Spectrogram of Concatenation of Sinusoids

Objective
To visualize how frequency changes over time when we join (concatenate) several sinusoidal signals, each with a different frequency.

What is Concatenation?

Concatenation just means joining signals one after another in time.
For example:

- From 0 to 1 sec → play 200 Hz tone
- From 1 to 2 sec → play 400 Hz tone
- From 2 to 3 sec → play 600 Hz tone

So the full signal looks like:

$$x(t) = \begin{cases} \sin(2\pi \times 200t), & 0 \le t < 1 \\ \sin(2\pi \times 400t), & 1 \le t < 2 \\ \sin(2\pi \times 600t), & 2 \le t < 3 \end{cases}$$

The frequency changes in steps over time – first low, then medium, then high.

What Does a Spectrogram Do?

Normally, when we just plot a signal x(t) vs time, we only see how amplitude changes over time – but we can't see frequency content directly.
To analyze both time and frequency, we use a Spectrogram – which is a 3D visualization:

- X-axis: time
- Y-axis: frequency
- Color (intensity): signal strength (amplitude) at that frequency and time

In simple terms – the spectrogram tells us "**which frequencies were present at what times.**"

Why Is This Useful?

Real-World Signals Aren't Stationary!!
In real-world signals (like speech, radar, or music), frequency keeps changing with time.

- STFT/Spectrogram helps we see how frequency changes with time.
- Then we analyze or modify that time–frequency map.
- Then we reconstruct or use the results for our application (speech, radar, fault detection, etc.).

Spectrogram of Addition of Sinusoids.

- When we add several sinusoids of different frequencies,

$$x(t)=A1\sin(2\pi f1t)+A2\sin(2\pi f2t)+A3\sin(2\pi f3t)$$

the resulting signal contains multiple frequencies — one for each sine wave.

- In the time domain, this combined signal looks complex because the sinusoids interfere with each other.
- In the frequency domain, each sinusoid appears as a distinct peak at its frequency.
- To visualize both time and frequency information together, we use a spectrogram.
- **If multiple sinusoids exist simultaneously, the spectrogram shows horizontal lines for each frequency, all appearing together throughout the duration of the signal.**

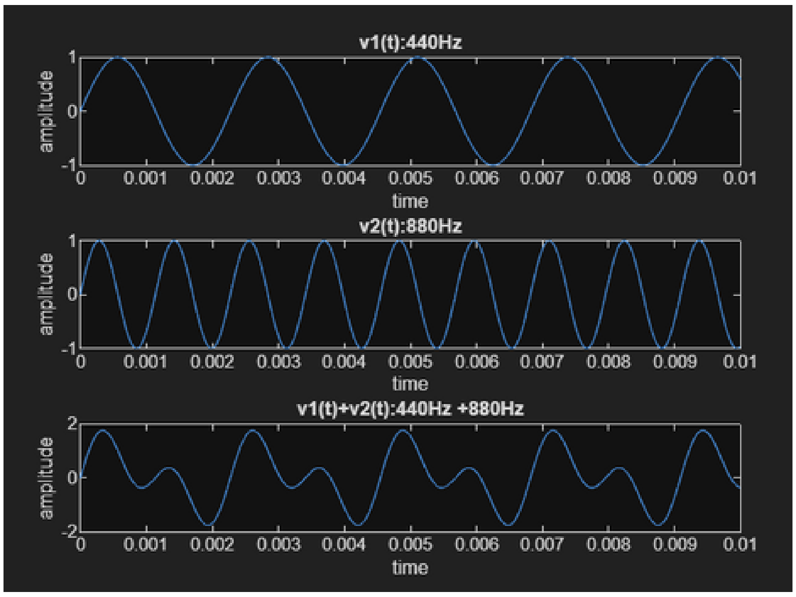
Observations

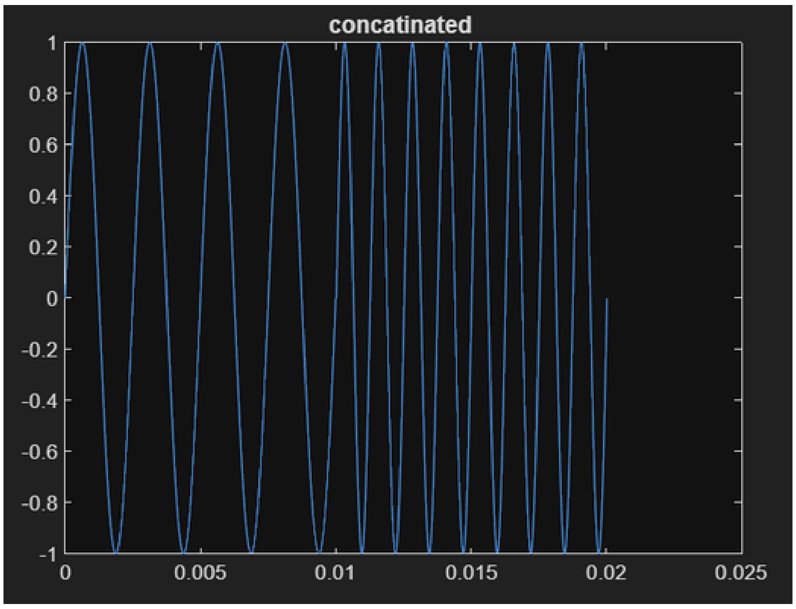
- In the time-domain plot, the combined waveform looks irregular and complex because multiple frequencies are mixed together.
- In the spectrogram, several horizontal bright lines appear — one for each sinusoid frequency (e.g., 100 Hz, 200 Hz, 300 Hz, 400 Hz, 500 Hz).
- These lines remain constant over time, indicating that all frequencies exist simultaneously throughout the signal duration.

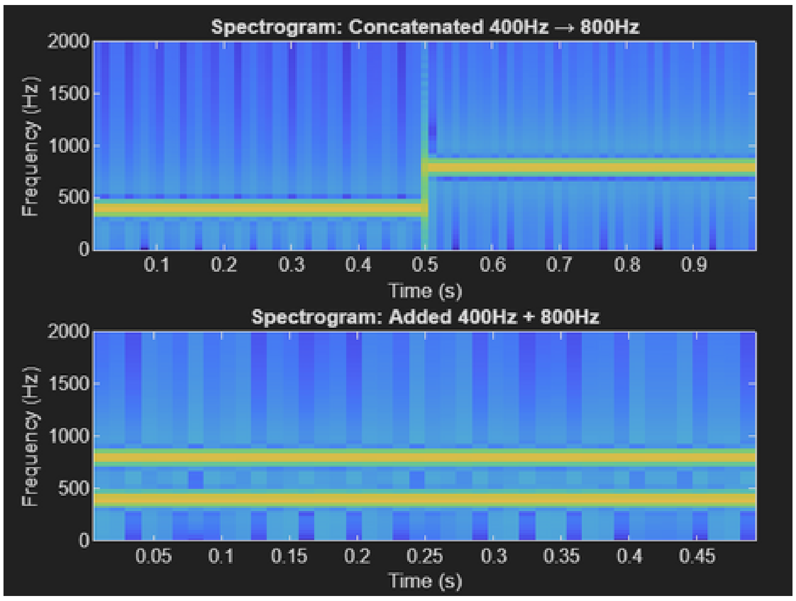
- *The spectrogram confirms that the signal contains multiple fixed frequencies.*
- *Unlike concatenation (Task 1), where frequencies appeared one after another, here they coexist.*
- *Each horizontal band corresponds to one frequency component that remains constant with time.*

Result

The spectrogram of the addition of three/five sinusoids shows multiple parallel horizontal bands, each representing one of the sinusoidal frequencies present in the combined signal. This verifies that all the frequencies exist simultaneously in the signal.


v1(t):440Hz
v2(t):880Hz
v1(t)+v2(t):440Hz +880Hz
amplitude
time


concatinated


Spectrogram: Concatenated 400Hz → 800Hz
Spectrogram: Added 400Hz + 800Hz
Frequency (Hz)
Time (s)

Difference between sound and soundsc commands

Objective

To understand and compare the two MATLAB audio playback functions — sound() and soundsc() — and observe how each handles signal amplitude and playback distortion.

sound() Command

The sound() function is used to play an audio signal stored in a MATLAB variable.

Syntax:

sound(y, Fs)

- y → the audio signal (vector or matrix)
- Fs → sampling frequency (in Hz)

- The function assumes that the values of y lie between −1 and +1.
- If any part of the signal exceeds this range, MATLAB clips those values to fit within it.
- This leads to distortion, making the sound harsh or unpleasant.
- Use sound() when our signal is already normalized (its amplitude lies within [-1, +1]).

soundsc() Command

The soundsc() function plays the same audio signal but with automatic amplitude scaling.

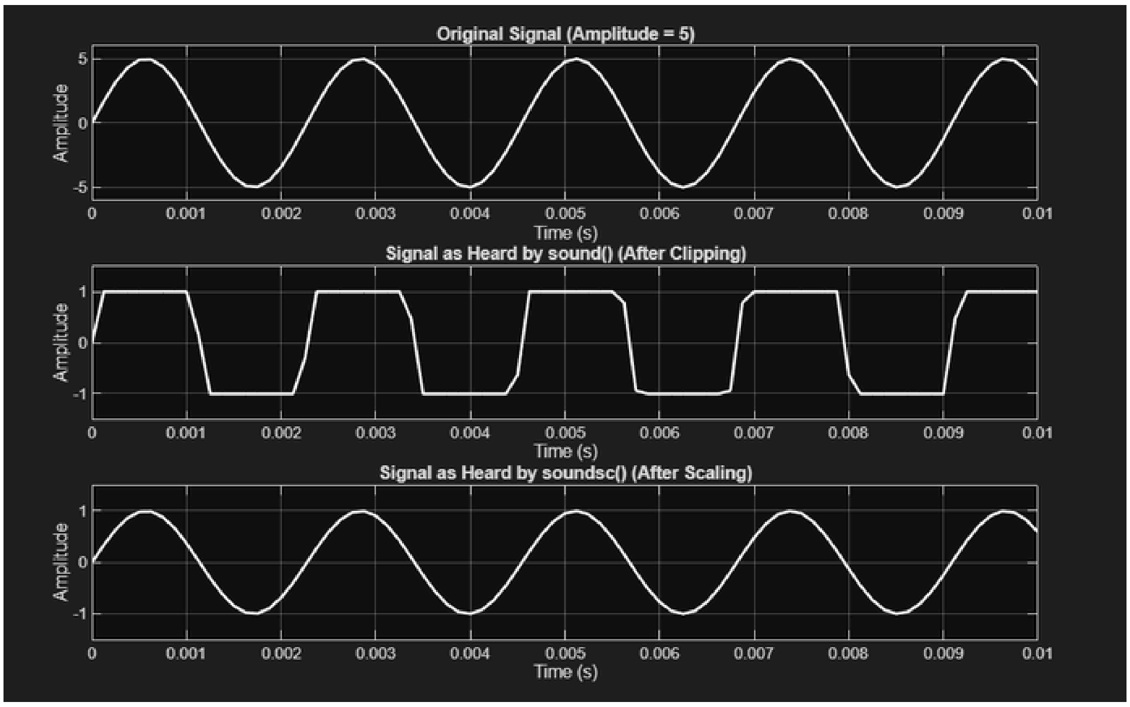
Syntax:

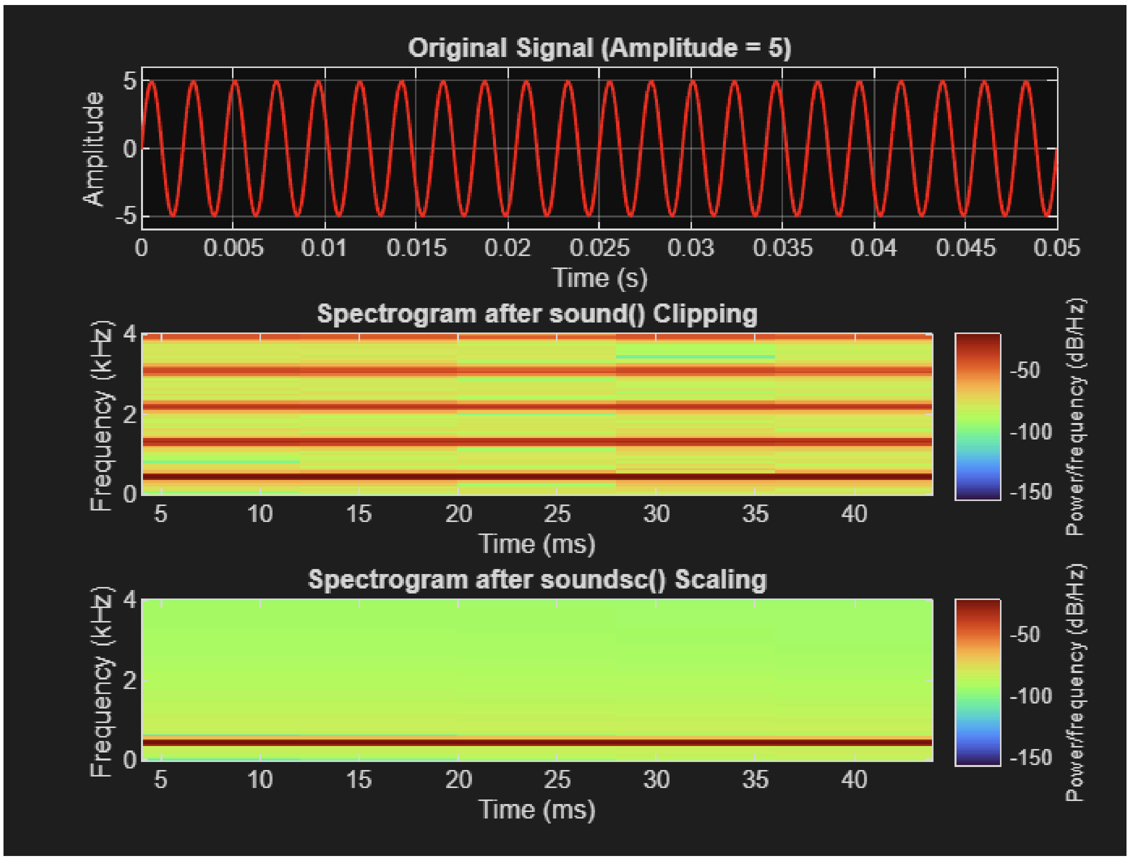
soundsc(y, Fs)

- Here, soundsc rescales the input signal so that its minimum and maximum fit perfectly into the −1 to +1 range before playback.
- This prevents clipping and allows the full dynamic range of the sound to be heard.
- Use soundsc() when our signal's amplitude is unknown or greater than 1.

Observations

- When played using sound(y, Fs), the output sounds distorted because the signal amplitude exceeds ±1, causing MATLAB to clip the waveform.
- When played using soundsc(y, Fs), the sound is clear and undistorted since MATLAB automatically scales the amplitude into the valid range
- Therefore, soundsc() is more reliable for listening to experimental signals with unknown or large amplitudes.


Original Signal (Amplitude = 5)
Amplitude
Time (s)
Signal as Heard by sound() (After Clipping)
Amplitude
Time (s)
Signal as Heard by soundsc() (After Scaling)
Amplitude
Time (s)


Original Signal (Amplitude = 5)
Amplitude
Time (s)
Spectrogram after sound() Clipping
Frequency (kHz)
Time (ms)
Power/frequency (dB/Hz)
Spectrogram after soundsc() Scaling
Frequency (kHz)
Time (ms)
Power/frequency (dB/Hz)

Study of audioread() and audiowrite() Functions

- An audio signal is a continuous-time sound wave (analog) that has been sampled at a specific rate to produce a digital signal — a sequence of discrete amplitude values.
- In MATLAB, these values are stored as numerical arrays that represent sound pressure variations over time.

audioread() Function

- audioread() is used to read (import) audio files such as .wav, .mp3 into MATLAB.

Syntax:

[y, Fs] = audioread('xyz.wav');

- y → Vector or matrix containing the sampled audio data.
  - If mono → single column.
  - If stereo → two columns (left and right channels).
- Fs → Sampling frequency (in Hz). Represents how many samples are taken per second.

The audiowrite() Function

- audiowrite() is used to save (export) an audio signal from MATLAB into a file format like .wav or .mp3.

Syntax:

audiowrite('xyzabcd.wav', y, Fs);

- 'xyzabcd.wav' → desired name of the output file.
- y → signal data (same format as obtained from audioread).
- Fs → sampling frequency.

Image Processing: Blur, Sharpen, and Edge Detection

Objective

To apply basic spatial filtering techniques — blurring, sharpening, and edge detection — on a digital image using MATLAB. This experiment helps understand how convolution with different filter kernels affects an image.

Theory

Digital Image

An image can be represented as a 2D matrix of pixel intensity values. For grayscale images, each element represents brightness from 0 (black) to 1 (white).

Image processing involves modifying these intensity values to enhance or extract useful information.

Spatial Filtering

Spatial filtering involves performing operations on an image using a kernel (mask) that moves over each pixel and modifies it based on its neighbors.

This process is mathematically a 2D convolution between the image and the filter kernel.

(a) Image Blurring

- Blurring (or smoothing) reduces noise and detail in an image.
- A simple way to blur an image is to use an averaging filter, which replaces each pixel with the average of its neighbors.

Kernel used:

$$h_{blur} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- This kernel takes the average of a 3×3 neighborhood.
  Effect:

The image becomes smooth, and fine details or edges get softened.

(b) Image Sharpening

- Sharpening enhances the edges and fine details by emphasizing high-frequency components of the image.

Kernel used:

$$h_{sharp} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

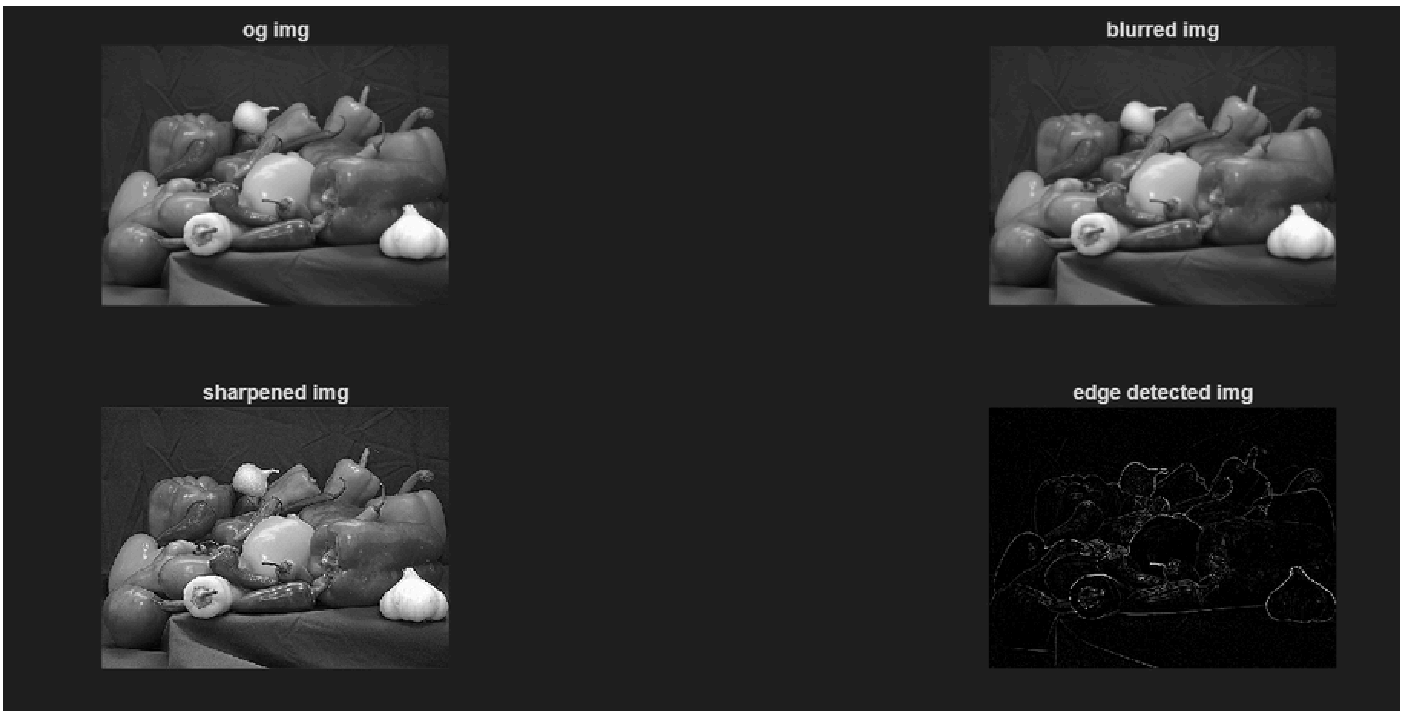
(c) Edge Detection

- Edge detection identifies sharp intensity changes in the image — regions where pixel values vary rapidly.
- It is used to extract object boundaries.

Kernel used (Laplacian operator):

$$h_{edge} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Produces an output highlighting only the edges of objects in the image.



FFT of Image and Frequency Domain Filtering

To compute the 2D Fast Fourier Transform (FFT) of an image and analyze its behavior by applying Low-Pass and High-Pass filters in the frequency domain using MATLAB.

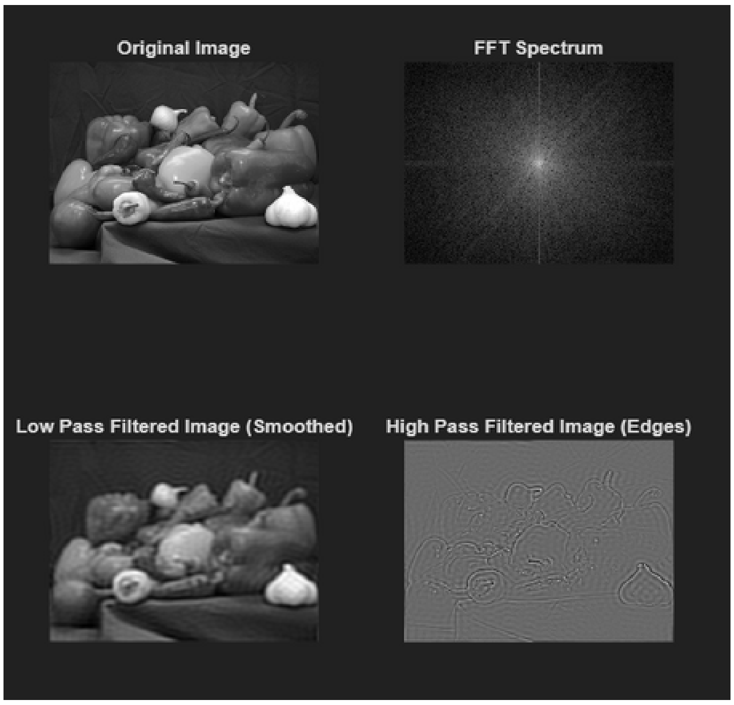
- The 2D Fast Fourier Transform (FFT) is used to convert an image from the spatial domain into the frequency domain.
- Low frequencies correspond to slow intensity changes — smooth regions or backgrounds.
- High frequencies correspond to rapid intensity changes — edges, fine details, or noise.
- The FFT of an image typically shows:

  A bright center → representing low-frequency content.

  Darker corners → representing high-frequency content.
- Filtering in the frequency domain means multiplying the FFT of the image by a filter mask:
- Low-Pass Filter (LPF): Keeps low frequencies → smooths the image (blurring effect).
- High-Pass Filter (HPF): Keeps high frequencies → enhances edges and fine details.

After applying FFT and filtering:

- The FFT Spectrum clearly shows that low frequencies are concentrated in the center.
- The Low-Pass Filtered Image appears smooth and slightly blurred, as high-frequency edges are removed.
- The High-Pass Filtered Image highlights edges and fine structures, as low-frequency background content is removed.
- Thus, frequency-domain filtering effectively controls the level of detail or sharpness in an image.

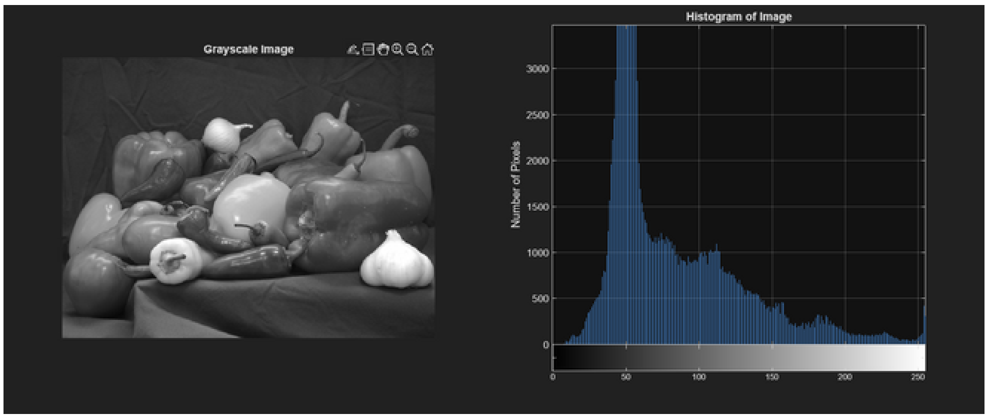

Original Image
FFT Spectrum
Low Pass Filtered Image (Smoothed)
High Pass Filtered Image (Edges)

Histogram

It plots intensity levels (gray levels) on the x-axis and the number of pixels corresponding to each intensity on the y-axis.

The histogram shows how brightness is distributed in the image.
For example:

- If peaks are toward the left → image is dark.
- If peaks are toward the right → image is bright.
- If spread evenly → good contrast.



From the plotted histogram, we can visually interpret the brightness and contrast characteristics of the image.