Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Exploring Convolution and Correlation in Discrete-Time Signals	
Experiment No: 18	**Date:20-11-2025**	**Enrollment No:92510133049**

Aim: Exploring Convolution and Correlation in Discrete-Time Signals

IDE:

What is Convolution?

Convolution is a mathematical operation that combines two functions (or signals) to produce a third function, showing how one signal modifies or filters another. It essentially describes how the shape of one signal is altered by another signal. In the context of **signal processing**, convolution is used to analyze the effect of a system (such as a filter) on an input signal.

Mathematically, the **convolution** of two discrete-time signals $x[n]$ and $h[n]$ is given by:

$$y[n] = (x * h)[n] = \sum_{k=-\infty}^{\infty} x[k].\, h[n-k]$$

In simple terms, the convolution between two signals involves "sliding" one signal over another, multiplying overlapping values, and summing the results to produce a new signal.

Types of Convolution:

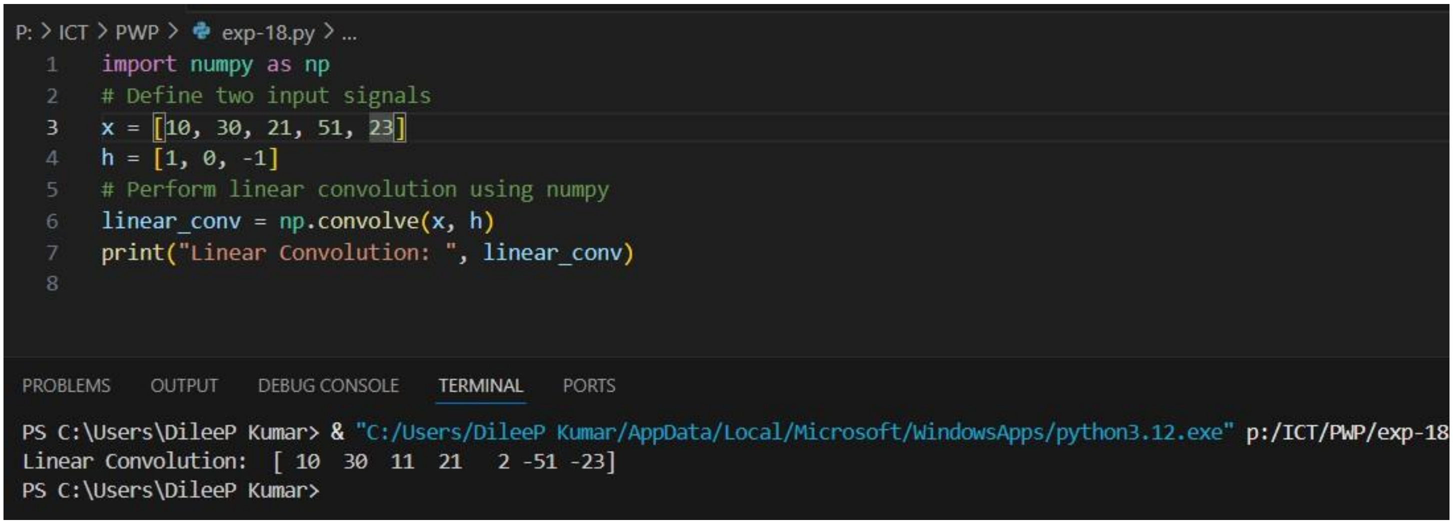
1. **Linear Convolution**: This computes the output for the entire range of the input signals and doesn't assume periodicity.
2. **Circular Convolution**: This assumes the signals are periodic and wraps the values around when they go beyond the signal length.

Python Implementation

Example 1:

```
import numpy as np
# Define two input signals
x = [1, 2, 3, 4]
h = [1, 0, -1]
# Perform linear convolution using numpy
linear_conv = np.convolve(x, h)
print("Linear Convolution: ", linear_conv)
```

output:



```python
import numpy as np
# Define two input signals
x = [10, 30, 21, 51, 23]
h = [1, 0, -1]
# Perform linear convolution using numpy
linear_conv = np.convolve(x, h)
print("Linear Convolution: ", linear_conv)
```

```
PS C:\Users\DileeP Kumar> & "C:/Users/DileeP Kumar/AppData/Local/Microsoft/WindowsApps/python3.12.exe" p:/ICT/PWP/exp-18
Linear Convolution:  [ 10  30  11  21   2 -51 -23]
PS C:\Users\DileeP Kumar>
```

Circular Convolution

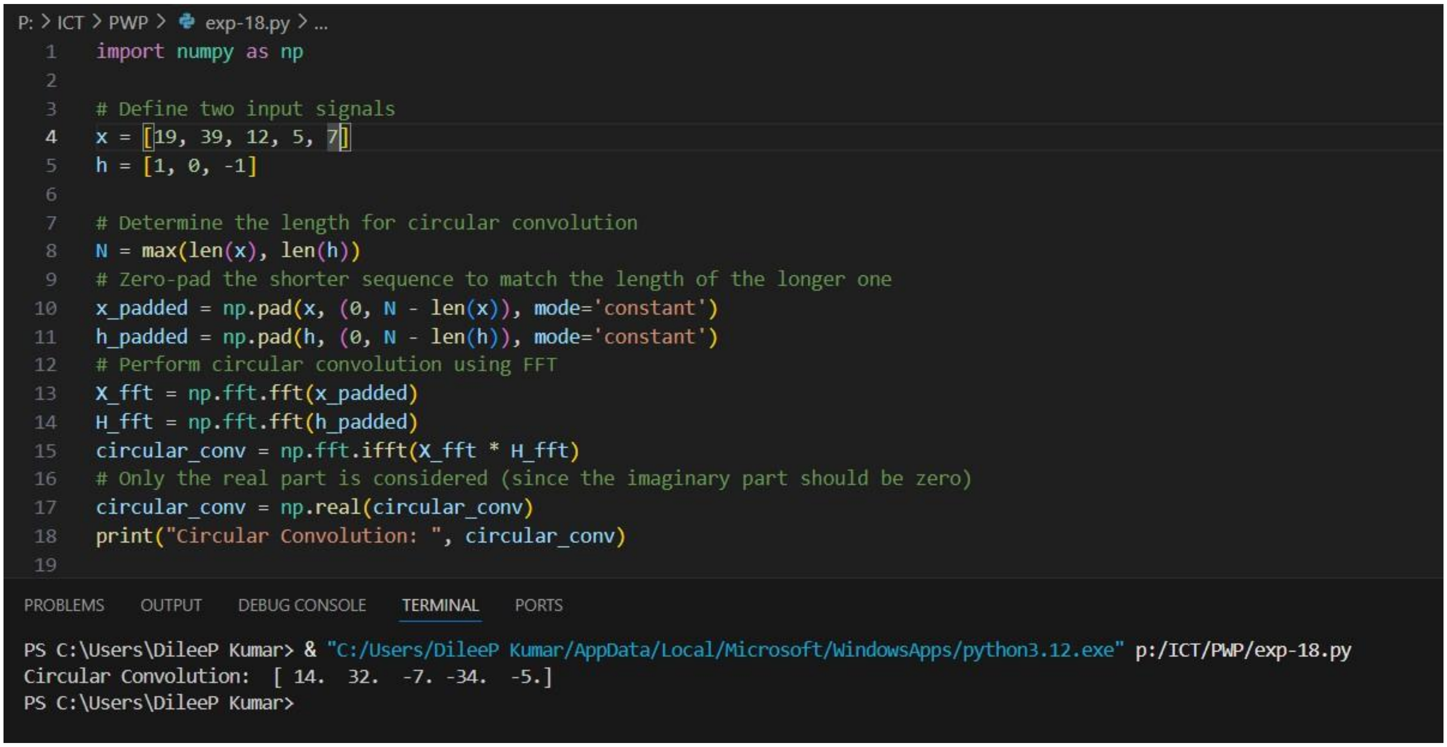
Circular convolution can be implemented by first padding the shorter sequence to the length of the longer one and then using the Fast Fourier Transform (FFT) for efficient computation.

Example 2:

```
import numpy as np

# Define two input signals
x = [1, 2, 3, 4]
h = [1, 0, -1]

# Determine the length for circular convolution
N = max(len(x), len(h))
# Zero-pad the shorter sequence to match the length of the longer one
x_padded = np.pad(x, (0, N - len(x)), mode='constant')
h_padded = np.pad(h, (0, N - len(h)), mode='constant')
# Perform circular convolution using FFT
X_fft = np.fft.fft(x_padded)
H_fft = np.fft.fft(h_padded)
circular_conv = np.fft.ifft(X_fft * H_fft)
# Only the real part is considered (since the imaginary part should be zero)
circular_conv = np.real(circular_conv)
```

print("Circular Convolution: ", circular_conv)

output:



```python
import numpy as np

# Define two input signals
x = [19, 39, 12, 5, 7]
h = [1, 0, -1]

# Determine the length for circular convolution
N = max(len(x), len(h))
# Zero-pad the shorter sequence to match the length of the longer one
x_padded = np.pad(x, (0, N - len(x)), mode='constant')
h_padded = np.pad(h, (0, N - len(h)), mode='constant')
# Perform circular convolution using FFT
X_fft = np.fft.fft(x_padded)
H_fft = np.fft.fft(h_padded)
circular_conv = np.fft.ifft(X_fft * H_fft)
# Only the real part is considered (since the imaginary part should be zero)
circular_conv = np.real(circular_conv)
print("Circular Convolution: ", circular_conv)
```

```
PS C:\Users\DileeP Kumar> & "C:/Users/DileeP Kumar/AppData/Local/Microsoft/WindowsApps/python3.12.exe" p:/ICT/PWP/exp-18.py
Circular Convolution:  [ 14.  32.  -7. -34.  -5.]
PS C:\Users\DileeP Kumar>
```

Key Points:

- **Linear convolution**: Extends beyond the input lengths.
- **Circular convolution**: Assumes periodicity, so the result has the same length as the longest sequence.

What is Correlation?

Correlation is a statistical measure that describes the extent to which two signals or datasets are related. It measures how much two variables change together. In signal processing, correlation is used to find similarities between two signals or between different parts of the same signal. Correlation helps in detecting patterns, shifts, or commonalities in signals.

Types of Correlation:

1. **Cross-Correlation**: Measures the similarity between two different signals as a function of time-lag.

2. **Autocorrelation**: Measures the similarity of a signal with a delayed version of itself, also as a function of time-lag.

Cross-Correlation

Cross-correlation is a measure of similarity between two different signals. It shows how one signal correlates with another signal over time, meaning how much one signal resembles a shifted version of another.

Mathematically, the cross-correlation of two discrete signals $x[n]$ and $y[n]$ is defined as:

$$R_{x,y}[k] = \sum_{n} x[n].\, y[n+k]$$

Where $x[n]$ and $y[n]$ are the two signals. k is the time-lag. $R_{x,y}[k]$ is the cross-correlation at lag k**.**

Cross-correlation shifts one signal relative to the other and calculates the dot product (similarity) for each shift. The peak in the cross-correlation function indicates the time-shift where the signals are most similar.

Example

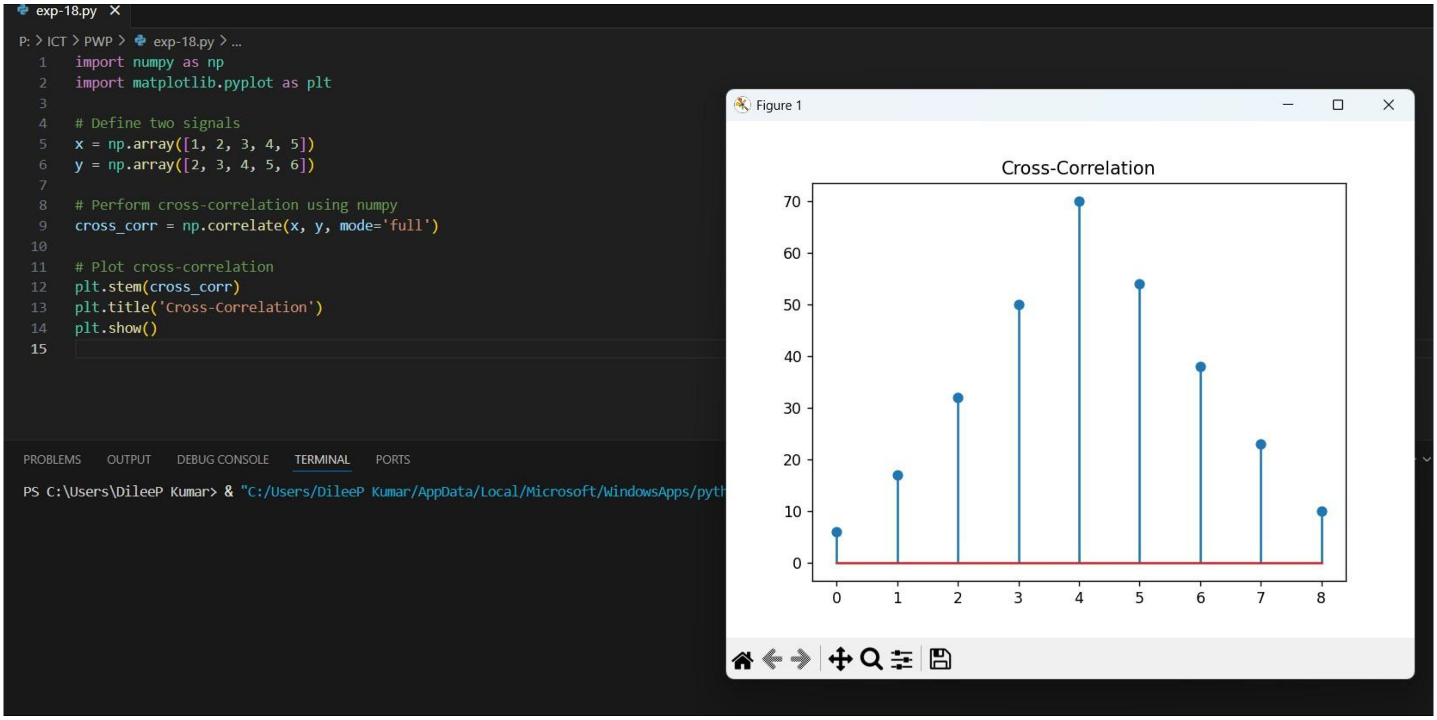
```
import numpy as np

import matplotlib.pyplot as plt


# Define two signals

x = np.array([1, 2, 3, 4, 5])

y = np.array([2, 3, 4, 5, 6])


# Perform cross-correlation using numpy

cross_corr = np.correlate(x, y, mode='full')


# Plot cross-correlation
```

plt.stem(cross_corr)

plt.title('Cross-Correlation')

plt.show()

output:



```
exp-18.py
P: > ICT > PWP > exp-18.py > ...
import numpy as np
import matplotlib.pyplot as plt

# Define two signals
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 4, 5, 6])

# Perform cross-correlation using numpy
cross_corr = np.correlate(x, y, mode='full')

# Plot cross-correlation
plt.stem(cross_corr)
plt.title('Cross-Correlation')
plt.show()
```

```
PS C:\Users\DileeP Kumar> & "C:/Users/DileeP Kumar/AppData/Local/Microsoft/WindowsApps/pyth
```

Autocorrelation

Autocorrelation is a special case of cross-correlation, where a signal is correlated with a delayed version of itself. It shows how much a signal resembles its own shifted version.

Mathematically, the autocorrelation of a signal $[n]$ and $y[n]$ is defined as:

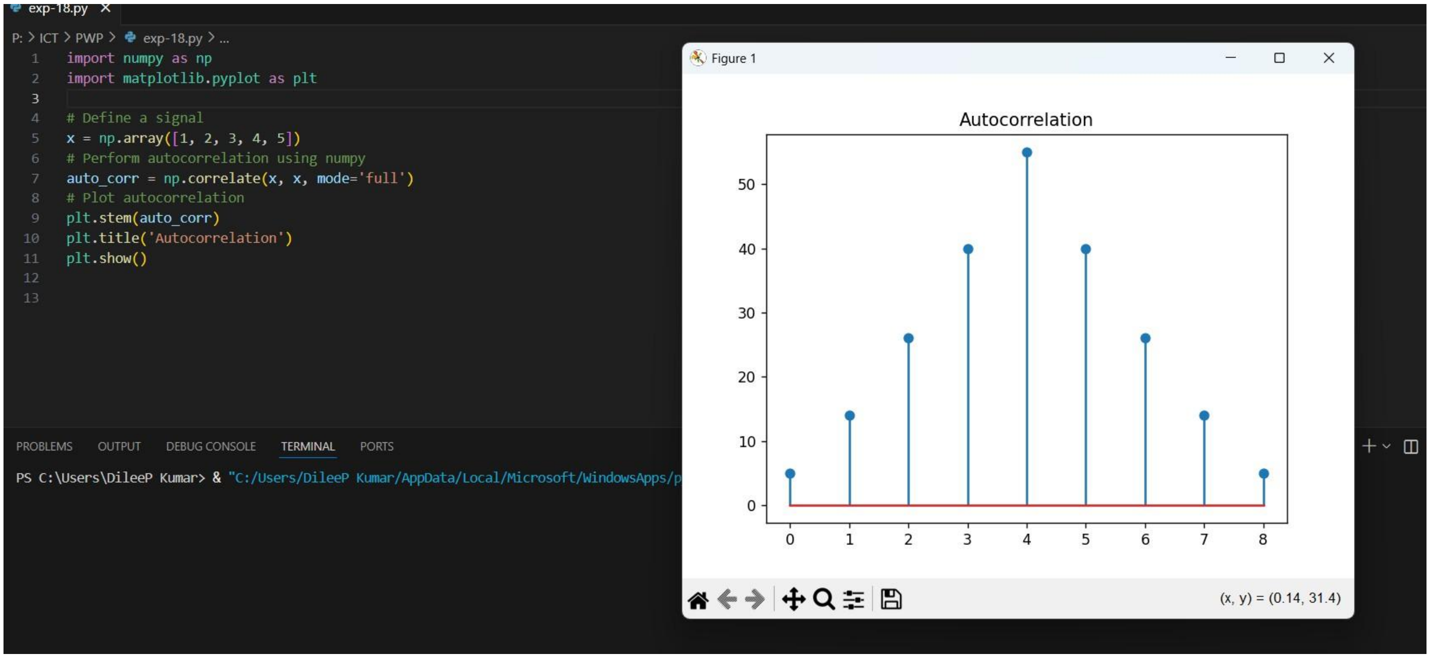
$$R_{x,x}[k] = \sum_{n} x[n].\, x[n+k]$$

Where $x[n]$ signal. k is the time-lag. $R_{x,x}[k]$ is the auto-correlation at lag k.

Autocorrelation measures the internal structure of a signal by checking how it correlates with itself over time. A high autocorrelation at some lag indicates a repeating or periodic pattern in the signal.

```
# Define a signal

x = np.array([1, 2, 3, 4, 5])

# Perform autocorrelation using numpy

auto_corr = np.correlate(x, x, mode='full')

# Plot autocorrelation

plt.stem(auto_corr)

plt.title('Autocorrelation')

plt.show()
```

output:

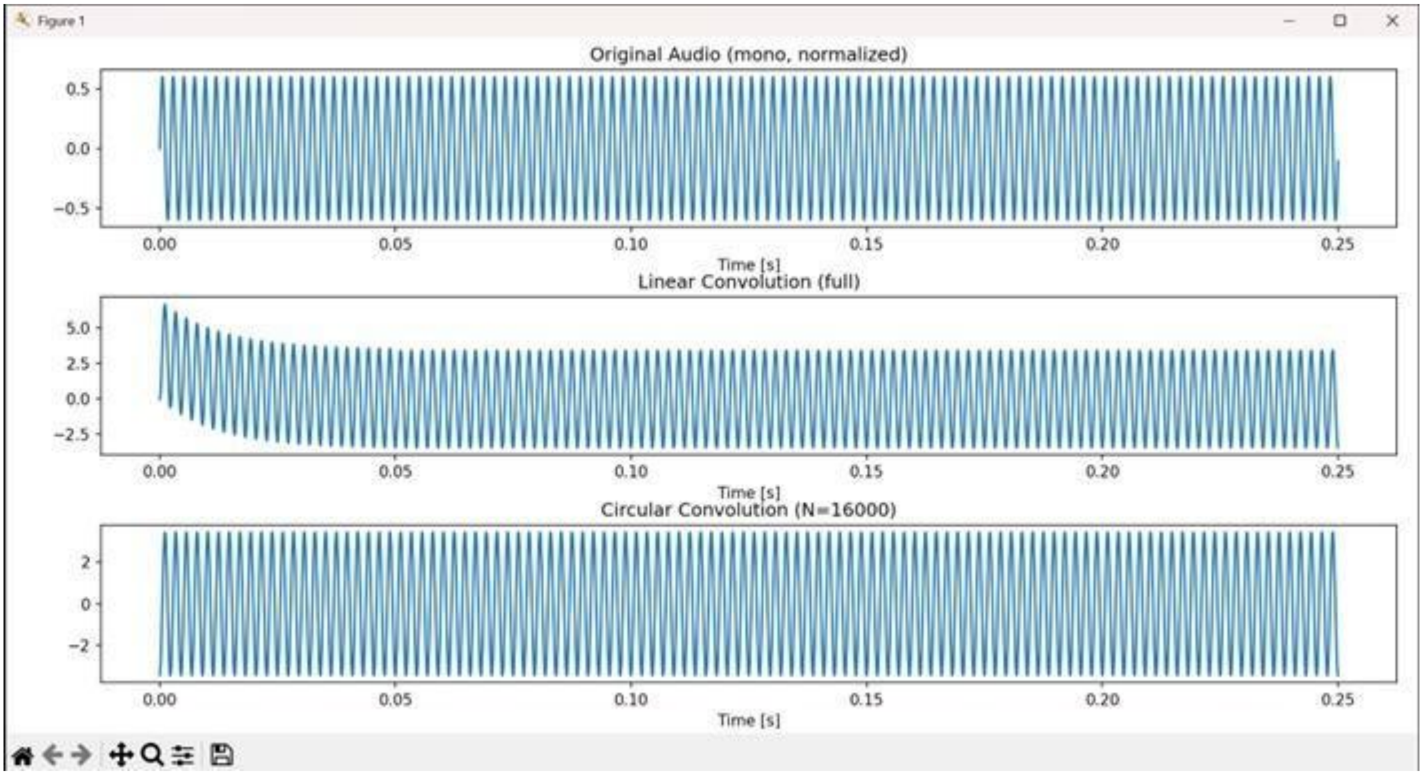


```
import numpy as np
import matplotlib.pyplot as plt

# Define a signal
x = np.array([1, 2, 3, 4, 5])
# Perform autocorrelation using numpy
auto_corr = np.correlate(x, x, mode='full')
# Plot autocorrelation
plt.stem(auto_corr)
plt.title('Autocorrelation')
plt.show()
```

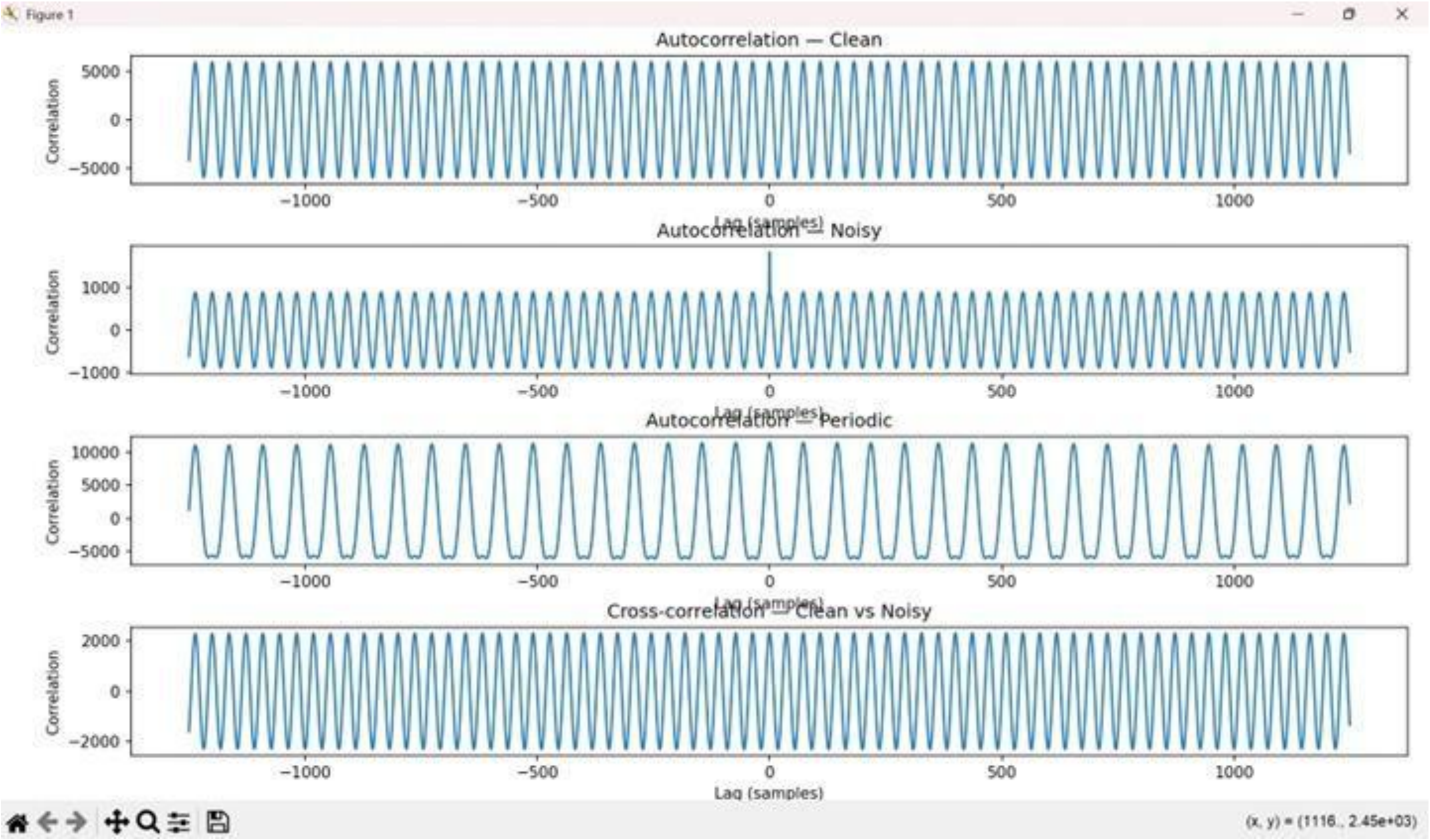
```
PS C:\Users\DileeP Kumar> & "C:/Users/DileeP Kumar/AppData/Local/Microsoft/WindowsApps/p
```

Post Lab Exercise:

- Use Python to create a script that performs both **linear** and **circular convolution** on an audio file with an impulse response. Compare and visualize the results.



- Use Python to implement both **cross-correlation** and **autocorrelation** on a set of audio files (clean_audio.wav, noisy_audio.wav, periodic_audio.wav). Visualize and compare the results.



Github link: