**REPORT**

Zajęcia: Analog and digital electronic circuits

Teacher: prof. dr hab. Vasyl Martsenyuk

**Lab 5 and 6**

17.02.2025

**Topic:**

5. Digital Filter Design and Analysis: Implementing FIR and IIR filters in Python.

6. Adaptive Filtering: Applying adaptive filtering algorithms to noise reduction.ls

**Variant 15**

Tobiasz Wojnar  
 Informatyka II stopień,   
niestacjonarne,   
1 semestr,   
Gr. B

# 1. Problem statement:

The goal of this task is to design FIR, IIR and LMS filters in Python.

1. FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.
2. IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.
3. Adaptive LMS filter in Python to reduce noise in the same noisy sinusoidal signal.

# 2. Input data:

1. FIR Filter Coefficients: b = {0.2, 0.3, 0.5}
2. IIR Filter Coefficients: b = {1, 0.5}, a = {1, −0.7}
3. LMS filter with a step size µ = 0.05 and filter length M = 6

# 3. Commands used (or GUI):

## a) source code

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

fir\_b = [ 0.2, 0.3, 0.5]

iir\_b = [ 1.0, 0.5 ]

iir\_a = [ 1.0, -0.7 ]

lms\_mu = 0.05

lms\_m = 6

fs = 1000 *# Sampling frequency*

t = np.linspace(0, 1, fs)

f = 36 *# base signal*

signal\_base = np.sin(2 \* np.pi \* f \* t)

signal\_harmonic\_1 = np.sin(2 \* np.pi \* 2 \* f \* t) /2

signal\_harmonic\_2 = np.sin(2 \* np.pi \* 3 \* f \* t) /3

signal = signal\_base + signal\_harmonic\_1 + signal\_harmonic\_2

noise = np.random.randn(len(t))/4

noisy\_signal = signal + noise

plt.figure(figsize=(10, 6))

plt.plot(t, noisy\_signal, ms=1, label= "Noisy Signal" )

plt.plot(t, signal, ms=3, label= "Pure Signal" )

plt.legend()

plt.title( "Signal" )

plt.xlabel( "Time [s]" )

plt.ylabel( "Amplitude" )

plt.grid()

plt.show()

*# $y[n] = \sum\_{k=0}^M b\_k x\[n − k\]$*

**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) *# Number of coefficients*

y = np.convolve(x, b, mode='full')[:len(x)] *# Apply filter*

**return** y

*# $y[n] = \sum\_{k=0}^M b\_k x[n − k] - \sum\_{k=1}^N a\_k y[n − k]$*

**def** iir\_filter(x, b, a):

"""

FIR filter implementation.

Parameters:

x : ndarray

Input signal.

b : ndarray

Filter coefficients.

a : ndarray

Denominator coefficients.

Returns:

y : ndarray

Filtered output signal.

"""

**return** lfilter(b, a, x)

**def** lms\_filter(x, d, mu, num\_taps):

"""

LMS adaptive filter implementation.

Parameters:

x : ndarray

Input signal (noisy).

d : ndarray

Desired signal.

mu : float

Step size.

num\_taps : int

Number of filter taps.

Returns:

y : ndarray

Filtered output signal.

e : ndarray

Error signal.

w : ndarray

Final filter weights.

"""

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

fir\_filtered = fir\_filter(noisy\_signal,fir\_b)

iir\_filtered = iir\_filter(noisy\_signal, iir\_b, iir\_a)

lms\_filtered, e, w = lms\_filter(noisy\_signal, signal, lms\_mu, lms\_m)

plt.figure(figsize=(15, 10))

plt.subplot(5, 1, 1)

plt.plot(t, noisy\_signal, label='Noisy Signal')

plt.title("Noisy Signal")

plt.grid()

plt.legend()

plt.subplot(5, 1, 2)

plt.plot(t, fir\_filtered, label='FIR Filtered', color='g')

plt.title("FIR Filter Output")

plt.grid()

plt.legend()

plt.subplot(5, 1, 3)

plt.plot(t, iir\_filtered, label='IIR Filtered', color='r')

plt.title("IIR Filter Output")

plt.grid()

plt.legend()

plt.subplot(5, 1, 4)

plt.plot(t, lms\_filtered, label='LMS Filtered', color='m')

plt.title("LMS Filter Output")

plt.grid()

plt.legend()

plt.subplot(5, 1, 5)

plt.plot(t, signal, label='Signal', color='b')

plt.title("Original signal")

plt.grid()

plt.legend()

plt.tight\_layout()

plt.show()

*# All results at once*

plt.figure(figsize=(15, 5))

plt.plot(signal / max(signal), ms=3, label='Original', color='b')

plt.plot(noisy\_signal / max(noisy\_signal), ms=3, label='with noise')

plt.plot(fir\_filtered / max(fir\_filtered), ms=3, label='FIR', color='g')

plt.plot(iir\_filtered / max(iir\_filtered), ms=3, label='IIR', color='r')

plt.plot(lms\_filtered / max(lms\_filtered), ms=3, label='LMS', color='m')

plt.xlabel('$time$')

plt.ylabel('$Amplitude$')

plt.xlim(360,730)

plt.legend()

plt.grid()

*# All results compared to original*

plt.figure(figsize=(15, 10))

plt.subplot(4, 1, 1)

plt.plot(signal / max(signal), ms=3, label='Original', color='b')

plt.plot(noisy\_signal / max(noisy\_signal), ms=3, label='with noise')

plt.plot(fir\_filtered / max(fir\_filtered), ms=3, label='FIR', color='g')

plt.plot(iir\_filtered / max(iir\_filtered), ms=3, label='IIR', color='r')

plt.plot(lms\_filtered / max(lms\_filtered), ms=3, label='LMS', color='m')

plt.xlabel('$time$')

plt.ylabel('$Amplitude$')

plt.xlim(0,370)

plt.legend()

plt.grid()

plt.subplot(4, 1, 2)

plt.plot(signal / max(signal), ms=3, label='Original', color='b')

plt.plot(fir\_filtered / max(fir\_filtered), ms=3, label='FIR', color='g')

plt.xlabel('$time$')

plt.ylabel('$Amplitude$')

plt.xlim(0,370)

plt.legend()

plt.grid()

plt.subplot(4, 1, 3)

plt.plot(signal / max(signal), ms=3, label='Original', color='b')

plt.plot(iir\_filtered / max(iir\_filtered), ms=3, label='IIR', color='r')

plt.xlabel('$time$')

plt.ylabel('$Amplitude$')

plt.xlim(0,370)

plt.legend()

plt.grid()

plt.subplot(4, 1, 4)

plt.plot(signal / max(signal), ms=3, label='Original', color='b')

plt.plot(lms\_filtered / max(lms\_filtered), ms=3, label='LMS', color='m')

plt.xlabel('$time$')

plt.ylabel('$Amplitude$')

plt.xlim(0,370)

plt.legend()

plt.grid()

plt.show()

## b) Link to remote repositorium

<https://github.com/TobiaszWojnar/DSP>

# 4. Outcomes:

A graph with blue and orange lines

AI-generated content may be incorrect.

A graph of a graph

AI-generated content may be incorrect.

A graph of a graph

AI-generated content may be incorrect.

# 5. Conclusions

## FIR filters

In general FIR filters excel at noise reduction without altering the signal's phase, a critical advantage in phase-sensitive applications. However, their performance hinges on coefficient selection and may necessitate higher filter orders for sharp frequency cutoffs.

For my signal filtered signal seemed to be shifted in phase a bit, but it was managed to to capture higher frequency signals

## IIR filters

In general IIR filters, offer superior frequency selectivity with fewer coefficients but introduce phase distortion and require careful design to guarantee stability, especially at higher orders.

For my singal it also shifted phase and higher frequencies where porly captured

## LMS adaptive filters

LMS adaptive filters dynamically adjust to changing noise characteristics, making them suitable for dynamic environments, though their effectiveness depends on the step size and filter length. This comparison underscores the need to carefully consider the trade-offs between these filter types to optimize digital signal processing performance.

For my singal it precisely captured fraquency and phase of the signal but seems to be trying to overemphasis aditinonal frequencies.