REPORT

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

Lab 5-6
Date 23.11.2024
Topic: "5. Digital Filter Design and Analysis: Implementing FIR and IIR filters in Python.
6. Adaptive Filtering: Applying adaptive filtering algorithms to noise reduction."
Variant 10

Anna Więzik
Informatyka II stopień,
niestacjonarne,
1 semestr,
Gr.1b

1. Problem statement:

Each task requires you to implement all three types of filters: FIR, IIR, and Adaptive LMS, using different parameters and observe the performance for noise reduction.

2. Input data:

Variant 10: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.4, 0.4, 0.4\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

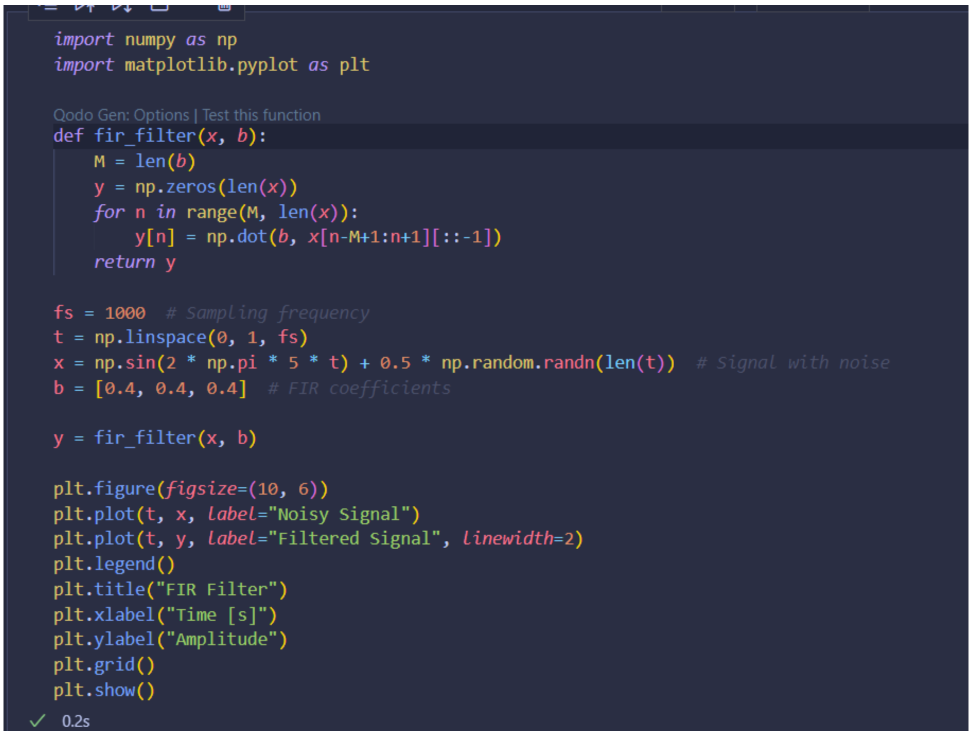
$$\text{IIR Filter Coefficients: } b = \{1, -0.6\}, \quad a = \{1, 0.5\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.1$ and filter length $M = 5$ to reduce noise in the same noisy sinusoidal signal.

link to remote repozytorium: https://github.com/AnaShiro/DSP_2024

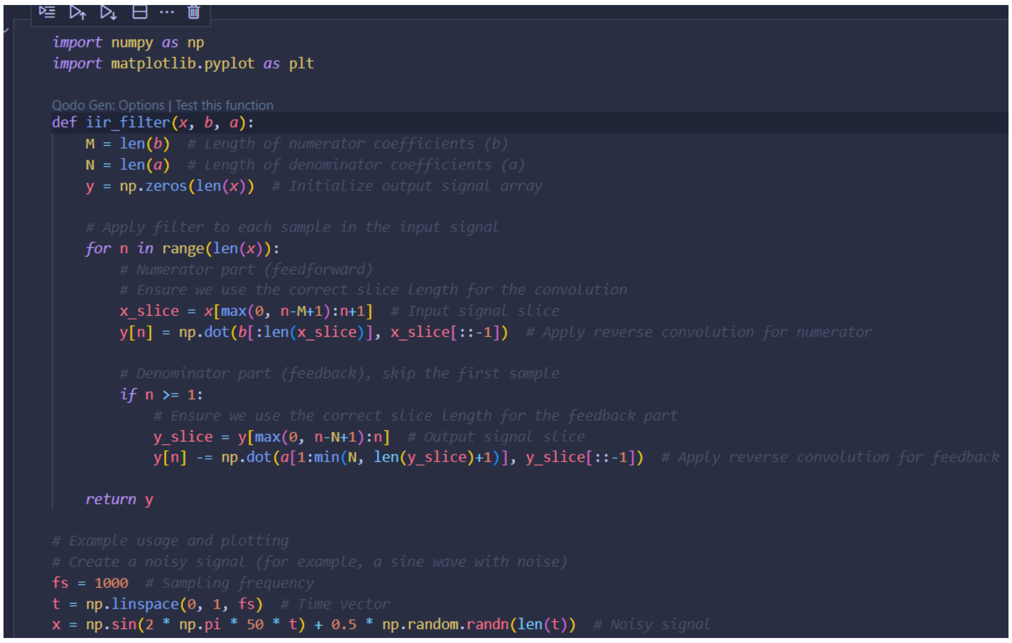
3. Commands used (or GUI):

- FIR



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

Qodo Gen: Options | Test this function
def fir_filter(x, b):
    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

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 = [0.4, 0.4, 0.4]  # FIR coefficients

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()
```
✓ 0.2s

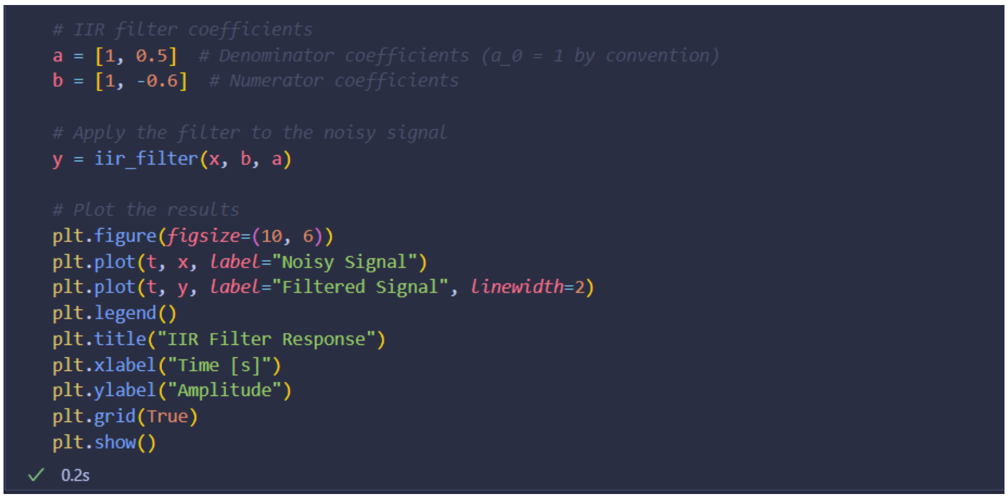
- IIR



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

Qodo Gen: Options | Test this function

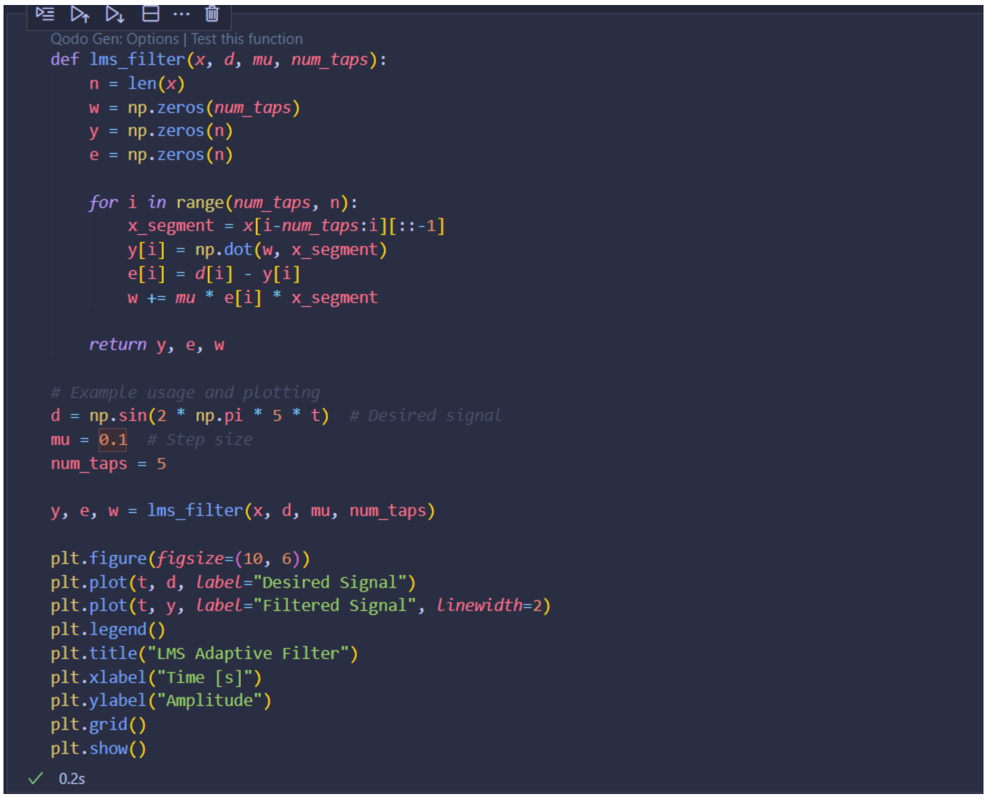
```python
def iir_filter(x, b, a):
    M = len(b)  # Length of numerator coefficients (b)
    N = len(a)  # Length of denominator coefficients (a)
    y = np.zeros(len(x))  # Initialize output signal array

    # 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
```



```python
# IIR filter coefficients
a = [1, 0.5]  # Denominator coefficients (a_0 = 1 by convention)
b = [1, -0.6]  # Numerator coefficients

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

# 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()
```

✓ 0.2s

- LMS

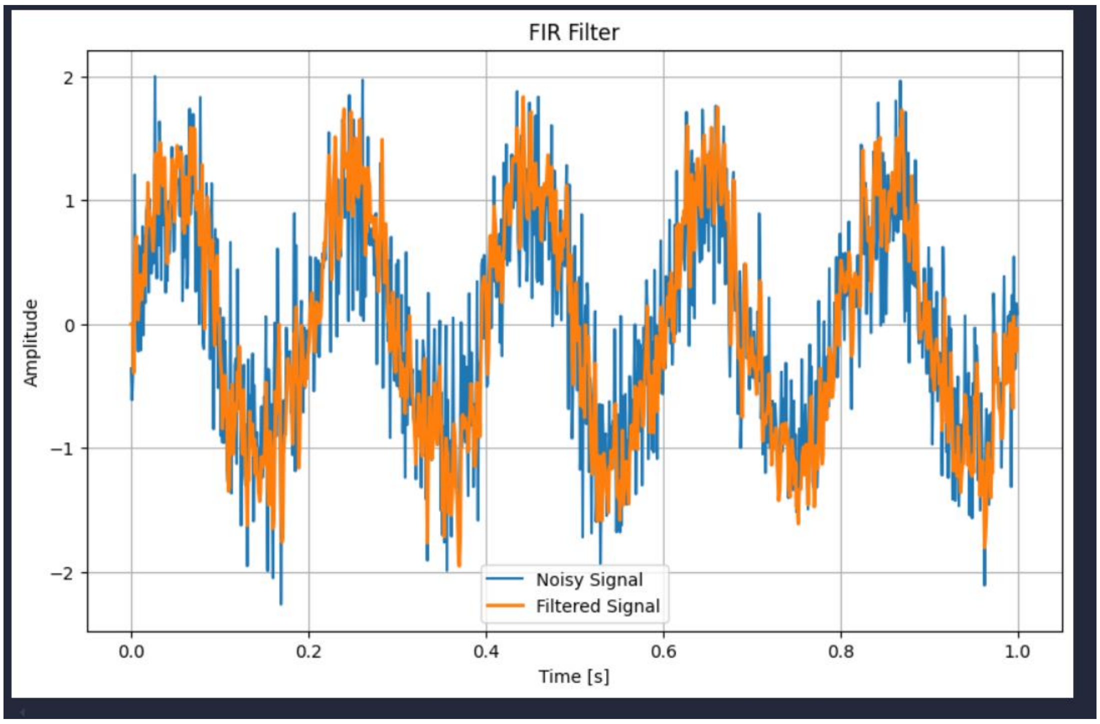


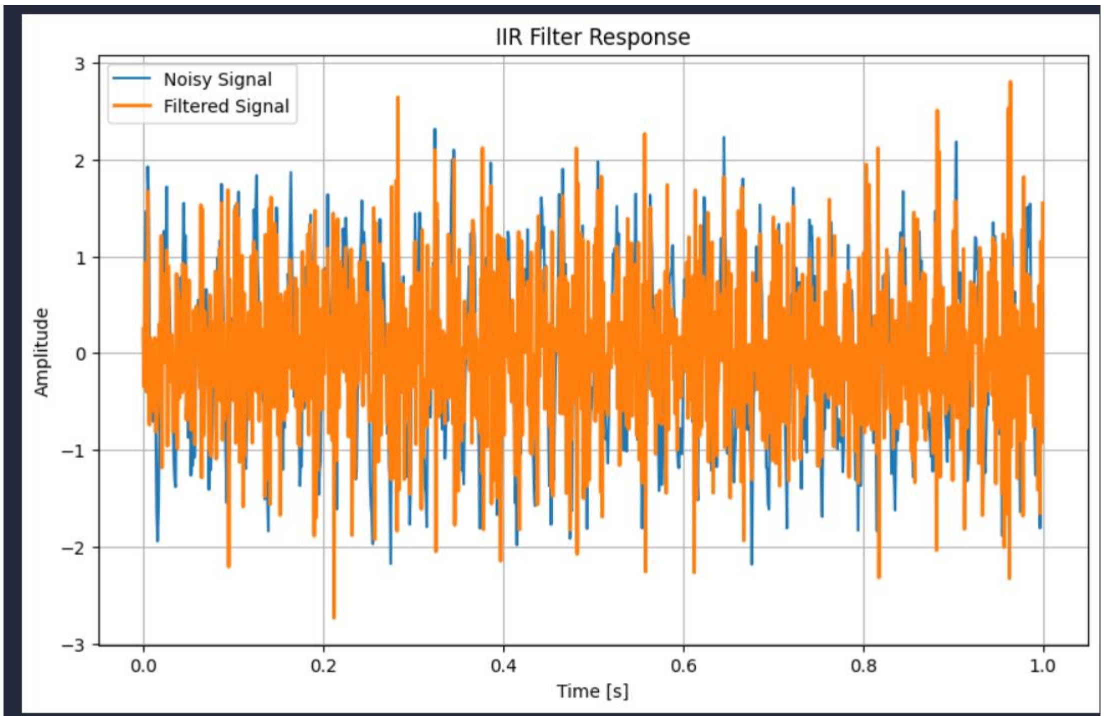
Qodo Gen: Options | Test this function

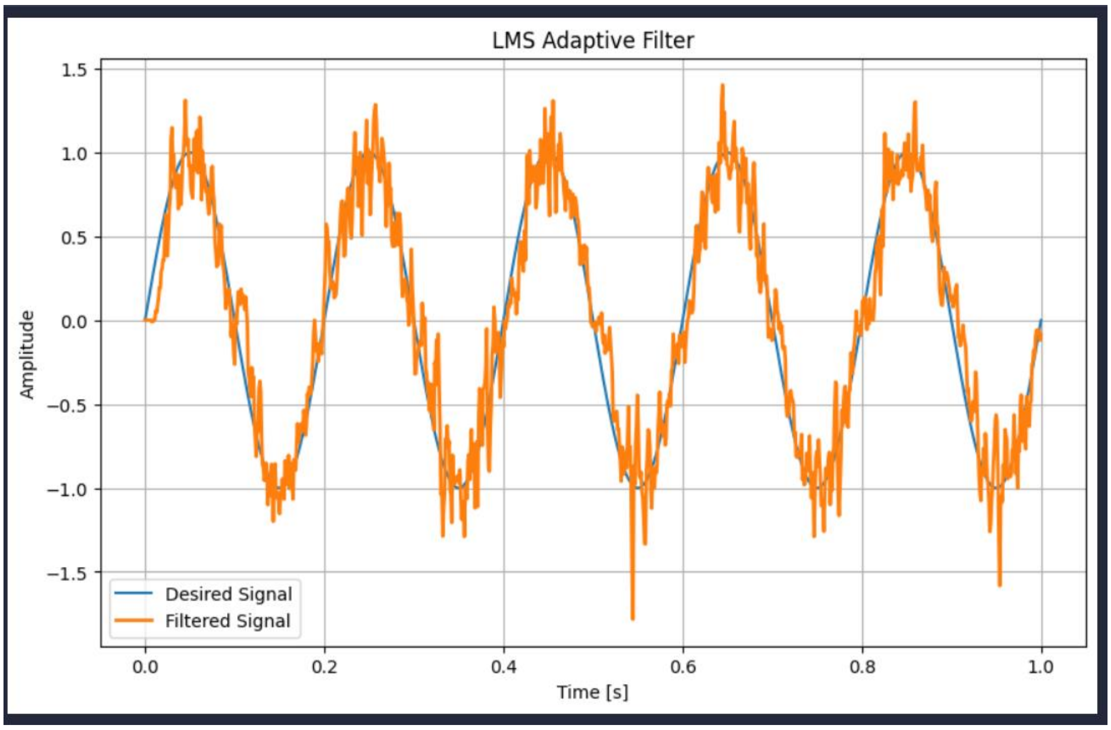
```
def lms_filter(x, d, mu, num_taps):
    n = len(x)
    w = np.zeros(num_taps)
    y = np.zeros(n)
    e = np.zeros(n)

    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

# Example usage and plotting
d = np.sin(2 * np.pi * 5 * t)  # Desired signal
mu = 0.1  # Step size
num_taps = 5

y, e, w = lms_filter(x, d, mu, num_taps)

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()
```

✓ 0.2s

4. Outcomes:




IIR Filter Response
Noisy Signal
Filtered Signal
Amplitude
Time [s]


LMS Adaptive Filter
Desired Signal
Filtered Signal
Amplitude
Time [s]

5. Conclusions:

This manual offers a comprehensive overview of the mathematical principles underlying FIR (Finite Impulse Response), IIR (Infinite Impulse Response), and LMS (Least Mean Squares) adaptive filters. It not only provides the theoretical background but also includes practical Python implementations for each filter type. Furthermore, the manual incorporates various visualization techniques to effectively demonstrate the performance and efficiency of these filters in real-world scenarios. This combination of theory, implementation, and visual aids ensures a clear understanding of the filters' functionality and their applications.