REPORT

Zajęcia: Analog and digital electronic circuits
Teacher: prof. dr hab. Vasyl Martsenyuk

Lab
Date: 23.11.2024
Topic: Design and Implementation of Digital Filters: FIR, IIR, and Adaptive LMS for Noise Reduction
Exploration of Adaptive Signal Processing Techniques Using LMS Algorithm and Python
Variant DSP 4

Tomasz Pietrzyk
Informatyka II stopień,
niestacjonarne,
1 semestr,
Gr.A

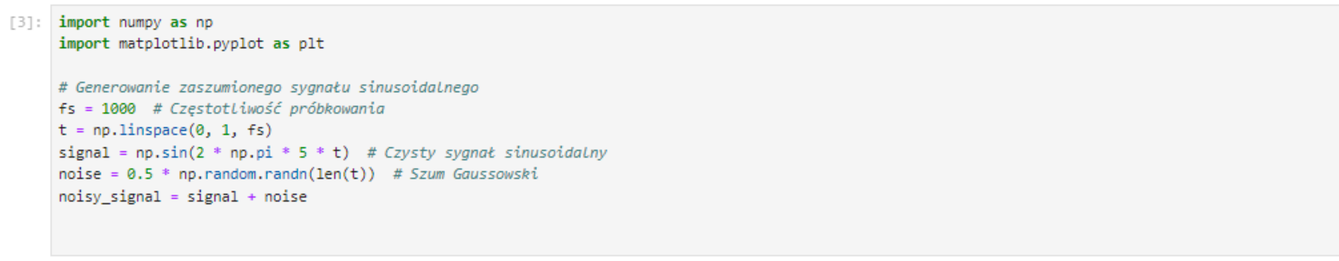
1. Problem statement:

The task is to design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.
- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal. IIR Filter Coefficients
 Implement an adaptive LMS filter in Python.

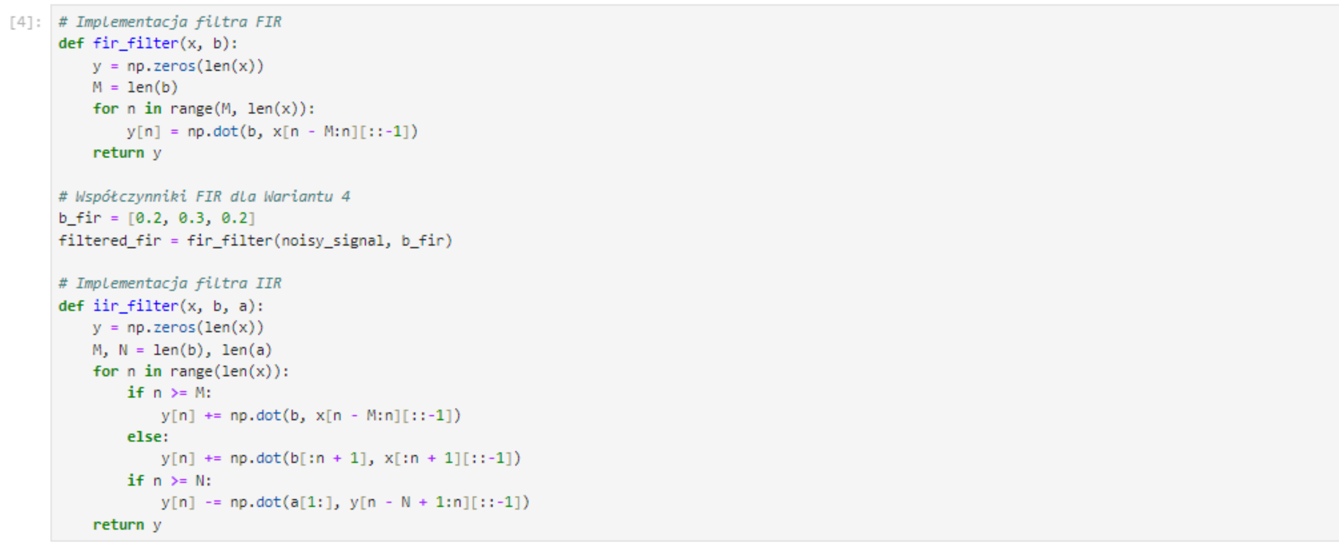
2. Input data:

Variant 4: - FIR Filter Coefficients: b = {0.2, 0.3, 0.2} - IIR Filter Coefficients: b = {1, 0.5, 0.3}, a = {1, −0.6} - Implement an adaptive LMS filter in Python with a step size μ = 0.1 and filter length M = 4 to reduce noise in the same noisy sinusoidal signal.

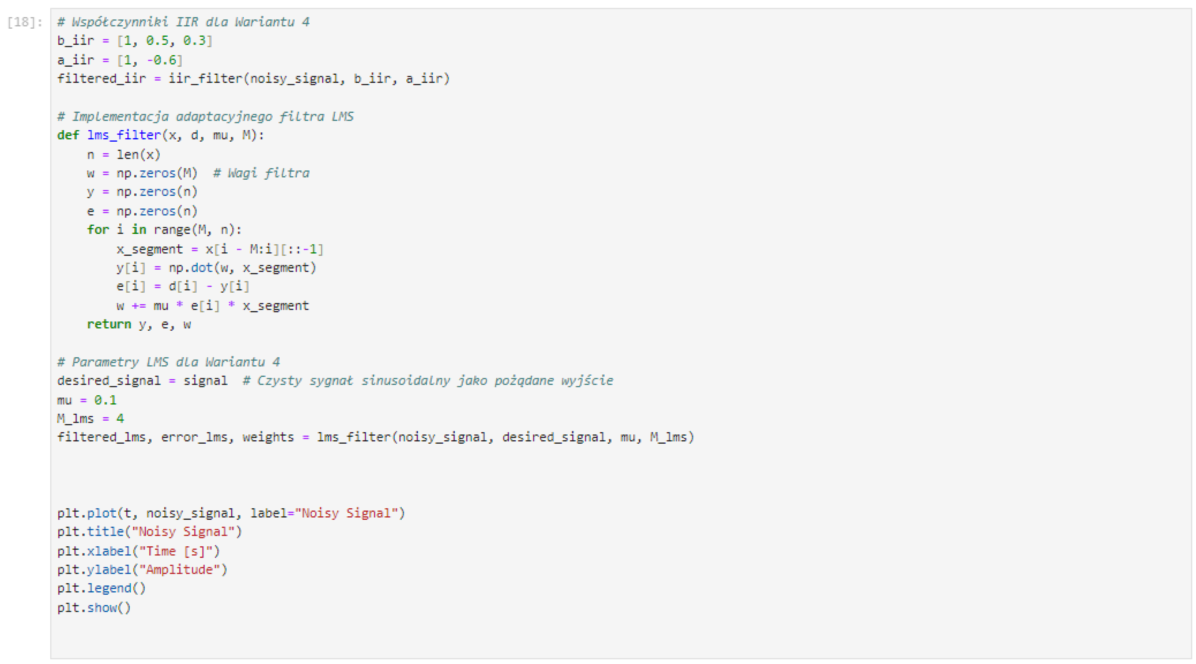
3.Commands used (or GUI):



[3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Generowanie zaszumionego sygnału sinusoidalnego
fs = 1000  # Częstotliwość próbkowania
t = np.linspace(0, 1, fs)
signal = np.sin(2 * np.pi * 5 * t)  # Czysty sygnał sinusoidalny
noise = 0.5 * np.random.randn(len(t))  # Szum Gaussowski
noisy_signal = signal + noise
```

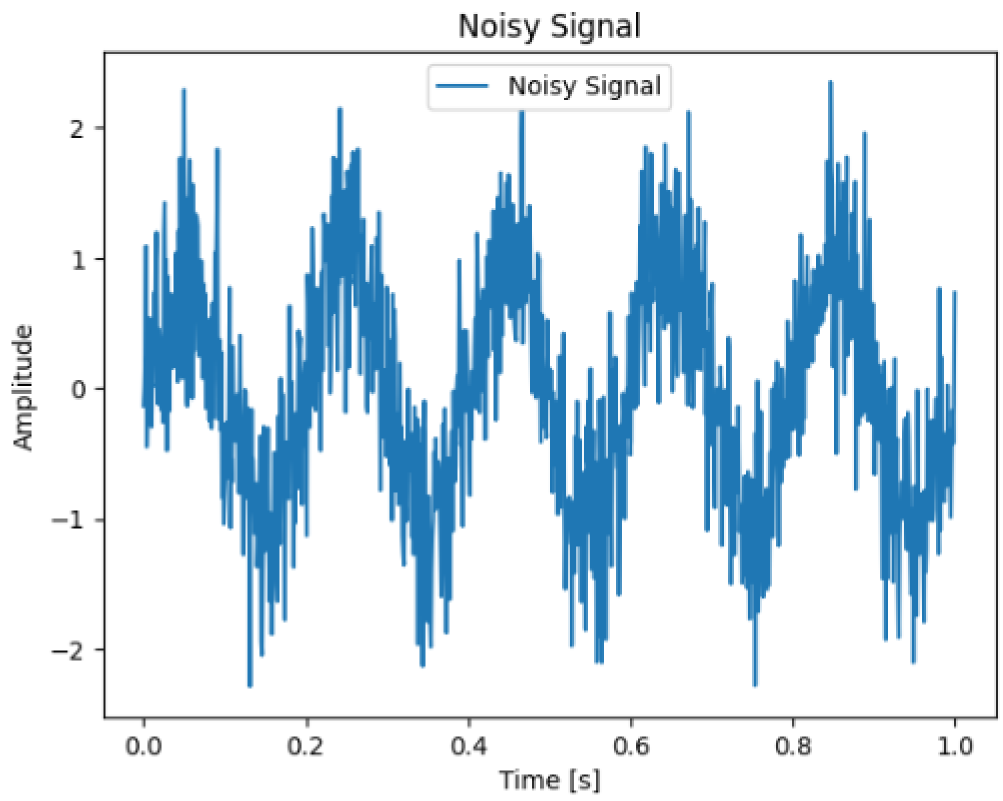


[4]:
```
# Implementacja filtra FIR
def fir_filter(x, b):
    y = np.zeros(len(x))
    M = len(b)
    for n in range(M, len(x)):
        y[n] = np.dot(b, x[n - M:n][::-1])
    return y

# Współczynniki FIR dla Wariantu 4
b_fir = [0.2, 0.3, 0.2]
filtered_fir = fir_filter(noisy_signal, b_fir)

# Implementacja filtra IIR
def iir_filter(x, b, a):
    y = np.zeros(len(x))
    M, N = len(b), len(a)
    for n in range(len(x)):
        if n >= M:
            y[n] += np.dot(b, x[n - M:n][::-1])
        else:
            y[n] += np.dot(b[:n + 1], x[:n + 1][::-1])
        if n >= N:
            y[n] -= np.dot(a[1:], y[n - N + 1:n][::-1])
    return y
```



```
[18]: # Współczynniki IIR dla Wariantu 4
      b_iir = [1, 0.5, 0.3]
      a_iir = [1, -0.6]
      filtered_iir = iir_filter(noisy_signal, b_iir, a_iir)

      # Implementacja adaptacyjnego filtra LMS
      def lms_filter(x, d, mu, M):
          n = len(x)
          w = np.zeros(M)  # Wagi filtra
          y = np.zeros(n)
          e = np.zeros(n)
          for i in range(M, n):
              x_segment = x[i - M: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

      # Parametry LMS dla Wariantu 4
      desired_signal = signal  # Czysty sygnał sinusoidalny jako pożądane wyjście
      mu = 0.1
      M_lms = 4
      filtered_lms, error_lms, weights = lms_filter(noisy_signal, desired_signal, mu, M_lms)



      plt.plot(t, noisy_signal, label="Noisy Signal")
      plt.title("Noisy Signal")
      plt.xlabel("Time [s]")
      plt.ylabel("Amplitude")
      plt.legend()
      plt.show()
```

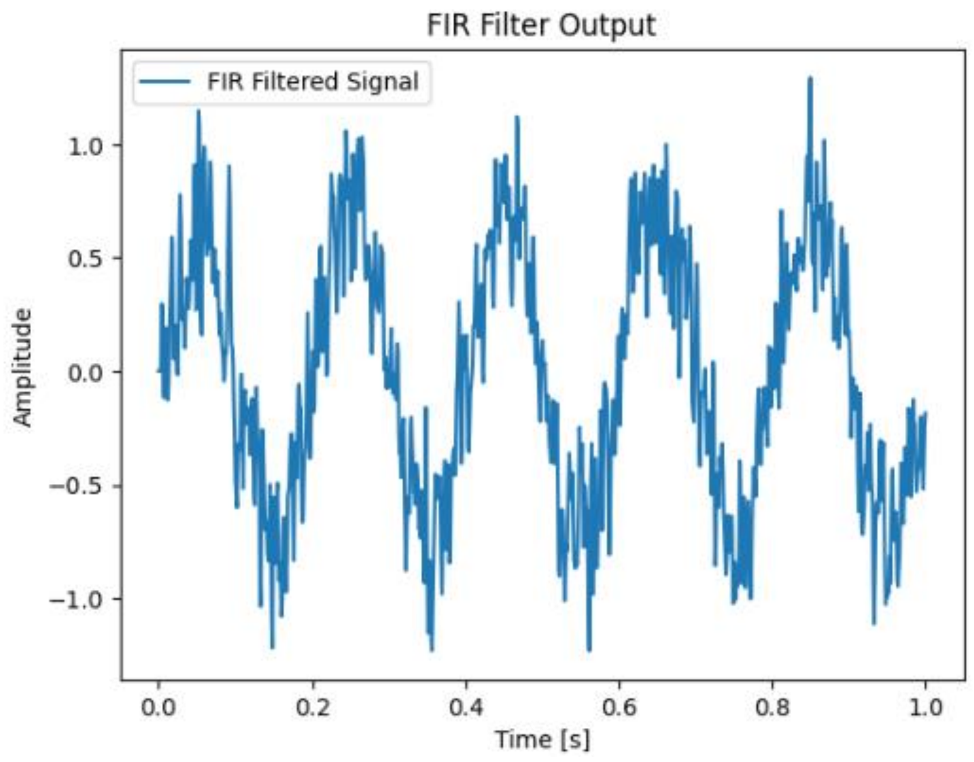
2.



3.

```
plt.plot(t, filtered_fir, label="FIR Filtered Signal")
plt.title("FIR Filter Output")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.legend()
plt.show()
```

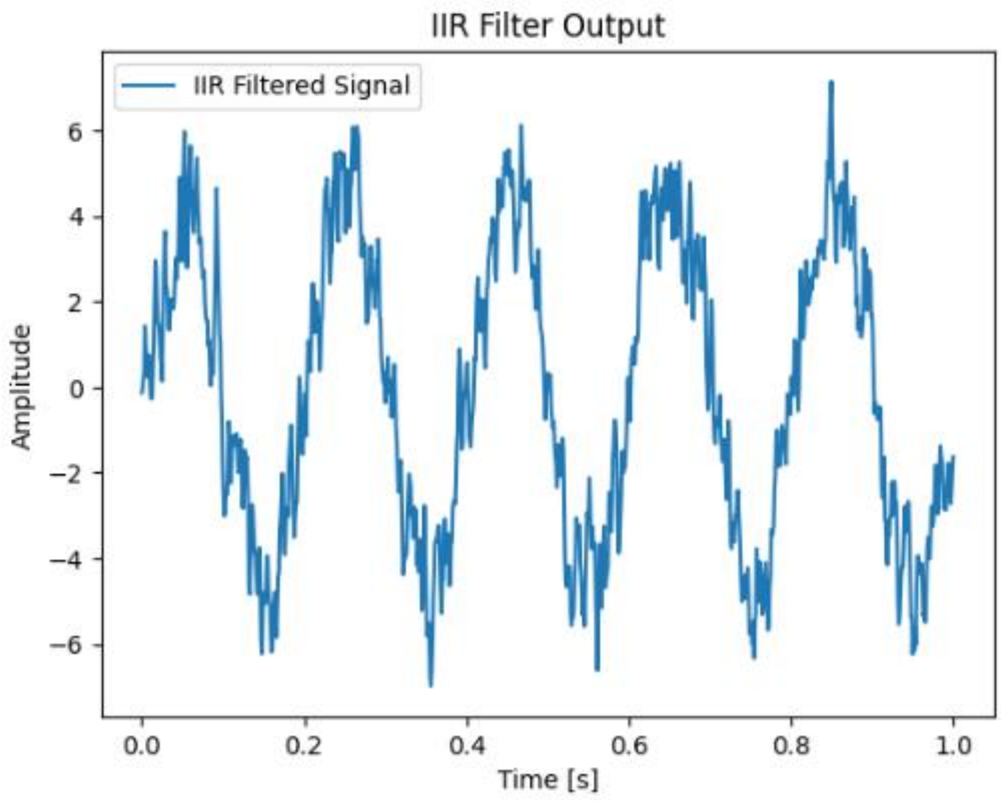
4.

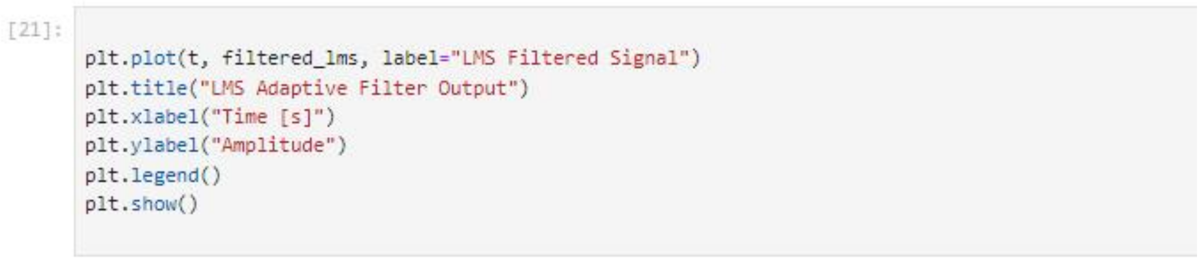


5.
6.
7.
8.

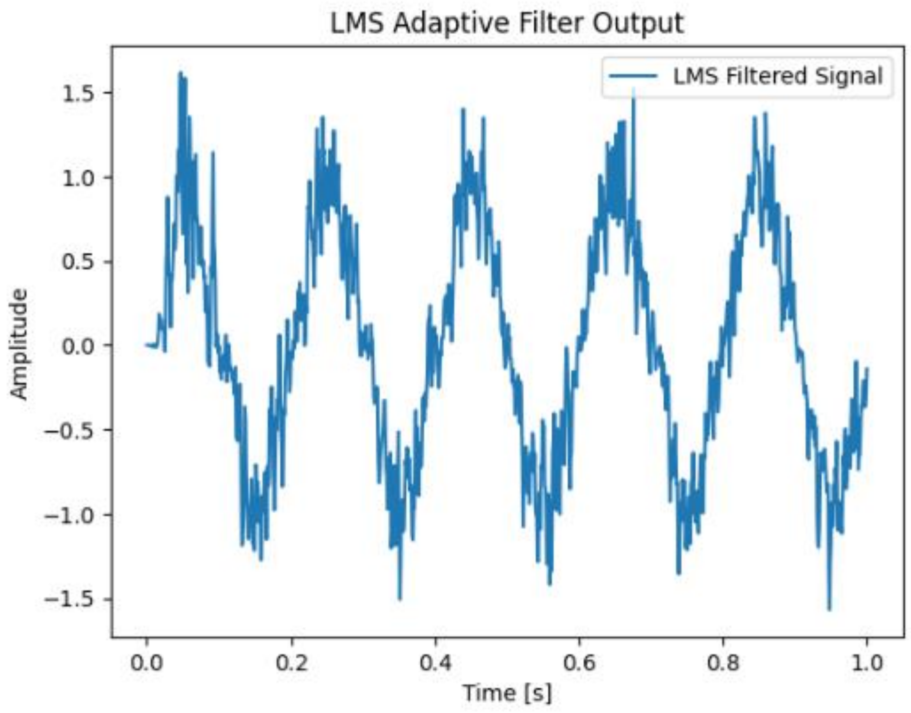
9.

```
plt.plot(t, filtered_iir, label="IIR Filtered Signal")
plt.title("IIR Filter Output")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.legend()
```



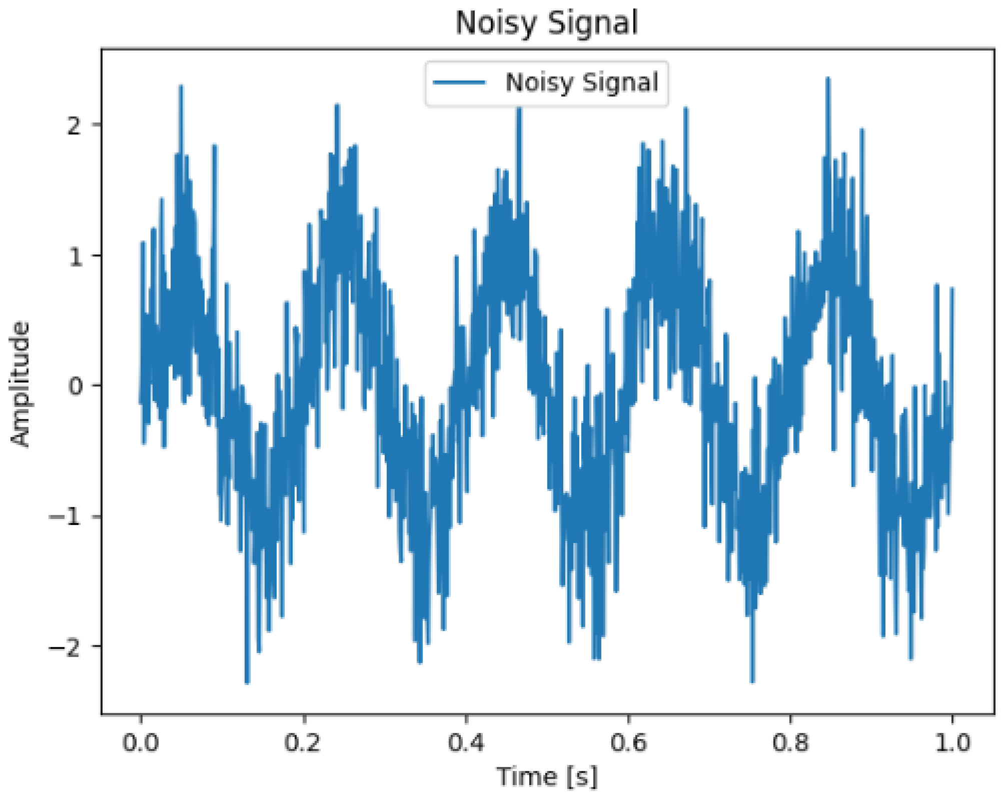


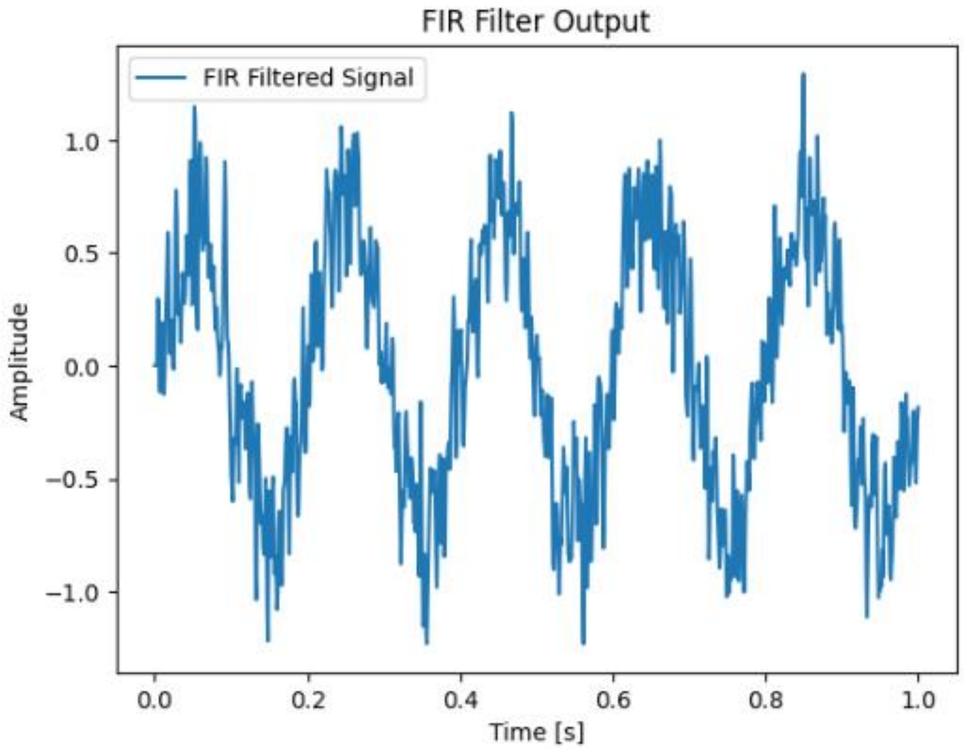
```
[21]:
plt.plot(t, filtered_lms, label="LMS Filtered Signal")
plt.title("LMS Adaptive Filter Output")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.legend()
plt.show()
```

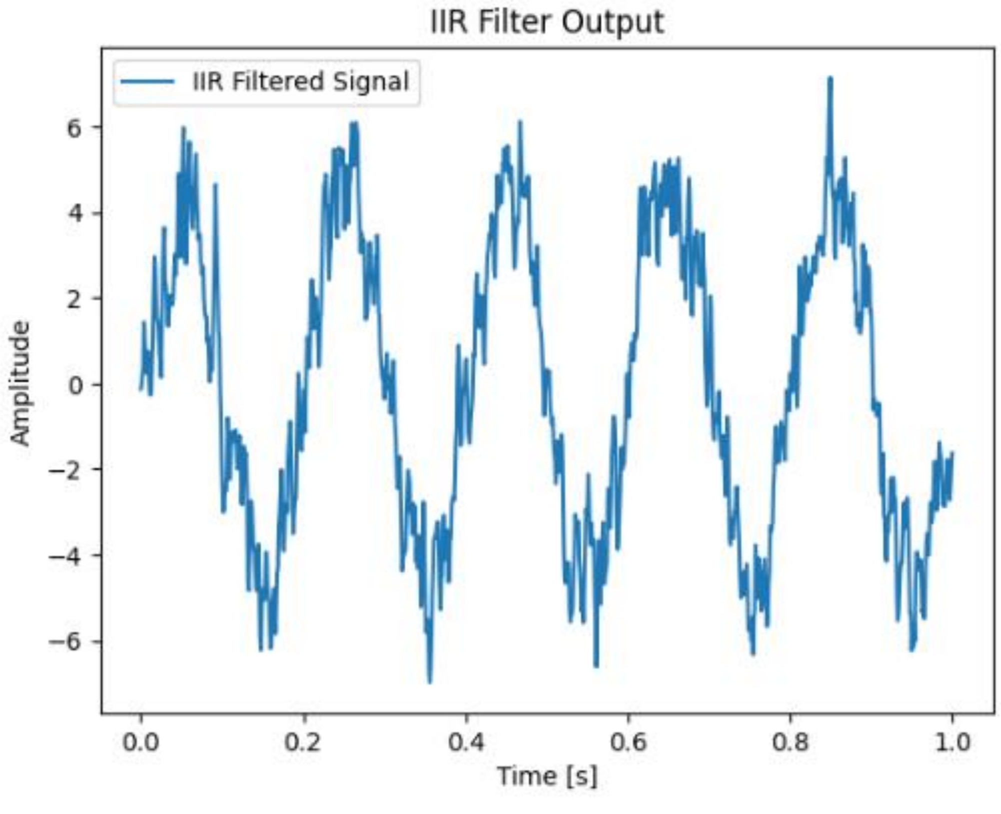


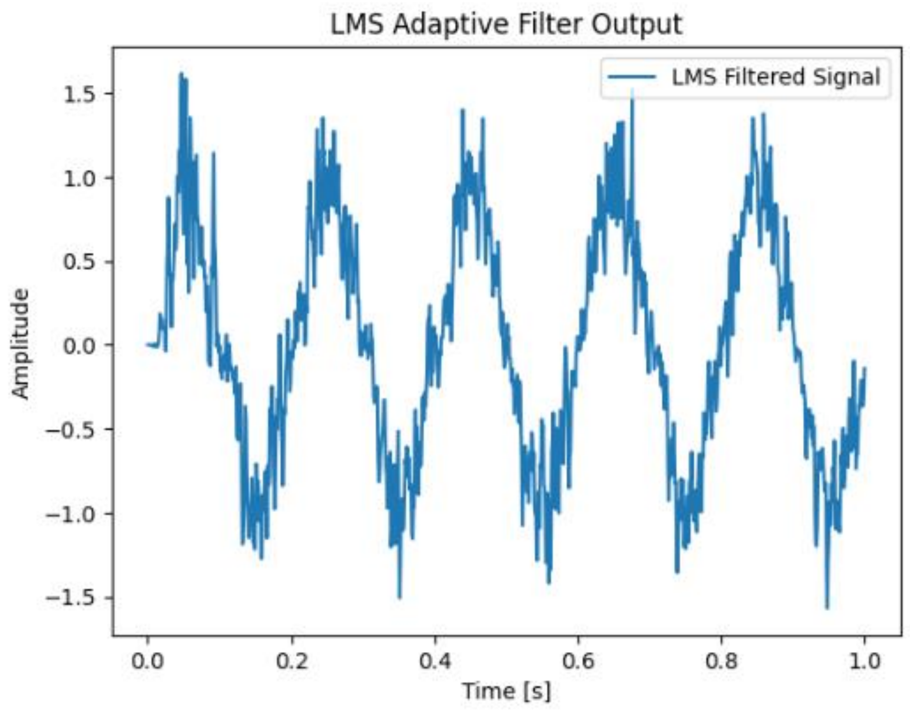
Link to remote repozytorium (e.g. GitHub)
https://github.com/TomekPietrzyk/DSP_2024_NS

4. Outcomes:


Noisy Signal
Noisy Signal
Amplitude
Time [s]
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


FIR Filter Output
FIR Filtered Signal
Amplitude
Time [s]
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


IIR Filter Output
IIR Filtered Signal
Amplitude
Time [s]
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0



5. Conclusions: The analysis and implementation of digital filters, including FIR, IIR, and adaptive LMS filters, highlighted their respective strengths and practical considerations in signal processing tasks. FIR filters proved to be inherently stable and phase-preserving due to their non-recursive structure, making them suitable for applications requiring precise phase alignment. However, achieving effective noise reduction required a higher filter order, resulting in increased computational demands. In contrast, IIR filters demonstrated superior efficiency by achieving comparable noise reduction with fewer coefficients. Their feedback mechanism, while advantageous for computational performance, necessitated careful stability analysis to ensure poles remained within the unit circle. The LMS adaptive filter showcased its versatility in dynamically reducing noise in signals with time-varying characteristics. Its performance depended heavily on selecting appropriate parameters, such as filter length and step size, which influenced convergence speed and noise suppression capability. Python proved to be an effective platform for implementing and visualizing these filters, providing a practical and flexible environment for exploring digital signal processing concepts. Overall, the study underscored the importance of selecting the right filter type and parameters to meet specific signal processing requirements efficiently.