TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATION

Signal Processing
Laboratory work 1

Theme: Discrete Time Systems. Noise Generation and its Distortion

Author: **Catlabuga Corneliu, st. gr. FAF-213**

Checked by: **Railean Serghei, university lecturer**

Chişinău – 2024

Objective:

Noise generation and filtering using the Discrete Time System "M-point Moving Average System"

Theoretical considerations:

Sytem is any device or algorithm that performs operations on the signal.

Discrete Time System is called any device or algorithm, which influences the discrete time Signal called Input or Excitation Signal - x(n)- in coordination with well defined rules to obtain another discrete signal called Output Signal - y(n) – or Answer (Fig. 1). The input signal x(n) is transformed by the system into the signal y(n) and the relationship between x(n) and y(n) is:

$$y(n) \equiv T[x(n)] \qquad \textbf{(1)}$$

where the symbol T is the transformation (or operator) performed by the system on x(n) to obtain y(n). Relationship (1) is illustrated in Fig. 1.

Description of the input-output relationship. The description of the input-output relationship consists of mathematical expressions or fixed rules, which define the relationship between the input signal and the output signal (input-output relationship). The exact internal structure of the system is often unknown or ignored.

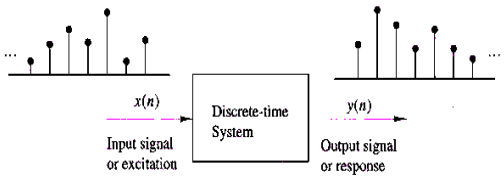

Figure 1: Block diagram of a discrete system

An alternative representation of the expression (1) is:

$$x(n) \xrightarrow{T} y(n) \qquad \textbf{(2)}$$

which represents y(n) as the response of the system T to the excitation x(n).

System representation:

The adder. Figure 2 shows a system (Adder) that adds two signals to obtain a third sequence (sum) y(n).

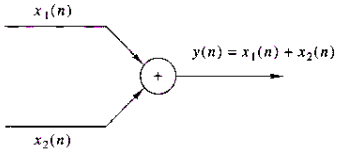

Figure 2: Diagram of an adder

The constant multiplier. This system is illustrated in Figure 3 and represents the multiplication by a constant of the input x(n).

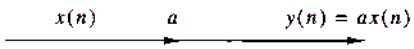

Figure 3: Diagram of a constant multiplier

The multiplier of two signals. Figure 4 illustrates the multiplication of two signals to obtain a third signal (the product), signed in runaway with y(n).

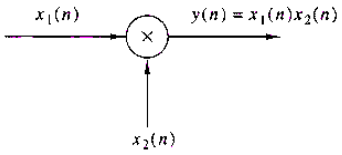

Figure 4: Diagram of a signal multiplier

Detention with a unit. One-unit hold is a system that holds signal values with one unit. Figure 5 illustrates this system. If the input signal is x(n), the output signal will be x(n - 1). The symbol z-1 is used to express this operation.

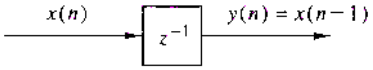

Figure 5: Diagram of a unit retainer

Advance by one unit. Shifting by one shifts the input signal x(n) such that the output signal is x(n +1).

System classification:

Static Systems and Dynamic Systems. The system is called Static or memoryless if the output signal at any time n depends on the input signal at the same time n, and not on previous or future times. In any other case the system is Dynamic cau with memory. If the output signal at a time n depends on the input signal with the interval n-N - up to n, then the System has a memory of duration N. If N=0, the system is Static.

The systems described by the input-output relations below are static systems:

$$y(n) = ax(n)$$

$$y(n) = nx(n) + bx^3(n) \qquad \textbf{(3)}$$

The systems described by the input-output relations below are dynamic systems:

$$y(n) = x(n) + 3x(n-1) \qquad \textbf{(4)}$$

$$y(n) = \sum_{k=0}^{n} x(n-k)$$

$$y(n) = \sum_{k=0}^{\infty} x(n-k)$$

Time Invariant Systems. The system is called time-invariant if the Input-Output characteristic does not depend on time. If we change the input signal by k units – x(n-k) – at the output we will get y(n-k).

$$x(n-k) \overset{T}{\rightarrow} y(n-k) \qquad \textbf{(5)}$$

Non-linear systems and linear systems. The system is called linear if it satisfies the overlap condition - the response of the system to the sum of some signals is equal to the sum of the responses of the system to each separate signal (fig. 6):

$$T[a_1x_1(n) + a_2x_2(n)] = a_1T[x_1(n)] + a_2T[x_2(n)] \qquad \textbf{(6)}$$

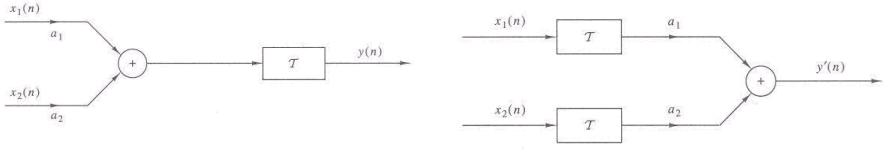

Figure 6: Signal systems for (6)

Tasks:

1. Study of random processes:

1.1-1.2 "White" noise with Gaussian representation is made by the rand procedure. Generate a random process as follows. Plot the histogram of the generated noise, replacing the plot function with hist (beforehand change time to 1 and change to place t,x1).:

```
import matplotlib.pyplot as plt
import random

def main():
    Ts = 0.01
    t = [round(i * Ts, 2) for i in range(int(5 / Ts))]
    x1 = [random.random() for _ in range(len(t))]

    plt.plot(t, x1, linewidth=0.75)
    plt.grid()
    plt.title("Random Signal")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task1.1.png")

    plt.clf()

    t = [round(i * Ts, 2) for i in range(int(1 / Ts))]
    plt.hist(x1, t)
    plt.grid()
    plt.title("Random Signal")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task1.2.png")

if __name__ == "__main__":
    main()
```

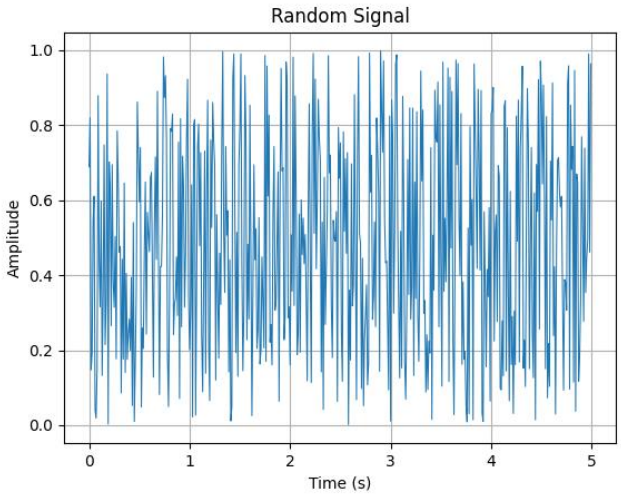

Figure 7: Task 1.1 random signal

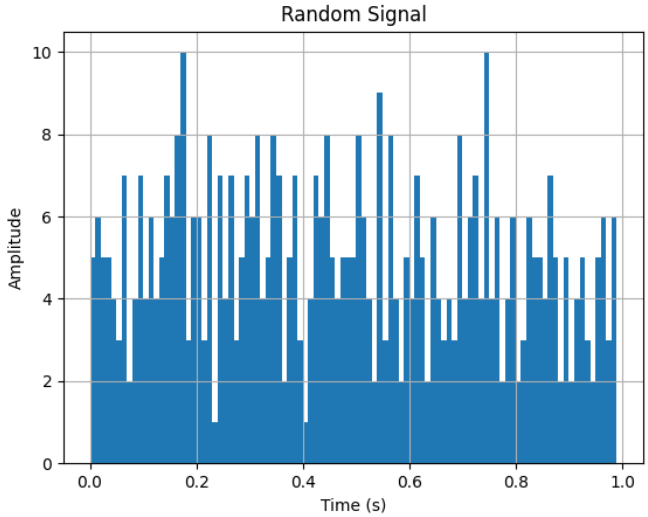

Figure 8: Task 1.2 random signal histogram

1.3-1.4 Repeat p. 1.1 for Ts=0.001 and generate a new noise x2. Represent the histogram of the noise generated x2 in p. 1.3.

```
import matplotlib.pyplot as plt
import random

def main():
    Ts = 0.001
    t = [round(i * Ts, 3) for i in range(int(5 / Ts))]
    x2 = [random.random() for _ in range(len(t))]

    plt.plot(t, x2, linewidth=0.25)
    plt.grid()
    plt.title("Random Signal")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task1.3.png")

    plt.clf()

    t = [round(i * Ts, 3) for i in range(int(1 / Ts))]
    plt.hist(x2, t)
    plt.grid()
    plt.title("Random Signal")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task1.4.png")

if __name__ == "__main__":
    main()
```

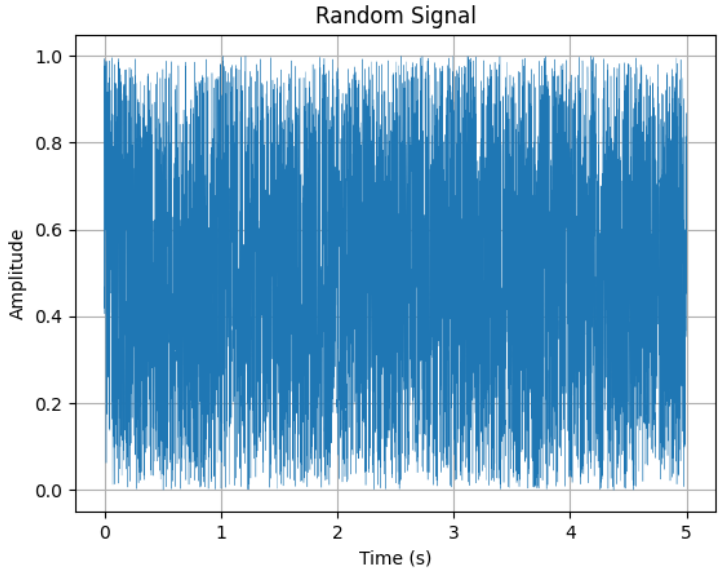

Figure 9: Task 1.3 random signal

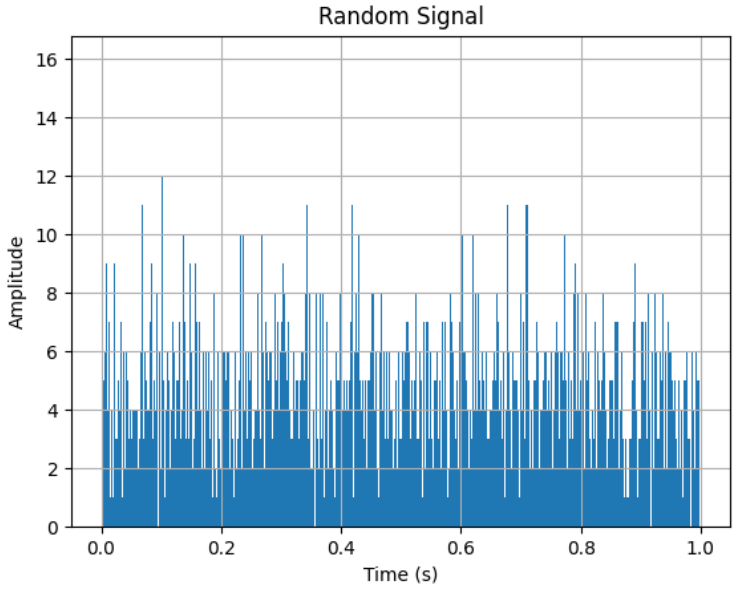

Figure 10: Task 1.4 random signal histogram

1.5 Design a second-order digital filter with the frequency of natural oscillations 1 Hz, pass through this filter the signal x1 and display the signal at the filter output:

```
import matplotlib.pyplot as plt
import random
from math import pi
from scipy.signal import lfilter

def main():
    Ts = 0.01
    om0 = 2 * pi
    dz = 0.005
    A = 1
    oms = om0 * Ts
    a = [
        1 + 2 * dz * oms + oms ** 2,
        -2 * (1 + dz * oms),
        1
    ]
    b = [
        A * 2 * oms ** 2
    ]
    t = [round(i * Ts, 2) for i in range(int(50 / Ts))]
    x1 = [random.random() for _ in range(len(t))]
    y1 = list(lfilter(b, a, x1))


    plt.plot(t, y1, linewidth=0.75)
    plt.grid()
    plt.title("Noise filtering with a 2nd order filter")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task1.5.png")

if __name__ == "__main__":
    main()
```

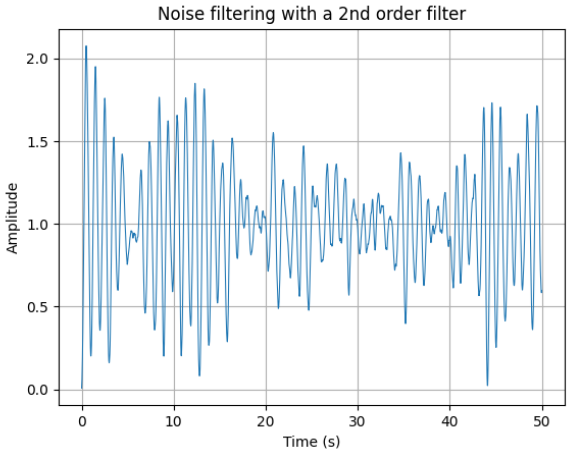

Figure 11: Task 1.5 noise filtering

1.6 Repeat p. 1.5 for Ts=0.001 and the generated noise x2.

```
import matplotlib.pyplot as plt
import random
from math import pi
from scipy.signal import lfilter

def main():
    Ts = 0.001
    om0 = 2 * pi
    dz = 0.005
    A = 1
    oms = om0 * Ts
    a = [
        1 + 2 * dz * oms + oms ** 2,
        -2 * (1 + dz * oms),
        1
    ]
    b = [
        A * 2 * oms ** 2
    ]
    t = [round(i * Ts, 3) for i in range(int(50 / Ts))]
    x2 = [random.random() for _ in range(len(t))]
    y2 = list(lfilter(b, a, x2))


    plt.plot(t, y2, linewidth=0.75)
    plt.grid()
    plt.title("Noise filtering with a 2nd order filter")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task1.6.png")

if __name__ == "__main__":
    main()
```

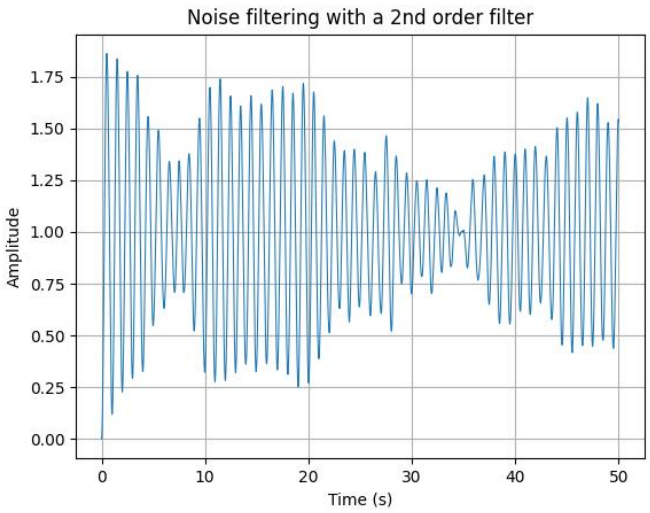

Figure 12: Task 1.6 2nd order noise filtering

2. Filtering signals affected by noise using a MAF filter.

2.1 Generate an original signal affected by noise s(n):

```
import matplotlib.pyplot as plt

def main():
    R = 50
    m = [i for i in range(0, R - 1)]
    s = [2 * i * 0.9 ** i for i in m]

    plt.stem(m, s)
    plt.grid()
    plt.title("Original Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.1.png")

if __name__ == "__main__":
    main()
```

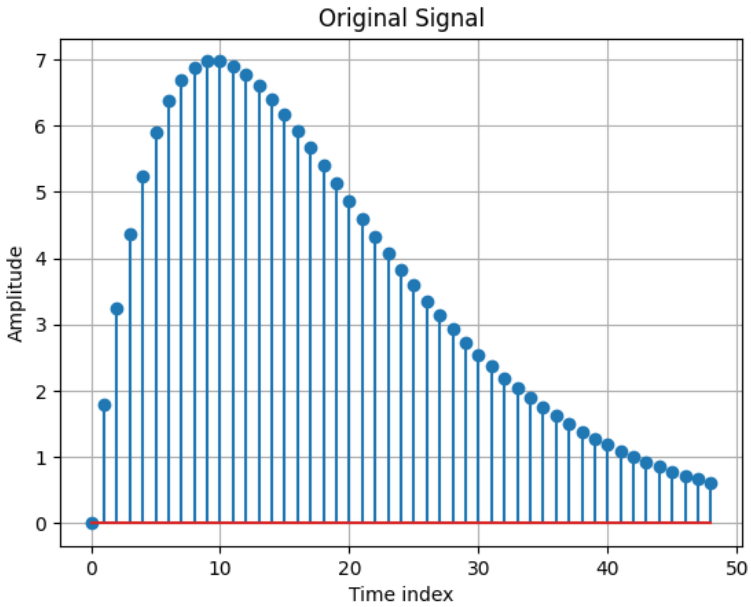

Figure 13: Task 2.1 original signal

2.2 Generate a noise, using the rand function by adding in p. 2.1 the noise d=rand(1,length(m))-0.5.

```
import matplotlib.pyplot as plt
import random

def main():
    R = 50
    m = [i for i in range(0, R - 1)]
    d = [random.random() - 0.5 for _ in range(len(m))]

    plt.stem(m, d)
    plt.grid()
    plt.title("Noise Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.2.png")

if __name__ == "__main__":
    main()
```

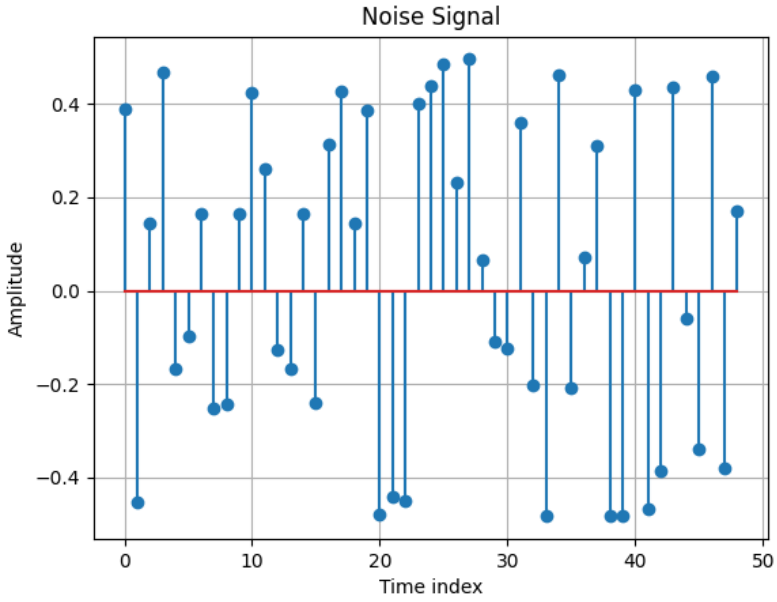

Figure 14: Task 2.2 noise signal

2.3 Plot both these signals as a continuous form on a single graph, using the plot function.

```
import matplotlib.pyplot as plt
import random

def main():
    R = 50
    m = [i for i in range(0, R - 1)]
    s = [2 * i * 0.9 ** i for i in m]
    d = [random.random() - 0.5 for _ in range(len(m))]

    plt.plot(m, s, linewidth=1, label="Original Signal")
    plt.plot(m, d, linewidth=1, label="Noise Signal")
    plt.grid()
    plt.title("Original Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.3.png")

if __name__ == "__main__":
    main()
```

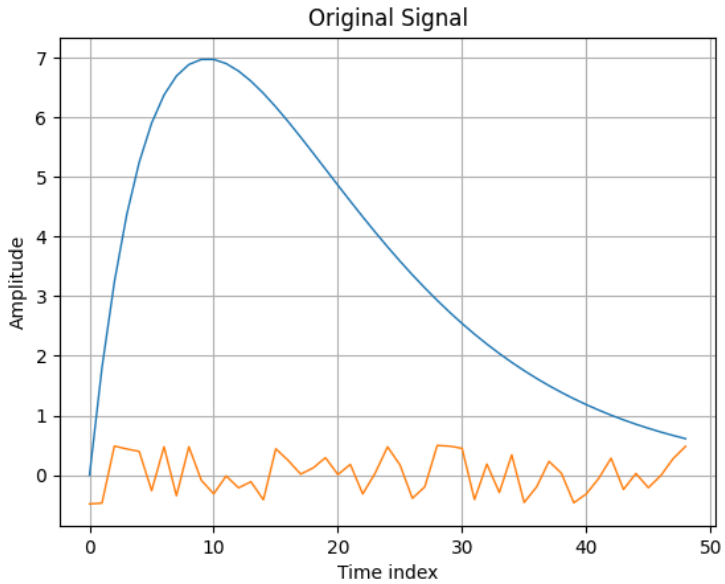

Figure 15: Task 2.3 original signal and noise

2.4 Represent the sum of these two signals x=s+d and represent the resulting signal x and the initial signal s on a single graph, using the plot function.

```
import matplotlib.pyplot as plt
import random

def main():
    R = 50
    m = [i for i in range(0, R - 1)]
    s = [2 * i * 0.9 ** i for i in m]
    d = [random.random() - 0.5 + s[i] for i in range(len(m))]

    plt.plot(m, s, linewidth=1, label="Original Signal")
    plt.plot(m, d, linewidth=1, label="Noisy Signal")
    plt.legend()
    plt.grid()
    plt.title("Original Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.4.png")

if __name__ == "__main__":
    main()
```

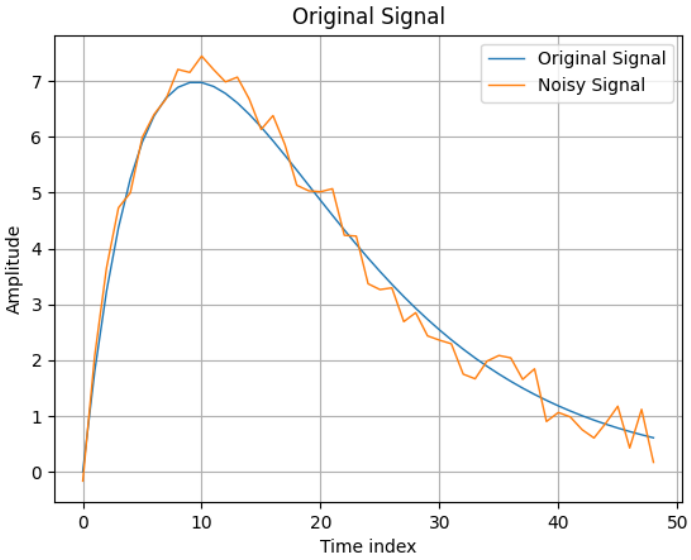

Figure 16: Task 2.4 noisy original signal

2.5 Design a MAF filter with parameters y=filter(b,1,x) b=ones(M,1)/M and previously specifying M=3 and filter the signal affected by noise. Represent the already filtered signal y and the one affected by the noiseg x, but also the initial one s on a single graph, using the plot function.

```
import matplotlib.pyplot as plt
import random
from scipy.signal import lfilter

def main():
    R = 50
    m = [i for i in range(0, R - 1)]
    s = [2 * i * 0.9 ** i for i in m]
    d = [random.random() - 0.5 + s[i] for i in range(len(m))]
    M = 3
    b = [1 / M for _ in range(M)]
    y = lfilter(b, 1, d)

    plt.plot(m, s, linewidth=1, label="Original Signal")
    plt.plot(m, d, linewidth=1, label="Noisy Signal")
    plt.plot(m, y, linewidth=1, label="Filtered Signal")
    plt.legend()
    plt.grid()
    plt.title("MAF Filtered Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.5.png")

if __name__ == "__main__":
    main()
```

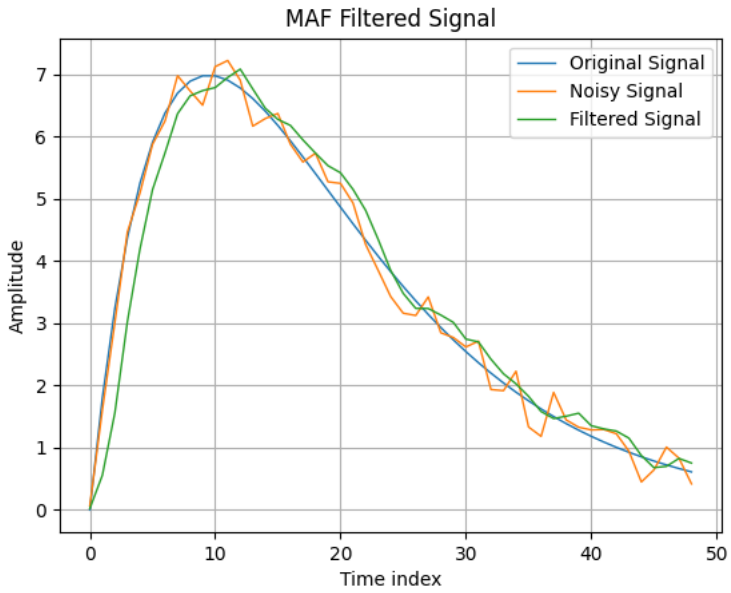

Figure 17: Task 2.5 M=3 MAF filtered signal

2.6 Repeat p.2.5 for M=5 and M=10. Compare the results obtained.

```
import matplotlib.pyplot as plt
import random
from scipy.signal import lfilter

def main():
    R = 50
    m = [i for i in range(0, R - 1)]
    s = [2 * i * 0.9 ** i for i in m]
    d = [random.random() - 0.5 + s[i] for i in range(len(m))]
    M0 = 3
    b0 = [1 / M0 for _ in range(M0)]
    y0 = lfilter(b0, 1, d)
    M1 = 5
    b1 = [1 / M1 for _ in range(M1)]
    y1 = lfilter(b1, 1, d)
    M2 = 10
    b2 = [1 / M2 for _ in range(M2)]
    y2 = lfilter(b2, 1, d)

    plt.plot(m, s, linewidth=1, label="Original Signal")
    plt.plot(m, d, linewidth=1, label="Noisy Signal")
    plt.plot(m, y0, linewidth=1, label="M=3 Filtered Signal")
    plt.plot(m, y1, linewidth=1, label="M=5 Filtered Signal")
    plt.plot(m, y2, linewidth=1, label="M=10 Filtered Signal")
    plt.legend()
    plt.grid()
    plt.title("MAF Filtered Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.6.png")

if __name__ == "__main__":
    main()
```

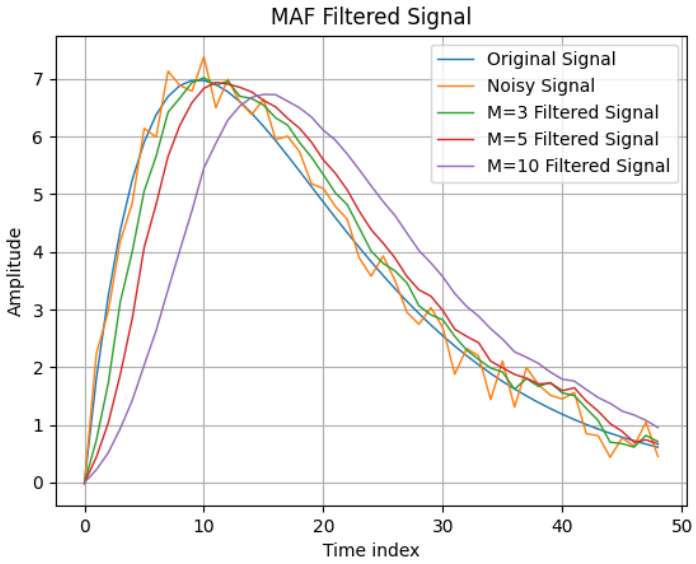

Figure 18: Task 2.6 M=3, 5 and 10 MAF filtered signal

2.7 Repeat p.2.5 for another signal – s=2* sawtooth(3*pi*m+pi/6) and changing the time step to a smaller one - m = 0:0.001:R. (R=1) and the value of M=20. Represent the already filtered signal y and the one affected by the noiseg x, but also the initial one s on a single graph, using the plot function.

```
import matplotlib.pyplot as plt
import random
from scipy.signal import lfilter, sawtooth
from math import pi

def main():
    R = 1
    m = [round(i * 0.001, 3) for i in range(0, R * 1000)]
    s = [2 * sawtooth(3 * pi * i + pi / 6) for i in m]
    d = [random.random() - 0.5 + i for i in s]
    M0 = 20
    b0 = [1 / M0 for _ in range(M0)]
    y0 = lfilter(b0, 1, d)

    plt.plot(m, s, linewidth=0.75, label="Original Signal")
    plt.plot(m, d, linewidth=0.75, label="Noisy Signal")
    plt.plot(m, y0, linewidth=0.75, label="M=20 Filtered Signal")
    plt.legend()
    plt.grid()
    plt.title("MAF Filtered Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.7.png")

if __name__ == "__main__":
    main()
```

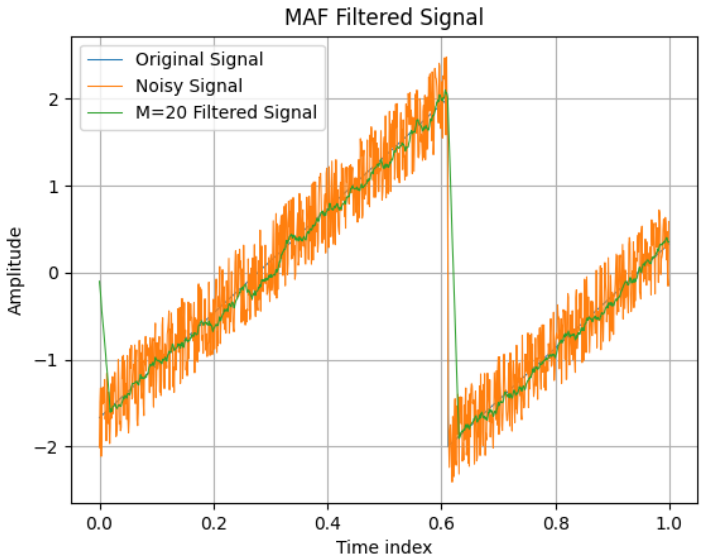

Figure 19: Task 2.7 M=20 MAF filtered signal

2.8 Repeat p.2.7 for M=50 and M=100. Compare the results obtained.

```
import matplotlib.pyplot as plt
import random
from scipy.signal import lfilter, sawtooth
from math import pi

def main():
    R = 1
    m = [round(i * 0.001, 3) for i in range(0, R * 1000)]
    s = [2 * sawtooth(3 * pi * i + pi / 6) for i in m]
    d = [random.random() - 0.5 + i for i in s]
    M0 = 20
    b0 = [1 / M0 for _ in range(M0)]
    y0 = lfilter(b0, 1, d)
    M1 = 50
    b1 = [1 / M1 for _ in range(M1)]
    y1 = lfilter(b1, 1, d)
    M2 = 100
    b2 = [1 / M2 for _ in range(M2)]
    y2 = lfilter(b2, 1, d)

    plt.plot(m, s, linewidth=0.75, label="Original Signal")
    plt.plot(m, d, linewidth=0.75, label="Noisy Signal")
    plt.plot(m, y0, linewidth=0.75, label="M=20 Filtered Signal")
    plt.plot(m, y1, linewidth=0.75, label="M=50 Filtered Signal")
    plt.plot(m, y2, linewidth=0.75, label="M=100 Filtered Signal")
    plt.legend()
    plt.grid()
    plt.title("MAF Filtered Signal")
    plt.xlabel("Time index")
    plt.ylabel("Amplitude")
    plt.savefig("plots/task2.8.png")

if __name__ == "__main__":
    main()
```

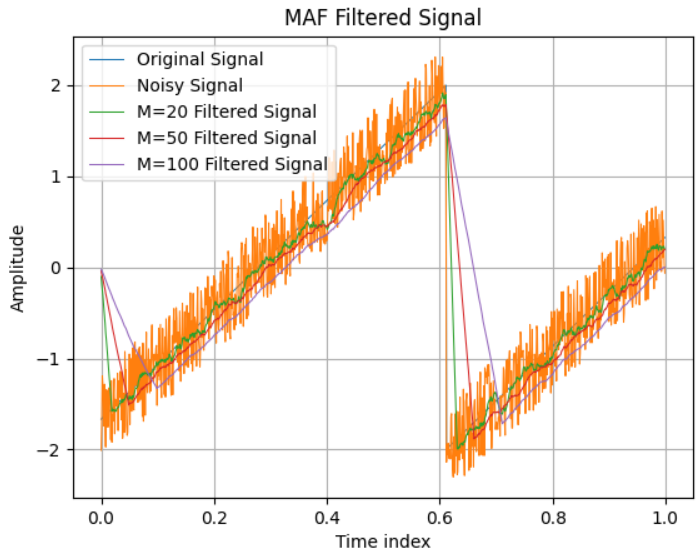

Figure 20: Task 2.8 M=20, 50 and 100 MAF filtered signal

3. 3. Investigate filtering processes using the LMS and RLS algorithm. For this:

3.1. Launch the MATLAB package.

3.2. Choose the demo command. Open the Blocksets app, then the DSP app. Open the Adaptive processing folder.

3.3. Open and launch the Niose canceller (LMS) application. Save the block diagram and the corresponding dependencies. Save the information about the LMS algorithm by pressing the INFO button in the modeling window.

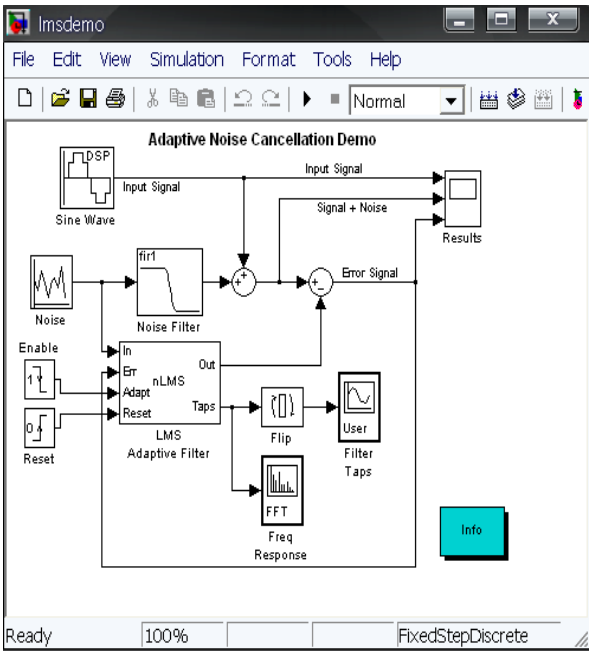

Figure 21: LMS diagram

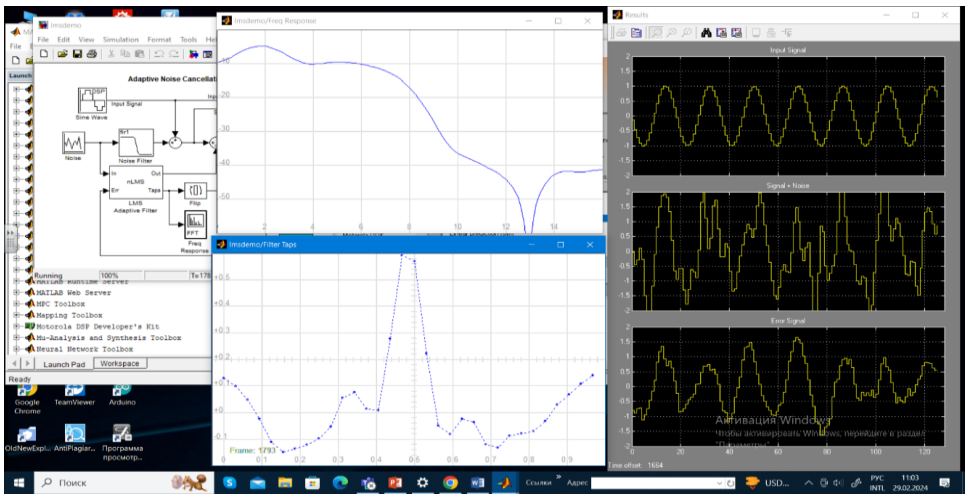

Figure 22: LMS output

3.4. Open and launch the Niose canceller (RLS) application. Save the block diagram and the corresponding dependencies. Save the information about the RLS algorithm by pressing the INFO button in the modeling window.

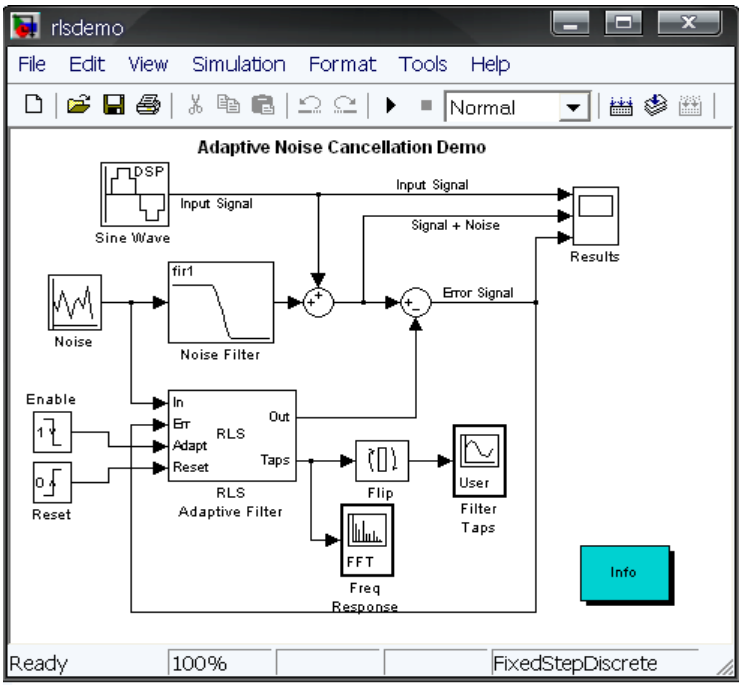

Figure 23: RMS diagram

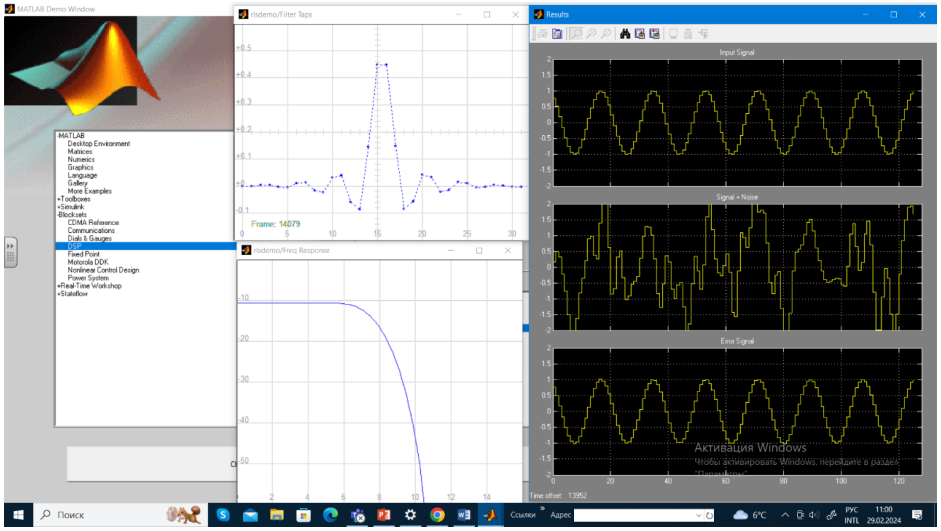

Figure 24: RMS output

Conclusions:

In this laboratory experiment, the objective was to explore noise generation and filtering using the Discrete Time System known as the M-point Moving Average System. Through the implementation of this system in MATLAB, we were able to simulate the generation of random noise and effectively filter it using the moving average technique. By generating random noise signals and passing them through the M-point moving average filter, we observed a significant reduction in noise amplitude, thus demonstrating the effectiveness of the filtering process in attenuating unwanted noise components. The MATLAB plots provided clear visual representations of the original noisy signals and the filtered outputs, aiding in the understanding and analysis of the signal processing techniques employed.

In conclusion, this laboratory work has provided valuable insights into the principles of noise generation and filtering using the M-point Moving Average System in the context of signal processing. Through hands-on experimentation and MATLAB simulations, we gained practical experience in generating random noise, implementing signal filtering algorithms, and analyzing the effects of filtering on noisy signals. The experiment not only enhanced our understanding of signal processing concepts but also equipped us with essential skills in MATLAB programming for signal analysis and manipulation. Overall, this exercise underscores the importance of noise reduction techniques in improving the quality and reliability of signal processing applications across various domains.

References:

1. Github laboratory work repository: https://github.com/muffindud/PS_Labs/tree/main/lab1