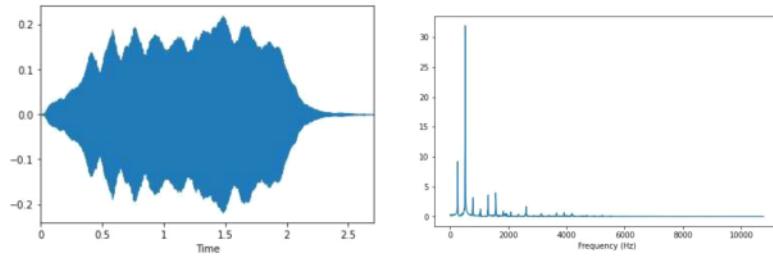
Band energy ratio  
Spectral centroid  
Spectral flux



## Signal domain

---

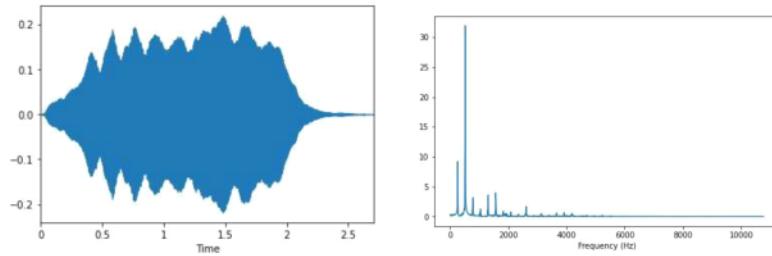
- Time domain
- Frequency domain



## Signal domain

---

- Time domain
- Frequency domain
- **Time-frequency representation**

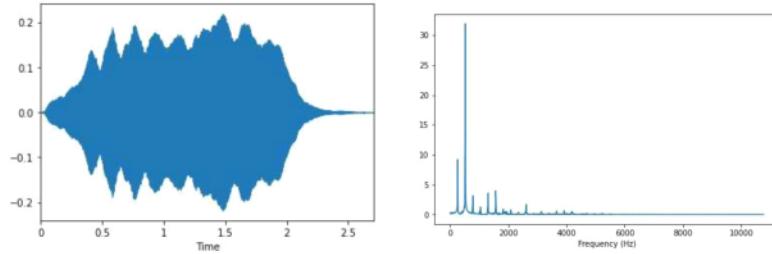


## Signal domain

---

- Time domain
- Frequency domain
- Time-frequency representation

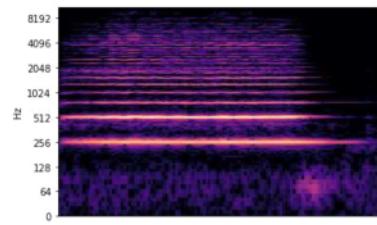
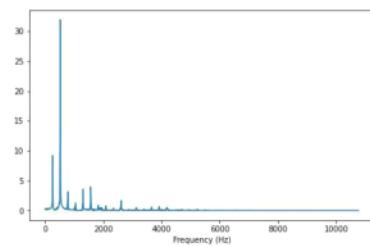
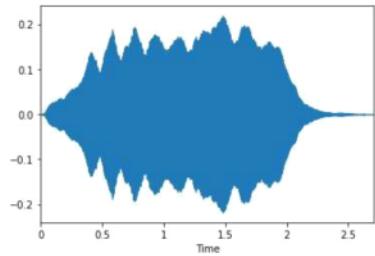
Spectrogram  
Mel-spectrogram  
Constant-Q transform



## Signal domain

---

- Time domain
- Frequency domain
- **Time-frequency representation**



## Machine learning approach

---

- Traditional machine learning
- Deep learning

## Traditional ML

---

Amplitude envelope  
Root-mean square energy  
Zero crossing rate  
Band energy ratio  
Spectral centroid  
Spectral flux  
Spectral spread  
Spectral roll-off  
...

## Traditional ML

---

Amplitude envelope

Root-mean square energy

Zero crossing rate

Band energy ratio

Spectral centroid

Spectral flux

Spectral spread

Spectral roll-off

...

## Traditional ML

---

Amplitude envelope  
Zero crossing rate  
Spectral flux

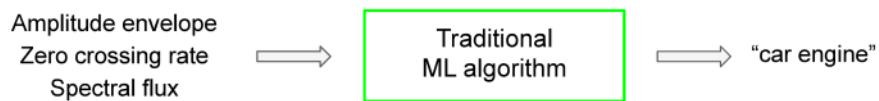
## Traditional ML

---



## Traditional ML

---

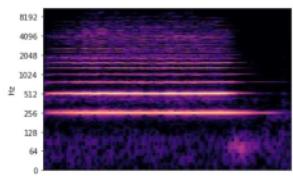


## Deep Learning

---

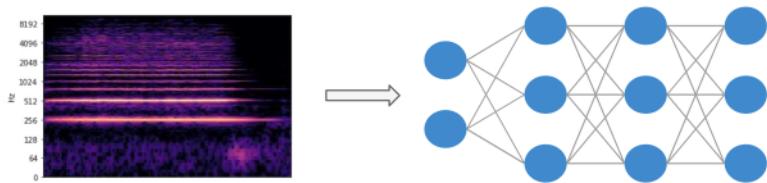
# Deep Learning

---



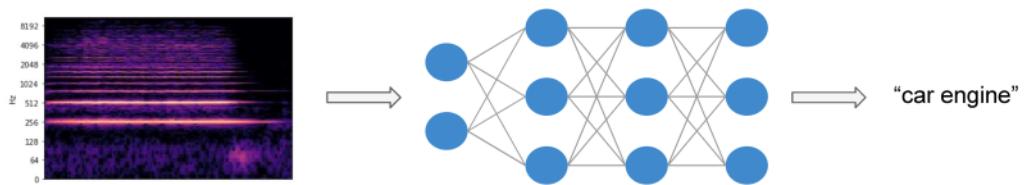
# Deep Learning

---



# Deep Learning

---

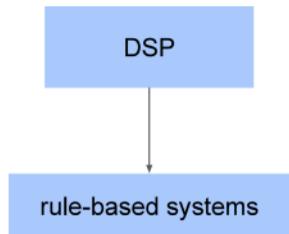


## Types of intelligent audio systems

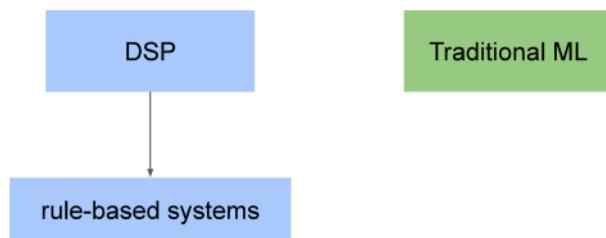
## Types of intelligent audio systems

DSP

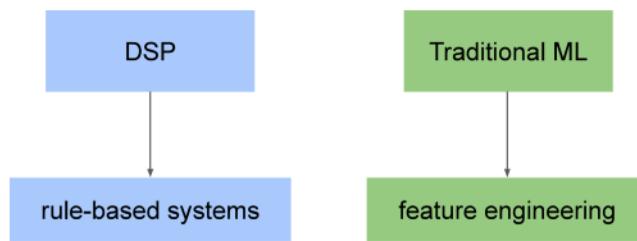
## Types of intelligent audio systems



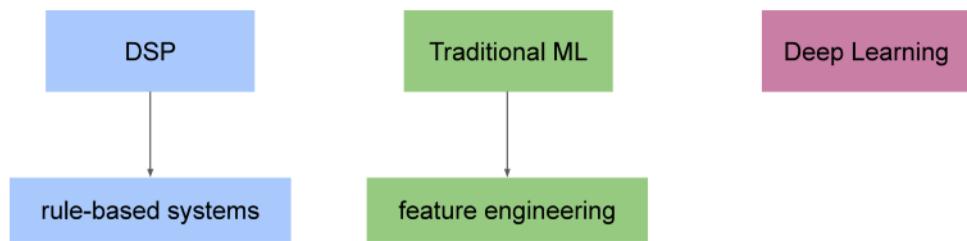
## Types of intelligent audio systems



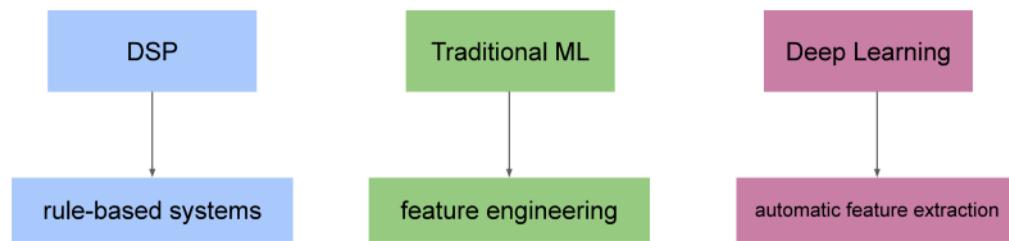
## Types of intelligent audio systems



## Types of intelligent audio systems



## Types of intelligent audio systems



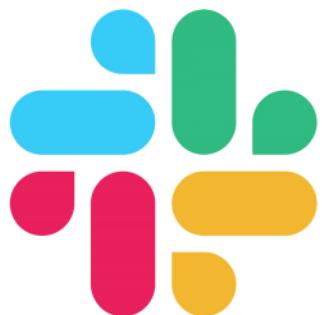
## What's up next?

---

- Feature extraction pipeline

**Join the community!**

---



**thesoundofai.slack.com**

## How do we extract audio features?

Valerio Velardo

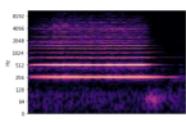
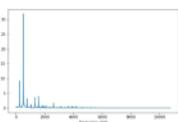
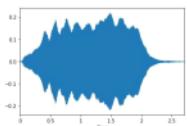
Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

Previously on Audio Processing for ML

- Time-domain features
- Frequency-domain features
- Time-frequency domain features



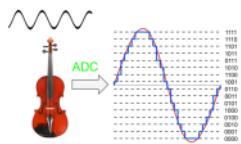
### Time-domain feature pipeline

---



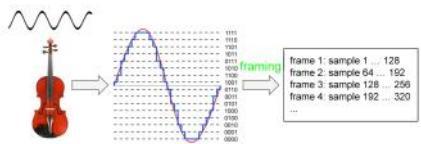
### Time-domain feature pipeline

---



### Time-domain feature pipeline

---



## Frames

---

- Perceivable audio chunk

## Frames

---

- Perceivable audio chunk

1 sample @44.1KHz = 0.0227ms

## Frames

---

- Perceivable audio chunk

1 sample @44.1KHz = 0.0227ms

Duration 1 sample << Ear's time resolution (10ms)

## Frames

---

- Perceivable audio chunk
- Power of 2 num. samples

↳ bcz when  
FFT  
↳  $2^n \rightarrow$  faster  
FFT

## Frames

---

- Perceivable audio chunk
- Power of 2 num. samples
- Typical values: 256 - 8192

## Frames

---

- Perceivable audio chunk
- Power of 2 num. samples
- Typical values: 256 - 8192

$$d_f = \frac{1}{s_r} \cdot K$$

## Frames

---

- Perceivable audio chunk
- Power of 2 num. samples
- Typical values: 256 - 8192

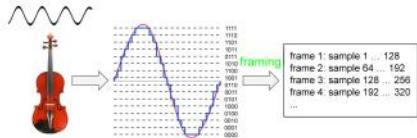
$$d_f = \frac{1}{\frac{s_r}{44100}} \cdot \boxed{K}$$

## Frames

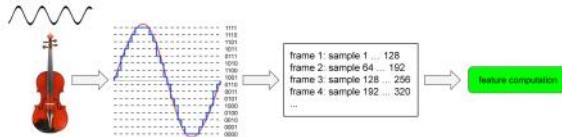
- Perceivable audio chunk
- Power of 2 num. samples
- Typical values: 256 - 8192

$$d_f = \frac{1}{S_r} \cdot \frac{512}{44100} = 11.6\text{ms}$$

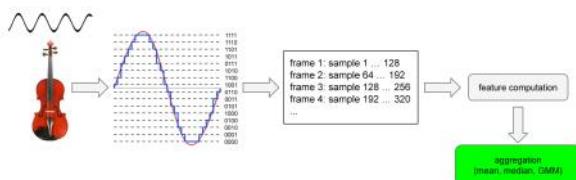
## Time-domain feature pipeline



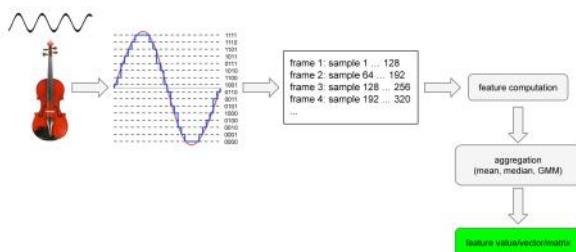
## Time-domain feature pipeline



### Time-domain feature pipeline



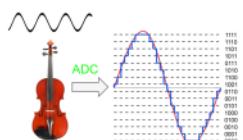
### Time-domain feature pipeline



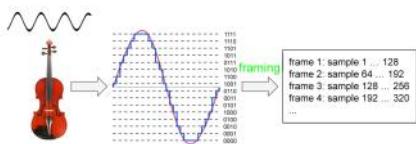
### Frequency-domain feature pipeline



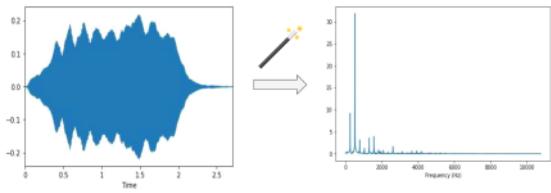
### Frequency-domain feature pipeline



## Frequency-domain feature pipeline



## From time to frequency domain



## Spectral leakage

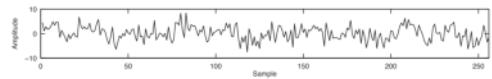
---

- Processed signal isn't an integer number of periods

## Spectral leakage

---

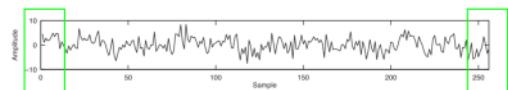
- Processed signal isn't an integer number of periods
- Endpoints are discontinuous



## Spectral leakage

---

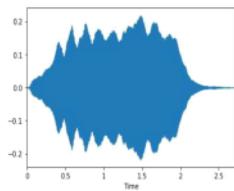
- Processed signal isn't an integer number of periods
- Endpoints are discontinuous



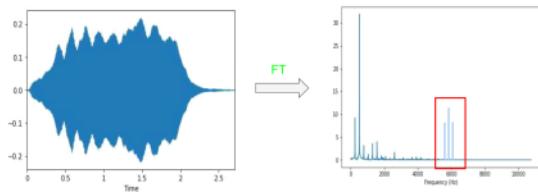
## Spectral leakage

- Processed signal isn't an integer number of periods
- Endpoints are discontinuous
- Discontinuities appear as high-frequency components not present in the original signal

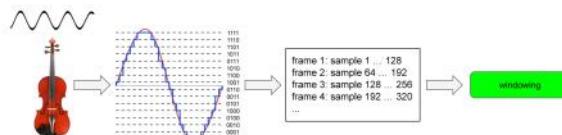
## Spectral leakage



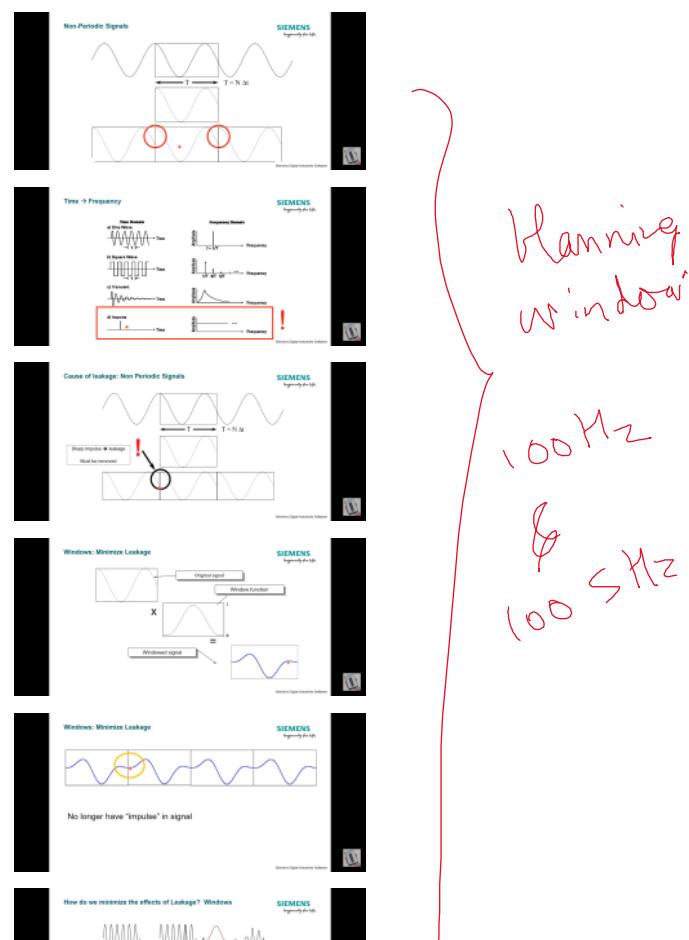
## Spectral leakage



## Frequency-domain feature pipeline

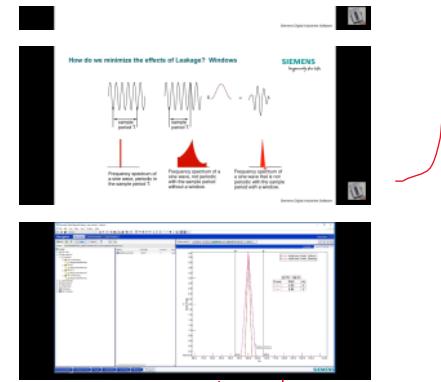


<https://youtu.be/pD7f6X9-Kg?si=Nm1L2vUe8Et9zVrZ>



## Windowing

- Apply windowing function to each frame



100Hz only  
→ windowing distorts  
the original, periodic  
sig

→ Can we make a NN  
to remove a periodicity  
instead of windowing

↳ or generate/interpolate

## Windowing

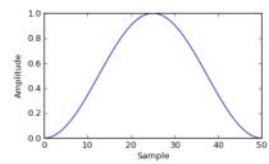
- Apply windowing function to each frame
- Eliminates samples at both ends of a frame

## Windowing

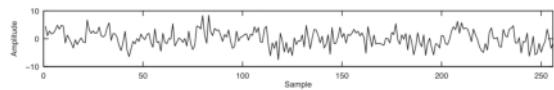
- Apply windowing function to each frame
- Eliminates samples at both ends of a frame
- Generates a periodic signal

## Hann window

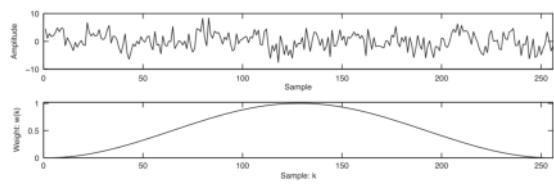
$$w(k) = 0.5 \cdot (1 - \cos(\frac{2\pi k}{K-1})), k = 1 \dots K$$



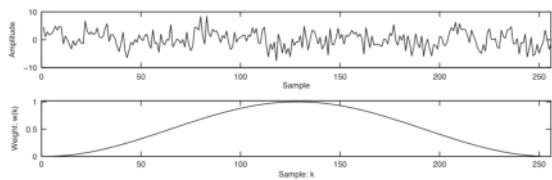
## Windowing



## Windowing

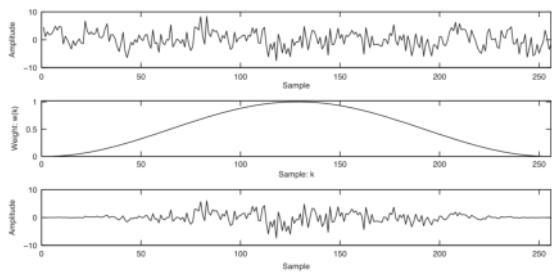


## Windowing

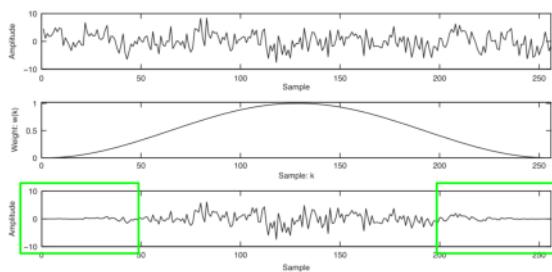


$$s_w(k) = s(k) \cdot w(k), k = 1 \dots K$$

## Windowing



## Windowing

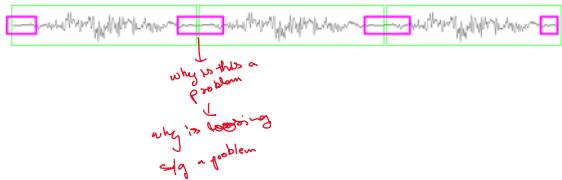


Houston we have another problem!

Houston we have another problem!



Houston we have another problem!



Non-overlapping frames



Non-overlapping frames



Non-overlapping frames

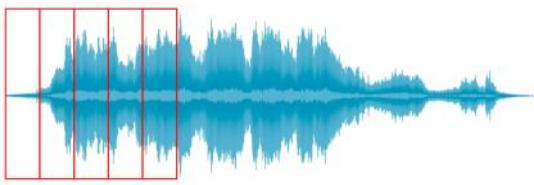


Non-overlapping frames



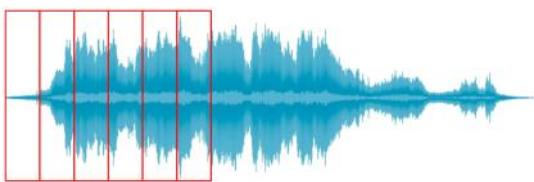
Non-overlapping frames

---



Non-overlapping frames

---



Overlapping frames

---



Overlapping frames

---



Overlapping frames

---



Overlapping frames

---



Overlapping frames

---



### Overlapping frames



### Overlapping frames



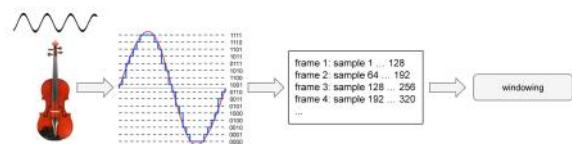
### Overlapping frames



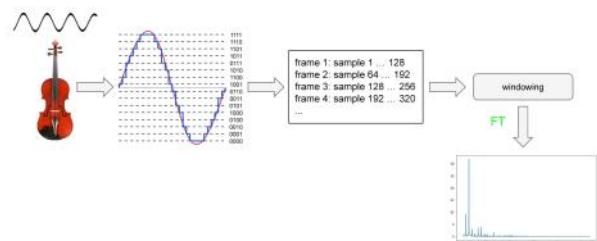
## Overlapping frames



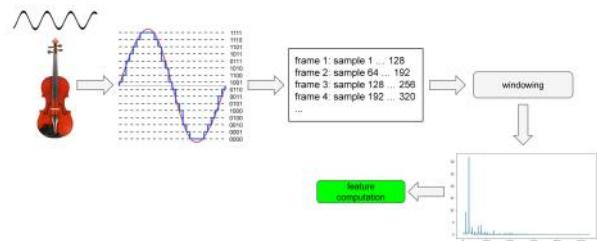
## Frequency-domain feature pipeline



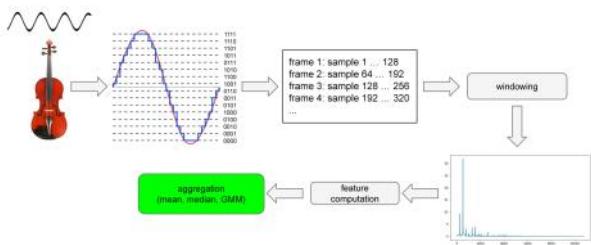
## Frequency-domain feature pipeline



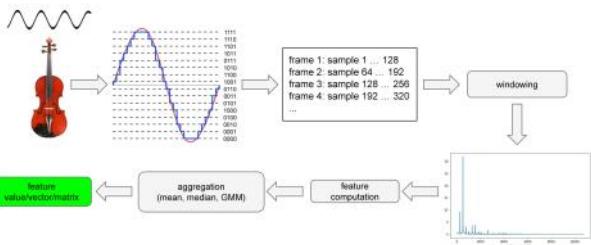
## Frequency-domain feature pipeline



## Frequency-domain feature pipeline



## Frequency-domain feature pipeline



## What's up next?

- Time-domain features

# Time-domain audio features

Valerio Velardo

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

## Time-domain features

---

- Amplitude envelope (AE)
- Root-mean-square energy (RMS)
- Zero-crossing rate (ZCR)
- ...

## Amplitude envelope

- Max amplitude value of all samples in a frame

## Amplitude envelope

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

max of all  $s(k)$   $k = t \cdot K + 1 \text{ to } (t+1)K - 1$

## Amplitude envelope

---

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

Amplitude envelope  
at frame  $t$

## Amplitude envelope

---

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

Amplitude envelope  
at frame  $t$

Amplitude of  
kth sample

## Amplitude envelope

---

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

Amplitude envelope  
at frame  $t$

Amplitude of  
kth sample

Frame size

## Amplitude envelope

---

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

Amplitude envelope  
at frame  $t$

Amplitude of  
kth sample

First sample of frame  $t$

## Amplitude envelope

---

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

Annotations:

- Amplitude envelope at frame  $t$  (red box around  $AE_t$ )
- Last sample of frame  $t$  (orange box around  $(t+1) \cdot K - 1$ )
- First sample of frame  $t$  (blue box around  $k=t \cdot K$ )
- Amplitude of  $k$ th sample (green box around  $s(k)$ )

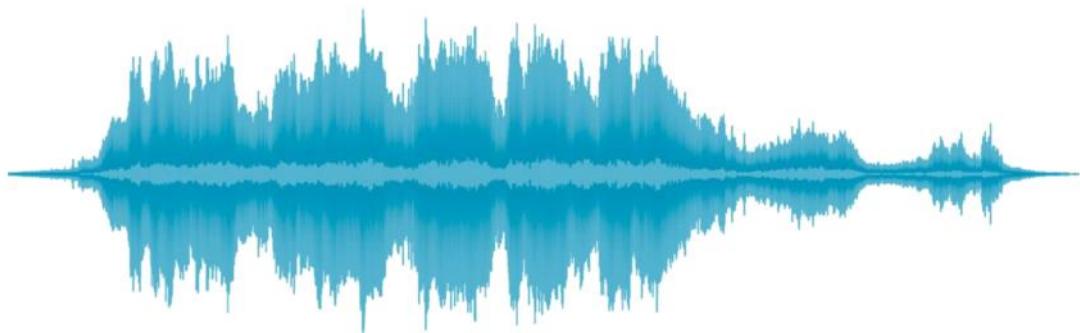
## Amplitude envelope

- Max amplitude value of all samples in a frame

$$AE_t = \max_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)$$

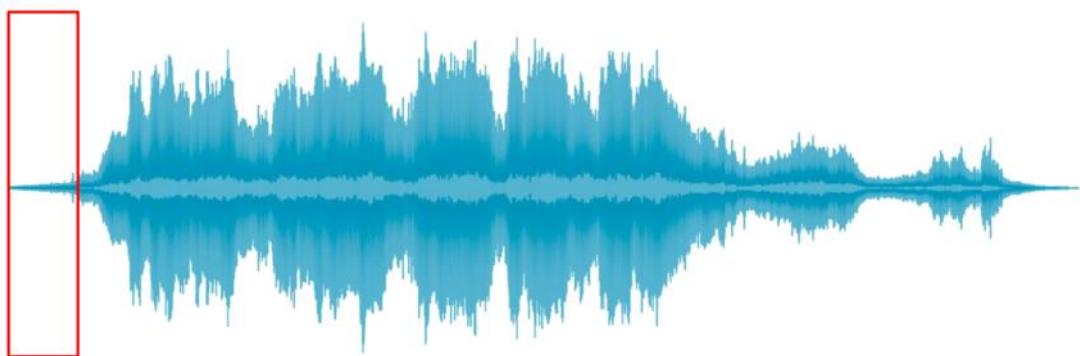
[Calculate AE for all the frames](#)

## Amplitude envelope



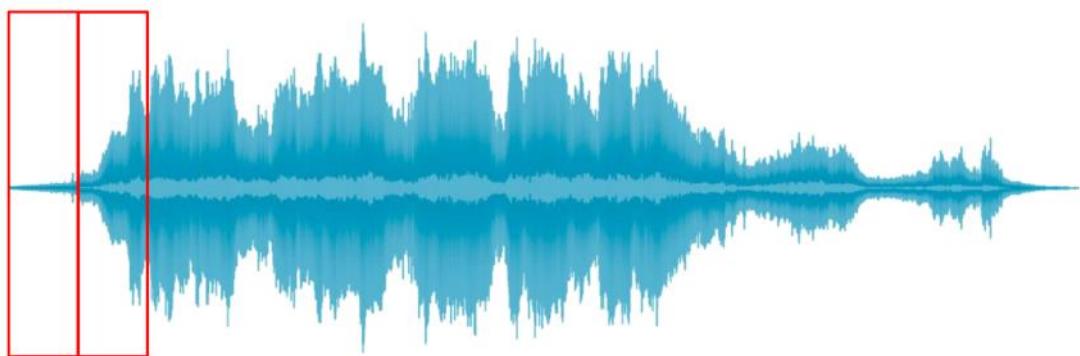
## Amplitude envelope

---



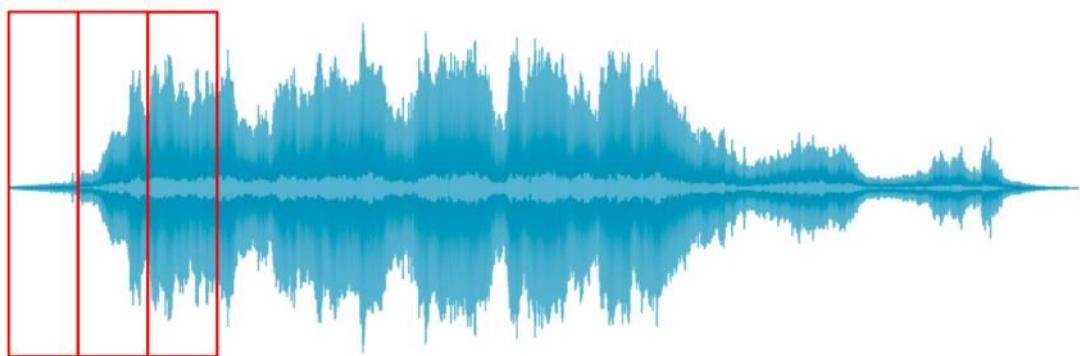
## Amplitude envelope

---



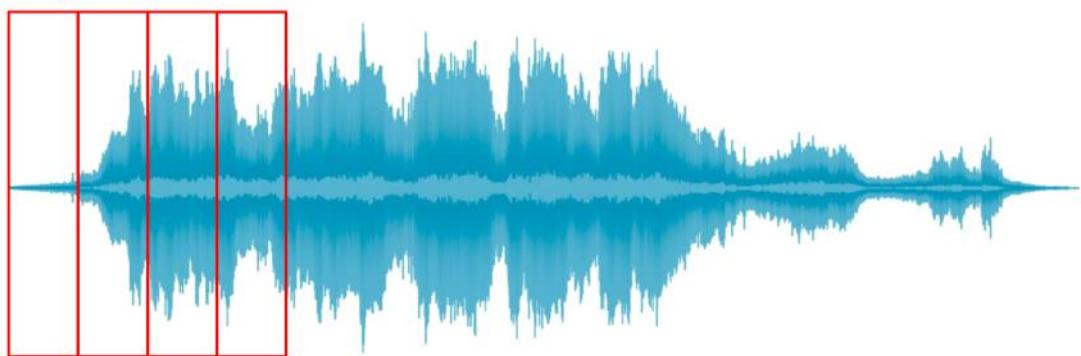
## Amplitude envelope

---



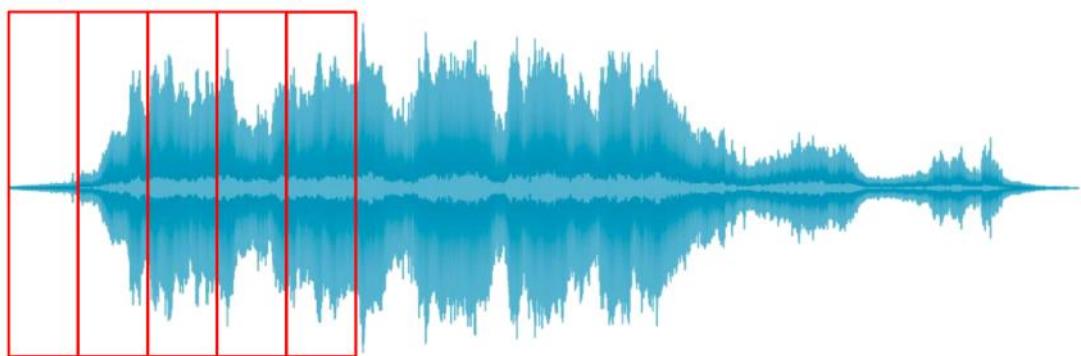
## Amplitude envelope

---



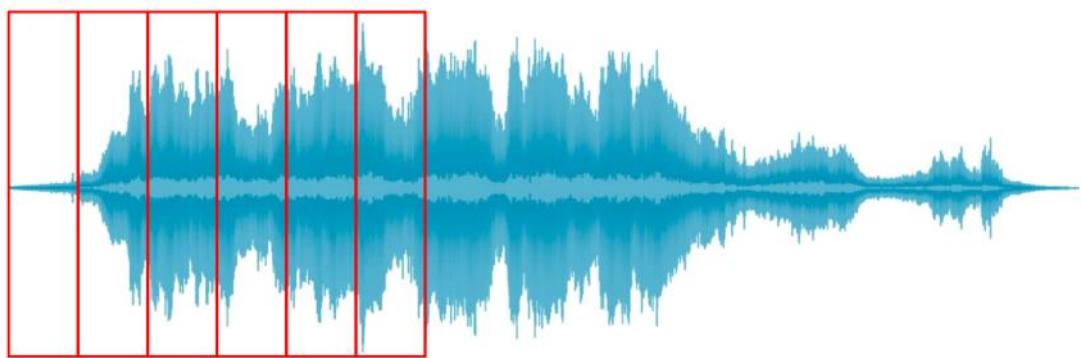
## Amplitude envelope

---



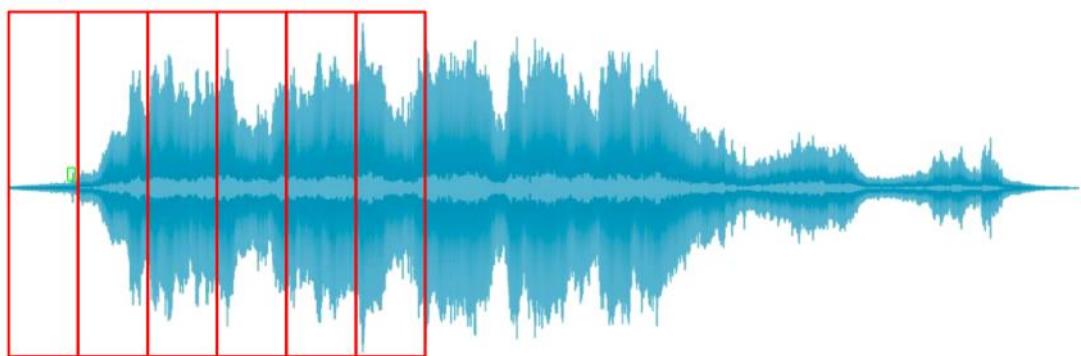
## Amplitude envelope

---



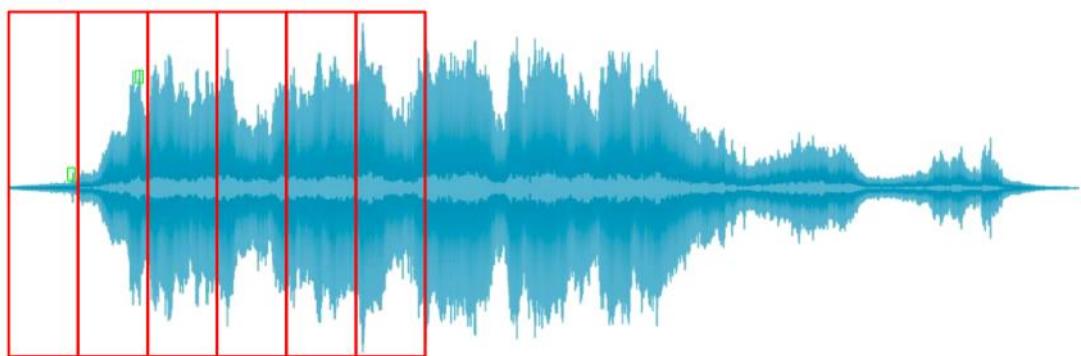
## Amplitude envelope

---



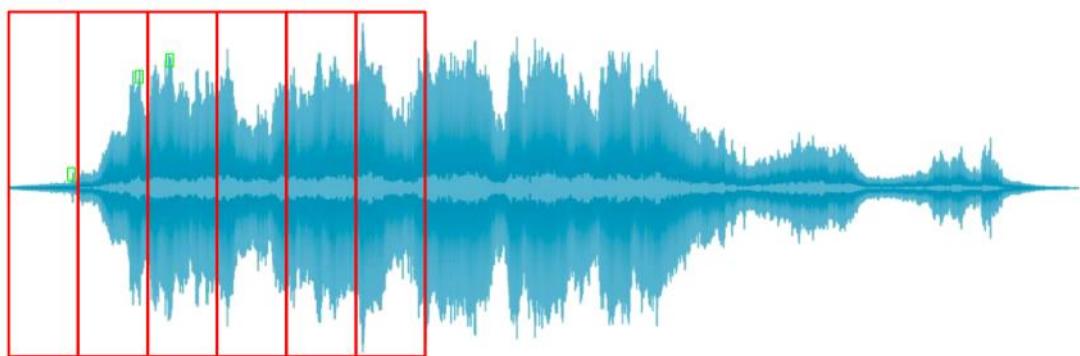
## Amplitude envelope

---



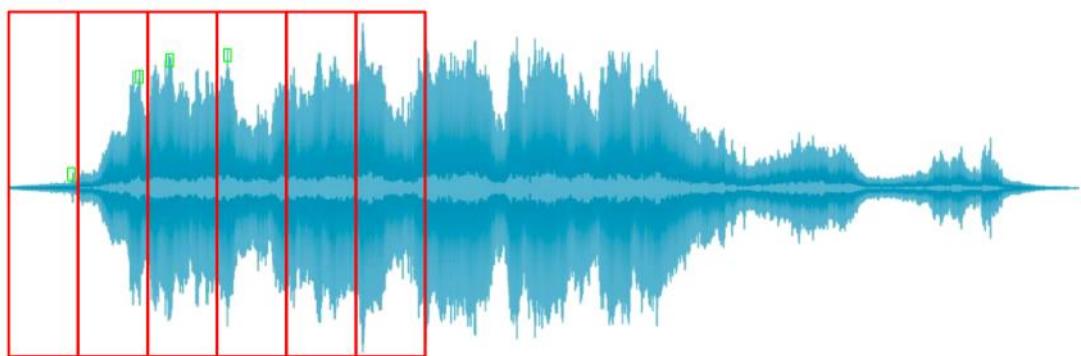
## Amplitude envelope

---



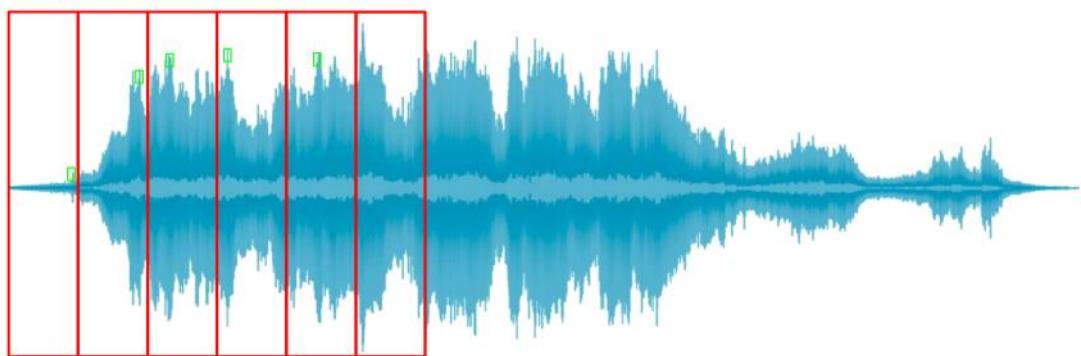
## Amplitude envelope

---



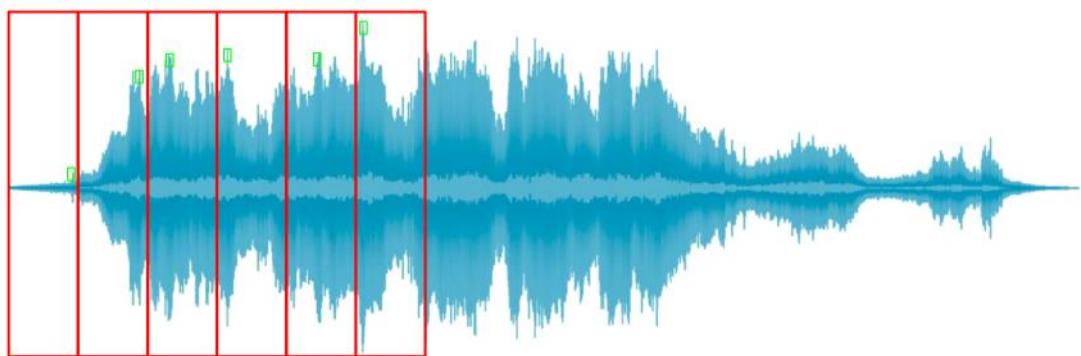
## Amplitude envelope

---



## Amplitude envelope

---



## Amplitude envelope

---

- Max amplitude value of all samples in a frame
- Gives rough idea of loudness

## Amplitude envelope

---

- Max amplitude value of all samples in a frame
- Gives rough idea of loudness
- Sensitive to outliers

## Amplitude envelope

---

- Max amplitude value of all samples in a frame
- Gives rough idea of loudness
- Sensitive to outliers
- Onset detection, music genre classification

## Root-mean-square energy

---

- RMS of all samples in a frame

## Root-mean-square energy

- RMS of all samples in a frame

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)^2}$$

## Root-mean-square energy

---

- RMS of all samples in a frame

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)^2}$$

Energy of kth sample

## Root-mean-square energy

---

- RMS of all samples in a frame

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)^2}$$

Sum of energy for all  
samples in frame  $t$

## Root-mean-square energy

---

- RMS of all samples in a frame

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} s(k)^2}$$

Mean of sum of energy

## Root-mean-square energy

---

- RMS of all samples in a frame
- Indicator of loudness

## Root-mean-square energy

---

- RMS of all samples in a frame
- Indicator of loudness
- Less sensitive to outliers than AE

## Root-mean-square energy

- RMS of all samples in a frame
- Indicator of loudness
- Less sensitive to outliers than AE
- Audio segmentation, music genre classification

↳ Someone is  
talking or not  
talking.

## Zero crossing rate

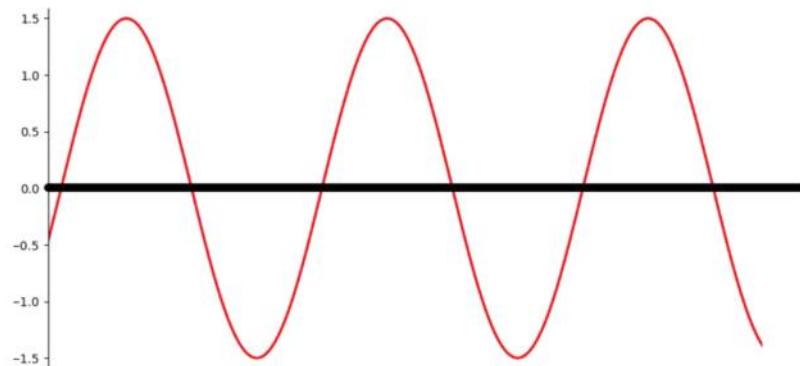
---

- Number of times a signal crosses the horizontal axis

## Zero crossing rate

---

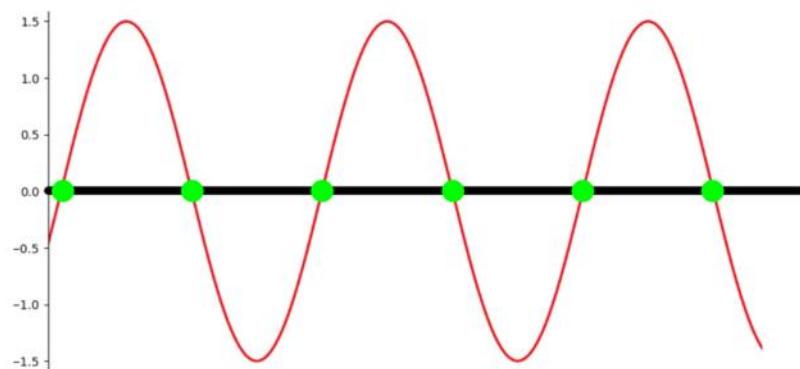
- Number of times a signal crosses the horizontal axis



## Zero crossing rate

---

- Number of times a signal crosses the horizontal axis



## Zero crossing rate

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | \operatorname{sgn}(s(k)) - \operatorname{sgn}(s(k + 1)) |$$

## Zero crossing rate

---

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | \boxed{\operatorname{sgn}(s(k))} - \operatorname{sgn}(s(k + 1)) |$$

Sign function:  
•  $s(k) > 0 \rightarrow +1$   
•  $s(k) < 0 \rightarrow -1$   
•  $s(k) = 0 \rightarrow 0$

## Zero crossing rate

---

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | \boxed{\operatorname{sgn}(s(k))} - \operatorname{sgn}(s(k + 1)) |$$

## Zero crossing rate

---

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | \boxed{\operatorname{sgn}(s(k))} - \boxed{\operatorname{sgn}(s(k + 1))} |$$

## Zero crossing rate

---

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | sgn(s(k)) + sgn(s(k+1)) |$$

## Zero crossing rate

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} \left| \textcolor{green}{\boxed{\operatorname{sgn}(s(k))}} - \textcolor{magenta}{\boxed{\operatorname{sgn}(s(k + 1))}} \right|$$

**0**

## Zero crossing rate

---

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | sgn(s(k)) - sgn(s(k + 1)) |$$

## Zero crossing rate

- Number of times a signal crosses the horizontal axis

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K - 1} | sgn(s(k)) - sgn(s(k + 1)) |$$

2

## Zero crossing rate applications

---

- Recognition of percussive vs pitched sounds

## Zero crossing rate applications

---

- Recognition of percussive vs pitched sounds
- Monophonic pitch estimation

## Zero crossing rate applications

---

- Recognition of percussive vs pitched sounds
- Monophonic pitch estimation
- Voice/unvoiced decision for speech signals

## What's up next?

---

- Implement amplitude envelope
- Visualise amplitude envelope for different music genres

## 08 Implementing the amplitude envelope\_ipynb

04 January 2025 15:42



c08\_ipynb

```
In [1]: import librosa
import librosa.display
import IPython.display as ipd
import matplotlib.pyplot as plt
import numpy as np

In [2]: debussy_file = "audio/debussy.wav"
redhot_file = "audio/redhot.wav"
duke_file = "audio/duke.wav"

In [3]: ipd.Audio(debussy_file)
Out[3]: ▶ ● 0:00 / 0:30 🔍 ●

In [4]: ipd.Audio(redhot_file)
Out[4]: ▶ ● 0:00 / 0:30 🔍 ●

In [5]: ipd.Audio(duke_file)
Out[5]: ▶ ● 0:00 / 0:30 🔍 ●

In [6]: debussy, sr = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)

In [7]: debussy
Out[7]: array([-0.01742554, -0.03567505, -0.04995728, ...,  0.00912476,
   0.00866699,  0.00964355], dtype=float32)

In [8]: debussy.size
Out[8]: 661500

In [9]: sample_duration = 1/sr
print(f"duration of one sample is {sample_duration:.6f}")
duration of one sample is 0.000045

In [10]: duration = sample_duration * len(debussy)
print(f"duration of audio is {duration:.6f}")
duration of audio is 30.000000

In [11]: plt.figure(figsize=(15,17))

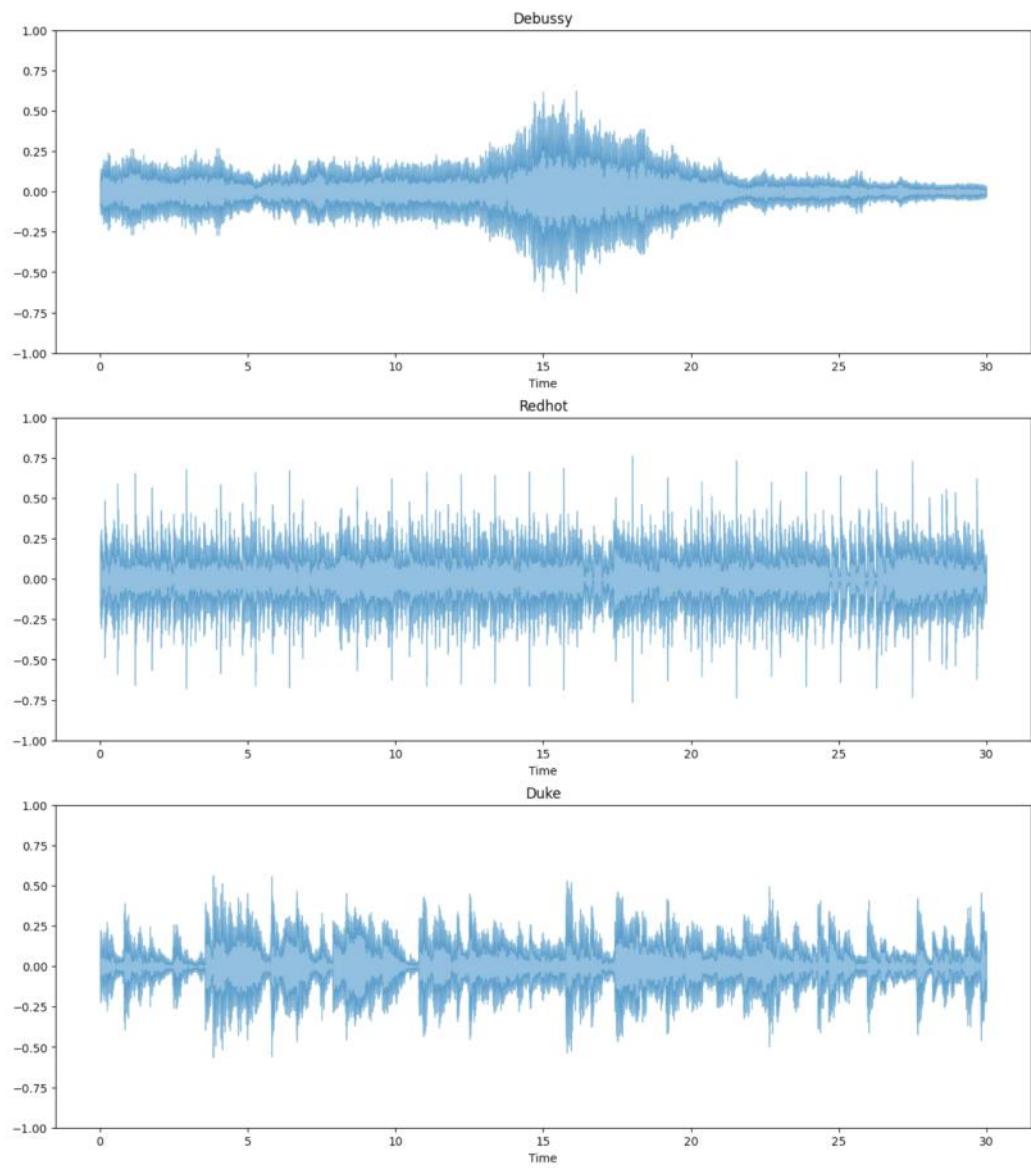
plt.subplot(3,1,1)
librosa.display.waveform(debussy, alpha=0.5)
plt.title("Debussy")
plt.ylim(-1,1)

plt.subplot(3,1,2)
librosa.display.waveform(redhot, alpha=0.5)
plt.title("Redhot")
plt.ylim(-1,1)

plt.subplot(3,1,3)
librosa.display.waveform(duke, alpha=0.5)
plt.title("Duke")
plt.ylim(-1,1)

#, alpha=0.5 for transparency
plt.show

Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
In [32]: FRAME_SIZE = 1024
HOP_LENGTH = 512

def amplitude_envelope(signal, frame_size, hop_length):
    amplitude_envelope = []
    for i in range(0, len(signal), hop_length):
        current_frame_amplitude_envelope = max(signal[i:i+frame_size])
        amplitude_envelope.append(current_frame_amplitude_envelope)
    return np.array(amplitude_envelope)

In [33]: def fancy_amplitude_envelope(signal, frame_size, hop_length):
    """Fancier Python code to calculate the amplitude envelope of a signal with a given frame size."""
    return np.array([max(signal[i:i+frame_size]) for i in range(0, len(signal), hop_length)])

In [34]: ae_debussy = amplitude_envelope(debussy, FRAME_SIZE, HOP_LENGTH)
len(ae_debussy)

Out[34]: 1292
```

```

In [35]: fancy_ae_debussy = fancy_amplitude_envelope(debussy, FRAME_SIZE, HOP_LENGTH)
len(ae_debussy)
Out[35]: 1292

In [36]: (ae_debussy == fancy_ae_debussy).all()
Out[36]: True

In [37]: # calculate amplitude envelope for RHCP and Duke Ellington
ae_redhot = amplitude_envelope(redhot, FRAME_SIZE, HOP_LENGTH)
ae_duke = amplitude_envelope(duke, FRAME_SIZE, HOP_LENGTH)

In [38]: frames = range(len(ae_debussy))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)

In [39]: # amplitude envelope is graphed in red
plt.figure(figsize=(15, 17))

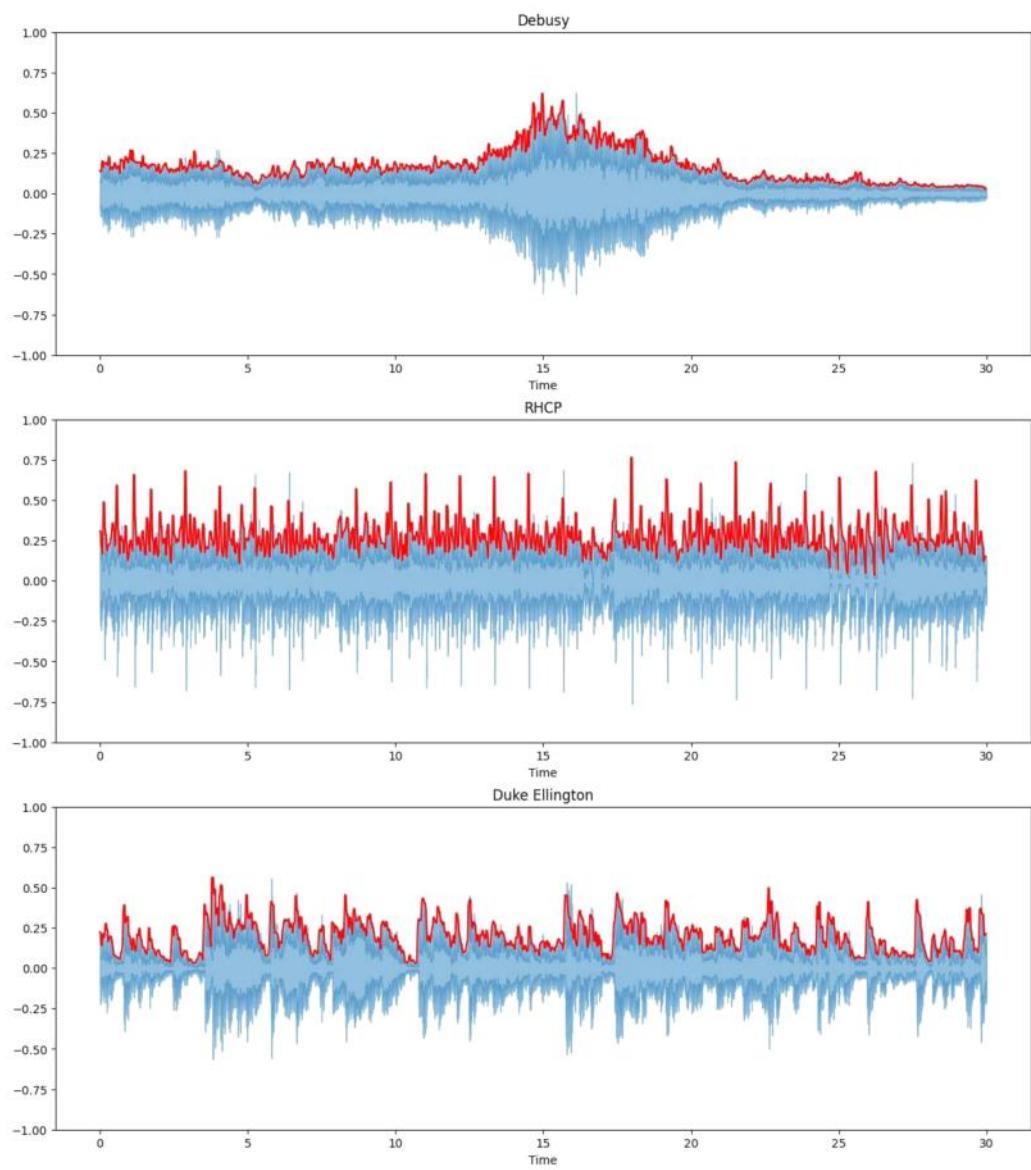
ax = plt.subplot(3, 1, 1)
librosa.display.waveplot(debussy, alpha=0.5)
plt.plot(t, ae_debussy, color="r")
plt.ylim((-1, 1))
plt.title("Debussy")

plt.subplot(3, 1, 2)
librosa.display.waveplot(redhot, alpha=0.5)
plt.plot(t, ae_redhot, color="r")
plt.ylim((-1, 1))
plt.title("RHCP")

plt.subplot(3, 1, 3)
librosa.display.waveplot(duke, alpha=0.5)
plt.plot(t, ae_duke, color="r")
plt.ylim((-1, 1))
plt.title("Duke Ellington")

plt.show()

```



## Implementing the amplitude envelope\_ipynb



# Implementing the amplitude envelope\_ipynb

```
In [1]: import matplotlib.pyplot as plt
import numpy as np
import librosa
import librosa.display
import IPython.display as ipd
```

## Loading audio files

```
In [2]: debussy_file = "audio/debussy.wav"
redhot_file = "audio/redhot.wav"
duke_file = "audio/duke.wav"
```

```
In [3]: ipd.Audio(debussy_file)
```



```
In [4]: ipd.Audio(redhot_file)
```



```
In [5]: ipd.Audio(duke_file)
```



```
In [6]: # Load audio files with Librosa
debussy, sr = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)
```

## Basic information regarding audio files

```
In [7]: debussy.shape
```

```
Out[7]: (661500,)
```

```
In [8]: # duration in seconds of 1 sample
sample_duration = 1 / sr
print(f"One sample lasts for {sample_duration:.6f} seconds")
```

One sample lasts for 0.000045 seconds

```
In [9]: # total number of samples in audio file
tot_samples = len(debussy)
tot_samples
```

```
Out[9]: 661500
```

```
In [10]: # duration of debussy audio in seconds
duration = 1 / sr * tot_samples
print(f"The audio lasts for {duration} seconds")
```

The audio lasts for 30.0 seconds

## Visualising audio signal in the time domain

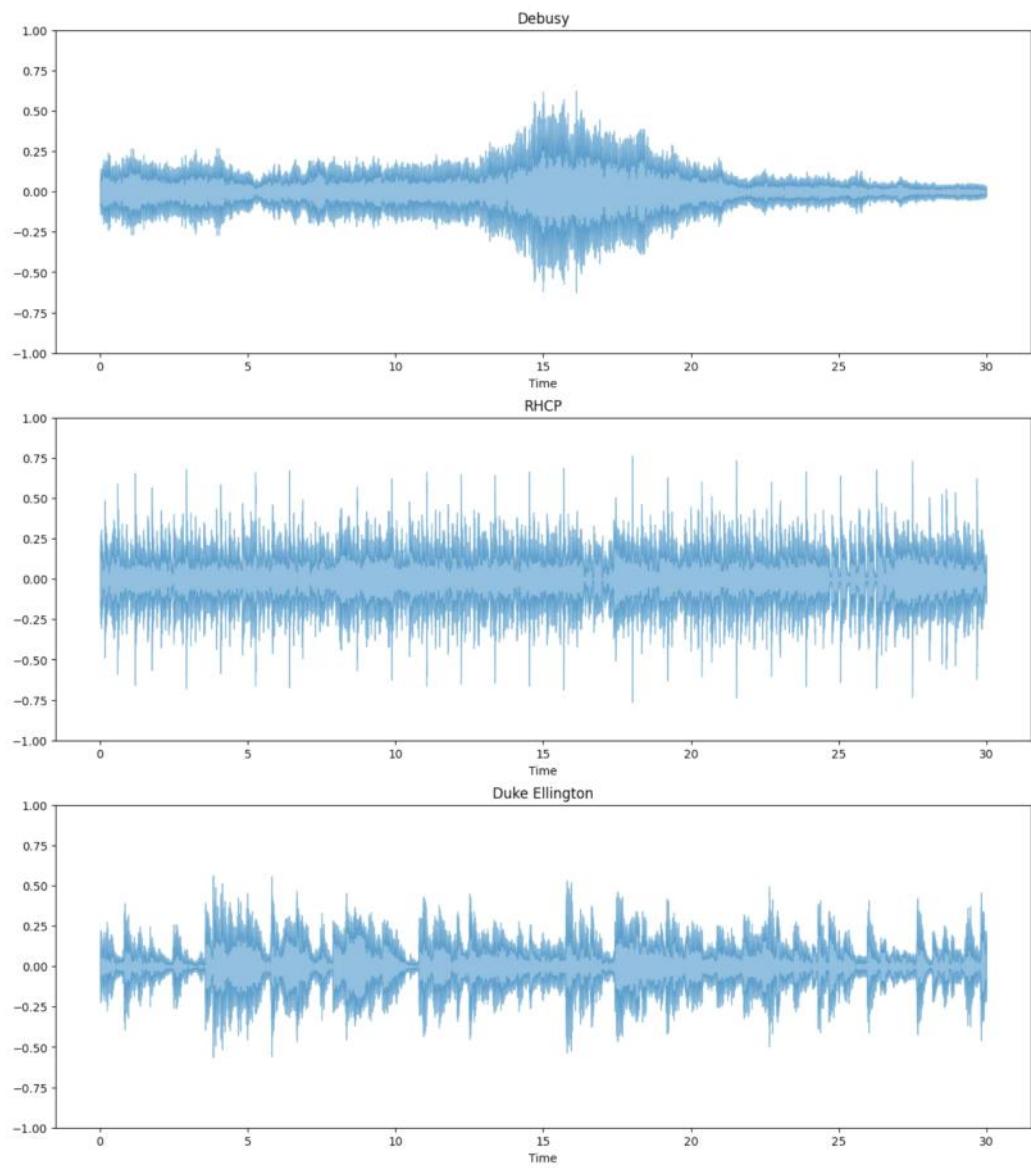
```
In [12]: plt.figure(figsize=(15, 17))

plt.subplot(3, 1, 1)
librosa.display.waveplot(debussy, alpha=0.5)
plt.ylim((-1, 1))
plt.title("Debussy")

plt.subplot(3, 1, 2)
librosa.display.waveplot(redhot, alpha=0.5)
plt.ylim((-1, 1))
plt.title("RHCP")

plt.subplot(3, 1, 3)
librosa.display.waveplot(duke, alpha=0.5)
plt.ylim((-1, 1))
plt.title("Duke Ellington")

plt.show()
```



### Calculating amplitude envelope

```
In [12]: FRAME_SIZE = 1024
HOP_LENGTH = 512

def amplitude_envelope(signal, frame_size, hop_length):
    """Calculate the amplitude envelope of a signal with a given frame size nad hop length."""
    amplitude_envelope = []
    # calculate amplitude envelope for each frame
    for i in range(0, len(signal), hop_length):
        amplitude_envelope_current_frame = max(signal[i:i+frame_size])
        amplitude_envelope.append(amplitude_envelope_current_frame)
    return np.array(amplitude_envelope)

In [13]: def fancy_amplitude_envelope(signal, frame_size, hop_length):
    """Fancier Python code to calculate the amplitude envelope of a signal with a given frame size."""
    return np.array([max(signal[i:i+frame_size]) for i in range(0, len(signal), hop_length)])
```

```
In [14]: # number of frames in amplitude envelope
ae_debussy = amplitude_envelope(debussy, FRAME_SIZE, HOP_LENGTH)
len(ae_debussy)
```

```
Out[14]: 1292
```

```
In [15]: # calculate amplitude envelope for RHCP and Duke Ellington
ae_redhot = amplitude_envelope(redhot, FRAME_SIZE, HOP_LENGTH)
ae_duke = amplitude_envelope(duke, FRAME_SIZE, HOP_LENGTH)
```

### Visualising amplitude envelope

```
In [16]: frames = range(len(ae_debussy))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)
```

```
In [17]: # amplitude envelope is graphed in red
```

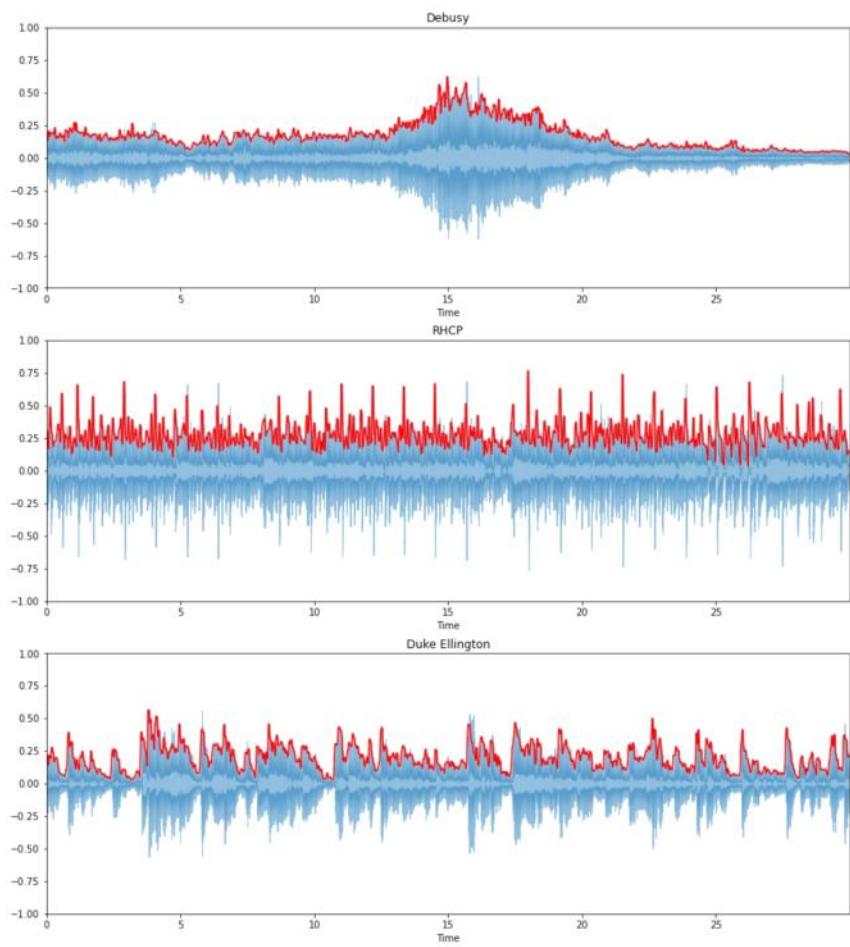
```
plt.figure(figsize=(15, 17))

ax = plt.subplot(3, 1, 1)
librosa.display.waveshow(debussy, alpha=0.5)
plt.plot(t, ae_debussy, color="r")
plt.ylim((-1, 1))
plt.title("Debussy")

plt.subplot(3, 1, 2)
librosa.display.waveshow(redhot, alpha=0.5)
plt.plot(t, ae_redhot, color="r")
plt.ylim((-1, 1))
plt.title("RHCP")

plt.subplot(3, 1, 3)
librosa.display.waveshow(duke, alpha=0.5)
plt.plot(t, ae_duke, color="r")
plt.ylim((-1, 1))
plt.title("Duke Ellington")

plt.show()
```



In [ ]:

## 09 RMS Energy and Zero-Crossing Rate\_ipynb

04 January 2025 15:46



RMS Energy and Zero-Crossing Rate\_ipynb

```
In [1]: import matplotlib.pyplot as plt
import numpy as np
import librosa
import librosa.display
import IPython.display as ipd
```

## Loading Audio Files

```
In [2]: debussy_file = "audio/debussy.wav"
redhot_file = "audio/redhot.wav"
duke_file = "audio/duke.wav"
```

```
In [3]: ipd.Audio(debussy_file)
```

```
Out[3]:
```



```
In [4]: ipd.Audio(redhot_file)
```

```
Out[4]:
```



```
In [5]: ipd.Audio(duke_file)
```

```
Out[5]:
```



```
In [6]: # Load audio files with librosa
debussy, sr = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)
```

## Root-mean-squared energy with Librosa

```
In [7]: FRAME_SIZE = 1024
HOP_LENGTH = 512
```

```
In [10]: rms_debussy = librosa.feature.rms(y=debussy, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
rms_redhot = librosa.feature.rms(y=redhot, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
rms_duke = librosa.feature.rms(y=duke, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
```

## Visualise RMSE + waveform

```
In [11]: frames = range(len(rms_debussy))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)
```

```
In [13]: # rms energy is graphed in red
```

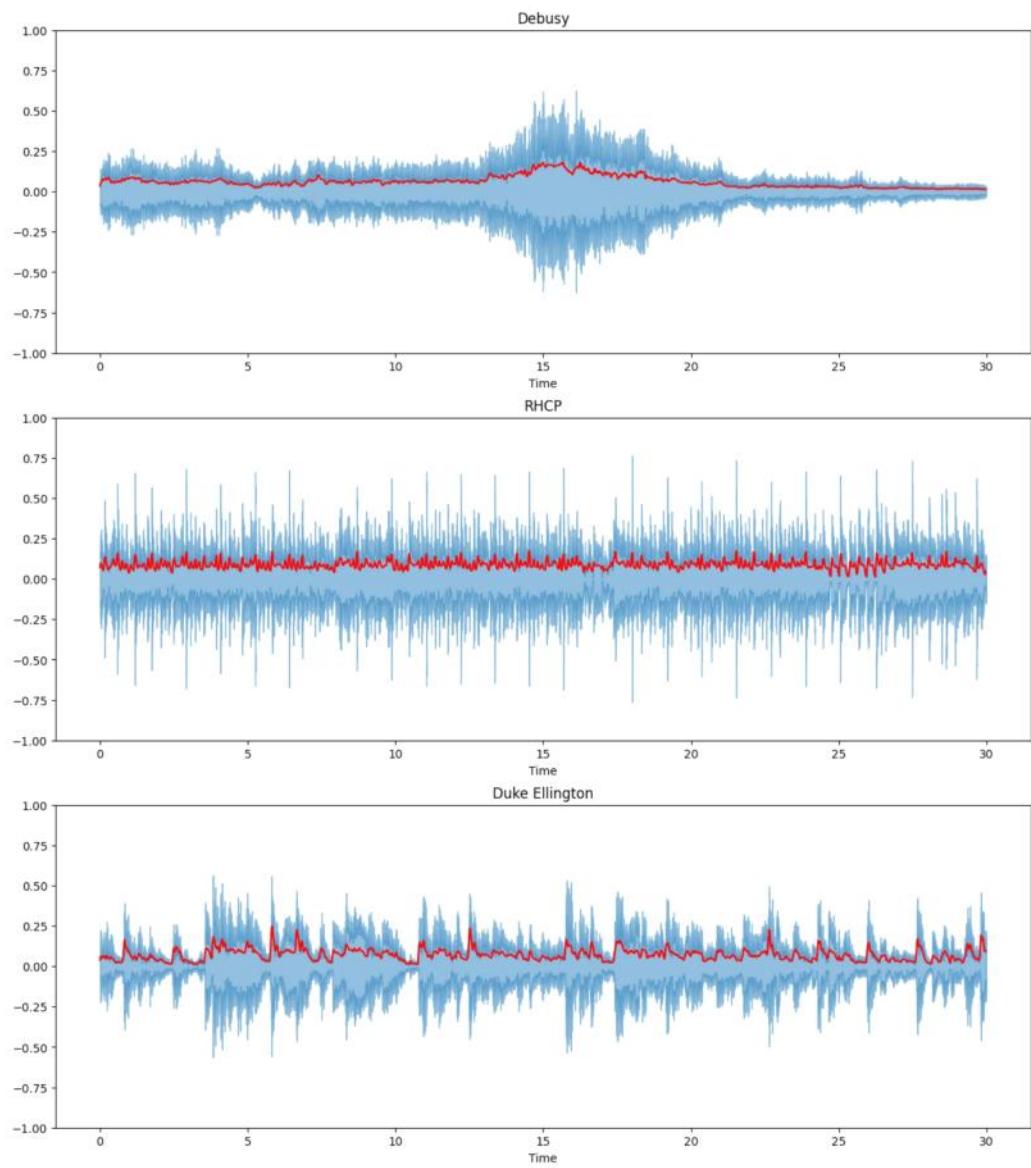
```
plt.figure(figsize=(15, 17))

ax = plt.subplot(3, 1, 1)
librosa.display.waveform(debussy, alpha=0.5)
plt.plot(t, rms_debussy, color="r")
plt.ylim((-1, 1))
plt.title("Debussy")

plt.subplot(3, 1, 2)
librosa.display.waveform(redhot, alpha=0.5)
plt.plot(t, rms_redhot, color="r")
plt.ylim((-1, 1))
plt.title("RHC P")

plt.subplot(3, 1, 3)
librosa.display.waveform(duke, alpha=0.5)
plt.plot(t, rms_duke, color="r")
plt.ylim((-1, 1))
plt.title("Duke Ellington")

plt.show()
```



### RMSE from scratch

```
In [14]: def rmse(signal, frame_size, hop_length):
    rmse = []

    # calculate rmse for each frame
    for i in range(0, len(signal), hop_length):
        rmse_current_frame = np.sqrt(sum(signal[i:i+frame_size]**2) / frame_size)
        rmse.append(rmse_current_frame)
    return np.array(rmse)
```

```
In [15]: rms_debussy1 = rmse(debussy, FRAME_SIZE, HOP_LENGTH)
rms_redhot1 = rmse(redhot, FRAME_SIZE, HOP_LENGTH)
rms_duke1 = rmse(duke, FRAME_SIZE, HOP_LENGTH)
```

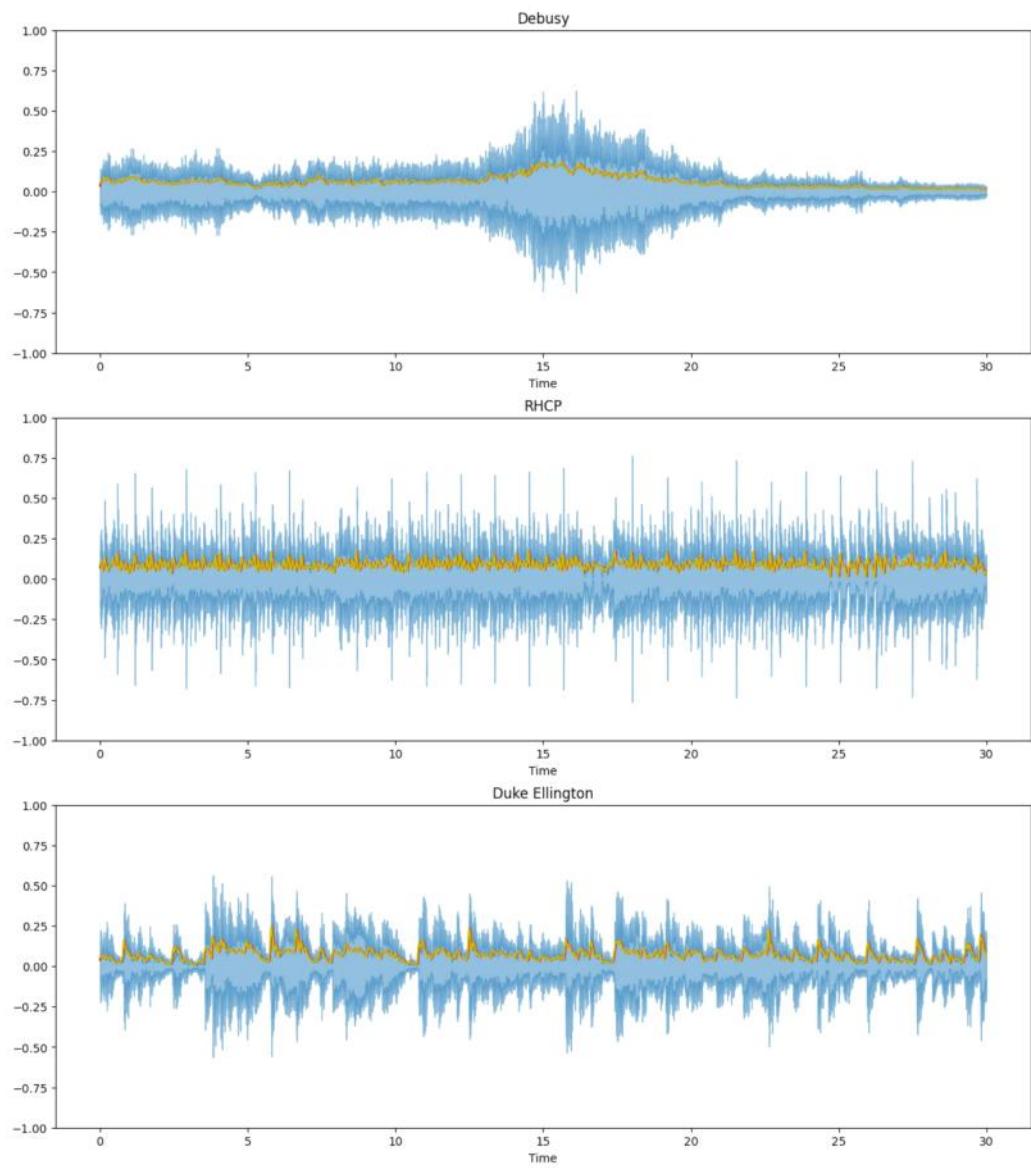
```
In [17]: plt.figure(figsize=(15, 17))

ax = plt.subplot(3, 1, 1)
librosa.display.waveplot(debussy, alpha=0.5)
plt.plot(t, rms_debussy, color="r")
plt.plot(t, rms_debussy1, color="y")
plt.ylim((-1, 1))
plt.title("Debussy")

plt.subplot(3, 1, 2)
librosa.display.waveplot(redhot, alpha=0.5)
plt.plot(t, rms_redhot, color="r")
plt.plot(t, rms_redhot1, color="y")
plt.ylim((-1, 1))
plt.title("RHCP")

plt.subplot(3, 1, 3)
librosa.display.waveplot(duke, alpha=0.5)
plt.plot(t, rms_duke, color="r")
plt.plot(t, rms_duke1, color="y")
plt.ylim((-1, 1))
plt.title("Duke Ellington")

plt.show()
```



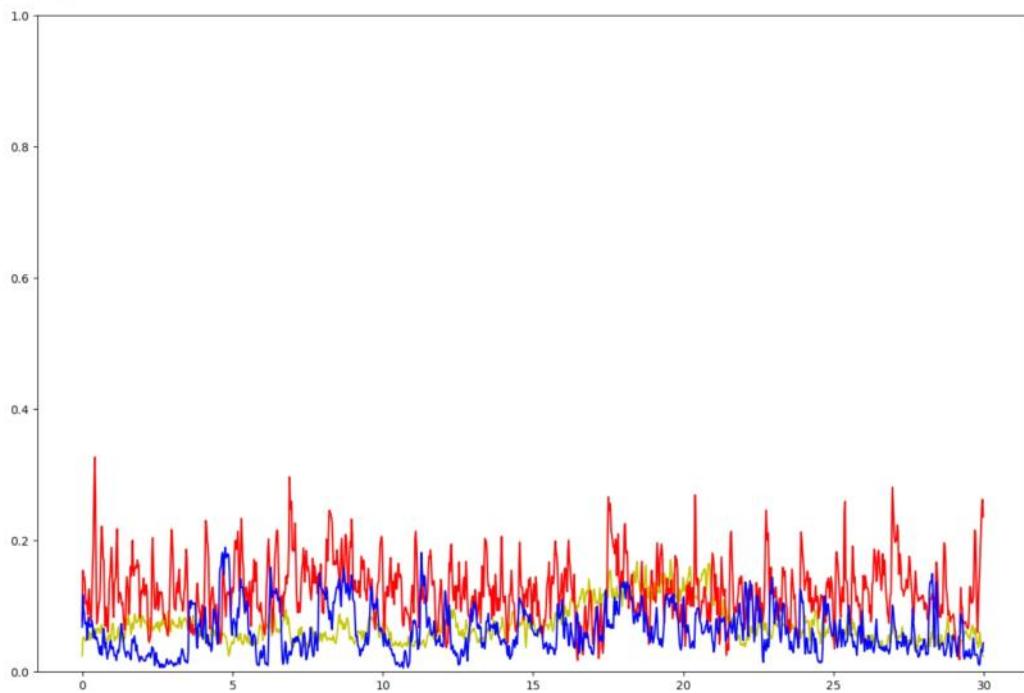
### Zero-crossing rate with Librosa

```
In [18]: zcr_debussy = librosa.feature.zero_crossing_rate(debussy, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
zcr_redhot = librosa.feature.zero_crossing_rate(redhot, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
zcr_duke = librosa.feature.zero_crossing_rate(duke, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]

In [19]: zcr_debussy.size
Out[19]: 1292
```

Visualise zero-crossing rate with Librosa

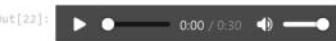
```
In [20]: plt.figure(figsize=(15, 10))
plt.plot(t, zcr_debussy, color="y")
plt.plot(t, zcr_redhot, color="r")
plt.plot(t, zcr_duke, color="b")
plt.ylim(0, 1)
plt.show()
```



ZCR: Voice vs Noise

```
In [21]: voice_file = "audio/voice.wav"
noise_file = "audio/noise.wav"
```

```
In [22]: ipd.Audio(voice_file)
```



```
In [23]: ipd.Audio(noise_file)
```



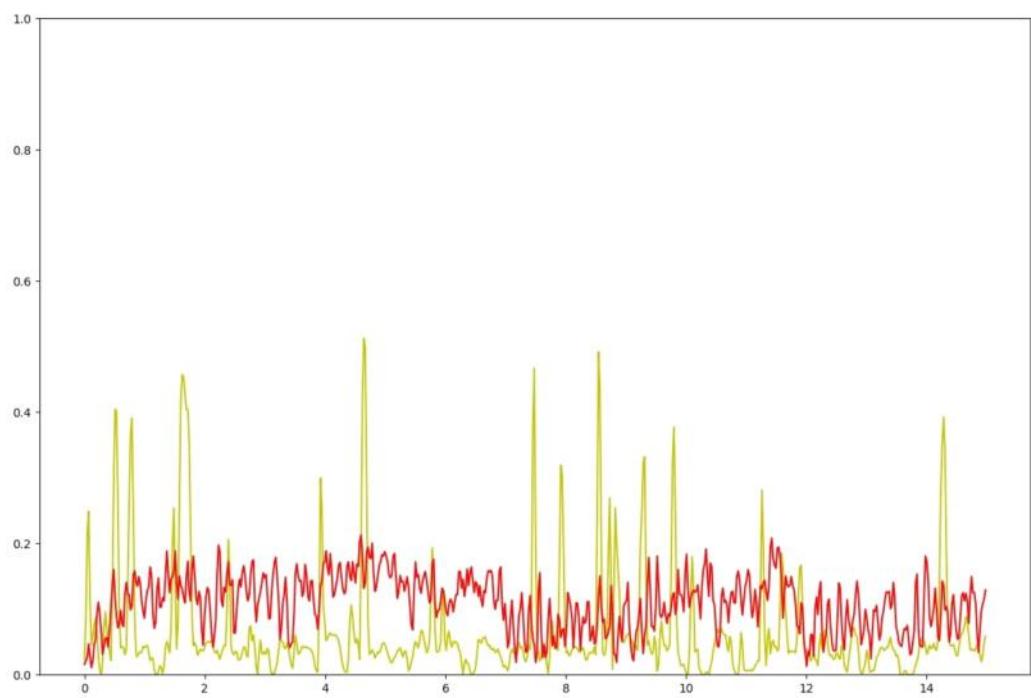
```
In [24]: # load audio files
voice, _ = librosa.load(voice_file, duration=15)
noise, _ = librosa.load(noise_file, duration=15)
```

```
In [25]: # get ZCR
zcr_voice = librosa.feature.zero_crossing_rate(voice, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
zcr_noise = librosa.feature.zero_crossing_rate(noise, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
```

```
In [26]: frames = range(len(zcr_voice))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)
```

```
In [27]: plt.figure(figsize=(15, 10))
```

```
plt.plot(t, zcr_voice, color="y")
plt.plot(t, zcr_noise, color="r")
plt.ylim(0, 1)
plt.show()
```



# Demystifying the Fourier Transform: The Intuition

Valerio Velardo

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)



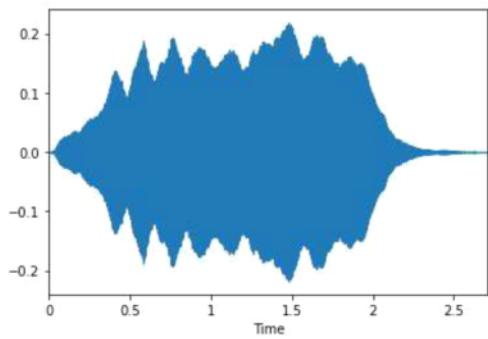
## Intuition

---

- Decompose a complex sound into its frequency components

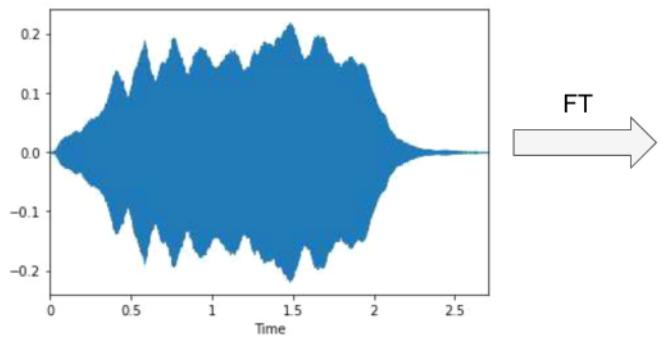
## From time to frequency domain

---



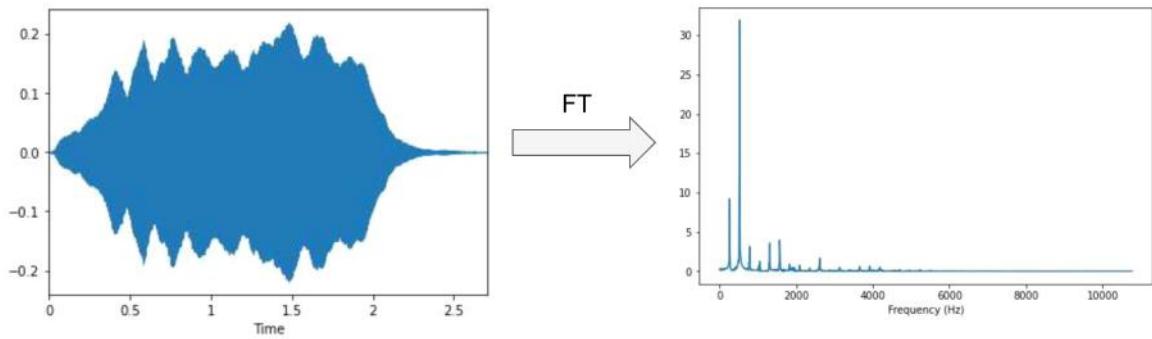
## From time to frequency domain

---



## From time to frequency domain

---



## Deeper intuition

---

- Compare signal with sinusoids of various frequencies

## Deeper intuition

---

- Compare signal with sinusoids of various frequencies
- For each frequency we get a magnitude and a phase

## Deeper intuition

---

- Compare signal with sinusoids of various frequencies
- For each frequency we get a magnitude and a phase
- **High magnitude indicates high similarity between the signal and a sinusoid**

**Sine wave**

---

$$\sin(2\pi \cdot (ft - \varphi))$$

## Deeper intuition

---

- Compare signal with sinusoids of various frequencies
- For each frequency we get a magnitude and a phase
- High magnitude indicates high similarity between the signal and a sinusoid

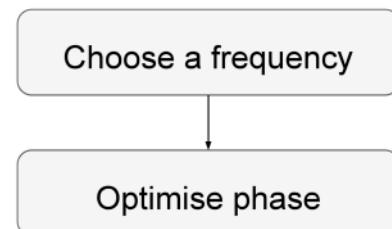
## Fourier transform: Step by step

---

Choose a frequency

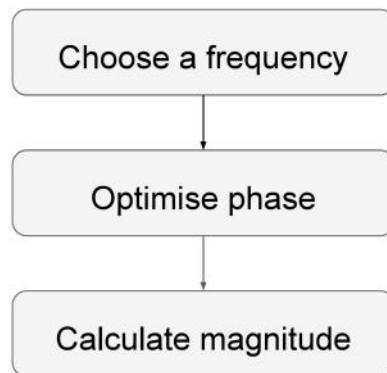
## Fourier transform: Step by step

---



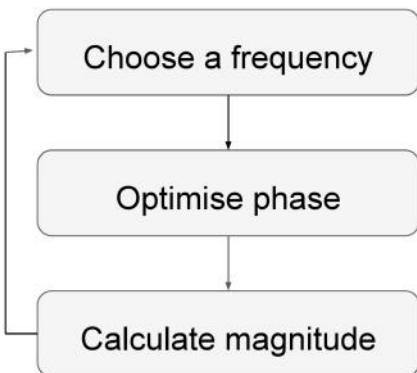
## Fourier transform: Step by step

---



## Fourier transform: Step by step

---



## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1]} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1]} \left( \int [s(t) \cdot \sin(2\pi \cdot (ft - \varphi))] \cdot dt \right)$$

Multiply signal and sinusoid

## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1]} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

[Calculate area](#)

## Fourier transform

---

$$\varphi_f = \underset{\varphi \in [0,1]}{\operatorname{argmax}} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

Select phase in  $[0, 1]$  that  
maximises the area

## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

Select max area

## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1]} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

t ∈ ℝ

$$d_f = \max_{\varphi \in [0,1]} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

## Fourier transform

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (\boxed{f}t - \varphi)) \cdot dt \right)$$

$$f \in \mathbb{R}$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (\boxed{f}t - \varphi)) \cdot dt \right)$$



## Reconstructing a signal

---

- Superimpose sinusoids

## Reconstructing a signal

---

- Superimpose sinusoids
- Weight them by the relative magnitude

## Reconstructing a signal

---

- Superimpose sinusoids
- Weight them by the relative magnitude
- Use relative phase

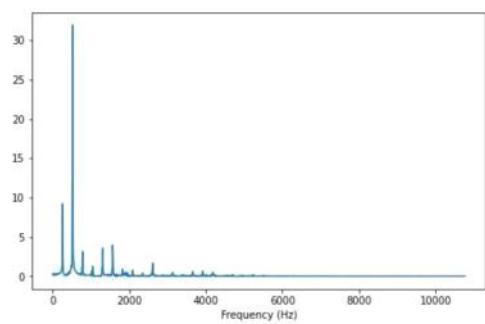
## Reconstructing a signal

---

- Superimpose sinusoids
- Weight them by the relative magnitude
- Use relative phase
- Original signal and FT have same information

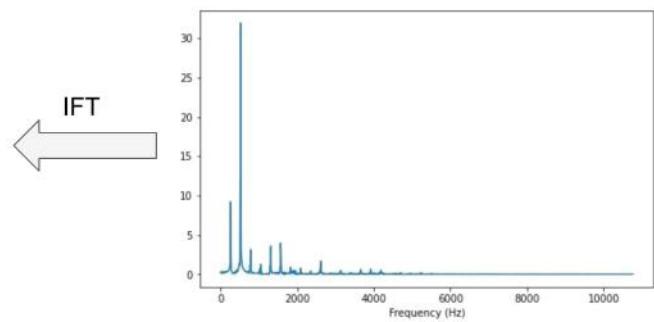
## Inverse Fourier transform

---



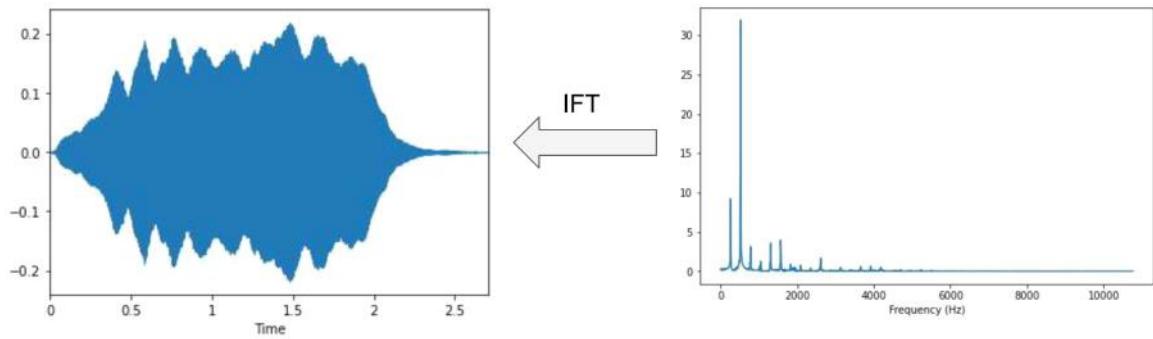
## Inverse Fourier transform

---



## Inverse Fourier transform

---



# Additive synthesis



## What's up next?

---

- Complex numbers

# 10 Code

04 January 2025 15:46



Fourier Transform\_ipynb

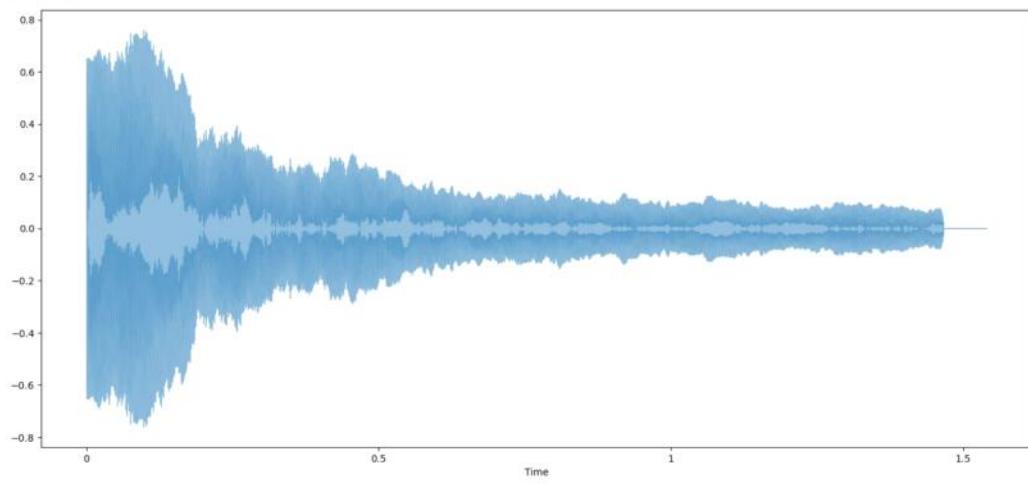
```
In [5]: import librosa
import librosa.display
import scipy as sp
import IPython.display as ipd
import matplotlib.pyplot as plt
import numpy as np
```

```
In [6]: # Load audio file in the player
audio_path = "audio/piano_c.wav"
ipd.Audio(audio_path)
```

```
Out[6]:
```

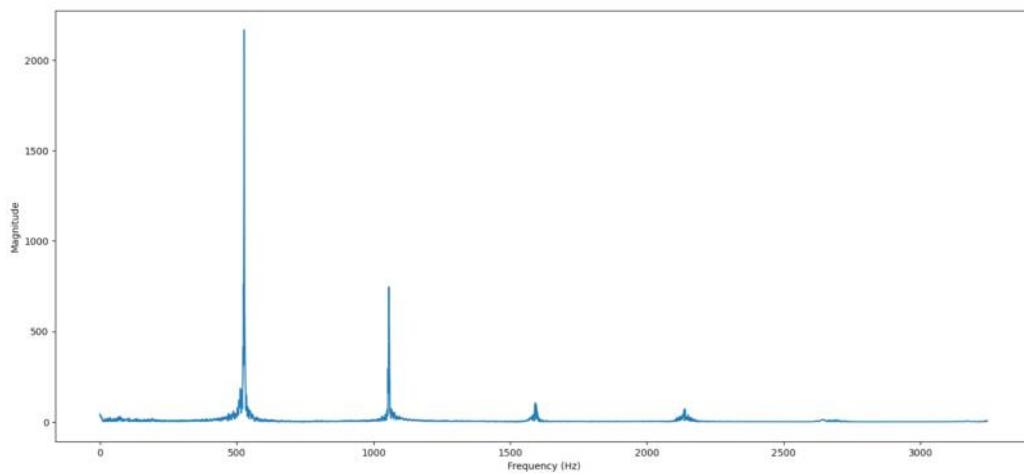
```
In [7]: # Load audio file
signal, sr = librosa.load(audio_path)
```

```
In [8]: # plot waveform
plt.figure(figsize=(18, 8))
librosa.display.waveplot(signal, sr=sr, alpha=0.5)
plt.show()
```



```
In [9]: # derive spectrum using FT
ft = sp.fft.fft(signal)
magnitude = np.absolute(ft)
frequency = np.linspace(0, sr, len(magnitude))
```

```
In [10]: # plot spectrum
plt.figure(figsize=(18, 8))
plt.plot(frequency[:5000], magnitude[:5000]) # magnitude spectrum
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.show()
```



```
In [11]: len(signal)
Out[11]: 33968

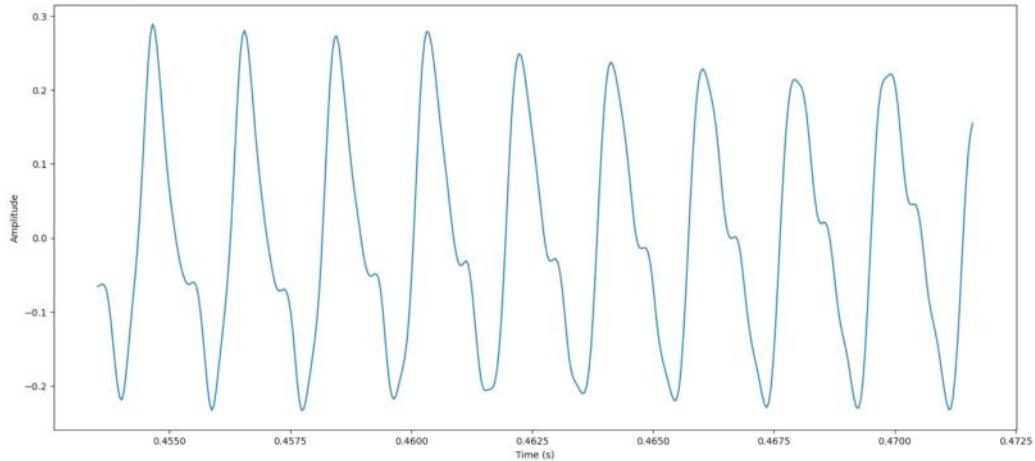
In [12]: d = 1 / sr
d
Out[12]: 4.5351473922902495e-05

In [13]: d_523 = 1 / 523
d_523
Out[13]: 0.0019120458891013384

In [14]: d_400_samples = 400 * d
d_400_samples
Out[14]: 0.018140589569160998
```

```
In [15]: # zoom in to the waveform
samples = range(len(signal))
t = librosa.samples_to_time(samples, sr=sr)

plt.figure(figsize=(18, 8))
plt.plot(t[10000:10400], signal[10000:10400])
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```

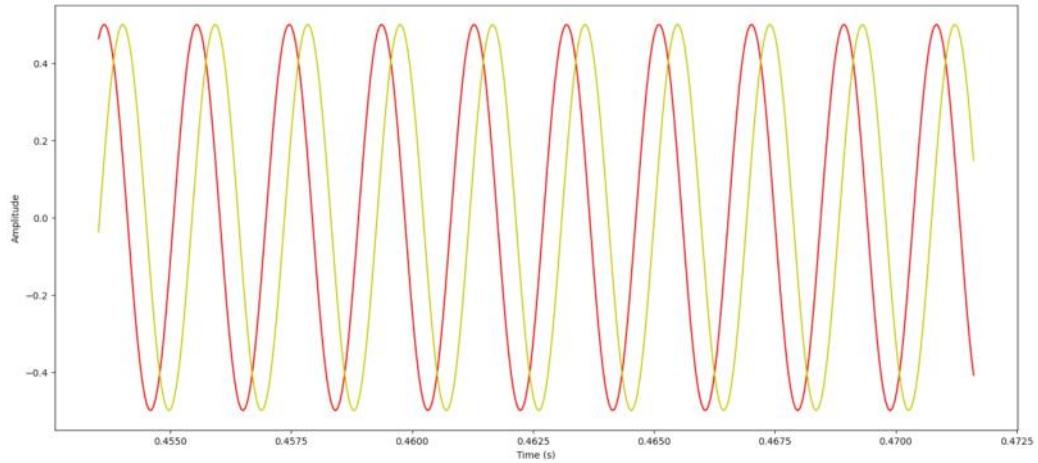


```
In [16]: # create a sinusoid
f = 523
phase = 0
phase2 = 0.2

sin = 0.5 * np.sin(2*np.pi * (f * t - phase))
sin2 = 0.5 * np.sin(2*np.pi * (f * t - phase2))

plt.figure(figsize=(18, 8))
plt.plot(t[10000:10400], sin[10000:10400], color="r")
plt.plot(t[10000:10400], sin2[10000:10400], color="y")

plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```



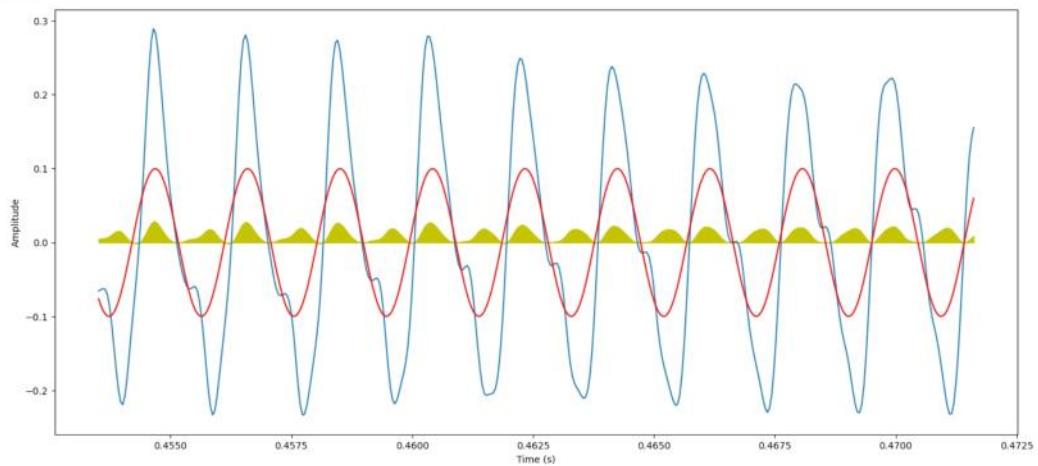
```
In [17]: # compare signal and sinusoids
f = 523
phase = 0.55

sin = 0.1 * np.sin(2*np.pi * (f * t - phase))

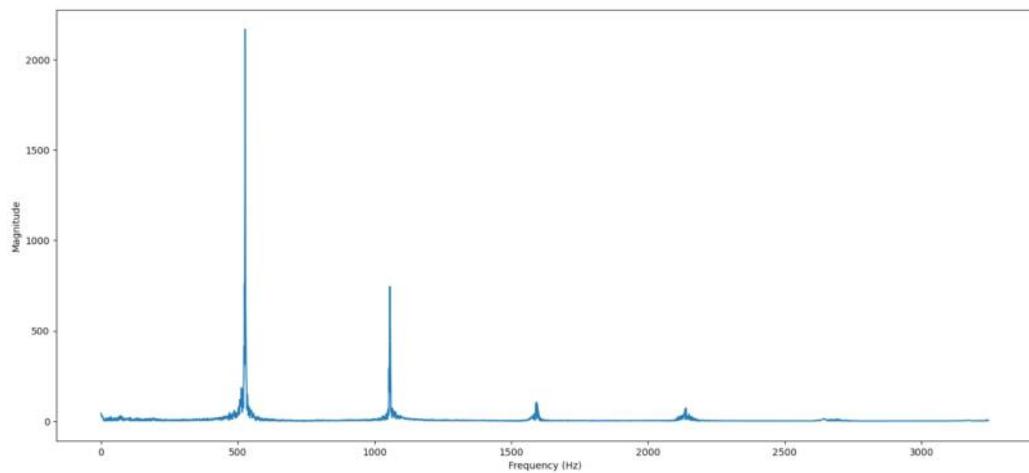
plt.figure(figsize=(18, 8))
plt.plot(t[10000:10400], signal[10000:10400])
plt.plot(t[10000:10400], sin[10000:10400], color="r")

plt.fill_between(t[10000:10400], sin[10000:10400]*signal[10000:10400], color="y")

plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```



```
In [18]: # plot spectrum
plt.figure(figsize=(18, 8))
plt.plot(frequency[:5000], magnitude[:5000]) # magnitude spectrum
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.show()
```



```
In [19]: # superimposing pure tones
f = 1
t = np.linspace(0, 10, 10000)

sin = np.sin(2*np.pi * (f * t))
sin2 = np.sin(2*np.pi * (2*f * t))
sin3 = np.sin(2*np.pi * (3*f * t))

sum_signal = sin + sin2 + sin3

plt.figure(figsize=(15, 10))

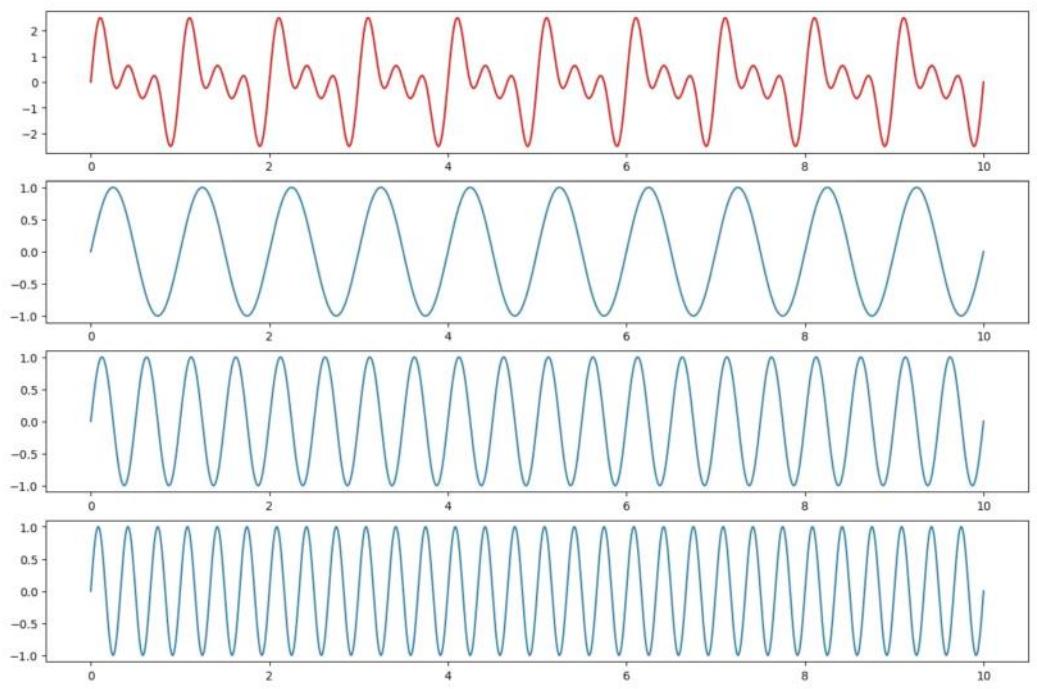
plt.subplot(4, 1, 1)
plt.plot(t, sum_signal, color="r")

plt.subplot(4, 1, 2)
plt.plot(t, sin)

plt.subplot(4, 1, 3)
plt.plot(t, sin2)

plt.subplot(4, 1, 4)
plt.plot(t, sin3)

plt.show()
```



# Complex Numbers for Audio Signal Processing

Valerio Velardo

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

**Why bother with complex numbers?**

---

## Why bother with complex numbers?

---

- Fourier transform -> magnitude and phase

## Why bother with complex numbers?

---

- Fourier transform -> magnitude and phase
- Magnitude is a real number

## Why bother with complex numbers?

---

- Fourier transform -> magnitude and phase
- Magnitude is a real number
- ... something with magnitude + phase?

**COMPLICATED  
NUMBERS?**



**NO SIRaj! IT'S  
COMPLEX  
NUMBERS**



## The genesis of CNs

---

## The genesis of CNs

---



$\text{sqrt}(-1)$

## The genesis of CNs

---



$\text{sqrt}(-1)$

$i^2 = -1$

Our first complex number

$$c = a + ib$$

$$a, b \in \mathbb{R}$$

## Our first complex number

---

Real part

$$c = \boxed{a} + ib$$
$$a, b \in \mathbb{R}$$

## Our first complex number

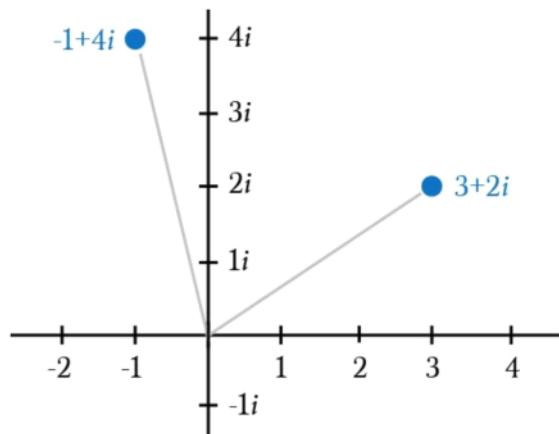
---

$$c = \boxed{a} + \boxed{ib}$$

$$a, b \in \mathbb{R}$$

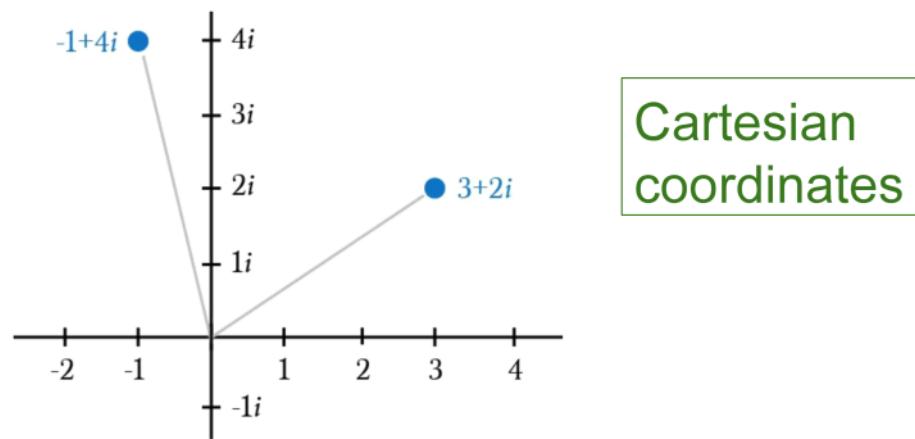
## Plotting complex numbers

---



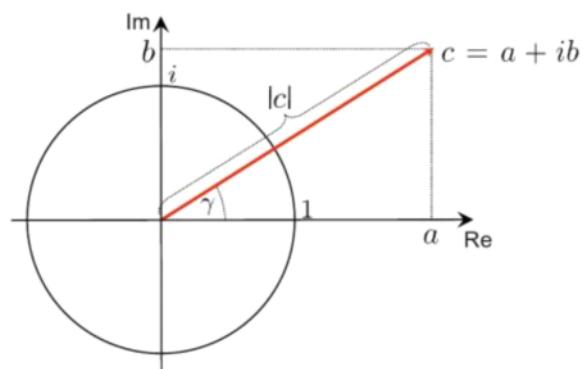
## Plotting complex numbers

---



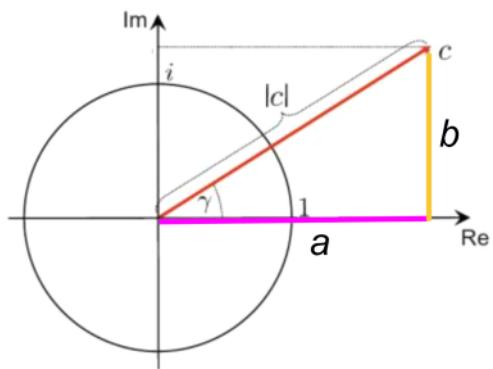
## Polar coordinate representation

---



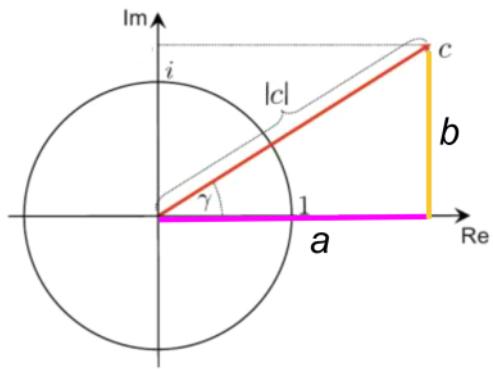
## Polar coordinate representation

---



## Polar coordinate representation

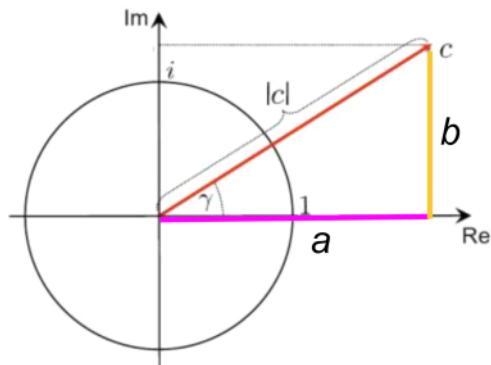
---



$$|c|^2 = a^2 + b^2$$

## Polar coordinate representation

---



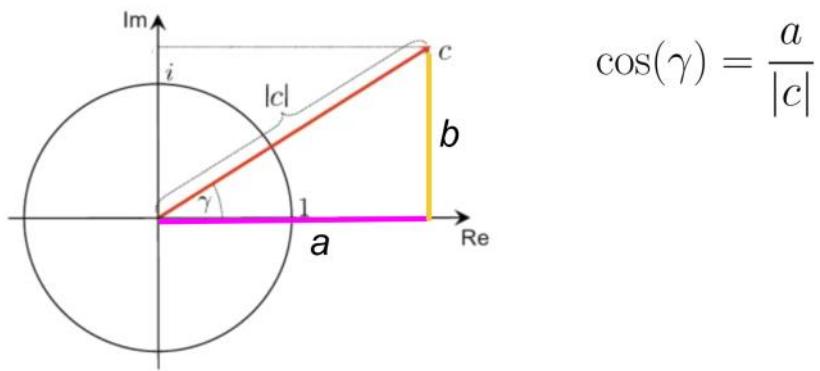
$$|c|^2 = a^2 + b^2$$

↓

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

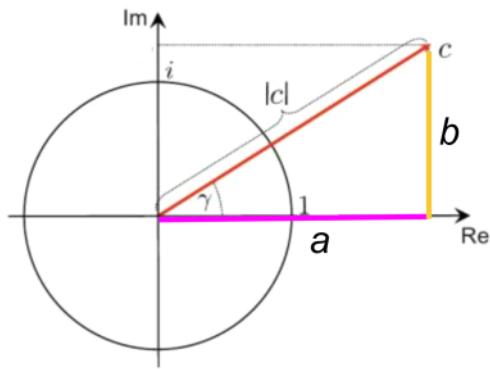
## Polar coordinate representation

---



## Polar coordinate representation

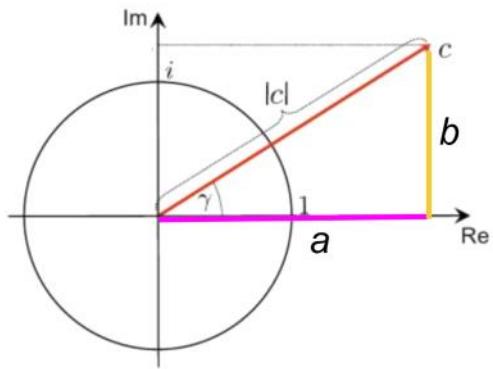
---



$$\cos(\gamma) = \frac{a}{|c|} \quad \sin(\gamma) = \frac{b}{|c|}$$

## Polar coordinate representation

---

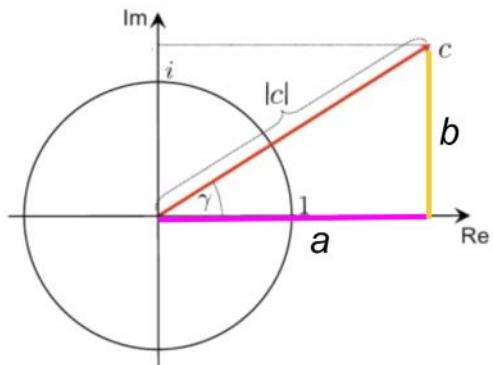


$$\cos(\gamma) = \frac{a}{|c|} \quad \sin(\gamma) = \frac{b}{|c|}$$

$$\frac{\sin(\gamma)}{\cos(\gamma)} = \frac{b}{a}$$

## Polar coordinate representation

---

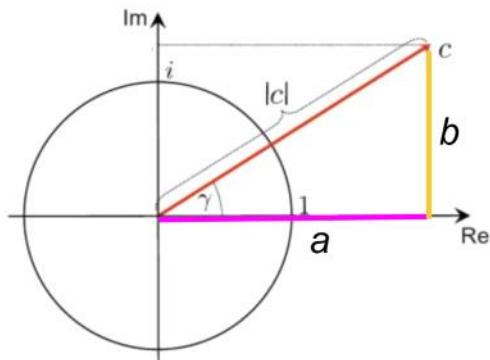


$$\cos(\gamma) = \frac{a}{|c|} \quad \sin(\gamma) = \frac{b}{|c|}$$

$$\tan(\gamma) \quad \boxed{\frac{\sin(\gamma)}{\cos(\gamma)}} = \frac{b}{a}$$

## Polar coordinate representation

---



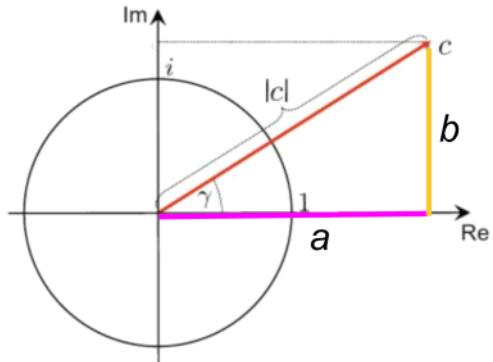
$$\cos(\gamma) = \frac{a}{|c|} \quad \sin(\gamma) = \frac{b}{|c|}$$

$$\frac{\sin(\gamma)}{\cos(\gamma)} = \frac{b}{a}$$

$$\gamma = \arctan\left(\frac{b}{a}\right)$$

## Polar coordinate representation

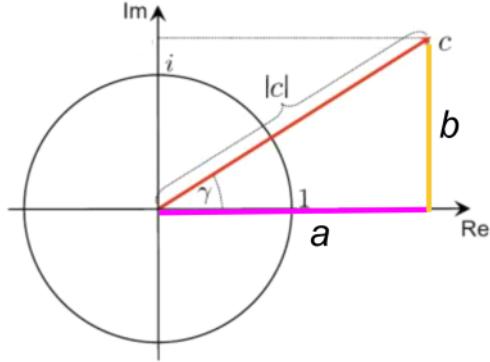
---



$$\gamma = \arctan\left(\frac{b}{a}\right)$$
$$|c| = \sqrt{a^2 + b^2}$$

## Polar coordinate representation

---

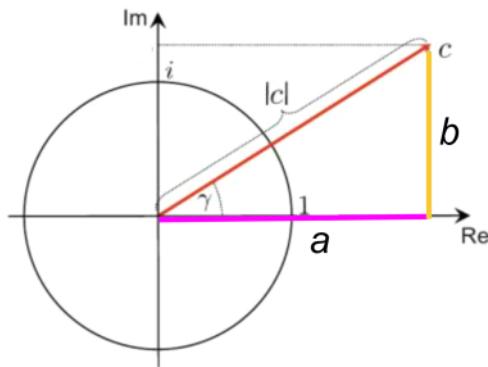


$$\gamma = \arctan\left(\frac{b}{a}\right)$$
$$|c| = \sqrt{a^2 + b^2}$$

$$a = |c| \cdot \cos(\gamma) \quad b = |c| \cdot \sin(\gamma)$$

## Polar coordinate representation

---



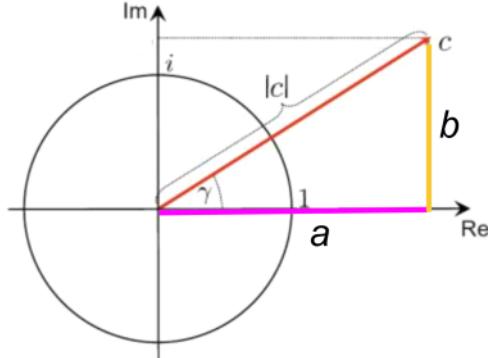
$$\gamma = \arctan\left(\frac{b}{a}\right)$$
$$|c| = \sqrt{a^2 + b^2}$$

$$a = |c| \cdot \cos(\gamma) \quad b = |c| \cdot \sin(\gamma)$$

$$c = a + ib$$

## Polar coordinate representation

---



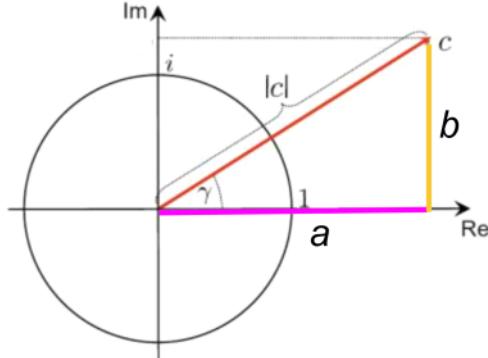
$$\gamma = \arctan\left(\frac{b}{a}\right)$$
$$|c| = \sqrt{a^2 + b^2}$$

$$a = |c| \cdot \cos(\gamma) \quad b = |c| \cdot \sin(\gamma)$$

$$c = \boxed{a} + ib$$

## Polar coordinate representation

---



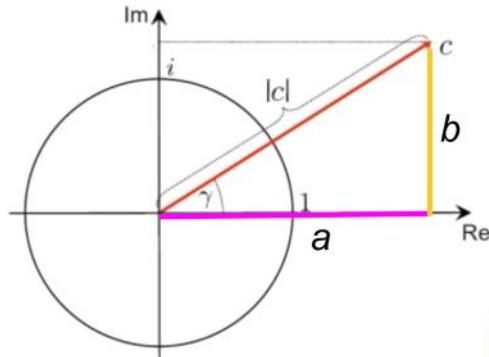
$$\gamma = \arctan\left(\frac{b}{a}\right)$$
$$|c| = \sqrt{a^2 + b^2}$$

$$a = |c| \cdot \cos(\gamma) \quad b = |c| \cdot \sin(\gamma)$$

$$c = a + ib$$

## Polar coordinate representation

---



$$\gamma = \arctan\left(\frac{b}{a}\right)$$
$$|c| = \sqrt{a^2 + b^2}$$

$$a = |c| \cdot \cos(\gamma) \quad b = |c| \cdot \sin(\gamma)$$

$$c = a + ib$$

$$c = |c| \cdot (\cos(\gamma) + i \sin(\gamma))$$

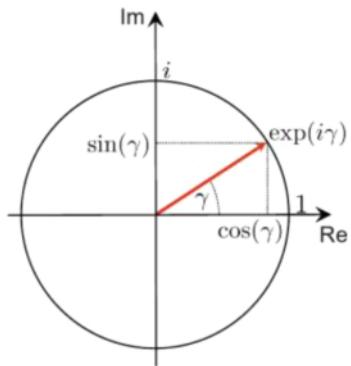
## Euler formula

---

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

## Euler formula

---



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

## Euler identity

---

$$e^{i\pi} + 1 = 0$$



## Euler identity

---

$$e^{i\pi} + 1 = 0$$

-1



## Euler identity

---

$$e^{i\pi} + 1 = 0$$

-1

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

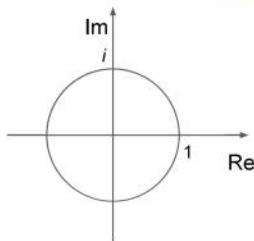


## Euler identity

---

$$e^{i\pi} + 1 = 0$$

-1

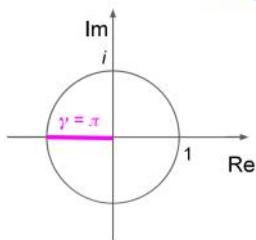


## Euler identity

---

$$e^{i\pi} + 1 = 0$$

-1



## Polar coordinates 2.0

---

$$c = |c| \cdot (\cos(\gamma) + i \sin(\gamma))$$

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

## Polar coordinates 2.0

---

$$c = |c| \cdot (\cos(\gamma) + i \sin(\gamma))$$

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

## Polar coordinates 2.0

---

$$c = |c| \cdot (\cos(\gamma) + i \sin(\gamma))$$

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

$$c = |c| \cdot e^{i\gamma}$$

## Polar coordinates interpretation

$$c = |c| \cdot e^{i\gamma}$$

## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

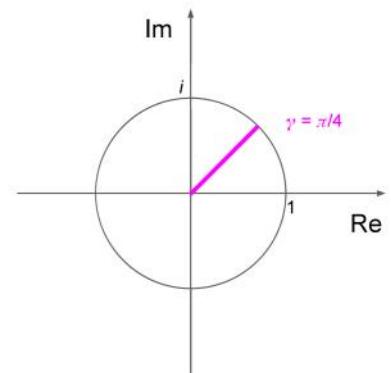
Direction of a number  
in the complex plane

## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Direction of a number  
in the complex plane

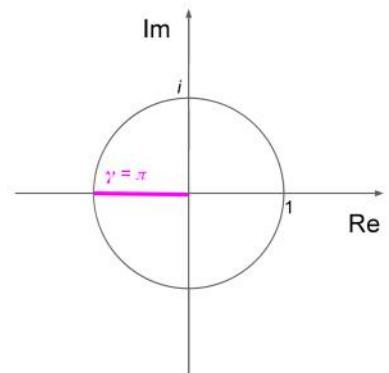


## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Direction of a number  
in the complex plane

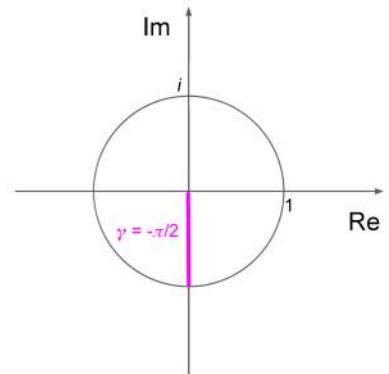


## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Direction of a number  
in the complex plane



## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Scales distance from  
origin

Direction of a number  
in the complex plane

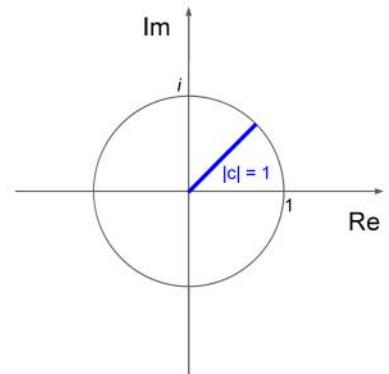
## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Scales distance from origin

Direction of a number in the complex plane



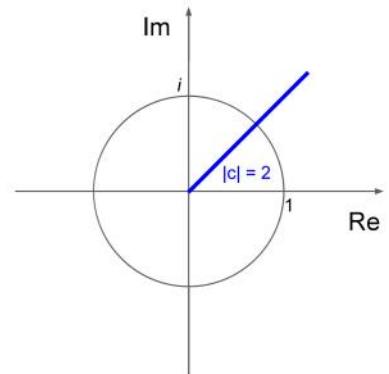
## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Scales distance from origin

Direction of a number in the complex plane



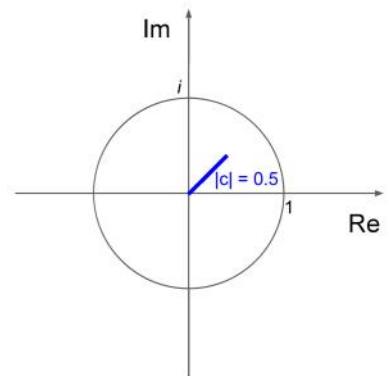
## Polar coordinates interpretation

---

$$c = |c| \cdot e^{i\gamma}$$

Scales distance from origin

Direction of a number in the complex plane



## What's up next?

---

- Complex representation of Fourier transform

# Defining the Fourier Transform Using Complex Numbers

Valerio Velardo

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

**Previously...**

---

## Previously...

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

## Previously...

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$c = |c| \cdot e^{i\gamma}$$

## The intuition

---

- Use magnitude and phase as polar coordinates

## The intuition

---

- Use magnitude and phase as polar coordinates
- Encode both coefficients in a single complex number

## **Complex Fourier transform coefficients**

---

## Complex Fourier transform coefficients

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$c = |c| \cdot e^{i\gamma}$$

## Complex Fourier transform coefficients

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$c = |c| \cdot e^{i\gamma}$$

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$

## Complex Fourier transform coefficients

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$c = \boxed{|c|} \cdot e^{i\gamma}$$

$$c_f = \boxed{\frac{d_f}{\sqrt{2}}} \cdot e^{-i2\pi\varphi_f}$$

## Complex Fourier transform coefficients

---

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

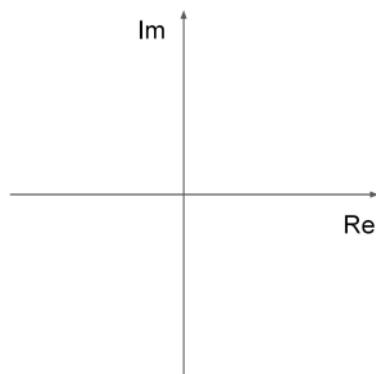
$$d_f = \max_{\varphi \in [0,1)} \left( \int s(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$c = |\boxed{c}| \cdot e^{i \boxed{\gamma}}$$

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i \boxed{2\pi \varphi_f}}$$

## Complex Fourier transform coefficients

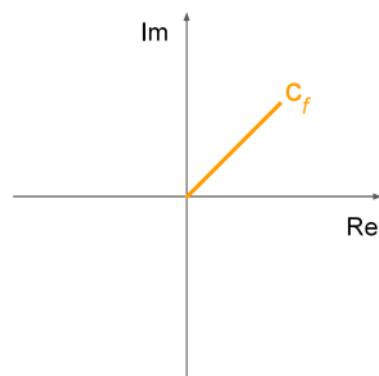
$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$



## Complex Fourier transform coefficients

---

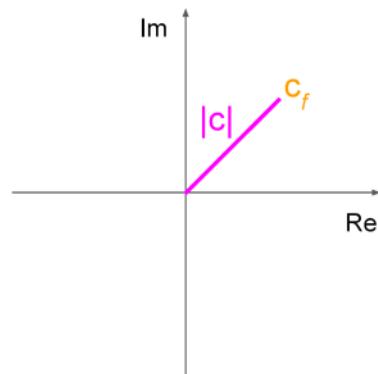
$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$



## Complex Fourier transform coefficients

---

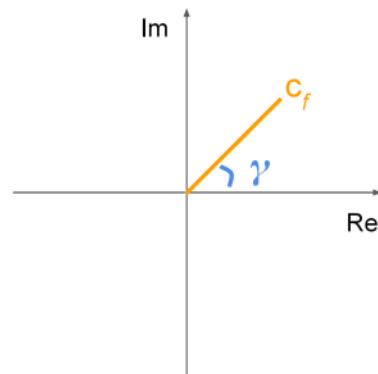
$$c_f = \boxed{\frac{d_f}{\sqrt{2}}} \cdot e^{-i2\pi\varphi_f}$$



## Complex Fourier transform coefficients

---

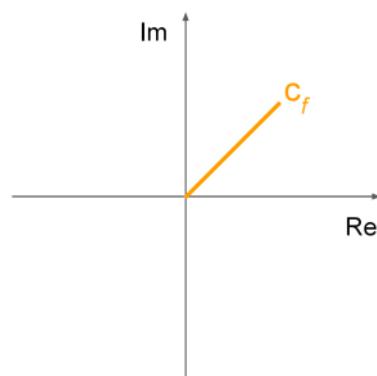
$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$



## Complex Fourier transform coefficients

---

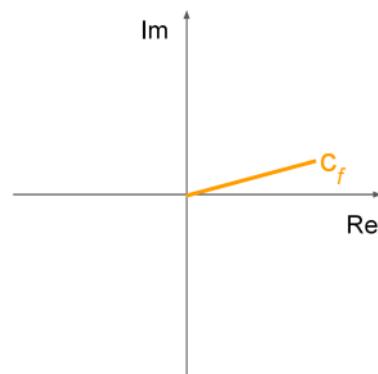
$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{\square i 2\pi \varphi_f}$$



## Complex Fourier transform coefficients

---

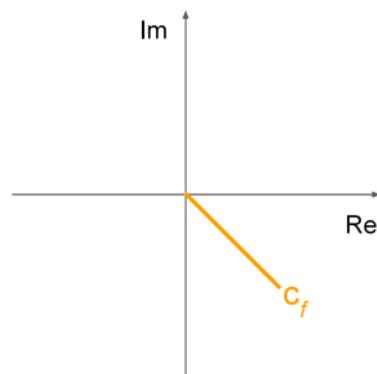
$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{\square i 2\pi \varphi_f}$$



## Complex Fourier transform coefficients

---

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{\square i 2\pi \varphi_f}$$



**Continuous audio signal**

---

$$g(t)$$

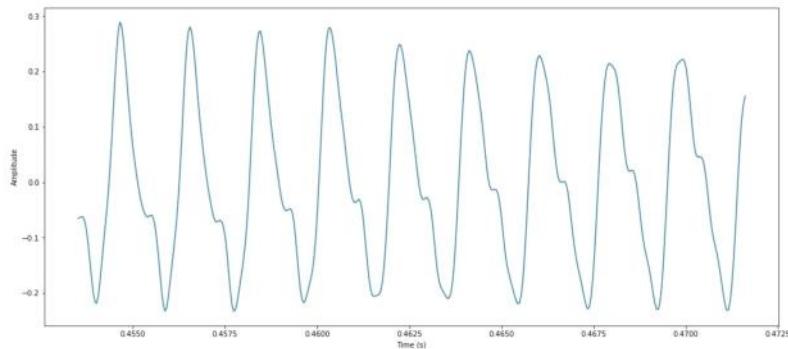
## Continuous audio signal

$$g(t) \quad g : \mathbb{R} \rightarrow \mathbb{R}$$

## Continuous audio signal

---

$$g(t) \quad g : \mathbb{R} \rightarrow \mathbb{R}$$



## Complex Fourier transform

$$\hat{g}(f) = c_f$$

## Complex Fourier transform

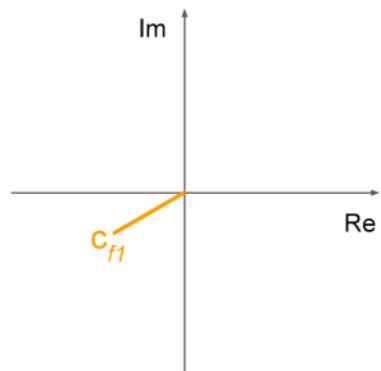
$$\hat{g}(f) = c_f$$

$$\hat{g} : \mathbb{R} \rightarrow \mathbb{C}$$

## Complex Fourier transform

$$\hat{g}(f) = c_f$$

$$\hat{g} : \mathbb{R} \rightarrow \mathbb{C}$$

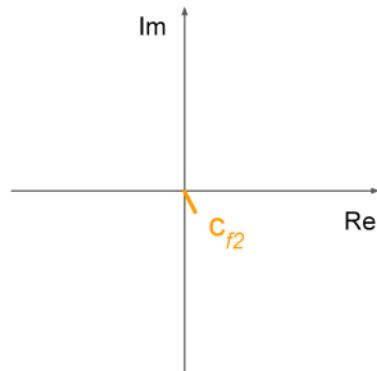


## Complex Fourier transform

---

$$\hat{g}(f) = c_f$$

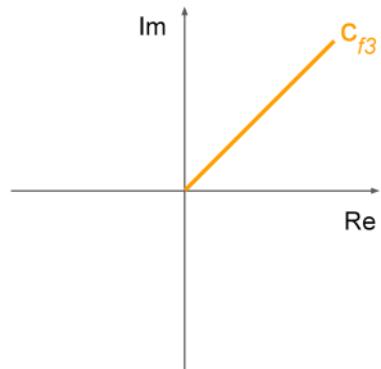
$$\hat{g} : \mathbb{R} \rightarrow \mathbb{C}$$



## Complex Fourier transform

$$\hat{g}(f) = c_f$$

$$\hat{g} : \mathbb{R} \rightarrow \mathbb{C}$$



## Complex Fourier transform

---

$$d_f = \max_{\varphi \in [0,1)} \left( \int g(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int g(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

## Complex Fourier transform

---

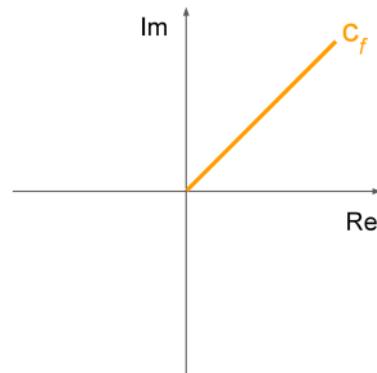
$$d_f = \max_{\varphi \in [0,1)} \left( \int g(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$
$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int g(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt$$

## Complex Fourier transform

$$d_f = \max_{\varphi \in [0,1)} \left( \int g(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$
$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left( \int g(t) \cdot \sin(2\pi \cdot (ft - \varphi)) \cdot dt \right)$$

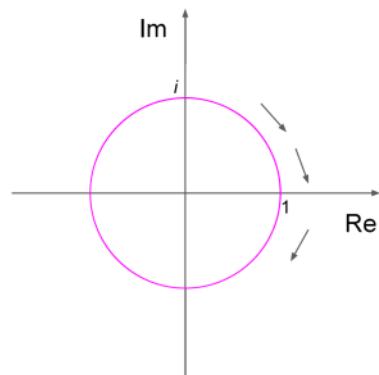
$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt$$



## Complex Fourier transform

---

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt$$



## Complex Fourier transform

---

$$\hat{g}(f) = \int [g(t)] \cdot e^{-i2\pi f t} dt$$

## Complex Fourier transform

---

$$\hat{g}(f) = \int [g(t) \cdot e^{-i2\pi ft}] dt$$

## Complex Fourier transform

---

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi f t} dt$$

## Complex Fourier transform

---

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

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt$$

## Complex Fourier transform

---

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

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt = \boxed{\int g(t) \cdot \cos(-2\pi ft) dt + i \int g(t) \cdot \sin(-2\pi ft) dt}$$

## Complex Fourier transform

---

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt = \boxed{\int g(t) \cdot \cos(-2\pi ft) dt} + i \boxed{\int g(t) \cdot \sin(-2\pi ft) dt}$$

Real part

Imaginary part

## Magnitude Fourier transform

---

$$|\hat{g}(f)|$$

## Magnitude and phase

---

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$

## Magnitude and phase

---

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$

$$d_f = \sqrt{2} \cdot |\hat{g}(f)|$$

## Magnitude and phase

---

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$

$$d_f = \sqrt{2} \cdot |\hat{g}(f)|$$

## Magnitude and phase

---

$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i2\pi\varphi_f}$$

$$d_f = \sqrt{2} \cdot |\hat{g}(f)|$$

$$\varphi_f = -\frac{\gamma_f}{2\pi}$$

## Magnitude and phase

---

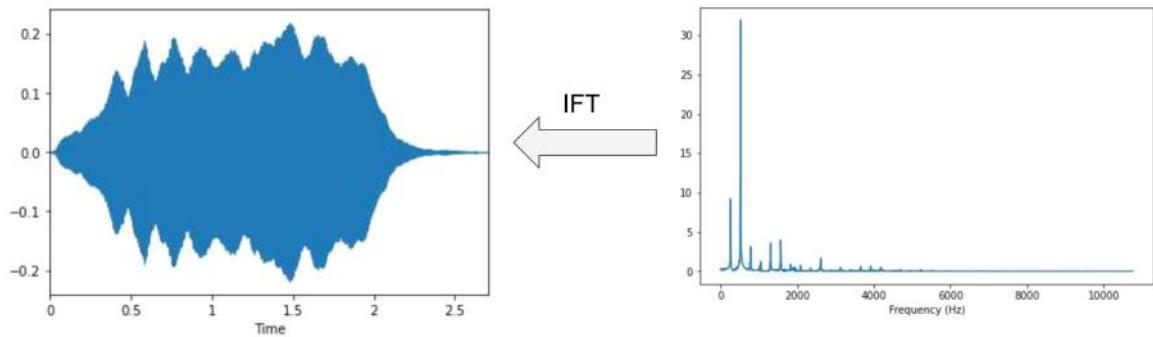
$$c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i \boxed{2\pi\varphi_f}}$$

$$d_f = \sqrt{2} \cdot |\hat{g}(f)|$$

$$\varphi_f = -\frac{\boxed{\gamma_f}}{2\pi}$$

## Inverse Fourier transform

---



## Fourier representation

$$g(t) = \int c_f \cdot e^{i2\pi ft} df$$

## Fourier representation

Pure tone of frequency  $f$

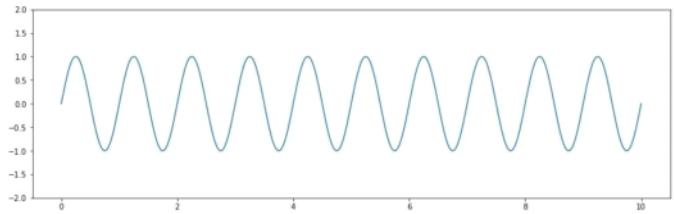
$$g(t) = \int c_f \cdot e^{i2\pi ft} df$$

## Fourier representation

---

Pure tone of frequency  $f$

$$g(t) = \int c_f \cdot e^{i2\pi ft} df$$

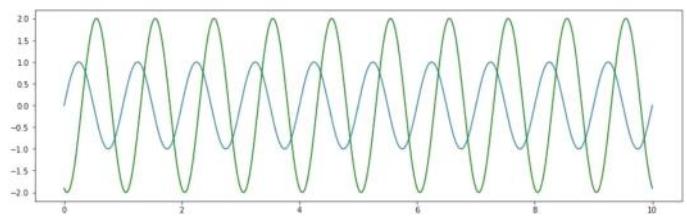


## Fourier representation

---

Weight pure tone with magnitude and add phase

$$g(t) = \int [c_f \cdot e^{i2\pi ft}] df$$

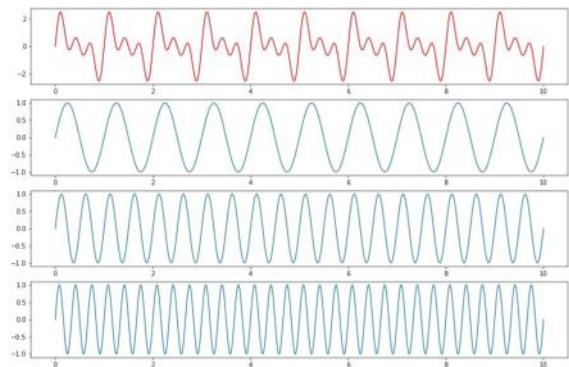


## Fourier representation

---

Add up all (weighted) sinusoids

$$g(t) = \int c_f \cdot e^{i2\pi ft} df$$



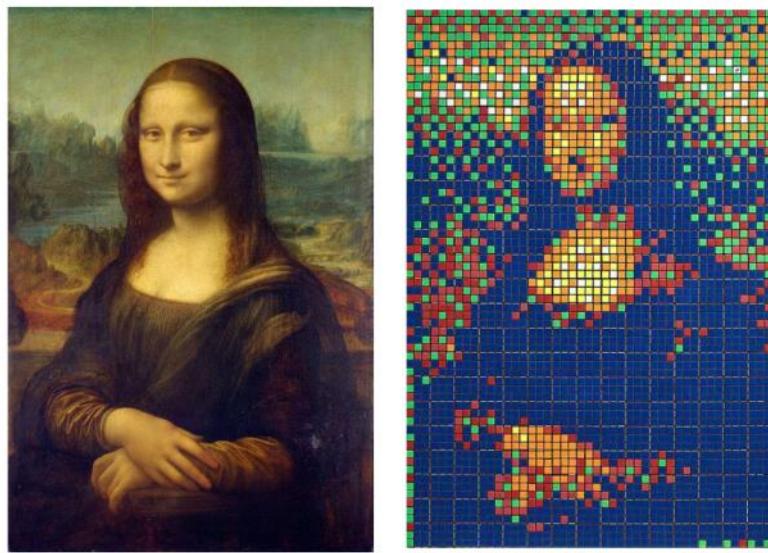
## A Fourier roundtrip

---

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt$$

$$g(t) = \int c_f \cdot e^{i2\pi ft} df$$





## 12 code\_Defining the Fourier Transform Using Complex Numbers

04 January 2025 23:00



code\_Defining the Fourier Transform Using Complex Nu...

```

In [1]: import cmath
import matplotlib.pyplot as plt
import numpy as np

In [2]: def create_signal(frequency, time):
    sin = np.sin(2 * np.pi * (frequency * time))
    sin2 = np.sin(2 * np.pi * (2 * frequency * time))
    sin3 = np.sin(2 * np.pi * (3 * frequency * time))

    return sin + sin2 + sin3

In [3]: def calculate_centre_of_gravity(mult_signal):
    x_centre = np.mean([x.real for x in mult_signal])
    y_centre = np.mean([x.imag for x in mult_signal])
    return x_centre, y_centre

In [4]: def calculate_sum(mult_signal):
    x_sum = np.sum([x.real for x in mult_signal])
    y_sum = np.sum([x.imag for x in mult_signal])
    return x_sum, y_sum

In [5]: def create_pure_tone(frequency, time):
    angle = -2 * np.pi * frequency * time
    return np.cos(angle) + 1j * np.sin(angle)

In [16]: def plot_fourier_transform(pure_tone_frequency,
                                 signal_frequency,
                                 time,
                                 plot_centre_of_gravity=False,
                                 plot_sum=False):

    # create sinusoid and signal
    pure_tone = create_pure_tone(pure_tone_frequency, time)
    signal = create_signal(signal_frequency, time)

    # multiply pure tone and signal
    mult_signal = pure_tone * signal

    X = [x.real for x in mult_signal]
    Y = [x.imag for x in mult_signal]

    plt.figure(figsize=(150, 100))
    plt.plot(X, Y, 'o')

    # calculate and plot centre of gravity
    if plot_centre_of_gravity:
        centre_of_gravity = calculate_centre_of_gravity(mult_signal)
        plt.plot([centre_of_gravity[0]], [centre_of_gravity[1]], marker='o', markerfacecolor='red')

    # calculate and plot sum
    if plot_sum:
        integral = calculate_sum(mult_signal)

```

```

plt.plot([integral[0]], [integral[1]], marker='o', markersize=10, color="green")

# set origin axes
ax = plt.gca()
ax.grid(True)
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')

if not plot_sum:
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)

plt.show()

```

In [17]:

```

def plot_signal(signal, time):
    plt.figure(figsize=(15, 10))
    plt.plot(signal, time)
    plt.xlabel("Time")
    plt.ylabel("Intensity")
    plt.show()

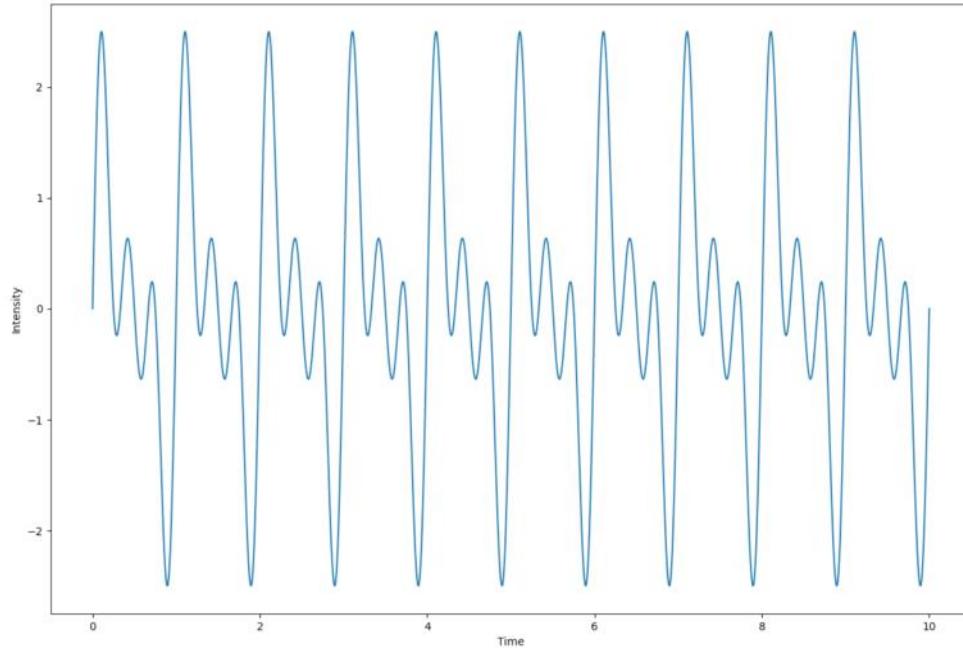
```

In [18]:

```

time = np.linspace(0, 10, 10000)
signal = create_signal(frequency=1, time=time)
plot_signal(time, signal)

```



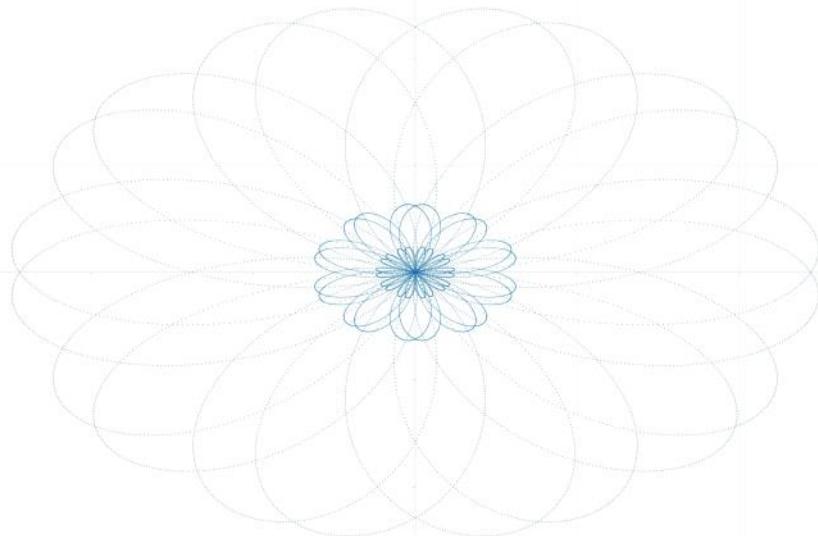
In [21]:

```

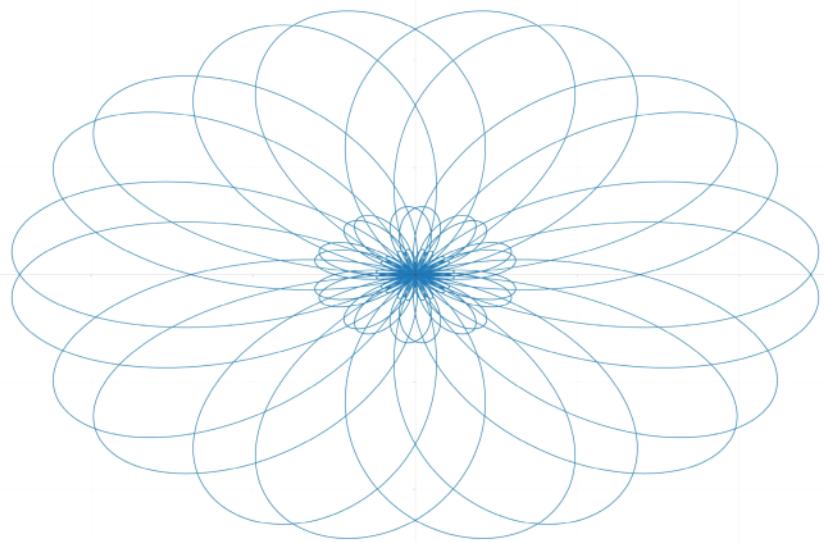
time = np.linspace(0, 10, 10000)
plot_fourier_transform(pure_tone_frequency=1.1,

```

```
    signal_frequency=1,  
    time=time,  
    plot_centre_of_gravity=True,  
    plot_sum=False)
```



```
In [20]: time = np.linspace(0, 10, 100000)  
plot_fourier_transform(pure_tone_frequency=1.1,  
                      signal_frequency=1,  
                      time=time,  
                      plot_centre_of_gravity=True,  
                      plot_sum=False)
```



In [ ]:

# Discrete Fourier Transform

Valerio Velardo

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

Previously...

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi f t} dt$$

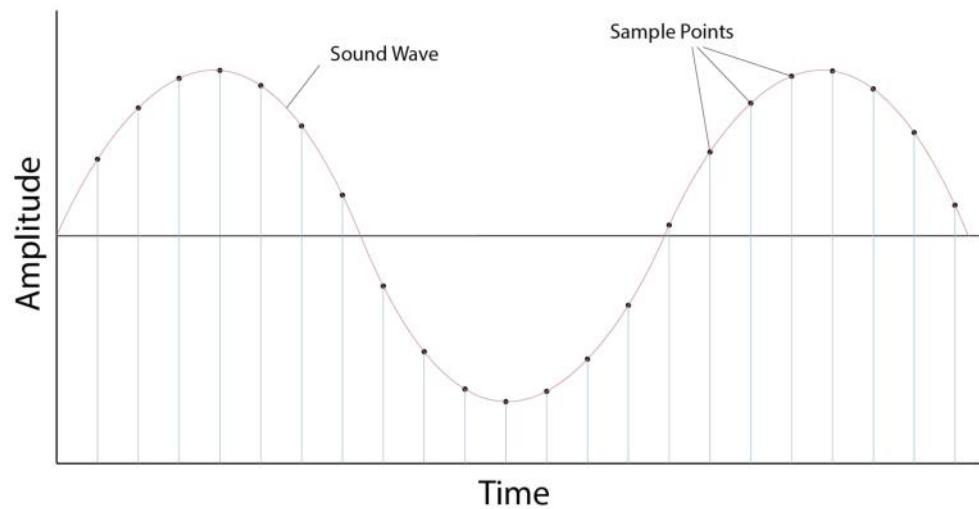
Previously...

---

$$\hat{g}(f) = \int [g(t)] \cdot e^{-i2\pi f t} dt$$

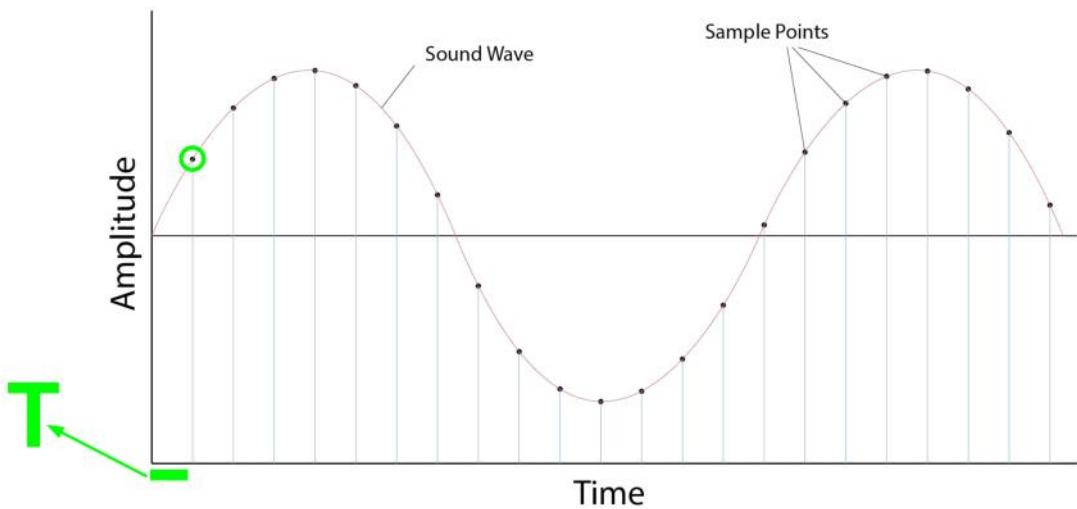
## Digitalization

---



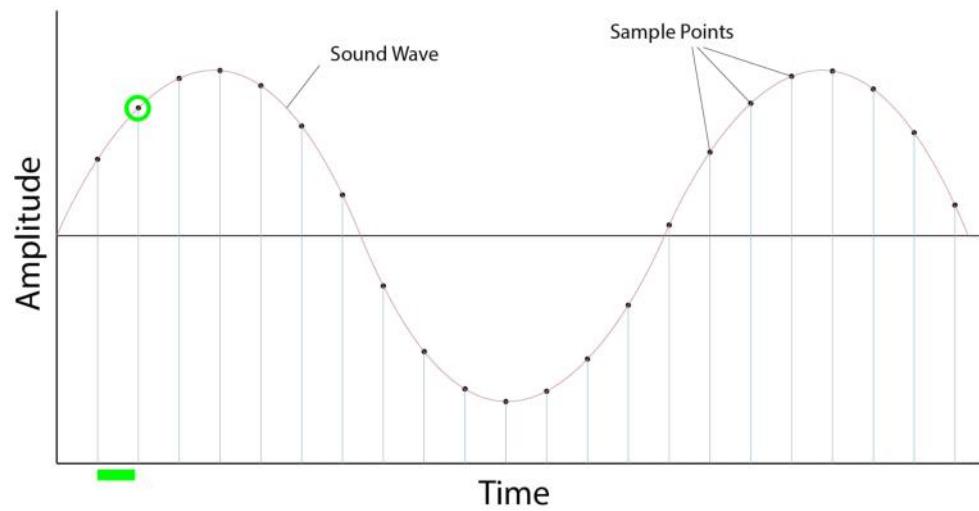
## Digitalization

---



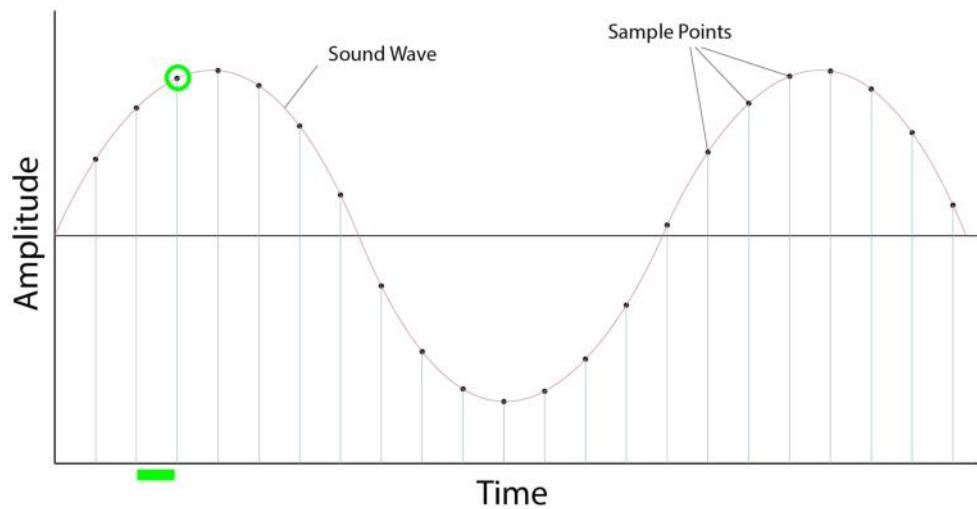
## Digitalization

---



## Digitalization

---



## Digital signal

---

$$g(t) \mapsto x(n)$$

## Digital signal

---

$$g(t) \mapsto x(n)$$

$$t = nT$$

## Building a discrete Fourier transform

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi f t} dt$$

## Building a discrete Fourier transform

---

$$\boxed{\hat{g}(f)} = \int g(t) \cdot e^{-i2\pi f t} dt$$

$$\boxed{\hat{x}(f)}$$

## Building a discrete Fourier transform

$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi f t} dt$$

$$\hat{x}(f) = \sum_n$$

## Building a discrete Fourier transform

$$\hat{g}(f) = \int [g(t)] \cdot e^{-i2\pi f t} dt$$

$$\hat{x}(f) = \sum_n [x(n)]$$

## Building a discrete Fourier transform

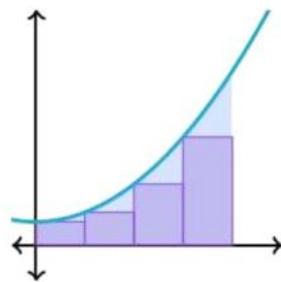
$$\hat{g}(f) = \int g(t) \cdot e^{-i2\pi ft} dt$$

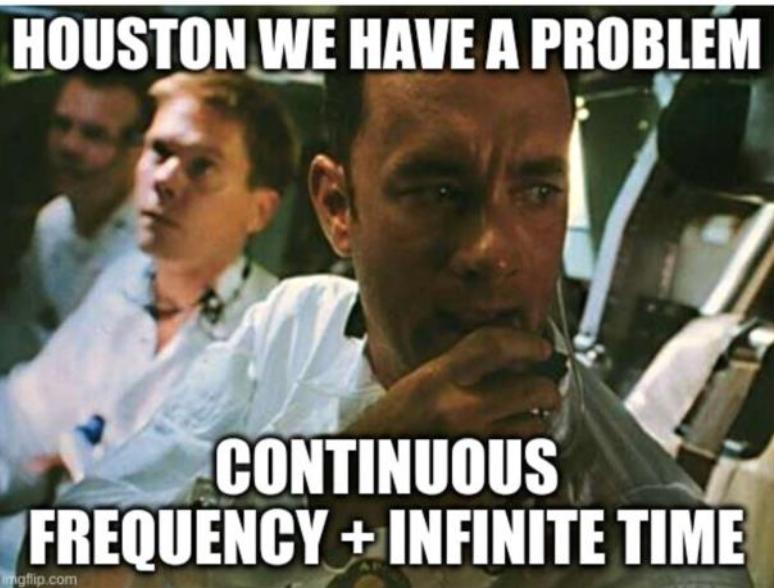
$$\hat{x}(f) = \sum_n x(n) \cdot e^{-i2\pi fn}$$

## DFT: Visual interpretation

---

$$\hat{x}(f) = \sum_n x(n) \cdot e^{-i2\pi f n}$$





## Building a discrete Fourier transform

$$\hat{x}(f) = \sum_n x(n) \cdot e^{-i2\pi f n}$$

## Hack 1: Time

---

- Consider  $f$  to be non 0 in a finite time interval
- $x(0), x(1), \dots, x(N-1)$

## Hack 2: Frequency

---

- Compute transform for finite # of frequencies

## Hack 2: Frequency

---

- Compute transform for finite # of frequencies
- $\# \text{ frequencies (M)} = \# \text{ samples (N)}$

## Hack 2: Frequency

---

- Compute transform for finite # of frequencies
- # frequencies ( $M$ ) = # samples ( $N$ )
- Why  $M = N$ ?
  - Invertible transformation
  - Computational efficient

Hacking our way around...

---



$$\hat{x}(f) = \sum_n x(n) \cdot e^{-i2\pi f n}$$

Hacking our way around...

---



$$\hat{x}(f) = \sum_n x(n) \cdot e^{-i2\pi f n}$$

Hacking our way around...

---



$$\hat{x}(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi f n}$$

Hacking our way around...

---



$$\hat{x}(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi f n}$$

Hacking our way around...

---


$$\hat{x}(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi f n}$$

Hacking our way around...

---



$$\hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

Hacking our way around...

---



$$\hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

Hacking our way around...

---



$$k = [0, M - 1] = [0, N - 1]$$

$$\hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

Hacking our way around...

---

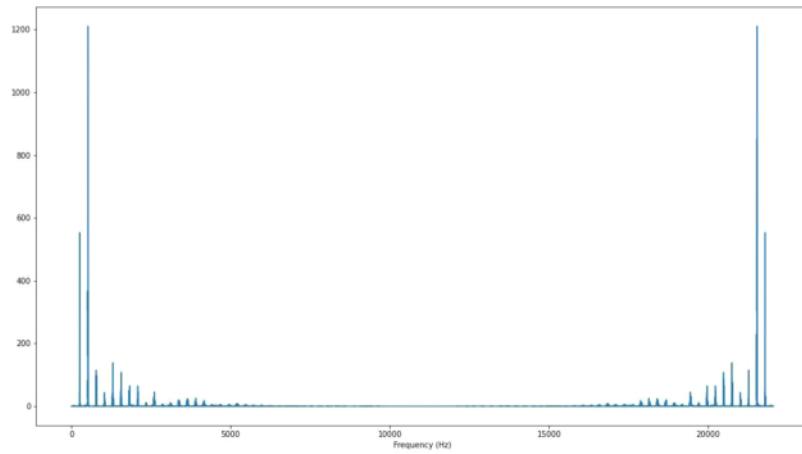
$$F(k) = \frac{k}{NT}$$

Hacking our way around...

$$F(k) = \frac{k}{NT} = \frac{ks_r}{N}$$

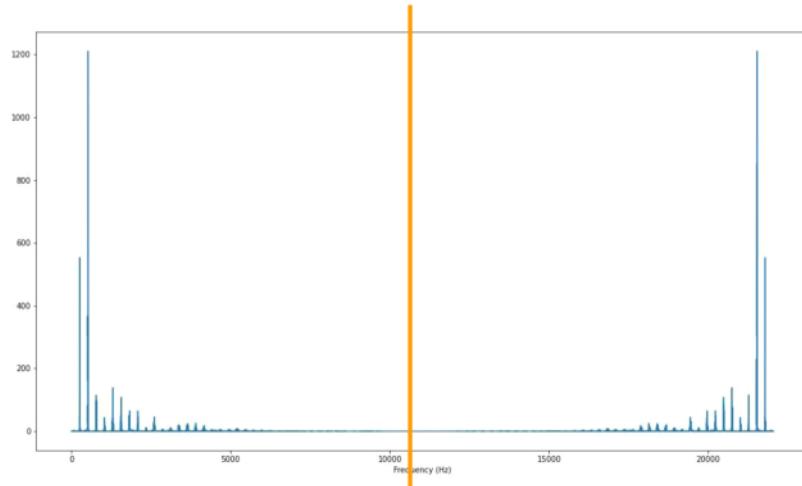
## Redundancy in DFT

---



## Redundancy in DFT

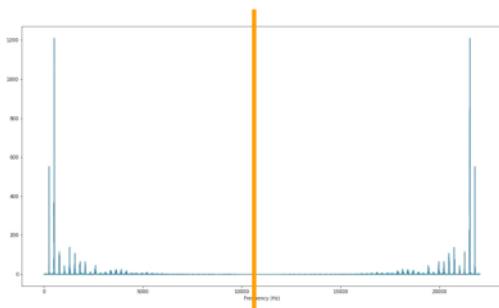
---



## Redundancy in DFT

---

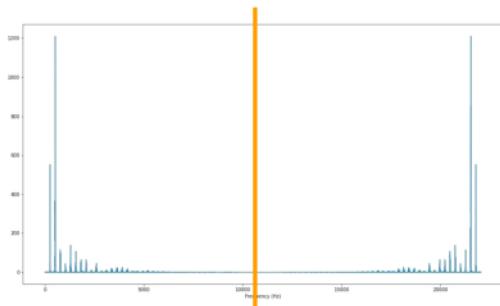
$$k = N/2$$



## Redundancy in DFT

---

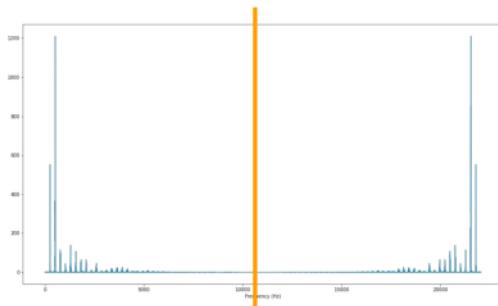
$$k = N/2 \rightarrow F(N/2) = s_r/2$$



## Redundancy in DFT

---

$$k = N/2 \rightarrow F(N/2) = s_r/2$$



Nyquist Frequency

## From DFT to Fast Fourier Transform

---

- DFT is computationally expensive ( $N^2$ )
- FFT is more efficient ( $N \log_2 N$ )
- FFT exploits redundancies across sinusoids
- FFT works when N is a power of 2

## What's up next?

---

- Play around with FFT

## 14 Code\_Visualising the Power Spectrum\_ipynb

04 January 2025 23:01



Code\_Visualising the Power Spectrum\_ipynb

```
In [7]: import os
import matplotlib.pyplot as plt
import librosa, librosa.display
import IPython.display as ipd
import numpy as np

In [8]: BASE_FOLDER = "audio/"
violin_sound_file = "violin_c.wav"
piano_sound_file = "piano_c.wav"
sax_sound_file = "sax.wav"
noise_sound_file = "noise.wav"

In [9]: ipd.Audio(os.path.join(BASE_FOLDER, violin_sound_file))

Out[9]:
```



```
In [10]: ipd.Audio(os.path.join(BASE_FOLDER, piano_sound_file))

Out[10]:
```



```
In [11]: ipd.Audio(os.path.join(BASE_FOLDER, sax_sound_file))

Out[11]:
```



```
In [12]: ipd.Audio(os.path.join(BASE_FOLDER, noise_sound_file))

Out[12]:
```



```
In [13]: # Load sounds
violin_c4, sr = librosa.load(os.path.join(BASE_FOLDER, violin_sound_file))
piano_c5, _ = librosa.load(os.path.join(BASE_FOLDER, piano_sound_file))
sax_c4, _ = librosa.load(os.path.join(BASE_FOLDER, sax_sound_file))
noise, _ = librosa.load(os.path.join(BASE_FOLDER, noise_sound_file))

In [14]: len(violin_c4)

Out[14]: 59772

In [15]: X = np.fft.fft(violin_c4)
len(X)

Out[15]: 59772

In [51]: def plot_magnitude_spectrum(signal, sr, title, f_ratio=1):
    X = np.fft.fft(signal)
    X_mag = np.absolute(X)

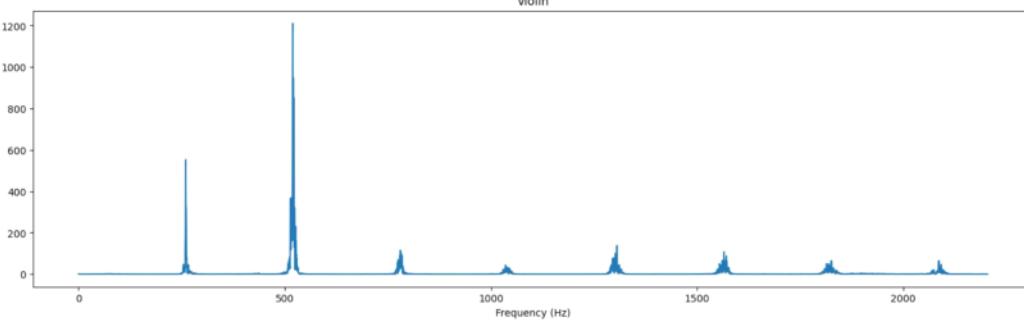
    plt.figure(figsize=(18, 5))

    f = np.linspace(0, sr, len(X_mag))
    f_bins = int(len(X_mag)*f_ratio)

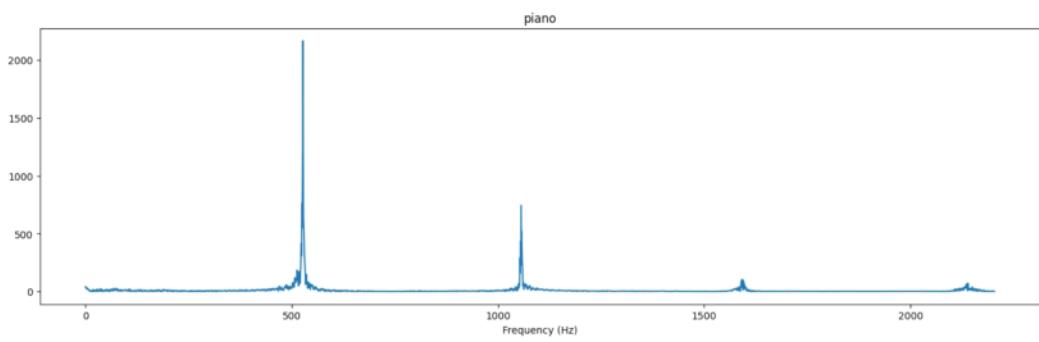
    plt.plot(f[:f_bins], X_mag[:f_bins])
    plt.xlabel('Frequency (Hz)')
    plt.title(title)

In [52]: plot_magnitude_spectrum(violin_c4, sr, "violin", 0.1)
```

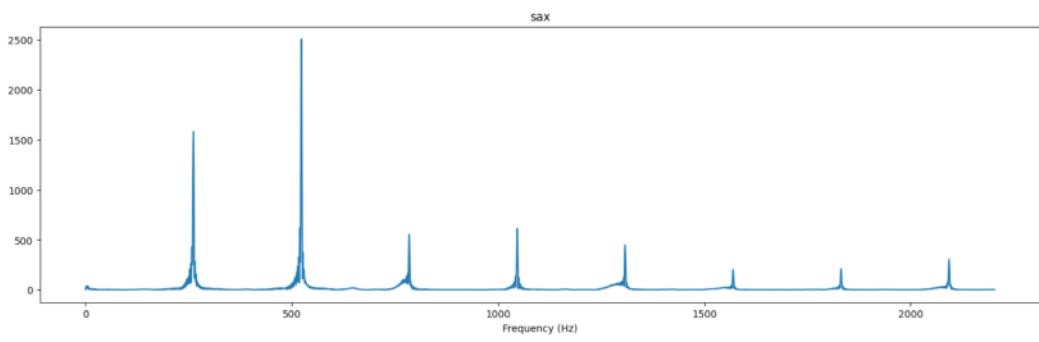
violin



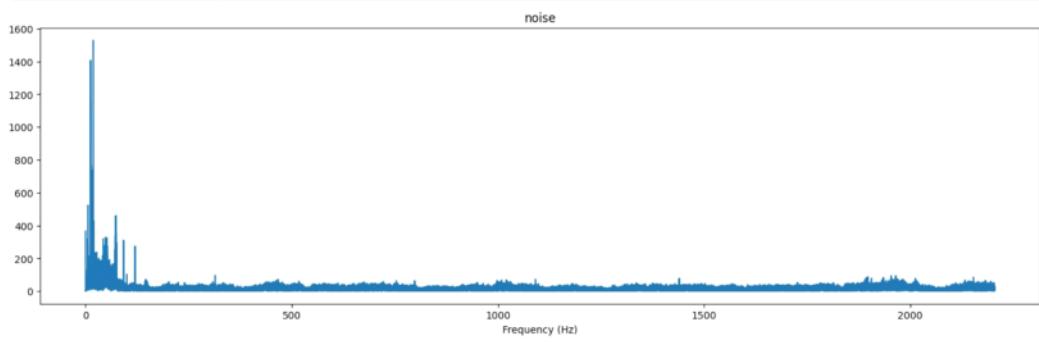
```
In [53]: plot_magnitude_spectrum(piano_c5, sr, "piano", 0.1)
```



```
In [54]: plot_magnitude_spectrum(sax_c4, sr, "sax", 0.1)
```



```
In [55]: plot_magnitude_spectrum(noise, sr, "noise", 0.1)
```



```
In [ ]:
```

# Short-Time Fourier Transform Explained Easily

Valerio Velardo

Join the community!



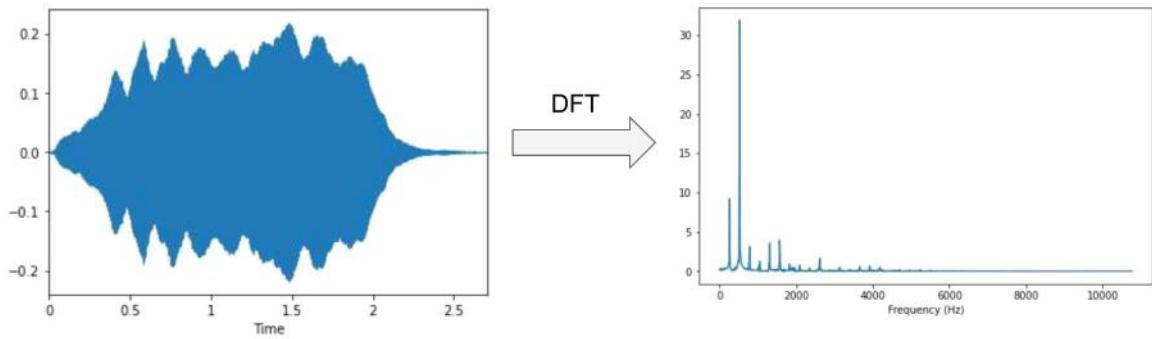
[thesoundofai.slack.com](https://thesoundofai.slack.com)

Previously...

$$\hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## Previously...

---



## **Fourier Transform Problem**

---

**WE KNOW WHAT  
WE DON'T KNOW WHEN**

**CONSIDER SMALL  
SEGMENTS OF THE SIGNAL**

**APPLY FFT LOCALLY**



imgflip.com

## STFT intuition

---

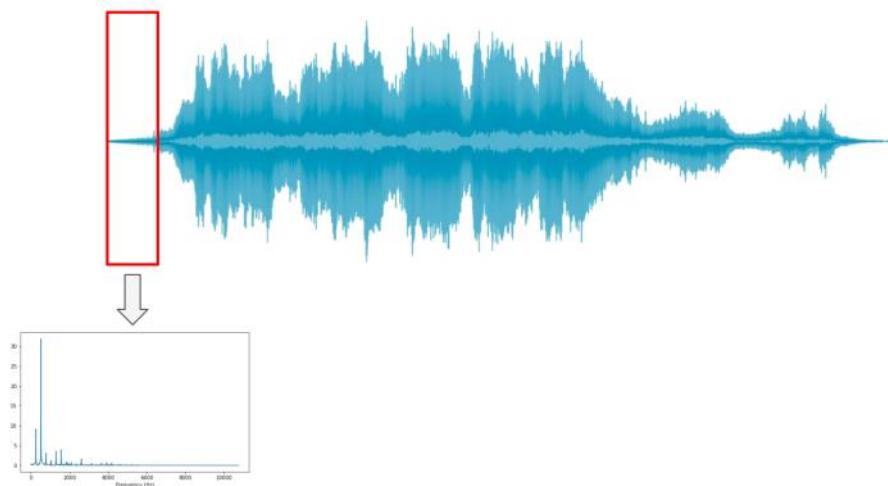


## STFT intuition

---

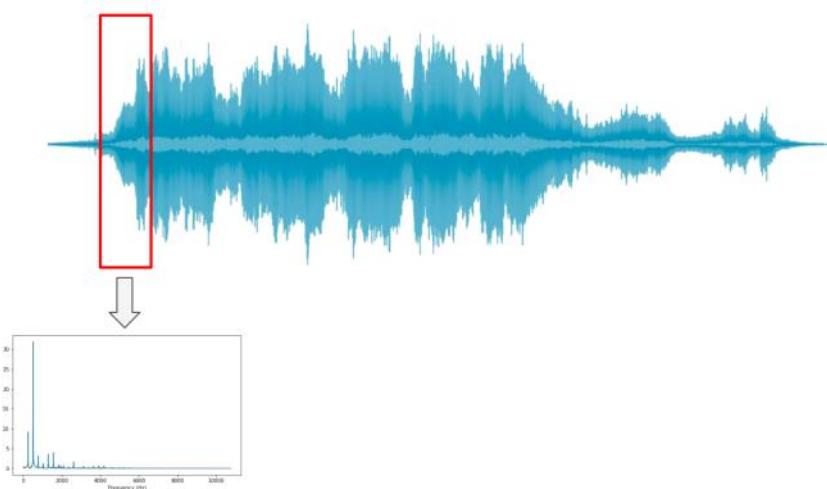


## STFT intuition



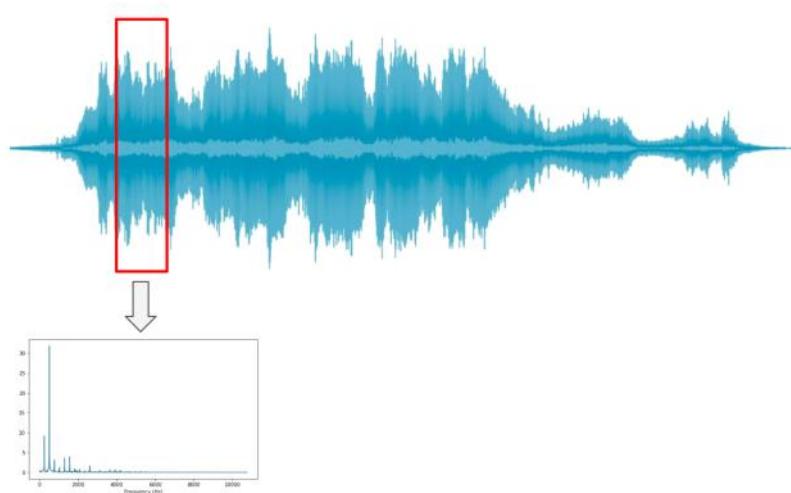
## STFT intuition

---



## STFT intuition

---



## Windowing

---

- Apply windowing function to signal

## Windowing

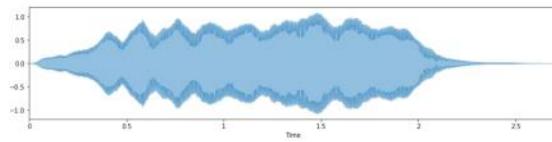
---

- Apply windowing function to signal

$$x_w(k) = x(k) \cdot w(k)$$

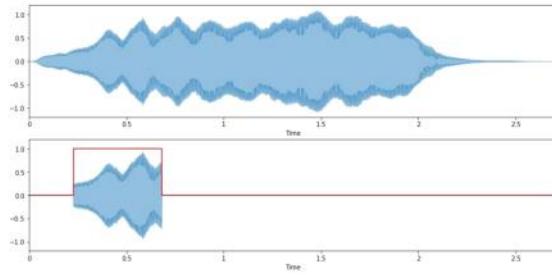
## Windowing

---



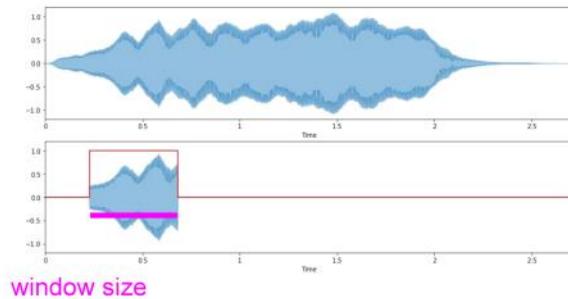
## Windowing

---



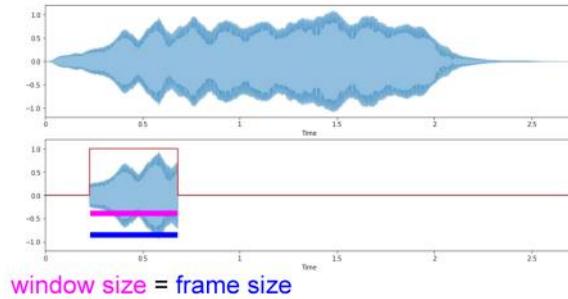
## Windowing

---



## Windowing

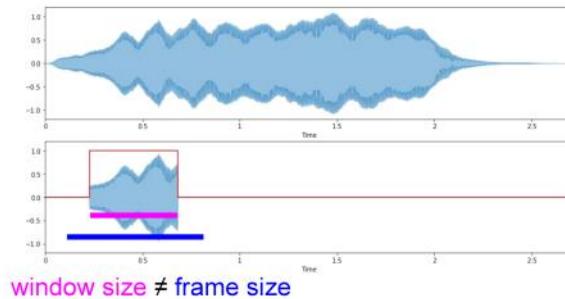
---



window size = frame size

## Windowing

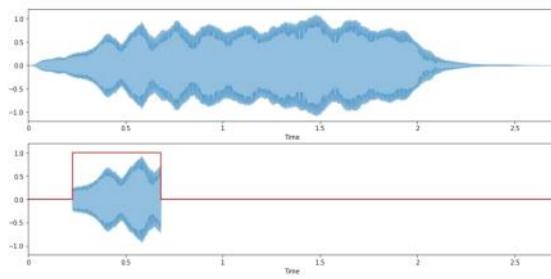
---



window size  $\neq$  frame size

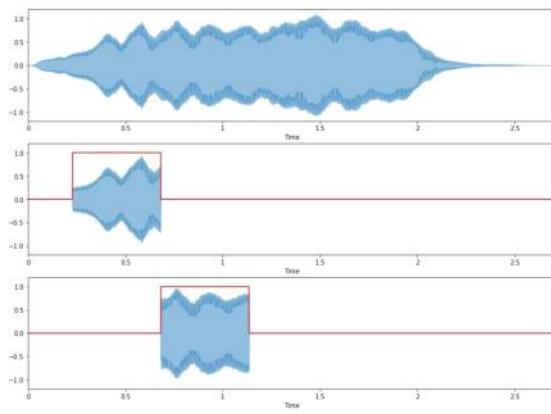
# STFT

---



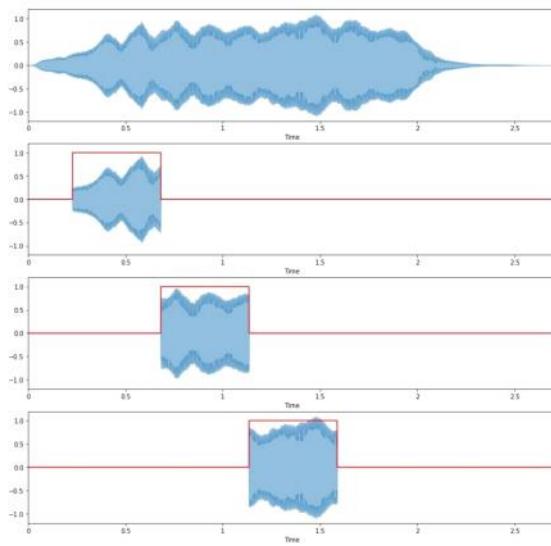
# STFT

---



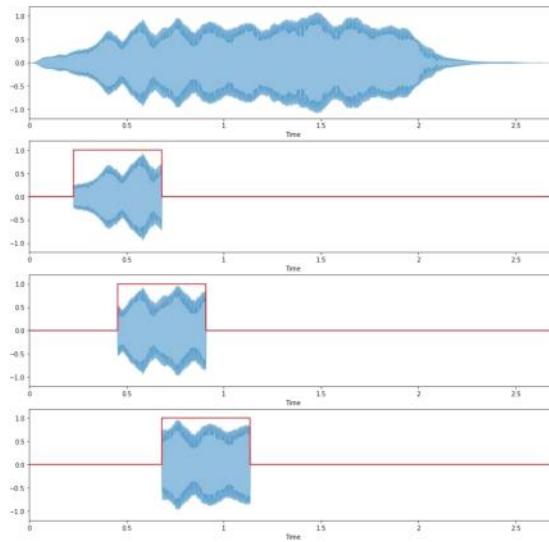
# STFT

---



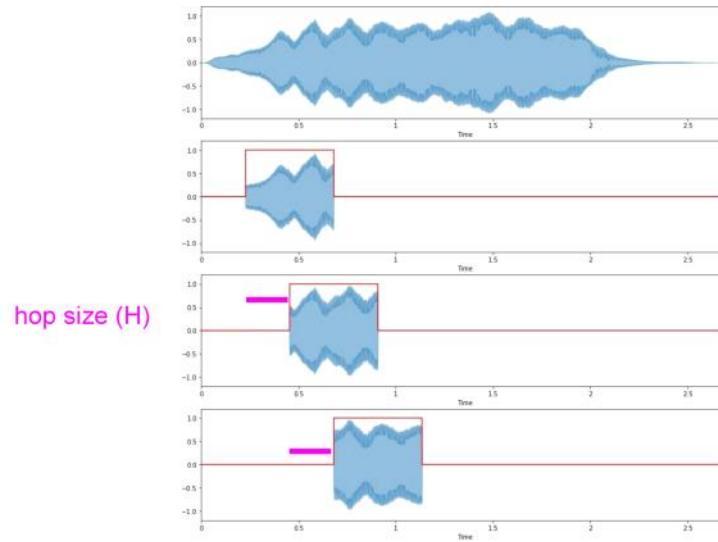
## Overlapping frames

---



## Overlapping frames

---



## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## From DFT to STFT

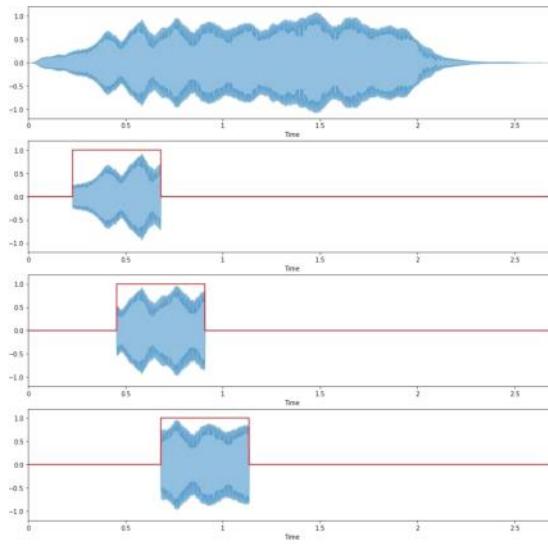
---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

$$S(\textcolor{green}{m}, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

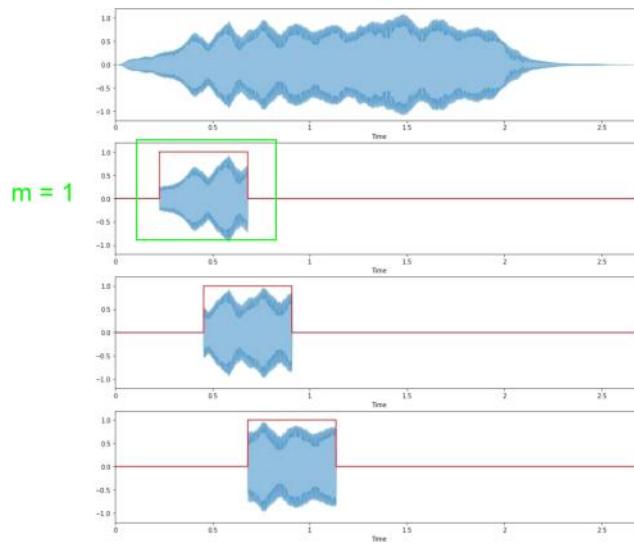
## From DFT to STFT

---



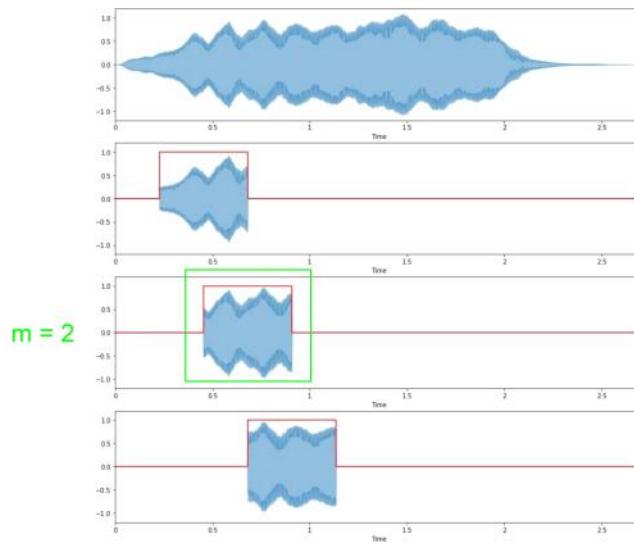
## From DFT to STFT

---



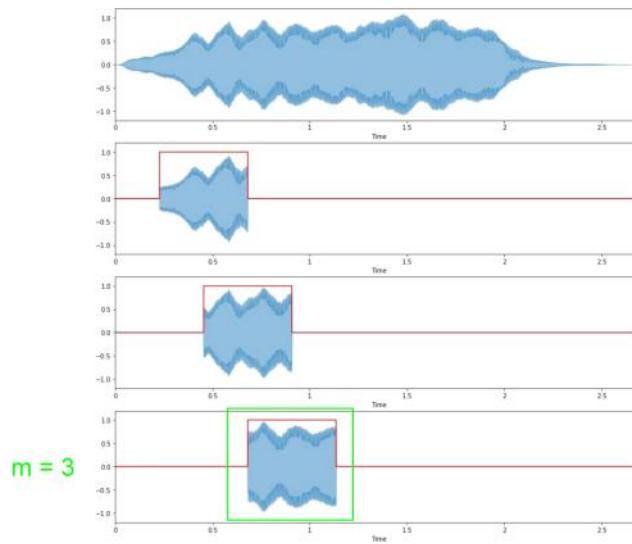
## From DFT to STFT

---



## From DFT to STFT

---



## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

*N = allSamples in signal*

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

*N = FrameSize*

## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

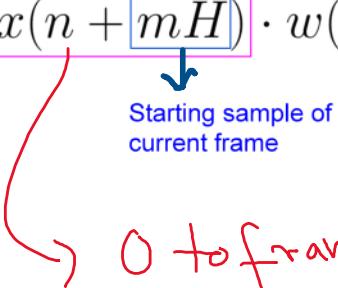
$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

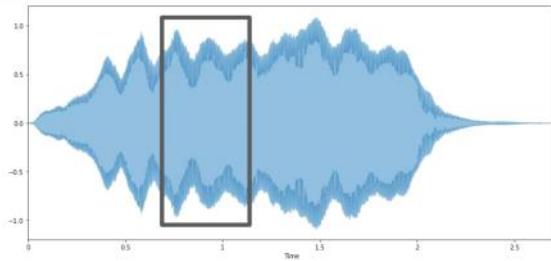
$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

  
Starting sample of  
current frame

→ 0 to frame size

## From DFT to STFT

---



## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$
$$S(m, k) = \sum_{n=0}^{N-1} x(n + \boxed{mH}) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

Starting sample of  
current frame

*windows*

*pure tone*

*fr*

## From DFT to STFT

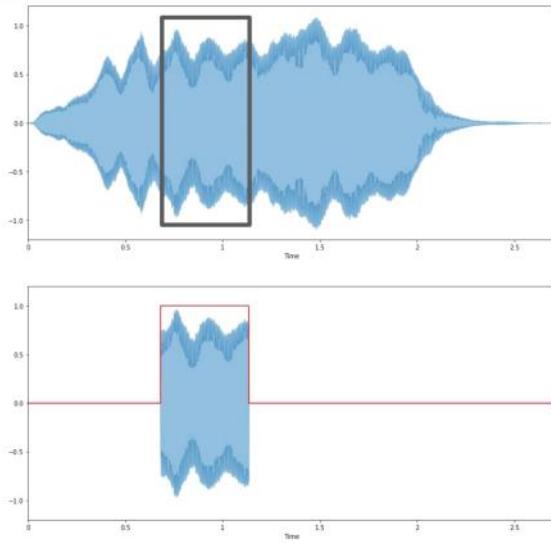
---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## From DFT to STFT

---



## From DFT to STFT

---

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}$$

## Outputs

---

- DFT
  - Spectral vector (# frequency bins)
  - N complex Fourier coefficients

## Outputs

---

- DFT
  - Spectral vector (# frequency bins)
  - N complex Fourier coefficients
- STFT
  - Spectral matrix (# frequency bins, # frames)
  - Complex Fourier coefficients

## Outputs

---

$$\# \text{ frequency bins} = \frac{\textit{framesize}}{2} + 1$$

## Outputs

---

$$\# \text{ frequency bins} = \frac{\textit{framesize}}{2} + 1$$

$$\# \text{ frames} = \frac{\textit{samples} - \textit{framesize}}{\textit{hopsize}} + 1$$

## Example STFT output

---

- Signal = 10K samples
- Frame size = 1000
- Hop size = 500

## Example STFT output

---

- Signal = 10K samples
- Frame size = 1000
- Hop size = 500

# frequency bins =  $1000 / 2 + 1 = 501$

## Example STFT output

---

- Signal = 10K samples
- Frame size = 1000
- Hop size = 500

# frequency bins =  $1000 / 2 + 1 = 501 \rightarrow (0, \text{ sampling rate}/2)$

## Example STFT output

---

- Signal = 10K samples
- Frame size = 1000
- Hop size = 500

# frequency bins =  $1000 / 2 + 1 = 501 \rightarrow (0, \text{sampling rate}/2)$

# frames =  $(10000 - 1000) / 500 + 1 = 19$

## Example STFT output

---

- Signal = 10K samples
- Frame size = 1000
- Hop size = 500

STFT -> (501, 19)

## **STFT parameters**

---

- Frame size

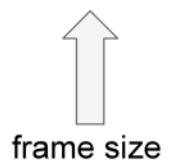
## STFT parameters

- Frame size

512, 1024, 2048, 4096, 8192

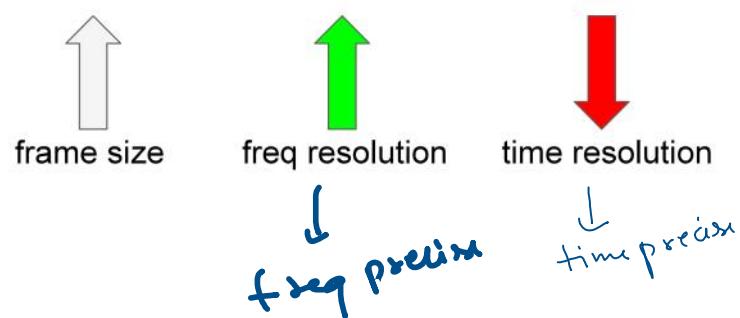
## Time / frequency trade off

---



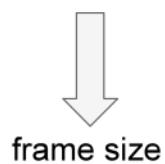
frame size

## Time / frequency trade off



## Time / frequency trade off

---



frame size

## Time / frequency trade off



App specific

## STFT parameters

---

- Frame size
- Hop size

## STFT parameters

---

- Frame size
- Hop size

256, 512, 1024, 2048, 4096

## STFT parameters

---

- Frame size
- Hop size

256, 512, 1024, 2048, 4096

$\frac{1}{2} K$ ,  $\frac{1}{4} K$ ,  $\frac{1}{8} K$

## STFT parameters

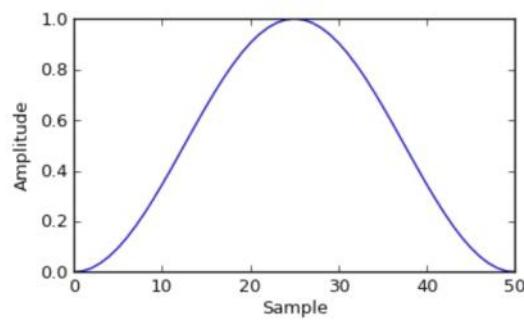
---

- Frame size
- Hop size
- Windowing function

## Hann window

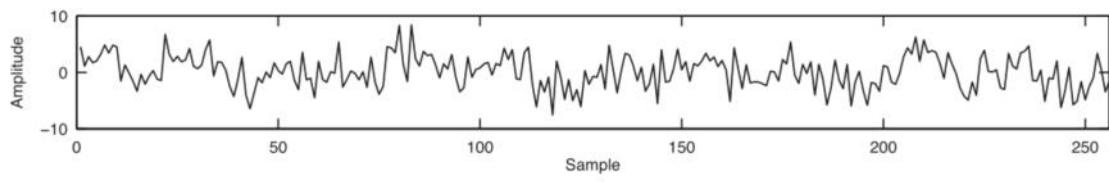
---

$$w(k) = 0.5 \cdot (1 - \cos\left(\frac{2\pi k}{K-1}\right)), k = 1 \dots K$$



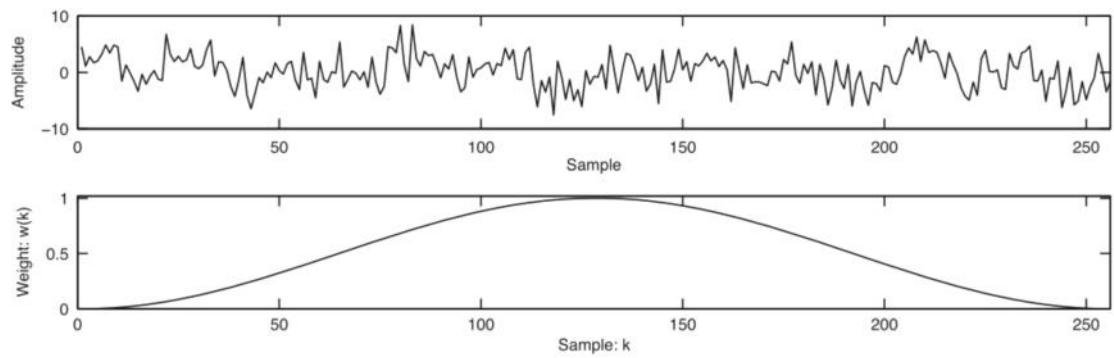
## Hann window

---



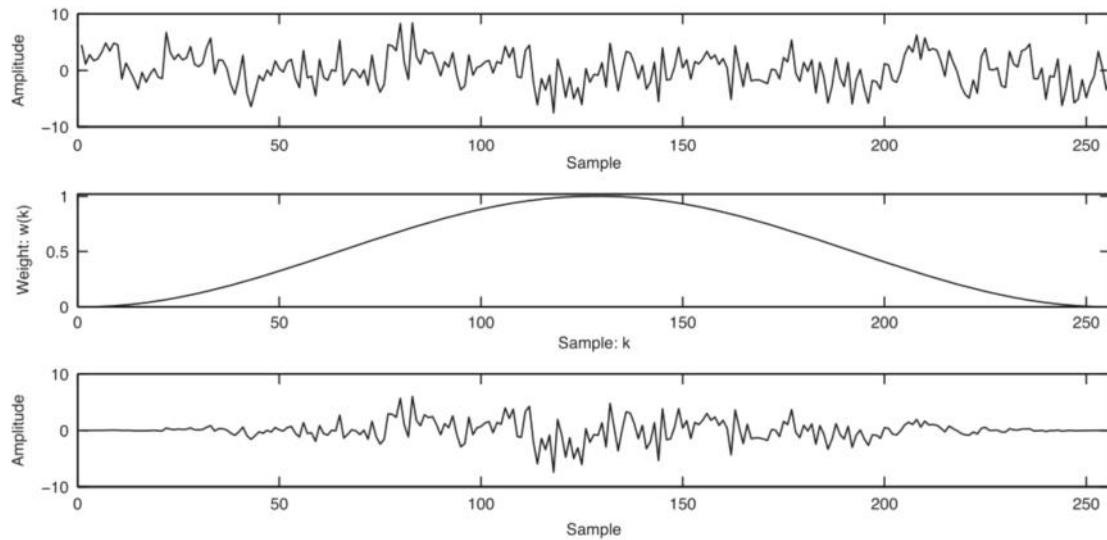
## Hann window

---



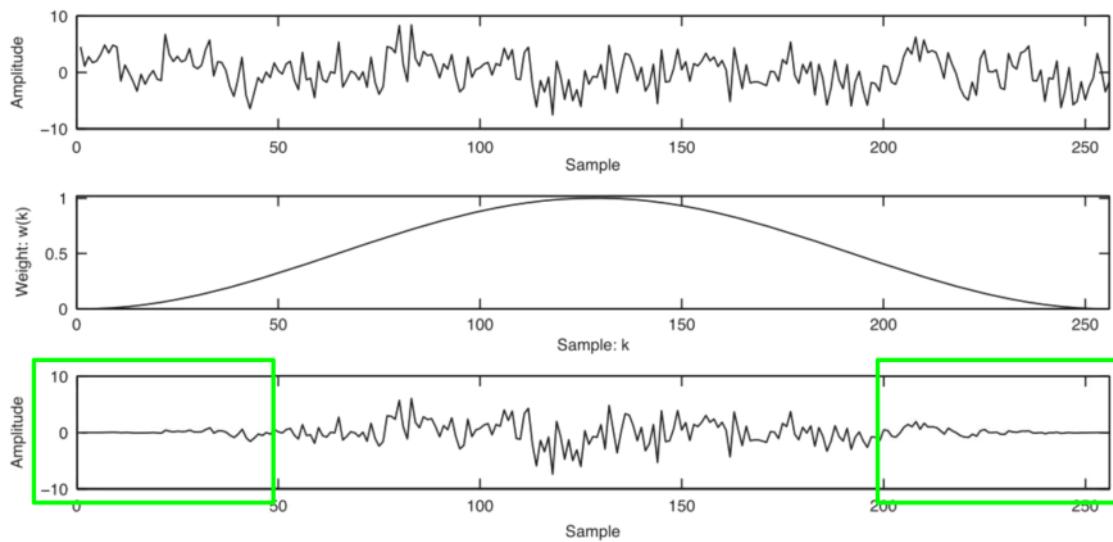
## Hann window

---



## Hann window

---



## Visualising sound

---

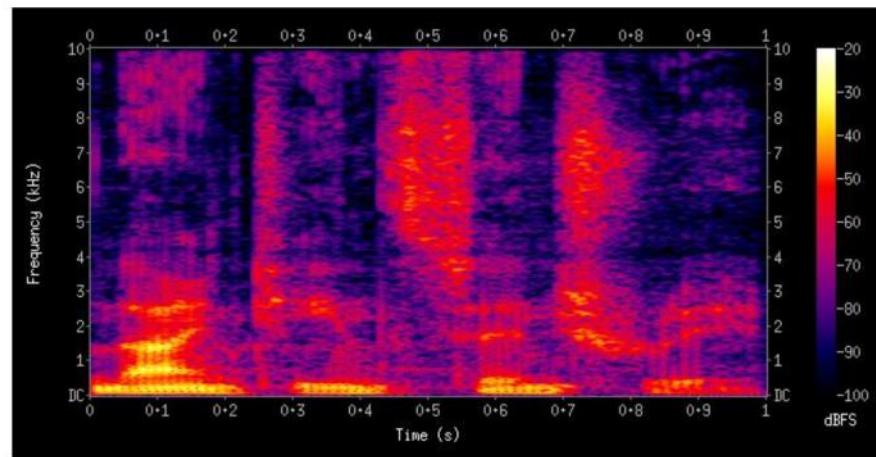
## Visualising sound

---

$$Y(m, k) = |S(m, k)|^2$$

## Spectrogram

---



## What's up next?

---

- Extract spectrograms with Librosa
- Discuss different flavours of spectrograms
- Examine different audio data

## 16 Code\_Extracting Spectrograms from Audio with Python\_ipynb

04 January 2025 23:02



Code\_Extracting Spectrograms from Audio with Python...

```
In [1]: import os
import librosa
import librosa.display
import IPython.display as ipd
import numpy as np
import matplotlib.pyplot as plt
```

### Loading audio files with Librosa

```
In [2]: scale_file = "audio/scale.wav"
debussy_file = "audio/debussy.wav"
redhot_file = "audio/redhot.wav"
duke_file = "audio/duke.wav"
```

```
In [3]: ipd.Audio(scale_file)
```

```
Out[3]:
```



```
In [4]: ipd.Audio(debussy_file)
```

```
Out[4]:
```



```
In [5]: ipd.Audio(redhot_file)
```

```
Out[5]:
```



```
In [6]: ipd.Audio(duke_file)
```

```
Out[6]:
```



```
In [7]: # load audio files with librosa
scale, sr = librosa.load(scale_file)
debussy, _ = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)
```

### Extracting Short-Time Fourier Transform

```
In [8]: FRAME_SIZE = 2048
HOP_SIZE = 512
```

```
In [9]: S_scale = librosa.stft(scale, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)
```

```
In [10]: S_scale.shape
```

```
Out[10]: (1025, 342)
```

```
In [11]: type(S_scale[0][0])
```

```
Out[11]: numpy.complex64
```

### Calculating the spectrogram

```
In [12]: Y_scale = np.abs(S_scale) ** 2
```

```
In [13]: Y_scale.shape
```

```
Out[13]: (1025, 342)
```

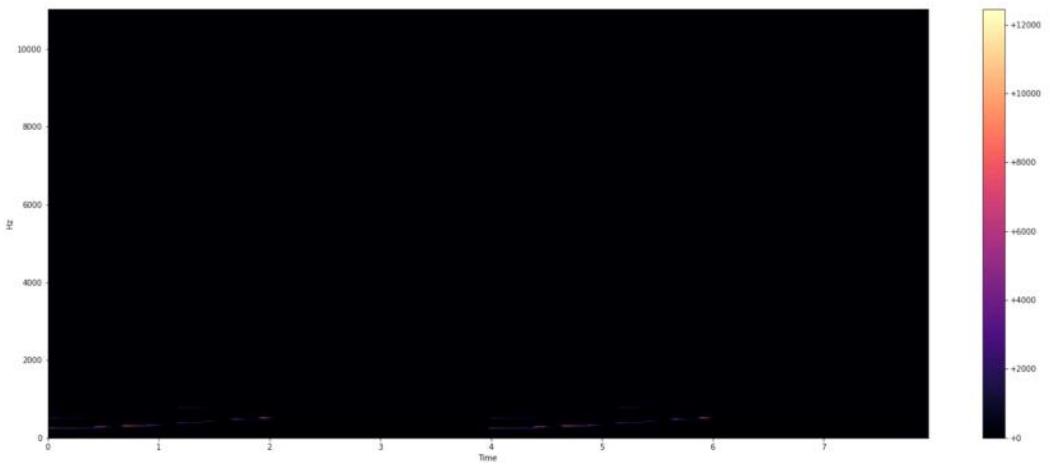
```
In [14]: type(Y_scale[0][0])
```

```
Out[14]: numpy.float32
```

### Visualizing the spectrogram

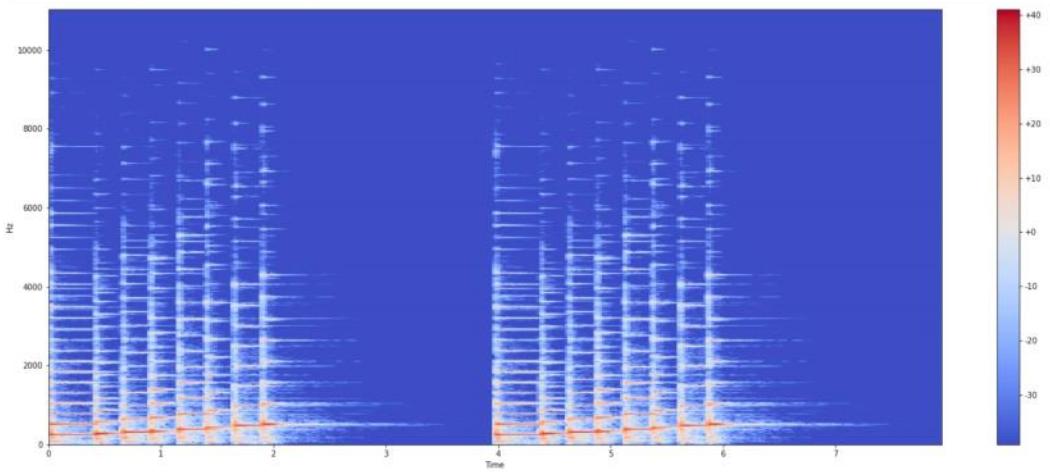
```
In [15]: def plot_spectrogram(Y, sr, hop_length, y_axis="linear"):
    plt.figure(figsize=(25, 10))
    librosa.display.specshow(Y,
                            sr=sr,
                            hop_length=hop_length,
                            x_axis="time",
                            y_axis=y_axis)
    plt.colorbar(format="%+2.f")
```

```
In [16]: plot_spectrogram(Y_scale, sr, HOP_SIZE)
```



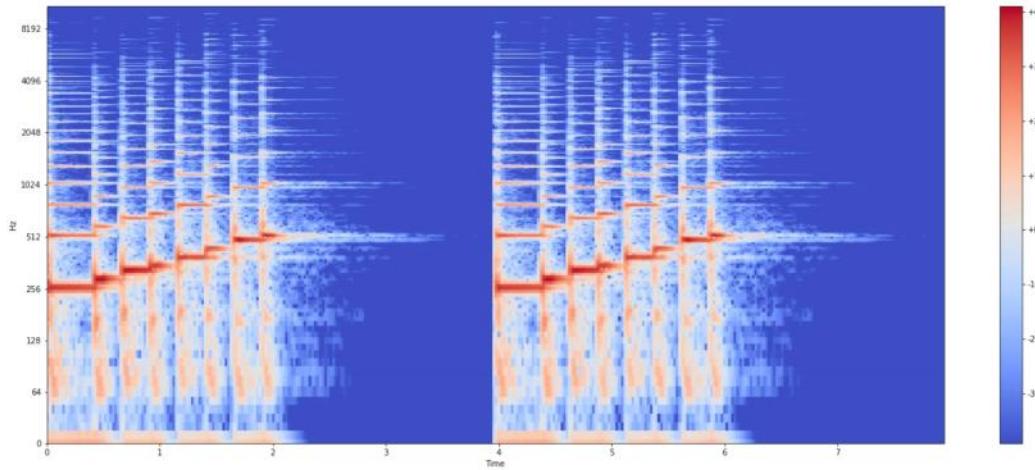
Log-Amplitude Spectrogram

```
In [21]: Y_log_scale = librosa.power_to_db(Y_scale)
plot_spectrogram(Y_log_scale, sr, HOP_SIZE)
```



Log-Frequency Spectrogram

```
In [22]: plot_spectrogram(Y_log_scale, sr, HOP_SIZE, y_axis="log")
```

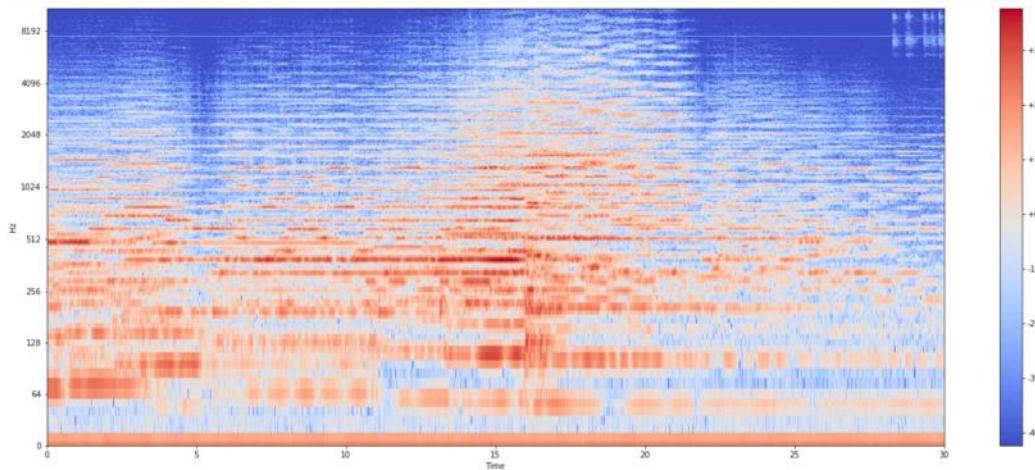


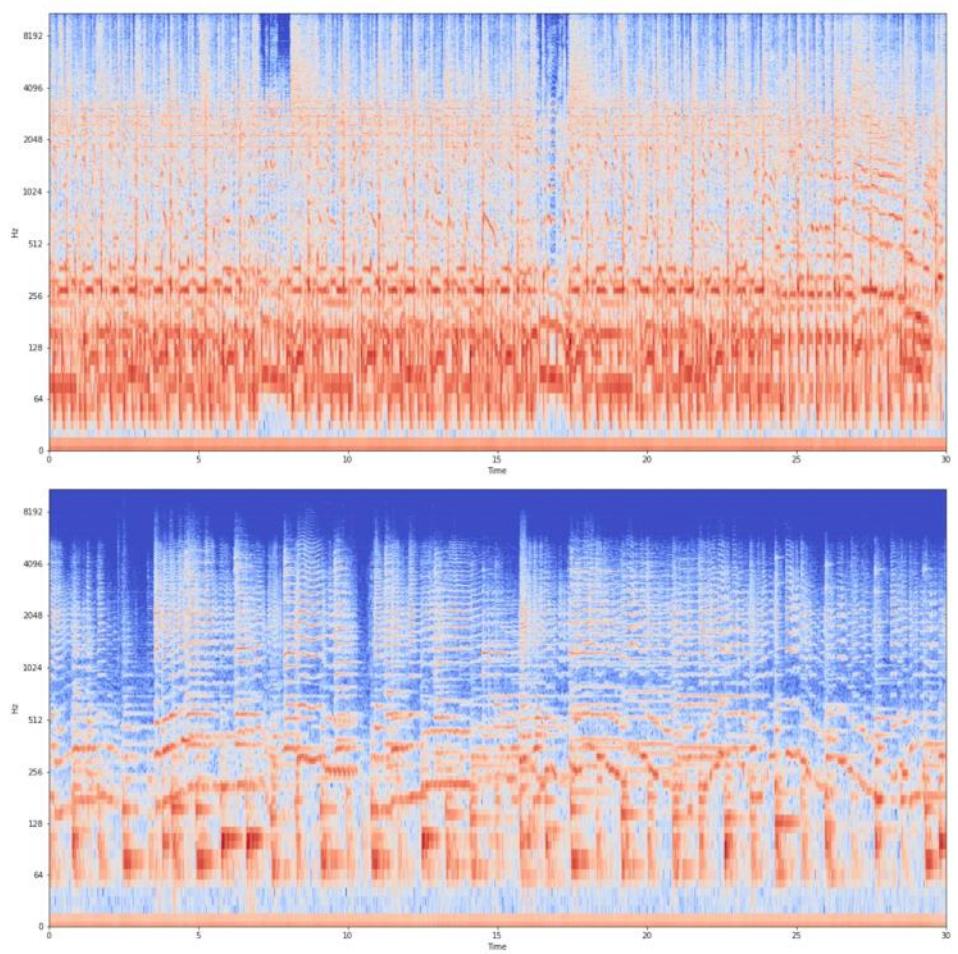
Visualising songs from different genres

```
In [23]: S_debussy = librosa.stft(debussy, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)
S_redhot = librosa.stft(redhot, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)
S_duke = librosa.stft(duke, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)

Y_debussy = librosa.power_to_db(np.abs(S_debussy) ** 2)
Y_redhot = librosa.power_to_db(np.abs(S_redhot) ** 2)
Y_duke = librosa.power_to_db(np.abs(S_duke) ** 2)

plot_spectrogram(Y_debussy, sr, HOP_SIZE, y_axis="log")
plot_spectrogram(Y_redhot, sr, HOP_SIZE, y_axis="log")
plot_spectrogram(Y_duke, sr, HOP_SIZE, y_axis="log")
```



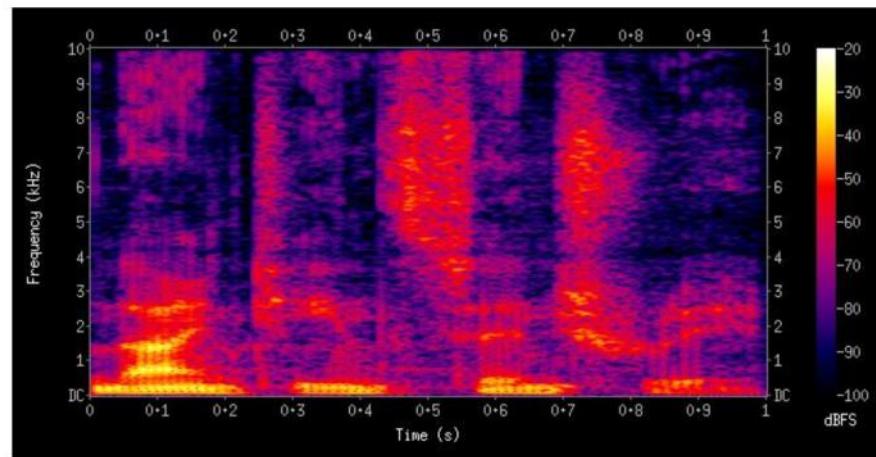


# Mel-spectrograms Explained Easily

Valerio Velardo

## Previously...

---



## Psychoacoustic experiment

## Psychoacoustic experiment

- 1st sample: C2 - C4 -> (65 - 262Hz)
- 2nd sample: G6 - A6 -> (1568 - 1760Hz)

## Psychoacoustic experiment

- 1st sample: C2 - C4 -> (65 - 262Hz)
- 2nd sample: G6 - A6 -> (1568 - 1760Hz)

**200 Hz**

We have a problem!

---

Humans perceive frequency  
logarithmically

## Ideal audio feature

---

- Time-frequency representation

## Ideal audio feature

---

- Time-frequency representation
- Perceptually-relevant amplitude representation

## Ideal audio feature

---

- Time-frequency representation
- Perceptually-relevant amplitude representation
- **Perceptually-relevant frequency representation**

## Ideal audio feature

---

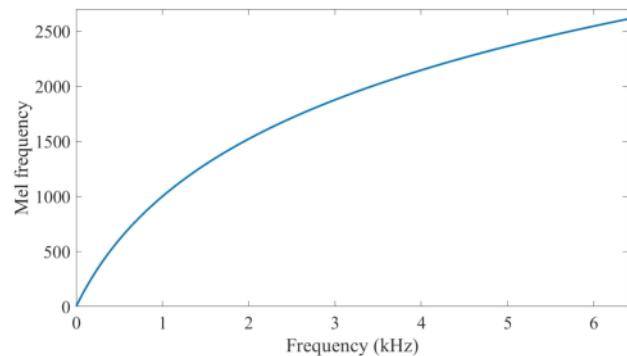
- Time-frequency representation
- Perceptually-relevant amplitude representation
- Perceptually-relevant frequency representation

## Mel spectrograms

## Mel-scale

---

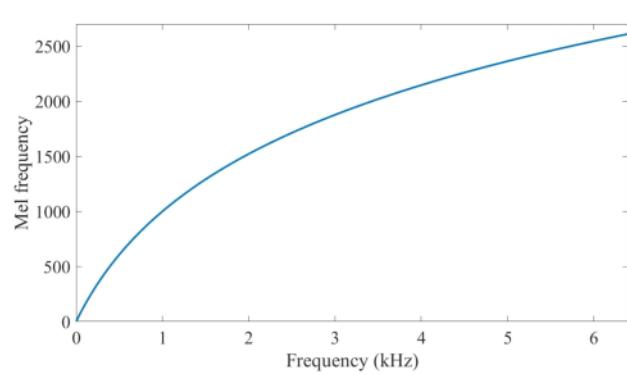
- Logarithmic scale
- Equal distances on the scale have same “perceptual” distance
- 1000 Hz = 1000 Mel



## Mel-scale

---

$$m = 2595 \cdot \log\left(1 + \frac{f}{500}\right)$$

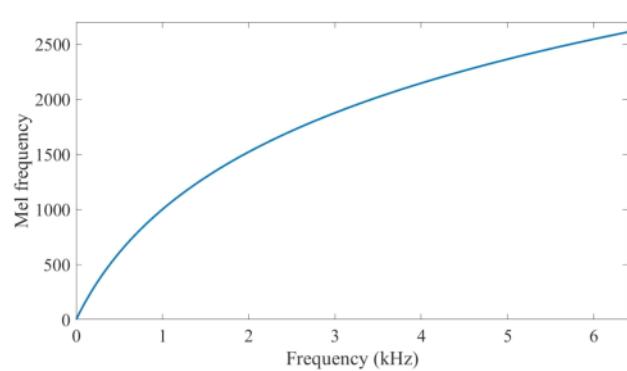


## Mel-scale

---

$$m = 2595 \cdot \log\left(1 + \frac{f}{500}\right)$$

$$f = 700\left(10^{m/2595} - 1\right)$$



## Recipe to extract Mel spectrogram

1. Extract STFT
2. Convert amplitude to DBs
3. Convert frequencies to Mel scale

## Recipe to extract Mel spectrogram

---

1. Extract STFT
2. Convert amplitude to DBs
3. Convert frequencies to Mel scale

## Convert frequencies to Mel scale

---

1. Choose number of mel bands
2. Construct mel filter banks
3. Apply mel filter banks to spectrogram

How many mel bands?

---

How many mel bands?

---

40

How many mel bands?

---

40

60

How many mel bands?

---

40

60

90

How many mel bands?

---

40      128  
60      90

How many mel bands?

---

40            128  
60            90

It depends on the problem!

## Convert frequencies to Mel scale

---

1. Choose number of mel bands
2. Construct mel filter banks
3. Apply mel filter banks to spectrogram

## Mel filter banks

---

1. Convert lowest / highest frequency to Mel

$$m = 2595 \cdot \log\left(1 + \frac{f}{500}\right)$$

## Mel filter banks

---

1. Convert lowest / highest frequency to Mel
2. Create # bands equally spaced points



## Mel filter banks

---

1. Convert lowest / highest frequency to Mel
2. Create # bands equally spaced points



## Mel filter banks

---

1. Convert lowest / highest frequency to Mel
2. Create # bands equally spaced points
3. Convert points back to Hertz

$$f = 700(10^{m/2595} - 1)$$

## Mel filter banks

---

1. Convert lowest / highest frequency to Mel
2. Create # bands equally spaced points
3. Convert points back to Hertz
4. Round to nearest frequency bin

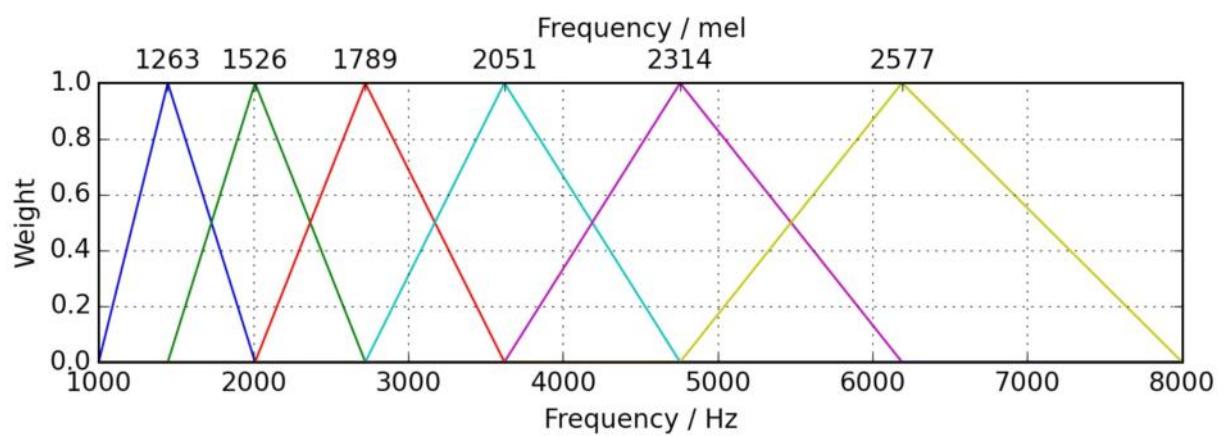
## Mel filter banks

---

1. Convert lowest / highest frequency to Mel
2. Create # bands equally spaced points
3. Convert points back to Hertz
4. Round to nearest frequency bin
5. **Create triangular filters**

## Mel filter banks

---



## Mel filter banks' shape

---

(# bands, framesize / 2 + 1)

## Convert frequencies to Mel scale

---

1. Choose number of mel bands
2. Construct mel filter banks
3. Apply mel filter banks to spectrogram

## Applying mel filter banks to spectrogram

$$M = (\# \text{ bands}, \text{framesize} / 2 + 1)$$

## Applying mel filter banks to spectrogram

$$M = (\# \text{ bands}, \text{framesize} / 2 + 1)$$

$$Y = (\text{framesize} / 2 + 1, \# \text{ frames})$$

## Applying mel filter banks to spectrogram

$M = (\# \text{ bands}, \text{framesize} / 2 + 1)$

$Y = (\text{framesize} / 2 + 1, \# \text{ frames})$

## Applying mel filter banks to spectrogram

Mel spectrogram =  $MY$

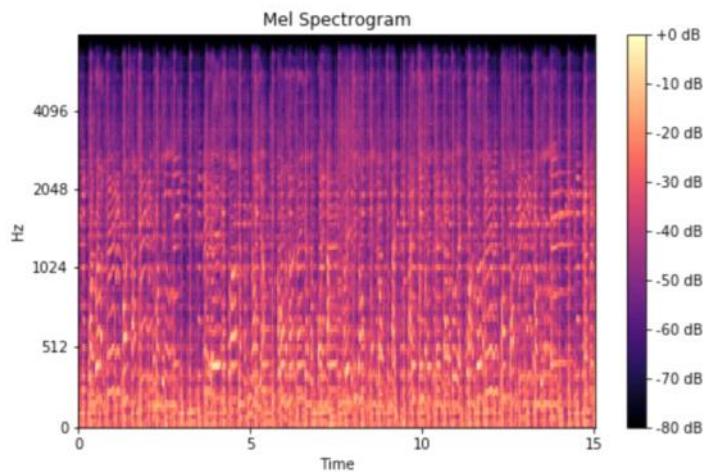
## Applying mel filter banks to spectrogram

Mel spectrogram =  $MY$

(# bands, # frames)

## Applying mel filter banks to spectrogram

---



## Mel spectrogram applications

---

- Audio classification
- Automatic mood recognition
- Music genre classification
- Music instrument classification
- ...

## What's up next?

---

- Extract Mel spectrograms with Python and Librosa
- Visualise Mel spectrograms
- Extract and visualise Mel filter banks

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

## 18 Code\_Extracting Mel Spectrograms

04 January 2025 23:03



Code\_Extracting Mel Spectrograms

```
In [1]: import librosa
import librosa.display
import IPython.display as ipd
import matplotlib.pyplot as plt
```

## Loading audio files with Librosa

```
In [2]: scale_file = "audio/scale.wav"
```

```
In [3]: ipd.Audio(scale_file)
```

```
Out[3]:
```



```
In [4]: # Load audio files with librosa
scale, sr = librosa.load(scale_file)
```

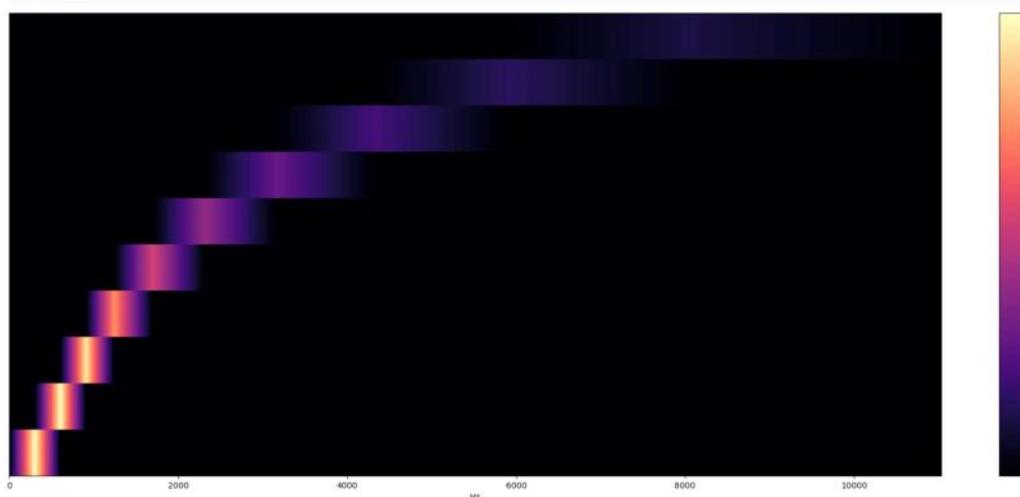
## Mel filter banks

```
In [5]: filter_banks = librosa.filters.mel(n_fft=2048, sr=22050, n_mels=10)
```

```
In [6]: filter_banks.shape
```

```
Out[6]: (10, 1025)
```

```
In [7]: plt.figure(figsize=(25, 10))
librosa.display.specshow(filter_banks,
                        sr=sr,
                        x_axis="linear")
plt.colorbar(format="%+2.f")
plt.show()
```



## Extracting Mel Spectrogram

```
In [9]: mel_spectrogram = librosa.feature.melspectrogram(y=scale, sr=sr, n_fft=2048, hop_length=512, n_mels=10)
```

```
In [10]: mel_spectrogram.shape
```

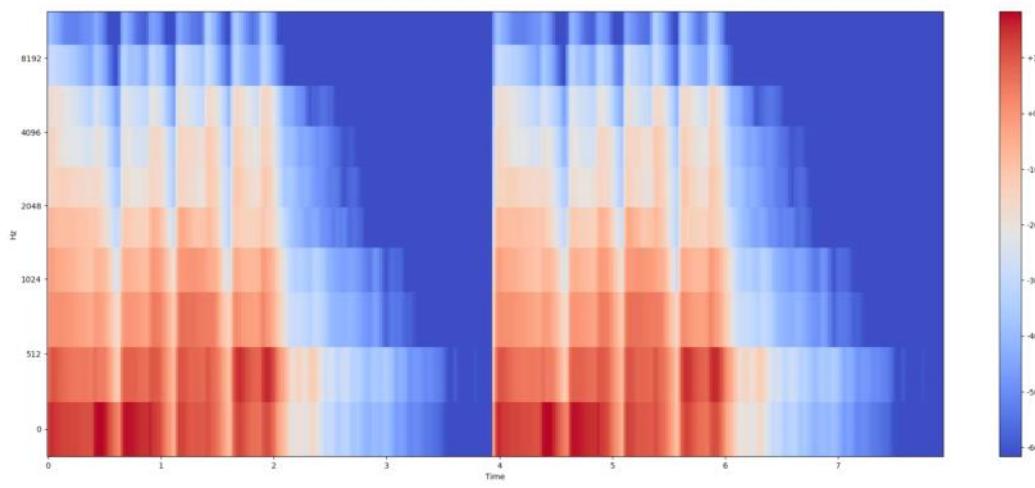
```
Out[10]: (10, 342)
```

```
In [11]: log_mel_spectrogram = librosa.power_to_db(mel_spectrogram)
```

```
In [12]: log_mel_spectrogram.shape
```

```
Out[12]: (10, 342)
```

```
In [13]: plt.figure(figsize=(25, 10))
librosa.display.specshow(log_mel_spectrogram,
                        x_axis="time",
                        y_axis="mel",
                        sr=sr)
plt.colorbar(format="%+2.f")
plt.show()
```



In [ ]:

# Mel-Frequency Cepstral Coefficients Explained Easily

Valerio Velardo

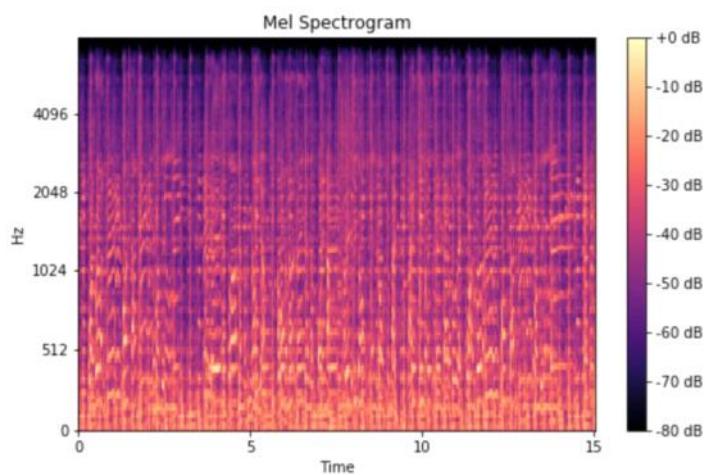
Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

## Previously...

---



# Mel-Frequency Cepstral Coefficients

# Mel-Frequency Cepstral Coefficients

# Mel-Frequency Cepstral **Coefficients**

# Mel-Frequency **Cepstral** Coefficients

# Cepstrum

# Cepstrum

Cepstrum  
↓  
Spectrum

Cepstrum  
↓  
Spectrum

Cepstrum      Quefrency      Liftering      Rhamonic  
↓  
Spectrum



## An historical note on Cepstrum

---

- Developed while studying echoes in seismic signals (1960s)
- Audio feature of choice for speech recognition / identification (1970s)
- Music processing (2000s)

## Computing the cepstrum

$$C(x(t)) = F^{-1}[\log(F[x(t)])]$$

## Computing the cepstrum

---

Time-domain  
signal

$$C[x(t)] = F^{-1}[\log(F[x(t)])]$$

## Computing the cepstrum

$$C(\boxed{x(t)}) = F^{-1}[\log(F[\boxed{x(t)}])]$$

## Computing the cepstrum

---

Time-domain signal

$$C[x(t)] = F^{-1}[\log(F[x(t)])]$$

Spectrum

Log spectrum

## Computing the cepstrum

---

Time-domain signal

$$C[x(t)] = F^{-1} \left[ \log(F[x(t)]) \right]$$

Spectrum

Log spectrum

Cepstrum

The diagram illustrates the mathematical process of computing the cepstrum. It starts with a 'Time-domain signal'  $x(t)$ , represented by a pink box. This signal is transformed into a 'Spectrum', shown as a blue box containing  $F[x(t)]$ . The spectrum is then converted into a 'Log spectrum', shown as an orange box containing  $\log(F[x(t)])$ . Finally, the inverse Fourier transform  $F^{-1}$  is applied to the log spectrum to produce the 'Cepstrum', represented by a green box.

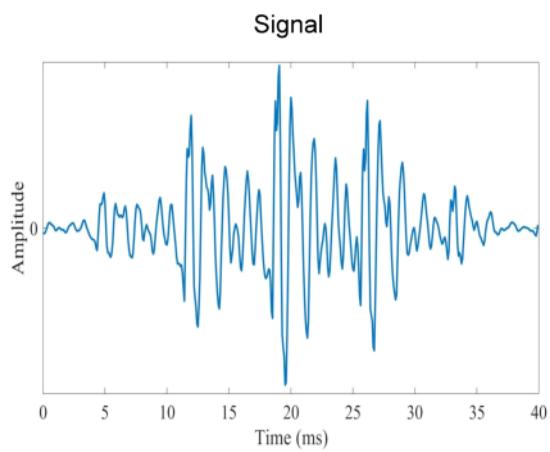


Spectrum  
of  
a spectrum

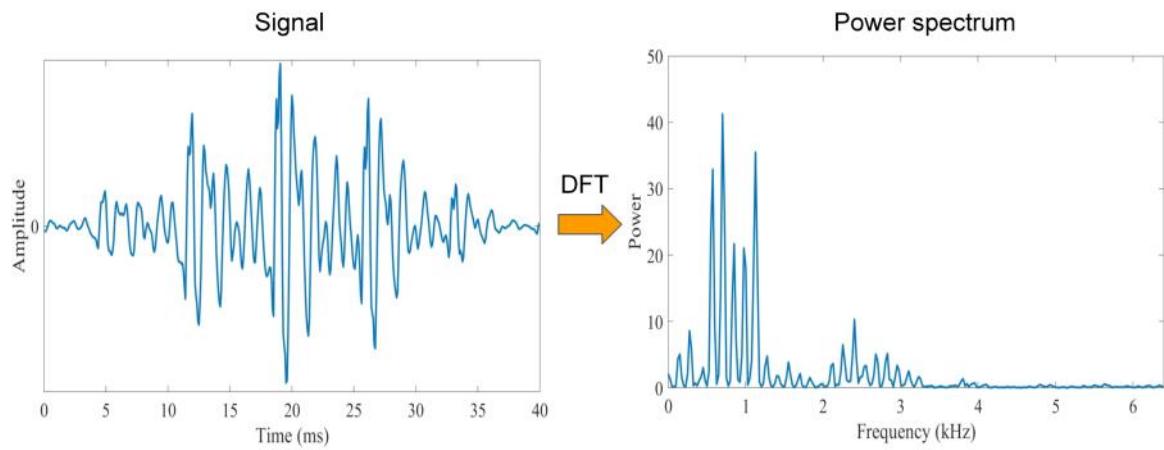
Cepstrum

## Visualising the cepstrum

---

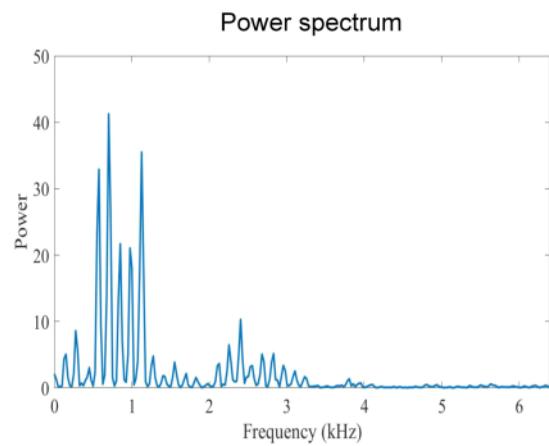


## Visualising the cepstrum

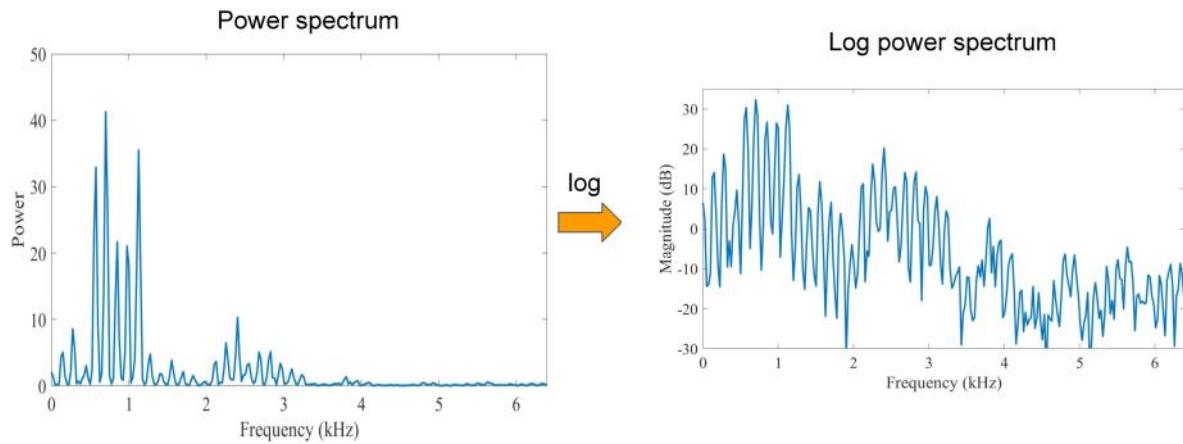


## Visualising the cepstrum

---

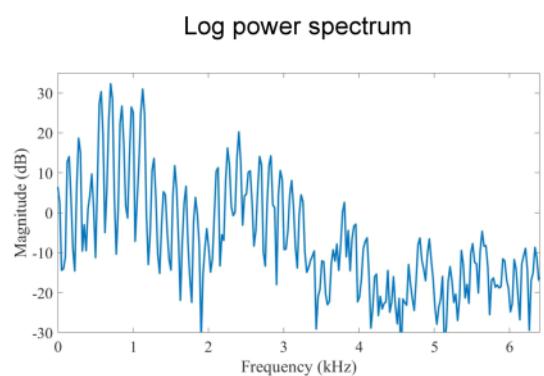


## Visualising the cepstrum

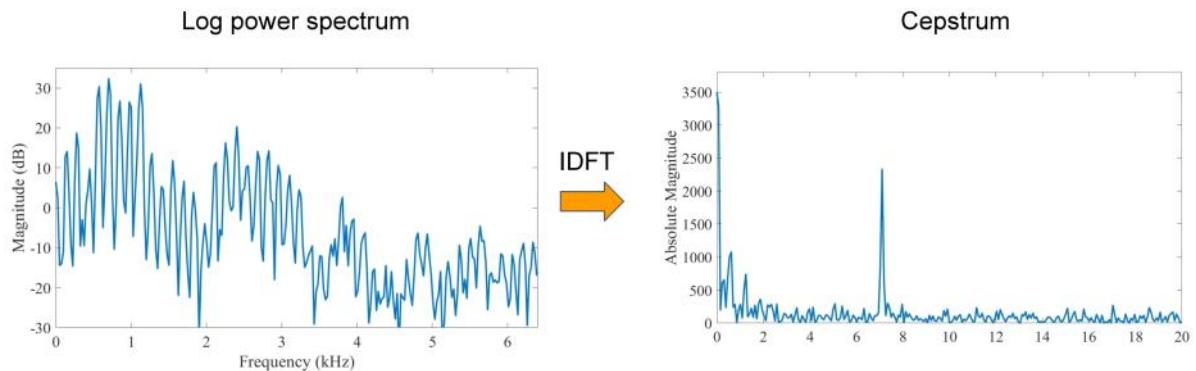


## Visualising the cepstrum

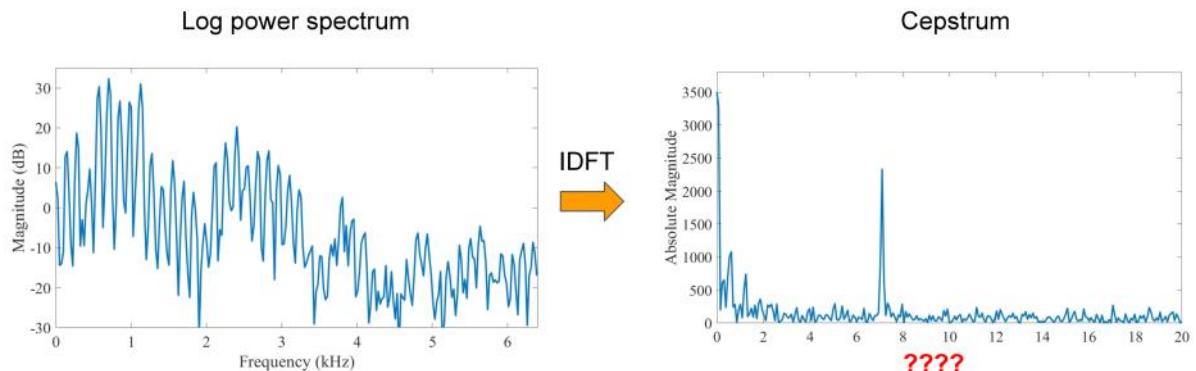
---



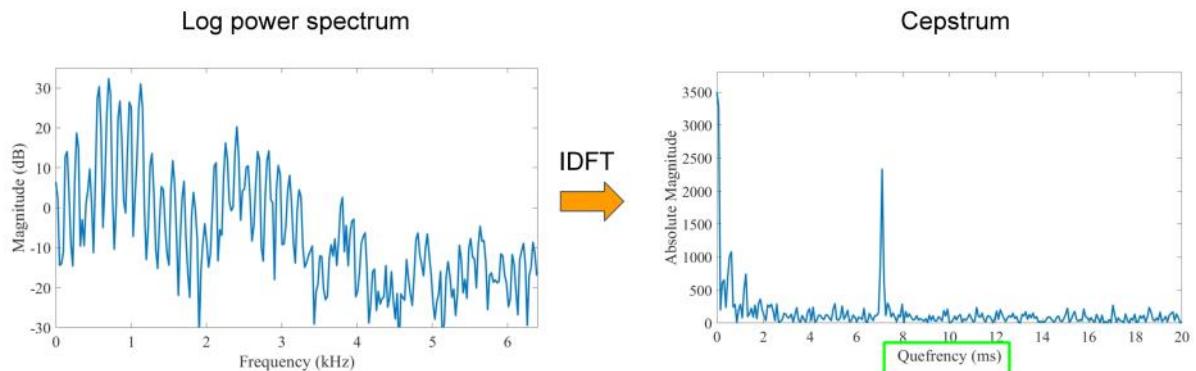
## Visualising the cepstrum



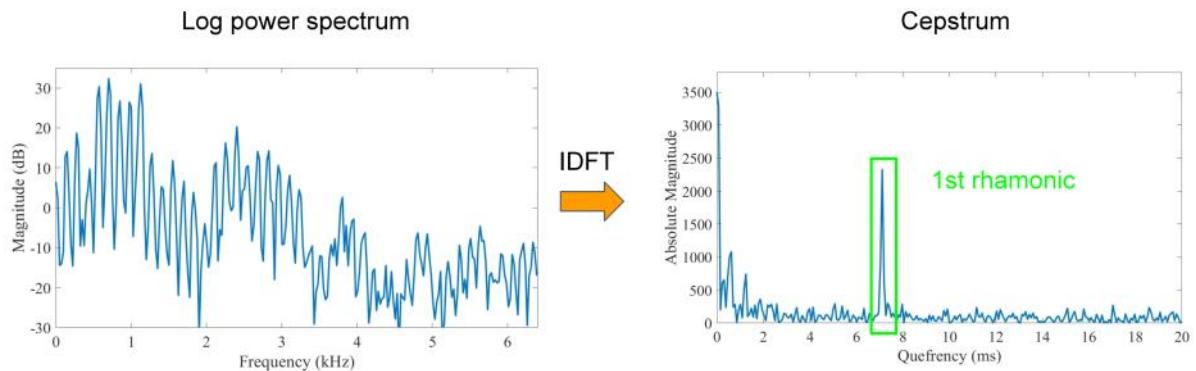
## Visualising the cepstrum



## Visualising the cepstrum

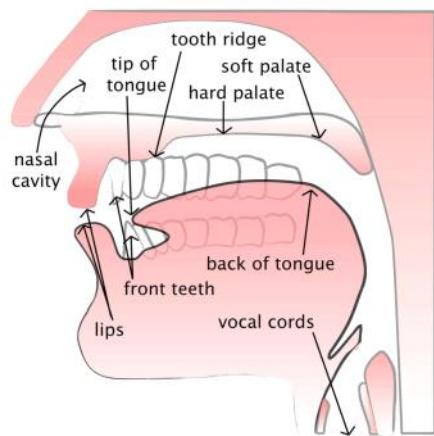


## Visualising the cepstrum



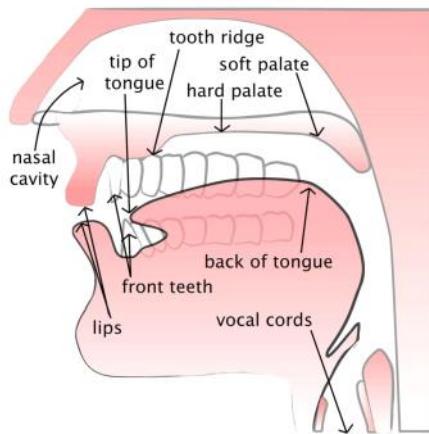
## The vocal tract

---



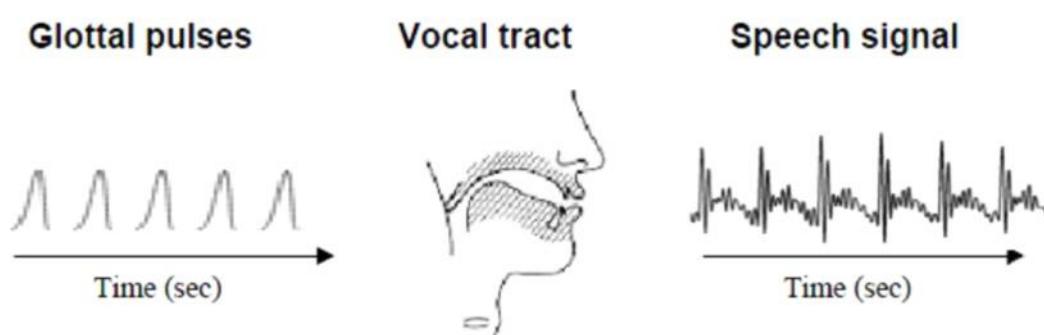
## The vocal tract

---



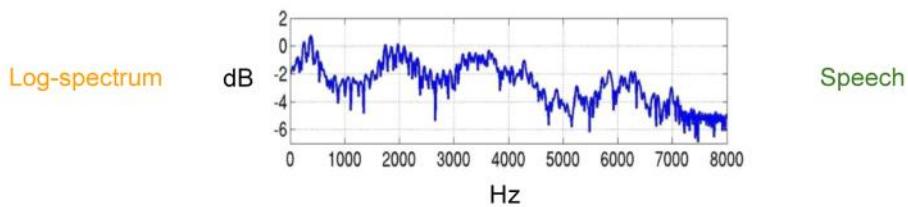
Vocal tract acts as a filter

## Speech generation

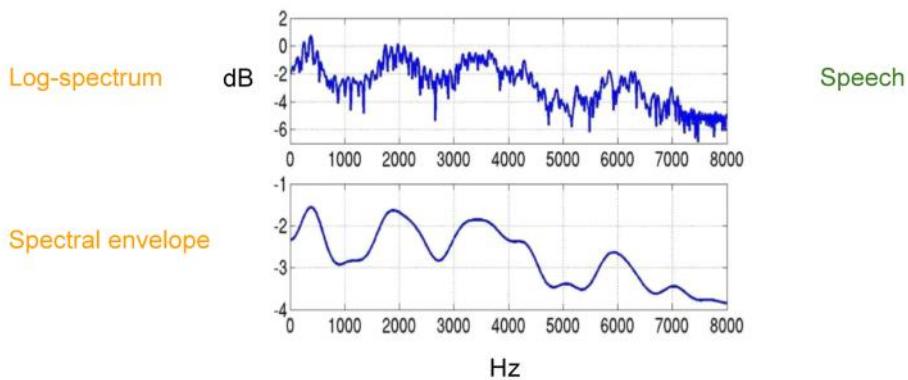


## Understanding the cepstrum

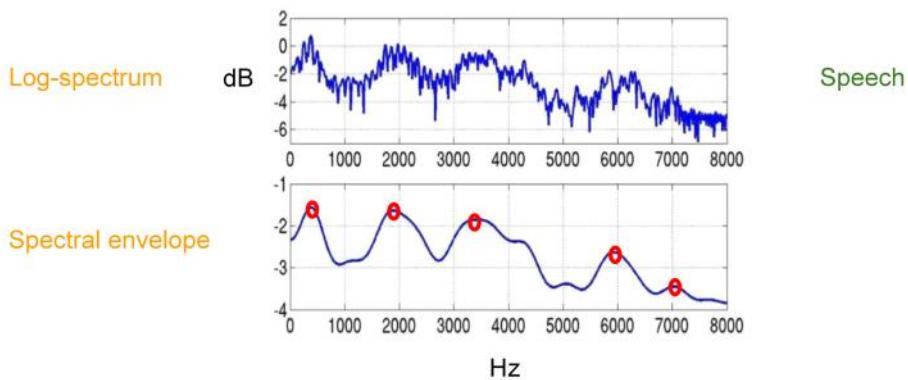
---



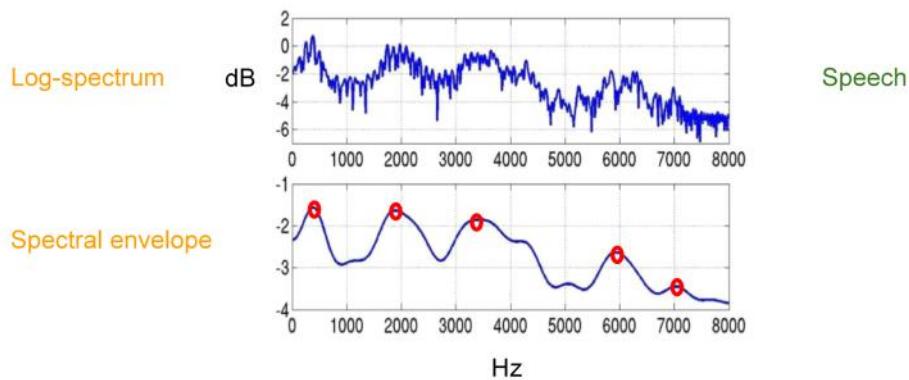
## Understanding the cepstrum



## Understanding the cepstrum

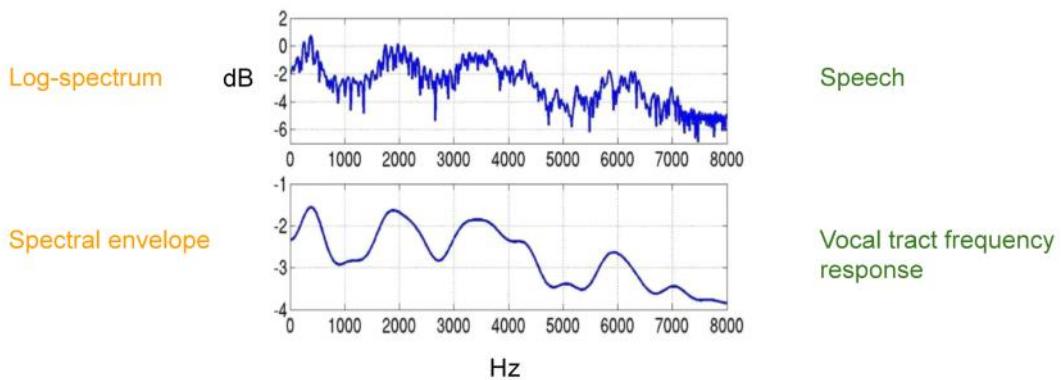


## Understanding the cepstrum

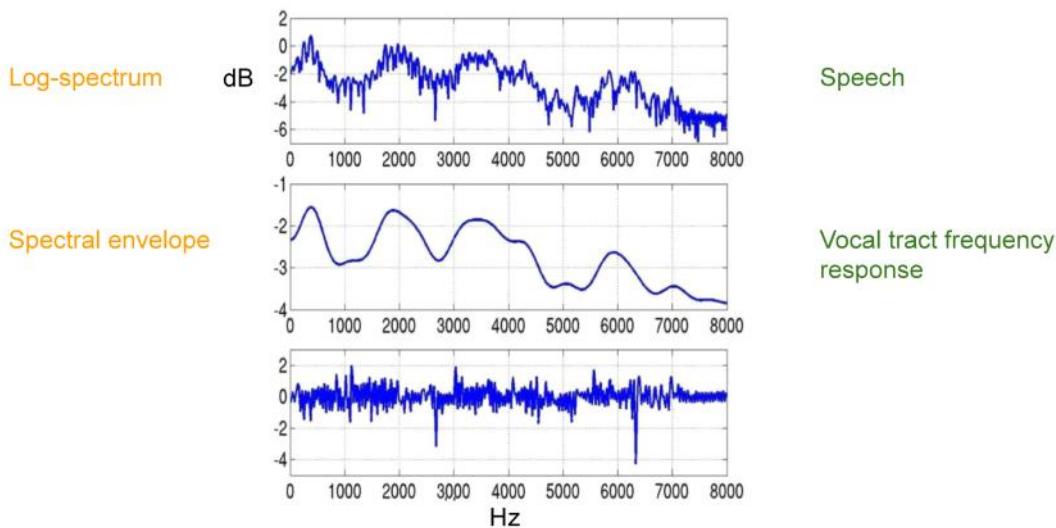


Formants = Carry identity of sound

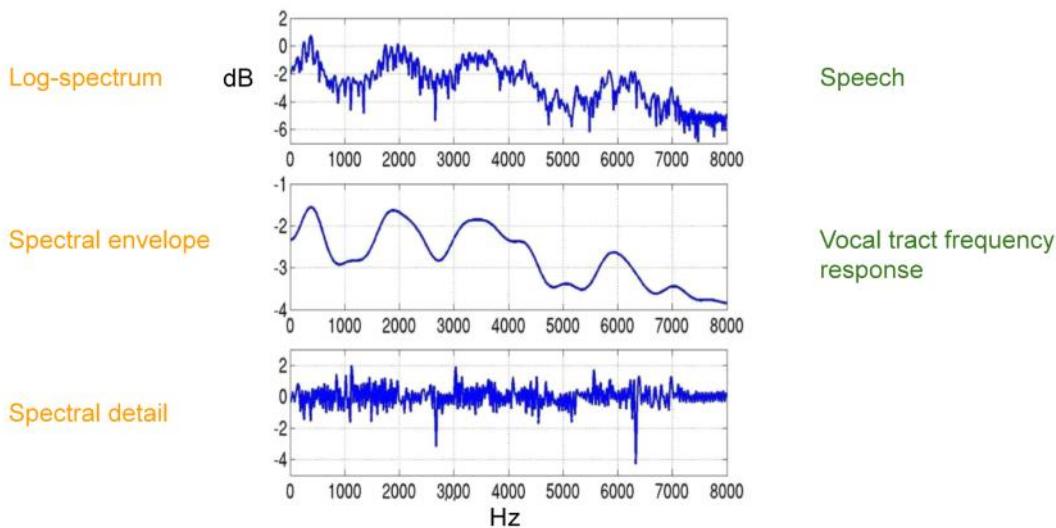
## Understanding the cepstrum



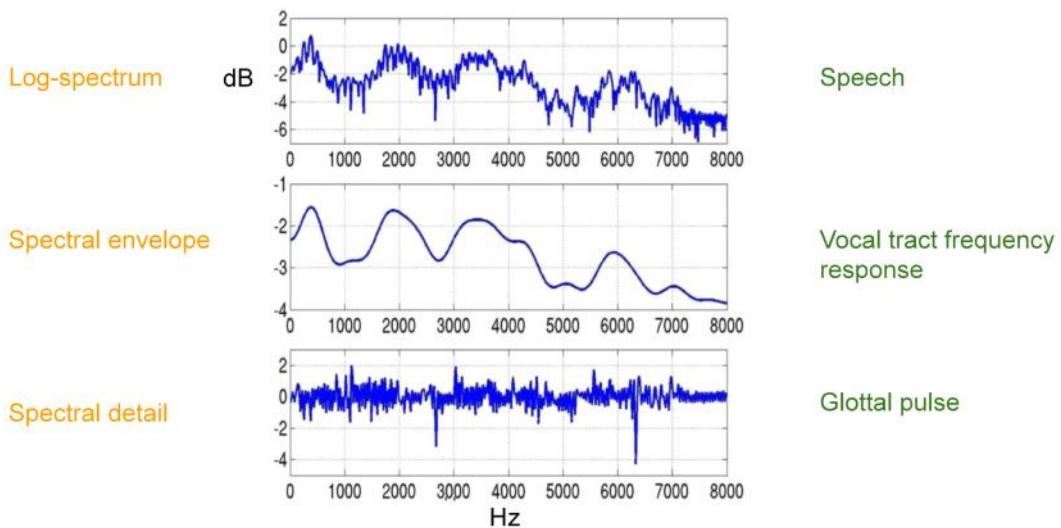
## Understanding the cepstrum



## Understanding the cepstrum



# Understanding the cepstrum



Speech

=

Convolution of vocal tract  
frequency response with  
glottal pulse

## Formalising speech

$$x(t) = e(t) \cdot h(t)$$

## Formalising speech

$$x(t) = e(t) \cdot h(t)$$

$$X(t) = E(t) \cdot H(t)$$

## Formalising speech

$$X(t) = E(t) \cdot H(t)$$

## Formalising speech

$$X(t) = E(t) \cdot H(t)$$
$$\downarrow$$
$$\log(X(t)) = \log(E(t) \cdot H(t))$$

## Formalising speech

$$X(t) = E(t) \cdot H(t)$$



$$\log(X(t)) = \log(E(t) \cdot H(t))$$



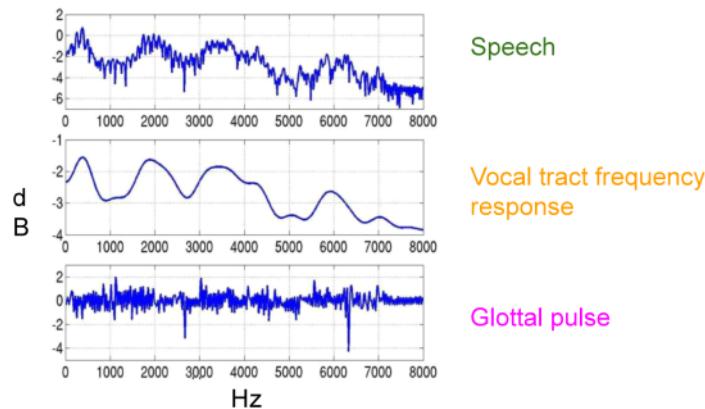
$$\log(X(t)) = \log(E(t)) + \log(H(t))$$

## Formalising speech

$$\log(X(t)) = \log(E(t)) + \log(H(t))$$

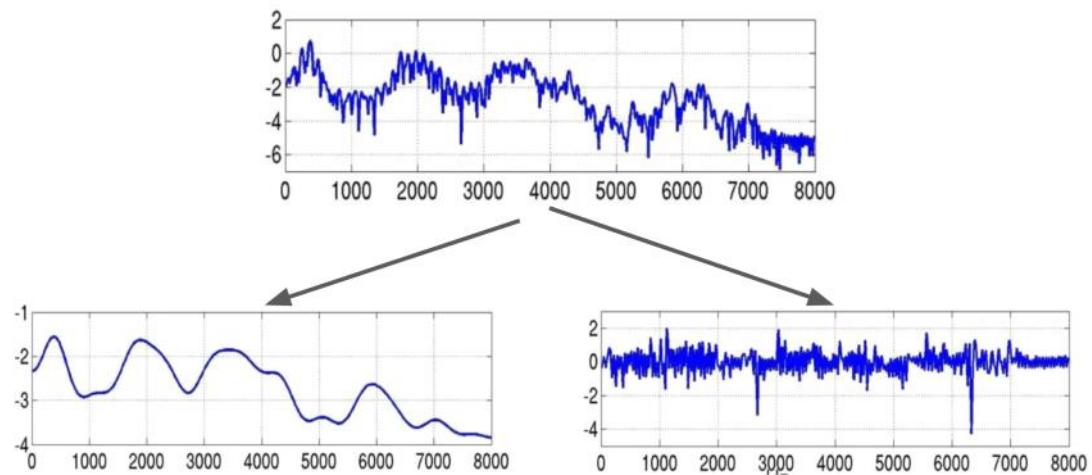
## Formalising speech

$$\log(X(t)) = \log(E(t)) + \log(H(t))$$



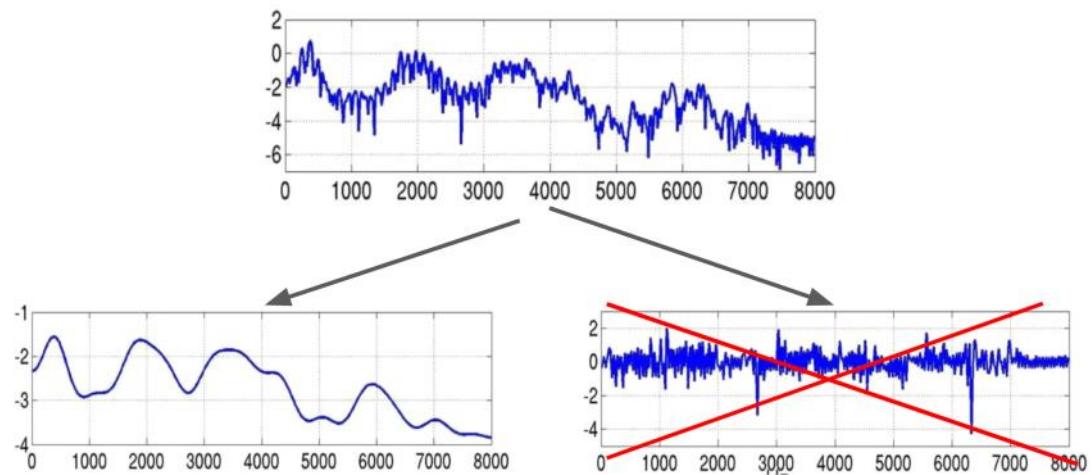
## The goal: Separating components

---



## The goal: Separating components

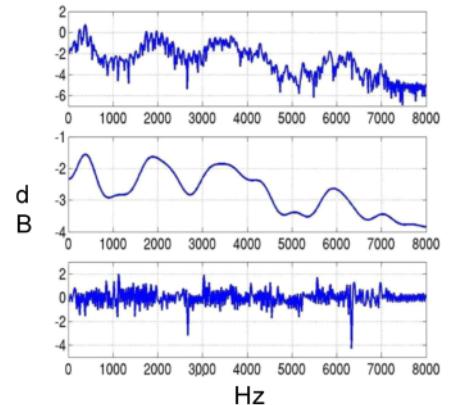
---



## Separating components

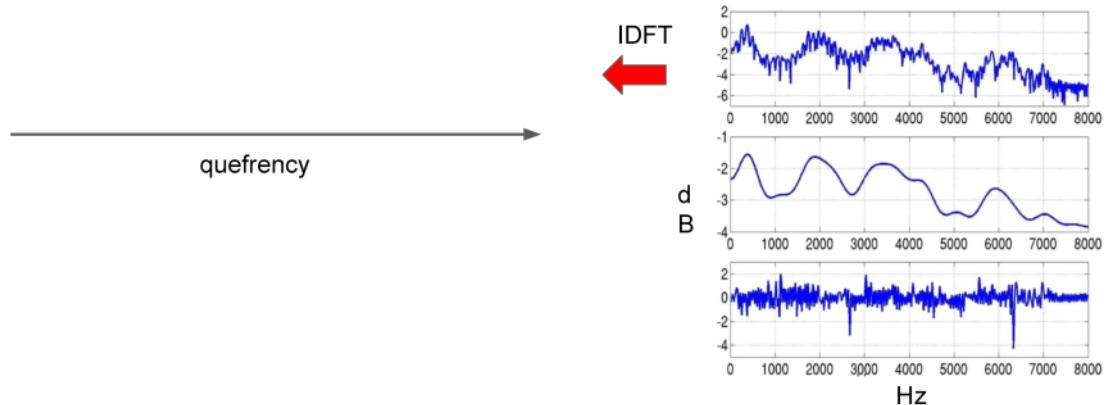
---

$$\log(X(t)) = \log(E(t)) + \log(H(t))$$



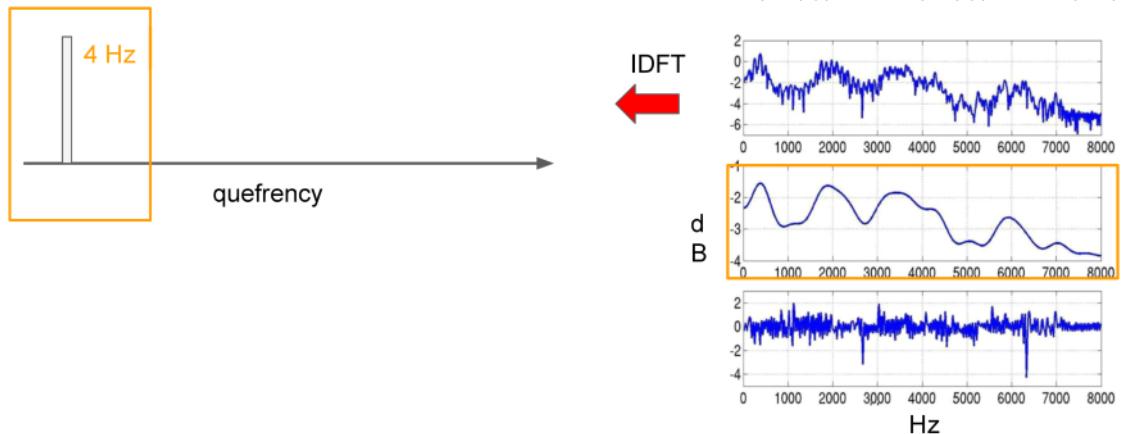
## Separating components

$$\log(X(t)) = \log(E(t)) + \log(H(t))$$

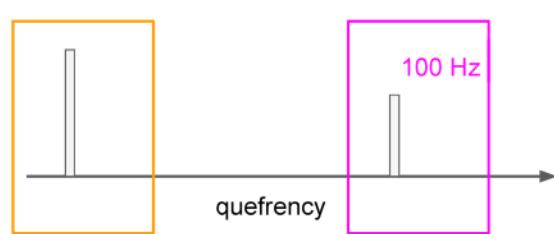


## Separating components

$$\log(X(t)) = \log(E(t)) + \log(H(t))$$

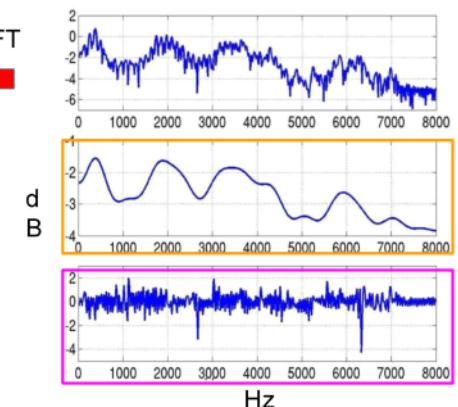


## Separating components

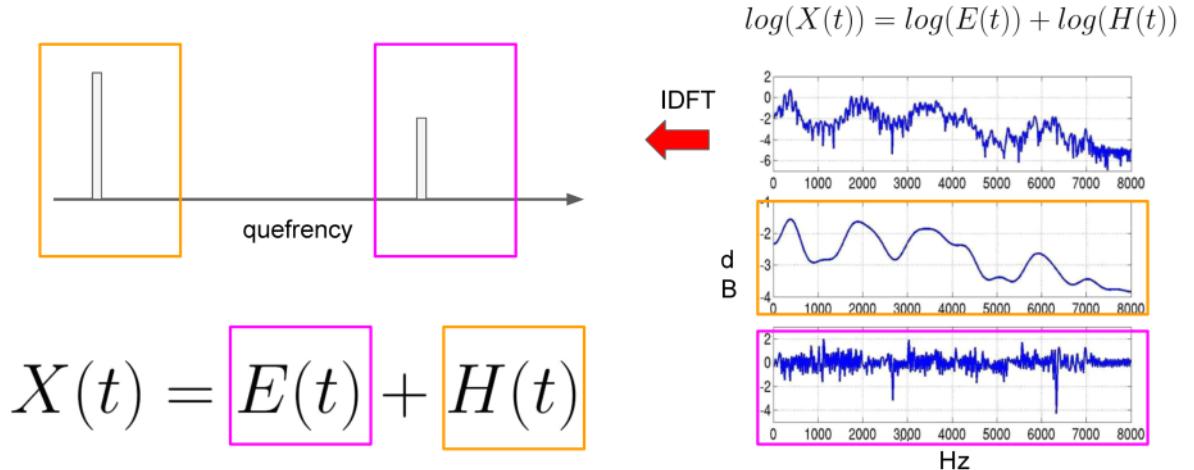


IDFT

$$\log(X(t)) = \log(E(t)) + \log(H(t))$$



## Separating components

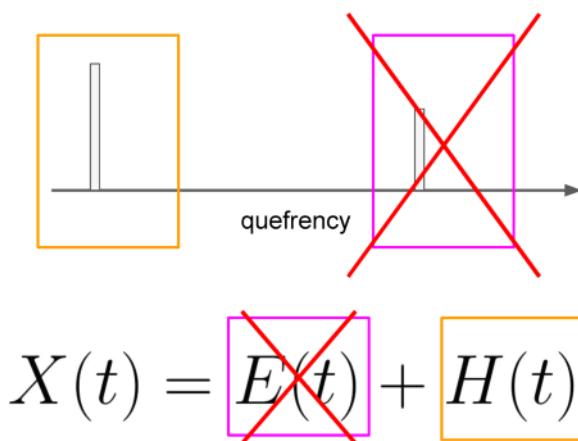




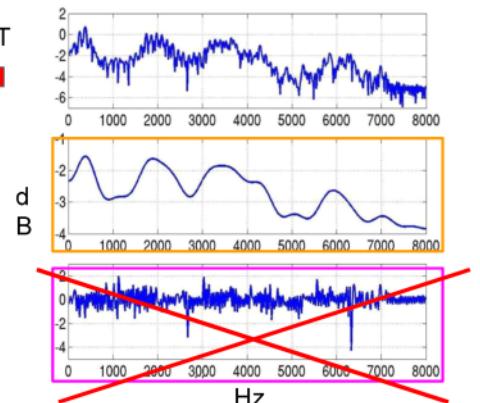
imgflip.com

JANE-CLARK-Tumblr

## Separating components



$$\log(X(t)) = \log(E(t)) + \log(H(t))$$



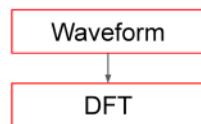
## Computing Mel-Frequency Cepstral Coefficients

---

Waveform

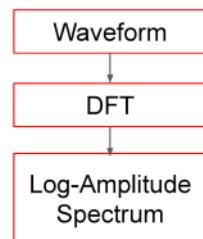
## Computing Mel-Frequency Cepstral Coefficients

---



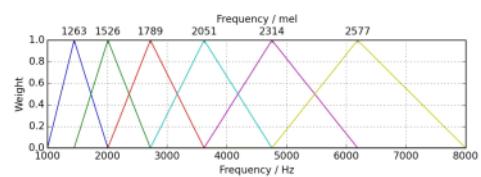
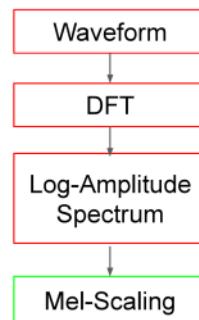
## Computing Mel-Frequency Cepstral Coefficients

---



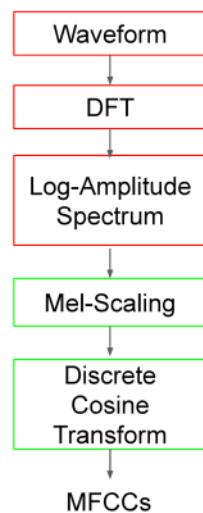
## Computing Mel-Frequency Cepstral Coefficients

---



## Computing Mel-Frequency Cepstral Coefficients

---



## Why Discrete Cosine Transform?

---

## Why Discrete Cosine Transform?

---

- Simplified version of Fourier Transform

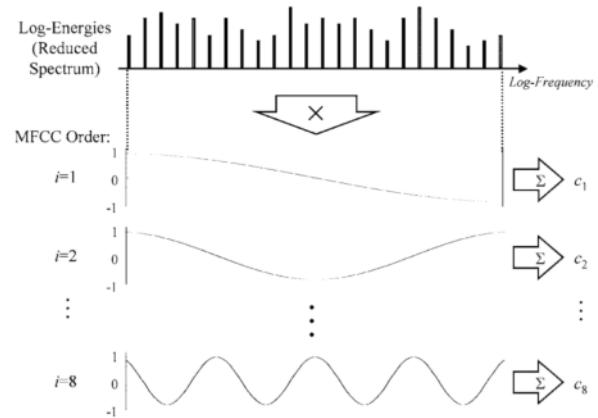
## Why Discrete Cosine Transform?

---

- Simplified version of Fourier Transform
- Get real-valued coefficient

## Why Discrete Cosine Transform?

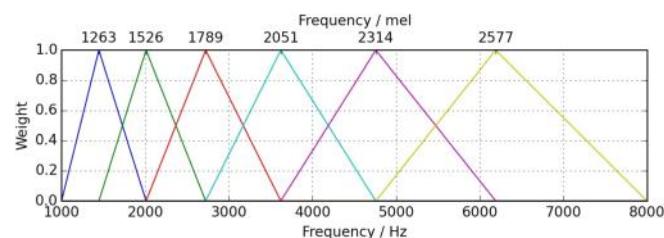
- Simplified version of Fourier Transform
- Get real-valued coefficient



## Why Discrete Cosine Transform?

---

- Simplified version of Fourier Transform
- Get real-valued coefficient
- Decorrelate energy in different mel bands



## Why Discrete Cosine Transform?

---

- Simplified version of Fourier Transform
- Get real-valued coefficient
- Decorrelate energy in different mel bands
- Reduce # dimensions to represent spectrum

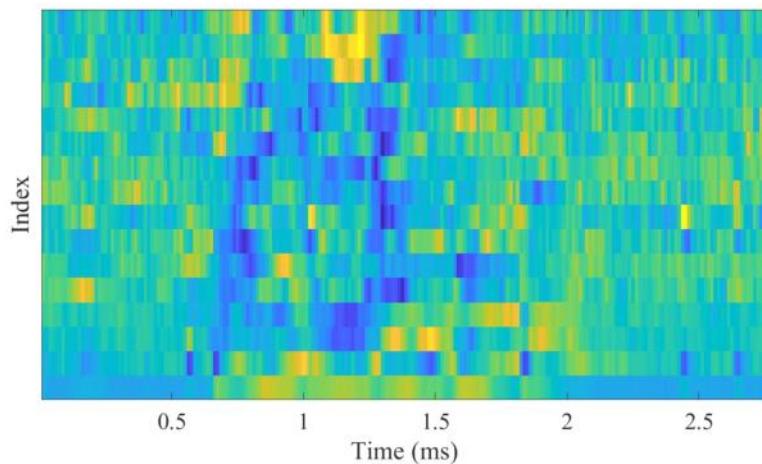
## How many coefficients?

---

- Traditionally: first 12 - 13 coefficients
- First coefficients keep most information (e.g., formants, spectral envelope)
- Use  $\Delta$  and  $\Delta\Delta$  MFCCs
- Total 39 coefficients per frame

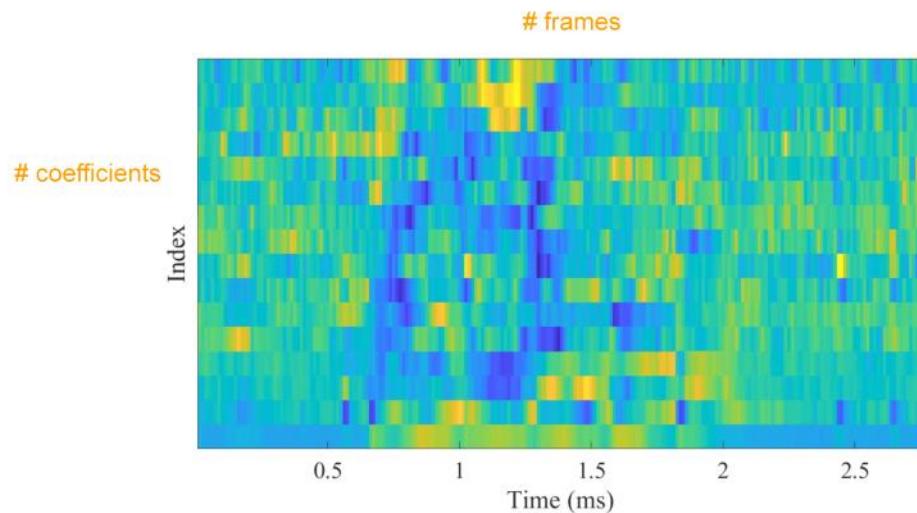
## Visualising MFCCs

---



## Visualising MFCCs

---



## MFCCs advantages

---

- Describe the “large” structures of the spectrum
- Ignore fine spectral structures
- Work well in speech and music processing

## MFCCs disadvantages

---

- Not robust to noise
- Extensive knowledge engineering
- Not efficient for synthesis

## MFCCs applications

---

- Speech processing
  - Speech recognition
  - Speaker recognition
  - ...
- Music processing
  - Music genre classification
  - Mood classification
  - Automatic tagging
  - ...

## What's up next?

---

- Extract MFCCs with Python and Librosa
- Visualise MFCCs

## 20 Code\_Extracting MFCCs

04 January 2025 23:03



Code\_Extracting MFCCs

```
In [1]: import librosa
import librosa.display
import IPython.display as ipd
import matplotlib.pyplot as plt
import numpy as np
```

### Loading audio files with Librosa

```
In [2]: audio_file = "audio/debussy.wav"
```

```
In [3]: ipd.Audio(audio_file)
```

```
Out[3]:
```

```
In [4]: # Load audio files with librosa
signal, sr = librosa.load(audio_file)
```

### Extracting MFCCs

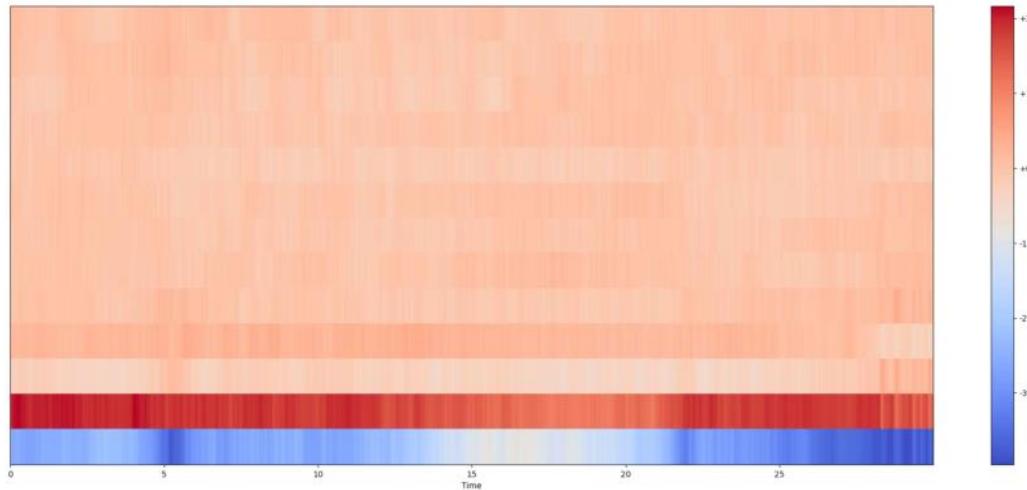
```
In [5]: mfccs = librosa.feature.mfcc(y=signal, n_mfcc=13, sr=sr)
```

```
In [6]: mfccs.shape
```

```
Out[6]: (13, 1292)
```

### Visualising MFCCs

```
In [7]: plt.figure(figsize=(25, 10))
librosa.display.specshow(mfccs,
                        x_axis="time",
                        sr=sr)
plt.colorbar(format="%+2.f")
plt.show()
```



### Computing first / second MFCCs derivatives

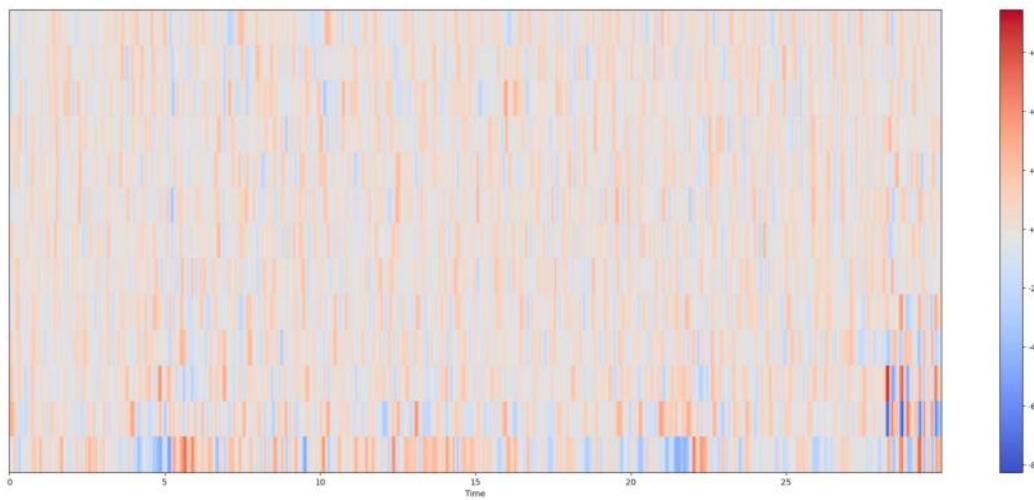
```
In [8]: delta_mfccs = librosa.feature.delta(mfccs)
```

```
In [9]: delta2_mfccs = librosa.feature.delta(mfccs, order=2)
```

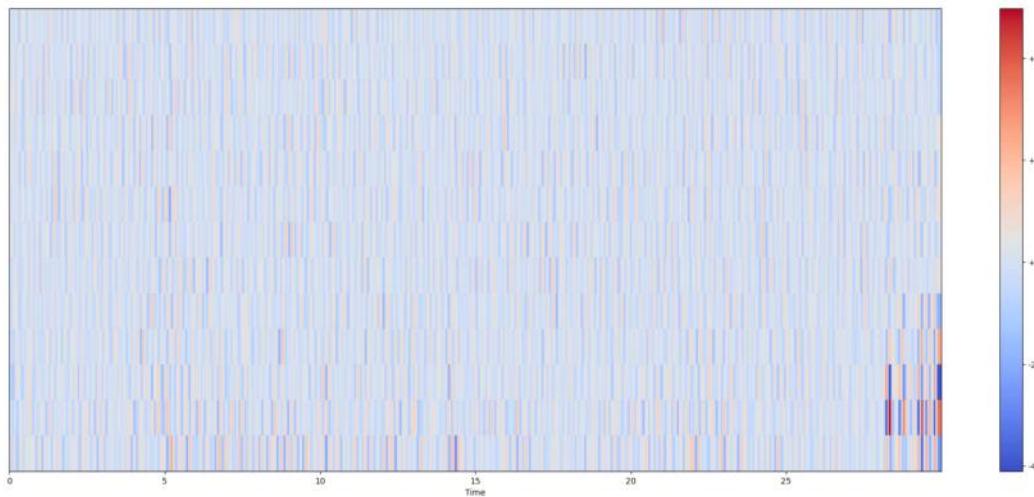
```
In [10]: delta_mfccs.shape
```

```
Out[10]: (13, 1292)
```

```
In [11]: plt.figure(figsize=(25, 10))
librosa.display.specshow(delta_mfccs,
                        x_axis="time",
                        sr=sr)
plt.colorbar(format="%+2.f")
plt.show()
```



```
In [12]: plt.figure(figsize=(25, 10))
librosa.display.specshow(delta2_mfccs,
    x_axis="time",
    sr=sr)
plt.colorbar(format="%+2.f")
plt.show()
```



```
In [13]: mfccs_features = np.concatenate((mfccs, delta_mfccs, delta2_mfccs))
```

```
In [14]: mfccs_features.shape
```

```
Out[14]: (39, 1292)
```

```
In [ ]:
```

# Frequency-domain audio features

Valerio Velardo

Join the community!



[thesoundofai.slack.com](https://thesoundofai.slack.com)

## Previously...

---

- Mel-Frequency Cepstral Coefficients

## Frequency-domain features

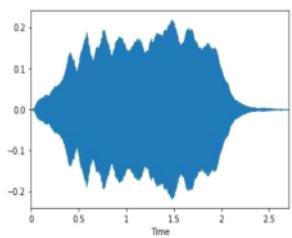
---

- Band energy ratio (BER)
- Spectral centroid (SC)
- Bandwidth (BW)
- ...

## Extracting frequency-domain features

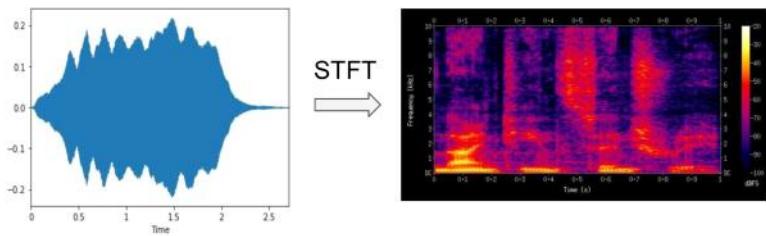
## Extracting frequency-domain features

---



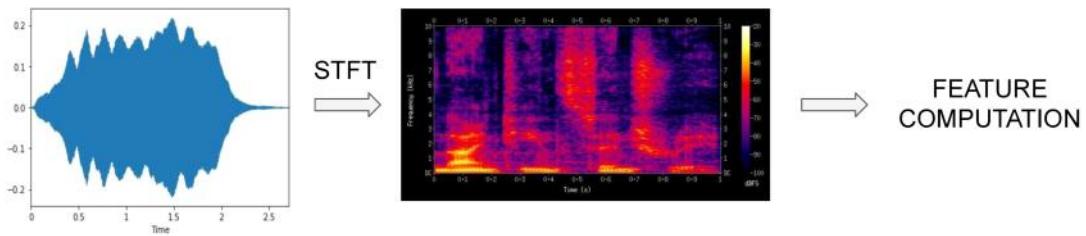
## Extracting frequency-domain features

---



## Extracting frequency-domain features

---



## Math conventions

---

- $m_t(n)$  -> Magnitude of signal at frequency bin  $n$  and frame  $t$

## Math conventions

---

- $m_t(n)$  -> Magnitude of signal at frequency bin  $n$  and frame  $t$
- $N$  -> # frequency bins

## Band energy ratio

---

- Comparison of energy in the lower/higher frequency bands
- Measure of how dominant low frequencies are

## Band energy ratio

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^N m_t(n)^2}$$

## Band energy ratio

---

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^N m_t(n)^2}$$

Power at  $t, n$

## Band energy ratio

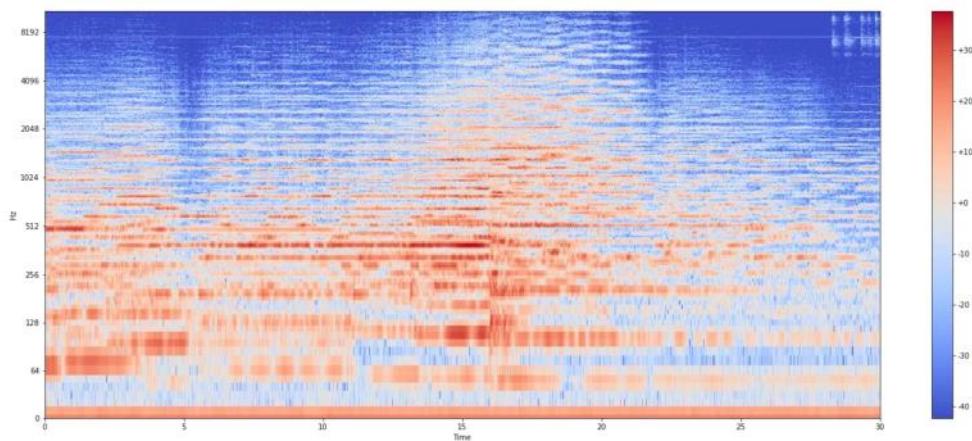
---

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^N m_t(n)^2}$$

Split frequency      Power at  $t, n$

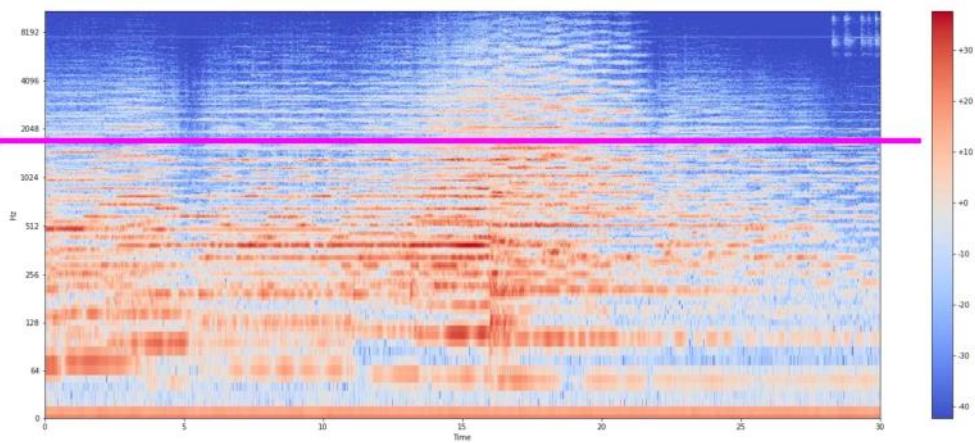
## Band energy ratio

---



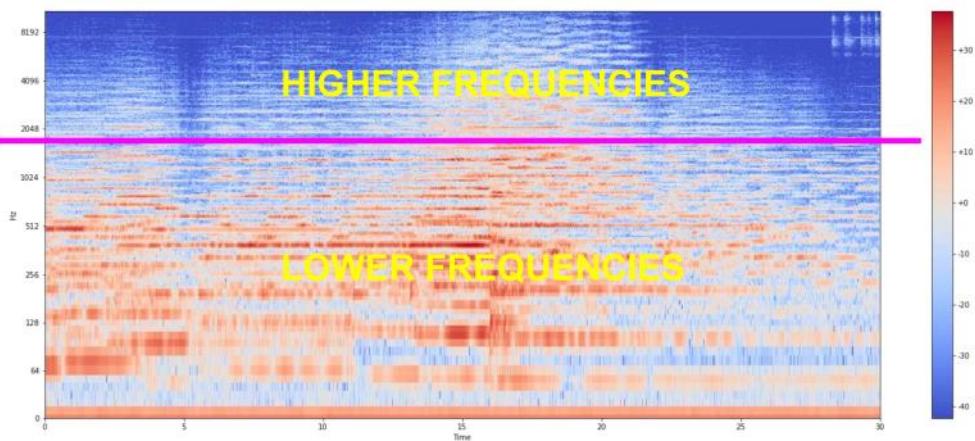
## Band energy ratio

---



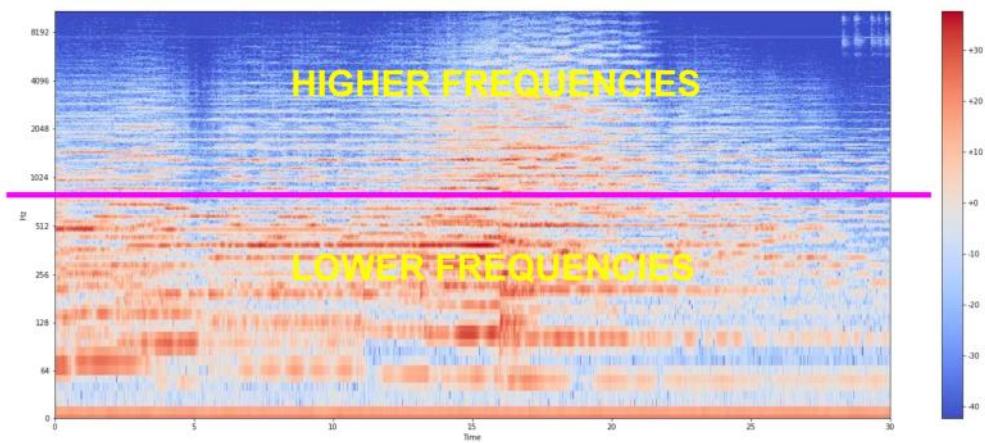
## Band energy ratio

---



## Band energy ratio

---



## Band energy ratio

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^N m_t(n)^2}$$

## Band energy ratio

---

Power in the lower frequency bands

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^N m_t(n)^2}$$

## Band energy ratio

---

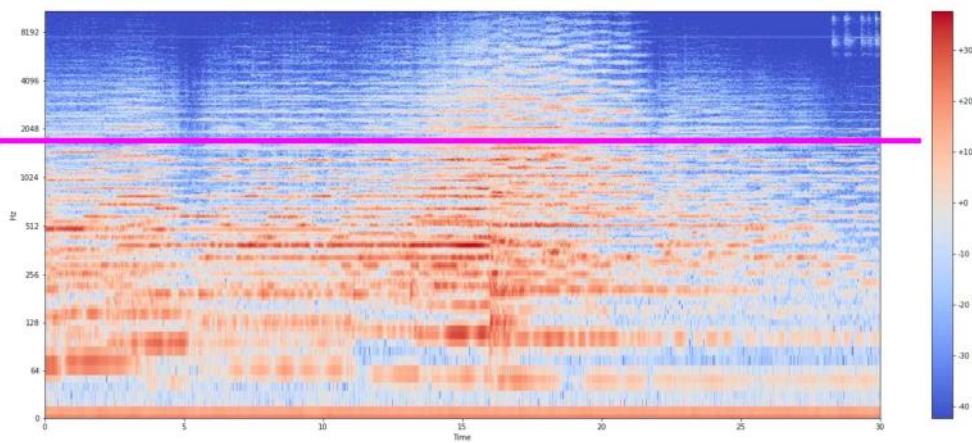
Power in the lower frequency bands

$$BER_t = \frac{\sum_{n=1}^{F-1} m_t(n)^2}{\sum_{n=F}^N m_t(n)^2}$$

Power in the higher frequency bands

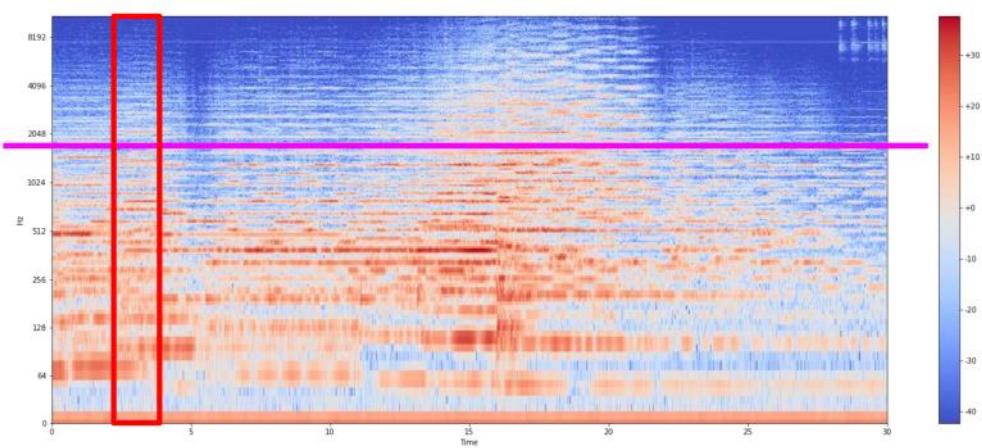
## Band energy ratio

---



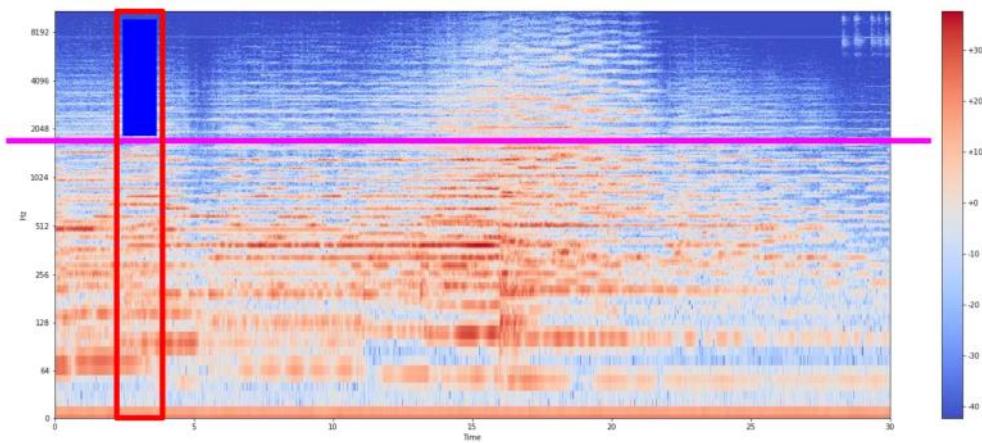
## Band energy ratio

---



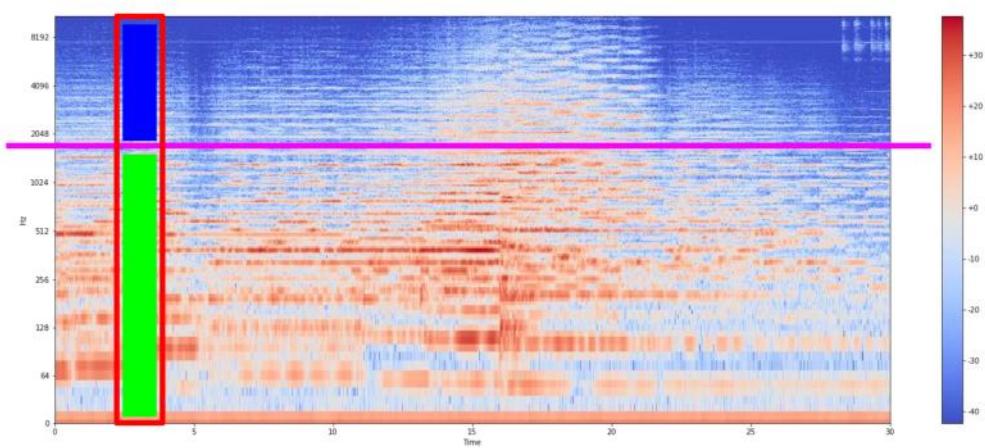
## Band energy ratio

---



## Band energy ratio

---



## Band energy ratio applications

---

- Music / speech discrimination
- Music classification (e.g., music genre classification)

## Spectral centroid

---

- Centre of gravity of magnitude spectrum
- Frequency band where most of the energy is concentrated
- Measure of “brightness” of sound

## Spectral centroid

---

- Weighted mean of the frequencies

## Spectral centroid

---

- Weighted mean of the frequencies

$$SC_t = \frac{\sum_{n=1}^N m_t(n) \cdot n}{\sum_{n=1}^N m_t(n)}$$

## Spectral centroid

---

- Weighted mean of the frequencies

$$SC_t = \frac{\sum_{n=1}^N m_t(n) \cdot n}{\sum_{n=1}^N m_t(n)}$$

## Spectral centroid

---

- Weighted mean of the frequencies

$$SC_t = \frac{\sum_{n=1}^N m_t(n) \cdot n}{\sum_{n=1}^N m_t(n)}$$

## Spectral centroid

---

- Weighted mean of the frequencies

$$SC_t = \frac{\sum_{n=1}^N m_t(n) \cdot n}{\sum_{n=1}^N m_t(n)}$$

Sum of weights

## Spectral centroid applications

---

- Audio classification
- Music classification

## Bandwidth

---

- Derived from spectral centroid
- Spectral range around the centroid
- Variance from the spectral centroid
- Describe perceived timbre

## Bandwidth

---

- Weighted mean of the distances of frequency bands from SC

## Bandwidth

---

- Weighted mean of the distances of frequency bands from SC

$$BW_t = \frac{\sum_{n=1}^N |n - SC_t| \cdot m_t(n)}{\sum_{n=1}^N m_t(n)}$$

## Bandwidth

---

- Weighted mean of the distances of frequency bands from SC

$$BW_t = \frac{\sum_{n=1}^N |n - SC_t| \cdot \boxed{m_t(n)} \text{Weight for } n}{\sum_{n=1}^N \boxed{m_t(n)}}$$

## Bandwidth

---

- Weighted mean of the distances of frequency bands from SC

$$BW_t = \frac{\sum_{n=1}^N |n - SC_t| \cdot m_t(n)}{\sum_{n=1}^N m_t(n)}$$

Distance of frequency band from  
spectral centroid      Weight for  $n$

$|n - SC_t|$        $m_t(n)$

## Bandwidth

---

- Weighted mean of the distances of frequency bands from SC

$$BW_t = \frac{\sum_{n=1}^N |n - SC_t| \cdot m_t(n)}{\sum_{n=1}^N m_t(n)}$$

Distance of frequency band from  
spectral centroid      Weight for  $n$   
 $|n - SC_t|$        $m_t(n)$

Sum of weights       $\sum_{n=1}^N m_t(n)$

## Bandwidth

---



Energy spread across  
frequency bands

## Bandwidth

---



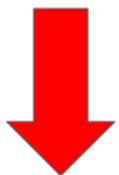
Energy spread across  
frequency bands



$BW_t$

## Bandwidth

---



Energy spread across  
frequency bands

## Bandwidth

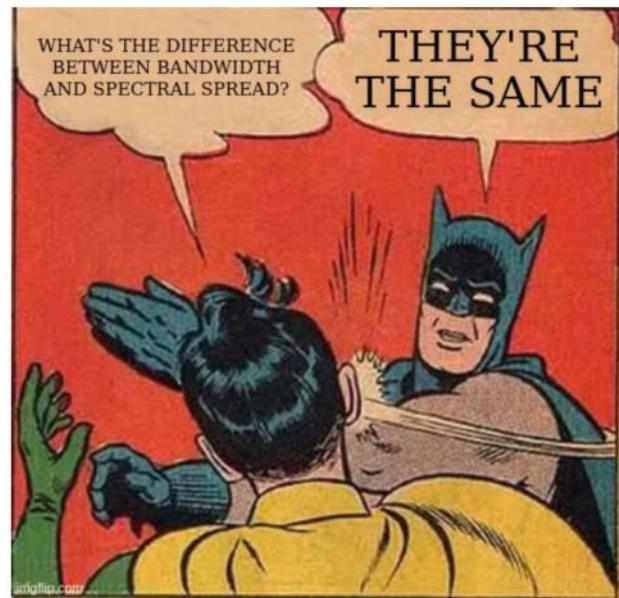
---



Energy spread across  
frequency bands



$BW_t$



## Bandwidth applications

---

- Music processing (e.g., music genre classification)

## What's up next?

---

- Implement band energy ratio in Python (almost!) from scratch
- Visualise BER for music in different genres

## 22 Code\_Implementing band energy ratio from scratch

04 January 2025 23:03



Code\_Implementing band energy ratio from scratch

```
In [1]: import math
import matplotlib.pyplot as plt
import numpy as np
import librosa
import librosa.display
import IPython.display as ipd
```

## Loading audio files

```
In [2]: debussy_file = "audio/debussy.wav"
redhot_file = "audio/redhot.wav"
duke_file = "audio/duke.wav"
```

```
In [3]: ipd.Audio(debussy_file)
```

```
Out[3]: 
```

```
In [4]: ipd.Audio(redhot_file)
```

```
Out[4]: 
```

```
In [5]: # Load audio files with librosa
debussy, sr = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)
```

## Extract spectrograms

```
In [6]: FRAME_SIZE = 2048
HOP_SIZE = 512
```

```
debussy_spec = librosa.stft(debussy, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)
redhot_spec = librosa.stft(redhot, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)
duke_spec = librosa.stft(duke, n_fft=FRAME_SIZE, hop_length=HOP_SIZE)
```

```
In [7]: debussy_spec.shape
```

```
Out[7]: (1025, 1292)
```

## Calculate Band Energy Ratio

```
In [8]: def calculate_split_frequency_bin(split_frequency, sample_rate, num_frequency_bins):
    """Infer the frequency bin associated to a given split frequency."""
    frequency_range = sample_rate / 2
    frequency_delta_per_bin = frequency_range / num_frequency_bins
    split_frequency_bin = math.floor(split_frequency / frequency_delta_per_bin)
    return int(split_frequency_bin)
```

```
In [9]: split_frequency_bin = calculate_split_frequency_bin(2000, 22050, 1025)
split_frequency_bin
```

```
Out[9]: 185
```

```
In [10]: def band_energy_ratio(spectrogram, split_frequency, sample_rate):
    """Calculate band energy ratio with a given split frequency."""
    split_frequency_bin = calculate_split_frequency_bin(split_frequency, sample_rate, len(spectrogram[0]))
```

```
band_energy_ratio = []
# calculate power spectrogram
power_spectrogram = np.abs(spectrogram) ** 2
power_spectrogram = power_spectrogram.T

# calculate BER value for each frame
for frame in power_spectrogram:
    sum_power_low_frequencies = frame[:split_frequency_bin].sum()
    sum_power_high_frequencies = frame[split_frequency_bin:].sum()
    band_energy_ratio_current_frame = sum_power_low_frequencies / sum_power_high_frequencies
    band_energy_ratio.append(band_energy_ratio_current_frame)
```

```
return np.array(band_energy_ratio)
```

```
In [11]: ber_debussy = band_energy_ratio(debussy_spec, 2000, sr)
ber_redhot = band_energy_ratio(redhot_spec, 2000, sr)
ber_duke = band_energy_ratio(duke_spec, 2000, sr)
```

```
In [12]: len(ber_debussy)
```

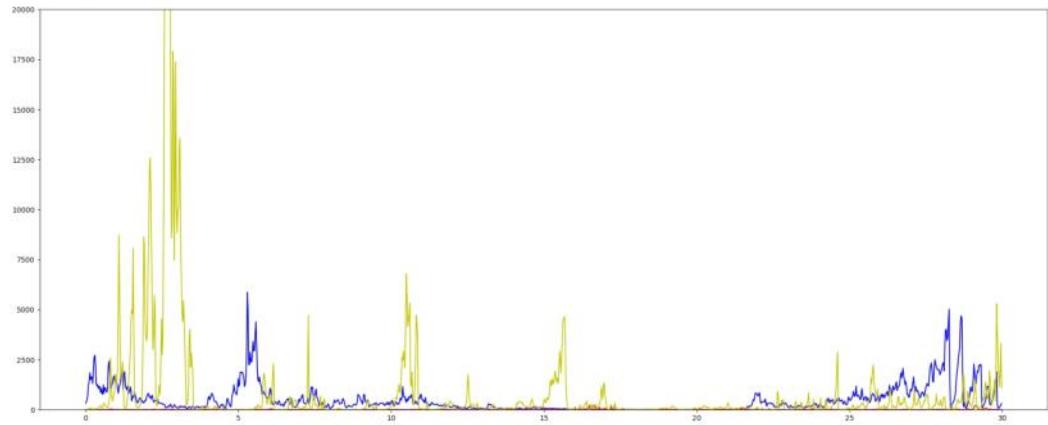
```
Out[12]: 1292
```

### Visualise Band Energy Ratio

```
In [13]: frames = range(len(ber_debussy))
t = librosa.frames_to_time(frames, hop_length=HOP_SIZE)

In [14]: plt.figure(figsize=(25, 10))

plt.plot(t, ber_debussy, color="b")
plt.plot(t, ber_redhot, color="r")
plt.plot(t, ber_duke, color="y")
plt.ylim(0, 20000)
plt.show()
```



## 23 Code\_Spectral centroid and bandwidth

04 January 2025 23:04



Code\_Spectral centroid and bandwidth

```
In [1]: import matplotlib.pyplot as plt
import numpy as np
import librosa
import IPython.display as ipd
```

### Loading audio files

```
In [2]: debussy_file = "audio/debussy.wav"
redhot_file = "audio/redhot.wav"
duke_file = "audio/duke.wav"
```

```
In [3]: ipd.Audio(debussy_file)
```

```
Out[3]:
```

```
In [4]: ipd.Audio(redhot_file)
```

```
Out[4]:
```

```
In [5]: ipd.Audio(duke_file)
```

```
Out[5]:
```

```
In [6]: # Load audio files with librosa
debussy, sr = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)
```

### Spectral centroid with Librosa

```
In [7]: FRAME_SIZE = 1024
HOP_LENGTH = 512
```

```
In [8]: sc_debussy = librosa.feature.spectral_centroid(y=debussy, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
sc_redhot = librosa.feature.spectral_centroid(y=redhot, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
sc_duke = librosa.feature.spectral_centroid(y=duke, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
```

```
In [9]: sc_debussy.shape
```

```
Out[9]: (1292,)
```

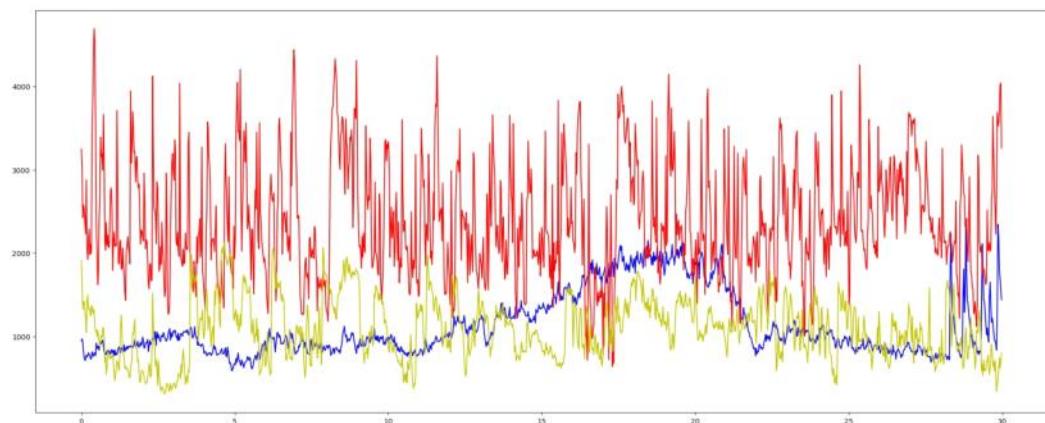
### Visualising spectral centroid

```
In [10]: frames = range(len(sc_debussy))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)
```

```
In [11]: len(t)
```

```
Out[11]: 1292
```

```
In [12]: plt.figure(figsize=(25,10))
plt.plot(t, sc_debussy, color='b')
plt.plot(t, sc_redhot, color='r')
plt.plot(t, sc_duke, color='y')
plt.show()
```



### Spectral bandwidth with Librosa

```
In [13]: ban_debussy = librosa.feature.spectral_bandwidth(y=debussy, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
ban_redhot = librosa.feature.spectral_bandwidth(y=redhot, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
ban_duke = librosa.feature.spectral_bandwidth(y=duke, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)[0]

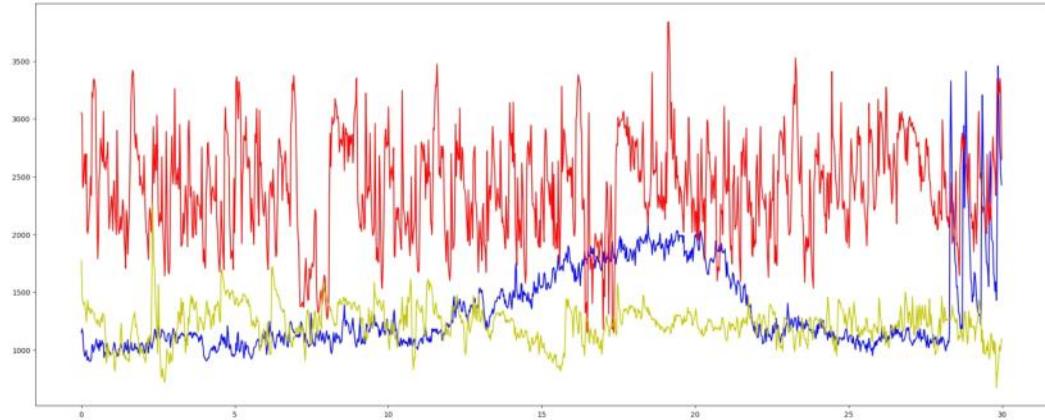
In [14]: ban_debussy.shape
Out[14]: (1292,)
```

### Visualising spectral bandwidth

```
In [15]: plt.figure(figsize=(25,10))

plt.plot(t, ban_debussy, color='b')
plt.plot(t, ban_redhot, color='r')
plt.plot(t, ban_duke, color='y')

plt.show()
```



```
In [ ]:
```