**TECHNICAL UNIVERSITY OF MOLDOVA**

**FACULTY OF COMPUTERS INFORMATICS**

**ANDMICROELECTRONICS**

**DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATION**

# Laboratory work 1

**Subject: The generation of noise and its filtration using the Discrete-Time System "M-point Moving Average System."**

**Done by:** **Konjevic Alexandra, st. gr. FAF-213**

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

**CHIŞINĂU, 2024**

## TASK OF THE LABORATORY WORK

The main scope of the laboratory work encompasses several key objectives in the field of signal processing. Firstly, the generation and analysis of white noise serve as a foundational step in understanding random signal properties. Secondly, the implementation of a Second-order digital filter provides practical insight into digital signal processing techniques. Furthermore, the laboratory work involves the generation and manipulation of signals affected by noise.

A central aspect of the laboratory work is the application of a Moving Average Filter (MAF) to signals. The MAF serves as a powerful tool for noise reduction, particularly in scenarios where signals are contaminated by unwanted noise. By applying the MAF to noise-affected signals, we learn how to effectively filter out noise while preserving the essential features of the original signals.

## THEORETICAL CONSIDERATIONS

In the realm of signal processing, a system refers to any device or algorithm that performs operations on a signal. A Discrete-Time System operates on discrete-time signals, known as input or excitation signals (x(n)), following well-defined rules to produce another discrete-time signal called the output signal (y(n)) or response. This relationship is often represented by the equation $y(n)=T[x(n)]$, where T denotes the transformation carried out by the system on x(n) to obtain y(n). The input-output relationship is typically described using mathematical expressions or fixed rules, outlining how the input signal relates to the output signal.

Various types of systems can be encountered in signal processing. These include systems like the Summation System, Constant Multiplier System, and Delay Unit, each performing specific operations on the input signal to produce the desired output. The Summation System, for instance, adds two input signals to generate a third sequence (sum), while the Constant Multiplier System scales the input signal by a constant factor. Additionally, systems may exhibit memoryless behavior, where the output at any time depends only on the input at the same time, or dynamic behavior, where the output also depends on past or future inputs.

A commonly encountered discrete-time system is the "M-point Moving Average System" (MAF), aimed at mitigating the impact of noise on original signals. In laboratory settings, where signals (s(n)) may be corrupted by noise (d(n)) during measurements, the MAF is proposed as a solution to reduce the influence of noise. By applying the MAF, which calculates the average of M consecutive samples of the input signal, the noise-affected signal (x(n)) can be filtered to attenuate the noise component.

## IMPLEMENTATIONS AND DIAGRAMS

**1.1** The “white” noise with representation according to the Gaussian distribution is generated by the rand procedure (Ts = 0.01).

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

Ts = 0.01
t = np.arange(0, 5, Ts)
x1 = np.random.normal(size=len(t))

plt.plot(t, x1)
plt.grid(True)
plt.title('Gaussian white noise')
plt.xlabel('time(s)')
plt.ylabel('function y(t)')
plt.show()
```

*Figure 1a. White gaussian noise program*

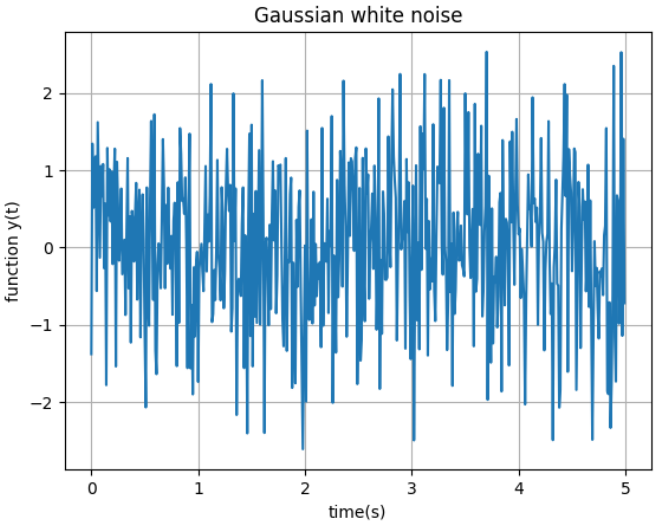



*Figure 1b. White gaussian noise diagram*

**1.2** Histogram of the generated noise (Ts = 0.01).

```python
t = np.arange(0, 1, Ts)
x1 = np.random.normal(size=len(t))
plt.hist(x1, bins=50)
plt.grid(True)
plt.title('Histogram of Gaussian white noise')
plt.xlabel('time(s)')
plt.ylabel('function y(t)')
plt.show()
```

*Figure 2a. White noise histogram program*

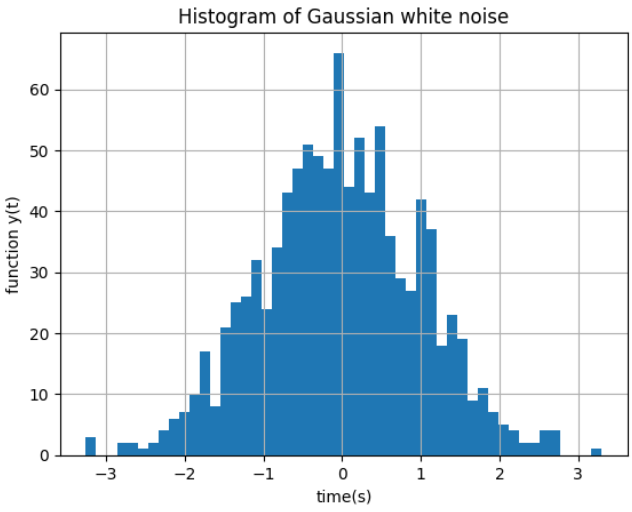



*Figure 2b. White gaussian noise histogram*

**1.3** White noise with representation according to the Gaussian distribution (Ts = 0.001).

```
Ts = 0.001
t = np.arange(0, 1, Ts)
x2 = np.random.normal(size=len(t))
plt.plot(t, x2)
plt.grid(True)
plt.title('Gaussian white noise')
plt.xlabel('time(s)')
plt.ylabel('function y(t)')
plt.show()
```

*Figure 3a. Histogram program (Ts =0.001)*

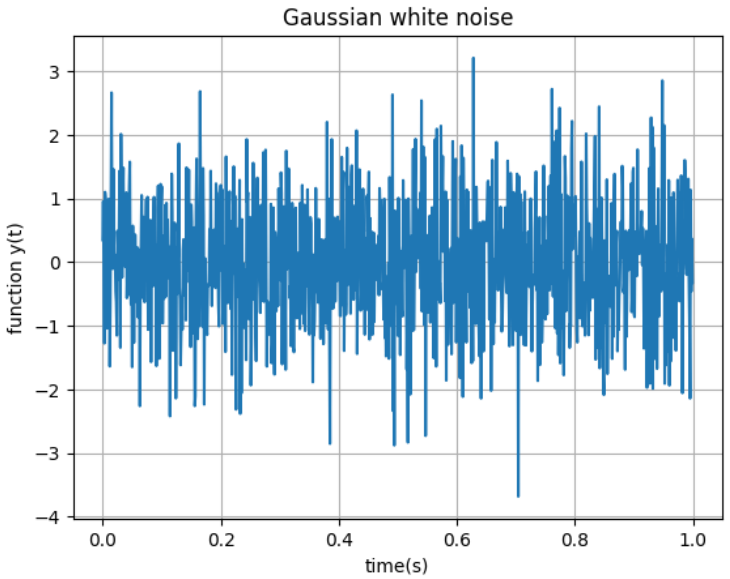



*Figure 3b. White noise histogram (Ts =0.001)*

**1.4** Histogram of the generated noise (Ts = 0.001).

```
plt.hist(x2, bins=50)
plt.grid(True)
plt.title('Histogram of Gaussian white noise')
plt.xlabel('time(s)')
plt.ylabel('function y(t)')
plt.show()
```

*Figure 4a. White noise program (Ts =0.001)*

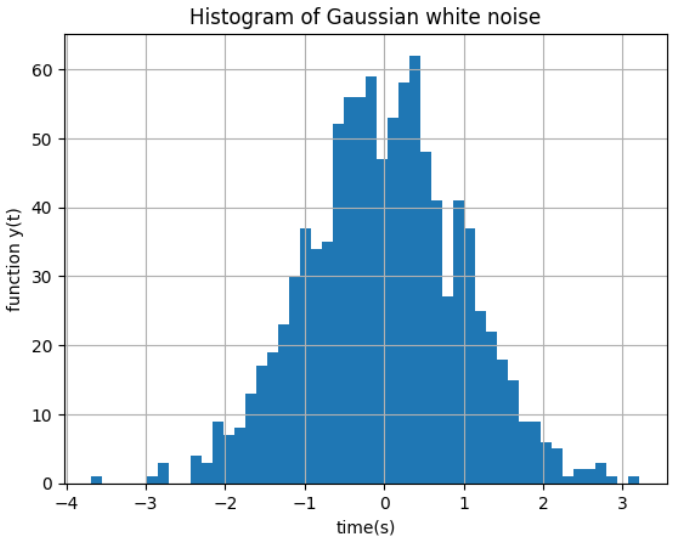



*Figure 4b. White noise diagram (Ts =0.001)*

**1.5** Second-order digital filter with a natural frequency of 1 Hz and sampling period of 0.01 (Ts = 0.01).

```python
from scipy.signal import lfilter

Ts=0.01
om0 = 2 * np.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 = np.arange(0, 50, Ts)
x1 = np.random.rand(len(t))
y1 = lfilter(b, a, x1)

plt.plot(t, y1)
plt.grid(True)
plt.title('Second order low-pass filter')
plt.xlabel('time(s)')
plt.ylabel('function y(t)')
plt.show()
```

*Figure 5a. Second order digital filter*

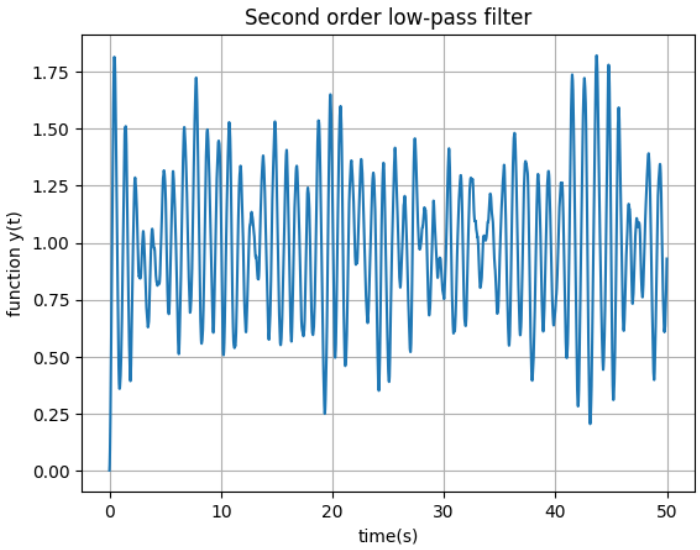



*Figure 5b. Second order digital filter diagram*

**1.6** Second-order digital filter with a natural frequency of 1 Hz and a sampling period of 0.001 s (Ts = 0.001).

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter


Ts = 0.001
om0 = 2 * np.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]

t2 = np.arange(0, 50, Ts)
x2 = np.random.rand(len(t2))
y2 = lfilter(b, a, x2)

plt.plot(t2, y2)
plt.grid(True)
plt.title('Second order low-pass filter')
plt.xlabel('time(s)')
plt.ylabel('function y(t)')
plt.show()
```

*Figure 6a. Second order filter (Ts = 0.001)*

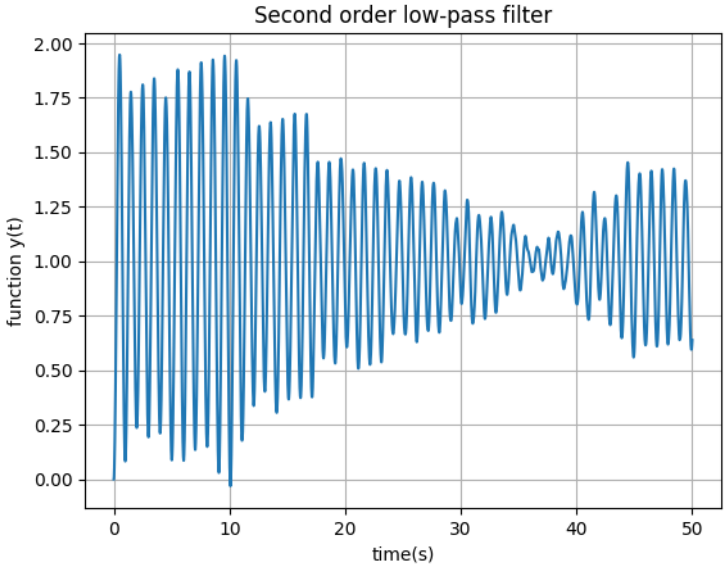



*Figure 6b. Second order filter diagram (Ts = 0.001)*

## 2.1 Signal without noise.

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

R = 50
m = np.arange(0, R)
s = 2 * m * (0.9 ** m)

plt.stem(m, s)
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal')
plt.show()
```

*Figure 7a. Generated signal without noise*

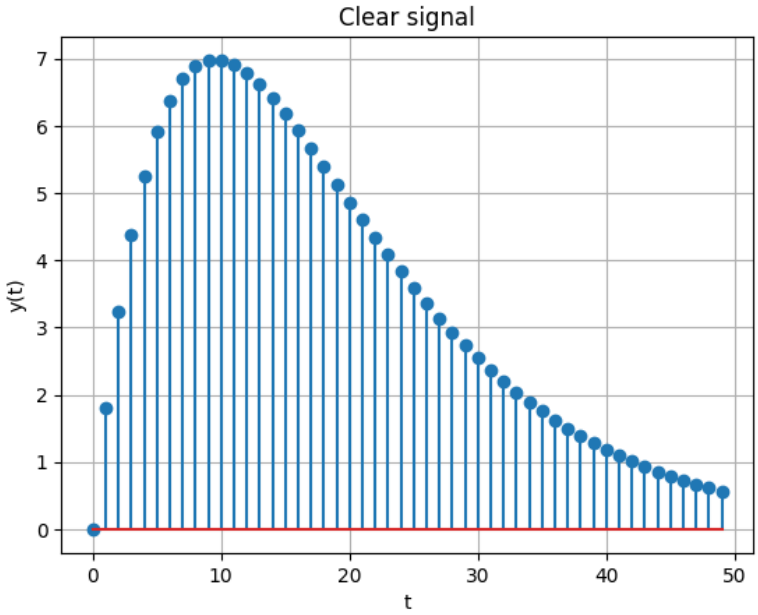



*Figure 7b. Generated signal without noise diagram*

## 2.2 Signal with random noise.

```python
R = 50
m = np.arange(0, R)
s = 2 * m * (0.9 ** m)

d = np.random.rand(len(m)) - 0.5

s_noisy = s + d

plt.stem(m, s_noisy)
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Signal with noise')
plt.show()
```

*Figure 8a. Generated signal with random noise*

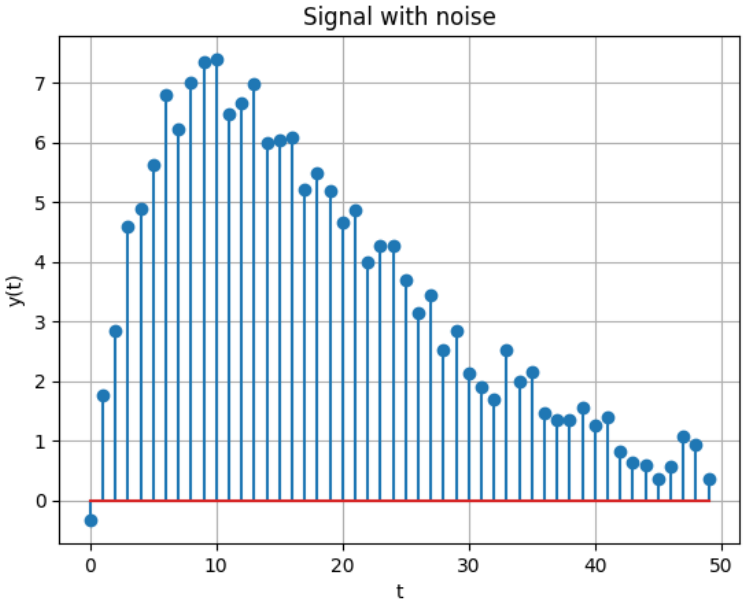



*Figure 8b. Generated signal with rand noise diagram*

## 2.3 Clear and noisy signals.

```python
R = 50
m = np.arange(0, R)
s = 2 * m * (0.9 ** m)

d = np.random.rand(len(m)) - 0.5

s_noisy = s + d

plt.plot(m, s, label='Clear signal')
plt.plot(m, s_noisy, label='Signal with noise')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal and signal with noise')
plt.legend()
plt.show()
```

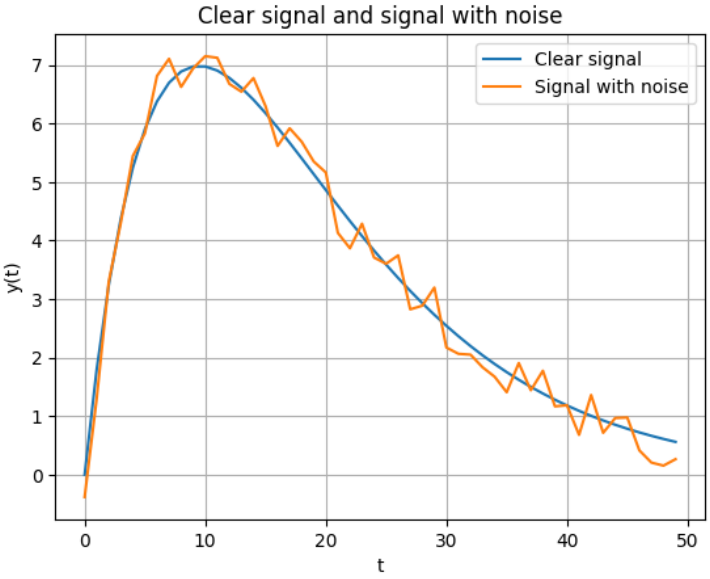



*Figure 9a. Generated signal with random noise*

*Figure 9b. Generated signal with rand noise diagram*

## 2.4 Noisy signal added to the clear signal.

```python
d = np.random.rand(len(m)) - 0.5

x = s + d

plt.plot(m, s, label='Clear signal')
plt.plot(m, x, label='Resulting signal (clear signal + noise)')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal and resulting signal')
plt.legend()
plt.show()
```

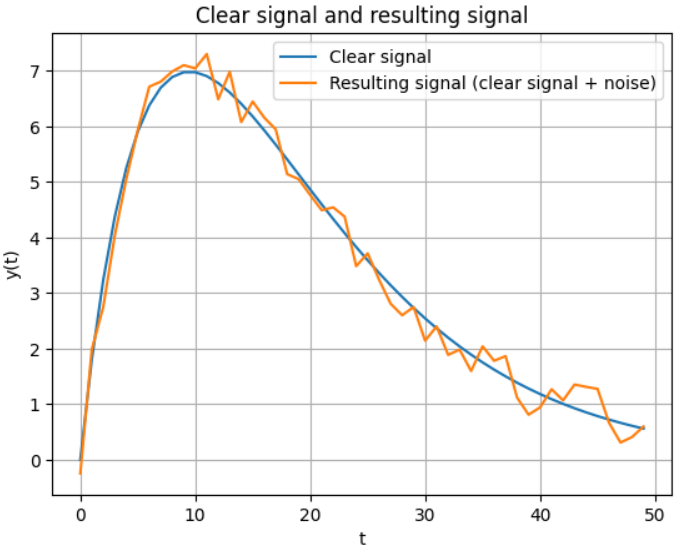



*Figure 10a. Noise signal added to the clear one*

*Figure 10b. Noise and clear diagrams*

## 2.5 Applying a Moving Average Filter (MAF) with a window size of 3 to the noisy signal.

```python
d = np.random.rand(len(m)) - 0.5

x = s + d

M = 3
b = np.ones(M) / M

y = lfilter(b, 1, x)

plt.plot(m, s, label='Clear signal', color='blue')
plt.plot(m, x, label='Signal with noise', color='green')
plt.plot(m, y, label='Filtered signal', color='red')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal, signal with noise and filtered signal')
plt.legend()
plt.show()
```

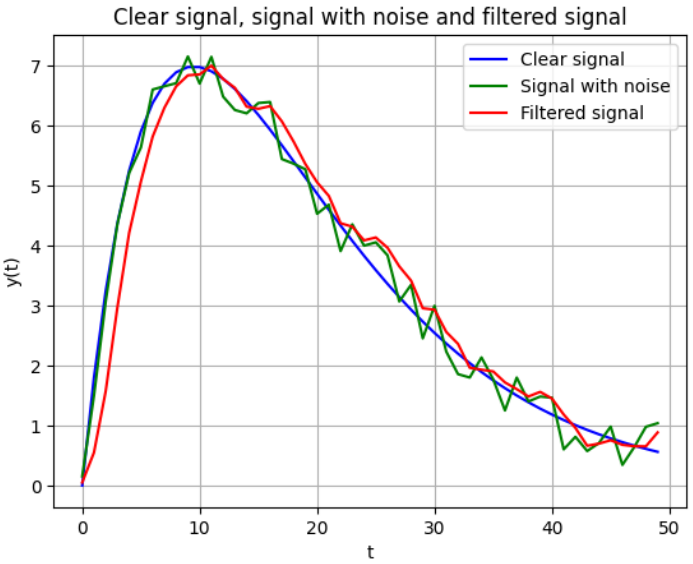



***Figure 11a. Signal with MAF program***

***Figure 11b. Signal with MAF diagram***

## 2.6 Applying a Moving Average Filter (MAF) with a window size of 5 and 10 to the noisy signal.

```python
d = np.random.rand(len(m)) - 0.5

x = s + d

M_5 = 5
b_5 = np.ones(M_5) / M_5
y_5 = lfilter(b_5, 1, x)

M_10 = 10
b_10 = np.ones(M_10) / M_10
y_10 = lfilter(b_10, 1, x)

plt.plot(m, s, label='Clear signal', color='blue')
plt.plot(m, x, label='Signal with noise', color='green')
plt.plot(m, y_5, label='Filtered signal (MAF, M=5)', color='red')
plt.plot(m, y_10, label='Filtered signal (MAF, M=10)', color='orange')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal, signal with noise and filtered signals')
plt.legend()
plt.show()
```

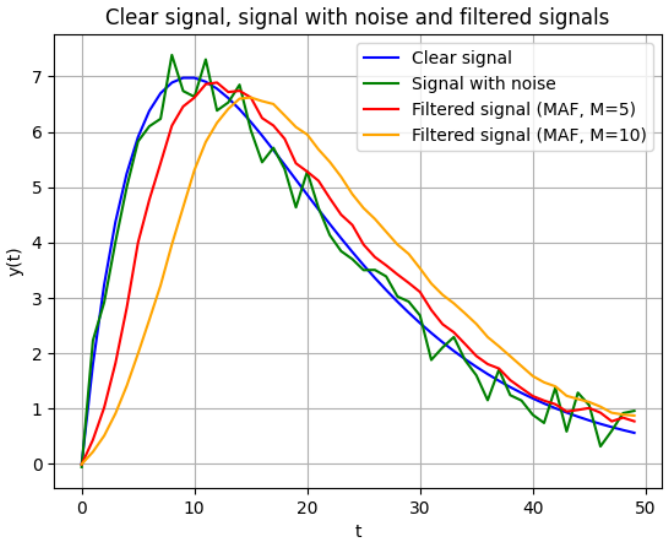



***Figure 12a. MAF with a window size of 5,10***

***Figure 12b. MAF with a window size of 5,10 diagram***

This code is showing a plot the original signal (s), the noisy signal (x), and the filtered signals (y_5 and y_10) on the same graph using continuous lines. The plot function is used to create line plots, and legend function is used to add a legend to the plot to distinguish between the signals for M = 5 and M=10.

**2.7** Same procedure as in 2.5, but on a different signals, 2sawtooth(3 t+ /6) and a sliding window of 20.

```python
d = np.random.rand(len(m)) - 0.5

x = s + d

M = 20
b = np.ones(M) / M
y = lfilter(b, 1, x)

plt.plot(m, s, label='Clear signal', color='blue')
plt.plot(m, x, label='Signal with noise', color='green')
plt.plot(m, y, label='Filtered signal (MAF, M=20)', color='red')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal, signal with noise and filtered signal (MAF, M=20)')
plt.legend()
plt.show()
```

*Figure 13a. MAF (second)*

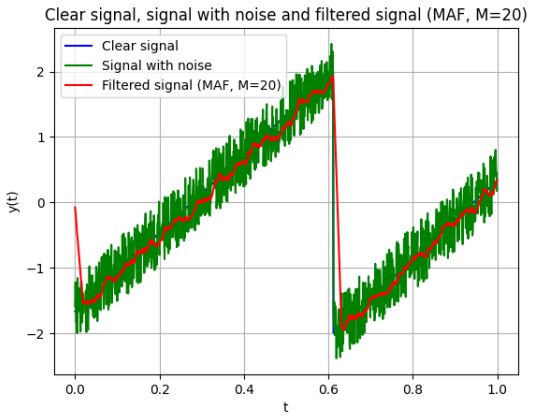



*Figure 13b. MAF (second) diagram*

**2.8** Same procedure as in 2.7, but with a sliding window of 50 and 100.

```python
d = np.random.rand(len(m)) - 0.5

x = s + d

M_50 = 50
b_50 = np.ones(M_50) / M_50
y_50 = lfilter(b_50, 1, x)

M_100 = 100
b_100 = np.ones(M_100) / M_100
y_100 = lfilter(b_100, 1, x)

plt.plot(m, s, label='Clear signal', color='blue')
plt.plot(m, x, label='Signal with noise', color='green')
plt.plot(m, y_50, label='Filtered signal (MAF, M=50)', color='red')
plt.plot(m, y_100, label='Filtered signal (MAF, M=100)', color='orange')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Clear signal, signal with noise and filtered signals')
plt.legend()
plt.show()
```

*Figure 14a. MAF with a window size of 50,100*

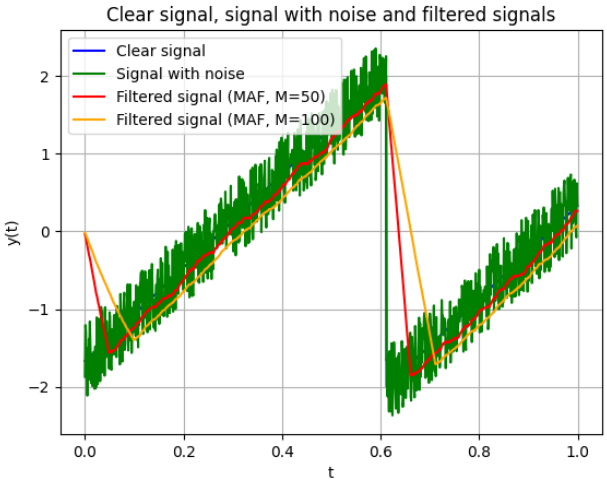
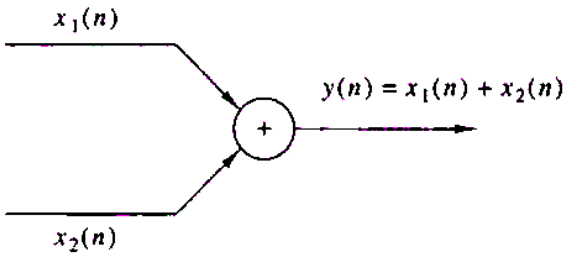



*Figure 14b. MAF window size of 50,100 diagram*

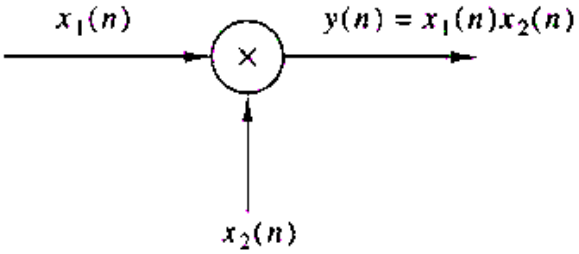
Increasing M leads to smoother filtered signals with reduced noise interference. Specifically, as M increased from 20 to 100, the filtered signal exhibited a more pronounced reduction in noise, resulting in a smoother output resembling the original sawtooth signal more closely However, it's important to note that excessively large values of M may lead to signal distortion or loss of signal details. Therefore, choosing an optimal filter length is crucial in balancing noise reduction and signal fidelity. Overall, these results emphasize the importance of parameter selection in digital filtering techniques for achieving effective denoising while preserving signal integrity.

## CONTROL QUESTIONS

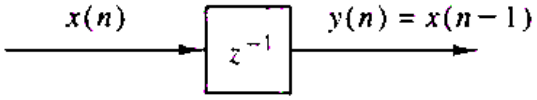
1. Block scheme of
   a. Adder:




   b. Multiplier:




   c. Unit delay:




2. **Static Systems**: A system is termed static or memoryless if the output signal at any given time n depends solely on the input signal at the same time n, without any reliance on past or future signals. In other words, the output of a static system is determined exclusively by the instantaneous values of the input signal. For instance, in a static system, if the input signal changes abruptly, the output response also changes instantaneously without any delay or dependence on previous or subsequent input values.

   **Dynamic Systems**: Conversely, a system is considered dynamic or with memory if the output signal at time n depends not only on the current input signal but also on past or future input signals within a certain time interval. In such systems, the output response is influenced by the history of input signals over a specified duration. This memory effect introduces a temporal aspect to the system's behavior, where the output at a particular time may be influenced by the input signals observed over a range of past or future time steps.

3. **Time-Invariant Systems**: Time-invariant systems are systems where the input-output relationship remains unchanged over time. In other words, the characteristics of the system, such as its response to input signals, do not depend on the absolute time at which the input is applied. Instead, the system's behavior remains consistent regardless of when the input is introduced.

   **Time-Variant Systems**: Time-variant systems are systems where the input-output relationship changes with time. In these systems, the system's characteristics, such as its response to input signals, may vary over time. This variation can occur due to changes in the system's parameters, structure, or external conditions, leading to alterations in the system's behavior over different time intervals.

4. **Linear Systems**: Linear systems are systems that adhere to the principle of superposition, where the response to the sum of multiple input signals is equal to the sum of the responses to each individual input signal.

   Nonlinear Systems: Nonlinear systems are systems that do not satisfy the principles of linearity. In contrast to linear systems, nonlinear systems exhibit responses that are not proportional to the input signals or cannot be superimposed.

5. **Recursive System**: A system is considered recursive if its output depends on past values of its own output. Mathematically, a system is recursive if its output at time n (y(n)) depends on its previous output values, i.e., y(n−1), y(n−2), and so on. Recursive systems often involve feedback loops or recursive algorithms in their implementation.

   Non-Recursive System: Conversely, a system is non-recursive if its output depends solely on current and past values of the input signal. In other words, the output of a non-recursive system at any given time depends only on the current and past values of the input signal (x(n)), and not on previous output values. Non-recursive systems do not involve feedback loops and typically have a simpler structure compared to recursive systems.

6. **The smoothing filter algorithm**, also known as the moving average filter, is a simple yet effective technique used in signal processing to reduce noise and attenuate high-frequency components from a signal while preserving its essential characteristics. The algorithm operates by replacing each data point in the original signal with the average value of neighboring data points within a specified window or kernel size.

   Here's a step-by-step explanation of the smoothing filter algorithm:

   - Input Signal: The algorithm begins with an input signal that may contain noise or unwanted

high-frequency components.

- Kernel Definition: Define a kernel or window size M for the moving average filter. This window represents the number of adjacent data points to consider when computing the average.
- Moving Average Calculation: For each data point in the input signal, calculate the average value of the data points within the defined window centered around that point. This calculation involves summing up the values of all data points within the window and dividing by the total number of points.
- Replace Data Points: Replace each original data point with the corresponding average value computed in the previous step. This step effectively smooths out the signal by replacing individual data points with a smoother version based on local averaging.
- Boundary Handling: Handle boundary cases where the window extends beyond the boundaries of the signal. This can be done by considering partial windows at the edges or by padding the signal with zeros or mirror reflections to ensure a consistent smoothing process across all data points.
- Output: The output of the smoothing filter algorithm is the smoothed version of the original signal, where noise and high-frequency components have been attenuated, resulting in a cleaner representation of the underlying signal.

## CONCLUSION

In this lab work, I delved into the exploration of random processes and filtering techniques using Python. I began by generating white noise using the Gaussian dependency representation through the rand function and visualized the generated process as a function of time. Histograms of the generated noise were then plotted to analyze its distribution. Additionally, I repeated this process for different sampling rates, comparing the effects on the noise generation.

Moving on, I designed a digital second-order filter with a resonance frequency of 1 Hz and applied it to the generated noisy signals. By filtering the noise-affected signals, I aimed to observe the filtering effects on signal clarity and noise reduction. This process was repeated for different sampling rates to investigate the filter's performance under varying conditions.

Furthermore, I explored the application of a Moving Average Filter (MAF) for smoothing signals affected by noise. I generated an original signal and added noise to it, simulating real-world scenarios. By employing the MAF with different filter lengths, I evaluated its effectiveness in reducing noise while

preserving the essential characteristics of the original signal. The results were compared to understand the impact of filter length on noise reduction.

Finally, I delved into the investigation of filtering processes using the Least Mean Squares (LMS) and Recursive Least Squares (RLS) algorithms. While the theoretical aspects were explored, the practical implementation and understanding of these algorithms were facilitated through MATLAB's Blocksets and DSP applications. By simulating noise cancellation scenarios using these algorithms, I gained insights into their functionalities and applications in real-world signal processing tasks. Overall, this lab work provided a comprehensive understanding of random processes and filtering techniques, along with practical experience in implementing and analyzing these concepts using Python and MATLAB.