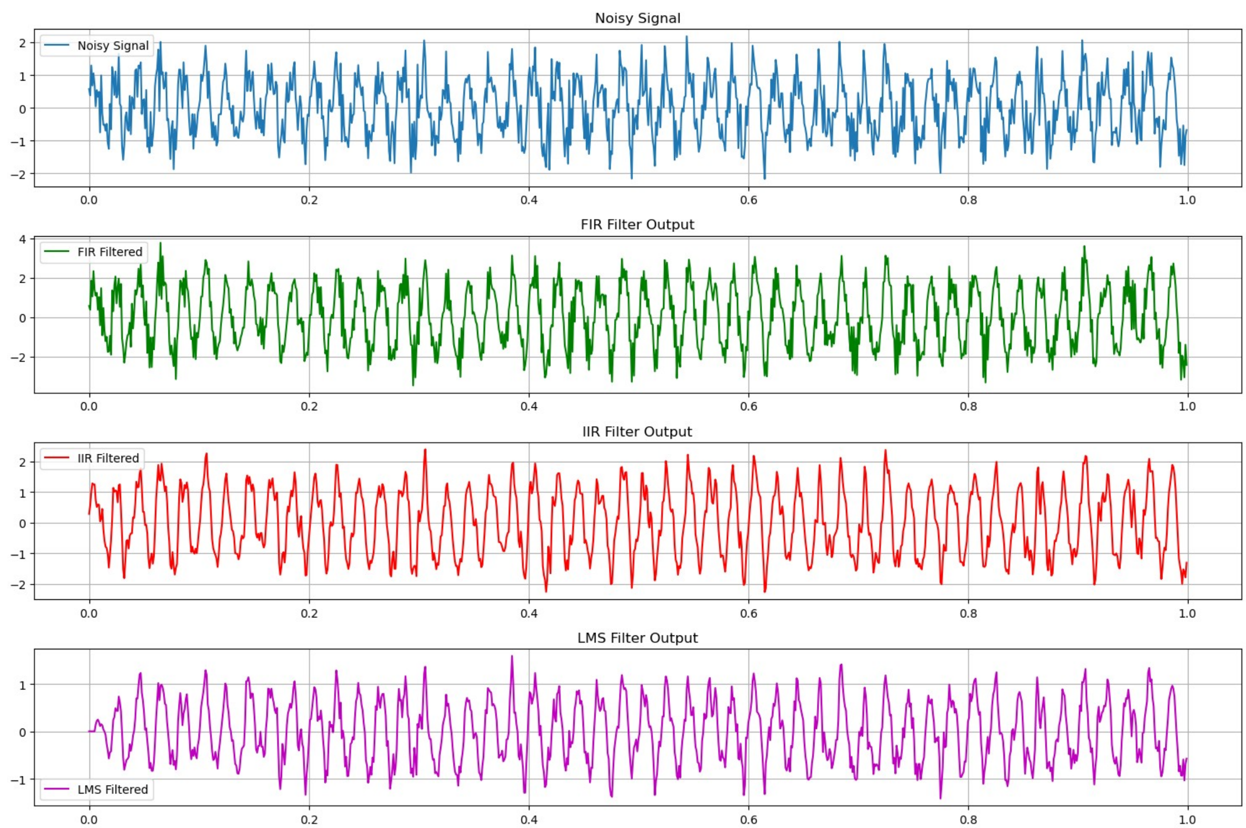
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter

# Generowanie zakłóconego sygnału sinusoidalnego
fs = 1000  # Częstotliwość próbkowania
t = np.linspace(0, 1, fs, endpoint=False)
f = 50  # Częstotliwość sygnału
signal = np.sin(2 * np.pi * f * t)  # Sygnał czysty
noise = np.random.normal(0, 0.5, signal.shape)  # Szum
noisy_signal = signal + noise  # Sygnał zakłócony

# FIR Filter
b_fir = [1, 0, 1]  # Zmiana współczynników dla Wariantu 8
filtered_fir = lfilter(b_fir, [1], noisy_signal)

# IIR Filter
b_iir = [0.5, 0.5]  # Zmiana współczynników dla Wariantu 8
a_iir = [1, -0.3]   # Zmiana współczynników dla Wariantu 8
filtered_iir = lfilter(b_iir, a_iir, noisy_signal)

# LMS Adaptive Filter
def lms_filter(x, d, mu, M):
    N = len(x)
    w = np.zeros(M)
    y = np.zeros(N)
    e = np.zeros(N)

    for n in range(M, N):
        x_n = x[n:n-M:-1]  # Fragment sygnału wejściowego
        y[n] = np.dot(w, x_n)
        e[n] = d[n] - y[n]
        w += mu * e[n] * x_n

    return y, e

mu = 0.05
M = 5  # Zmiana długości filtru LMS dla Wariantu 8
desired_signal = signal  # Sygnał czysty jako odniesienie
filtered_lms, error = lms_filter(noisy_signal, desired_signal, mu, M)

# Wizualizacja wyników
plt.figure(figsize=(15, 10))

plt.subplot(4, 1, 1)
plt.plot(t, noisy_signal, label='Noisy Signal')
plt.title("Noisy Signal")
plt.grid()
plt.legend()
```

```
plt.subplot(4, 1, 2)
plt.plot(t, filtered_fir, label='FIR Filtered', color='g')
plt.title("FIR Filter Output")
plt.grid()
plt.legend()

plt.subplot(4, 1, 3)
plt.plot(t, filtered_iir, label='IIR Filtered', color='r')
plt.title("IIR Filter Output")
plt.grid()
plt.legend()

plt.subplot(4, 1, 4)
plt.plot(t, filtered_lms, label='LMS Filtered', color='m')
plt.title("LMS Filter Output")
plt.grid()
plt.legend()

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

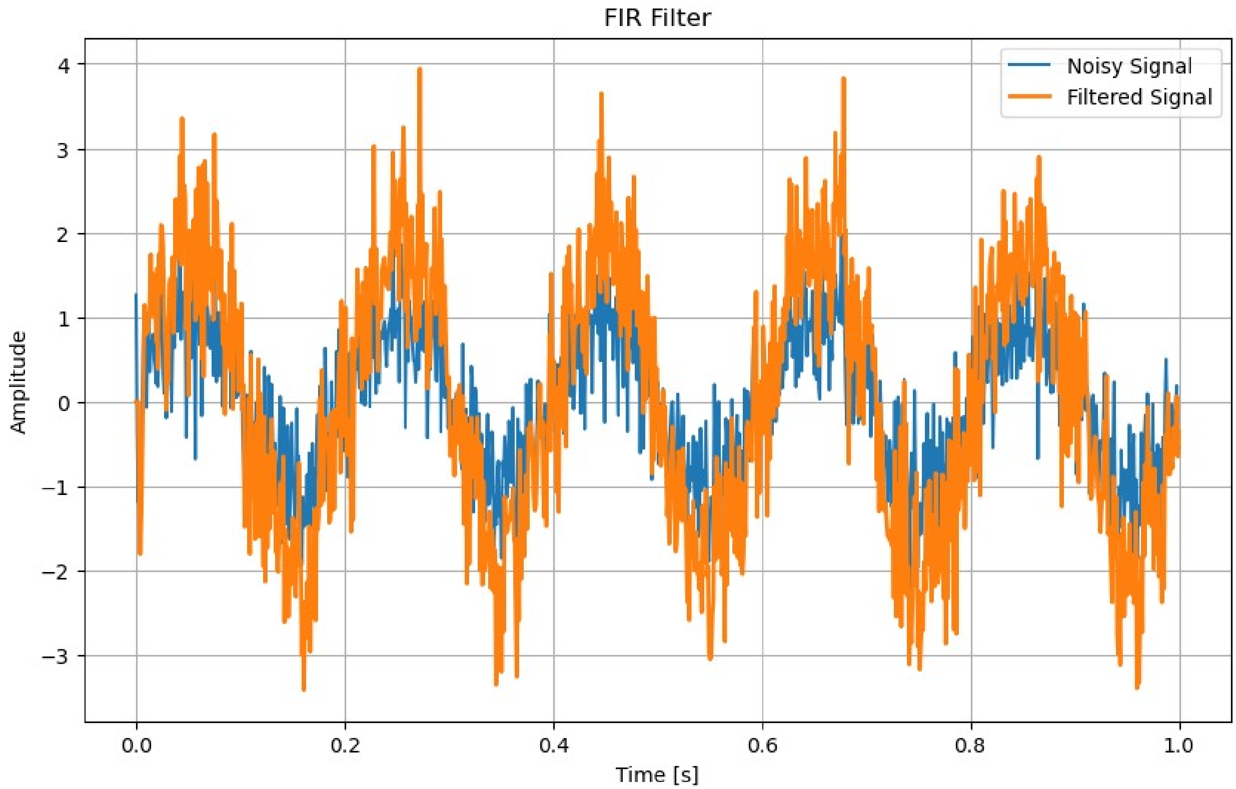


5. Digital Filter Design and Analysis: Implementing FIR and IIR filters in Python. 6. Adaptive Filtering: Applying adaptive filtering algorithms to noise reduction.

FIR implementation

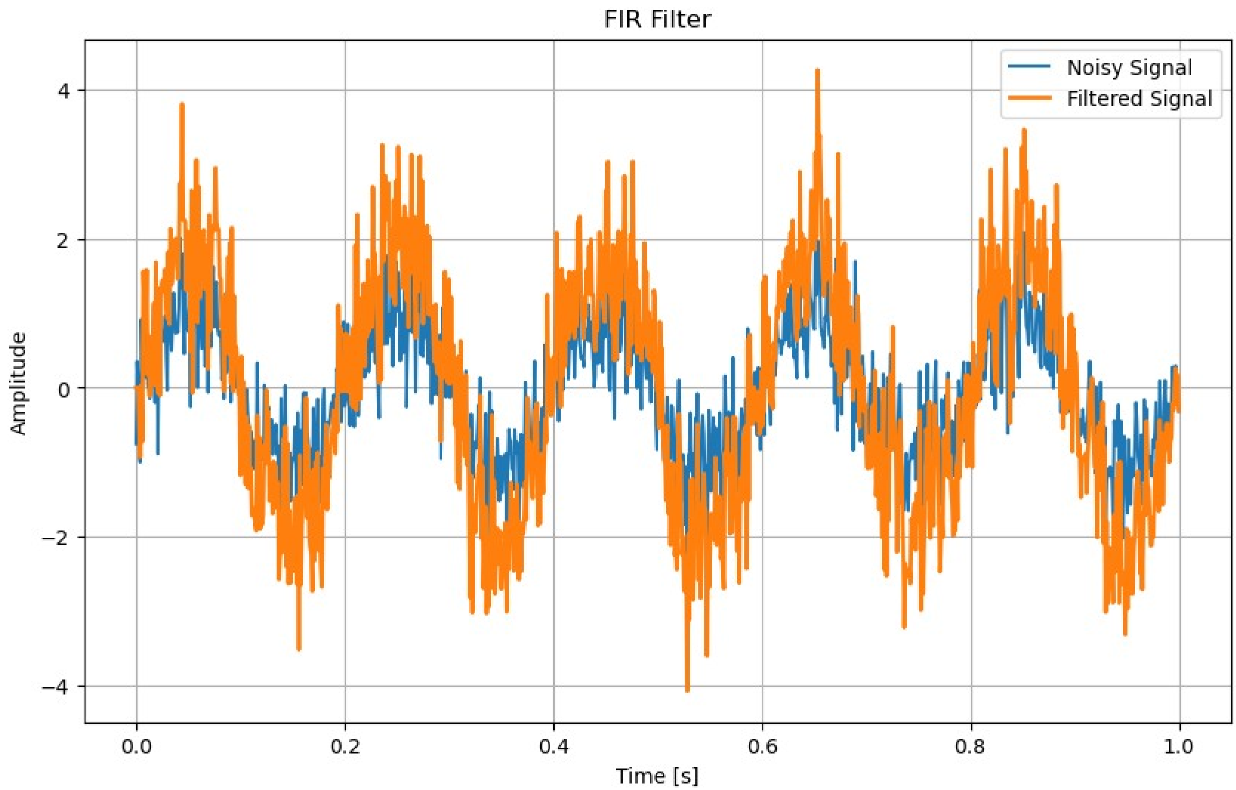
```
import numpy as np
import matplotlib.pyplot as plt

def fir_filter(x, b):
    """
    FIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Filter coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)
    y = np.zeros(len(x))
    for n in range(M, len(x)):
        y[n] = np.dot(b, x[n-M+1:n+1][::-1])
    return y

# Example usage and plotting
fs = 1000  # Sampling frequency
t = np.linspace(0, 1, fs)
x = np.sin(2 * np.pi * 5 * t) + 0.5 * np.random.randn(len(t))  #
Signal with noise
b = [1, 0, 1]  # FIR coefficients from Variant 8

y = fir_filter(x, b)

plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("FIR Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
```

```
plt.grid()
plt.show()
```



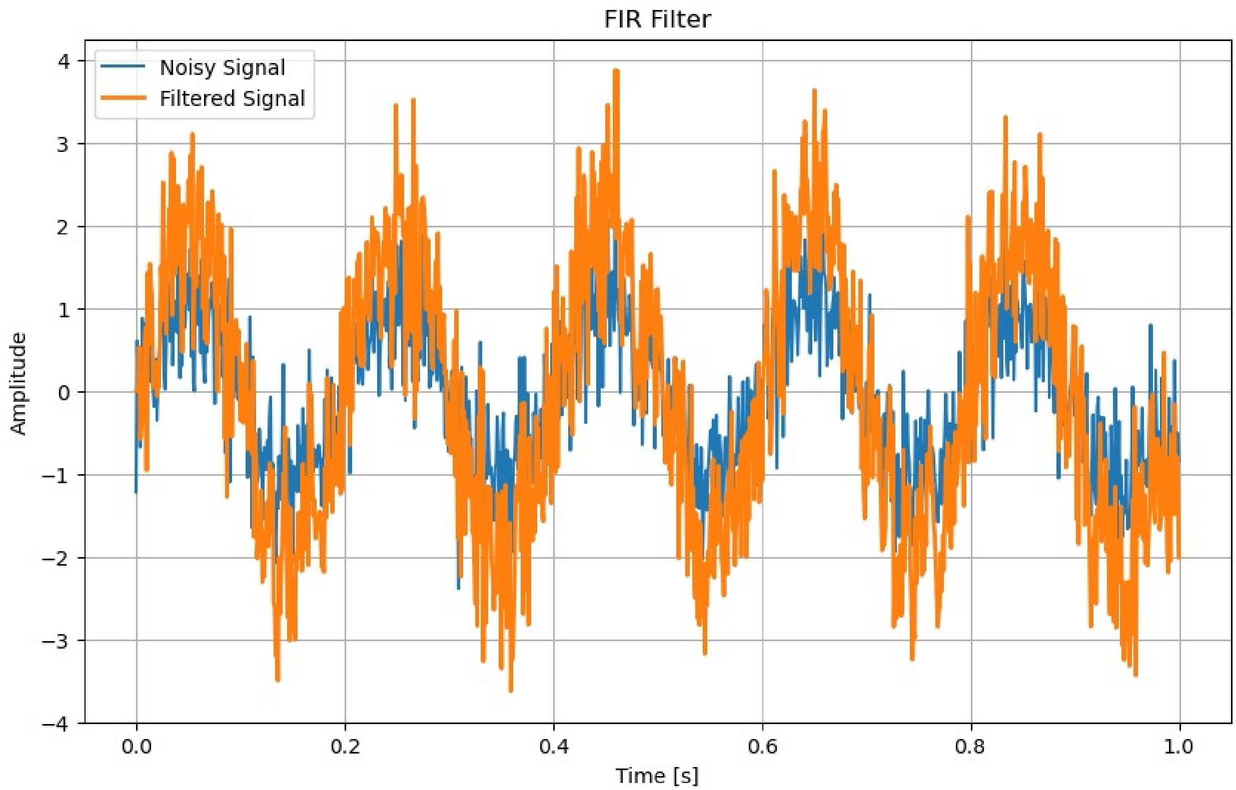

```python
import numpy as np
import matplotlib.pyplot as plt

def fir_filter(x, b):
    """
    FIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Filter coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)
    y = np.zeros(len(x))
    for n in range(M, len(x)):
        y[n] = np.dot(b, x[n-M+1:n+1][::-1])
```

```
    return y

# Example usage and plotting
fs = 1000  # Sampling frequency
t = np.linspace(0, 1, fs)
x = np.sin(2 * np.pi * 5 * t) + 0.5 * np.random.randn(len(t))  #
Signal with noise
b = [1, 0, 1]  # FIR coefficients for Variant 8

y = fir_filter(x, b)

plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("FIR Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```



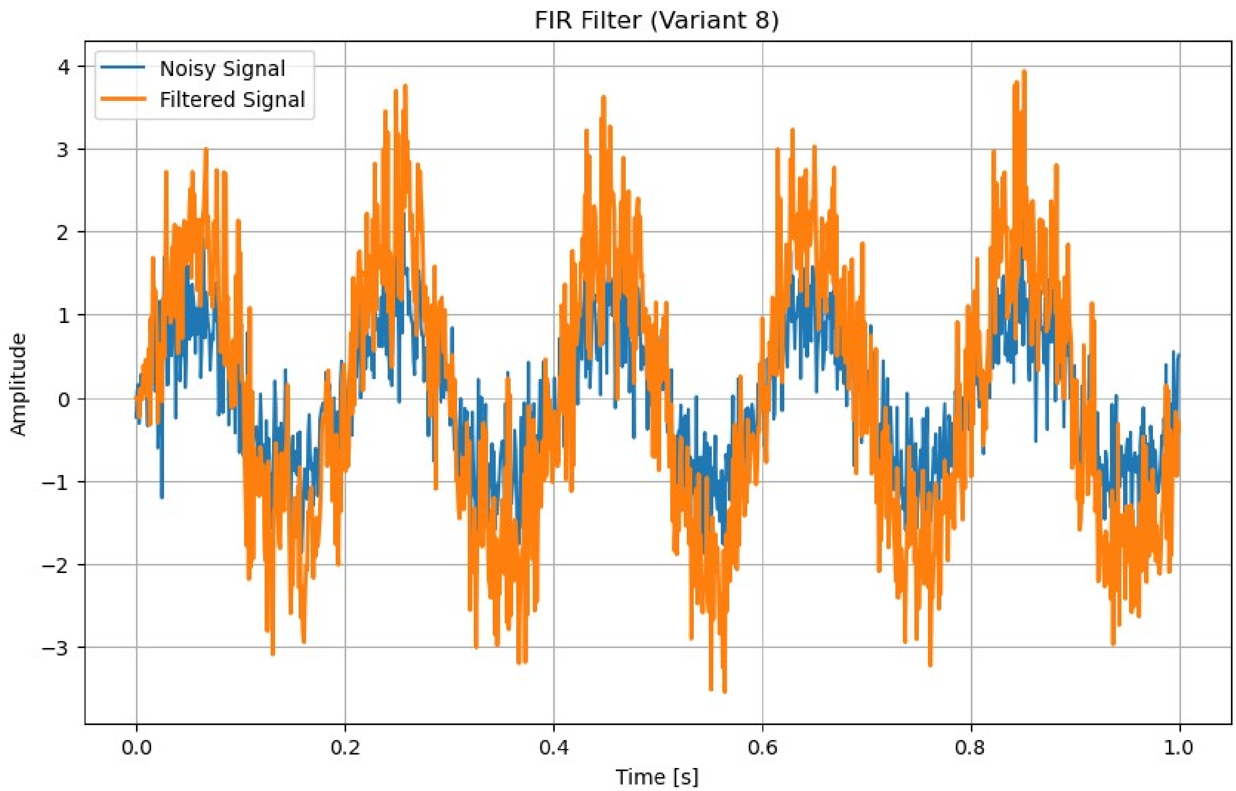

```
import numpy as np
import matplotlib.pyplot as plt

def fir_filter(x, b):
```

```python
    """
    FIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Filter coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)
    y = np.zeros(len(x))
    for n in range(M, len(x)):
        y[n] = np.dot(b, x[n-M+1:n+1][::-1])
    return y

# Example usage and plotting
fs = 1000  # Sampling frequency
t = np.linspace(0, 1, fs)
x = np.sin(2 * np.pi * 5 * t) + 0.5 * np.random.randn(len(t))  #
Signal with noise
b = [1, 0, 1]  # FIR coefficients for Variant 8

y = fir_filter(x, b)

plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("FIR Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```




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

def fir_filter(x, b):
    """
    FIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Filter coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)
    y = np.zeros(len(x))
    for n in range(M, len(x)):
        y[n] = np.dot(b, x[n-M+1:n+1][::-1])
    return y

# Example usage and plotting
fs = 1000  # Sampling frequency
```

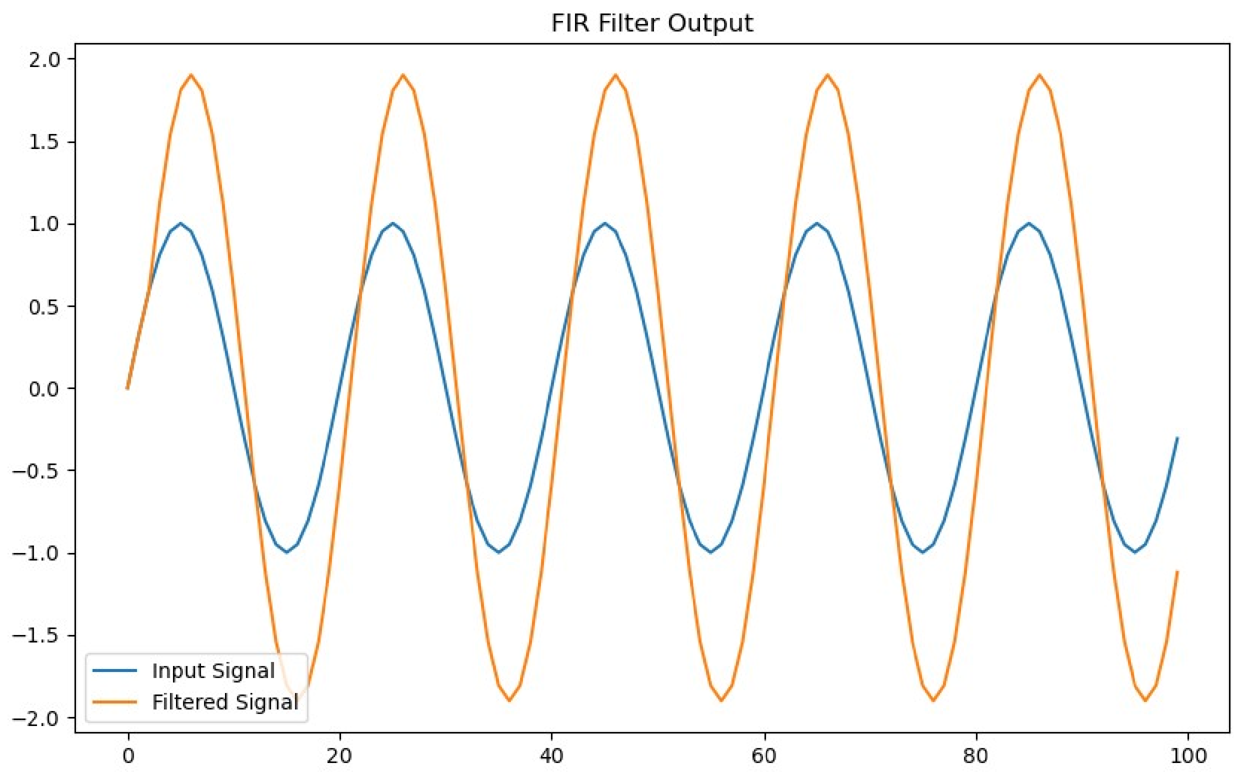
```
t = np.linspace(0, 1, fs)
x = np.sin(2 * np.pi * 5 * t) + 0.5 * np.random.randn(len(t))  #
Signal with noise
b = [1, 0, 1]  # FIR coefficients for Variant 8

y = fir_filter(x, b)

plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("FIR Filter (Variant 8)")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```



FIR using scipy.signal library

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

# Define FIR Filter
def fir_filter(b, x):
```

```
    M = len(b)  # Number of coefficients
    y = np.convolve(x, b, mode='full')[:len(x)]  # Apply filter
    return y

# Example FIR Coefficients and Input Signal
b = [1, 0, 1]  # FIR coefficients
x = np.sin(2 * np.pi * 0.05 * np.arange(100))  # Example input signal

# Filter the Signal
y = fir_filter(b, x)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(x, label="Input Signal")
plt.plot(y, label="Filtered Signal")
plt.legend()
plt.title("FIR Filter Output")
plt.show()
```

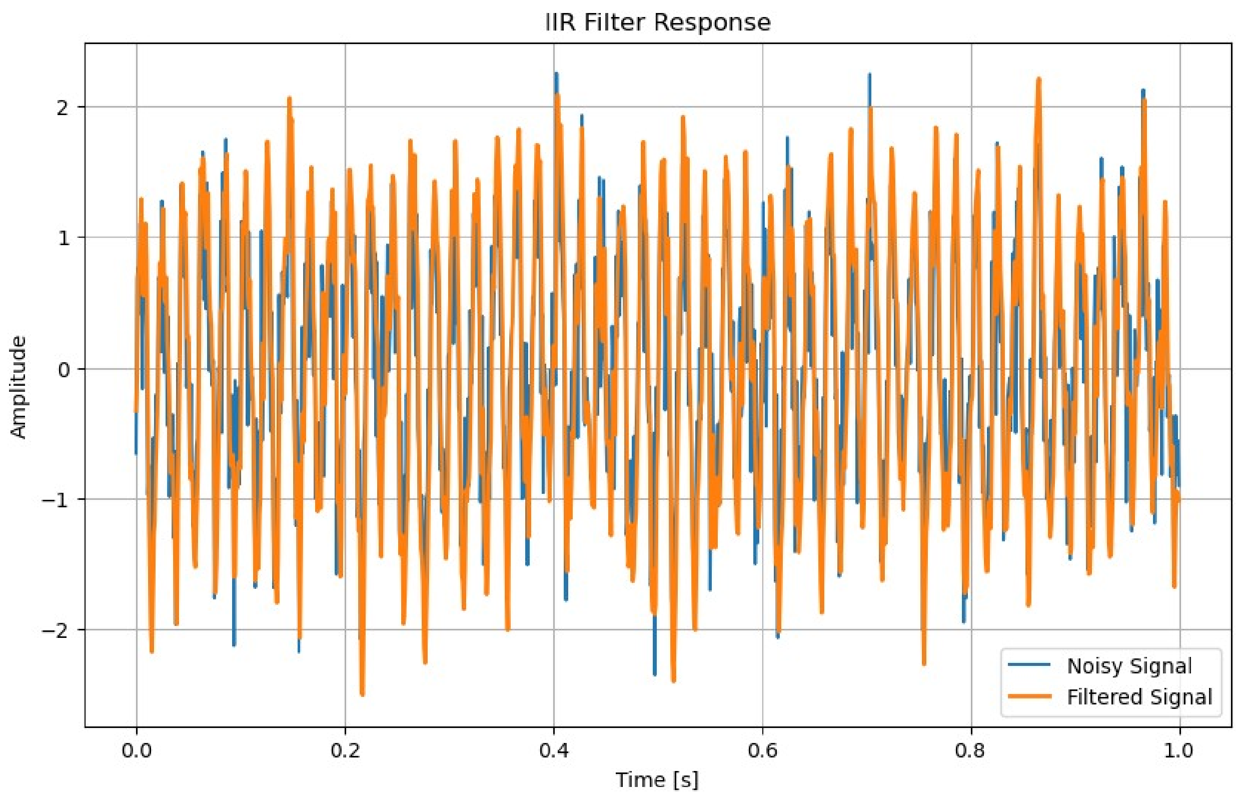


IIR Filter implementation

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

def iir_filter(x, b, a):
    """
```

```
    IIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Numerator coefficients.
    a : ndarray
        Denominator coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)  # Length of numerator coefficients (b)
    N = len(a)  # Length of denominator coefficients (a)
    y = np.zeros(len(x))  # Initialize output signal array

    # Apply filter to each sample in the input signal
    for n in range(len(x)):
        # Numerator part (feedforward)
        # Ensure we use the correct slice length for the convolution
        x_slice = x[max(0, n-M+1):n+1]  # Input signal slice
        y[n] = np.dot(b[:len(x_slice)], x_slice[::-1])  # Apply
reverse convolution for numerator

        # Denominator part (feedback), skip the first sample
        if n >= 1:
            # Ensure we use the correct slice length for the feedback
part
            y_slice = y[max(0, n-N+1):n]  # Output signal slice
            y[n] -= np.dot(a[1:min(N, len(y_slice)+1)], y_slice[::-1])
# Apply reverse convolution for feedback

    return y

# Example usage and plotting
# Create a noisy signal (for example, a sine wave with noise)
fs = 1000  # Sampling frequency
t = np.linspace(0, 1, fs)  # Time vector
x = np.sin(2 * np.pi * 50 * t) + 0.5 * np.random.randn(len(t))  #
Noisy signal

# IIR filter coefficients
a = [1, -0.3]  # Denominator coefficients (a_0 = 1 by convention)
b = [0.5, 0.5]  # Numerator coefficients

# Apply the filter to the noisy signal
y = iir_filter(x, b, a)
```

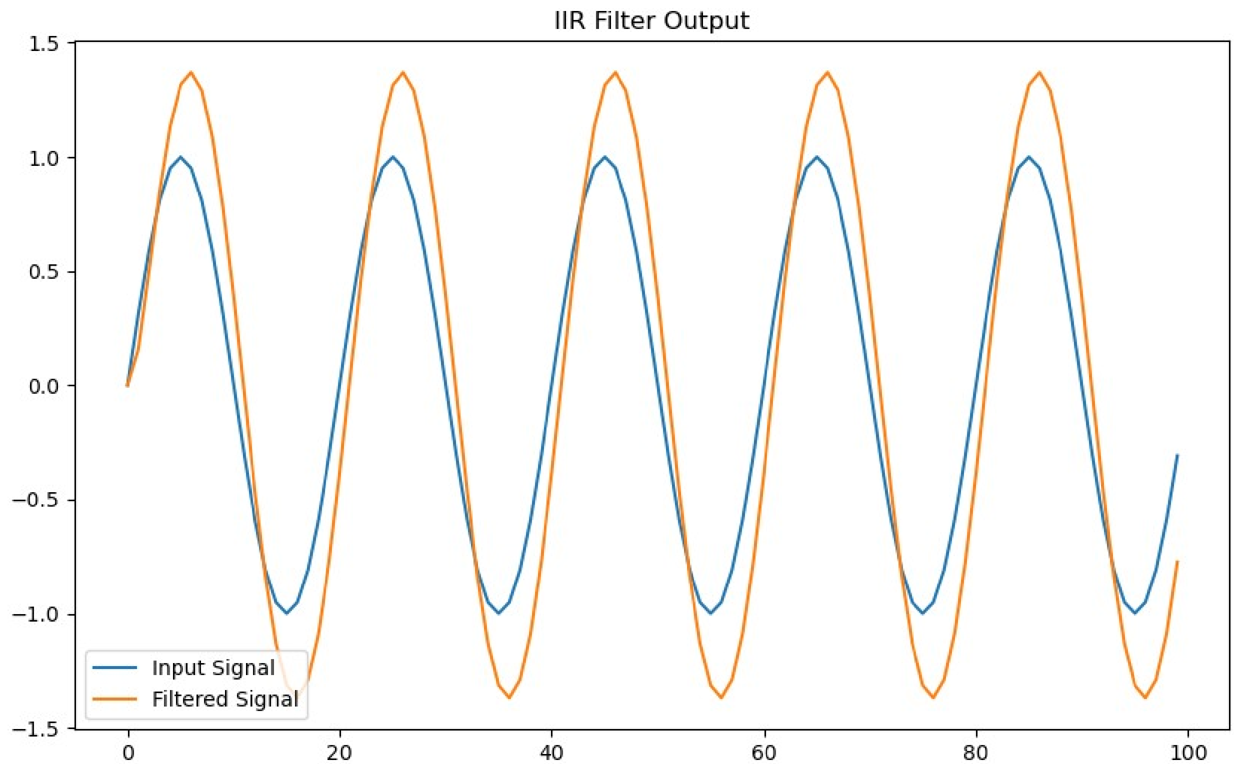
```python
# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("IIR Filter Response")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```



IIR implementation using scipy.signal

```python
from scipy.signal import lfilter, freqz

# Define IIR Filter Coefficients
a = [1, -0.3]
b = [0.5, 0.5]
# Example Input Signal
x = np.sin(2 * np.pi * 0.05 * np.arange(100))

# Filter the Signal
y = lfilter(b, a, x)

# Plot the Output
```

```
plt.figure(figsize=(10, 6))
plt.plot(x, label="Input Signal")
plt.plot(y, label="Filtered Signal")
plt.legend()
plt.title("IIR Filter Output")
plt.show()
```



Adaptive LMS Filter Implementation

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

# Example Input Signal
t = np.arange(0, 1, 0.001)  # Time vector of length 1000
x = np.sin(2 * np.pi * 0.05 * t)  # Input signal (noisy)
d = np.sin(2 * np.pi * 5 * t)  # Desired signal

# LMS Filter
def lms_filter(x, d, mu, num_taps):
    n = len(x)
    w = np.zeros(num_taps)
    y = np.zeros(n)
    e = np.zeros(n)

    for i in range(num_taps, n):
        x_segment = x[i-num_taps:i][::-1]
```

```
        y[i] = np.dot(w, x_segment)
        e[i] = d[i] - y[i]
        w += mu * e[i] * x_segment

    return y, e, w

# LMS filter parameters
mu = 0.01  # Step size
num_taps = 5

# Apply LMS filter
y, e, w = lms_filter(x, d, mu, num_taps)

# Plot the desired signal and filtered output
plt.figure(figsize=(10, 6))
plt.plot(t, d, label="Desired Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("LMS Adaptive Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```