

Submitted By:  
22-te-70  
22-te-98

## Lab Report:03

### Observing Convolution Technique in Signal Processing

#### Objective:

The objective of this lab is to study impulse response, observe the convolution technique in signal processing, and verify properties such as causality, commutative, distributive, and associative properties.

#### Introduction:

Convolution is a fundamental operation in signal processing used to determine the output of a system when its impulse response and input signal are known. The convolution of two discrete-time signals  $x(n)$  and  $h(n)$  is given by:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(n-k)$$

#### Key Concepts:

1. **Impulse Response:** The impulse response  $h(n)$  of a system is the output of the system when the input is an impulse function  $\delta(n)$ .
2. **Length of Convolution:** The length of the resulting convolution sequence  $y(n)$  is  $N+M-1$ , where  $N$  and  $M$  are the lengths of  $x(n)$  and  $h(n)$ , respectively.
3. **Causality:** A system is causal if its output depends only on the present and past inputs, not on future inputs. For a causal system, the impulse response  $h(n) = 0$  for  $n < 0$ .

#### 4. Properties of Convolution:

- **Commutative:**  $x(n) * h(n) = h(n) * x(n)$
- **Associative:**  
 $x(n) * [h_1(n) * h_2(n)] = [x(n) * h_1(n)] * h_2(n)$   
 $x(n) * [h_1(n) * h_2(n)] = [x(n) * h_1(n)] * h_2(n)$
- **Distributive:**  
 $x(n) * [h_1(n) + h_2(n)] = x(n) * h_1(n) + x(n) * h_2(n)$   
 $x(n) * [h_1(n) + h_2(n)] = x(n) * h_1(n) + x(n) * h_2(n)$

#### > Procedure:

##### 1. Impulse Response:

- Define an impulse signal  $\delta(n)$  and a system's impulse response  $h(n)$ .
- Compute the output  $y(n)$  using convolution.

## 2. Convolution Technique:

- Take two finite-length signals  $x(n)$  and  $h(n)$ .
- Compute the convolution  $y(n) = x(n) * h(n)$  manually and verify using software (e.g., MATLAB, Python).

## 3. Verification of Properties:

- Commutative Property: Verify  $x(n) * h(n) = h(n) * x(n)$ .
- Associative Property: Verify  $x(n) * [h_1(n) * h_2(n)] = [x(n) * h_1(n)] * h_2(n)$ .
- Distributive Property: Verify  $x(n) * [h_1(n) + h_2(n)] = x(n) * h_1(n) + x(n) * h_2(n)$ .

## 4. Causality Check:

- Check if the impulse response  $h(n)$  satisfies  $h(n) = 0$  for  $n < 0$ .

## Observations:

### 1. Impulse Response:

- The output  $y(n)$  matched the expected result when convolving  $x(n)$  with  $h(n)$ .

### 2. Convolution Length:

- The length of  $y(n)$  was  $N+M-1$ , as expected.

### 3. Properties of Convolution:

- Commutative Property: Verified successfully.
- Associative Property: Verified successfully.
- Distributive Property: Verified successfully.

### 4. Causality:

- For a causal system,  $h(n) = 0$  for  $n < 0$ .

### Task 1a: -Run the code and comment on the output:

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

h = np.array([3, 2, 1, -2, 1, 0, -4, 0, 3])
org_h = 1
nh = np.arange(len(h)) - org_h + 1

x = np.array([1, -2, 3, -4, 3, 2, 1])
org_x = 1
nx = np.arange(len(x)) - org_x + 1

y = np.convolve(h, x)
ny = np.arange(nh[0] + nx[0], nh[-1] + nx[-1] + 1)

plt.figure(figsize=(10, 8))

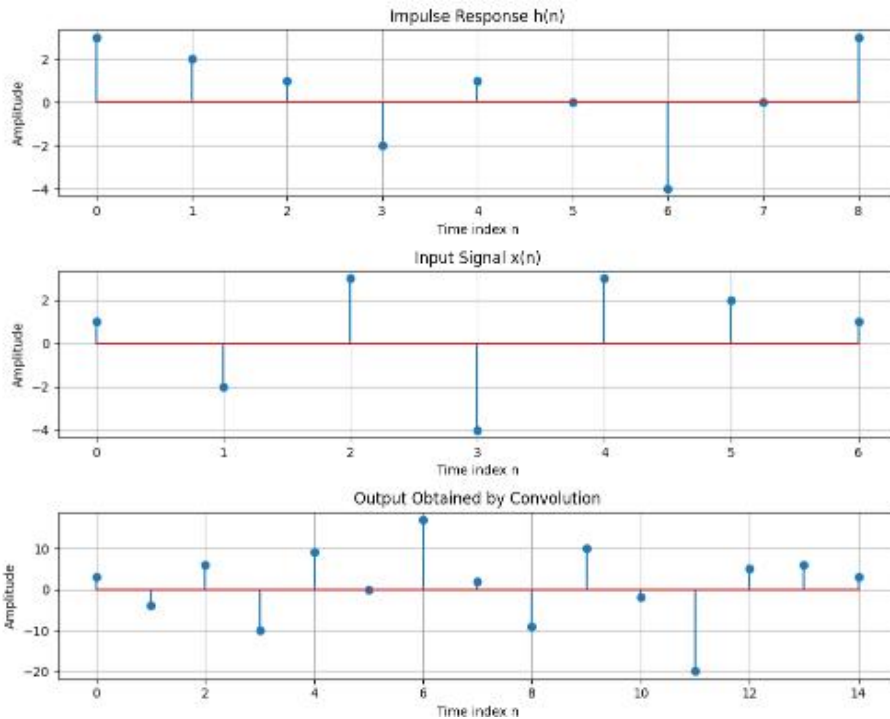
plt.subplot(3, 1, 1)
plt.stem(nh, h)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Impulse Response h(n)')
plt.grid()

plt.subplot(3, 1, 2)
plt.stem(nx, x)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Input Signal x(n)')
plt.grid()

plt.subplot(3, 1, 3)
plt.stem(ny, y)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Output Obtained by Convolution')
plt.grid()

plt.tight_layout()
plt.show()
```

**output:**



**Task 1b:** Calculate the length of input signal (N) and impulse response (M) :

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

h = np.array([3, 2, 1, -2, 1, 0, -4, 0, 3])
org_h = 1
nh = np.arange(len(h)) - org_h + 1

x = np.array([1, -2, 3, -4, 3, 2, 1])
org_x = 1
nx = np.arange(len(x)) - org_x + 1

# Calculate lengths of input signal and impulse response
N = len(x) # Length of input signal
M = len(h) # Length of impulse response
print(f"Length of input signal (N): {N}")
print(f"Length of impulse response (M): {M}")

# Perform convolution
y = np.convolve(h, x)
ny = np.arange(nh[0] + nx[0], nh[-1] + nx[-1] + 1)

plt.figure(figsize=(10, 8))
```

```

plt.subplot(3, 1, 1)
plt.stem(nh, h)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Impulse Response h(n)')
plt.grid()

plt.subplot(3, 1, 2)
plt.stem(nx, x)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Input Signal x(n)')
plt.grid()

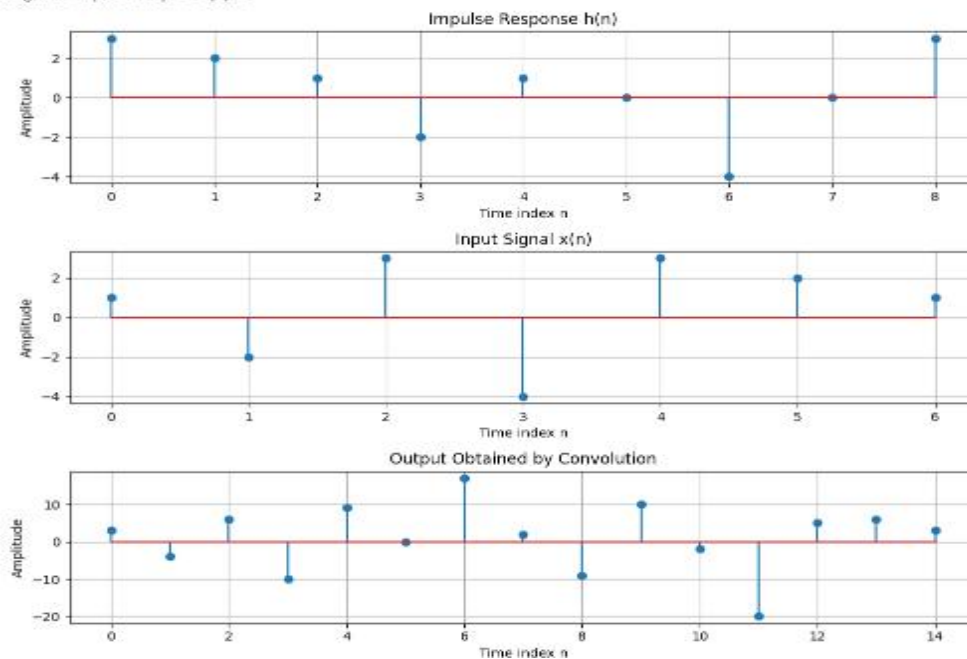
plt.subplot(3, 1, 3)
plt.stem(ny, y)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Output Obtained by Convolution')
plt.grid()

plt.tight_layout()
plt.show()

```

## Output:

\* Length of input signal (N): 7  
 Length of impulse response (M): 9



**Task1 c:** Calculate the length of the output sequence and verify the result with  $N+M-1$ :

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

h = np.array([3, 2, 1, -2, 1, 0, -4, 0, 3])
org_h = 1
nh = np.arange(len(h)) - org_h + 1

x = np.array([1, -2, 3, -4, 3, 2, 1])
org_x = 1
nx = np.arange(len(x)) - org_x + 1

# Calculate lengths of input signal and impulse response
N = len(x) # Length of input signal
M = len(h) # Length of impulse response
print(f"Length of input signal (N): {N}")
print(f"Length of impulse response (M): {M}")

# Perform convolution
y = np.convolve(h, x)
ny = np.arange(nh[0] + nx[0], nh[-1] + nx[-1] + 1)

# Calculate and verify output length
output_length = len(y)
expected_length = N + M - 1
print(f"Length of output sequence: {output_length}")
print(f"Expected length (N+M-1): {expected_length}")
print(f"Verification: {'Correct' if output_length == expected_length else 'Incorrect'}")

plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)
plt.stem(nh, h)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Impulse Response h(n)')
plt.grid()

plt.subplot(3, 1, 2)
plt.stem(nx, x)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Input Signal x(n)')
```

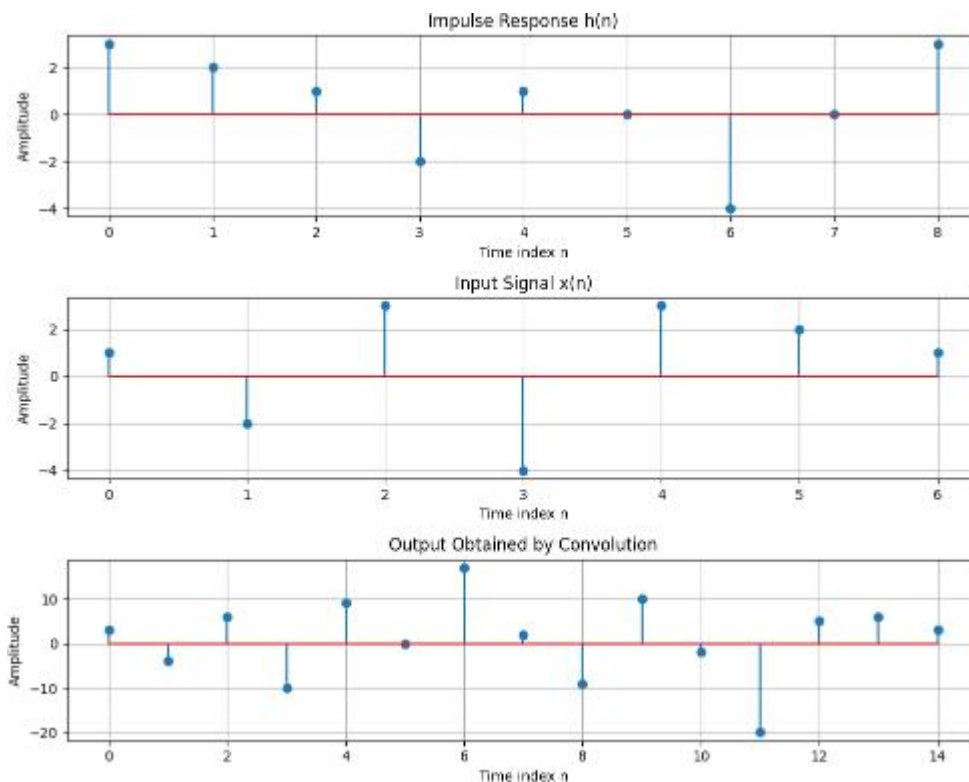
```
plt.grid()

plt.subplot(3, 1, 3)
plt.stem(ny, y)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Output Obtained by Convolution')
plt.grid()

plt.tight_layout()
plt.show()
```

## Output:

```
Length of input signal (N): 7
Length of impulse response (M): 9
Length of output sequence: 15
Expected length (N+M-1): 15
Verification: Correct
```



**Task1 d:** Now modify the above code such that  $h(n) = \{3, 2, 1, -2, 1, 0, -4, 0, 3\}$  (origin is shifted) and check for causality.

```
import numpy as np
```

```

import matplotlib.pyplot as plt

h = np.array([3, 2, 1, -2, 1, 0, -4, 0, 3])
org_h = 4 # Shifted origin
nh = np.arange(len(h)) - org_h

x = np.array([1, -2, 3, -4, 3, 2, 1])
org_x = 1
nx = np.arange(len(x)) - org_x + 1

# Calculate lengths of input signal and impulse response
N = len(x) # Length of input signal
M = len(h) # Length of impulse response
print(f"Length of input signal (N): {N}")
print(f"Length of impulse response (M): {M}")

# Perform convolution
y = np.convolve(h, x)
ny = np.arange(nh[0] + nx[0], nh[-1] + nx[-1] + 1)

# Calculate and verify output length
output_length = len(y)
expected_length = N + M - 1
print(f"Length of output sequence: {output_length}")
print(f"Expected length (N+M-1): {expected_length}")
print(f"Verification: {'Correct' if output_length == expected_length else 'Incorrect'}")

# Check for causality
is_causal = np.all(nh >= 0)
print(f"System is {'Causal' if is_causal else 'Non-Causal'}")

plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)
plt.stem(nh, h)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Impulse Response h(n)')
plt.grid()

plt.subplot(3, 1, 2)
plt.stem(nx, x)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')

```



```
plt.title('Input Signal x(n)')
plt.grid()

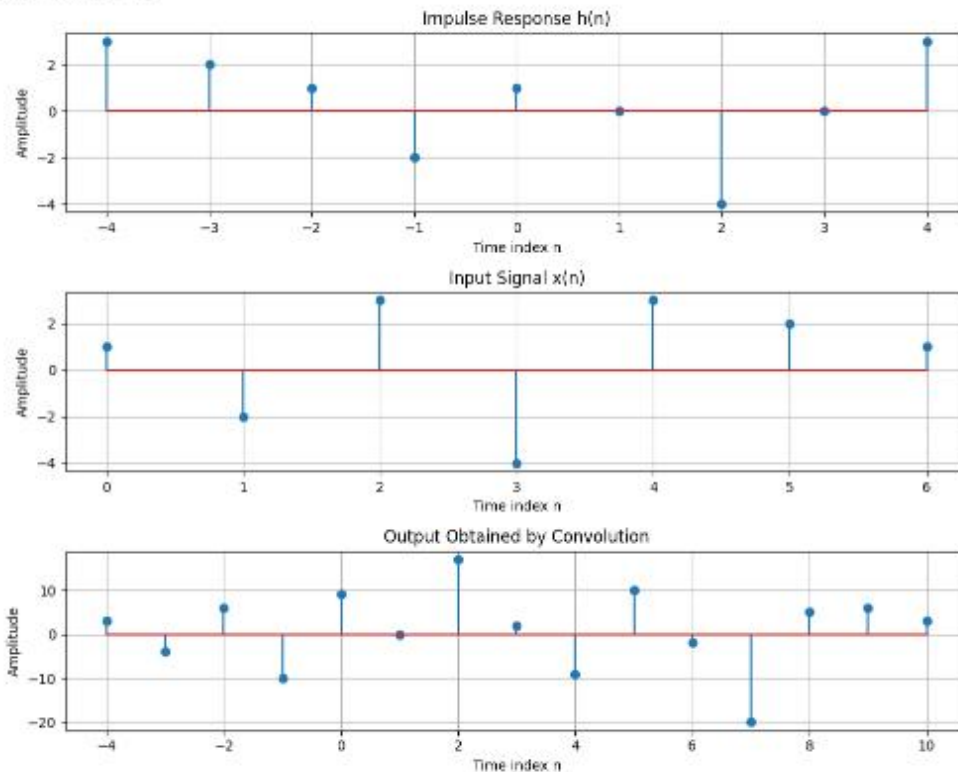
plt.subplot(3, 1, 3)
plt.stem(ny, y)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.title('Output Obtained by Convolution')
plt.grid()

plt.tight_layout()
plt.show()
```

## Output:

```
> Length of input signal (N): 7
  Length of impulse response (M): 9
  Length of output sequence: 15
  Expected length (N+M-1): 15
  Verification: Correct
  System is Non-Causal
```

System is non-causal



**Task2 a:** . What will happen if we input  $x(n)=\{0,0,1,0,0\}$  into the above system:

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

# Define impulse response
h = np.array([3, 2, 1, -2, 1, 0, -4, 0, 3]) # Impulse response
org_h = 1 # Sample number where origin exists
nh = np.arange(len(h)) - org_h + 1 # Time indices for h

# Define input sequence
x = np.array([0, 0, 1, 0, 0]) # Input sequence (Impulse-like input)
org_x = 2 # Sample number where origin exists
nx = np.arange(len(x)) - org_x + 1 # Time indices for x

# Perform convolution
y = np.convolve(h, x) # Convolution of h and x
ny = np.arange(nh[0] + nx[0], nh[-1] + nx[-1] + 1) # Time indices for y

# Plot impulse response
plt.figure(figsize=(10, 8))
plt.subplot(3, 1, 1)
plt.stem(nh, h)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.xlim([nh[0] - 1, nh[-1] + 1])
plt.title('Impulse Response h(n)')
plt.grid()

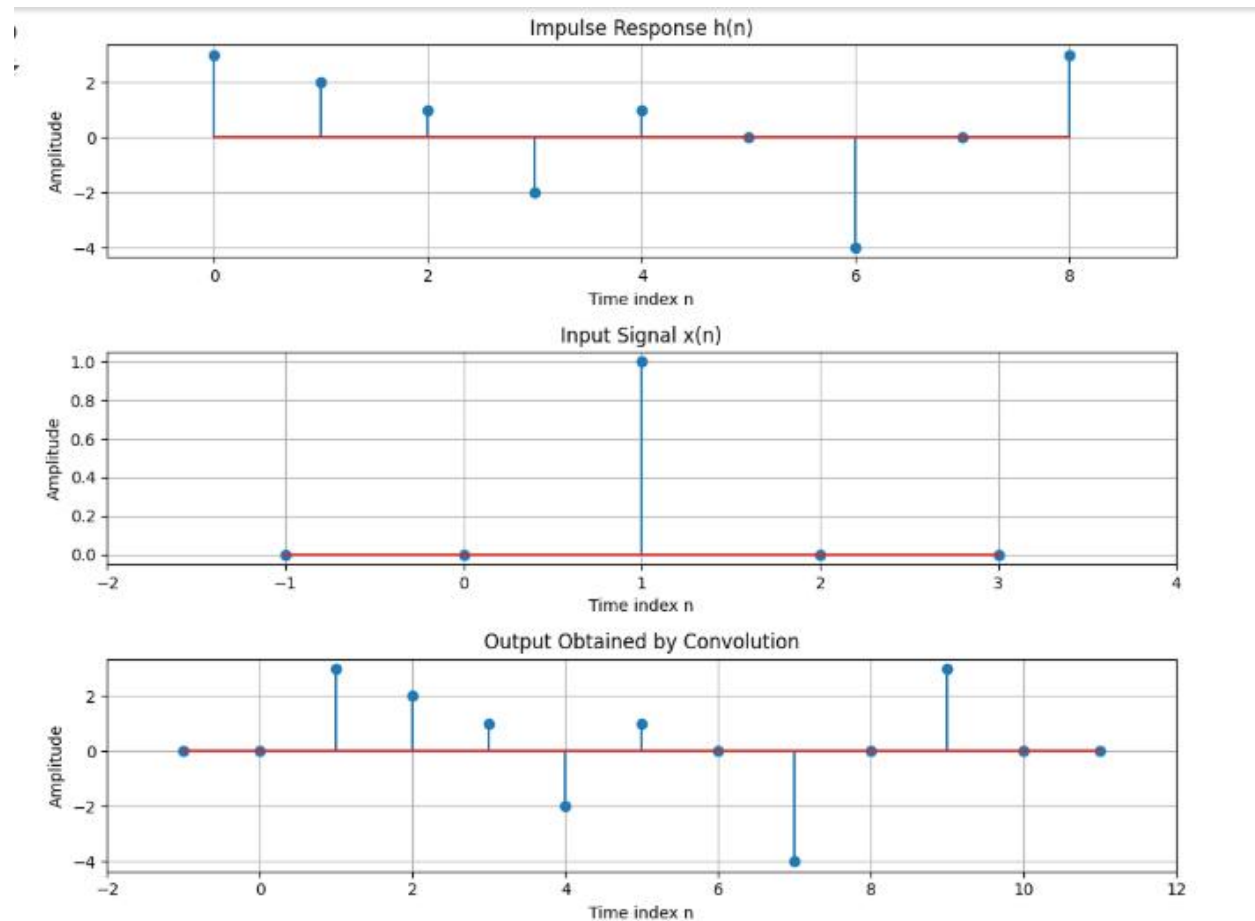
# Plot input signal
plt.subplot(3, 1, 2)
plt.stem(nx, x)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.xlim([nx[0] - 1, nx[-1] + 1])
plt.title('Input Signal x(n)')
plt.grid()

# Plot output signal
plt.subplot(3, 1, 3)
plt.stem(ny, y)
plt.xlabel('Time index n')
plt.ylabel('Amplitude')
plt.xlim([ny[0] - 1, ny[-1] + 1])
plt.title('Output Obtained by Convolution')
```

```
plt.grid()

# Show the plots
plt.tight_layout()
plt.show()
```

## Output:



## Task2 b: prove the commutative property of the convolution:

```
import numpy as np

h = np.array([3, 2, 1, -2, 1, 0, -4, 0, 3])
x = np.array([0, 0, 1, 0, 0])

def convolution(h, x):
    len_h = len(h)
    len_x = len(x)
    len_y = len h + len x - 1
```

```

y = np.zeros(len_y)

for n in range(len_y):
    for k in range(len_h):
        if 0 <= n - k < len_x:
            y[n] += h[k] * x[n - k]

return y

y_hx = convolution(h, x)
y_xh = convolution(x, h)

print("h(n) * x(n) = ", y_hx)
print("x(n) * h(n) = ", y_xh)

if np.array_equal(y_hx, y_xh):
    print("The commutative property holds: h(n) * x(n) = x(n) * h(n)")
else:
    print("The commutative property does not hold.")

```

## Output:

```

➡ h(n) * x(n) = [ 0.  0.  3.  2.  1. -2.  1.  0. -4.  0.  3.  0.  0.]
  x(n) * h(n) = [ 0.  0.  3.  2.  1. -2.  1.  0. -4.  0.  3.  0.  0.]
  The commutative property holds: h(n) * x(n) = x(n) * h(n)

```

## Task2 c: prove Associative and Distributed properties of the convolution:

```

h1 = np.array([1, -1, 2])
h2 = np.array([0, 1, 3])

lhs = np.convolve(np.convolve(x, h1), h2)
rhs = np.convolve(x, np.convolve(h1, h2))

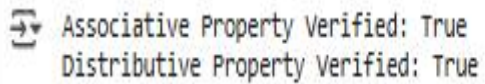
print("Associative Property Verified:", np.array_equal(lhs, rhs))

lhs_dis = np.convolve(x, h1 + h2)
rhs_dis = np.convolve(x, h1) + np.convolve(x, h2)

print("Distributive Property Verified:", np.array_equal(lhs_dis, rhs_dis))

```

## Output:



Associative Property Verified: True  
Distributive Property Verified: True

**Task2 d1:** Convolve your recorded sound with drumloop.wav. Note your observation

a) Plot the output.

```
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
import IPython.display as ipd

recorded_sound, sr_recorded = librosa.load('/content/sunflower-street-
drumloop-85bpm-163900 (1).mp3', sr=None)
drumloop, sr_drumloop = librosa.load('/content/sunflower-street-drumloop-
85bpm-163900 (1).mp3', sr=None)

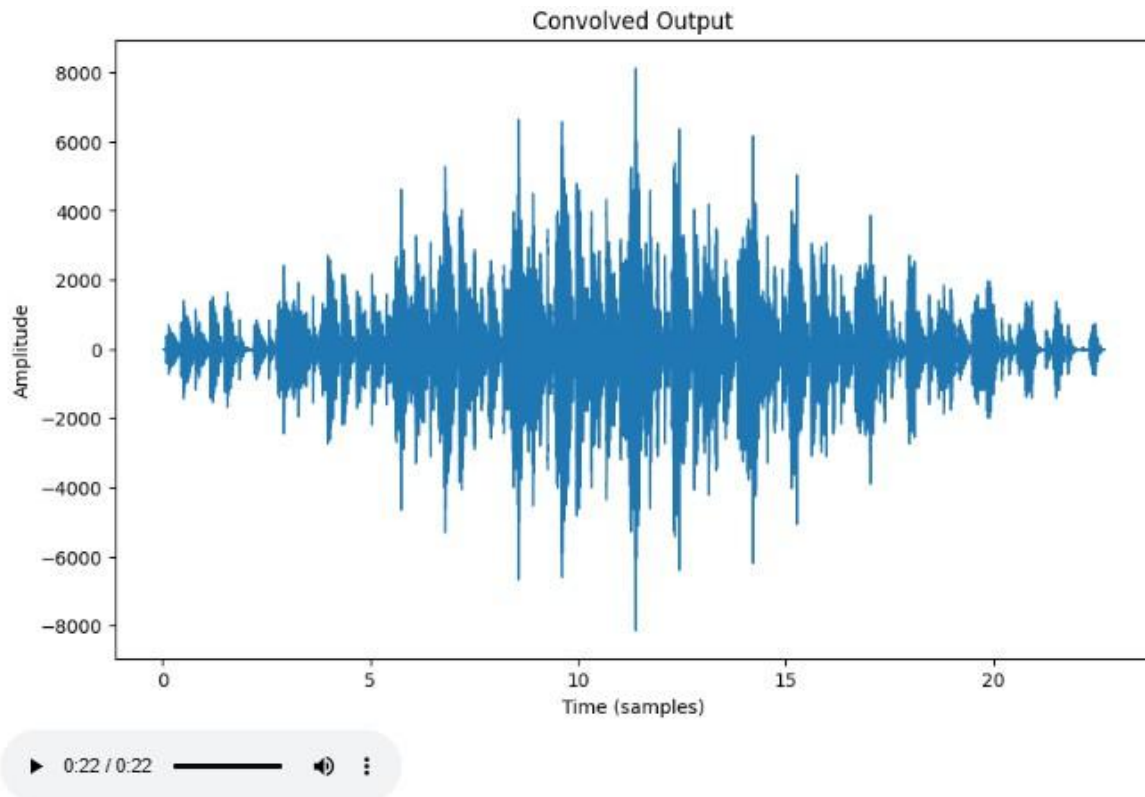
# Check if both signals have the same sample rate
if sr_recorded != sr_drumloop:
    raise ValueError("Sample rates do not match. Please use audio files
with the same sample rate.")

# Perform convolution
convolved_output = np.convolve(recorded_sound, drumloop, mode='full')

# a) Plot the output of the convolution
plt.figure(figsize=(10, 6))
librosa.display.waveshow(convolved_output, sr=sr_recorded)
plt.title('Convolved Output')
plt.xlabel('Time (samples)')
plt.ylabel('Amplitude')
plt.show()

# b) Play the convolved output
ipd.display(ipd.Audio(convolved_output, rate=sr_recorded))
```

## Output:



## Task2 d2: Listen the output:

```
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display

# Load the uploaded files (replace with your actual file names after
upload)
recorded_sound_path = '/content/sunflower-street-drumloop-85bpm-163900
(1).mp3' # Replace with your file path
drum_loop_path = '/content/bossa-nova-eletric-guitar-loop-258055.mp3' #
Replace with your file path

# Load the recorded sound and drum loop
recorded_sound, sr_recorded = librosa.load(recorded_sound_path, sr=None,
mono=True)
drum_loop, sr_drum = librosa.load(drum_loop_path, sr=None, mono=True)

# Ensure both signals have the same length by padding the shorter one
if len(recorded_sound) > len(drum_loop):
    drum_loop = np.pad(drum_loop, (0, len(recorded_sound) -
len(drum_loop)))
```

```

else:
    recorded_sound = np.pad(recorded_sound, (0, len(drum_loop) -
len(recorded_sound)))

# Perform convolution
convolved_signal = np.convolve(recorded_sound, drum_loop, mode='same')

# Plot the original and convolved signals
plt.figure(figsize=(12, 6))

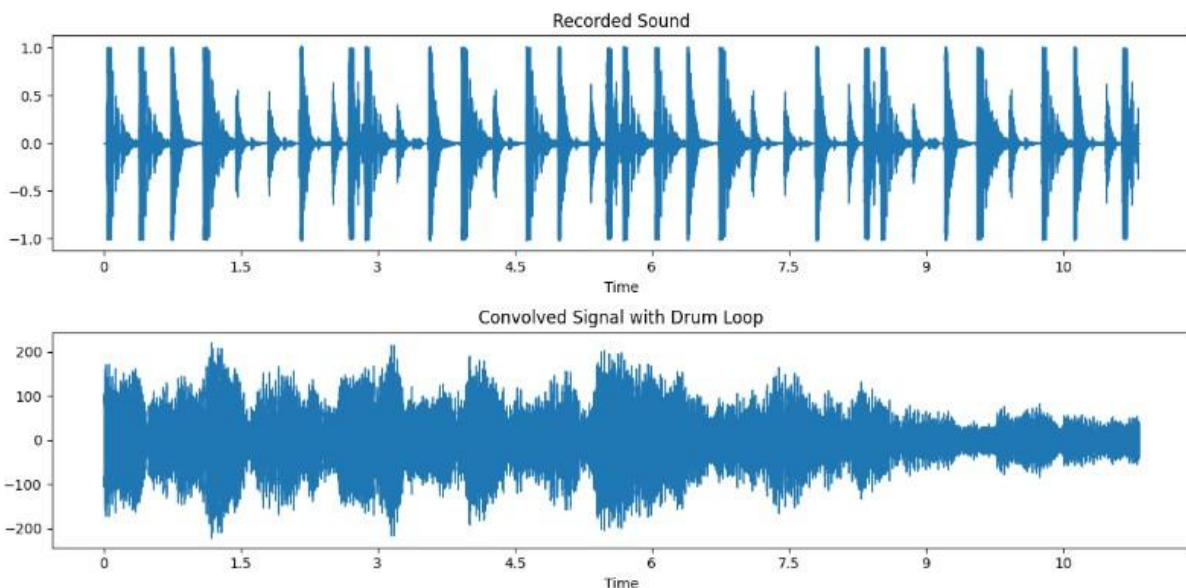
plt.subplot(2, 1, 1)
plt.title('Recorded Sound')
librosa.display.waveshow(recorded_sound, sr=sr_recorded)

plt.subplot(2, 1, 2)
plt.title('Convolved Signal with Drum Loop')
librosa.display.waveshow(convolved_signal, sr=sr_recorded)

plt.tight_layout()
plt.show()

```

## Output:



**Task 3a:** Create a sine wave signal as the original signal.

- Use a sampling frequency Hz.
- Signal duration: 2 seconds.
- Signal frequency: 440 Hz (A4 note).

```

import numpy as np
import matplotlib.pyplot as plt

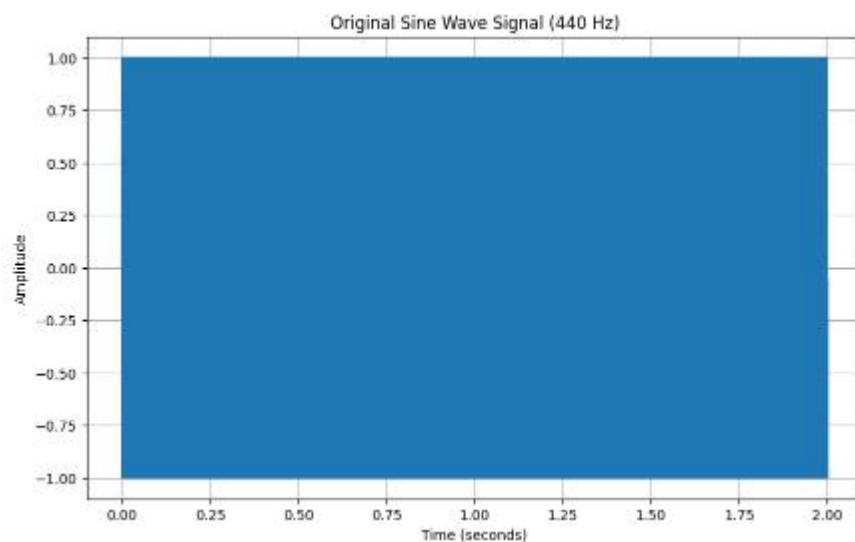
sampling_frequency = 44100
signal_duration = 2
signal_frequency = 440

t = np.linspace(0, signal_duration, int(sampling_frequency *
signal_duration), endpoint=False)
signal = np.sin(2 * np.pi * signal_frequency * t)

plt.figure(figsize=(10, 6))
plt.plot(t, signal)
plt.title('Original Sine Wave Signal (440 Hz)')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()

```

## Output:



### Task3b: 1. Define parameters for the echo:

- o Delay: 0.2 seconds
- o Attenuation factor

2. Add the echo to the original signal by creating a delayed and scaled copy of it.

```
import numpy as np
```



```

import matplotlib.pyplot as plt

sampling_frequency = 44100
signal_duration = 2
signal_frequency = 440

t = np.linspace(0, signal_duration, int(sampling_frequency *
signal_duration), endpoint=False)
signal = np.sin(2 * np.pi * signal_frequency * t)

delay = 0.2
attenuation_factor = 0.5

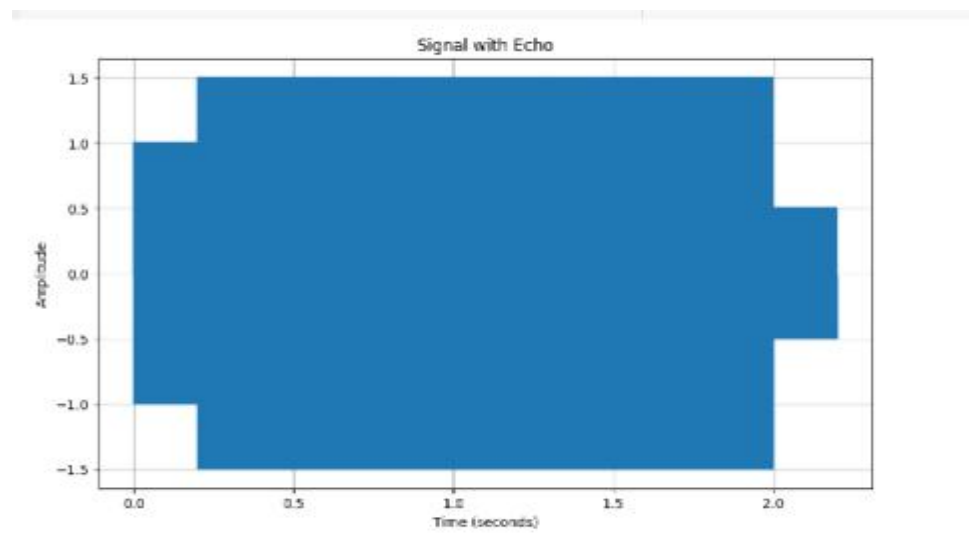
delay_samples = int(delay * sampling_frequency)

echo_signal = np.zeros(len(signal) + delay_samples)
echo_signal[:len(signal)] = signal
echo_signal[delay_samples:] += attenuation_factor * signal

plt.figure(figsize=(10, 6))
plt.plot(np.linspace(0, signal_duration + delay, len(echo_signal)),
echo_signal)
plt.title('Signal with Echo')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()

```

## Output:



**Task3c:** . Create an impulse response with:

- o 1 at index 0
- o at the delayed index

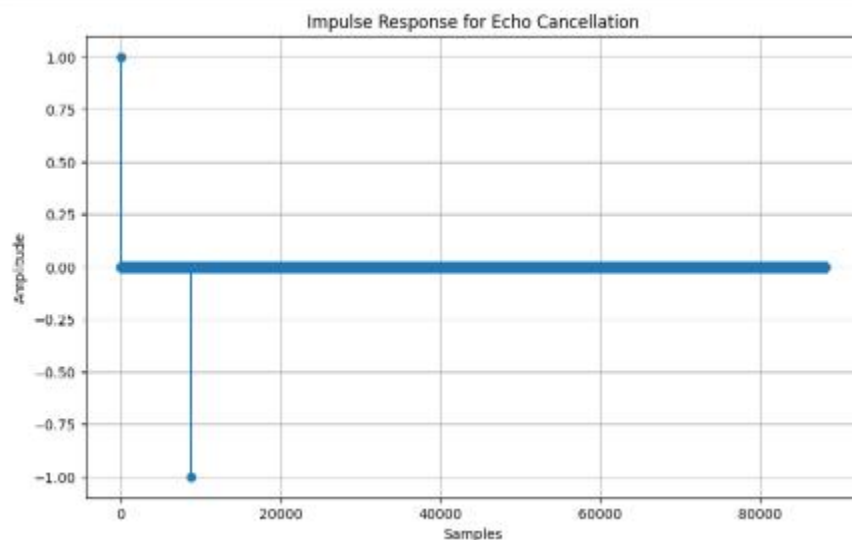
```
import numpy as np
import matplotlib.pyplot as plt

sampling_frequency = 44100
delay = 0.2
delay_samples = int(delay * sampling_frequency)

impulse_response = np.zeros(int(sampling_frequency * 2)) # Allocate
enough space
impulse_response[0] = 1 # Set 1 at index 0
impulse_response[delay_samples] = -1 # Set -1 at the delayed index for
echo cancellation

plt.figure(figsize=(10, 6))
plt.stem(np.arange(len(impulse_response)), impulse_response, basefmt=" ")
plt.title('Impulse Response for Echo Cancellation')
plt.xlabel('Samples')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
```

**Output:**



**Task3d:** Convolve the echoed signal with the impulse response to recover the original signal.

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

sampling_frequency = 44100
signal_duration = 2
signal_frequency = 440

t = np.linspace(0, signal_duration, int(sampling_frequency *
signal_duration), endpoint=False)
signal = np.sin(2 * np.pi * signal_frequency * t)

delay = 0.2
attenuation_factor = 0.5
delay_samples = int(delay * sampling_frequency)

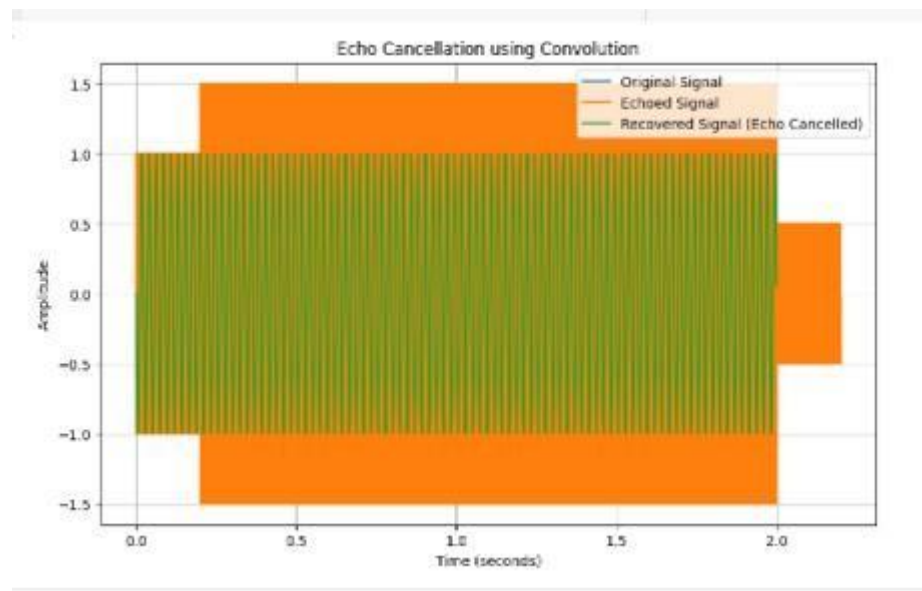
echo_signal = np.zeros(len(signal) + delay_samples)
echo_signal[:len(signal)] = signal
echo_signal[delay_samples:] += attenuation_factor * signal

impulse_response = np.zeros(int(sampling_frequency * 2))
impulse_response[0] = 1
impulse_response[delay_samples] = -1

recovered_signal = np.convolve(echo_signal, impulse_response,
mode='valid')

plt.figure(figsize=(10, 6))
plt.plot(t, signal, label='Original Signal') plt.plot(np.linspace(0,
signal_duration + delay, len(echo_signal)), echo_signal,
label='Echoed Signal')
plt.plot(np.linspace(0, signal_duration, len(recovered_signal)),
recovered_signal, label='Recovered Signal (Echo Cancelled)')
plt.title('Echo Cancellation using Convolution')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid(True)
plt.show()
```

## Output:



## Task3e: Plot the following:

- o Original signal
- o Signal with echo
- o Recovered signal

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

sampling_frequency = 44100
signal_duration = 2
signal_frequency = 440

t = np.linspace(0, signal_duration, int(sampling_frequency *
signal_duration), endpoint=False)
signal = np.sin(2 * np.pi * signal_frequency * t)

delay = 0.2
attenuation_factor = 0.5
delay_samples = int(delay * sampling_frequency)

echo_signal = np.zeros(len(signal) + delay_samples)
echo_signal[:len(signal)] = signal
echo_signal[delay_samples:] += attenuation_factor * signal
```

```

impulse_response = np.zeros(int(sampling_frequency * 2))
impulse_response[0] = 1
impulse_response[delay_samples] = -1

recovered_signal = np.convolve(echo_signal, impulse_response,
mode='valid')

plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.plot(t, signal, label='Original Signal')
plt.title('Original Signal')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)

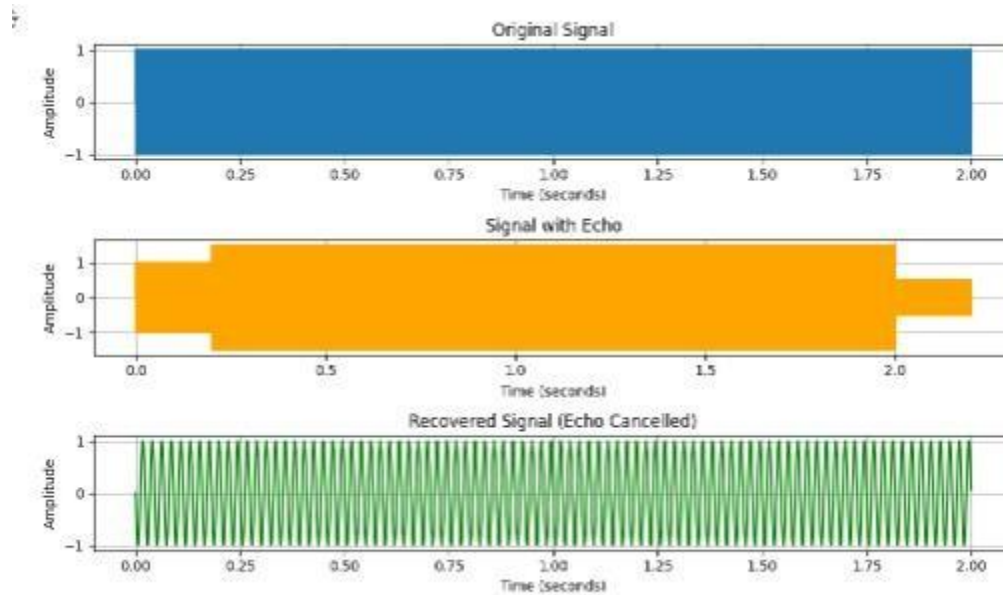
plt.subplot(3, 1, 2)
plt.plot(np.linspace(0, signal_duration + delay, len(echo_signal)),
echo_signal, label='Signal with Echo', color='orange')
plt.title('Signal with Echo')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(np.linspace(0, signal_duration, len(recovered_signal)),
recovered_signal, label='Recovered Signal (Echo Cancelled)',
color='green')
plt.title('Recovered Signal (Echo Cancelled)')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)

plt.tight_layout()
plt.show()

```

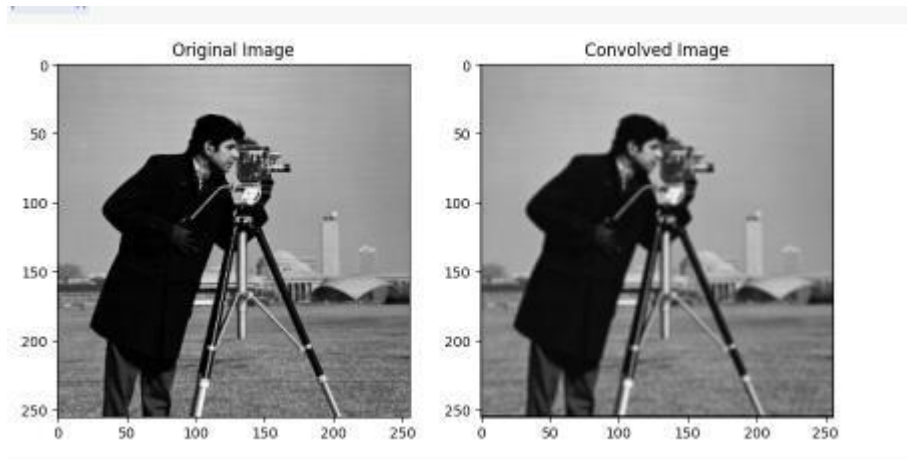
**Output:**



**Task4a:** Reconstruct the original image signal:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the image in grayscale mode
im = cv2.imread('cameraman.tif', cv2.IMREAD_GRAYSCALE)
# Define the kernel (mask)
mask = np.ones((3, 3)) / 9
# Prepare an output array of the same shape as the input
im_conv = np.zeros_like(im)
# Perform convolution
for i in range(1, im.shape[0] - 1):
    for j in range(1, im.shape[1] - 1):
        # Element-wise multiplication and summation
        im_conv[i, j] = np.sum(im[i-1:i+2, j-1:j+2] * mask)
# Plot the original and convolved images
plt.figure(figsize=(10, 15))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(im, cmap='gray')
plt.subplot(1, 2, 2)
plt.title("Convolved Image")
plt.imshow(im_conv, cmap='gray')
plt.show()
```

## Output:



## Task4b:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def bilateral_filter(img, d=5, sigma_color=75, sigma_space=75):
    # Get image dimensions
    height, width = img.shape

    # Create output image
    im_bilateral = np.zeros_like(img)

    # Create Gaussian kernels for spatial distance and intensity
    difference

    half_d = d // 2
    spatial_kernel = np.zeros((d, d))
    intensity_kernel = np.zeros((d, d))

    # Create the spatial Gaussian kernel
    for i in range(d):
        for j in range(d):
            spatial_kernel[i, j] = np.exp(-((i - half_d) ** 2 + (j -
half_d) ** 2) / (2 * sigma_space ** 2))

    # Normalize spatial kernel
    spatial_kernel /= np.sum(spatial_kernel)

    # Perform bilateral filtering
    for i in range(half_d, height - half_d):
        for j in range(half_d, width - half_d):
```

```

        weighted_sum = 0
        weight_sum = 0
        for k in range(-half_d, half_d + 1):
            for l in range(-half_d, half_d + 1):
                # Intensity difference between the central pixel and
the surrounding pixels
                intensity_diff = img[i, j] - img[i + k, j + l]
                intensity_kernel[k + half_d, l + half_d] = np.exp(-(
intensity_diff ** 2) / (2 * sigma_color ** 2))

                # Combined weight (spatial * intensity)
                combined_weight = spatial_kernel[k + half_d, l +
half_d] * intensity_kernel[k + half_d, l + half_d]

                # Update weighted sum
                weighted_sum += img[i + k, j + l] * combined_weight
                weight_sum += combined_weight

        # Normalize the result
        im_bilateral[i, j] = weighted_sum / weight_sum

    return im_bilateral

# Read the image in grayscale mode
im = cv2.imread('cameraman.tif', cv2.IMREAD_GRAYSCALE)

# Apply the bilateral filter
im_bilateral = bilateral_filter(im)

# Plot the original and bilateral filtered images
plt.figure(figsize=(10, 15))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(im, cmap='gray')

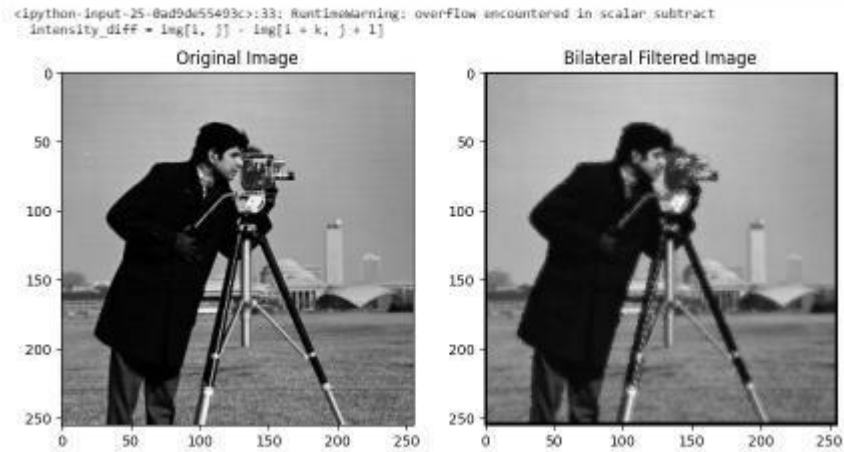
plt.subplot(1, 2, 2)
plt.title("Bilateral Filtered Image")
plt.imshow(im_bilateral, cmap='gray')

plt.show()

```

**Output:**





### Task4c: RECONSTRUCT ORIGINAL IMAGE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image in grayscale mode
im = cv2.imread('cameraman.tif', cv2.IMREAD_GRAYSCALE)

# Define the kernel (mask) used for blurring
mask = np.ones((3, 3)) / 9

# Prepare an output array of the same shape as the input
im_conv = np.zeros_like(im)

# Perform convolution (blurring)
for i in range(1, im.shape[0] - 1):
    for j in range(1, im.shape[1] - 1):
        im_conv[i, j] = np.sum(im[i-1:i+2, j-1:j+2] * mask)

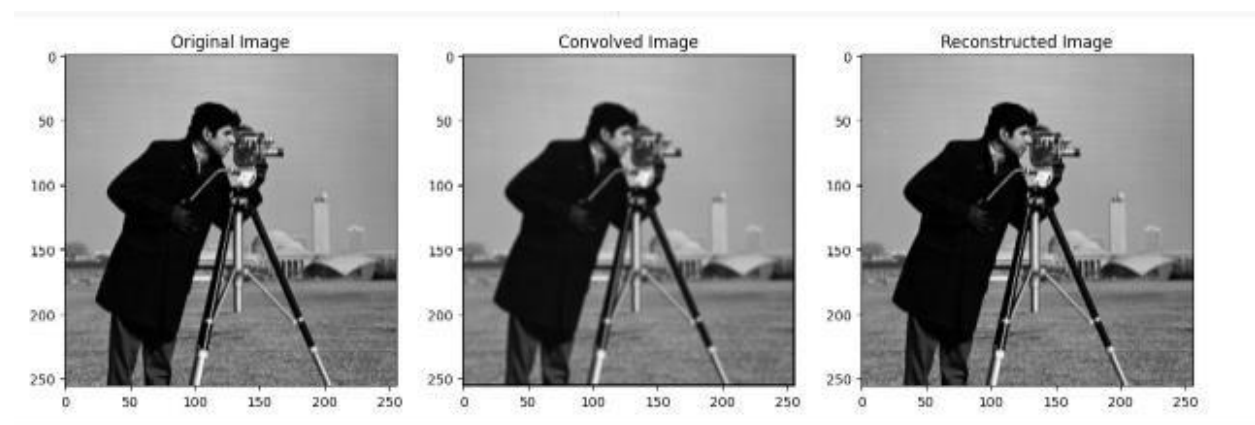
# Attempt to approximate the original image using unsharp masking
# Calculate the difference between the original and convolved image
# (blurring effect)
high_pass = im - im_conv

# Reconstruct the image by adding the high-frequency details back to the
# convolved image
reconstructed_image = im_conv + high_pass

# Plot the original, convolved, and reconstructed images
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
```

```
plt.title("Original Image")
plt.imshow(im, cmap='gray')
plt.subplot(1, 3, 2)
plt.title("Convolved Image")
plt.imshow(im_conv, cmap='gray')
plt.subplot(1, 3, 3)
plt.title("Reconstructed Image")
plt.imshow(reconstructed_image, cmap='gray')
plt.show()
```

## Output:



## Conclusion:

In this lab, the convolution technique in signal processing was studied, and the impulse response of a system was analyzed. The properties of convolution, such as commutative, associative, and distributive, were verified experimentally. Additionally, the concept of causality was tested by ensuring that the impulse response  $h(n)$  of a causal system is zero for  $n < 0$ . The results confirmed the theoretical understanding of convolution and its properties, demonstrating its importance in analyzing and designing linear time-invariant (LTI) systems.