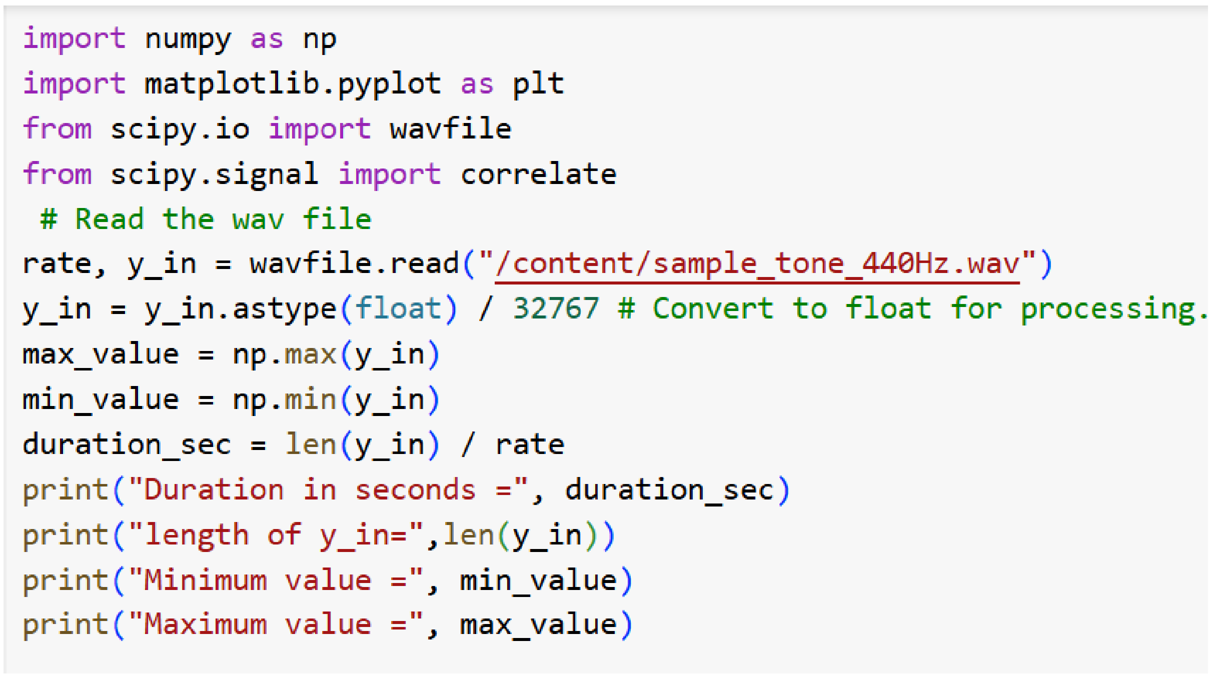
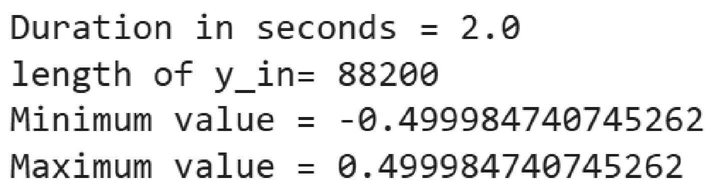
Experiment 3: Echo Cancellation and DFT Properties Verification

2.1 Load the audio file

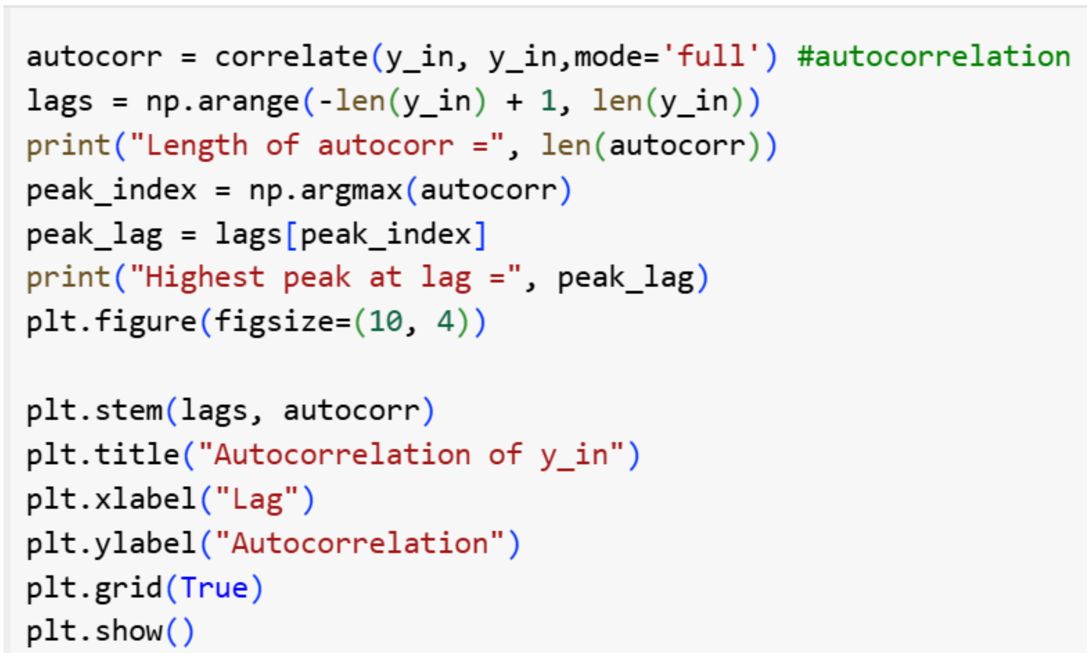


```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import correlate
 # Read the wav file
rate, y_in = wavfile.read("/content/sample_tone_440Hz.wav")
y_in = y_in.astype(float) / 32767 # Convert to float for processing.
max_value = np.max(y_in)
min_value = np.min(y_in)
duration_sec = len(y_in) / rate
print("Duration in seconds =", duration_sec)
print("length of y_in=",len(y_in))
print("Minimum value =", min_value)
print("Maximum value =", max_value)
```



```
Duration in seconds = 2.0
length of y_in= 88200
Minimum value = -0.499984740745262
Maximum value = 0.499984740745262
```

2.2 Echo Delay using Autocorrelation



```
autocorr = correlate(y_in, y_in,mode='full') #autocorrelation
lags = np.arange(-len(y_in) + 1, len(y_in))
print("Length of autocorr =", len(autocorr))
peak_index = np.argmax(autocorr)
peak_lag = lags[peak_index]
print("Highest peak at lag =", peak_lag)
plt.figure(figsize=(10, 4))

plt.stem(lags, autocorr)
plt.title("Autocorrelation of y_in")
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.grid(True)
plt.show()
```

Figure 2: python code

```
Length of autocorr = 176399
Highest peak at lag = 0
```

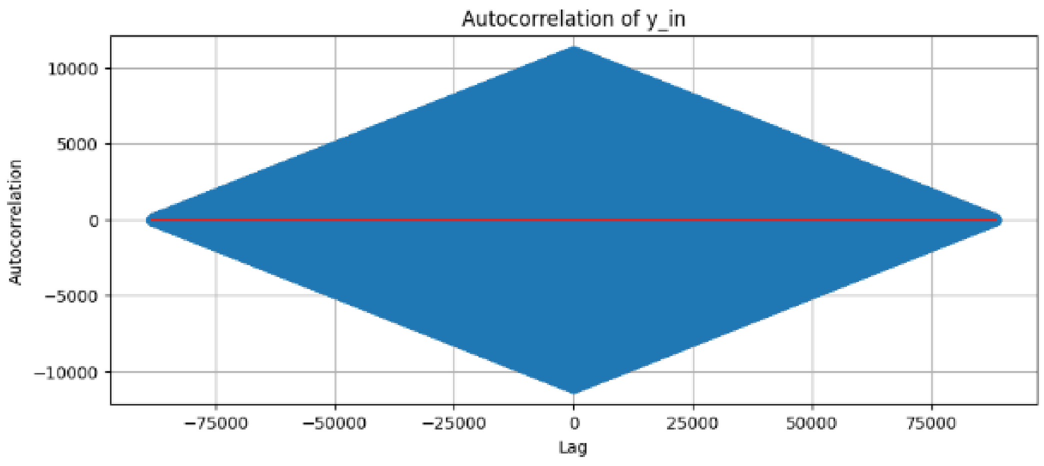


Figure 3: autocorrelation plot

```python
autocorr = correlate(y_in, y_in, mode='full')
lags = np.arange(-len(y_in) + 1, len(y_in))

peaks, _ = find_peaks(autocorr)
peak_lags = lags[peaks]
peak_vals = autocorr[peaks]

sorted_indices = np.argsort(peak_vals)[::-1]

peak1_lag = peak_lags[sorted_indices[0]]
peak2_lag = peak_lags[sorted_indices[1]]

peak1_val = peak_vals[sorted_indices[0]]
peak2_val = peak_vals[sorted_indices[1]]

print("first peak Lag =", peak1_lag)
print(" Autocorrelation Value =", peak1_val)
print("second peak Lag =", peak2_lag)
print(" Autocorrelation Value =", peak2_val)
print("Delay =",peak2_lag)
```

```python
plt.figure(figsize=(10, 5))
plt.plot(lags, autocorr, label='Autocorrelation')
plt.plot(peak1_lag, autocorr[lags == peak1_lag], 'ro', label=f'Peak1 (Lag = {peak1_lag})')
plt.plot(peak2_lag, autocorr[lags == peak2_lag], 'go', label=f'Peak2 (Lag = {peak2_lag})')

plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation of Echoed Signal')
plt.legend()
plt.xlim(0,1000)
plt.grid(True)
plt.show()
```

```
first peak Lag = 0
 Autocorrelation Value = 11024.14529279037
second peak Lag = 100
 Autocorrelation Value = 11010.50028641732
Delay = 100
```

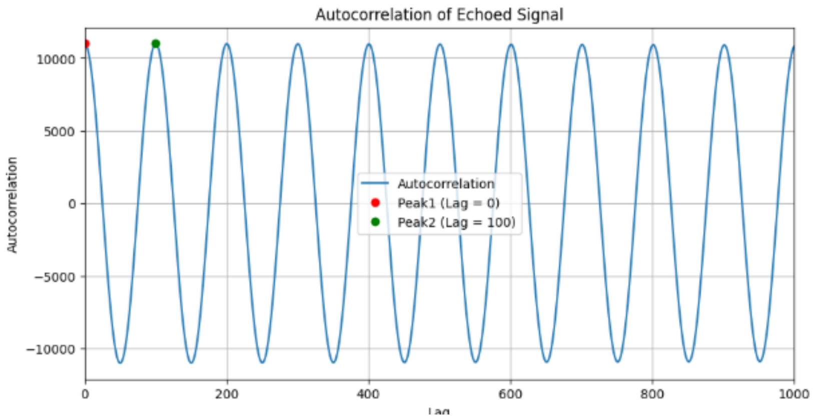
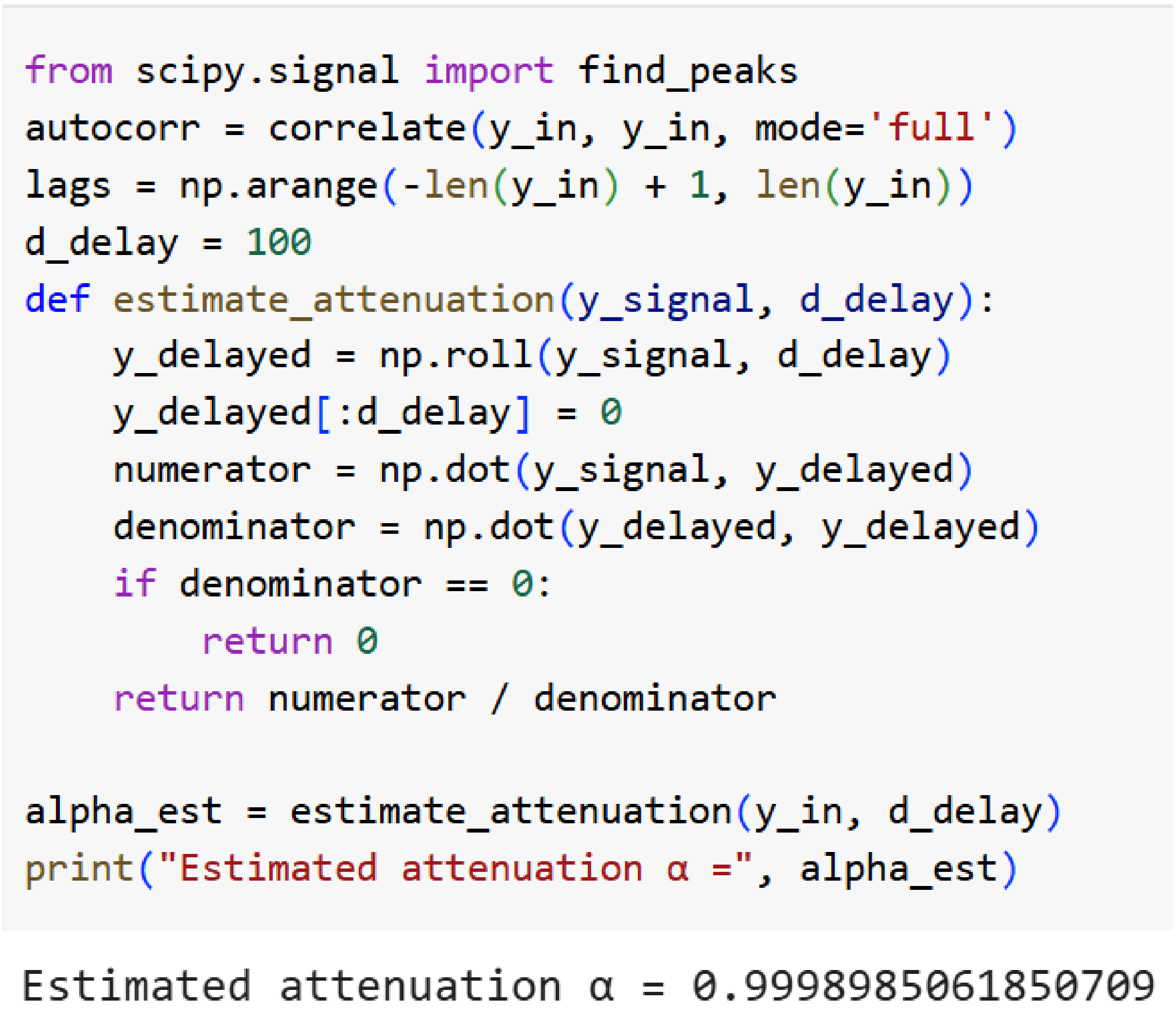


Figure 4: Output Plot

2.3 Estimation of Attenuation

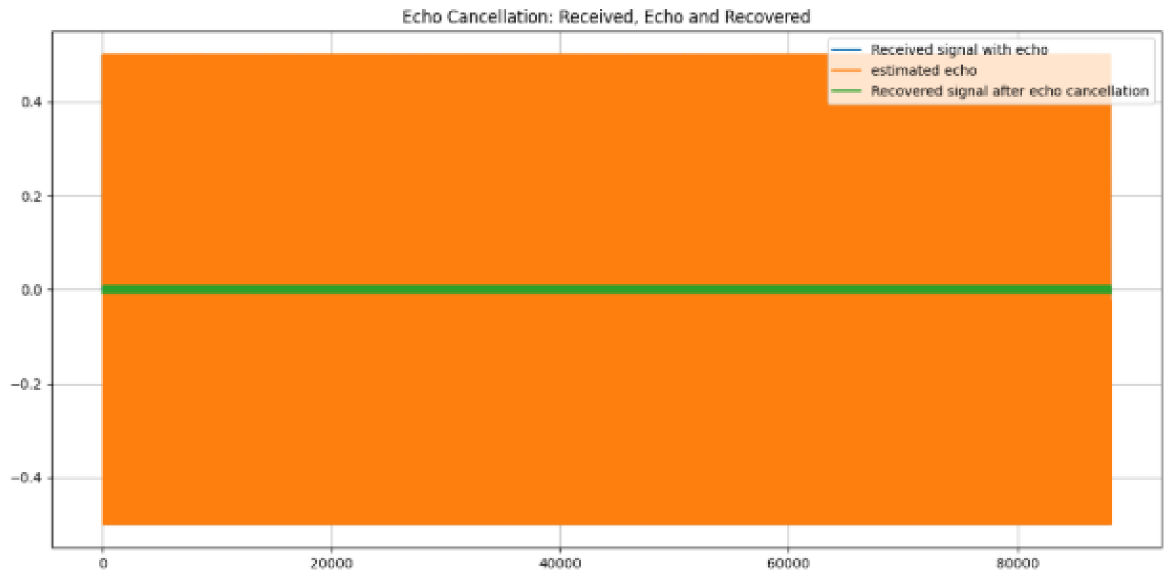


```python
from scipy.signal import find_peaks
autocorr = correlate(y_in, y_in, mode='full')
lags = np.arange(-len(y_in) + 1, len(y_in))
d_delay = 100
def estimate_attenuation(y_signal, d_delay):
    y_delayed = np.roll(y_signal, d_delay)
    y_delayed[:d_delay] = 0
    numerator = np.dot(y_signal, y_delayed)
    denominator = np.dot(y_delayed, y_delayed)
    if denominator == 0:
        return 0
    return numerator / denominator

alpha_est = estimate_attenuation(y_in, d_delay)
print("Estimated attenuation α =", alpha_est)
```

```
Estimated attenuation α = 0.9998985061850709
```

2.4 Echo Cancellation

```python
d_delay = 100
alpha_est = 0.9998985061850709
estimated_echo = np.roll(y_in, d_delay)
estimated_echo *= alpha_est
x_hat = y_in - estimated_echo

plt.figure(figsize=(12, 6))
plt.plot(y_in, label="Received signal with echo")
plt.plot(estimated_echo,label="estimated echo")
plt.plot(x_hat, label="Recovered signal after echo cancellation")
plt.legend()
plt.title("Echo Cancellation: Received, Echo and Recovered")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Part 2: DFT Properties Verification

1. Time Reversal Property

x[(-n)N] DFT $\rightarrow$ X[(-k)N] = X[N-k].

```python
from numpy.fft import fft
N = len(y_in)
y_in = np.array(y_in)

y_in_rev = np.zeros_like(y_in)
y_in_rev[0] = y_in[0]
y_in_rev[1:] = y_in[:0:-1]

Y = fft(y_in)
Y_rev = fft(y_in_rev)

k_vals = np.arange(N)
Y_N_refl_shift = Y[np.mod(-k_vals, N)]  # X[N-k] = X[(-k)%N]

verification_err = np.sum(np.abs(Y_N_refl_shift - Y_rev)**2)
print("The value of the verification error =", verification_err)
```

```
The value of the verification error = 2.129757581245839e-22
```

2.Circular Time Shift Property

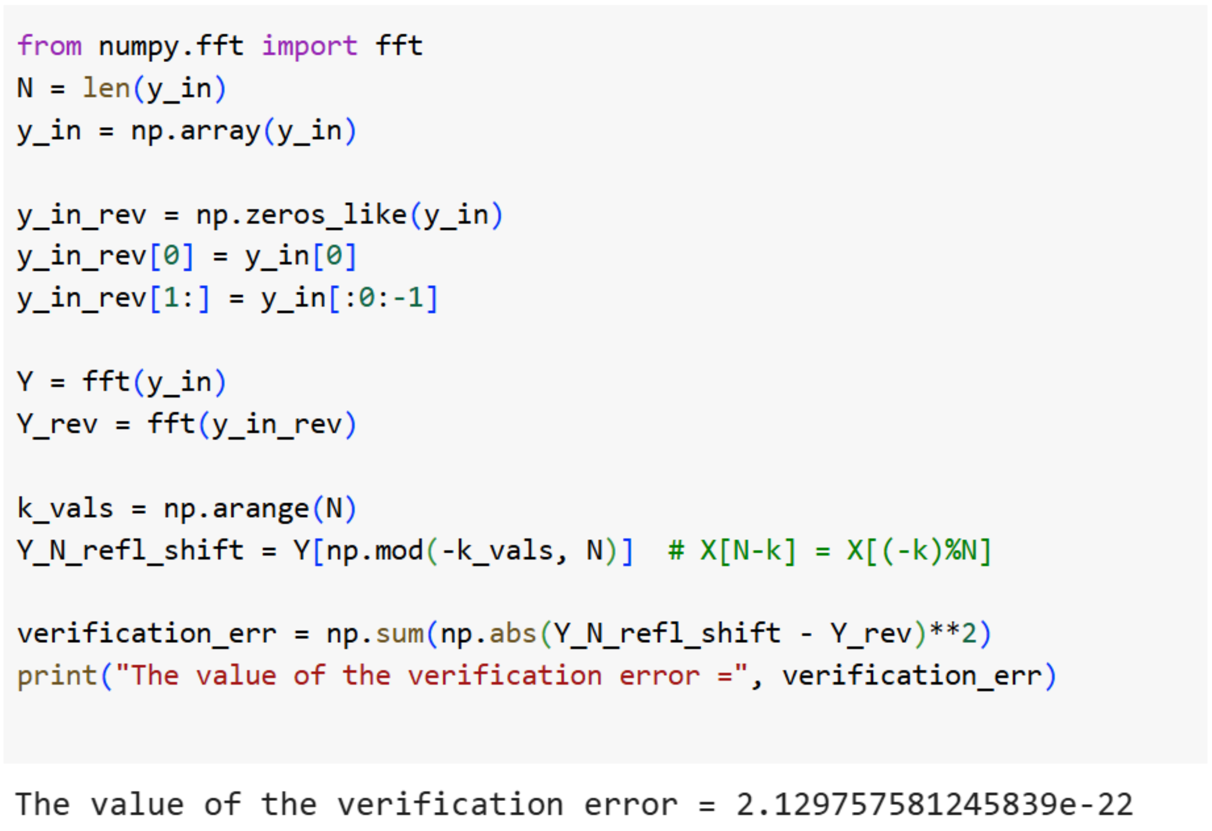
x[(n-n0)N] DFT →X[k]e-j2*pi*kn0/N.

```python
N = len(y_in)
n0 = 12
k = np.arange(N)

y_in_shifted = np.roll(y_in, n0)


Y = np.fft.fft(y_in)
Y_shifted_dft = np.fft.fft(y_in_shifted)

Y_expected_shift_dft = Y * np.exp(-1j * 2 * np.pi * k * n0 / N)

verification_err = np.sum(np.abs(Y_expected_shift_dft - Y_shifted_dft) ** 2)

print("The value of the verification error =", verification_err)
```

```
The value of the verification error = 7.831868761409966e-21
```

3. Frequency Shift Property

x[n]ej2*pi*k0n/N DFT →X[(k-k0)N]

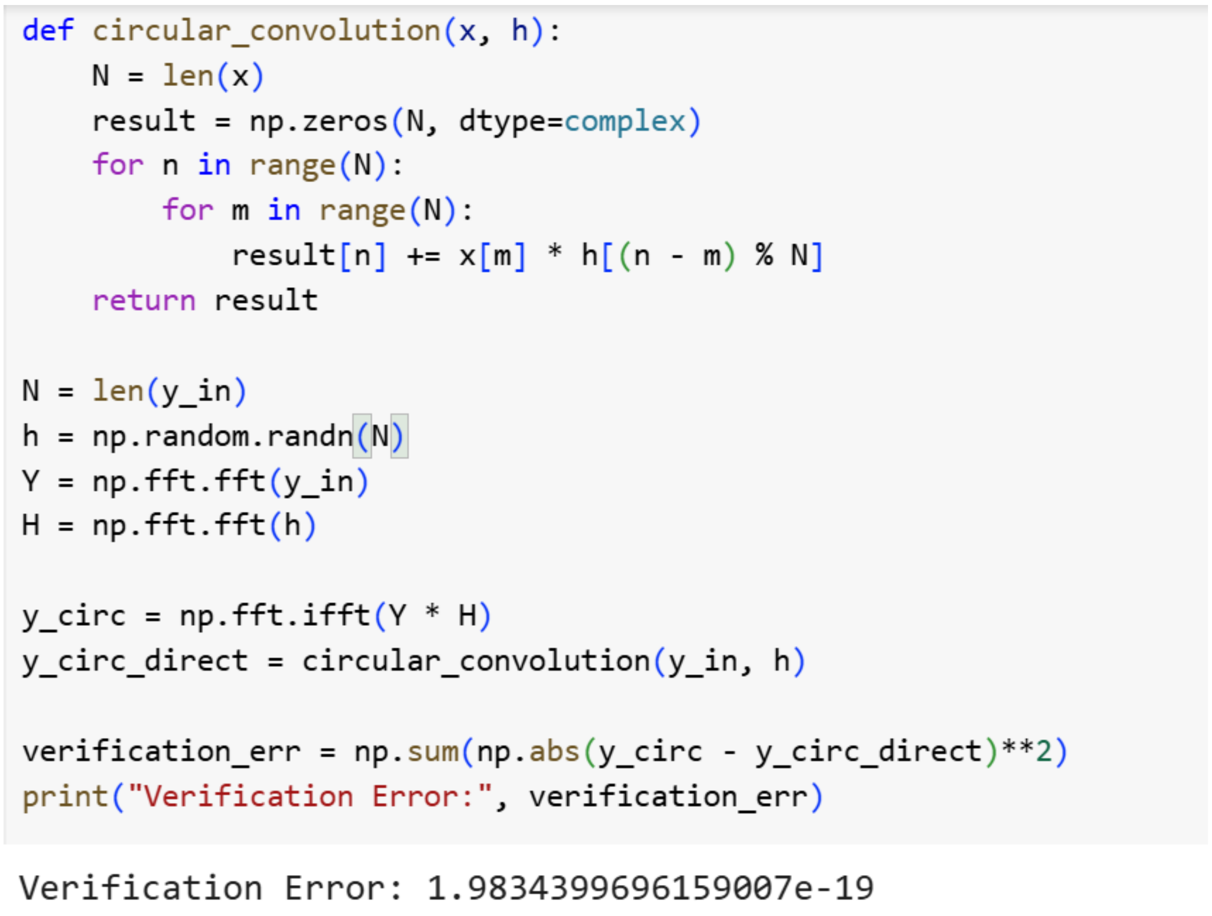


```python
from numpy.fft import fft
N = len(y_in)
y_in = np.array(y_in)

y_in_rev = np.zeros_like(y_in)
y_in_rev[0] = y_in[0]
y_in_rev[1:] = y_in[:0:-1]

Y = fft(y_in)
Y_rev = fft(y_in_rev)

k_vals = np.arange(N)
Y_N_refl_shift = Y[np.mod(-k_vals, N)]  # X[N-k] = X[(-k)%N]

verification_err = np.sum(np.abs(Y_N_refl_shift - Y_rev)**2)
print("The value of the verification error =", verification_err)
```

```
The value of the verification error = 2.129757581245839e-22
```

4.Circular Convolution Property

x[n] o h[n]=IDFT(X[k]H[k])



```python
def circular_convolution(x, h):
    N = len(x)
    result = np.zeros(N, dtype=complex)
    for n in range(N):
        for m in range(N):
            result[n] += x[m] * h[(n - m) % N]
    return result

N = len(y_in)
h = np.random.randn(N)
Y = np.fft.fft(y_in)
H = np.fft.fft(h)

y_circ = np.fft.ifft(Y * H)
y_circ_direct = circular_convolution(y_in, h)

verification_err = np.sum(np.abs(y_circ - y_circ_direct)**2)
print("Verification Error:", verification_err)
```

```
Verification Error: 1.9834399696159007e-19
```