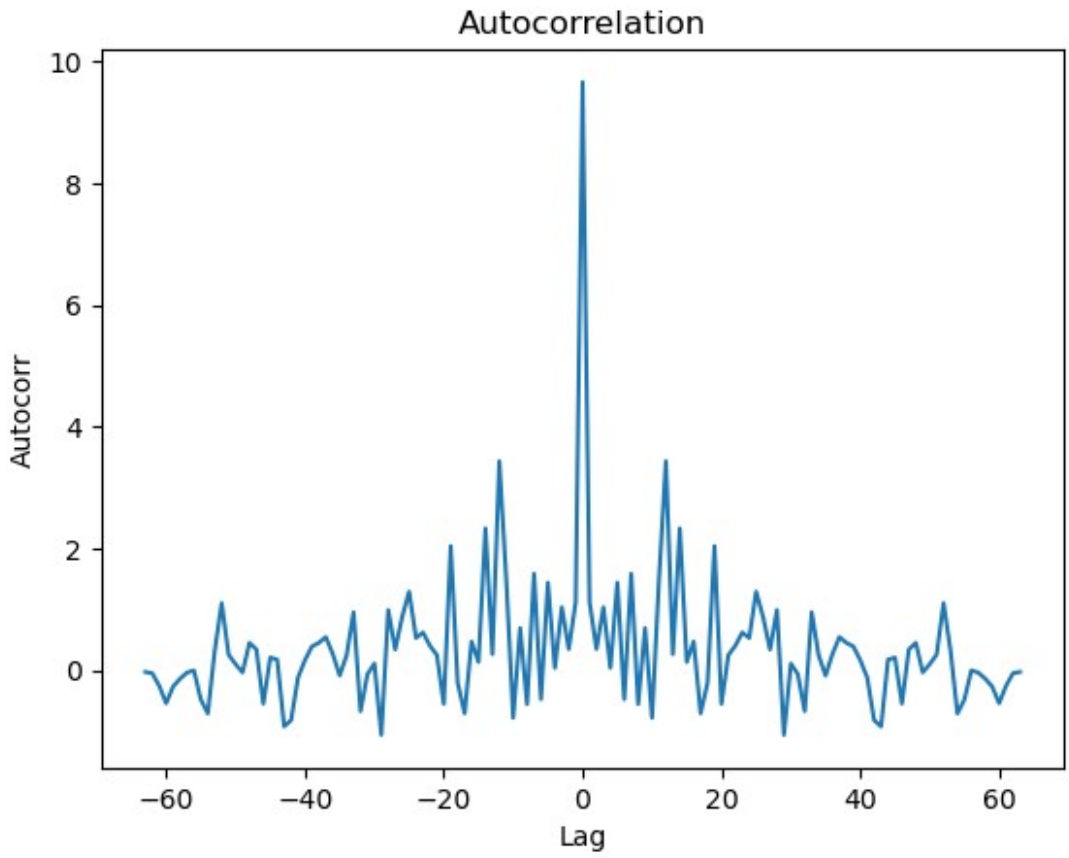
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import correlate
 # Read the wav file
rate, y_in = wavfile.read("echo_signal_random.wav")
y_in = y_in.astype(float) / 32767 # Convert to float for processing.
print('maximum value of audio sample :',np.max(y_in))
print('minimum value of audio sample :',np.min(y_in))
print('length value of audio sample :',len(y_in))
```

```
maximum value of audio sample : 0.7250282296212653
minimum value of audio sample : -1.0
length value of audio sample : 64
```

```
autocorr = correlate(y_in, y_in,mode='full')
lags = np.arange(-len(y_in)+1, len(y_in))
print('Total length of autocorrelation=',2*len(y_in)-1)
max_index=np.argmax(autocorr)
max_loc=lags[max_index]
print('Highest peak is at lag =',max_loc)
plt.plot(lags,autocorr)
plt.title("Autocorrelation")
plt.xlabel('Lag')
plt.ylabel('Autocorr')
plt.show()
```

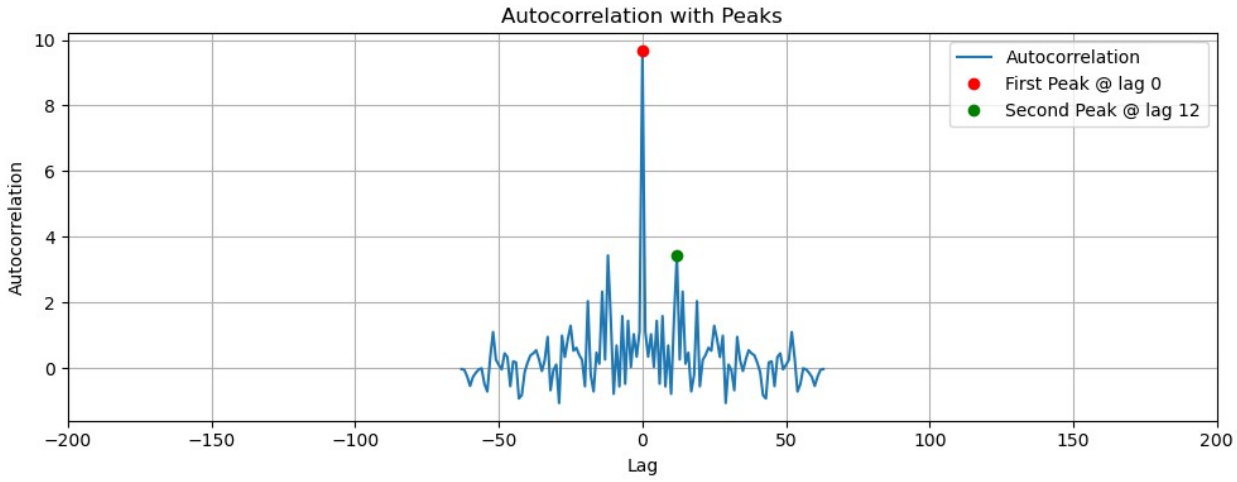
```
Total length of autocorrelation= 127
Highest peak is at lag = 0
```




```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import correlate

# Compute autocorrelation
autocorr = correlate(y_in, y_in, mode='full')
lags = np.arange(-len(y_in)+1, len(y_in))

# Find first (maximum) peak (should be at lag = 0)
first_peak_index = np.argmax(autocorr)
first_peak_lag = lags[first_peak_index]

# Find second peak (excluding center part)
# Search only on the right side of lag=0
center_index = len(y_in) - 1  # this corresponds to lag=0
right_side = autocorr[center_index + 1:]

second_peak_index = np.argmax(right_side)
second_peak_lag = lags[center_index + 1 + second_peak_index]
estimated_delay = lags[second_peak_index]
print("Estimated delay =", estimated_delay)
```

```
# Plot
plt.figure(figsize=(10, 4))
plt.plot(lags, autocorr, label="Autocorrelation")
plt.plot(first_peak_lag, autocorr[first_peak_index], 'ro',
label=f"First Peak @ lag {first_peak_lag}")
plt.plot(second_peak_lag, autocorr[center_index + 1 +
second_peak_index], 'go', label=f"Second Peak @ lag
{second_peak_lag}")
plt.title("Autocorrelation with Peaks")
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.xlim(-200,200)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Print values
print("First Peak: Index =", first_peak_index, ", Lag =",
first_peak_lag)
print("Second Peak: Index =", center_index + 1 + second_peak_index, ",
Lag =", second_peak_lag)
```

```
Estimated delay = -52
```



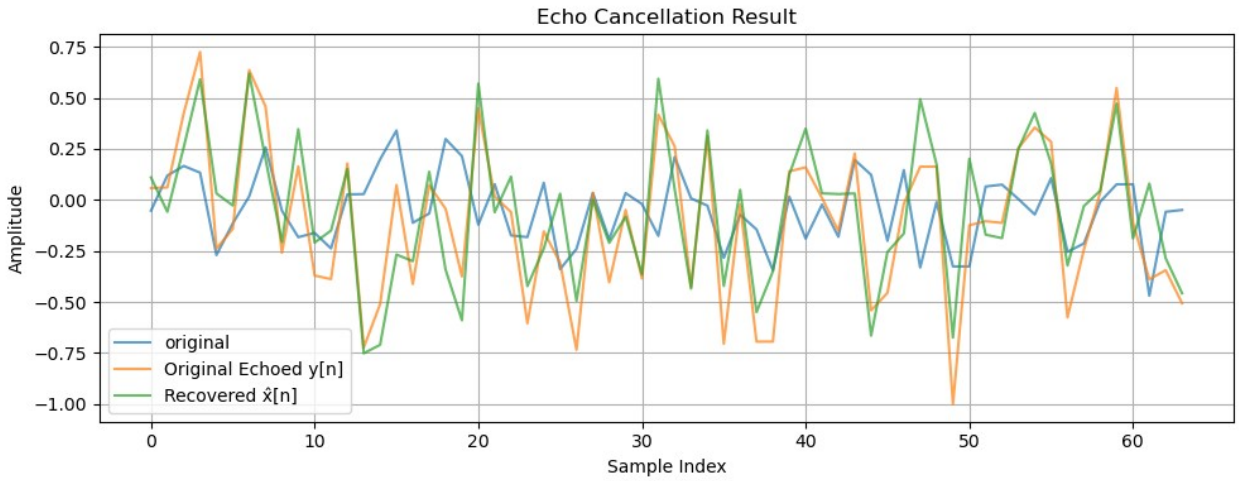

```
First Peak: Index = 63 , Lag = 0
Second Peak: Index = 75 , Lag = 12
```

```
def estimate_attenuation(y_signal, d_delay):
    y_delayed = np.roll(y_signal, d_delay)
    numerator = np.dot(y_signal, y_delayed)
    denominator = np.dot(y_delayed, y_delayed)
    if denominator == 0:
        return 0
```

```
    return numerator / denominator

estimated_alpha = estimate_attenuation(y_in, estimated_delay)
print("Estimated attenuation alpha =", estimated_alpha)
```

```
Estimated attenuation alpha = 0.46930832874074707
```

```
estimated_echo = np.roll(y_in, estimated_delay) * estimated_alpha
x_hat = y_in - estimated_echo

plt.figure(figsize=(10, 4))
plt.plot(estimated_echo,label='original',alpha=0.7)
plt.plot(y_in, label='Original Echoed y[n]', alpha=0.7)
plt.plot(x_hat, label='Recovered x̂[n]', alpha=0.7)
plt.title("Echo Cancellation Result")
plt.xlabel("Sample Index")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
import numpy as np
from scipy.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 = 1.508696481235185e-29
```

```
import numpy as np

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 = 8.466182521183715e-27
```

```
N=len(y_in)
n=np.arange(N)
k0 = 3
y_in_freq_shifted = y_in * np.exp(1j * 2 * np.pi * k0 * n / N)
Y_freq_shifted = np.fft.fft(y_in_freq_shifted)
Y_expected_freq_shift = np.roll(Y, k0)
verification_err_fs = np.sum(np.abs(Y_freq_shifted -
Y_expected_freq_shift)**2)
print(" Frequency shift verification error =", verification_err_fs)
```

```
 Frequency shift verification error = 2.6565653415119426e-28
```

```
import numpy as np
from scipy.io import wavfile
rate, y_in = wavfile.read("echo_signal_random.wav")
y_in = y_in.astype(float) / 32767 # Convert to float for processing
y_in_1000=y_in[:1000]
N=len(y_in_1000)
def circular_convolution(x, h):
    N = len(x)
    x = np.array(x)
```

```
    h = np.array(h)
    result = np.zeros(N)  # Allocate output array
    for n in range(N):
        for m in range(N):
            result[n] += x[m] * h[(n - m) % N]  # Circular index
    return result

h = np.random.randn(N)  # Random Gaussian sequence
y_circ = np.real(np.fft.ifft(np.fft.fft(y_in_1000) * np.fft.fft(h)))
y_circ_direct = circular_convolution(y_in_1000, h)
verification_err = np.sum(np.abs(y_circ - y_circ_direct)**2)
print("Circular convolution verification error:", verification_err)
```

```
Circular convolution verification error: 7.439636263574566e-29
```