REPORT

Subject: Digital Signal Processing
Lecturer: prof. dr hab. Vasyl Martsenyuk

Laboratory #3 Date: 23.11.2024 Topic: Projektowanie i analiza filtrów cyfrowych: implementacja filtrów FIR i IIR w Matlabu/Pythonie. Filtracja adaptacyjna: zastosowanie algorytmów filtracji adaptacyjnej w redukcji szumów. Second variant (2)	Krzysztof Świerczek IT Science II degree, 1 semester, gr.A

1. Task:

Variant 2:
- Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.
FIR Filter Coefficients: b = {1, 1, 2}
- 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: b = {0.6, 0.2}, a = {1, −1}
- Implement an adaptive LMS filter in Python with a step size μ = 0.05 and filter length
M = 4 to reduce noise in the same noisy sinusoidal signal.

2. Source code Github:

https://github.com/Krzycho165/STUDIA/tree/main/DSP

TD. 1. Definition of filters

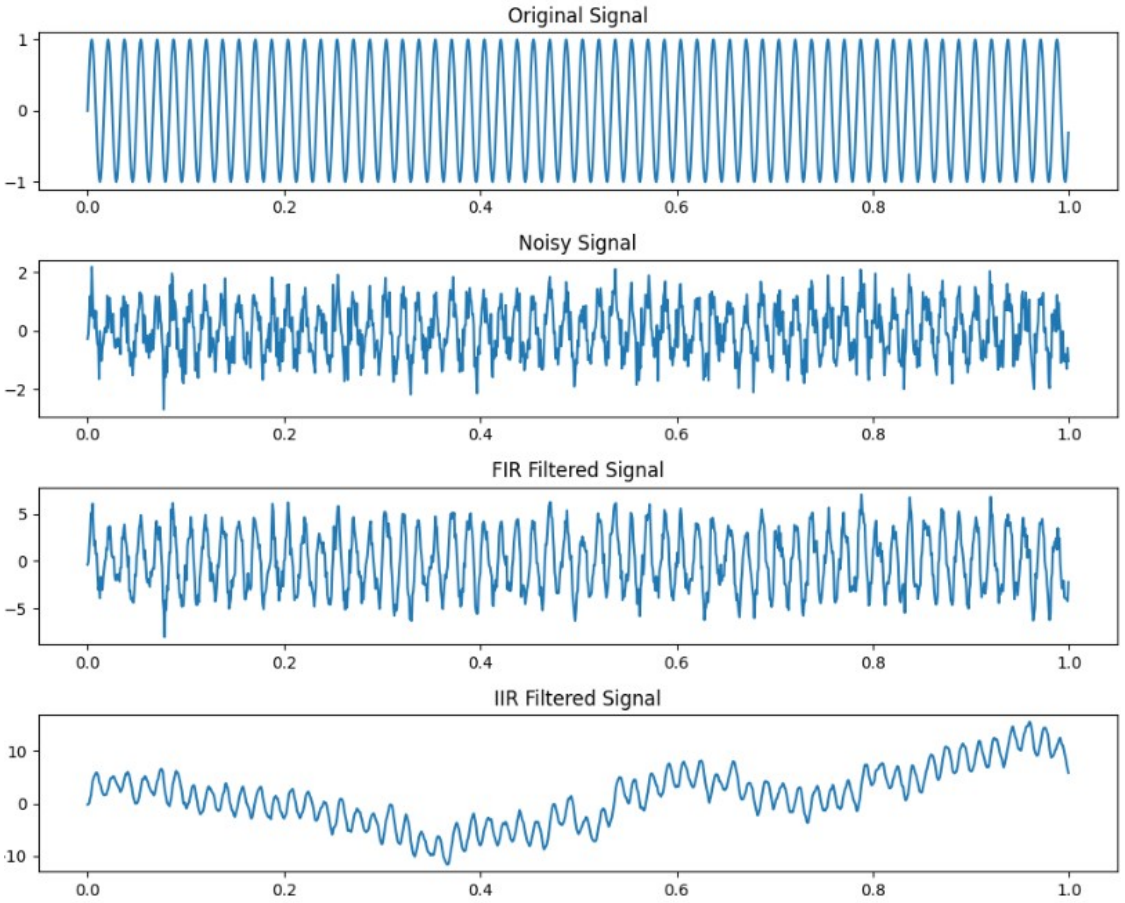
Functions shown at technical drawing number one are implementing basic digital signal processing filters in Python. The FIR filter (fir_filter) applies a finite impulse response filter to the input signal using convolution with given coefficients. The IIR filter (iir_filter) processes the input signal with recursive coefficients, where each output depends on previous outputs. The LMS adaptive filter (lms_filter) adjusts its filter weights based on the approximation between the desired signal and the current output, optimizing for noise reduction in real-time.

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

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

fs = 1200
t = np.arange(0, 1, 1/fs)
f_signal = 60
signal = np.sin(2 * np.pi * f_signal * t)
noise = np.random.normal(0, 0.5, len(t))
noisy_signal = signal + noise

b_fir = np.array([1, 1, 2])
b_iir = np.array([0.6, 0.2])
a_iir = np.array([1, -1])

fir_output = fir_filter(noisy_signal, b_fir)

iir_output = iir_filter(noisy_signal, b_iir, a_iir)

mu = 0.05
M = 4
lms_output, lms_error = lms_filter(noisy_signal, signal, mu, M)
```

TD. 2. Applying data from task and plotting.

The code displayed above assigns the data provided with task and generating a noisy sinusoidal signal, applying the previously defined filters, and visualizing the results.

The code creates a clean sinusoidal signal and then adds a noise, mimicking a real-world noisy signal. Then, the FIR filter, IIR filter, and LMS adaptive filter are applied to see the result of how does this signal works. Each filter is designed to reduce noise, so filtered signals supposed to be comparable to the original and noisy signals in the plots.



T.D. 3. The result of nosing signal and filtering.

3. Conclusions

The FIR and IIR filters significantly reduce the noise, but very noisy signal drastically changes comparing to the original.

The FIR filter, though simple, performs very well in smoothing the signal. Meanwhile the IIR filter (with its recursive nature) achieves a more efficient way to get similar results with fewer coefficients.