
Deep
Learning
School

DLS

Yermekova Assel

Skoltech ‘22
MLE in Noah Ark Lab, Huawei

Lecture 2.
Audio representation: Spectrograms

DLS

Content

Lecture: Spectrograms

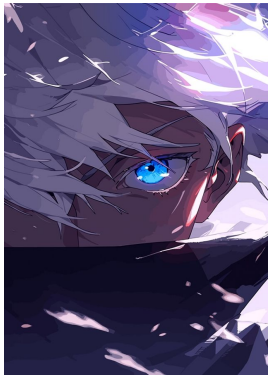
Content:

- problems in using waveforms
- Frequency spectrum
  - Fourier Series
  - Fourier Transform
  - Discrete Fourier Transform
- Spectrograms and STFT
- Mel-Scale
  - Mel filter bank
  - Mel-Spectrogram
- Lecture summary

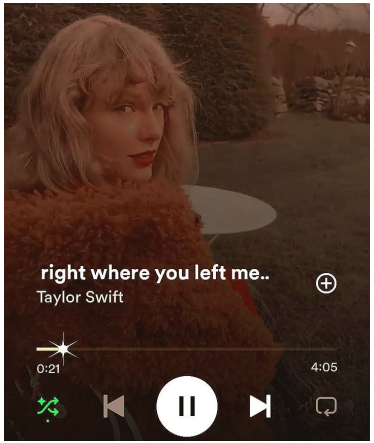
Why not use only waveforms?

Why not use only waveforms?

High Dimensionality

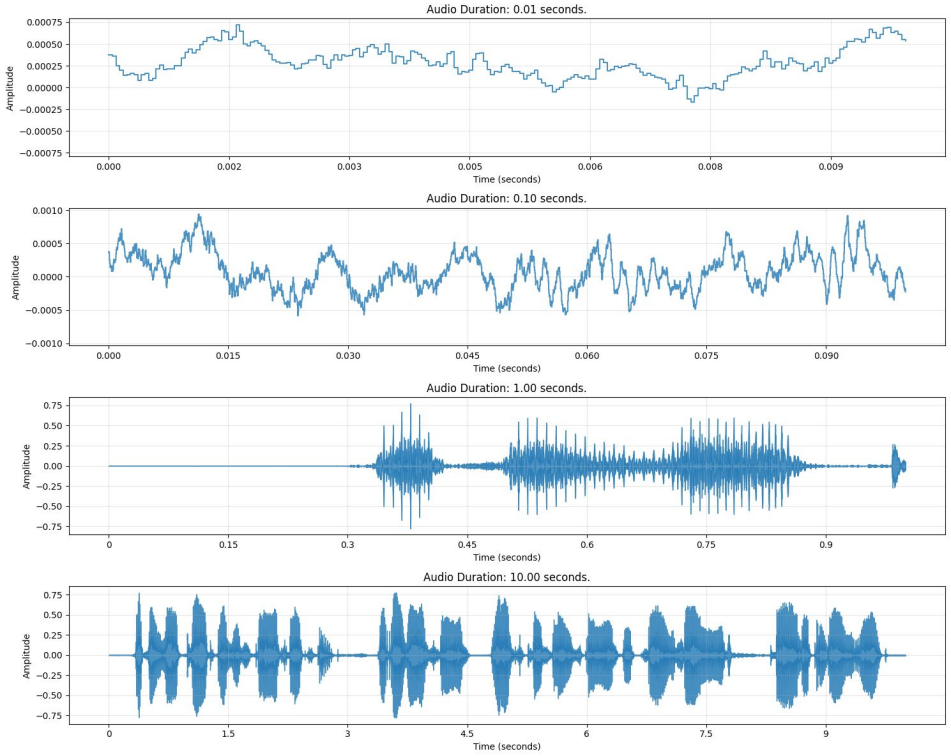


Full HD (1080p): 1920 × 1080 x 3 = 6220800 = **6M**



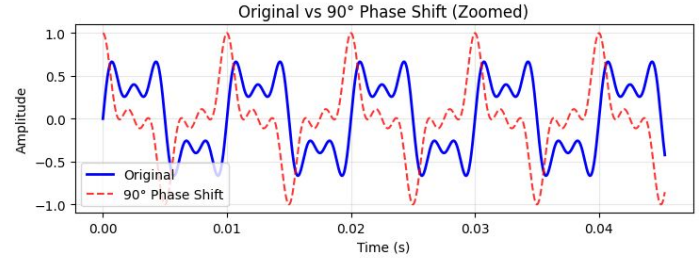
Song: 4.05 minutes = 245 secs = 245 * 16 kHz * 2 channels = 7680000 = **7.6M**

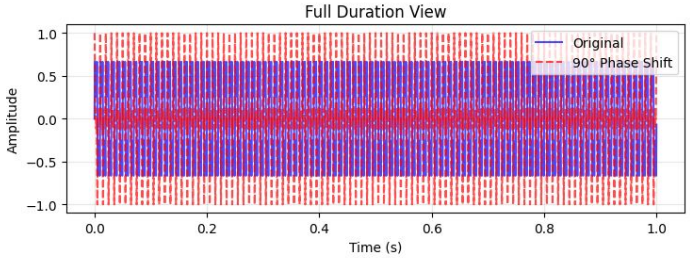
Low Information Density

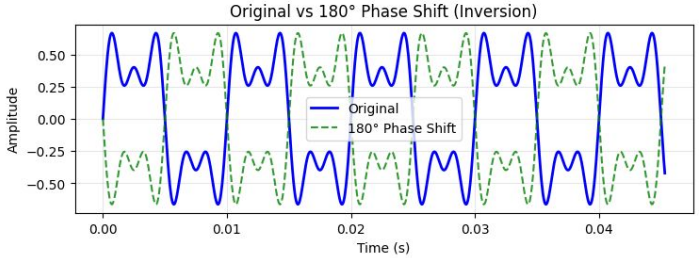


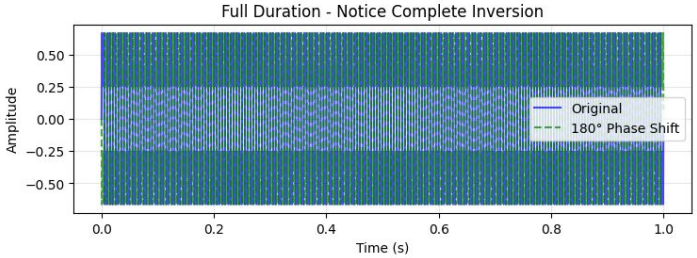
Why not use only waveforms?

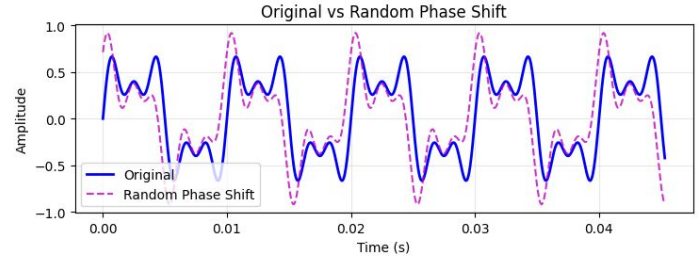
Phase Sensitivity

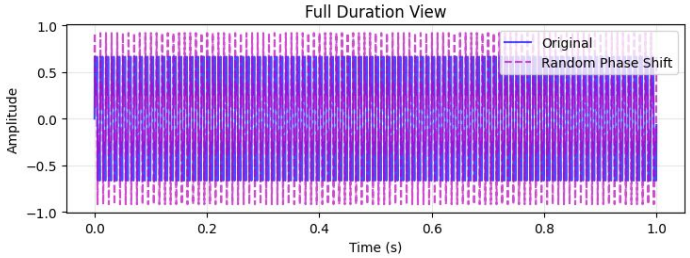








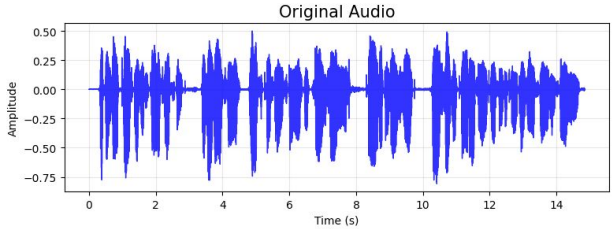


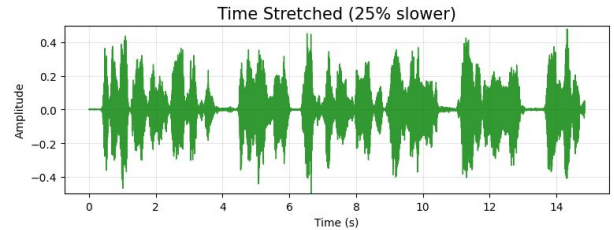


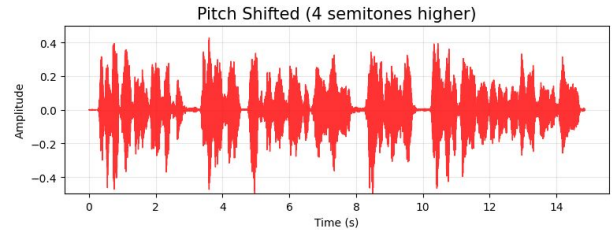
For model original and shifted audio signal will be considered as totally different signals.

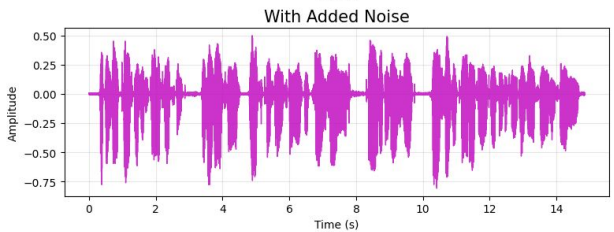
Why not use only waveforms?

Looks almost the same!









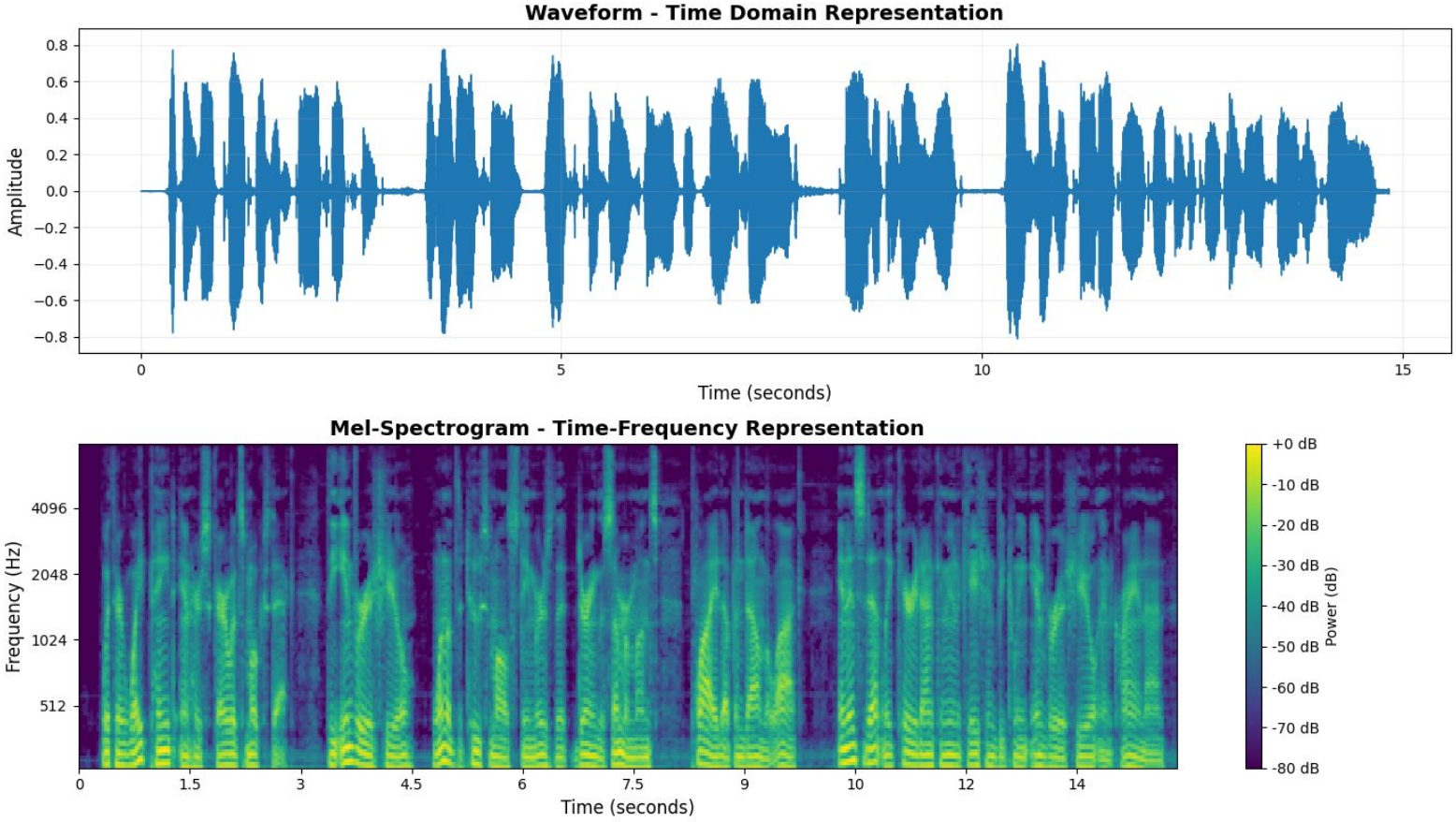
Time Stretch, Pitch Shift, Noise

We need another compact and informative representation!

Spoiler: **spectrograms!**

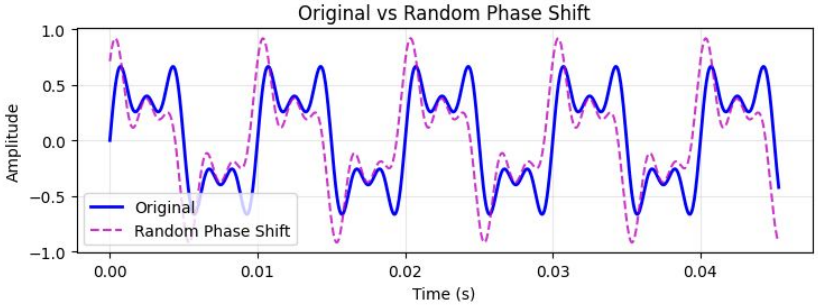
New representation: Spectrograms

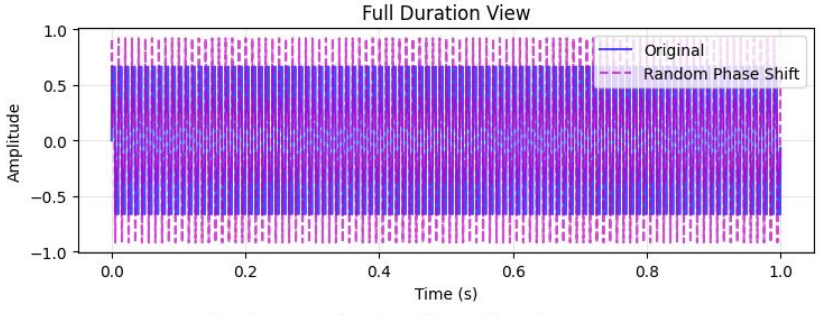
Spectrograms!

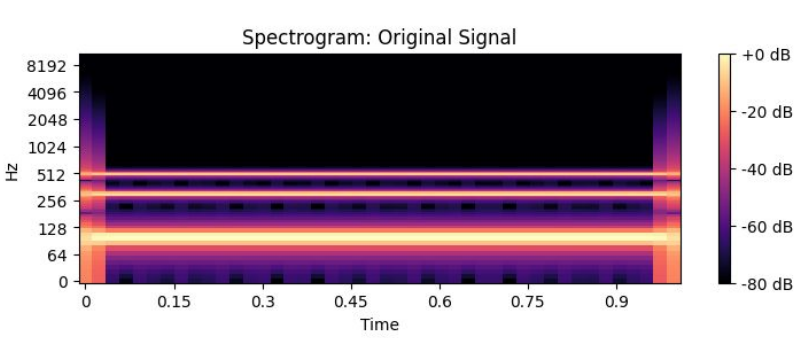


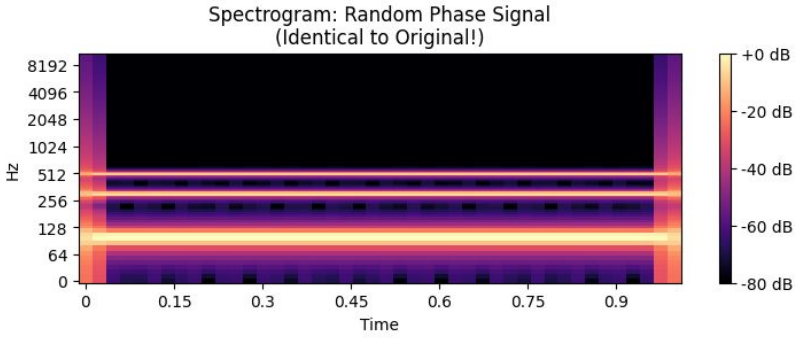
Why spectrograms?

Phase Sensitivity



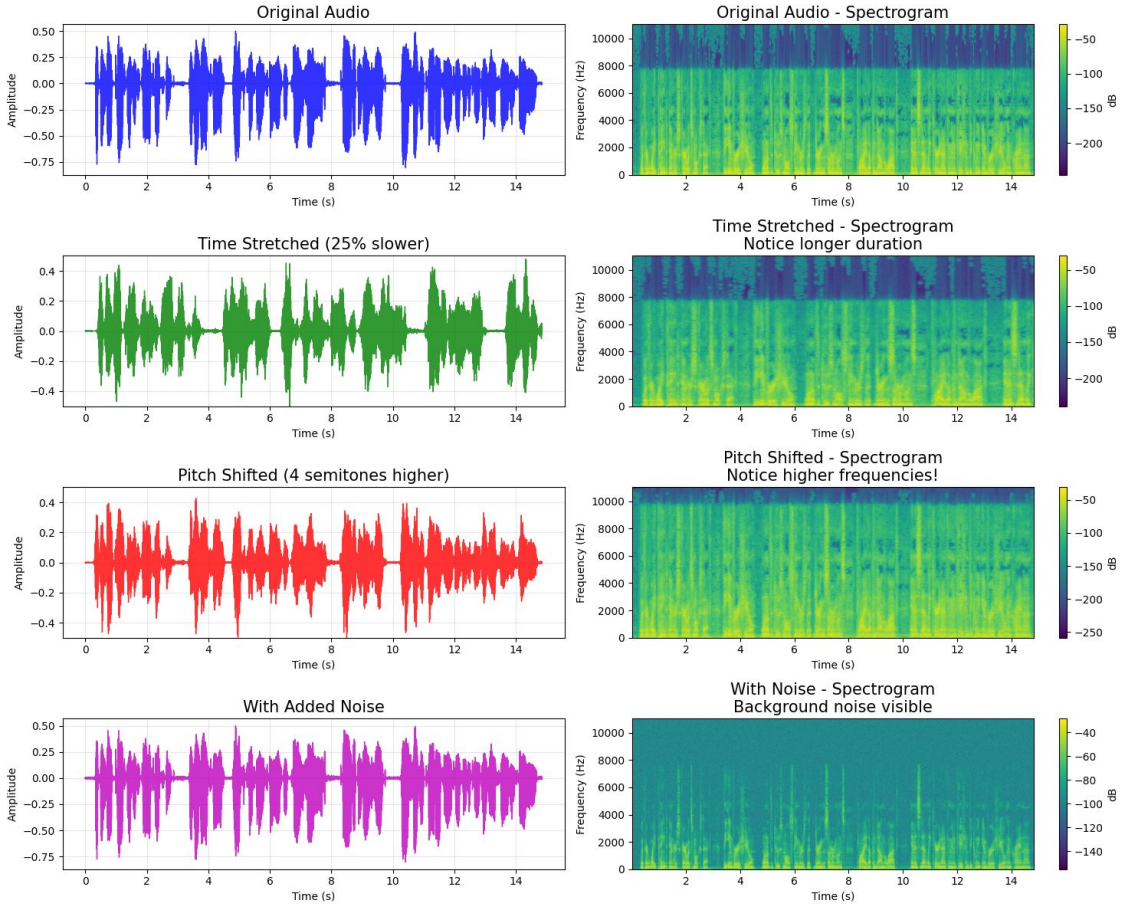






Spectrograms are the same!

New representation: Spectrograms

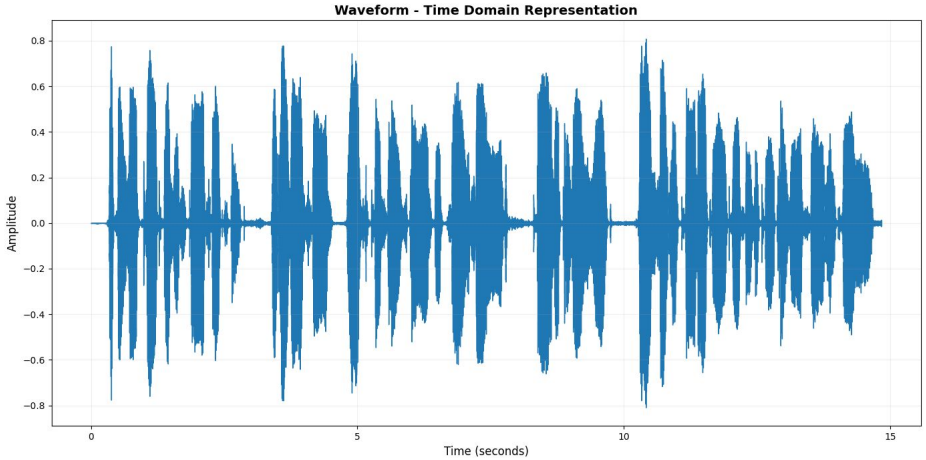


Time Stretch, Pitch Shift, Noise

Spectrograms are different!

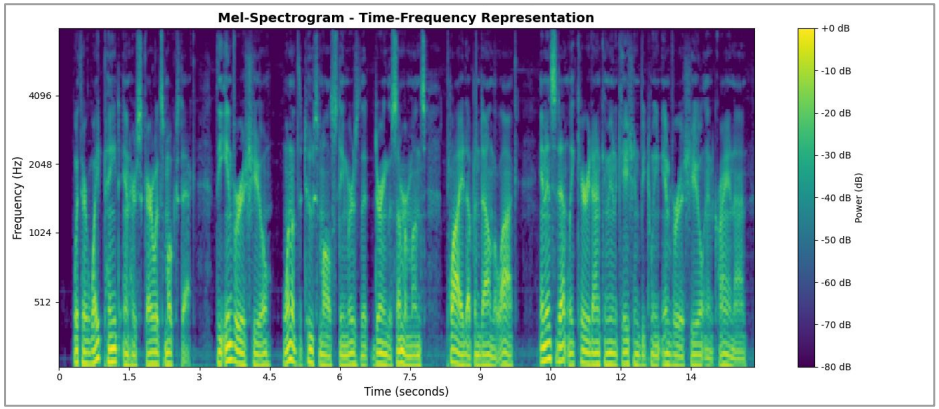
New representation: Spectrograms

Waveform



- **Size**: 327222 → **327K**
- Low Information density
- Sensitive to phase shift
- Not sensitive to time stretch, pitch shift, noise addition

Mel-Spectrogram

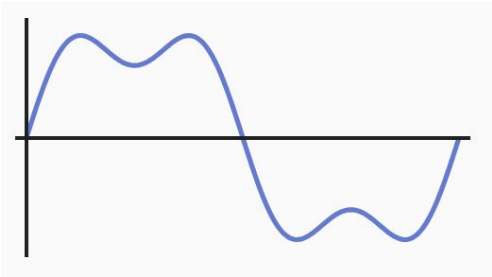


- **Size**: 81920 → **81K**
- High Information density
- Not sensitive to phase shift
- Sensitive to time stretch, pitch shift, noise addition

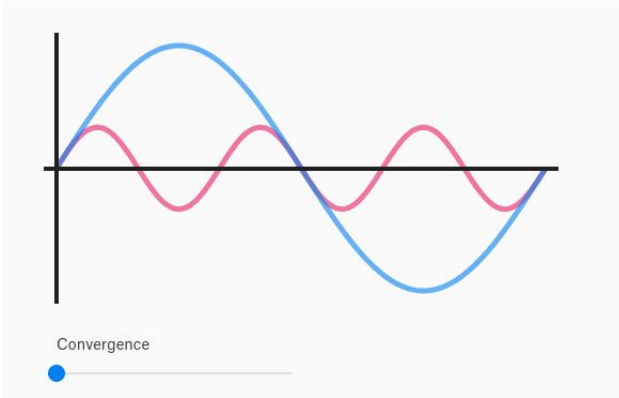
Fourier Series

Signal as Fourier Series

Complex signal



Sum of simple signals

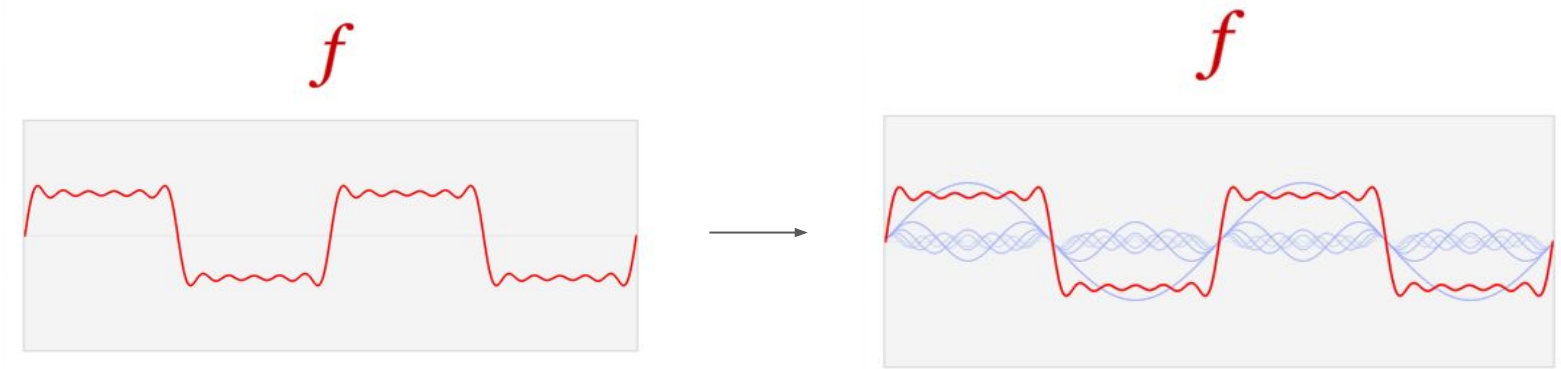


Any complex signal can be expand into simple signals!

Images source:https://pudding.cool/2018/02/waveforms/

Signal as Fourier Series

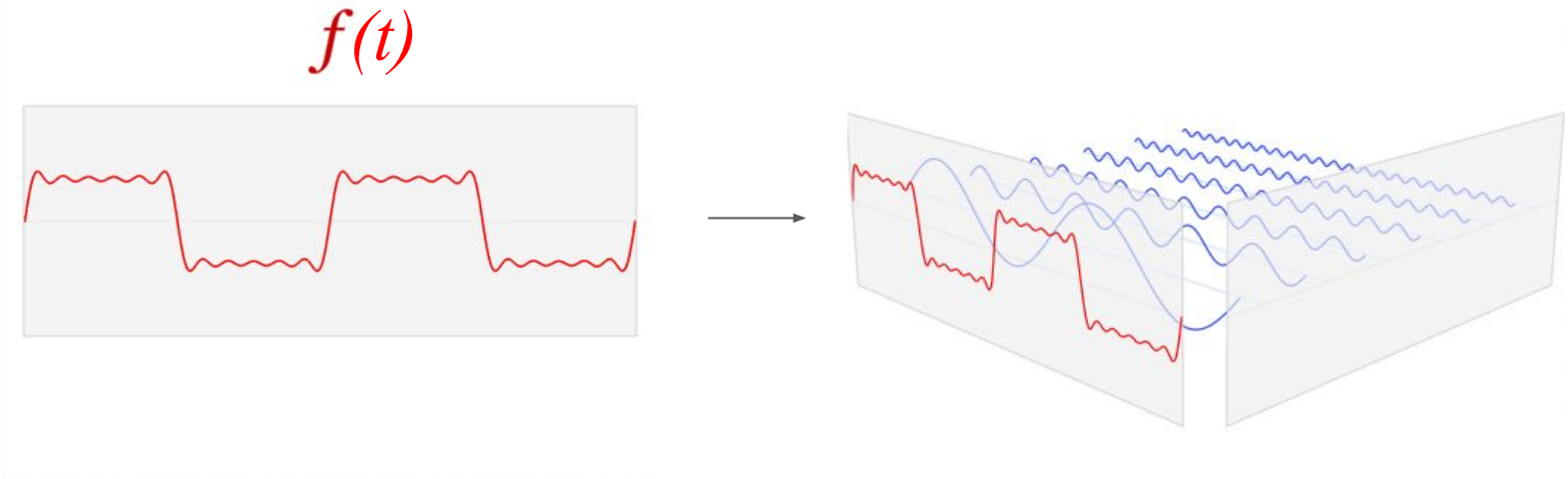
Specifically: Any periodic function can be decomposed into **a sum of sine and cosine waves**.



Images from the source: Link to the source

Signal as Fourier Series

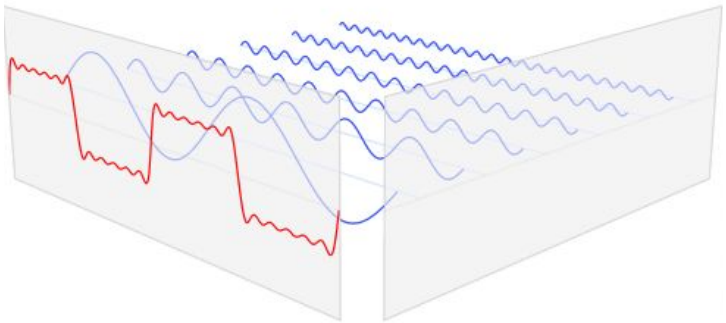
More Specific: Any periodic function ***f(t)*** with period *T* and integrable over the interval *[0, T]*, can be represented as a **Fourier Series**.



Fourier Series $\longrightarrow$ $$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

Images from the source: Link to the source

Signal as Fourier Series



More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval $[0, T]$, can be represented as a **Fourier Series**:

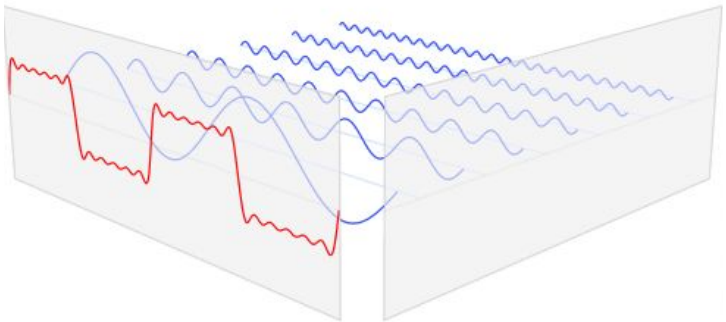
$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

The **harmonic number**.

- n=1 is the **fundamental frequency** (the main pitch).
- n=2 is the second harmonic (one octave higher).
- n=3 is the third harmonic, and so on.

Images from the source: Link to the source

Signal as Fourier Series



More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval $[0, T]$, can be represented as a **Fourier Series**:

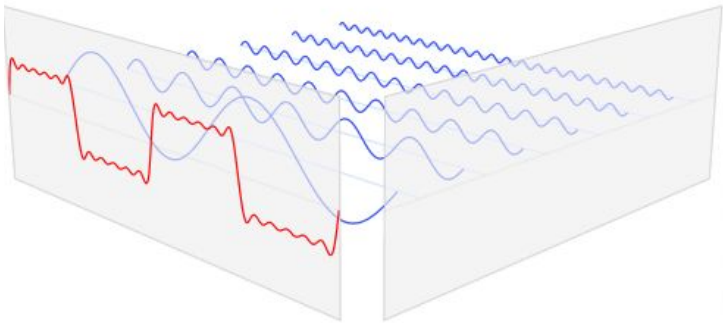
$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$a_0 = \frac{2}{T} \int_0^T f(t)\,dt$$

Просто среднее значение амплитуды за период T.

Images from the source: Link to the source

Signal as Fourier Series



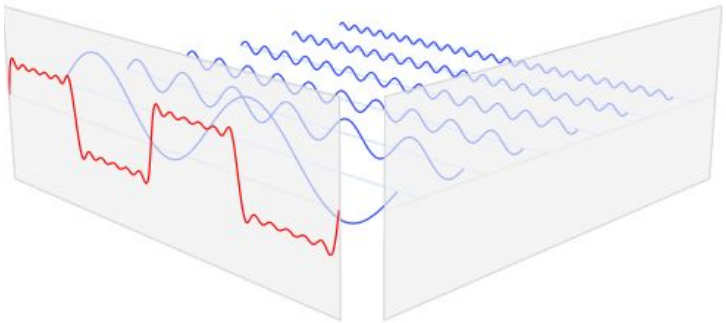
More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval $[0, T]$, can be represented as a **Fourier Series**:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi nt}{T}\right) + b_n \sin\left(\frac{2\pi nt}{T}\right) \right]$$

They are **Fourier Coefficients**.

Images from the source: Link to the source

Signal as Fourier Series



More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval *[0, T]*, can be represented as a **Fourier Series**:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi nt}{T}\right) + b_n \sin\left(\frac{2\pi nt}{T}\right) \right]$$

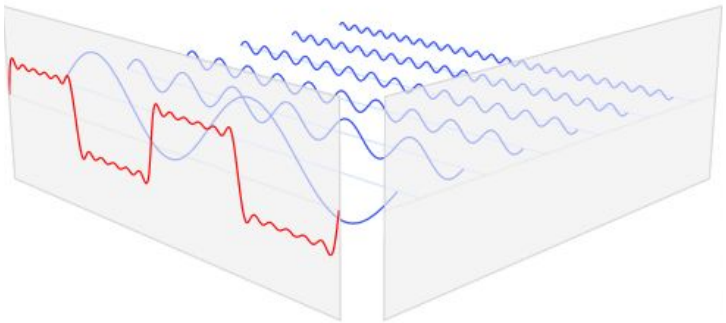
$$a_n = \frac{2}{T} \int_0^T f(t) \cdot \cos\left(\frac{2\pi nt}{T}\right) dt$$

Показывает насколько сигнал похож на cosine wave при частоте n-гармоники.

If `f(t)` is an **odd function** (antisymmetric, like `sin(t)`), all `a□` (including `a₀`) will be zero.

Images from the source: Link to the source

Signal as Fourier Series



More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval *[0, T]*, can be represented as a **Fourier Series**:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi nt}{T}\right) + b_n \sin\left(\frac{2\pi nt}{T}\right) \right]$$

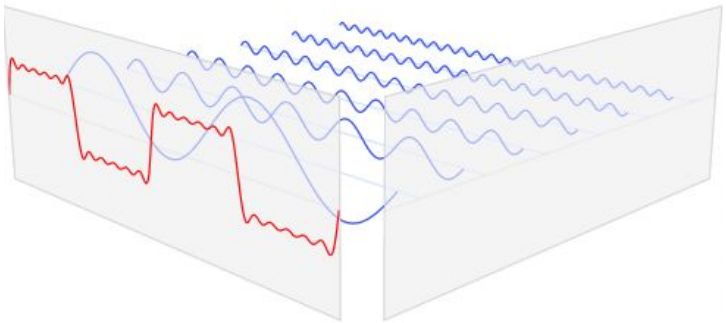
$$b_n = \frac{2}{T} \int_0^T f(t) \cdot \sin\left(\frac{2\pi nt}{T}\right) dt$$

Показывает насколько сигнал похож на sine wave при частоте n-гармоники.

If `f(t)` is an **even function** (symmetric around the y-axis, like `cos(t)`), all `b□` will be zero.

Images from the source: Link to the source

Signal as Fourier Series



More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval $[0, T]$, can be represented as a **Fourier Series**:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

Instead of tracking two sets of real coefficients (a□ and b□), we can **track one** set of complex coefficients (c□) by using **Euler formula**:

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

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

Images from the source: Link to the source

Signal as Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

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

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$\omega = \frac{2\pi}{T}$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cdot \frac{e^{in\omega t} + e^{-in\omega t}}{2} + b_n \cdot \frac{e^{in\omega t} - e^{-in\omega t}}{2i} \right)$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n}{2} + \frac{b_n}{2i} \right) \cdot e^{in\omega t} + \left(\frac{a_n}{2} - \frac{b_n}{2i} \right) \cdot e^{-in\omega t}$$

$$\frac{1}{i} = -i$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n}{2} - \frac{b_n}{2} i \right) \cdot e^{in\omega t} + \left(\frac{a_n}{2} + \frac{b_n}{2} i \right) \cdot e^{-in\omega t}$$

Images from the source: Link to the source

Signal as Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$\frac{1}{i} = -i$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n}{2} - \frac{b_n}{2} i \right) \cdot e^{in\omega t} + \left(\frac{a_n}{2} + \frac{b_n}{2} i \right) \cdot e^{-in\omega t}$$

$$n = 1, 2, 3, \cdots$$

$$\sum_{n=1}^{\infty} \left(\frac{a_n}{2} + \frac{ib_n}{2} \right) e^{-in\omega t} = \sum_{n=-\infty}^{-1} \left(\frac{a_{|n|}}{2} + \frac{ib_{|n|}}{2} \right) e^{in\omega t}$$

Signal as Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$\frac{1}{i} = -i$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n}{2} - \frac{b_n}{2} i \right) \cdot e^{in\omega t} + \left(\frac{a_n}{2} + \frac{b_n}{2} i \right) \cdot e^{-in\omega t}$$

$$n = -1, -2, -3, \ldots$$

$$\sum_{n=1}^{\infty} \left(\frac{a_n}{2} + \frac{ib_n}{2} \right) e^{-in\omega t} = \sum_{n=-\infty}^{-1} \left(\frac{a_{|n|}}{2} + \frac{ib_{|n|}}{2} \right) e^{in\omega t}$$

Signal as Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$\frac{1}{i} = -i$$

$$f(t) = \boxed{\frac{a_0}{2}} + \boxed{\sum_{n=1}^{\infty}} \left(\frac{a_n}{2} - \frac{b_n}{2} i \right) \cdot e^{in\omega t} + \boxed{\sum_{n=-\infty}^{-1}} \left(\frac{a_{|n|}}{2} + \frac{i b_{|n|}}{2} \right) e^{in\omega t}$$

Signal as Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$\frac{1}{i} = -i$$

$$f(t) = \boxed{\frac{a_0}{2}} + \boxed{\sum_{n=1}^{\infty}} \left(\frac{a_n}{2} - \frac{b_n}{2} i \right) \cdot e^{in\omega t} + \boxed{\sum_{n=-\infty}^{-1}} \left(\frac{a_{|n|}}{2} + \frac{i b_{|n|}}{2} \right) e^{in\omega t}$$

$$c_n = \frac{a_n - b_n i}{2}, n > 0$$

$$c_n = \frac{a_n + b_n i}{2}, n < 0$$

$$c_n = \frac{a_0}{2}, n = 0$$

Signal as Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

$$cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$\frac{1}{i} = -i$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n}{2} - \frac{b_n}{2} i \right) \cdot e^{in\omega t} + \sum_{n=-\infty}^{-1} \left(\frac{a_{|n|}}{2} + \frac{i b_{|n|}}{2} \right) e^{in\omega t}$$

$$c_n = \frac{a_n - b_n i}{2}, n > 0$$

$$c_n = \frac{a_n + b_n i}{2}, n < 0$$

$$c_n = \frac{a_0}{2}, n = 0$$

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i(n\omega t)}$$

Signal as Fourier Series

Second thing we should prove is:

$$c_n = \frac{1}{T}\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt$$

Let's start with:

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{2\pi kt}{T}}$$

Multiply both sides by $e^{-i\frac{2\pi nt}{T}}$ and integrate over one period:

$$\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt = \sum_{k=-\infty}^{\infty} c_k \int_0^T e^{i\frac{2\pi(k-n)t}{T}}\,dt$$

Signal as Fourier Series

Multiply both sides by $e^{-i\frac{2\pi nt}{T}}$ and integrate over one period:

$$\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt = \sum_{k=-\infty}^{\infty} c_k \int_0^T e^{i\frac{2\pi(k-n)t}{T}}\,dt$$

Use the **orthogonality** of complex exponentials:

$$\int_0^T e^{i\frac{2\pi(k-n)t}{T}}\,dt = \begin{cases} T, & k = n \\ 0, & k \neq n \end{cases}$$

So only the term $k = n$ survives:

$$\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt = c_n \cdot T \quad \Rightarrow \quad c_n = \frac{1}{T}\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt$$

Signal as Fourier Series

Multiply both sides by $e^{-i\frac{2\pi nt}{T}}$ and integrate over one period:

$$\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt = \sum_{k=-\infty}^{\infty} c_k \int_0^T e^{i\frac{2\pi(k-n)t}{T}}\,dt$$

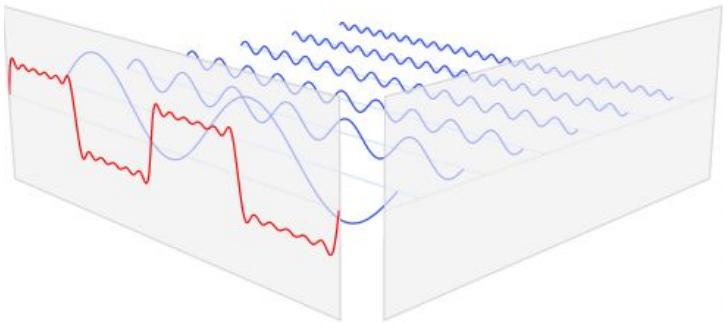
Use the **orthogonality** of complex exponentials:

$$\int_0^T e^{i\frac{2\pi(k-n)t}{T}}\,dt = \begin{cases} T, & k = n \\ 0, & k \neq n \end{cases}$$

So only the term $k = n$ survives:

$$\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt = c_n \cdot T \quad \Rightarrow \quad \boxed{c_n = \frac{1}{T}\int_0^T f(t)e^{-i\frac{2\pi nt}{T}}\,dt}$$

Signal as Fourier Series



More Specific: Any periodic function ***f(t)*** with period T and integrable over the interval *[0, T]*, can be represented as a **Fourier Series**:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

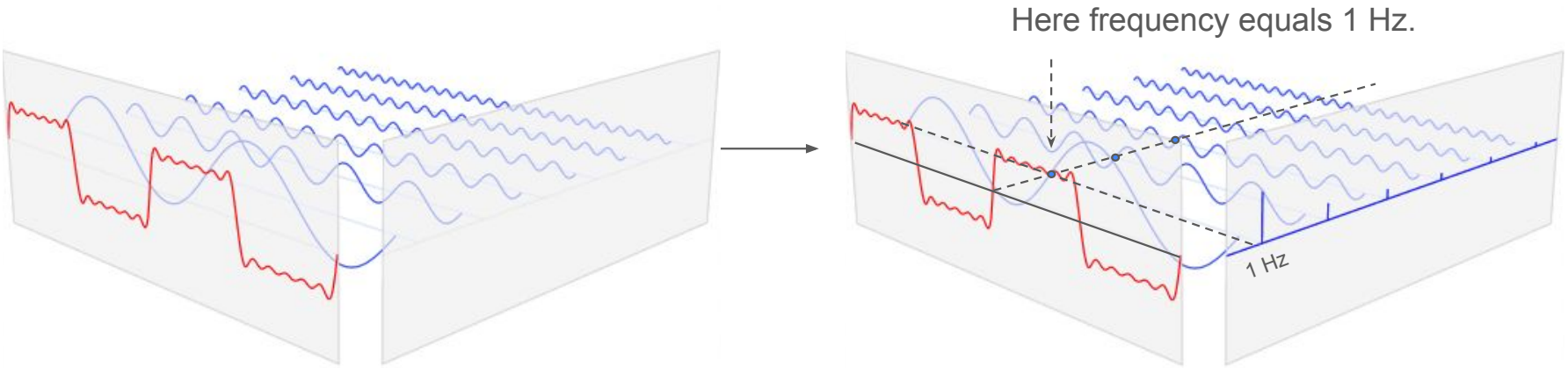
Instead of tracking two sets of real coefficients (a□ and b□), we can **track one** set of complex coefficients (c□) by using **Euler formula**:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i(n\omega t)}$$

$$c_n = \frac{1}{T} \int_0^T f(t) \cdot e^{-i(n\omega t)} dt$$

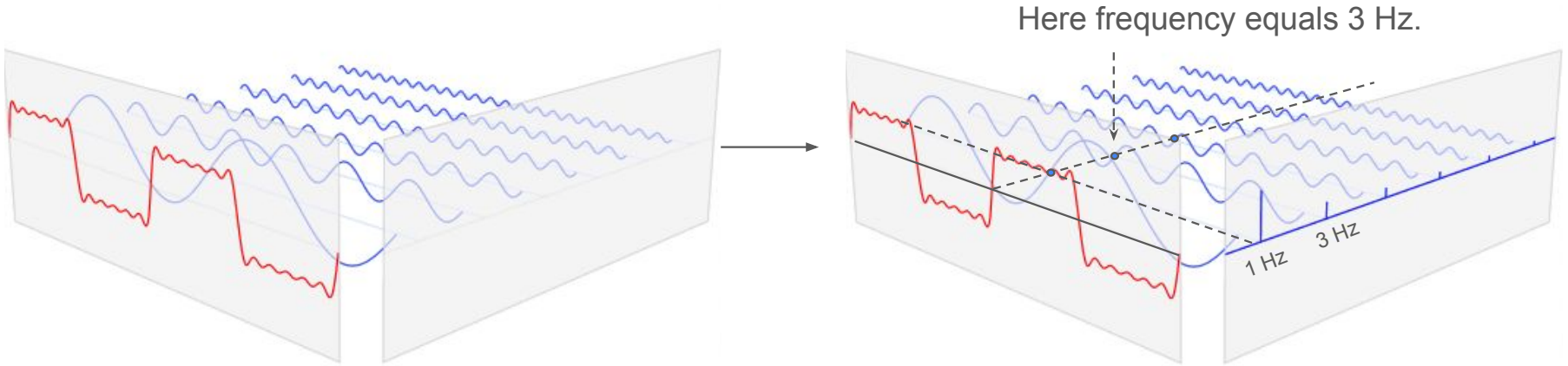
Images from the source: Link to the source

Signal as Fourier Series



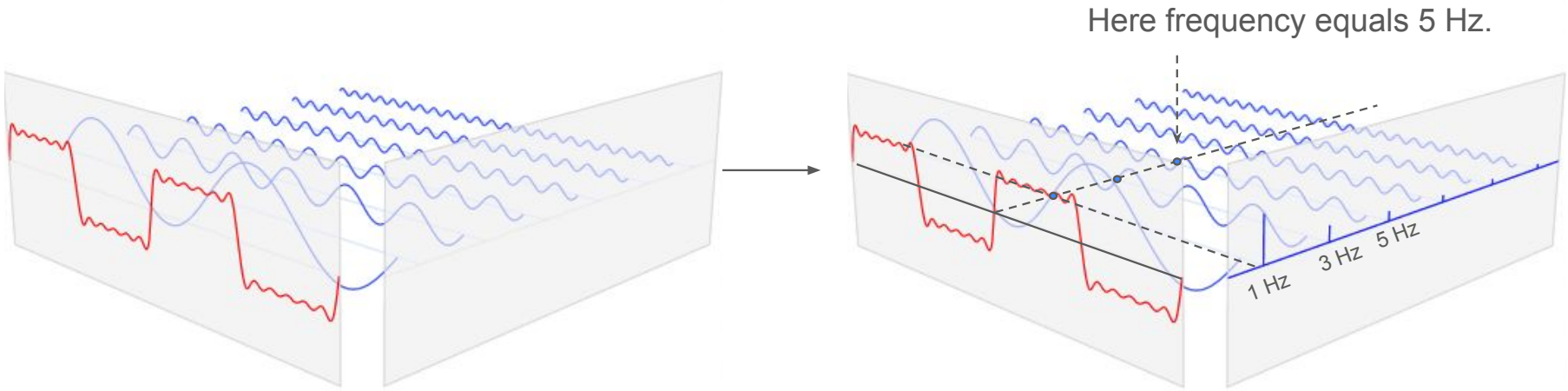
Images from the source: [Link to the source]

Signal as Fourier Series



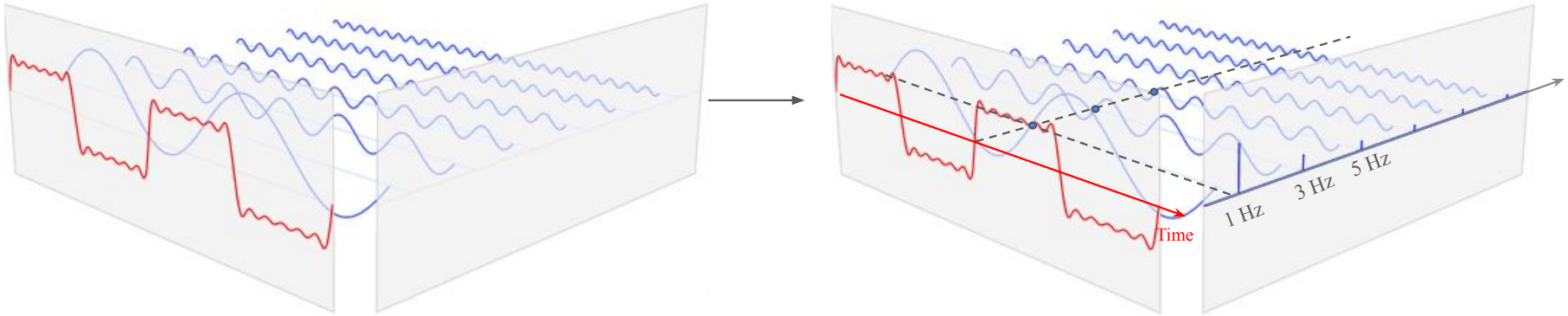
Images from the source: Link to the source

Signal as Fourier Series



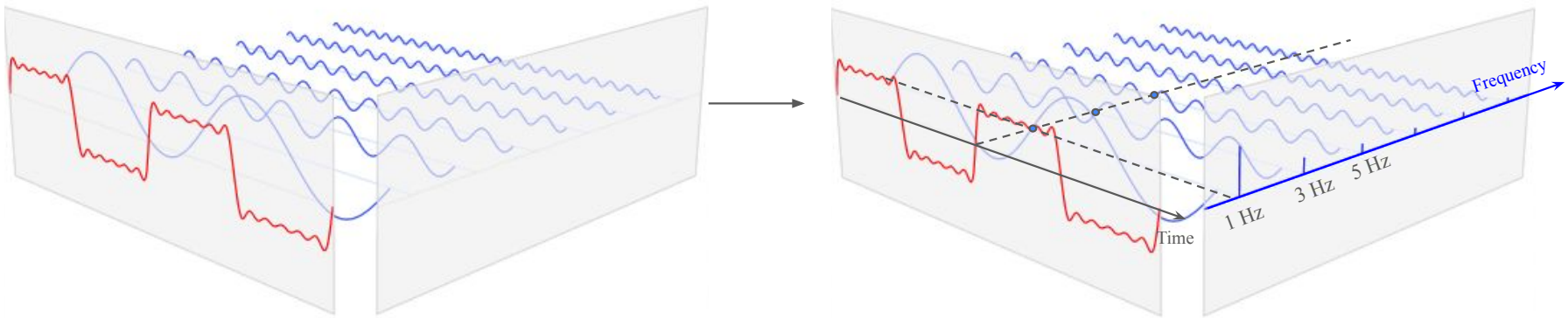
Images from the source: [Link to the source]

Signal as Fourier Series



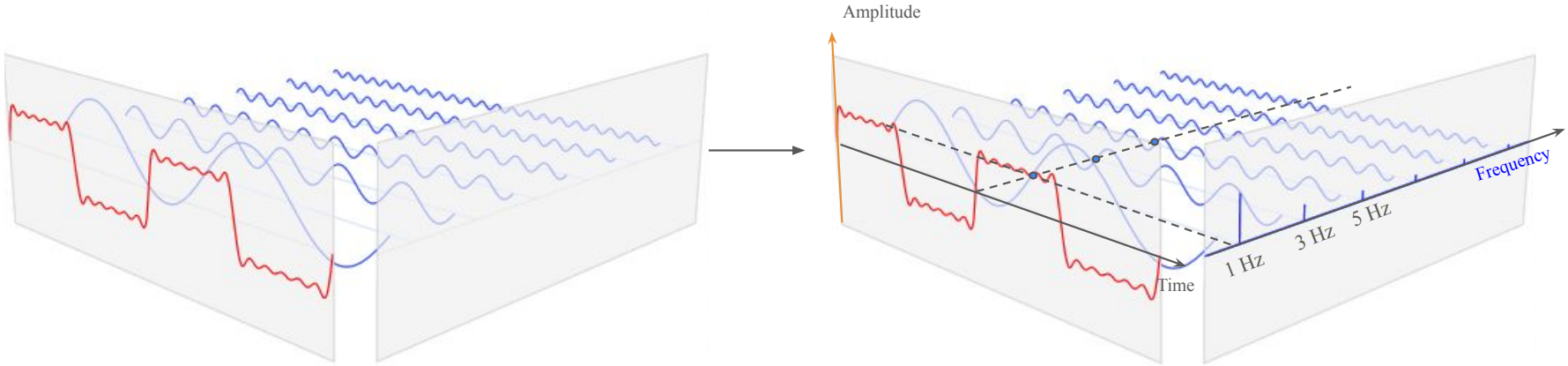
Images from the source: [Link to the source]

Signal as Fourier Series



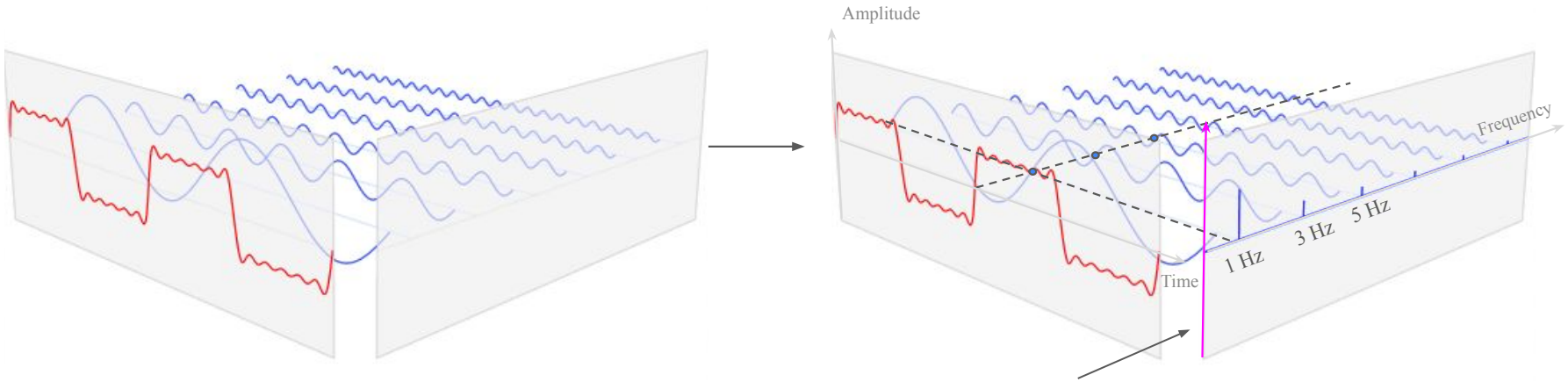
Images from the source: [Link to the source]

Signal as Fourier Series



Images from the source: [Link to the source]

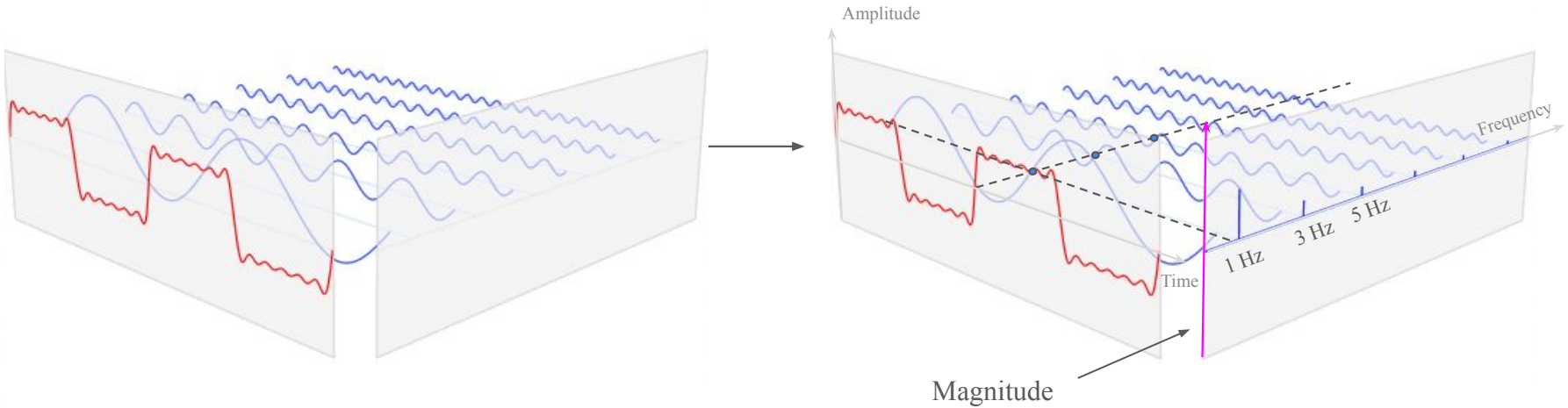
Signal as Fourier Series



But what is this axis?

Images from the source: Link to the source

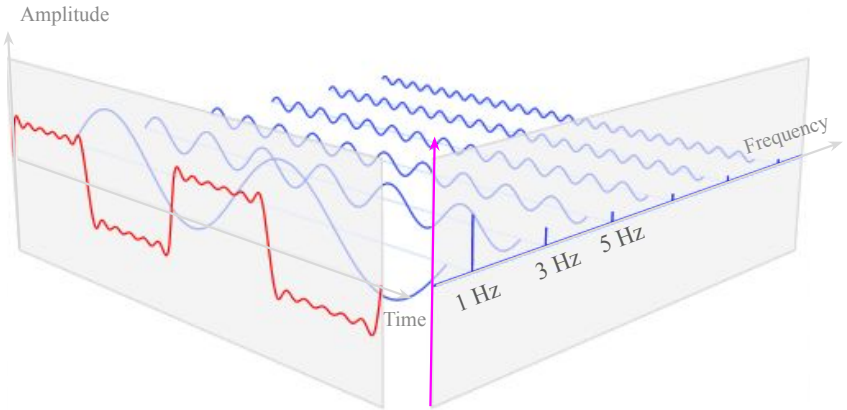
Signal as Fourier Series



Higher magnitude indicates higher similarity between signal and sinusoid.

Images from the source: Link to the source

Signal as Fourier Series



We are interested in this frequency domain.

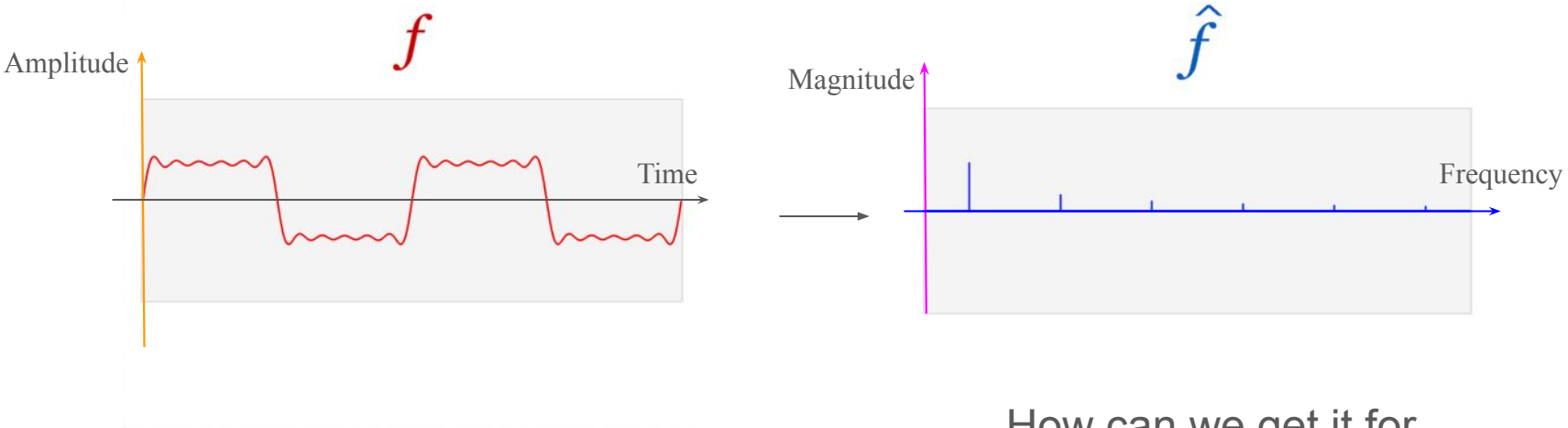
It is called Frequency Spectrum.

Limitation of Fourier Series: periodic signal and discrete frequency spectrum.

Images from the source: Link to the source

Signal as Fourier Series

We are interested in this frequency domain.

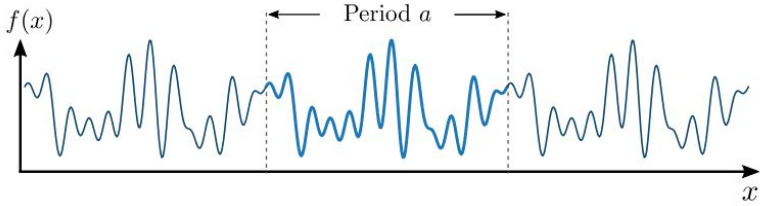


How can we get it for non-periodic signals?

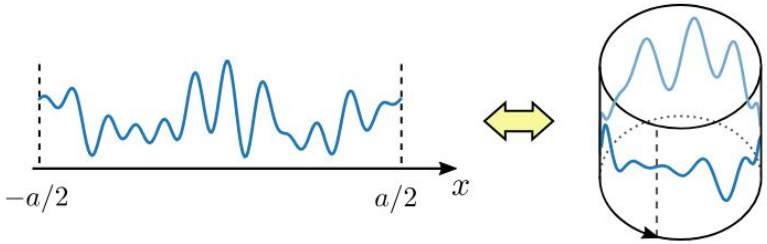
Images from the source: Link to the source

Fourier Transform → **Fourier Series** for **non-periodic** signals**.**

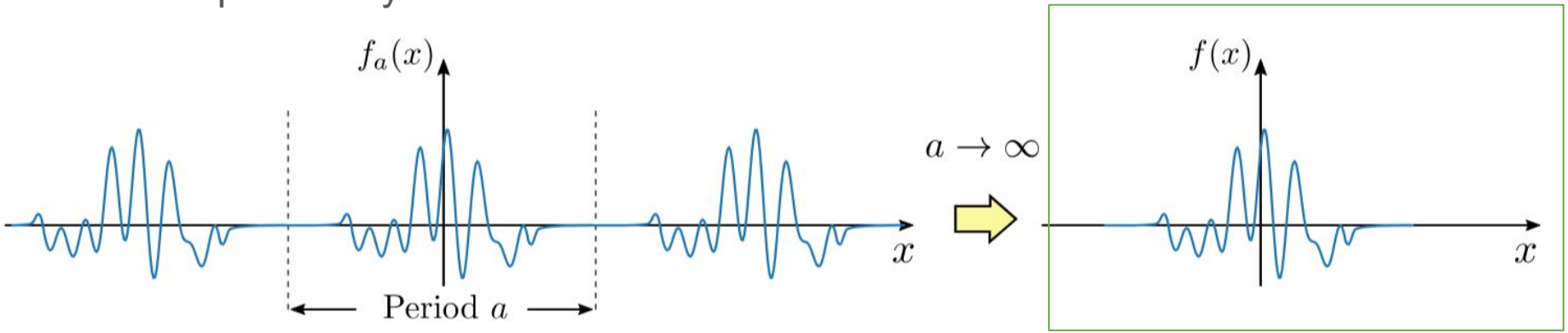
Fourier Transform



$$f(x+a) = f(x), \quad \forall\ x \in \mathbb{R}.$$



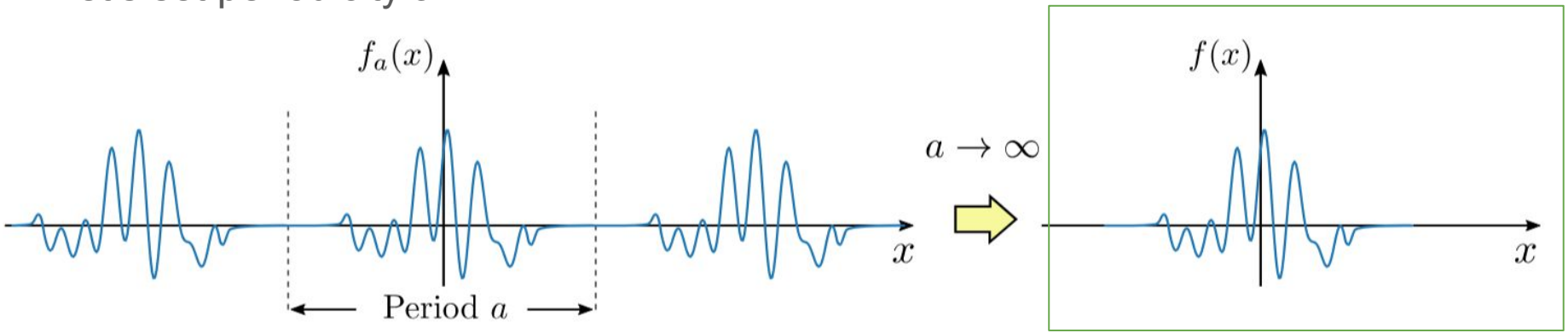
Let's set periodicity a → ∞



Source:https://www1.spms.ntu.edu.sg/~ydchong/teaching/10_fourier_transform.pdf

Fourier Transform

Let's set periodicity a → ∞

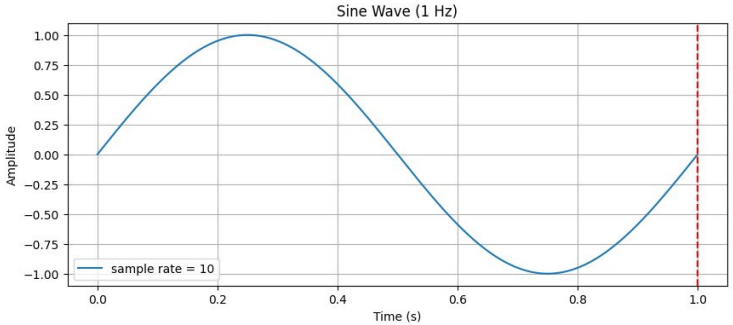


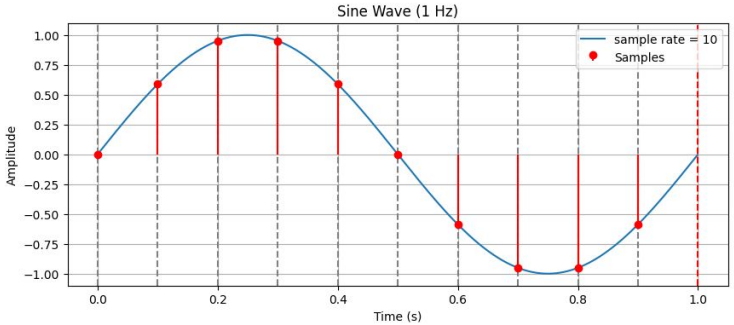
$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-i\omega t} dt$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \cdot e^{i\omega t} d\omega$$

Source:https://www1.spms.ntu.edu.sg/~ydchong/teaching/10_fourier_transform.pdf

Discrete Fourier Transform





$g(t)$ $\quad t = nT \quad$ $x[n]$

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

Discrete Fourier Transform

continuous

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

Discrete Fourier Transform

infinite

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

Discrete Fourier Transform

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

1. Let's make finite number of samples [0, …, N-1]

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

Discrete Fourier Transform

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

1. Let's make finite number of samples [0, …, N-1]

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

2. Let's make finite number of frequencies and M = N

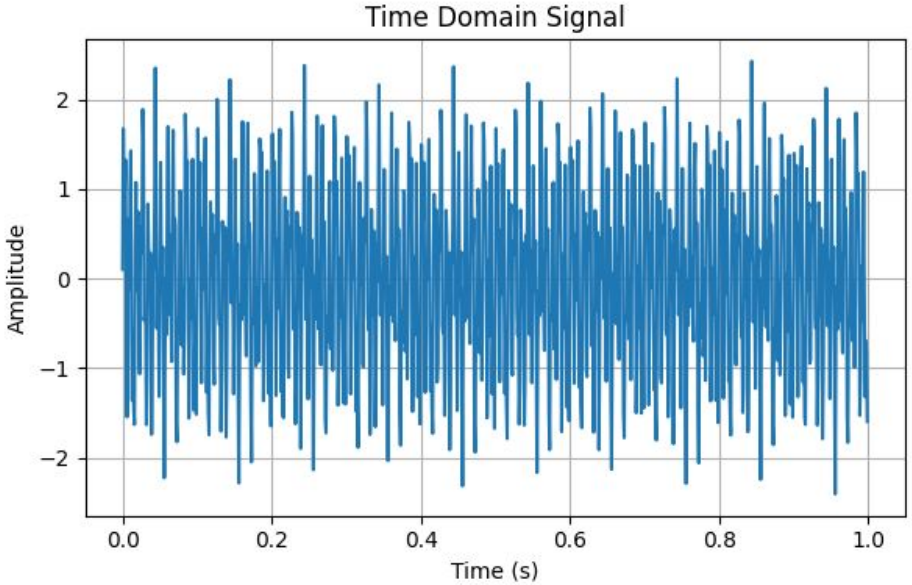
$$k = [0, M-1] = [0, N-1]$$

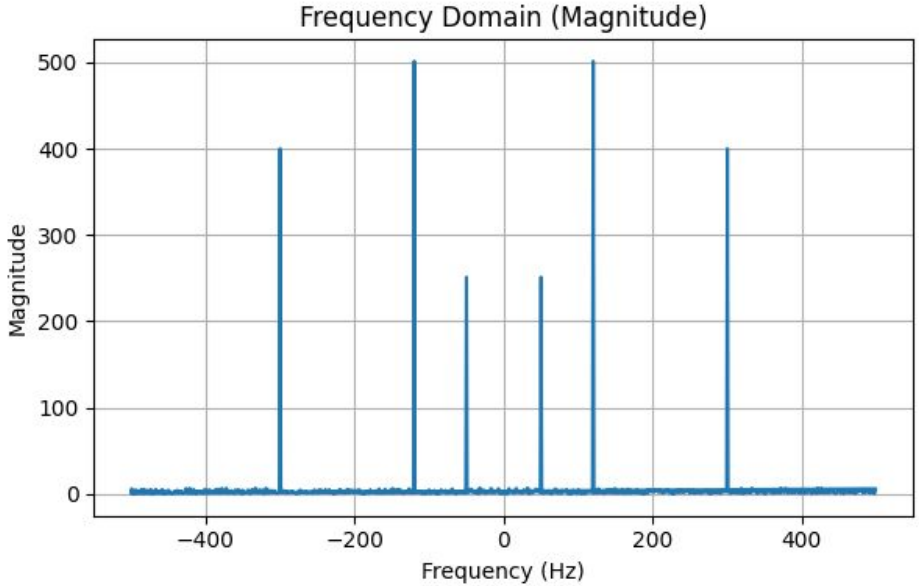
$$\hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n\frac{k}{N}}$$

To make the representation **fully discrete**, we **sample x(f)** at **N equally spaced frequencies** over one period.

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

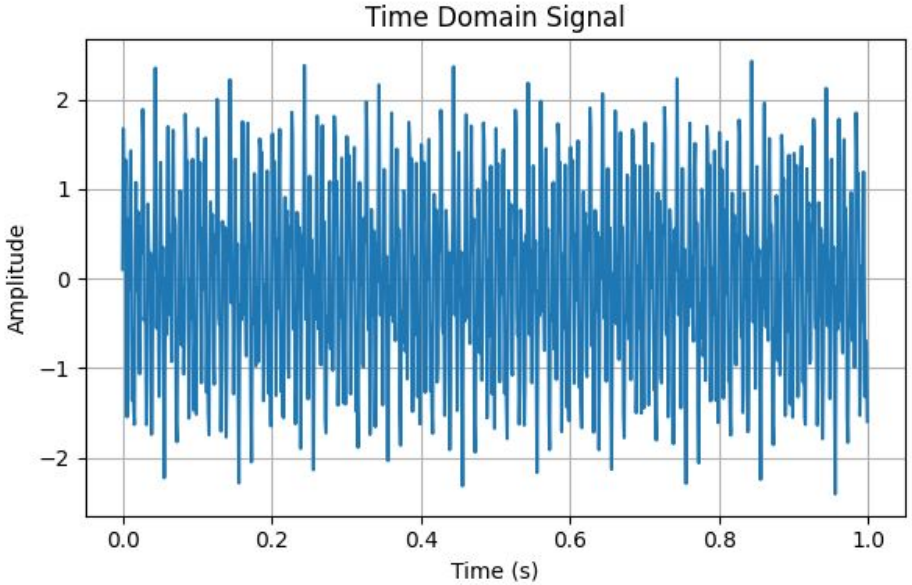
How to Extract the Fourier Transform with Python

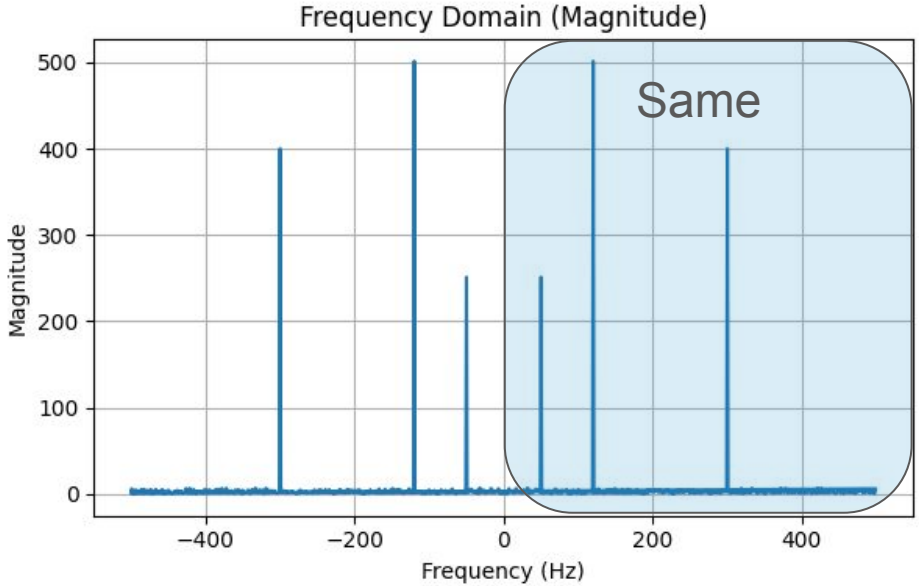





```
# Compute FFT
fft_result = np.fft.fft(signal)
fft_freq = np.fft.fftfreq(len(signal), 1/sr)
```

How to Extract the Fourier Transform with Python

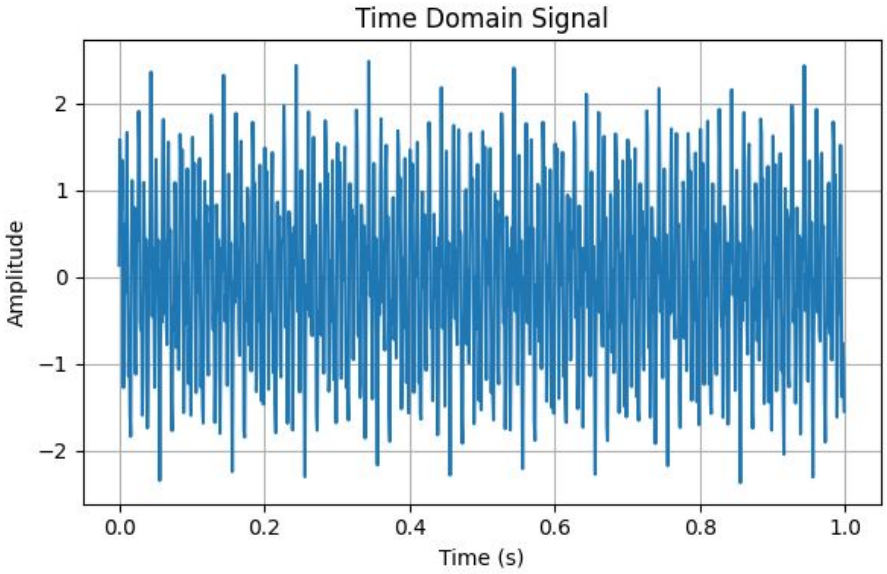


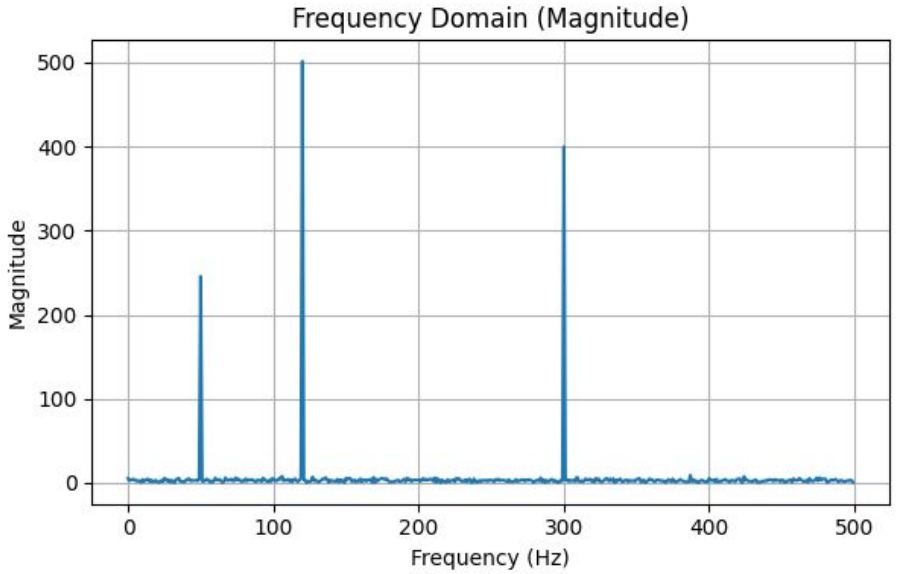



```
# Compute FFT
fft_result = np.fft.fft(signal)
fft_freq = np.fft.fftfreq(len(signal), 1/sr)
```

Since they are symmetric…

How to Extract the Fourier Transform with Python





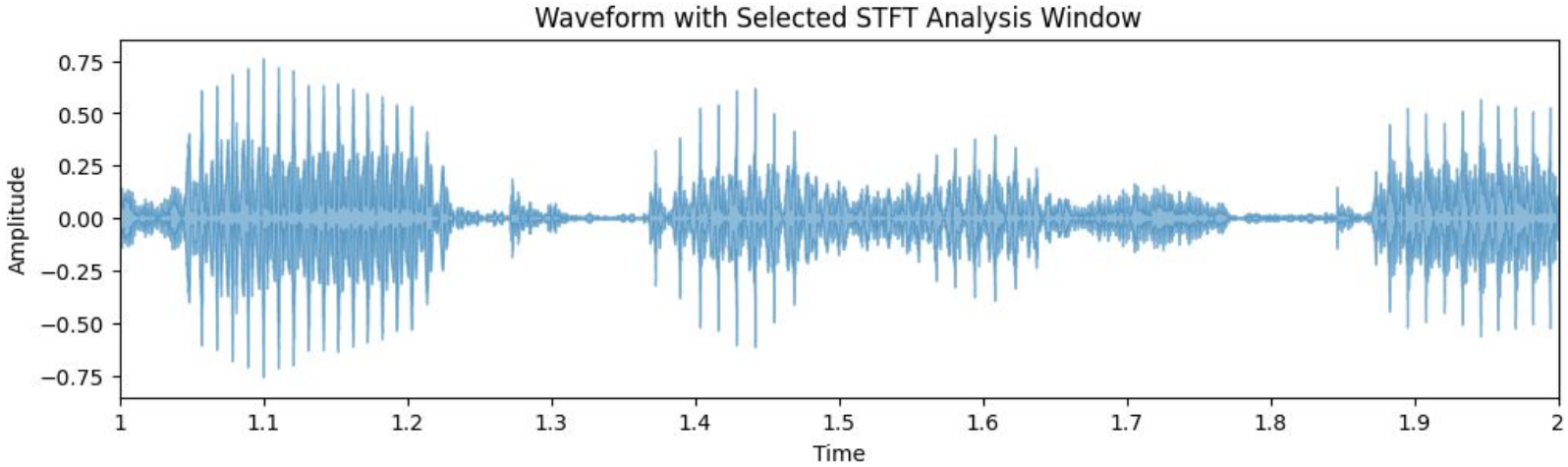

```
# Compute FFT
fft_result = np.fft.fft(signal)
fft_freq = np.fft.fftfreq(len(signal), 1/sr)
```

People plot and use just half of it.

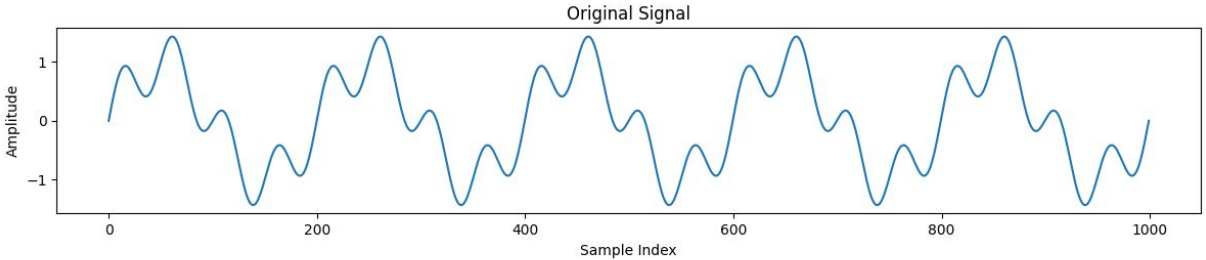
Short Time Fourier Transform (Spectrograms)

Catch changes of the spectrum over time: Short Time Fourier Transform (STFT)

But our signal changes over time.

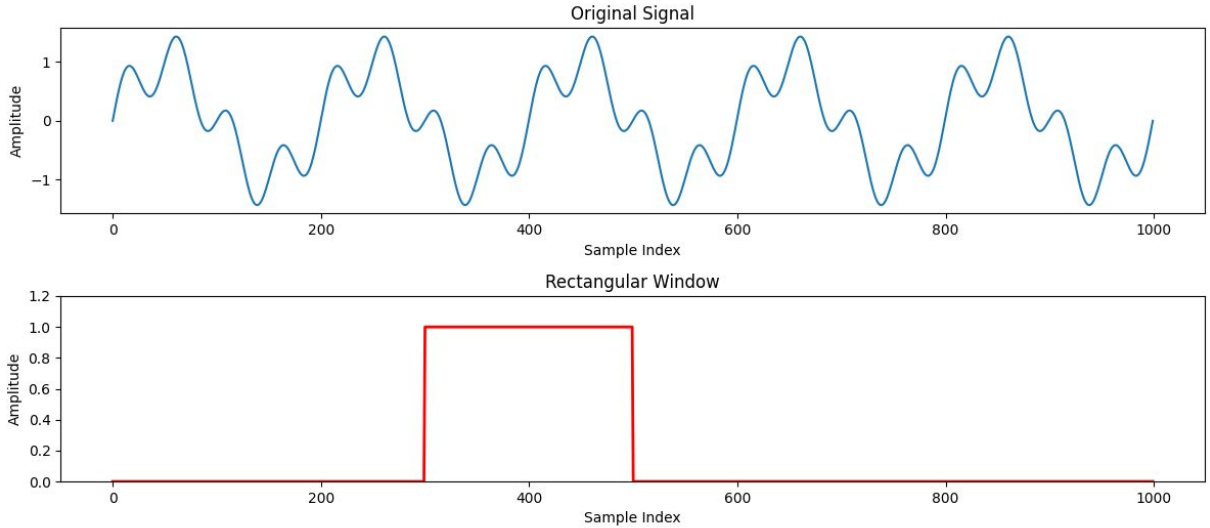


STFT: window function



Let’s consider simple signal.

STFT: window function

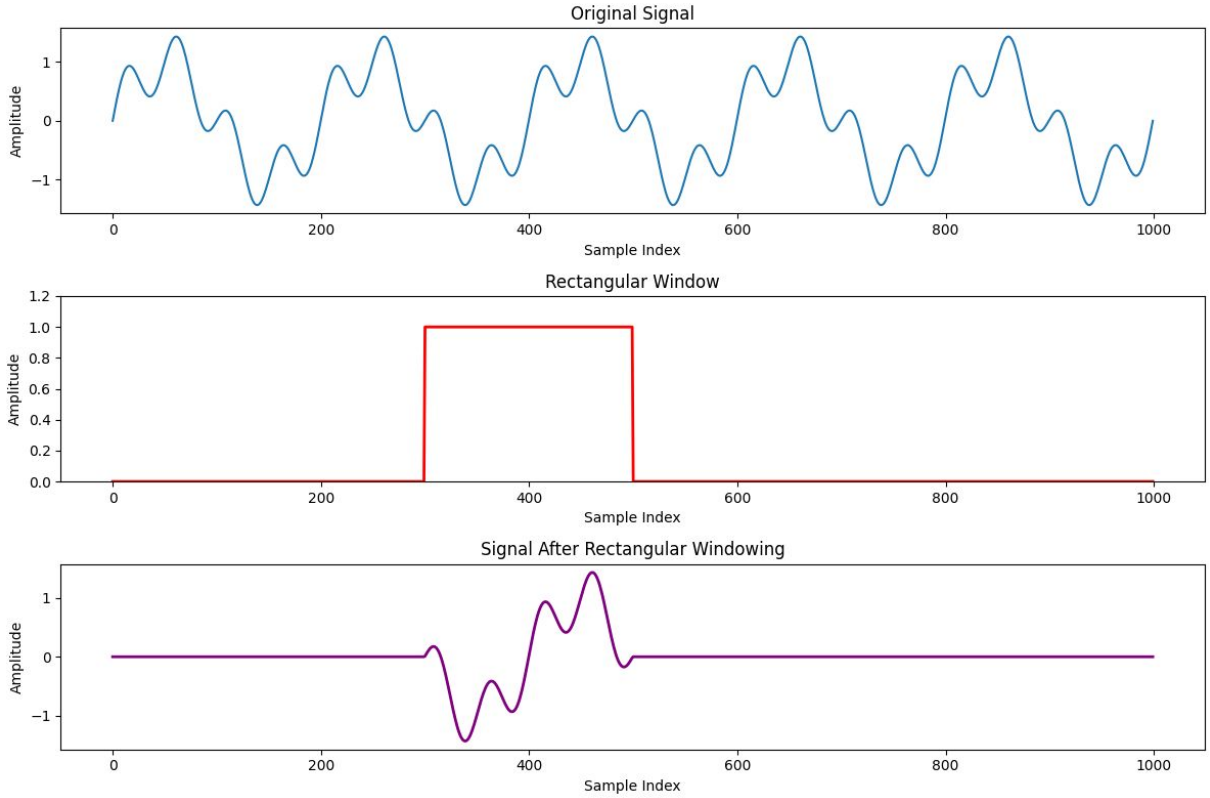


Step 1. Apply window function to a signal

```python
import numpy as np
import matplotlib.pyplot as plt

# Create a test signal
t = np.linspace(0, 1, 1000)
signal = np.sin(2*np.pi*5*t) + 0.5*np.sin(2*np.pi*20*t)

# Create rectangular window for entire signal length
window_length = 200
start_index = 300
rectangular_window = np.zeros_like(signal)
rectangular_window[start_index:start_index + window_length] = 1.0

# Apply window to the entire signal
windowed_signal = signal * rectangular_window
```

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

STFT: window function

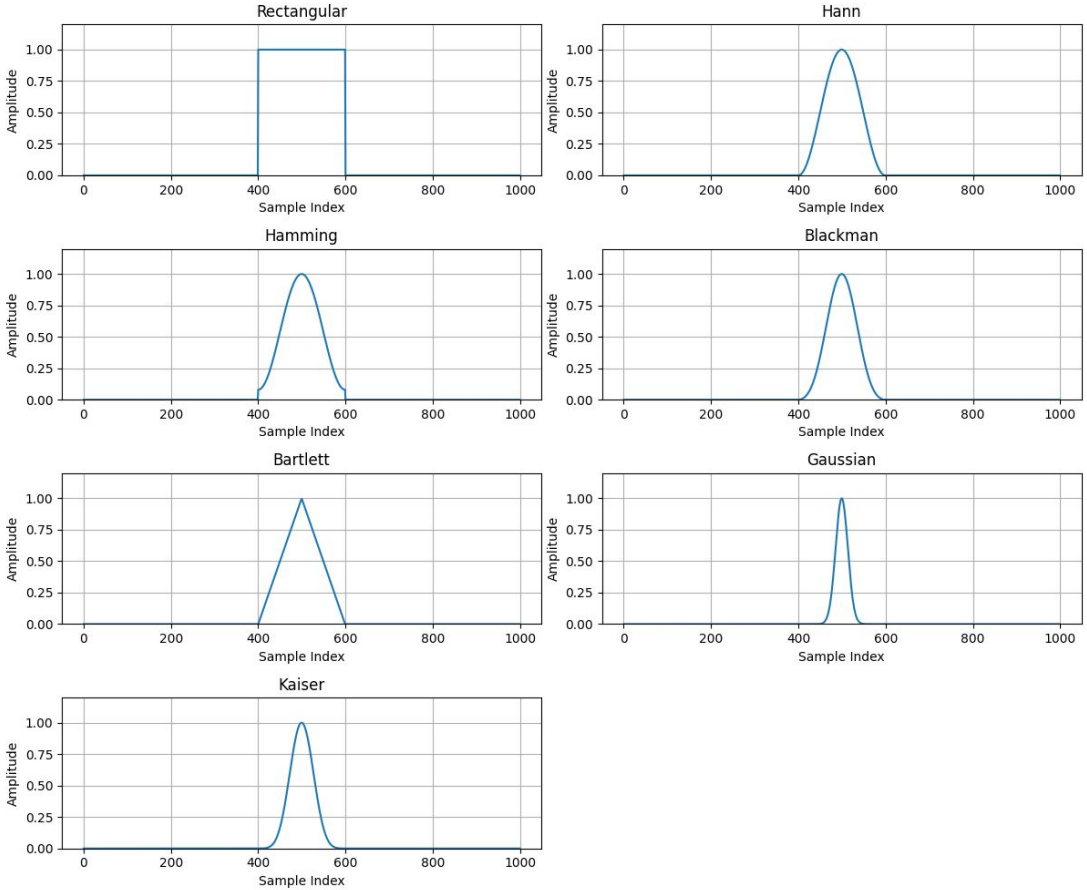


Step 1. Apply window function to a signal

```python
import numpy as np
import matplotlib.pyplot as plt

# Create a test signal
t = np.linspace(0, 1, 1000)
signal = np.sin(2*np.pi*5*t) + 0.5*np.sin(2*np.pi*20*t)

# Create rectangular window for entire signal length
window_length = 200
start_index = 300
rectangular_window = np.zeros_like(signal)
rectangular_window[start_index:start_index + window_length] = 1.0

# Apply window to the entire signal
windowed_signal = signal * rectangular_window
```

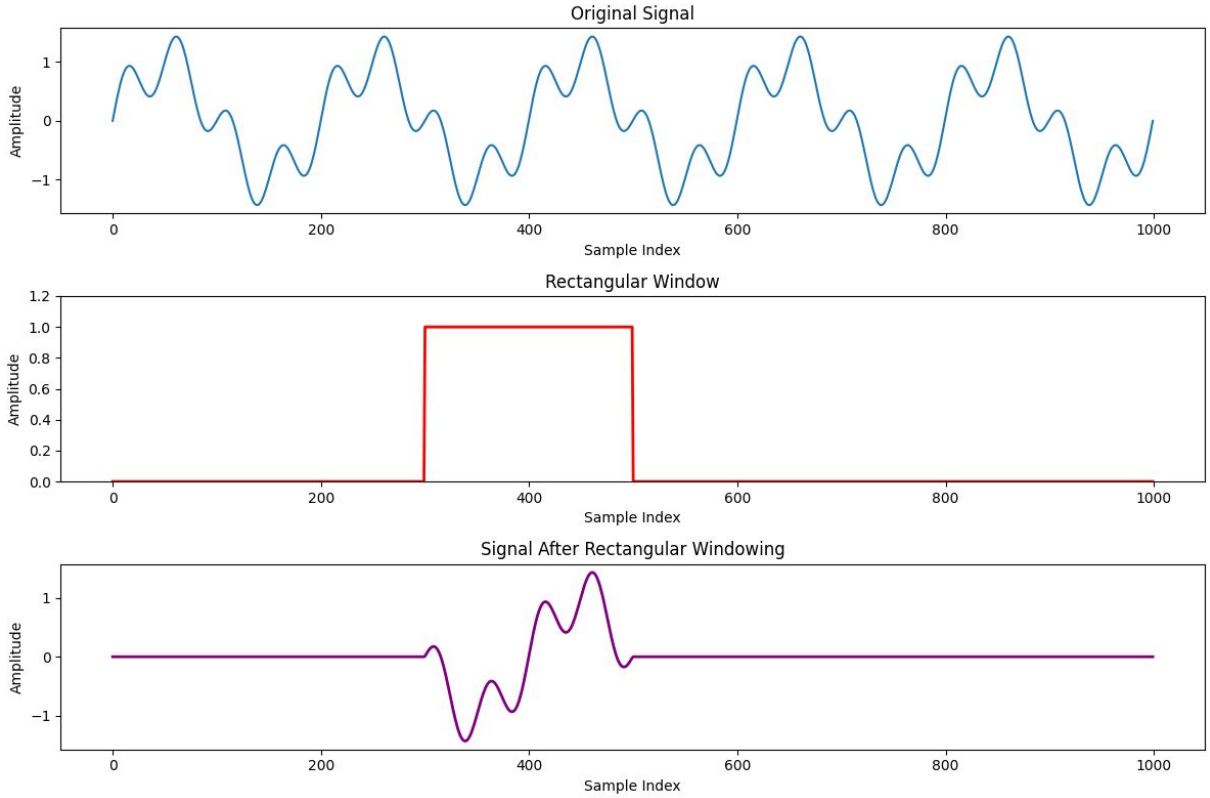
$$x_w(t) = x(t) \cdot w(t)$$

STFT: Different type of windows



There are lots of different window functions.

STFT: window function



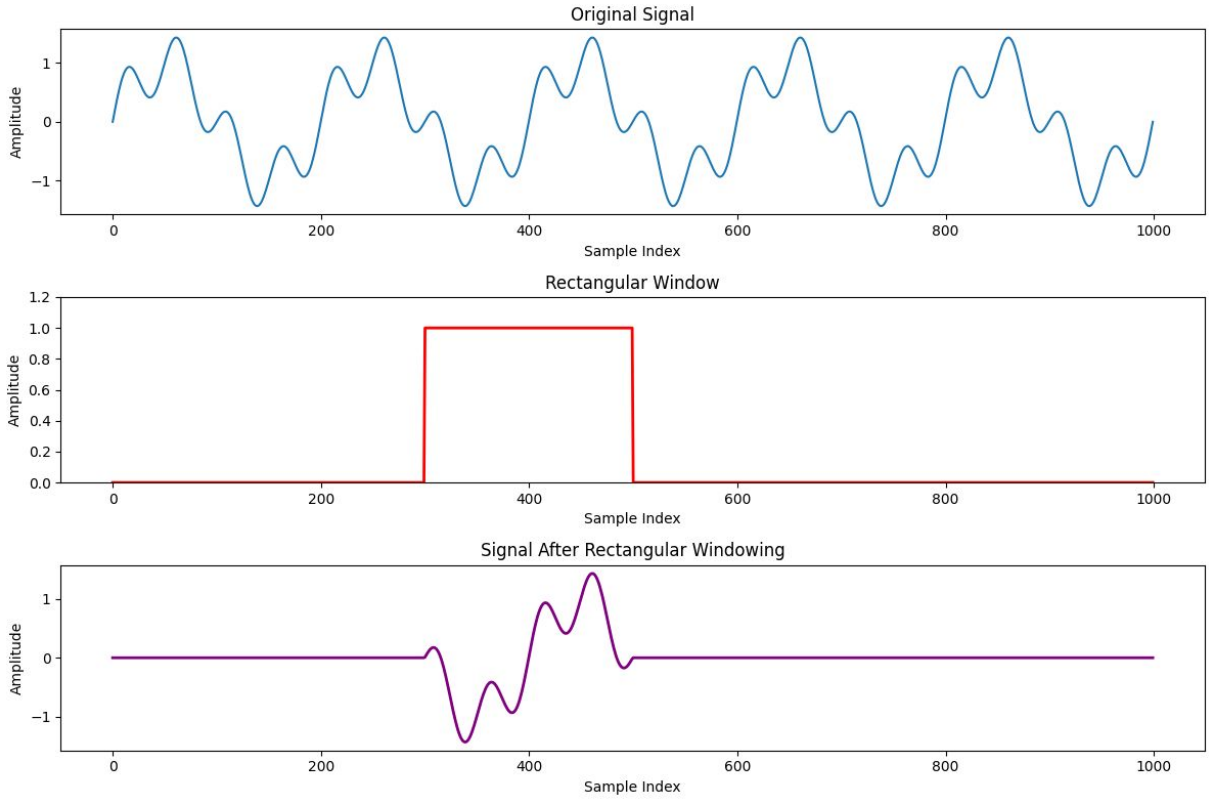
Why rectangular window is just not enough?

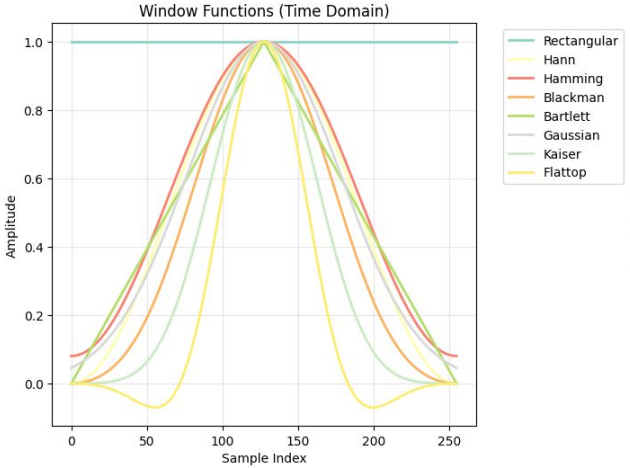
This will be in your HW2.

STFT: window function

Important parameters:

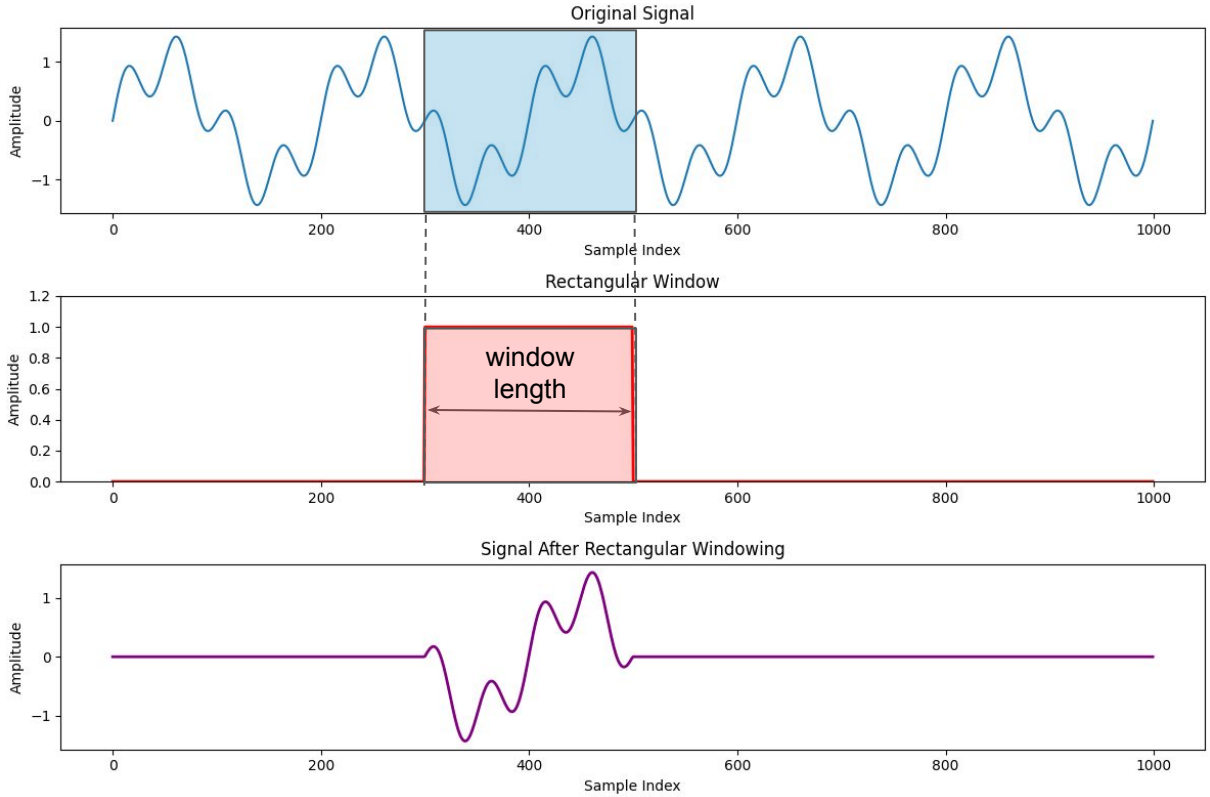
- Window Function






```
windows = {
    'Rectangular': np.ones(length),
    'Hann': np.hanning(length),
    'Hamming': np.hamming(length),
    'Blackman': np.blackman(length),
    'Bartlett': np.bartlett(length),
    'Gaussian': signal.windows.gaussian(length, std=length/5),
    'Kaiser': signal.windows.kaiser(length, beta=14),
    'Flattop': signal.windows.flattop(length)
}
```

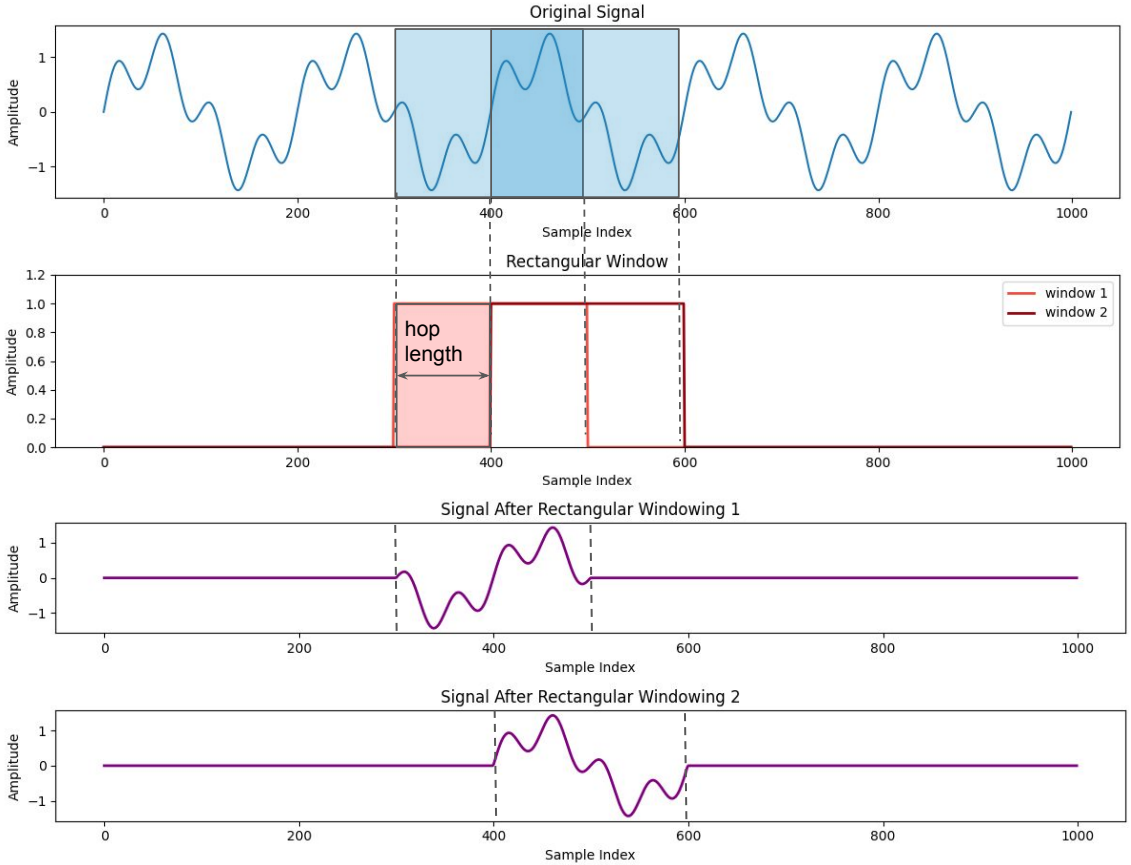
STFT: window function



Important parameters:

- Window Function
- Window Length

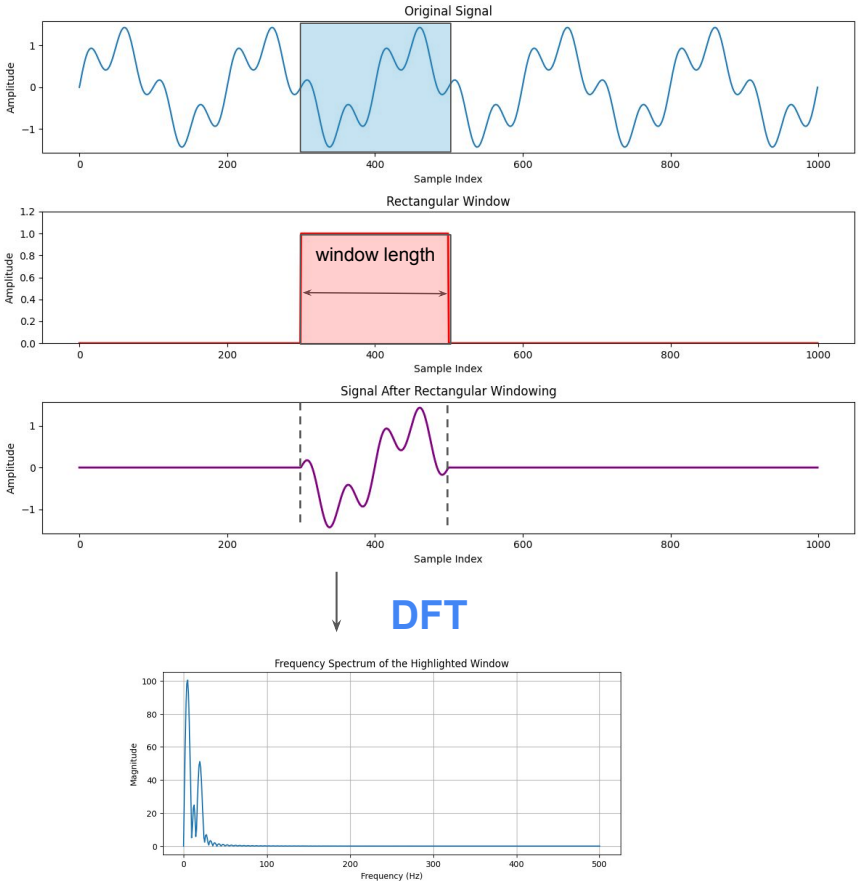
STFT: window function



Important parameters:

- Window function
- Window Length
- HOP Length

STFT: Discrete Fourier Transform



Important parameters:

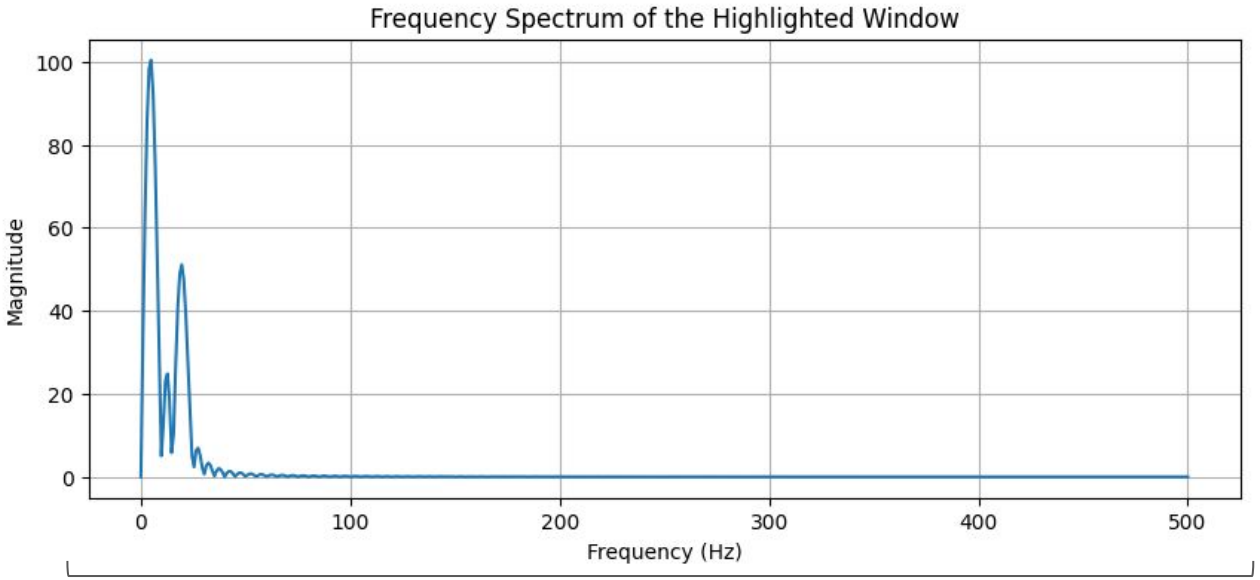
- Window function
- Window Length
- HOP Length
- n_fft

Step 2.

Each windowed segment (e.g., window length = 256 samples) is then transformed into the frequency domain using the **DFT**.

`n_fft` specifies the **length of the DFT**, i.e., how many frequency bins you want in the output.

STFT: Discrete Fourier Transform

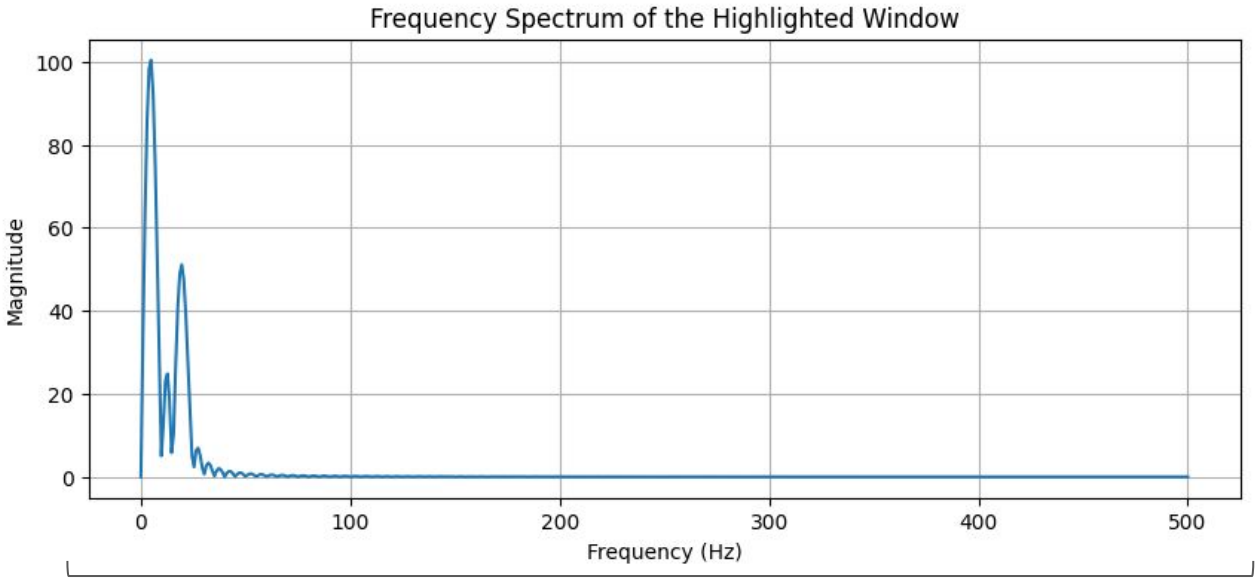


The frequency axis goes from **0 Hz to the Nyquist frequency.**

```
f_max = sample_rate / 2
```

Here: sample_rate = 1000 → f_max = 500 Hz

STFT: Discrete Fourier Transform

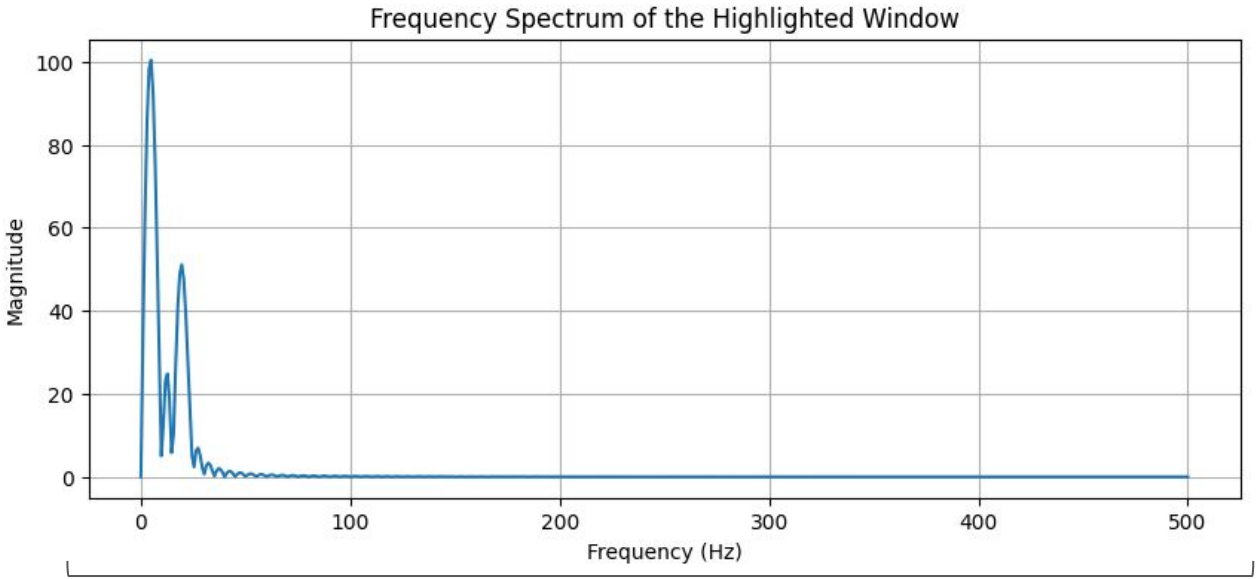


`n_fft` controls how many frequency bins are computed per time frame,

i.e. how many points between 0 Hz and f_max Hz.

```
frequencies = np.linspace(0, sample_rate / 2, n_fft // 2 + 1)
```

STFT: Discrete Fourier Transform



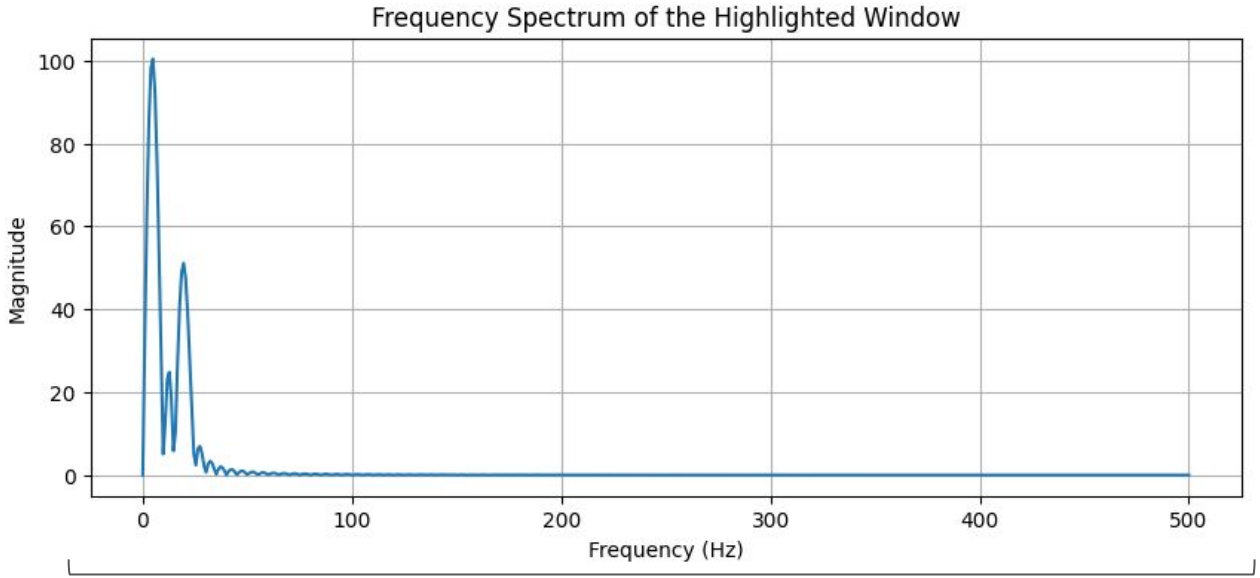
`n_fft` controls how many frequency bins are computed per time frame,

i.e. how many points between 0 Hz and f_max Hz.

```
frequencies = np.linspace(0, sample_rate / 2, n_fft // 2 + 1)
```

due to symmetry,
return just half of it.

STFT: Discrete Fourier Transform




```
sample_rate = 1000
n_fft = 1024
```

frequency values diapason:
[o Hz, sr/2] → [0 Hz, 500 Hz]

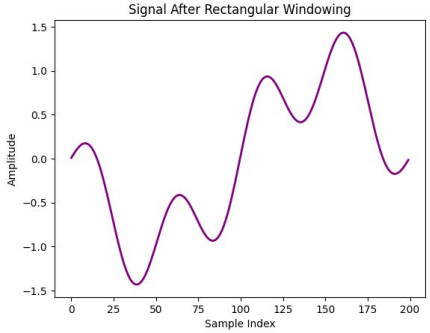
frequency.shape:
[n_fft//2+1,1] → [513,1]

`n_fft` controls how many frequency bins are computed per time frame,
i.e. how many points between 0 Hz and f_max Hz.

```
frequencies = np.linspace(0, sample_rate / 2, n_fft // 2 + 1)
```

due to symmetry,
return just half of it.

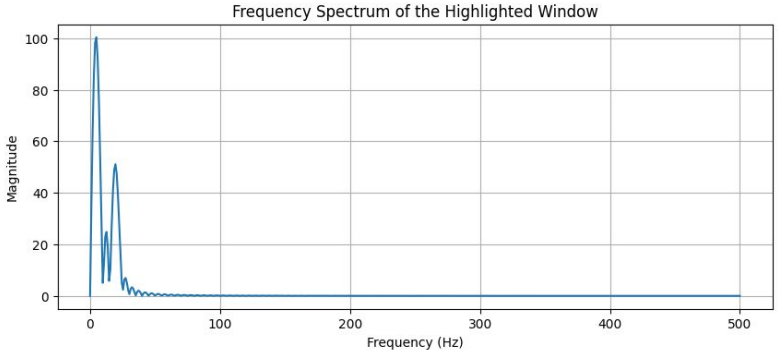
STFT: Discrete Fourier Transform



(200, 1)

```
win_length = 200
sample_rate = 1000
n_fft = 1024
```

DFT

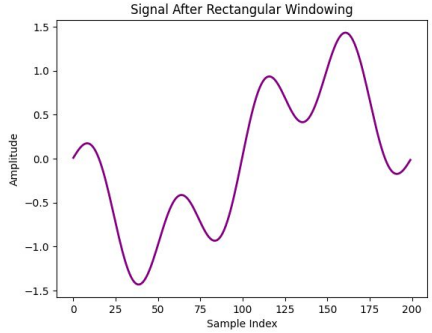


(513,1)

Zero-padding:

- If your window length (`win_length`) is **less than** `n_fft`, the windowed segment is **zero-padded** to length `n_fft` before applying the DFT.
- Example: `win_length = 256`, `n_fft = 512` → each 256-sample window is padded with 256 zeros before the DFT.
- This increases **frequency resolution in the display** (more bins), but **does not add real information**—it just interpolates the spectrum.

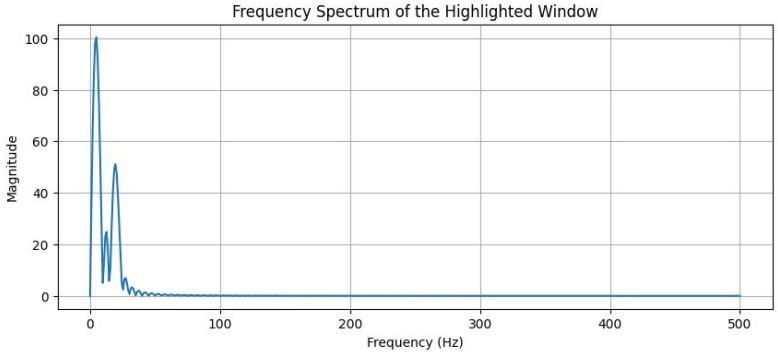
STFT: Discrete Fourier Transform



(200, 1)

```
win_length = 200
sample_rate = 1000
n_fft = 1024
```

DFT

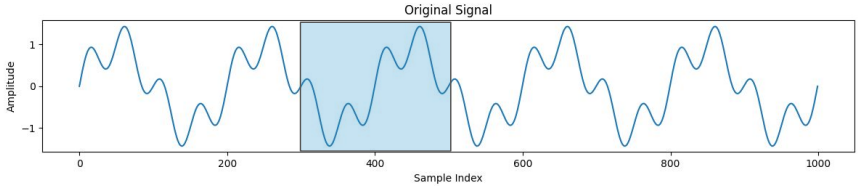


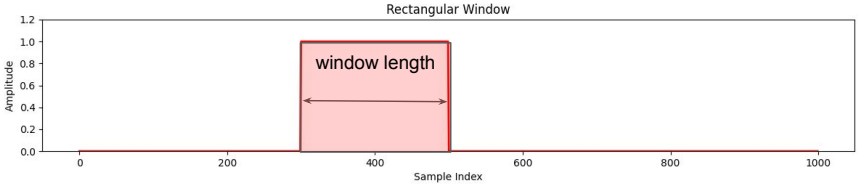
(513,1)

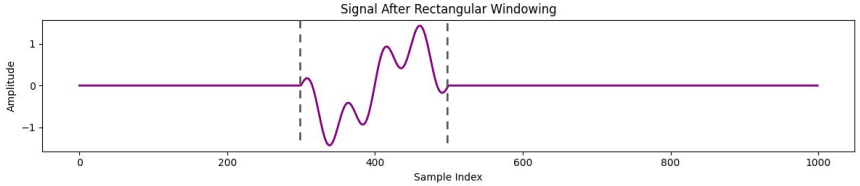
Summary:

`n_fft` controls how many frequency bins are computed per time frame. It can be larger than the actual window length (via zero-padding), which interpolates the spectrum but doesn't increase true frequency resolution. True resolution depends on the **window duration**, not `n_fft`.

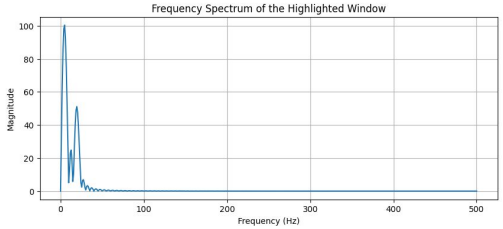
STFT: Discrete Fourier Transform

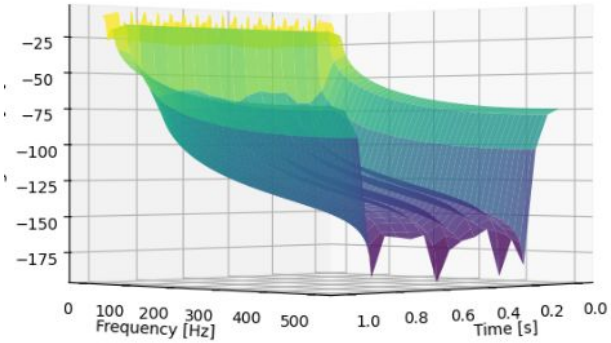


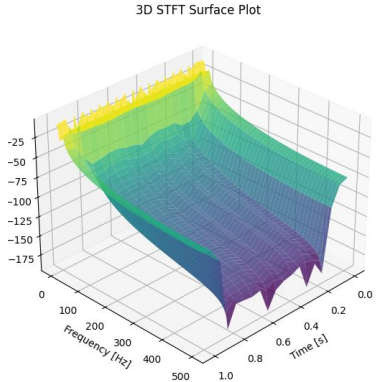


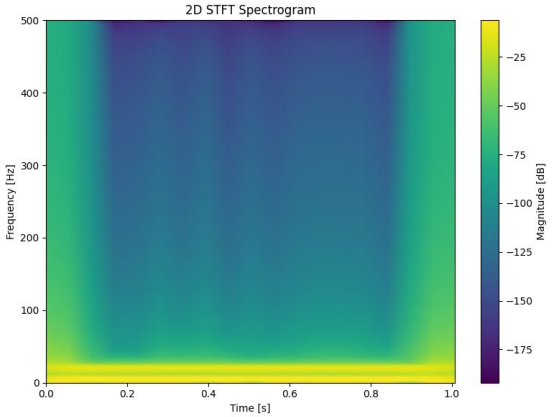


DFT

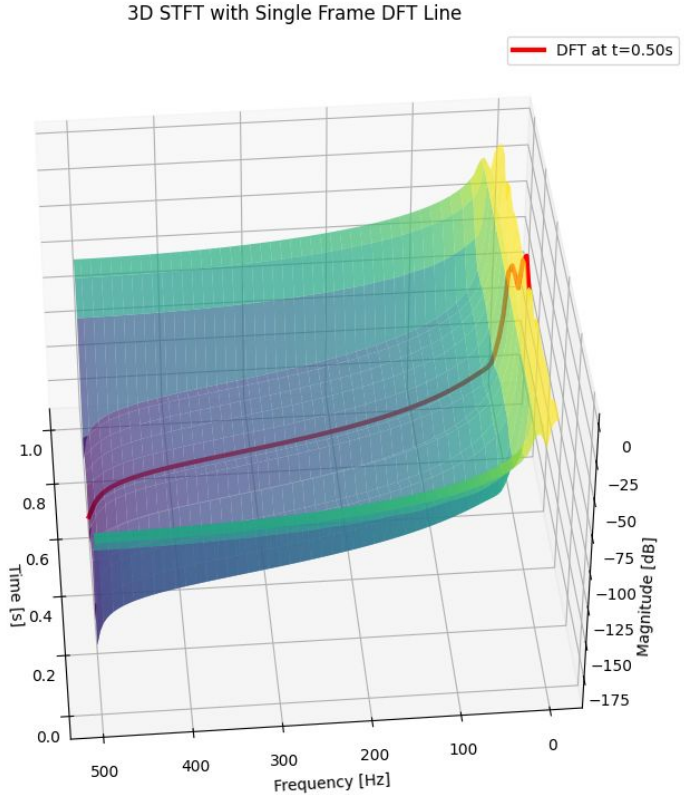


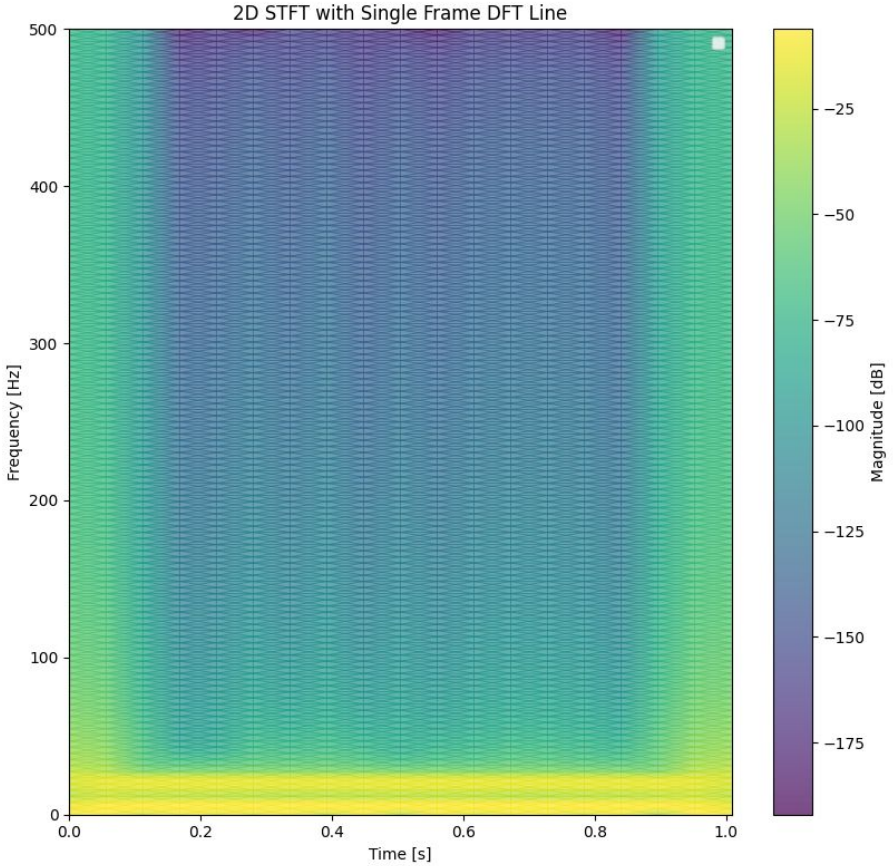




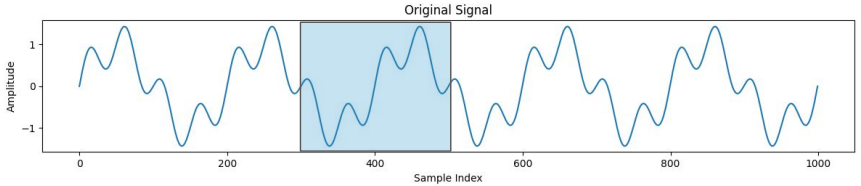


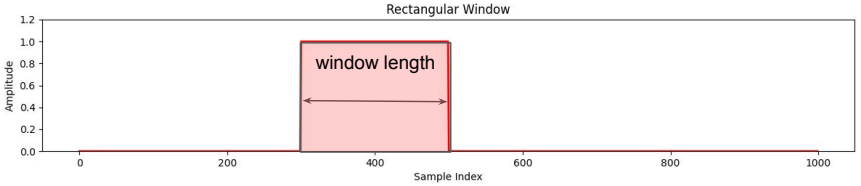
STFT: Discrete Fourier Transform

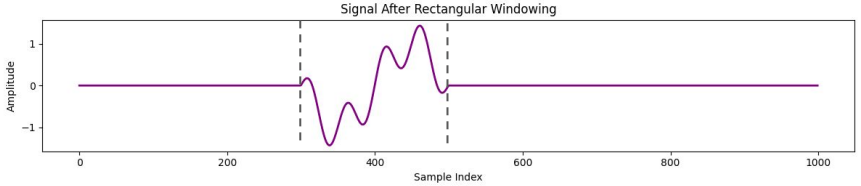




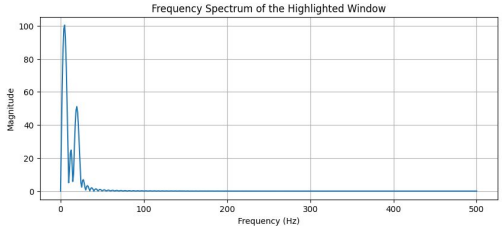
STFT: Discrete Fourier Transform





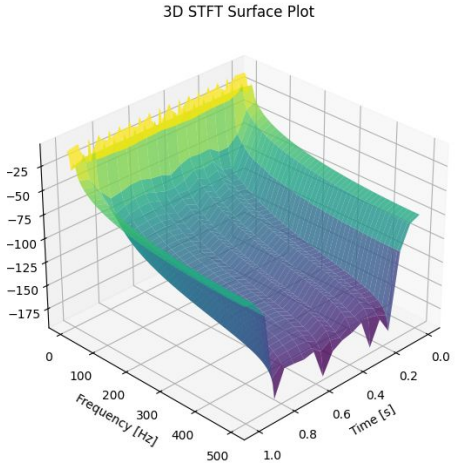


DFT

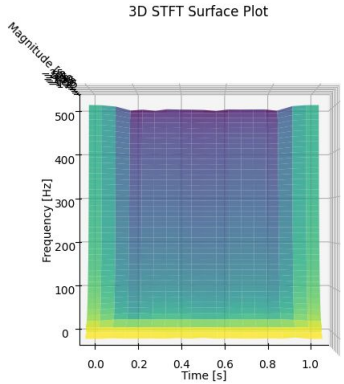


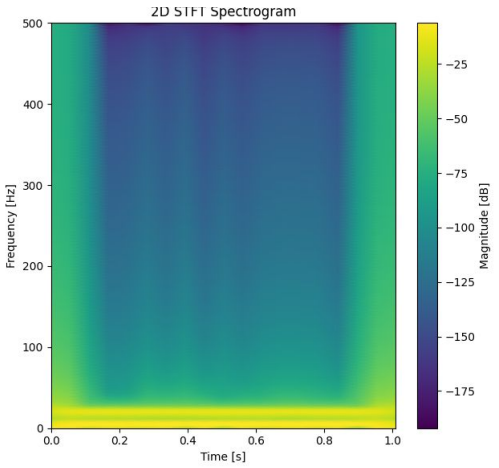
Important parameters:

- Window function
- Window Length
- HOP Length
- n_fft

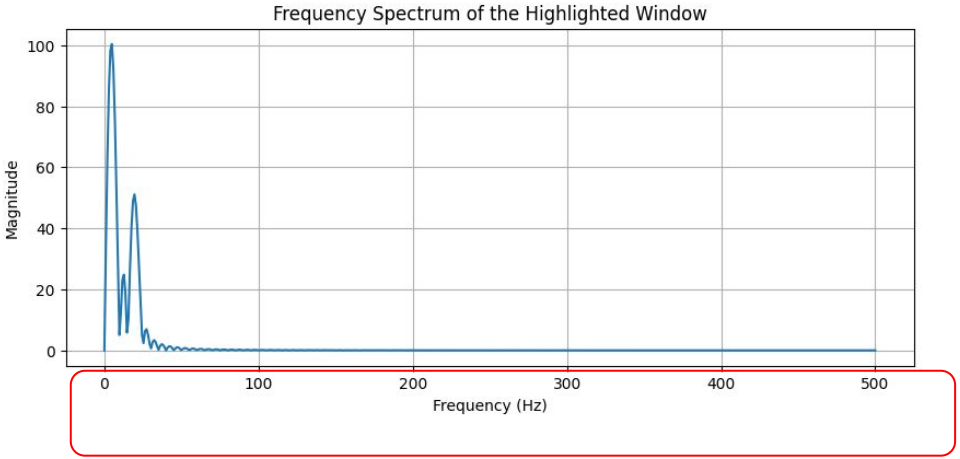


view from the top

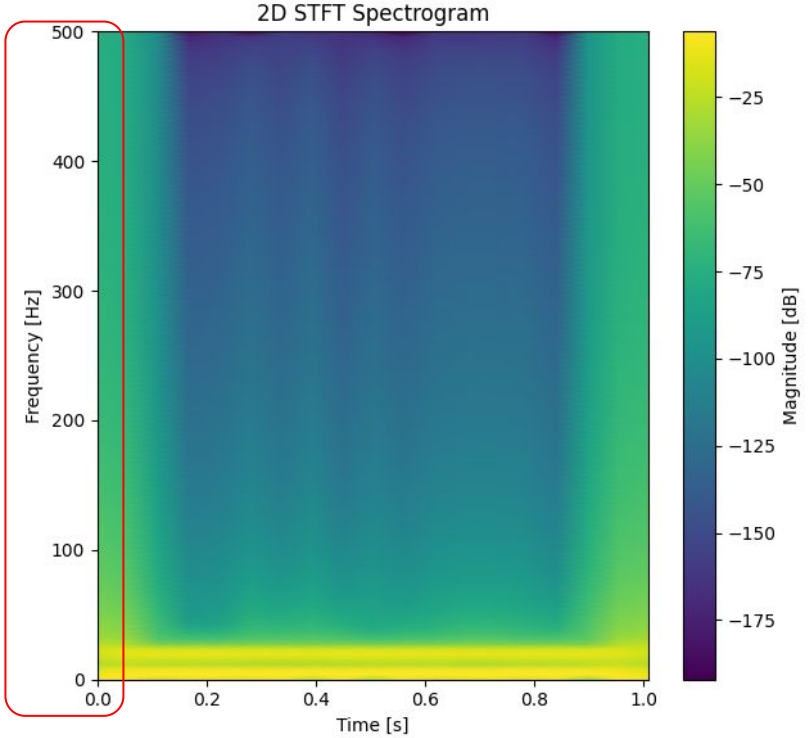




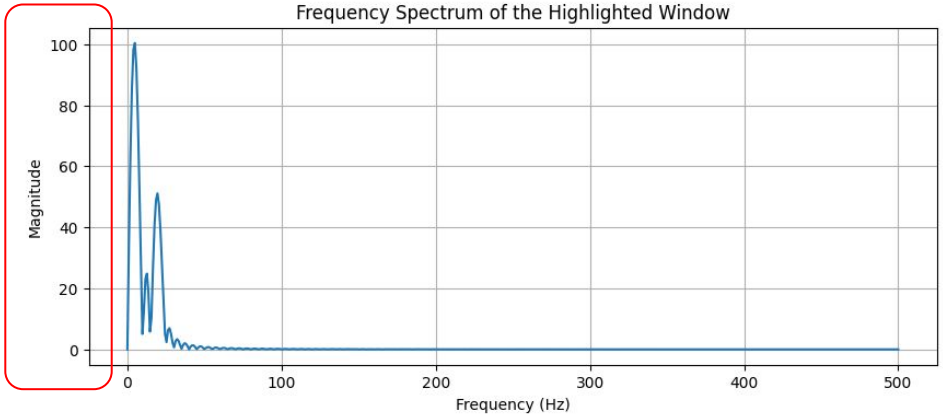
STFT: Discrete Fourier Transform

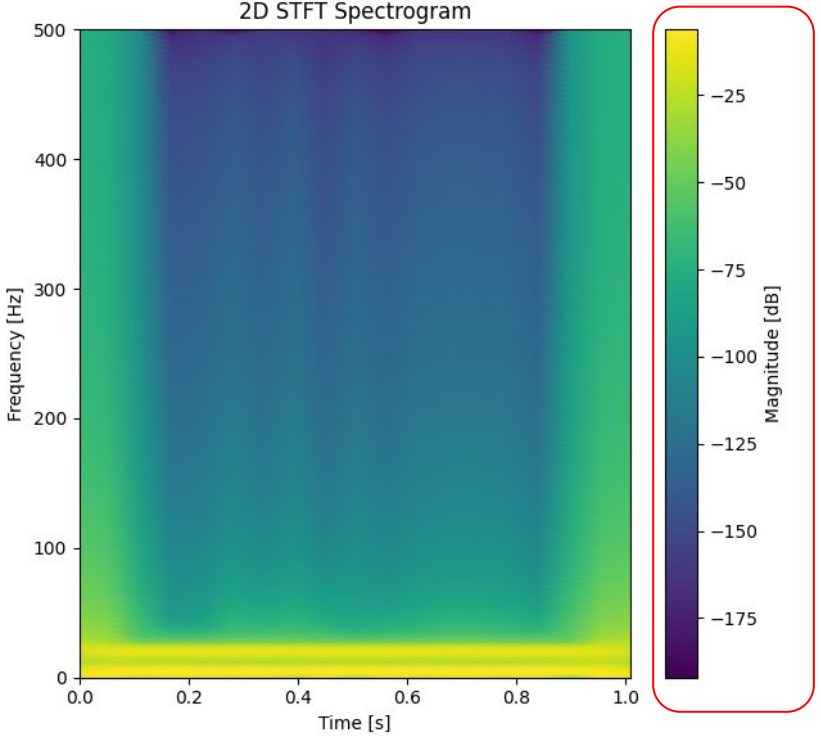


They are the same!



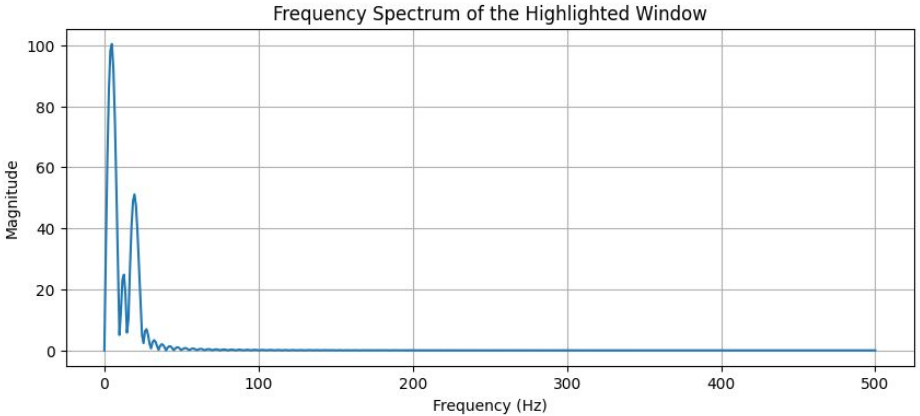
STFT: Discrete Fourier Transform





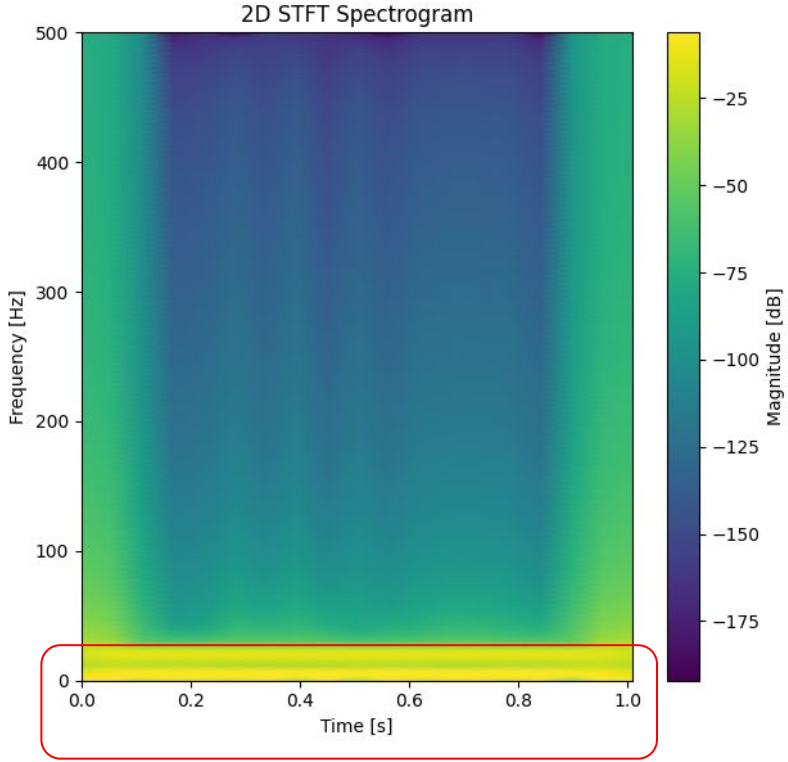
They are different!
Magnitude is normalized and rescaled implicitly in `stft()`.

STFT: Discrete Fourier Transform

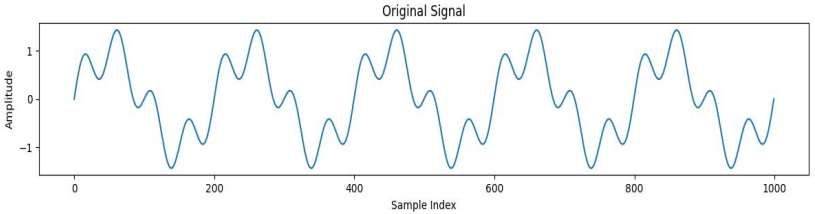


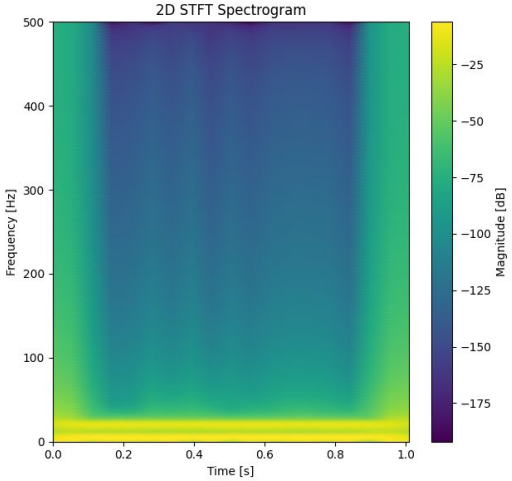
$$\text{Time frames} = \frac{(\text{number of samples} - \text{win\_length})}{\text{hop length}}$$

Here time is in seconds. But usually it remains in time frames.



Short Time Fourier Transform (STFT)





Summary:

Step 1. the input signal is divided into short, overlapping segments (windows).

Step 2. Each windowed segment (e.g., 256 samples) is then transformed into the frequency domain using the **DFT.**

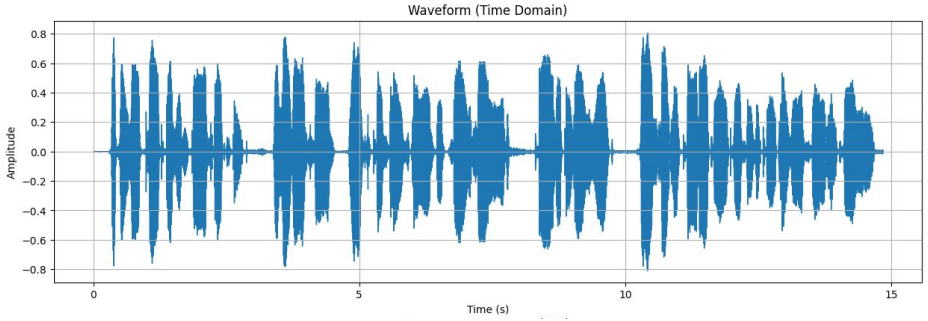
Step 3. Plot all DFT frames all together into one spectral matrix.

```
Spectral matrix shape: [frequency_bins, frames]

frequency_bins = n_fft // 2 + 1
frames = (n_samples - win_length)/ hop_length


Default in librosa:
hop_length = win_length // 4
win_length = n_fft
```

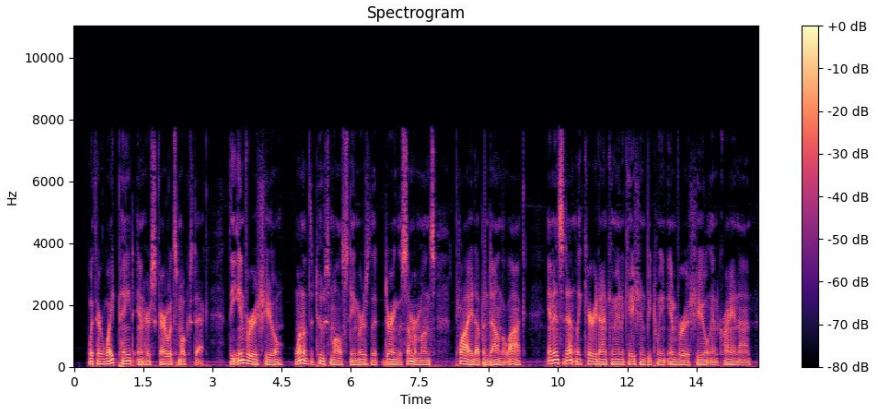
STFT with Librosa

```
import numpy as np

y, sr = librosa.load(librosa.ex('libri1'))

D = librosa.stft(y)
S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)

plt.figure().set_figwidth(12)
librosa.display.specshow(S_db, x_axis="time", y_axis="hz")
plt.colorbar(format='%+2.0f dB')
plt.title("Spectrogram")
plt.show()
```

Here is a real speech.



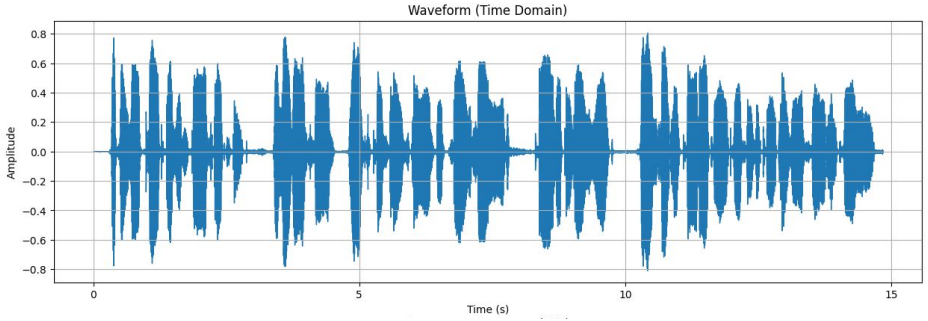
STFT



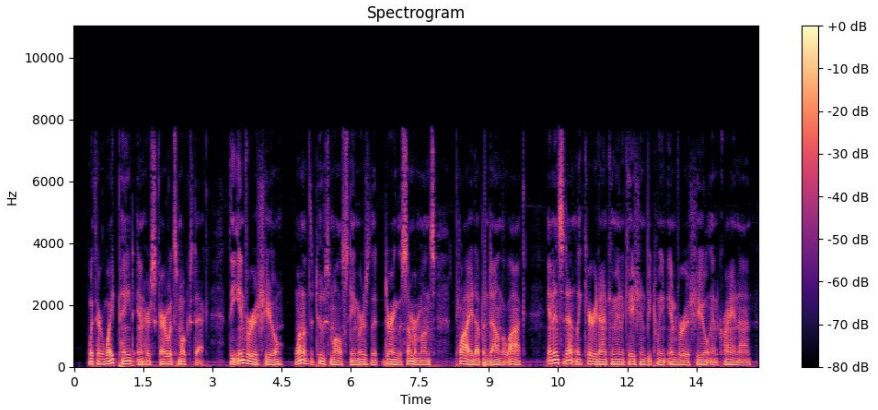
STFT with Torchaudio

```python
import torchaudio
import torchaudio.transforms as T
import matplotlib.pyplot as plt
import torch

# Load an audio file
filename = "your_audio.wav"
waveform, sample_rate = torchaudio.load(filename)

# Create Spectrogram transform with detailed parameters
spectrogram_transform = T.Spectrogram(
    n_fft=400,            # Size of FFT: 400 samples
    win_length=None,      # Window size: defaults to n_fft (400)
    hop_length=None,      # Hop length: defaults to win_length // 2 (200)
    pad=0,                # Padding: 0 samples
    window_fn=torch.hann_window,  # Window function: Hann window
    power=2.0,            # Power exponent: 2 for power spectrogram
    normalized=False,     # Normalization: disabled
    center=True,          # Center padding: enabled
    pad_mode='reflect',   # Padding mode: reflection
    onesided=True         # Onesided: True for real signals
)

# Apply the transform
spec = spectrogram_transform(waveform)

# Convert to decibels for better visualization
db_transform = T.AmplitudeToDB()
spec_db = db_transform(spec)

# Display the spectrogram
plt.figure(figsize=(10, 4))
plt.imshow(spec_db[0].numpy(), cmap='magma', origin='lower', aspect='auto')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram')
plt.ylabel('Frequency Bin')
plt.xlabel('Time Frame')
plt.tight_layout()
plt.show()
```

Here is a real speech.

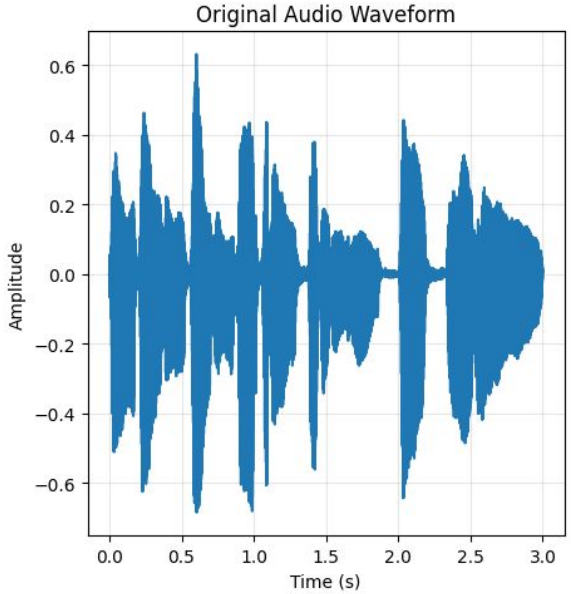


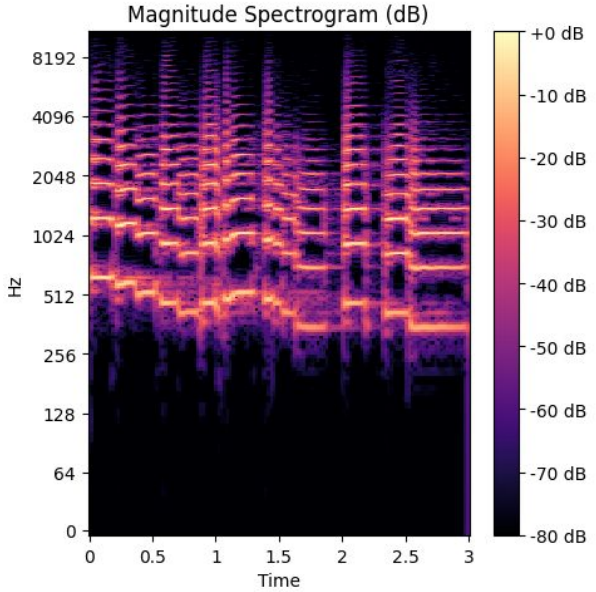
STFT

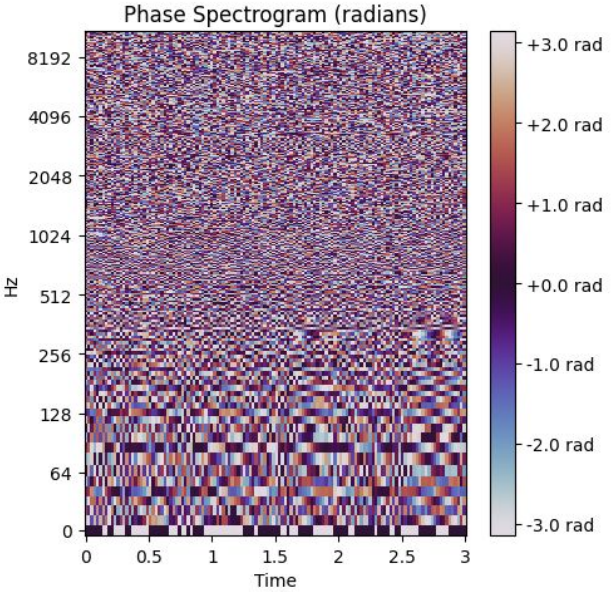


STFT: Magnitude and Phase

The output of STFT is a complex matrix. Usually we do not use phase values and just look at the magnitude.

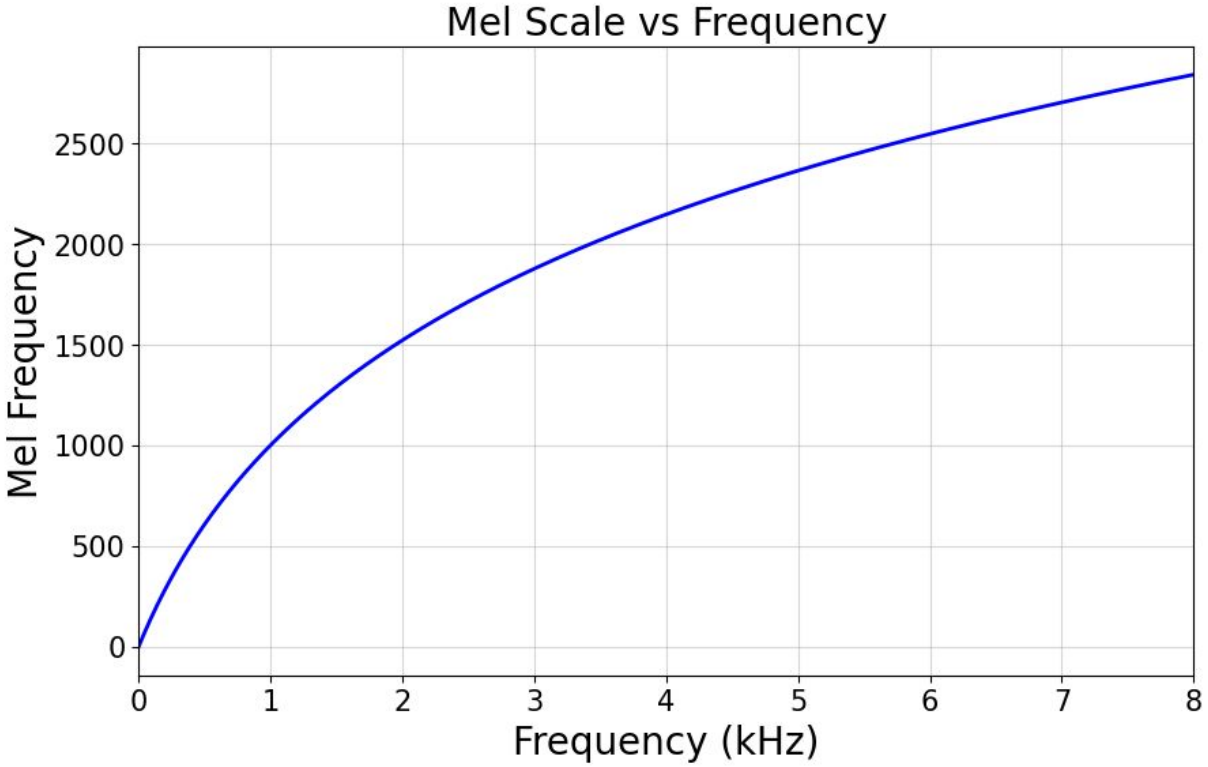






Mel-Spectrograms

Human Perception: Mel-scale



We created this Mel-scale doing psychological experiments with human perception.

Empirical formula:

$$m = 2595 \cdot \log\left(1 + \frac{f}{700}\right)$$

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

"**Mel**" is from the word "**melody**", because melody is connected with the concept of pitch.

How to create Mel-Spectrograms from Spectrogram

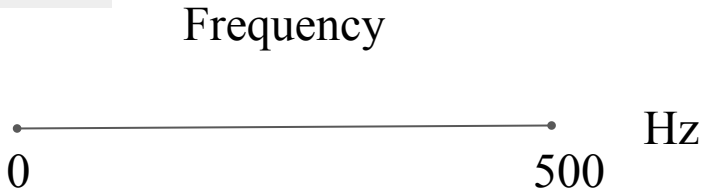
Let’s consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$



How to create Mel-Spectrograms from Spectrogram

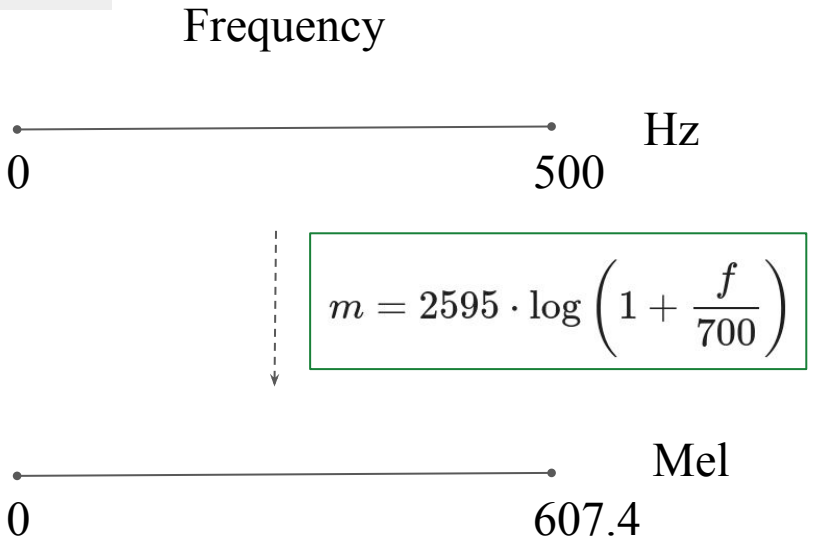
Let’s consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$



What is a Mel Bands?

Let’s consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

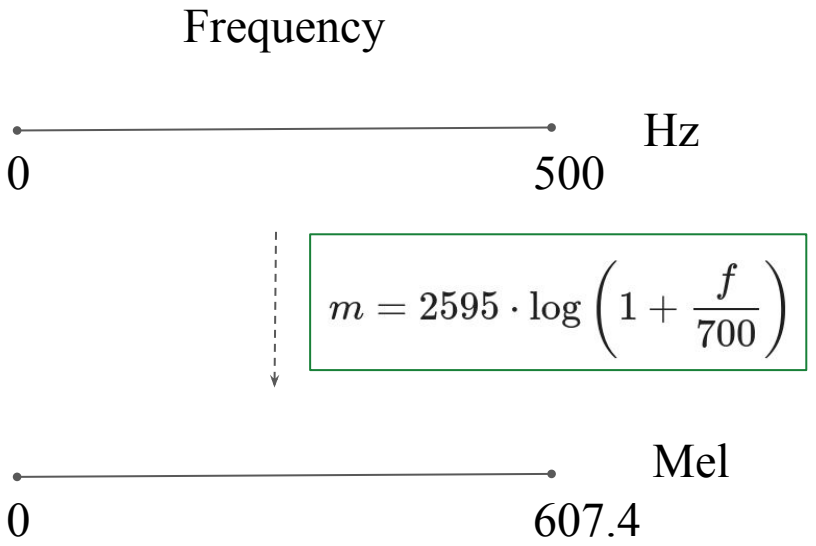
$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let’s set:

$$\text{n\_mels} = 2$$



What is a Mel Bands?

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

$$f_{min} = 0\text{Hz}$$

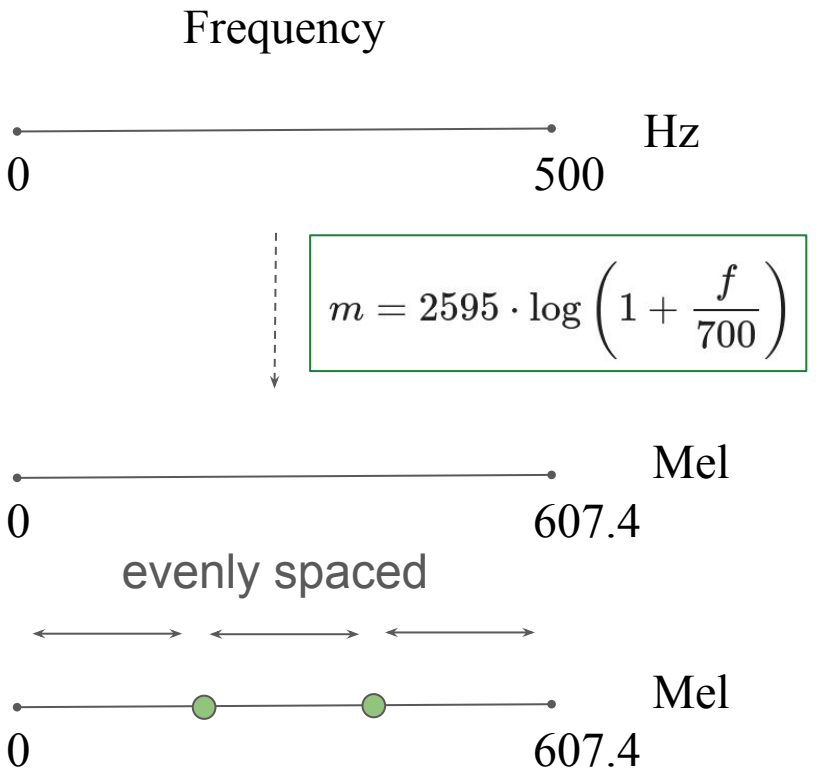
$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$

how many points between f_min and f_max we will set



What is a Mel Bands?

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

$$f_{min} = 0\text{Hz}$$

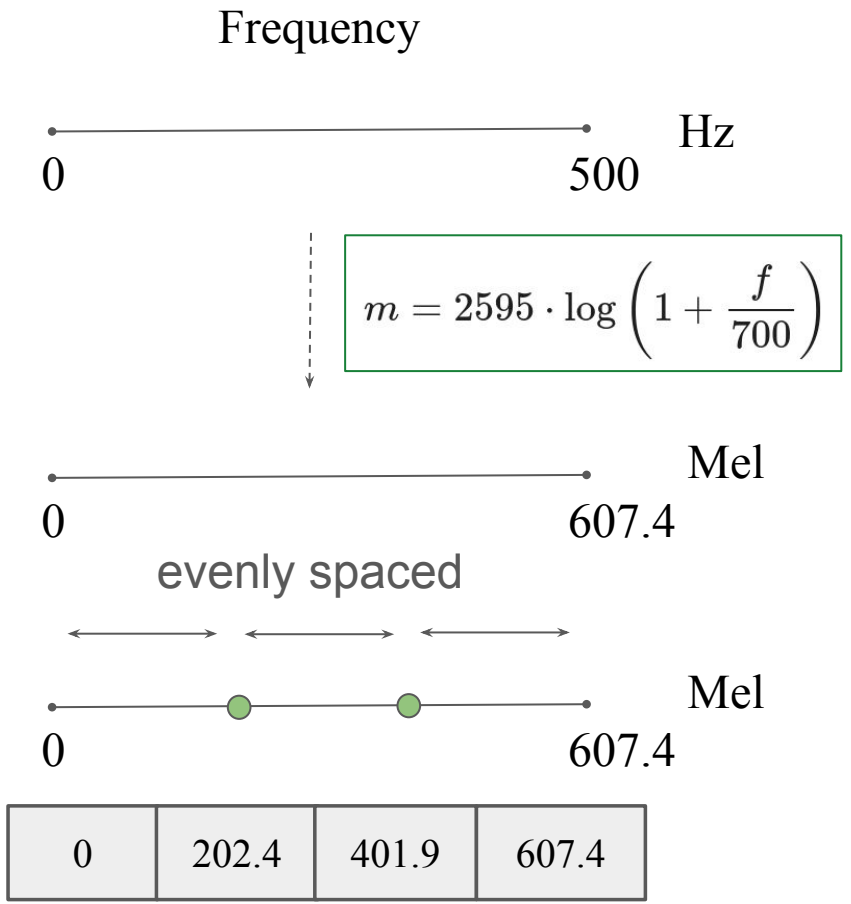
$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$

how many points between f_min and f_max we will set



What is a Mel Bands?

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$

how many points between f_min and f_max we will set

0	202.4	401.9	607.4	Mel

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

0	137.7	302.6	500	Hz

Remainder what is a n_fft

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

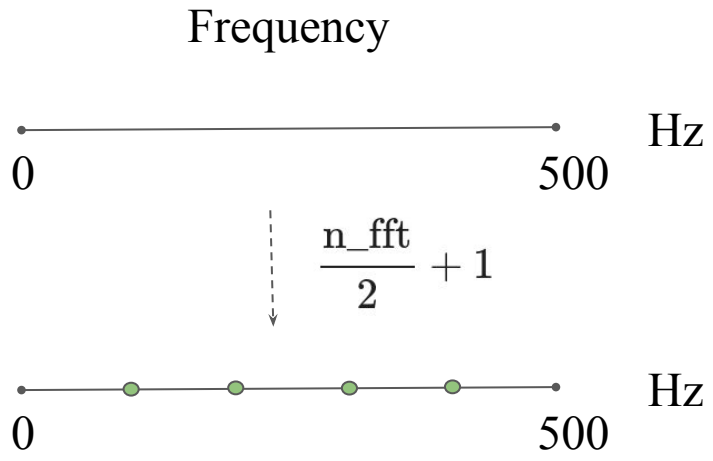
$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$



0	100	200	300	400	500

fft bins

0	1	2	3	4	5

fft bin indices

Remainder what is a n_fft

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

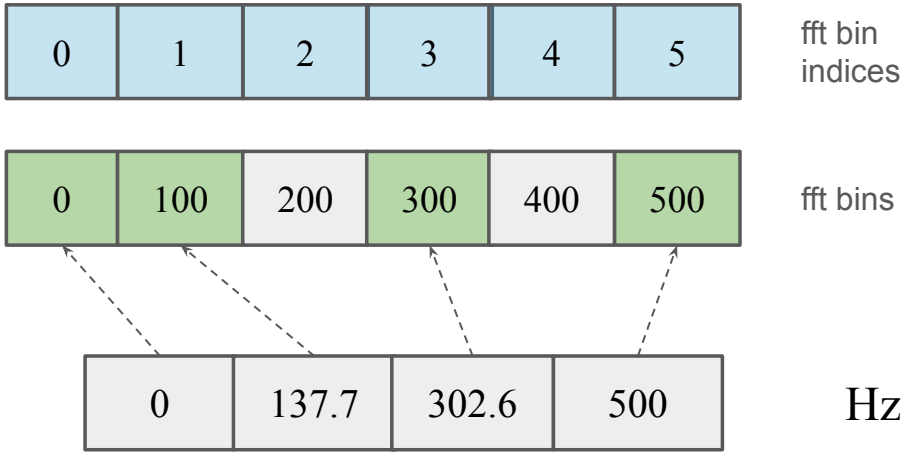
$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$



Remainder what is a n_fft

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

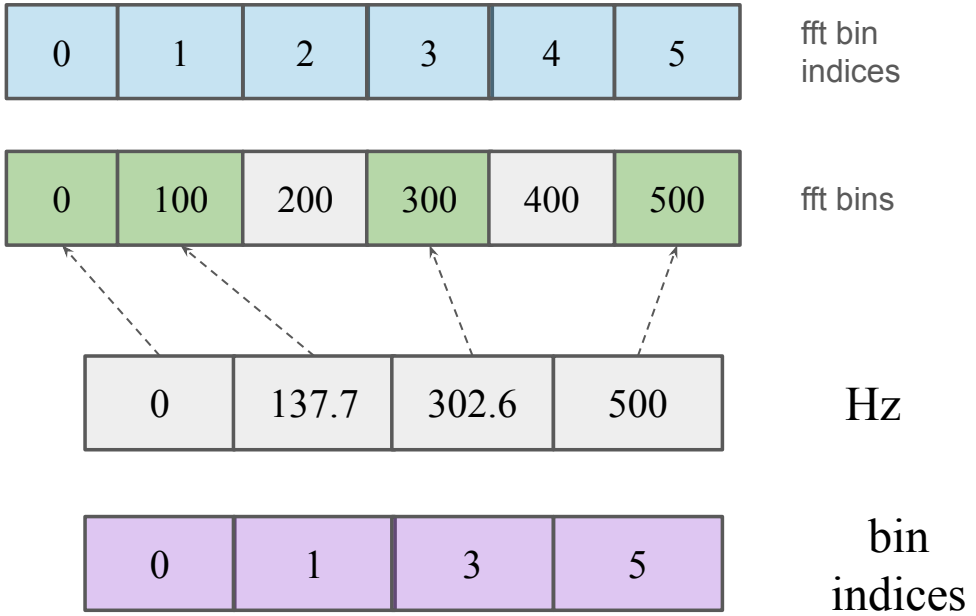
$$f_{min} = 0\text{Hz}$$

$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$



Mel-filterbanks

Let's consider simple example:

$$\text{sample\_rate} = 1000\ \text{Hz}$$

$$f_{min} = 0\text{Hz}$$

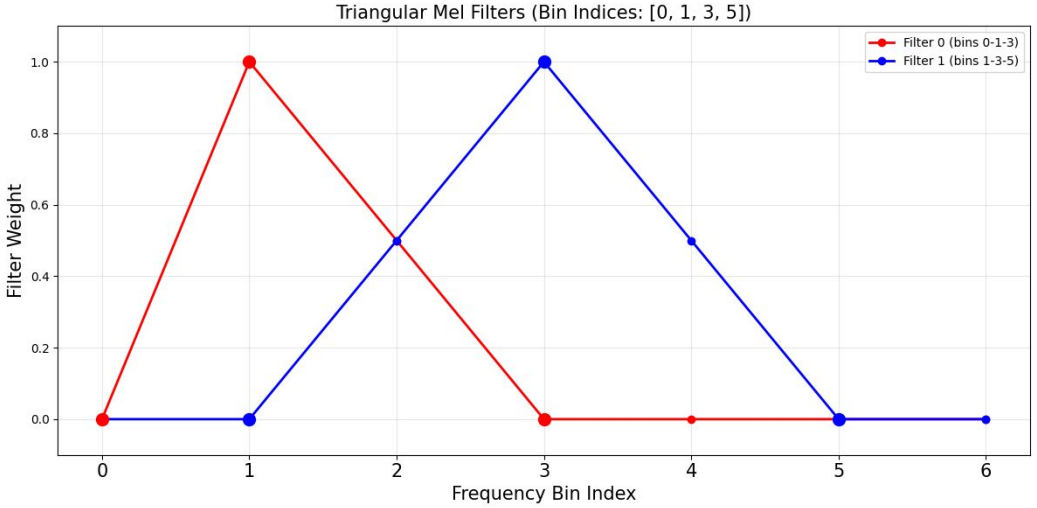
$$f_{max} = \frac{\text{sample\_rate}}{2} = 500\text{Hz}$$

$$\text{n\_fft} = 10$$

Let's set:

$$\text{n\_mels} = 2$$

0	137.7	302.6	500	Hz
0	1	3	5	bin indices



Mel-filterbanks

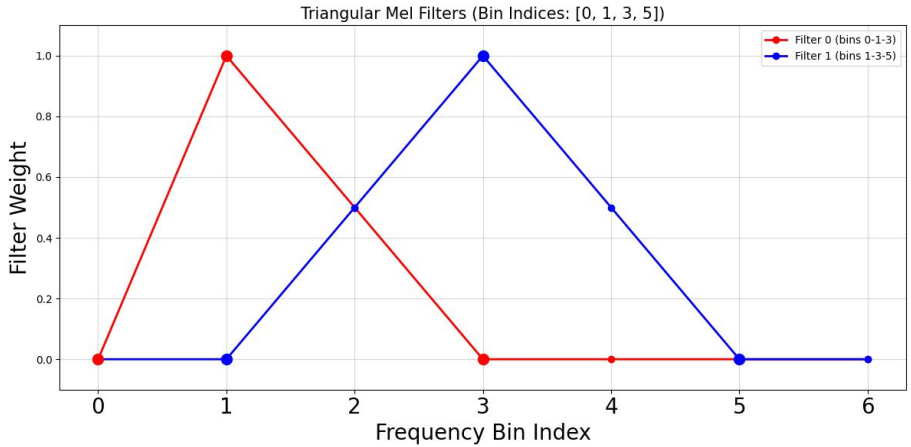
Let’s create a filter bank.

$$\text{n\_fft} = 10$$

$$\text{n\_mels} = 2$$

$$\frac{\text{n\_fft}}{2} + 1$$

$$\text{n\_mels} = 2$$



Mel-filterbanks

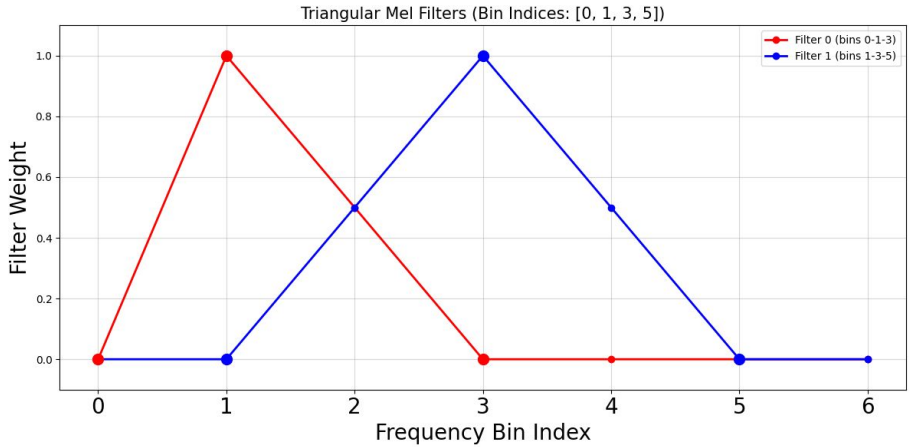
Let's create a filter bank.

$$\text{n\_fft} = 10$$

$$\text{n\_mels} = 2$$

$$\frac{\text{n\_fft}}{2} + 1$$

$$\text{n\_mels} = 2$$




```python
for i in range(n_mels):
    left = bin_indices[i]      # f_low bin index
    center = bin_indices[i+1]  # f_center bin index
    right = bin_indices[i+2]   # f_high bin index

    # Create rising slope (left to center)
    if center > left:  # Avoid division by zero
        for j in range(left, center):
            filter_bank[i, j] = (j - left) / (center - left)

    # Create falling slope (center to right)
    if right > center:  # Avoid division by zero
        for j in range(center, right):
            filter_bank[i, j] = 1 - (j - center) / (right - center)
```

Mel-filterbanks

$$\text{n\_fft} = 10$$

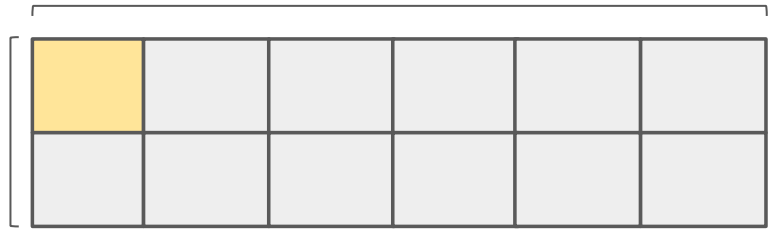
$$\text{n\_mels} = 2$$

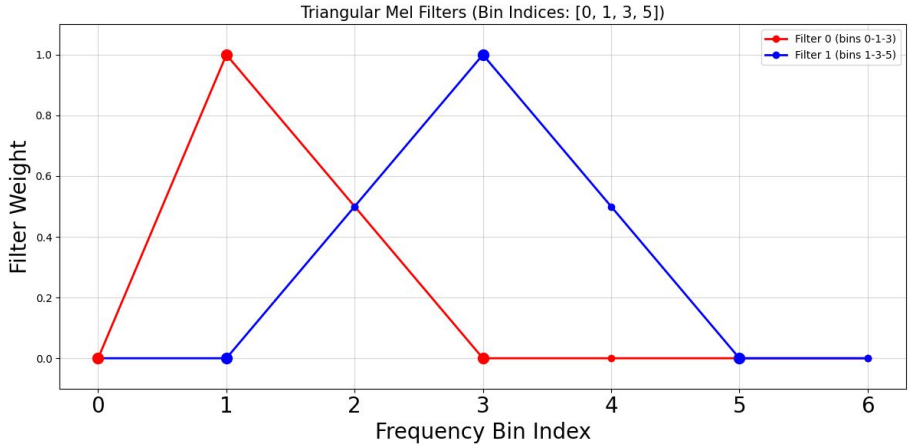
bin indices

Let's create a filter bank.

$$\frac{\text{n\_fft}}{2} + 1$$

$$\text{n\_mels} = 2$$






```python
for i in range(n_mels):
    left = bin_indices[i]      # f_low bin index
    center = bin_indices[i+1]  # f_center bin index
    right = bin_indices[i+2]   # f_high bin index

    # Create rising slope (left to center)
    if center > left:  # Avoid division by zero
        for j in range(left, center):
            filter_bank[i, j] = (j - left) / (center - left)

    # Create falling slope (center to right)
    if right > center:  # Avoid division by zero
        for j in range(center, right):
            filter_bank[i, j] = 1 - (j - center) / (right - center)
```

Mel-filterbanks

$n\_fft = 10$

$n\_mels = 2$

bin indices

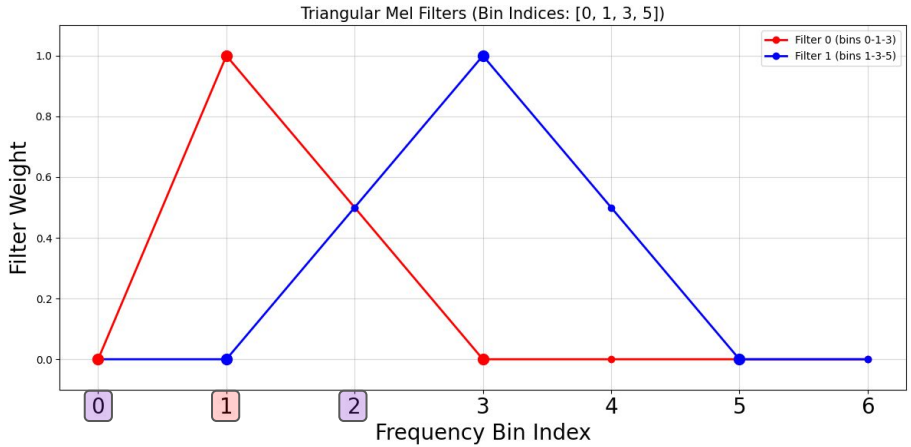
0	1	3	5

Let's create a filter bank.

$$\frac{n\_fft}{2} + 1$$

$n\_mels = 2$

0					



fft bin indices

0	1	2	3	4	5

```
for i in range(n_mels):
    left = bin_indices[i]      # f_low bin index
    center = bin_indices[i+1]  # f_center bin index
    right = bin_indices[i+2]   # f_high bin index

    # Create rising slope (left to center)
    if center > left:  # Avoid division by zero
        for j in range(left, center):
            filter_bank[i, j] = (j - left) / (center - left)

    # Create falling slope (center to right)
    if right > center:  # Avoid division by zero
        for j in range(center, right):
            filter_bank[i, j] = 1 - (j - center) / (right - center)
```

Mel-filterbanks

Let's create a filter bank.

$$\text{n\_fft} = 10$$

$$\text{n\_mels} = 2$$

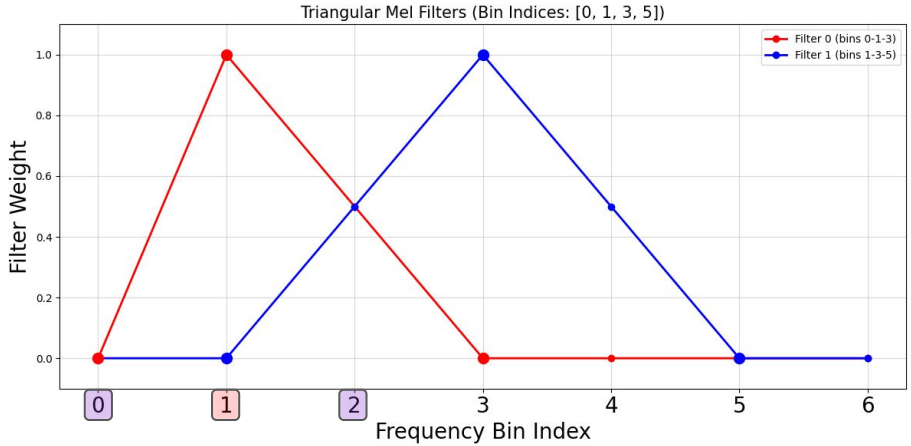
bin indices

0	1	3	5

$$\frac{\text{n\_fft}}{2} + 1$$

$$\text{n\_mels} = 2$$

0	1				



fft bin indices

0	1	2	3	4	5

$$1 - \frac{1-1}{3-1} = 1$$

```python
for i in range(n_mels):
    left = bin_indices[i]      # f_low bin index
    center = bin_indices[i+1]  # f_center bin index
    right = bin_indices[i+2]   # f_high bin index

    # Create rising slope (left to center)
    if center > left:  # Avoid division by zero
        for j in range(left, center):
            filter_bank[i, j] = (j - left) / (center - left)

    # Create falling slope (center to right)
    if right > center:  # Avoid division by zero
        for j in range(center, right):
            filter_bank[i, j] = 1 - (j - center) / (right - center)
```

Mel-filterbanks

Let's create a filter bank.

$$\text{n\_fft} = 10$$

$$\text{n\_mels} = 2$$

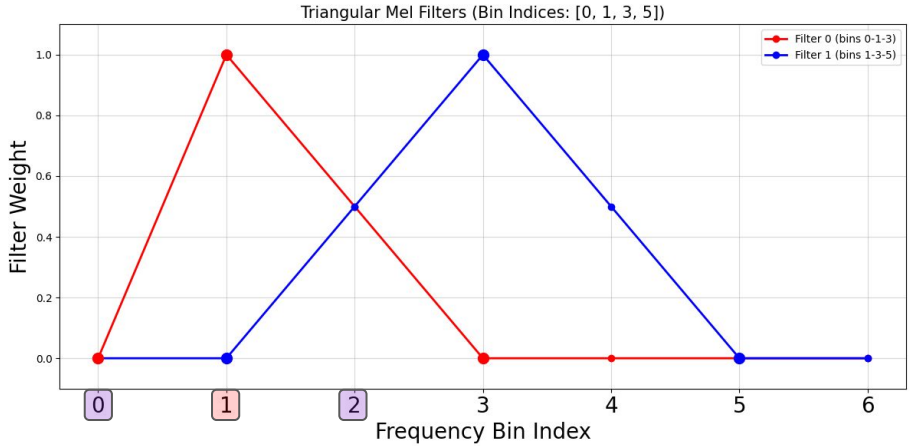
bin indices

0	1	3	5

$$\frac{\text{n\_fft}}{2} + 1$$

$$\text{n\_mels} = 2$$

0	1	0.5			



fft bin indices

0	1	2	3	4	5

```
for i in range(n_mels):
    left = bin_indices[i]      # f_low bin index
    center = bin_indices[i+1]  # f_center bin index
    right = bin_indices[i+2]   # f_high bin index

    # Create rising slope (left to center)
    if center > left:  # Avoid division by zero
        for j in range(left, center):
            filter_bank[i, j] = (j - left) / (center - left)

    # Create falling slope (center to right)
    if right > center:  # Avoid division by zero
        for j in range(center, right):
            filter_bank[i, j] = 1 - (j - center) / (right - center)
```

$$1 - \frac{2-1}{3-1} = 0.5$$

Mel-filterbanks

$$\text{n\_fft} = 10$$

$$\text{n\_mels} = 2$$

bin indices

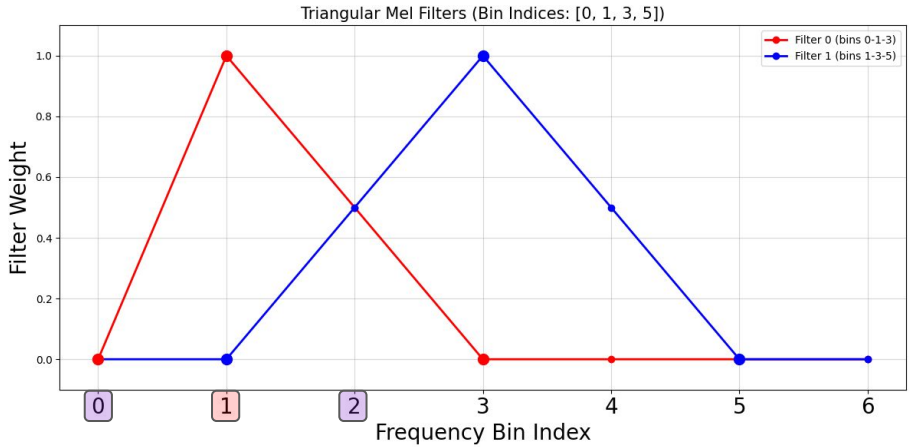
0	1	3	5

Let's create a filter bank.

$$\frac{\text{n\_fft}}{2} + 1$$

$$\text{n\_mels} = 2$$

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0



fft bin indices

0	1	2	3	4	5

```
for i in range(n_mels):
    left = bin_indices[i]      # f_low bin index
    center = bin_indices[i+1]  # f_center bin index
    right = bin_indices[i+2]   # f_high bin index

    # Create rising slope (left to center)
    if center > left:  # Avoid division by zero
        for j in range(left, center):
            filter_bank[i, j] = (j - left) / (center - left)

    # Create falling slope (center to right)
    if right > center:  # Avoid division by zero
        for j in range(center, right):
            filter_bank[i, j] = 1 - (j - center) / (right - center)
```

Mel-filterbanks

Filter Bank

$\frac{\text{n\_fft}}{2} + 1$

$\text{n\_mels} = 2$

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0

Spectrogram

time frames

Mel-filterbanks

Filter Bank

$\frac{\text{n\_fft}}{2} + 1$

$\text{n\_mels} = 2$

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0

Spectrogram

Time frames

$$\text{Time frames} = \frac{(\text{number of samples} - \text{win\_length})}{\text{hop length}}$$

Mel-filterbanks

Spectrogram

Time frames

Filter Bank

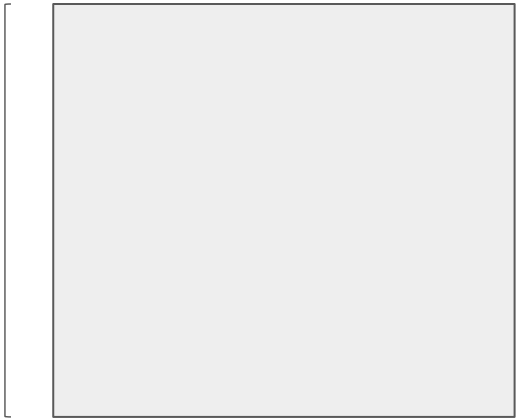
$\frac{\text{n\_fft}}{2} + 1$

$\text{n\_mels} = 2$

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0

$*$

$\frac{\text{n\_fft}}{2} + 1$



$$\text{Time frames} = \frac{(\text{number of samples} - \text{win\_length})}{\text{hop length}}$$

Mel-filterbanks

Filter Bank

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0

$\text{n\_mels} = 2$, $\frac{\text{n\_fft}}{2} + 1$

*

Spectrogram

$\frac{\text{n\_fft}}{2} + 1$, Time frames

=

Mel-Spectrogram

$\text{n\_mels} = 2$, Time frames

Mel-filterbanks

Optional:

Filter Bank

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0

$\text{n\_mels} = 2$, $\frac{\text{n\_fft}}{2} + 1$

*

Spectrogram

$\frac{\text{n\_fft}}{2} + 1$, Time frames

=

log(Mel-Spectrogram)

$\text{n\_mels} = 2$, Time frames

Mel-filterbanks

Optional:

Filter Bank

0	1	0.5	0	0	0
0	0	0.5	1	0.5	0

$\text{n\_mels} = 2$, $\frac{\text{n\_fft}}{2} + 1$

*

Spectrogram

$\frac{\text{n\_fft}}{2} + 1$, Time frames

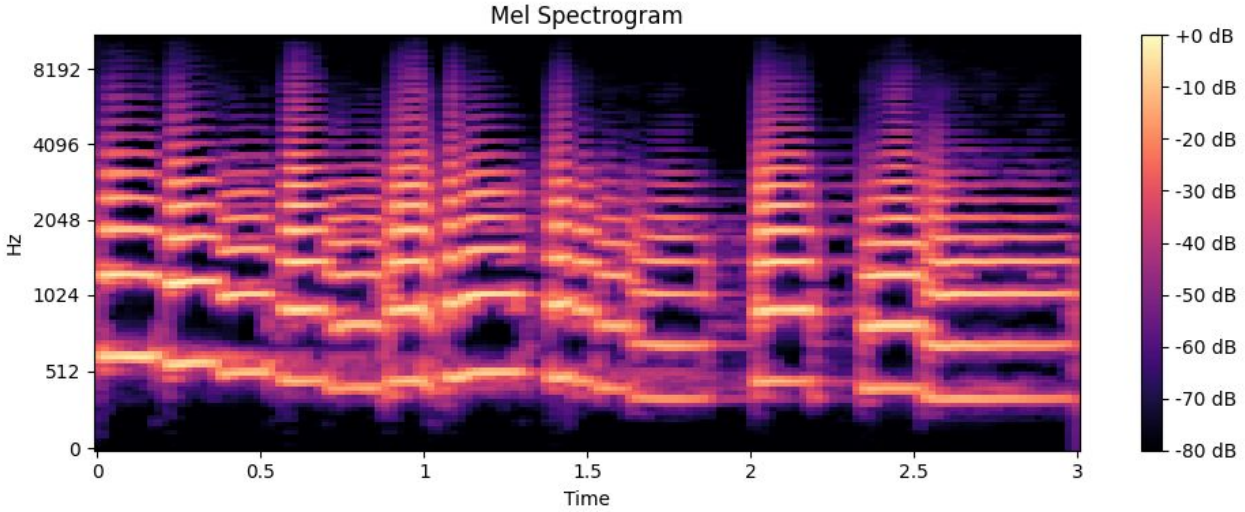
=

Normalize(log(Mel-Spectrogram))

$\text{n\_mels} = 2$, Time frames

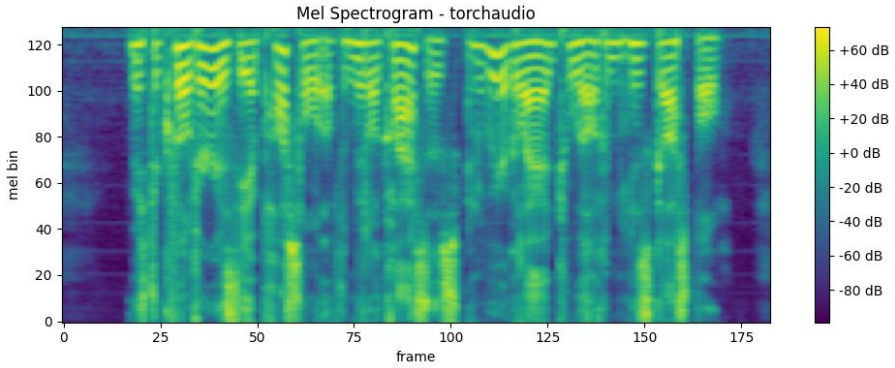
How to plot mel-spectrograms in Librosa?

```
# Load an audio file
y, sr = librosa.load(librosa.ex('trumpet'), duration=3.0)
# Calculate the mel-spectrogram
mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)

# Convert the mel-spectrogram to a decibel (log) scale
mel_spectrogram_db = librosa.power_to_db(mel_spectrogram, ref=np.max)

# Plot the mel-spectrogram
plt.figure(figsize=(10, 4))

librosa.display.specshow(mel_spectrogram_db,
                         x_axis='time',
                         y_axis='mel',
                         sr=sr)
plt.colorbar(format='%+2.0f dB')
plt.title('Mel Spectrogram')
plt.tight_layout()
plt.show()
```



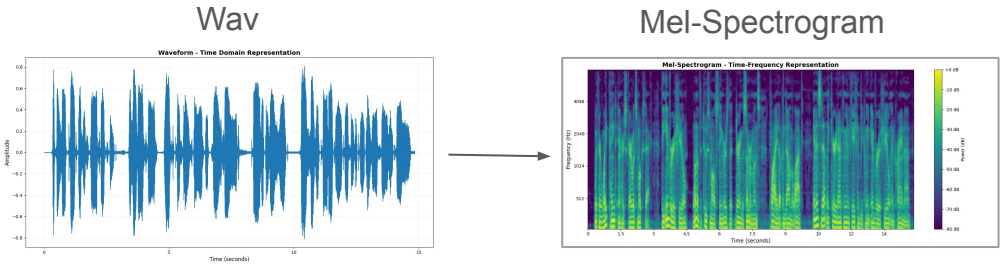
How to plot mel-spectrograms in Torchaudio?

```python
import torch
import torchaudio
import torchaudio.transforms as T
import matplotlib.pyplot as plt

# Load a sample from LibriSpeech dataset
dataset = torchaudio.datasets.LIBRISPEECH("./", url="dev-clean", download=True)
waveform, sample_rate, _, _, _, _ = dataset[0]  # Get first sample

# Define the MelSpectrogram transform
mel_spectrogram_transform = T.MelSpectrogram(
    sample_rate=sample_rate,
    n_fft=2048,
    hop_length=512,
    n_mels=128
)

# Calculate the mel-spectrogram
mel_spec = mel_spectrogram_transform(waveform)

# Convert to decibel scale. Squeeze to remove channel dimension for single audio.
mel_spec_db = torchaudio.functional.amplitude_to_DB(
    mel_spec,
    multiplier=20,
    amin=1e-10,
    db_multiplier=0.0
).squeeze(0)

# Plot the mel-spectrogram
plt.figure(figsize=(10, 4))
plt.imshow(mel_spec_db.flip(0),  # Flip so low frequencies are at the bottom
           aspect='auto',
           origin='lower',
           cmap='viridis')
plt.title("Mel Spectrogram - torchaudio")
plt.ylabel("mel bin")
plt.xlabel("frame")
plt.colorbar(format='%+2.0f dB')
plt.tight_layout()
plt.show()
```

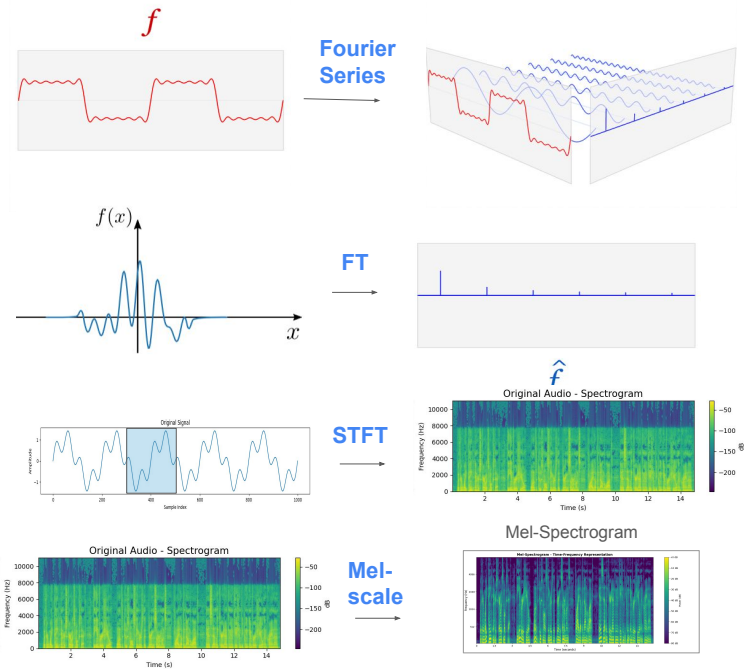


Summary

- Waveform is not convenient, because they are high dimensional with low information density. Spectrograms give you more compact and informative representation.



- From waveform to mel-spectrogram:
  1. Any periodic function can be expand into a Fourier series, which are sinus and cosinus waves.
  2. Fourier transform can calculate frequencies of all sinus waves which form our complex signal.
  3. We use STFT to create dynamic spectrum, and this is how we get a spectrogram.
  4. Then we scale it to the human preference with mel-scale and get Mel-Spectrogram.



DLS

Yermekova Assel

Telegram: @zametki_polukrovki - тут я рассказываю какие то вещи.

Спасибо за внимание!

Если у вас остались вопросы, пишите в чат курса.

2025