# List 3, variant 15 - Tobiasz Wojnar

## Digital Signal Processing 2024/25

## University of Bielsko-Biala, semestr 1

# Task (Variant 15)

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

$$b = 0.2, 0.3, 0.5$$

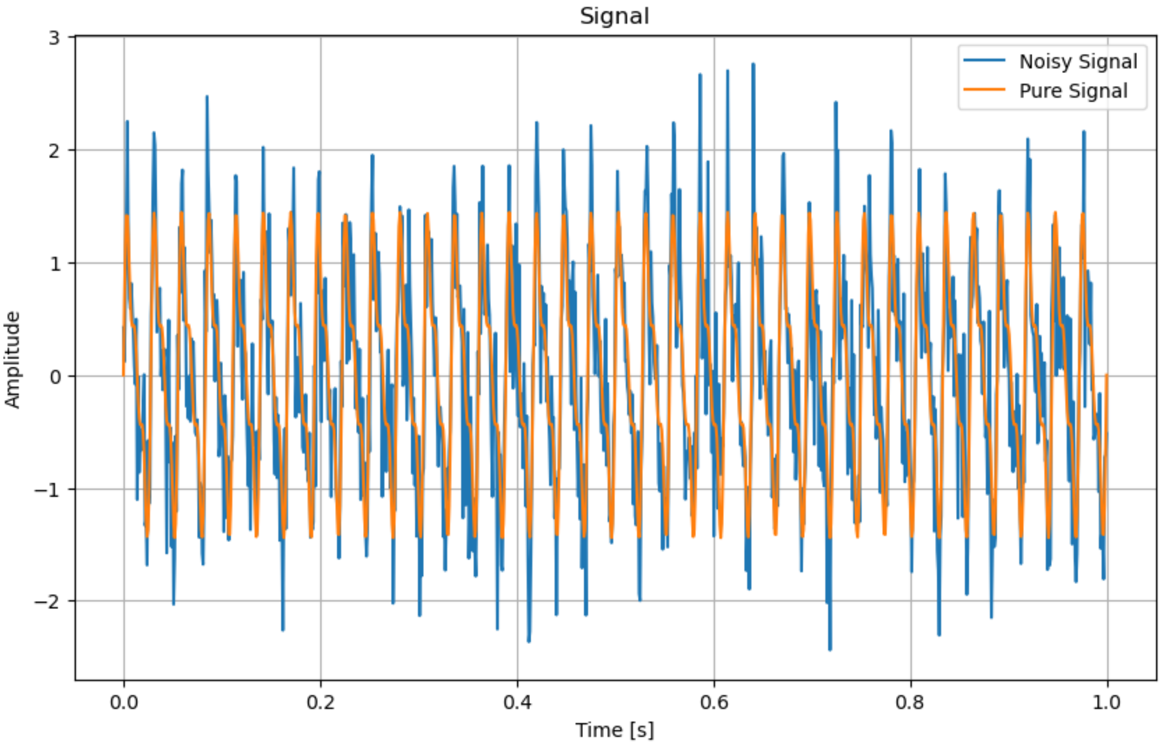
- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal. IIR Filter Coefficients:

$$b = 1, 0.5, a = 1, -0.7$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 6$ to reduce noise in the same noisy sinusoidal signal.

In [12]:

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

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

singal = signal_base + signal_harmonic_1 + signal_harmonic_2

noise = 0.5 * np.random.randn(len(t))

noisy_singal = singal + noise

plt.figure(figsize=(10, 6))
plt.plot(t, noisy_singal, ms=1, label="Noisy Signal")
plt.plot(t, singal, ms=3, label="Pure Signal")
plt.legend()
plt.title("Signal")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```



In [20]:

```
from scipy.signal import lfilter

# $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
```

```python
        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
```