



Technical University of Moldova
faculty of Computers, Informatics and Microelectronics
Bachelor's Degree Course in Software Engineering

Signal Processing Laboratory Work 1

Discrete systems in time. Noise generation and it's filtration

Executed by:

Polisciuc Vlad

FAF-223

Verified by:

Conf., Univ., Dr. Railean Sergiu

Chisinau 2025

- **Purpose of laboratory work**

Design and study of elementary signals using MATLAB. Noise generation and filtering using discrete-time system "M-point Moving Average System".

- **Theoretical notions**

A *system* is any device or algorithm that performs operations on the signal.

Discrete-Time System is any device or algorithm that influences the *Discrete-Time Signal* called the *Input Signal* or *Excitation Signal* - $x(n)$ - in coordination with well-defined rules to obtain another discrete signal called *Output signal* - $y(n)$ - or *Answer* (see figure Nr.1). The input signal $x(n)$ is transformed by the system into the signal $y(n)$ and the relation between $x(n)$ and $y(n)$ is:

$$y(n) \equiv \mathcal{T}[x(n)] \quad (1)$$

where the symbol T is the transformation (or operator) performed by the system on $x(n)$ to obtain $y(n)$.

The relationship above is illustrated in figure Nr.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.

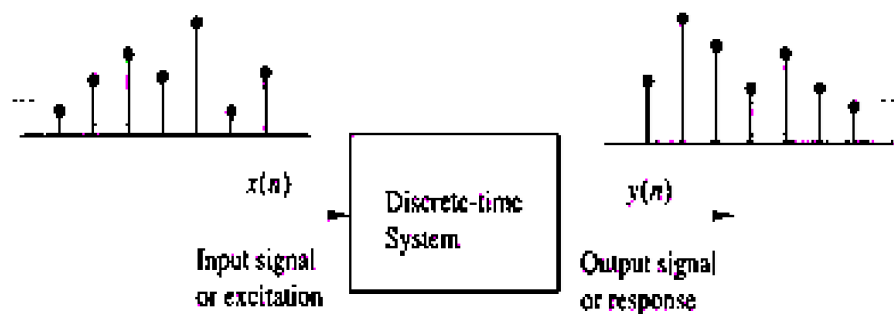


Figure 1: Block diagram of the representation of a discrete system

An alternative notation for first expression is following:

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

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

System representation.

Adder. In figure Nr. 2 is shown a system (Adder) that combines two signals to obtain a third sequence (sum) $y(n)$.

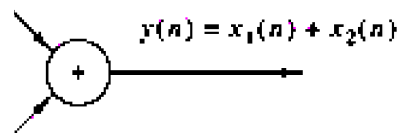


Figure 2: Graphical representation of the adder

Note that it is not necessary to store either one of the sequences in order to perform the addition. In other words, the addition operation is *memoryless*.

Constant multiplier. This system is illustrated in figure Nr. 3 and it represents multiplication with a constant with the input $x(n)$.



Figure 3: Graphical representation of the constant multiplier

Multiplier of two signals. Figure Nr. 4 illustrates the multiplication of two signals to obtain a third signal (product), signed in the figure with $y(n)$.

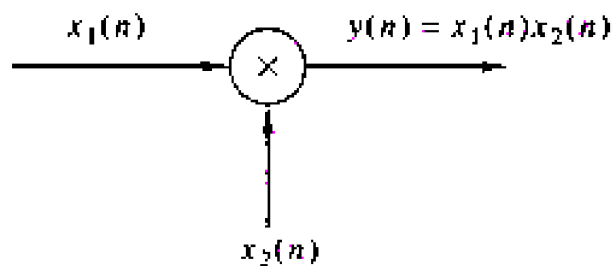


Figure 4: Graphical representation of the multiplier

Unit delay. One-unit delay is a system that retains the values of the one-unit signal. Figure Nr. 5 illustrates this system. If the input signal is $x(n)$, the output signal will be $x(n - 1)$. The z^{-1} symbol is used to express this operation.

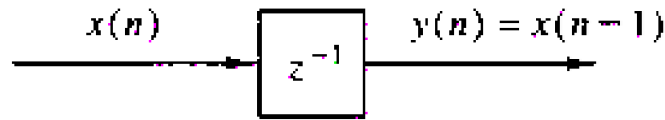


Figure 5: Graphical representation of the unit delay

A unit advance element. Advance by one unit moves the input signal $x(n)$ so that the output signal is $x(n + 1)$.

Systems classification.

Static Systems and Dynamic Systems. The system is called *Static* or *without memory* if the output signal at any time n depends on the input signal at the same time n , and not on the previous or future moments. In any other case, the system is *Dynamic* or *with memory*. If the output signal at time n depends on the input signal with the range $n - N$ - up to n , then the system has a memory with the duration N . If $N = 0$, the system is Static.

The systems described by the input output relationships below are static systems:

$$\begin{aligned}
 y(n) &= ax(n) \\
 y(n) &= nx(n) + bx^3(n)
 \end{aligned}
 \tag{3}$$

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

$$y(n) = x(n) + 3x(n - 1)$$

$$y(n) = \sum_{k=0}^n x(n - k) \quad (4)$$

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

Time invariant systems. The system is called *time invariant* if the Input-Output feature is not time dependent. If we change the input signal by k units - $x(n - k)$ - at the output we will receive $y(n - k)$.

$$x(n - k) \xrightarrow{\mathcal{T}} y(n - k) \quad (5)$$

Some examples with time-variant systems:

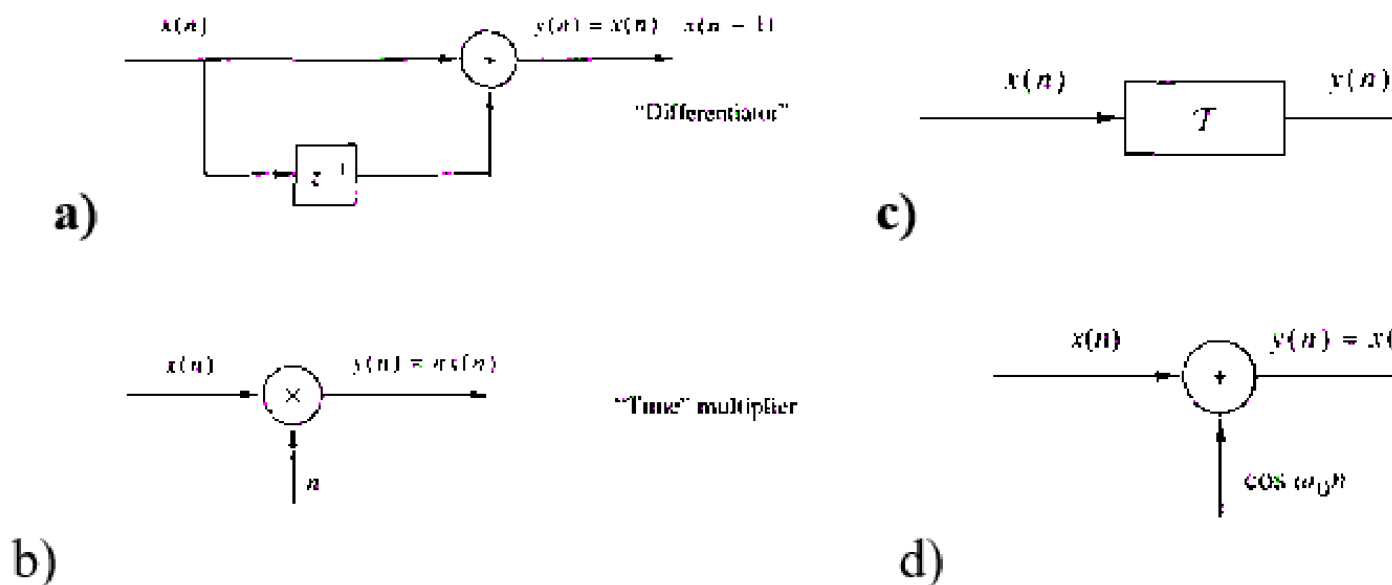


Figure 6: Examples of time-variant systems

Linear systems and Nonlinear systems. The system is called *linear* if it satisfies the overlap condition - the system response to the sum of some signals is equal to the sum of the system responses to each separate signal, see figure Nr. 7:

$$\mathcal{T}[a_1x_1(n) + a_2x_2(n)] = a_1\mathcal{T}[x_1(n)] + a_2\mathcal{T}[x_2(n)] \quad (6)$$

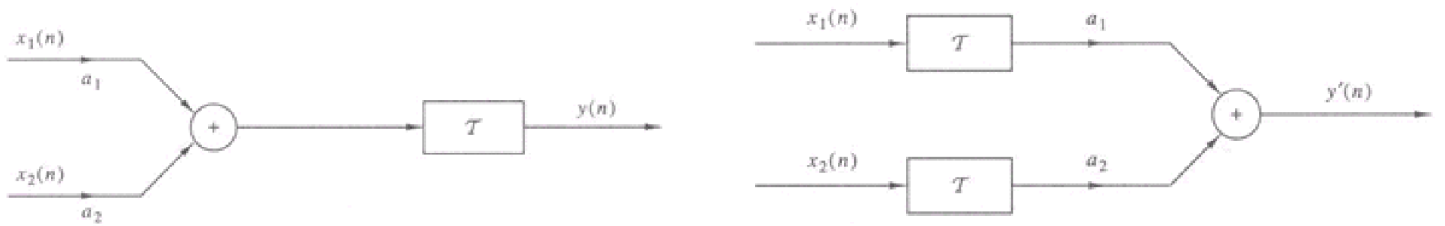


Figure 7. Linear System example

A simple example of a Discrete Time System is the MAF (M-point Moving Average System), which defines:

$$y(n) = \frac{1}{M} \sum_{k=0}^{M-1} x(n-k) \quad (4)$$

Let us have a signal $s(n)$, affected by the noise $d(n)$ for $n \geq 0$ as a result of the measurements:

$$x(n) = s(n) + d(n) \quad (5)$$

In the laboratory work it is proposed to reduce the effect of damage to the original signal by noise. To achieve this goal it is proposed to use the MAF System (Moving Average Filter).

- **Code and plots for signal modeling**

Research of random processes

Exercise 1.1

```
Ts=0.01;  
t=0:Ts:5;  
x1=rand(1,length(t));  
plot(t,x1), grid, set (gca,'FontName', ...  
'Arial Cyr','FontSize',16)  
title('Exemplu de utilizare a procedurii rand')  
xlabel('timpul(s)'),ylabel('functia y(t)'),grid
```

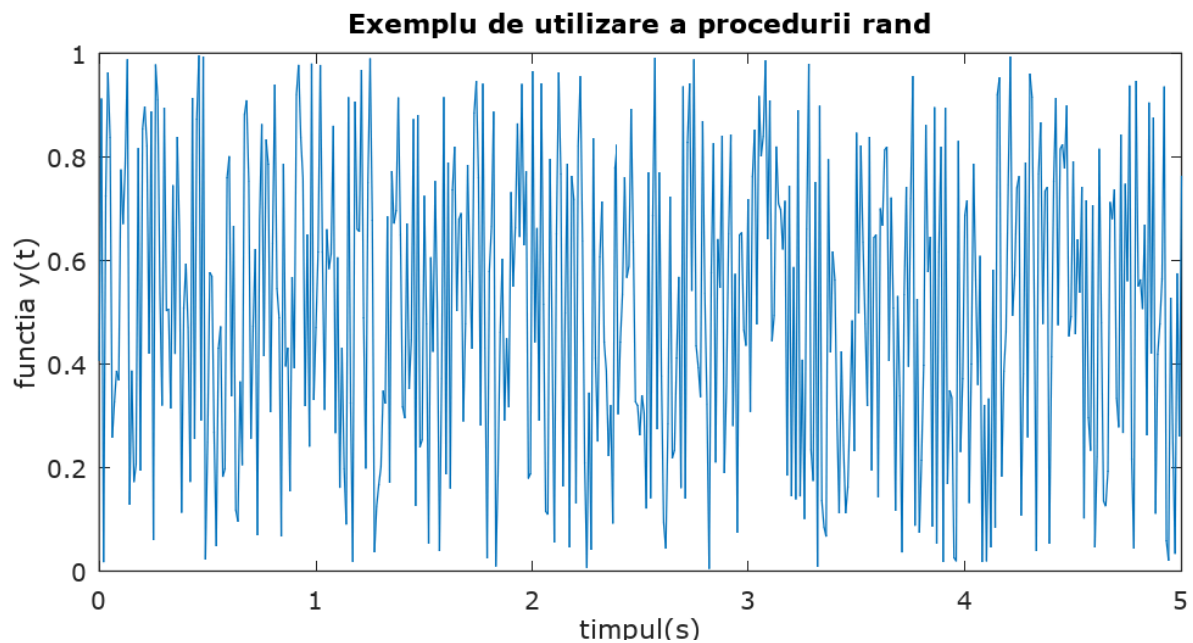


Figure 8: Diagram created using 'rand' function, with 'plot'

This shows a completely random signal with values between 0 and 1. The signal has no pattern and jumps rapidly between values, which is exactly what we'd expect from random noise.

Exercise 1.2

```
Ts = 0.01;  
t = 0 : Ts : 1 ;  
x1 = rand(1, length(t));  
hist(x1, t), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16)  
title('Example of using rand function');  
xlabel('time(s)'), ylabel('functiony(t)'), grid;
```

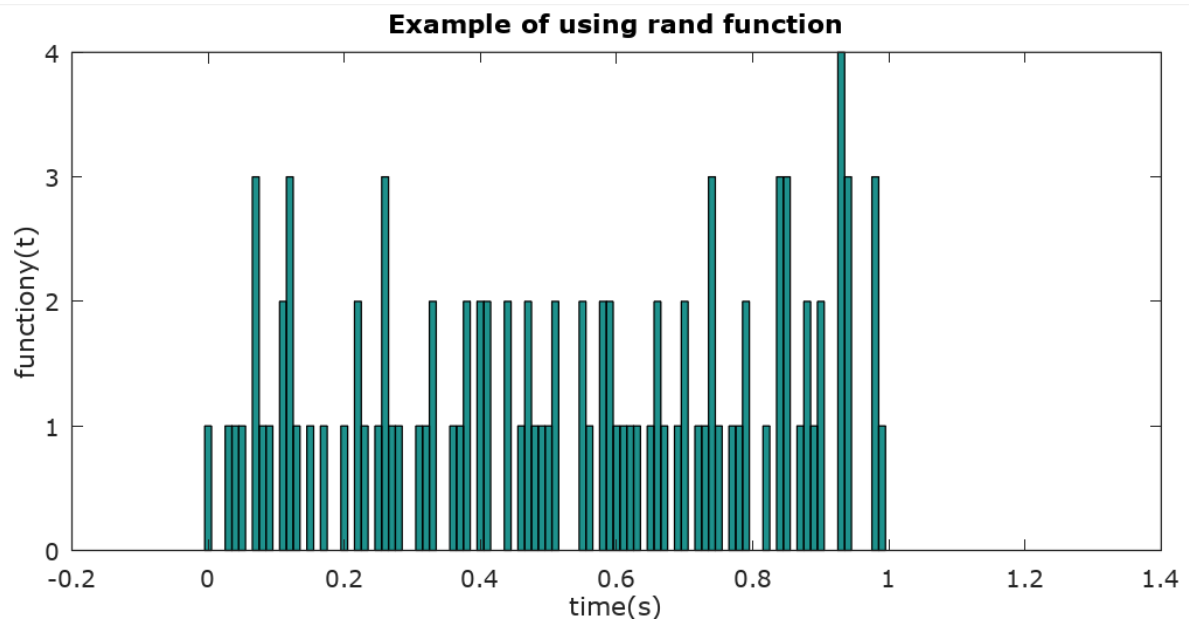


Figure 9: Diagram created using 'rand' function, with 'hist'

The histogram shows that our random values are fairly evenly distributed, confirming that we're getting a good uniform distribution of random numbers. This is important for simulating realistic noise.

Exercise 1.3

```
Ts = 0.001;  
t = 0 : Ts : 5;  
x2 = rand(1, length(t));  
plot(t, x2), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16)  
title('Example of using rand function');  
xlabel('time(s)'), ylabel('functiony(t)'), grid
```

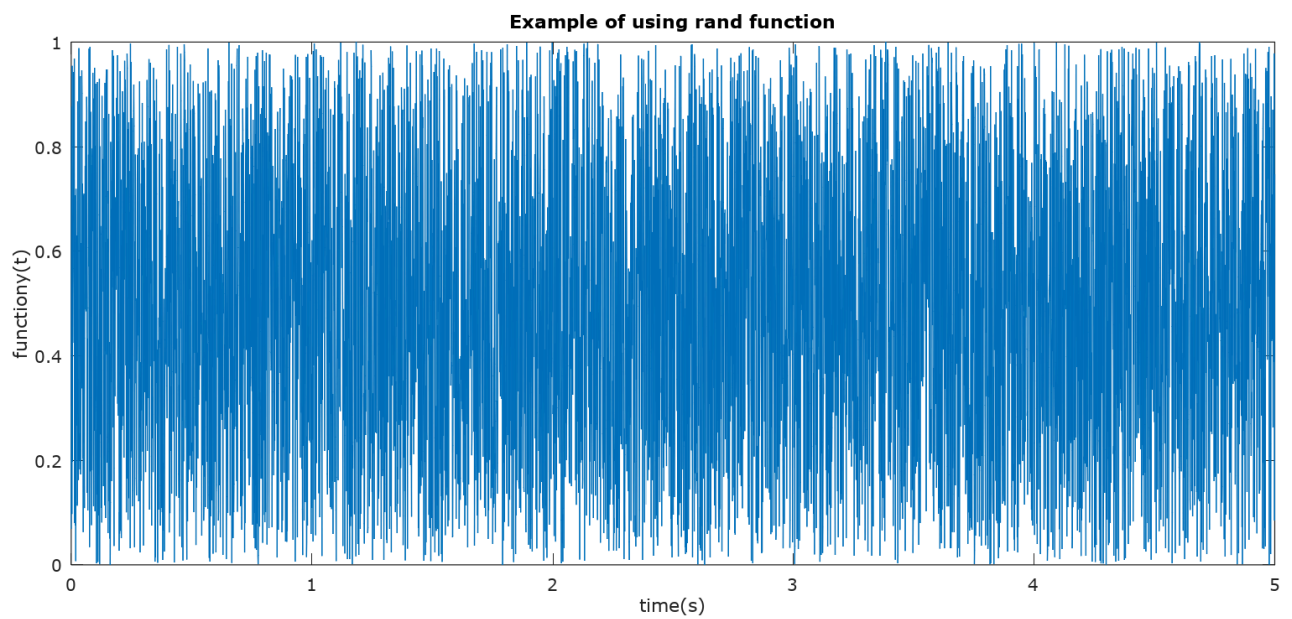


Figure 10: Diagram created of using 'rand' function, with 'plot',
 $T_s = 0.001$

Using a smaller time step ($T_s = 0.001$) gives us a more detailed random signal. You can see many more data points, making the noise look even more chaotic than before.

Exercise 1.4

```
Ts = 0.001;  
t = 0 : Ts : 5;  
x1 = rand(1, length(t));  
hist(x2, t), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16)  
title('Example of using rand function');  
xlabel('time(s)'), ylabel('functiony(t)'), grid
```

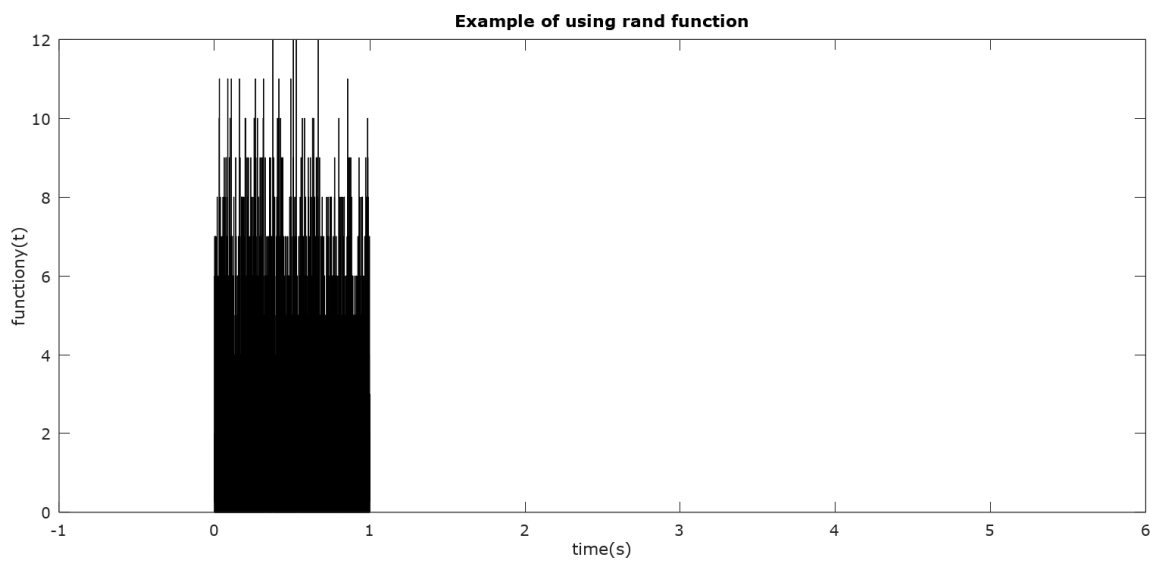


Figure 11: Diagram created using 'rand' function, with 'hist', $T_s = 0.001$

With the smaller time step, our histogram still shows uniform distribution, but now with many more samples. This gives us a smoother, more reliable picture of the randomness.

Exercise 1.5

```
Ts = 0.01;  
om0=2*pi;  
dz = 0.005;  
A = 1;  
oms = om0 * Ts;  
a(1) = 1 + 2 * dz * oms + oms^2;  
a(2) = -2 * (1 + dz * oms);  
a(3) = 1;  
b(1) = A * 2 * oms^2;  
t = 0 : Ts : 50;  
x1 = rand(1, length(t));  
y1 = filter(b, a, x1);  
plot(t, y1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);  
title('Noise filtering with a second orderfilter');  
xlabel('time(s)'), ylabel('functiony(t)'), grid;
```

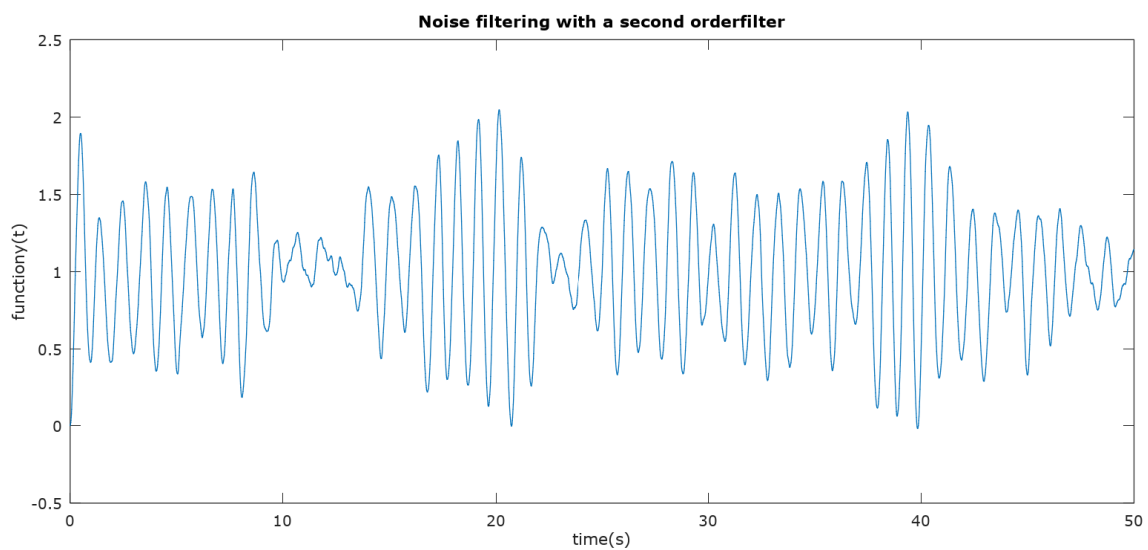


Figure 12: Diagram created for noise filtering with a second order filter, $T_s = 0.01$

Exercise 1.6

```
Ts = 0.001;  
om0=2*pi;  
dz = 0.005;
```

```

A = 1;
oms = om0 * Ts;
a(1) = 1 + 2 * dz * oms + oms^2;
a(2) = -2 * (1 + dz * oms);
a(3) = 1;
b(1) = A * 2 * oms^2;
t = 0 : Ts : 50;
x1 = rand(1, length(t));
y1 = filter(b, a, x1);
plot(t, y1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
title('Noise filtering with a second orderfilter');
xlabel('time(s)'), ylabel('functiony(t)'), grid

```

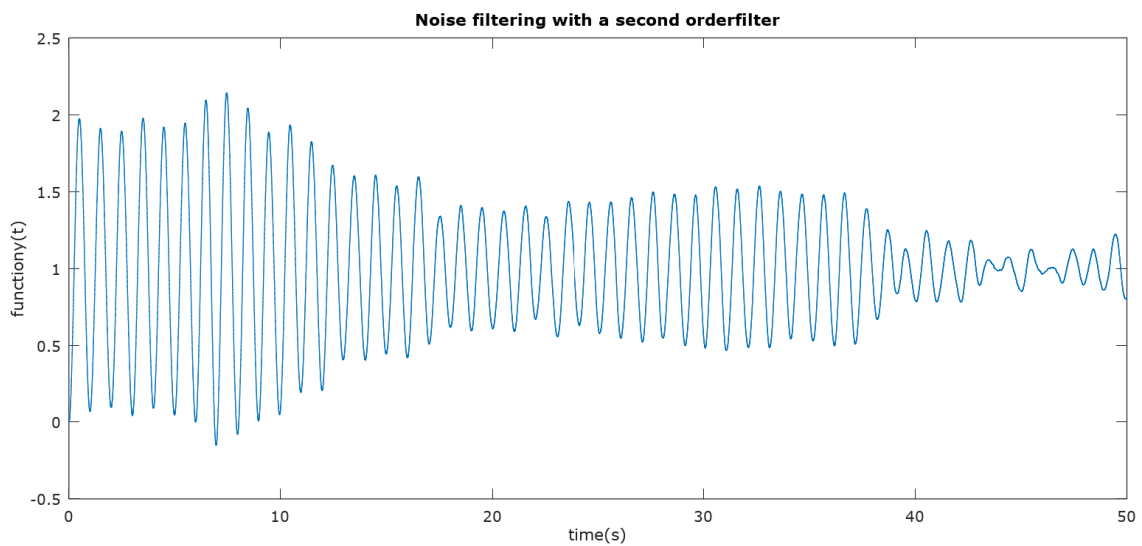


Figure 13: Diagram created for noise filtering with a second order filter, $T_s = 0.001$

These filtered outputs show how the second-order filter smooths out the random noise. The result is a much gentler signal that follows a more natural oscillating pattern, demonstrating the filter's ability to extract meaningful patterns from noise.

Filter out noise-affected signals using an MAF filter

Exercise 2.1

$R = 50;$

$m = 0:1:R-1;$

$s = 2*m.*(0.9.^m) ;$

`stem(m,s), grid, set (gca, 'FontName', 'Arial Cyr', 'FontSize', 16)`

`xlabel('Indecsul de timp n'); ylabel('Amplitudinea')`

`title('Semnalul original')`

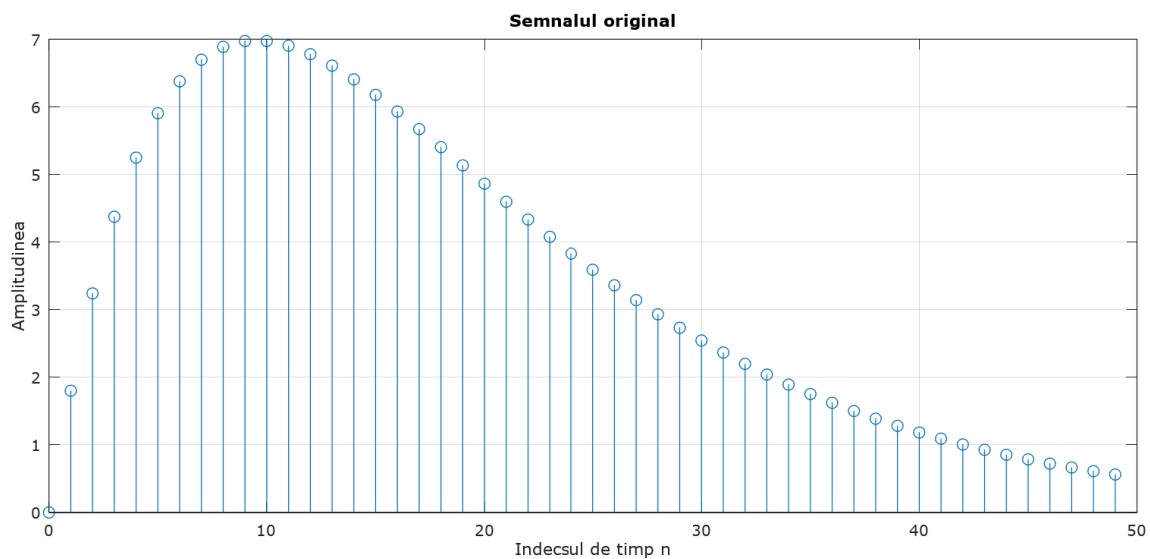


Figure 14: Diagram of unaffected by noise signal

This shows our clean test signal - an exponentially decaying oscillation. This will be our reference for comparing how well we can clean up noise-corrupted versions.

Exercise 2.2

$R = 50;$

$m = 0:1:R-1;$

$s = 2*m.*(0.9.^m);$

```

d=rand(1,length(m))-0.5;
stem(m,d), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
xlabel('Indecsul de timp n'); ylabel('Amplitudinea')
title('Semnalul original')

```

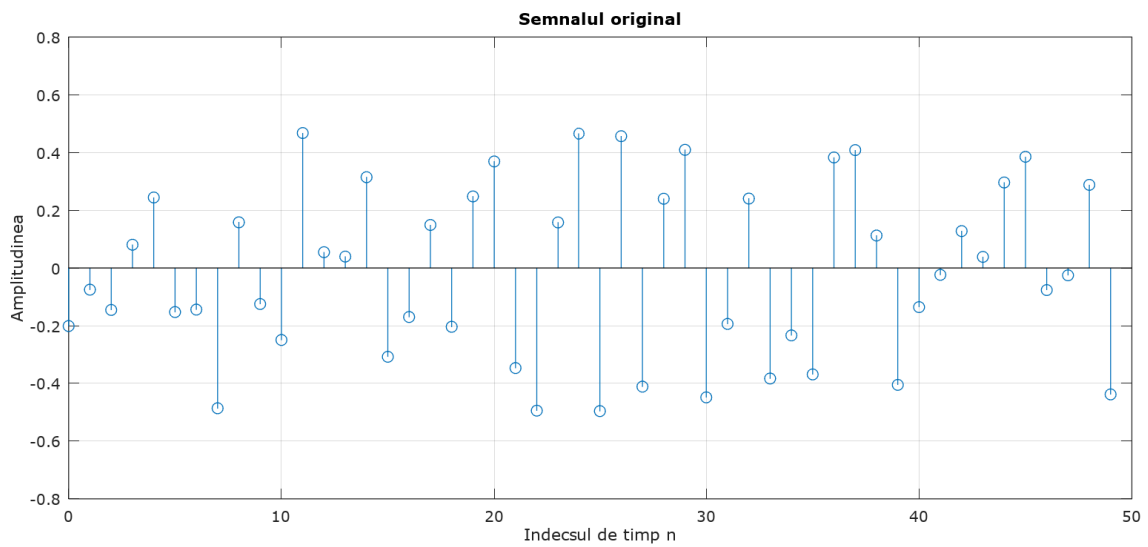


Figure 15: Random generated noise signal

Here's what our pure noise looks like - random fluctuations around zero that we'll be adding to our clean signal. This represents the kind of interference we want to filter out.

Exercise 2.3

```
R = 50;  
m = 0:1:R-1;  
s = 2*m.*(0.9.^m);  
d=rand(1,length(m))-0.5;  
plot(m, d, 'r', 'Linewidth', 1);  
hold on  
plot(m, s, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);  
xlabel('Indecsul de timp n'); ylabel('Amplitudinea')  
title('Semnalul original')  
legend('Sound Signal', 'Original Signal')
```

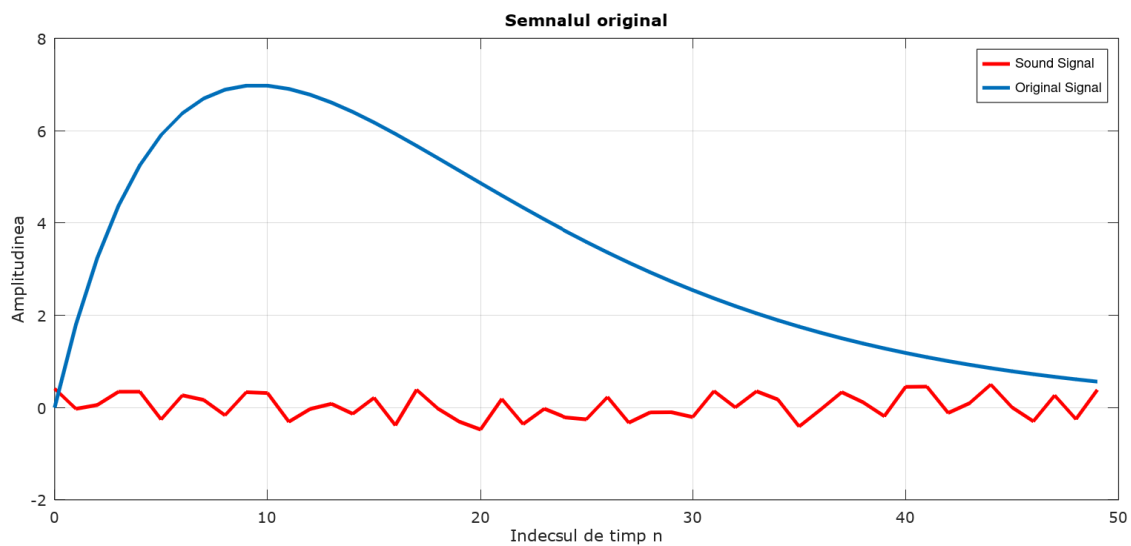


Figure 16: Original signal with sound signal

This comparison shows how different the random noise (red) is from our original signal. They have completely different patterns and characteristics.

Exercise 2.4

```
R = 50;  
m = 0:1:R-1;  
s = 2*m.*(0.9.^m);  
d=rand(1,length(m))-0.5;  
x = s+d;  
plot(m, s, 'r', 'Linewidth', 1);  
hold on  
plot(m, x, 'b', 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);  
xlabel('Indecul de timp n'); ylabel('Amplitudinea')  
title('Sum of original signal with noise signal')  
legend('Original signal', 'Noise Signal + Original signal')
```

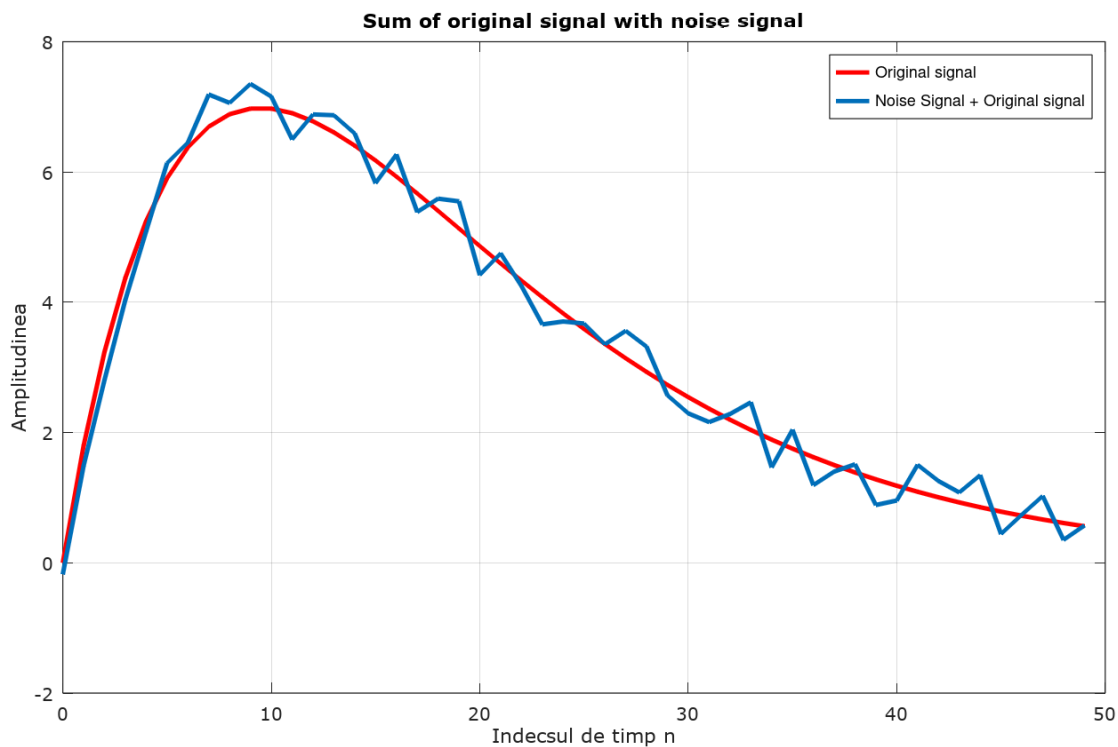


Figure 17: Sum of original signal with noise signal / Original signal

Now we can see how noise corrupts our original signal. The noisy signal (blue) follows the general shape of the original (red) but with lots of unwanted fluctuations.

Exercise 2.5

```
R = 50;
m = 0:1:R-1;
s = 2*m.*(0.9.^m);
d=rand(1,length(m))-0.5;
x = s+d;

M = 3;
b = ones(M, 1)/M;
y = filter(b, 1, x);
plot(m, y, 'g', 'Linewidth', 1);
hold on
plot(m, s, 'r', 'Linewidth', 1);
hold on
plot(m, x, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Indecsul de timp n'); ylabel('Amplitudinea')
title('Filtered Sound in Comparison')
legend('Filtered affected sound', 'Original signal', 'Noise Signal + Original signal')
```

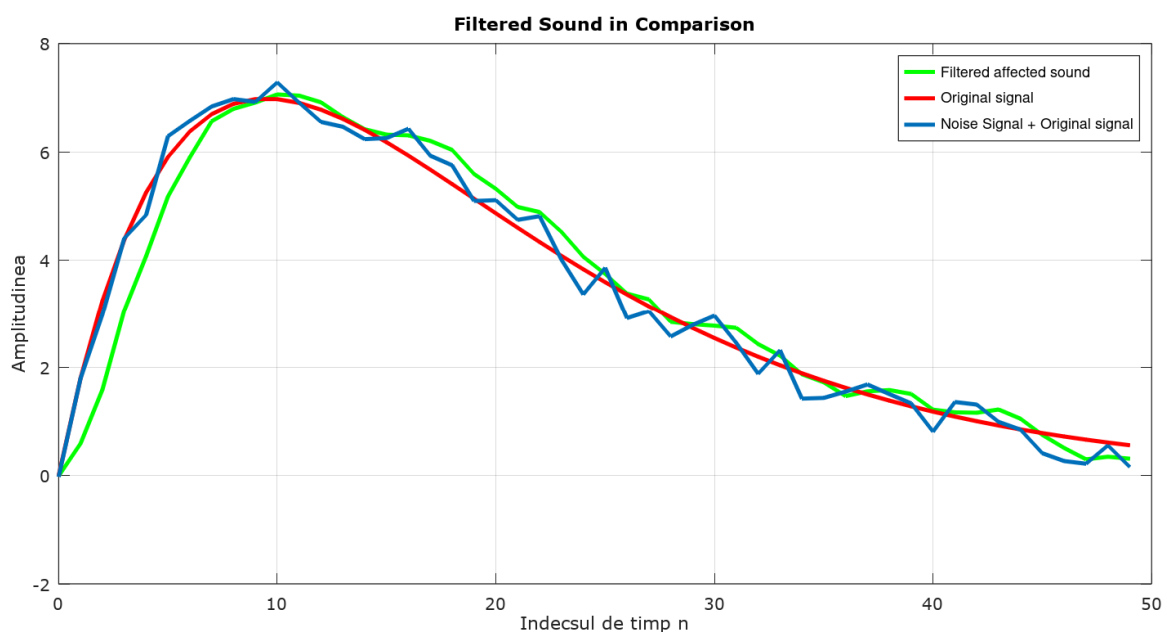


Figure 18: Sum of original signal with noise signal / Original signal / Filtered signal

The green line shows our first attempt at filtering with $M=3$. It's already doing a decent job of smoothing out the noise while preserving the main shape of our signal.

Exercise 2.6

```
R = 50;
m = 0:1:R-1;
s = 2*m.*(0.9.^m);
d=rand(1,length(m))-0.5;
x = s+d;

M1 = 3;
b1 = ones(M1, 1)/M1;
y1 = filter(b1, 1, x);

M2 = 5;
b2 = ones(M2, 1)/M2;
y2 = filter(b2, 1, x);

M3 = 10;
b3 = ones(M3, 1)/M3;
y3 = filter(b3, 1, x);

plot(m, s, 'm', 'Linewidth', 1);
hold on

plot(m, y1, 'g', 'Linewidth', 1);
hold on

plot(m, y2, 'r', 'Linewidth', 1);
hold on

plot(m, y3, 'y', 'Linewidth', 1);
hold on

plot(m, x, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Indecul de timp n'); ylabel('Amplitudinea')
title('Comparison of different filters');
legend('Original Signal', 'Filtered affected signal, M = 3', 'Filtered affected signal, M = 5',
'Filteredaffectedsignal, M = 10', 'Original affected signal')
```

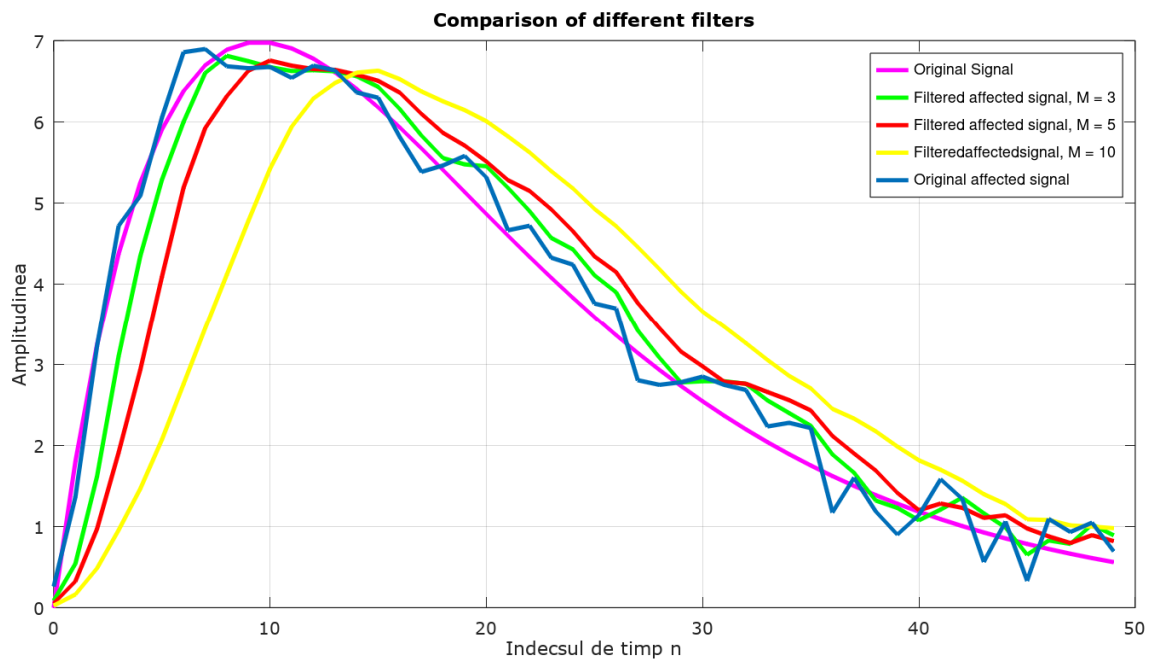


Figure 19: Filtered signals comparison

This interesting comparison shows how different filter lengths ($M=3,5,10$) affect the result. Larger M values give smoother output but might lose some signal detail - it's always a trade-off.

Exercise 2.7

```
R = 1;  
m = 0:0.001:R;  
s = 2*sawtooth(3*pi*m+pi/6);  
d=rand(1,length(m))-0.5;  
x = s+d;  
M = 20;  
b = ones(M, 1)/M;  
y = filter(b, 1, x);  
plot(m, y, 'g', 'Linewidth', 1);  
hold on  
plot(m, s, 'r', 'Linewidth', 1);  
hold on  
plot(m, x, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);  
xlabel('Indecsul de timp n'); ylabel('Amplitudinea')  
title('Filtered Sound in Comparison with changed sound')  
legend('Filtered affected sound', 'Original signal', 'Noise Signal + Original signal')
```

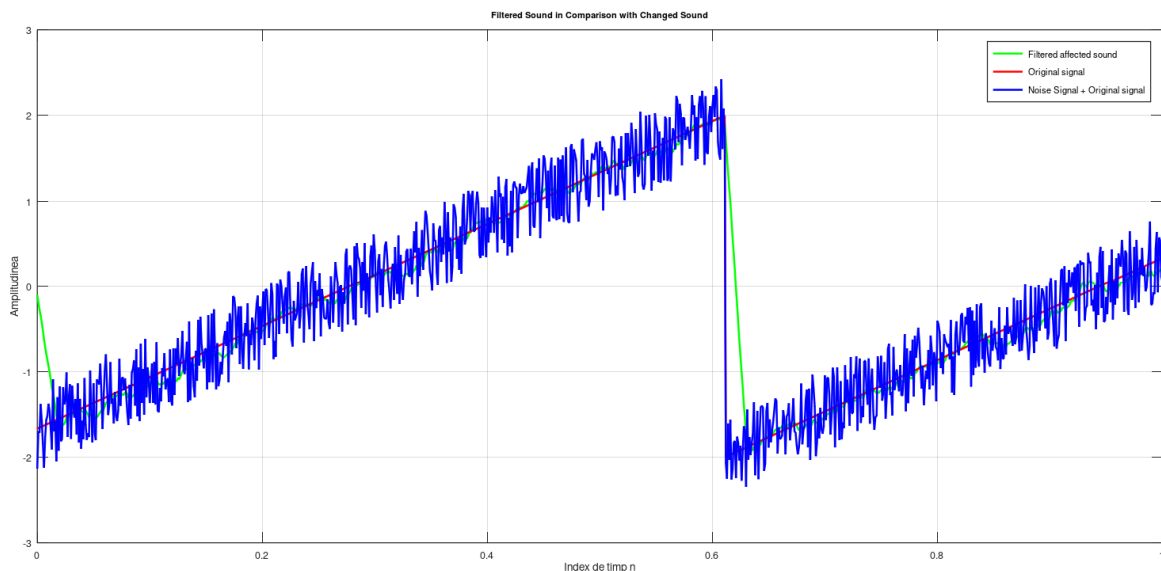


Figure 20: Sum of original signal with noise signal / Original signal / New
Filtered signal Exercise 2.7

Sinusoidal function:

```
R = 1;  
m = 0:0.001:R;  
s = 2*sin(3*pi*m+pi/6);  
d=rand(1,length(m))-0.5;  
x = s+d;  
M = 20;  
b = ones(M, 1)/M;  
y = filter(b, 1, x);  
plot(m, y, 'g', 'Linewidth', 1);  
hold on  
plot(m, s, 'r', 'Linewidth', 1);hold on  
plot(m, x, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);  
xlabel('Indecsul de timp n'); ylabel('Amplitudinea')  
title('Sinusoid of Filtered Sound in Comparison with changed sound')  
legend('Filtered affected sound', 'Original signal', 'Noise Signal + Original signal')
```

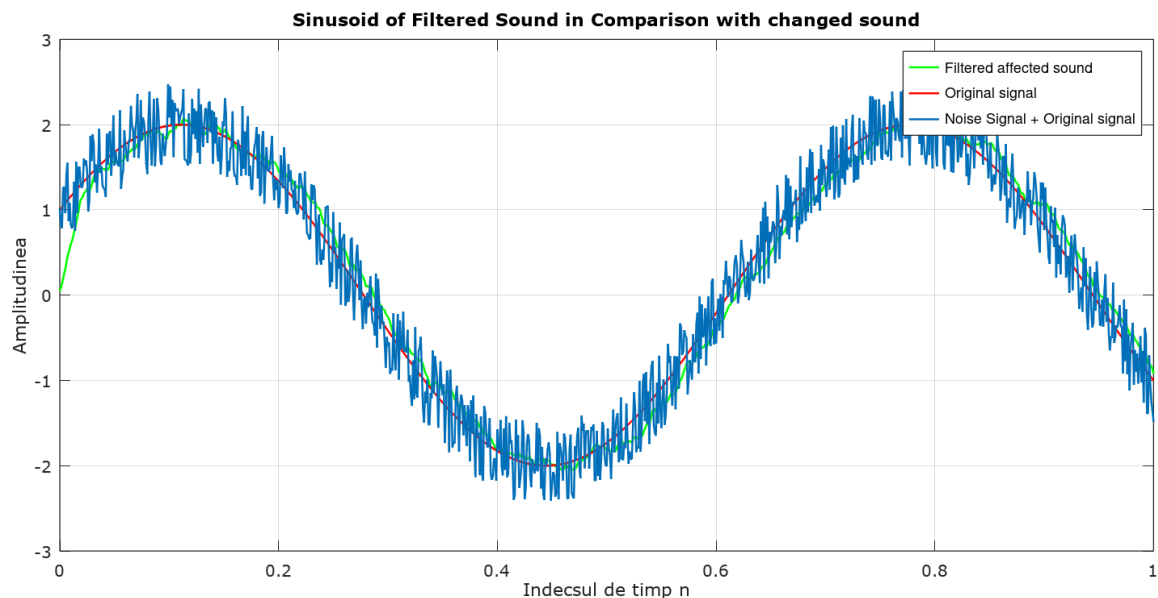


Figure 21: Sinusoid of Sum of original signal with noise signal / Original signal / New Filtered signal Exercise 2.7

Testing with different filter lengths ($M=20, 50, 100$) on both sawtooth and sinusoidal signals shows that longer filters give smoother results but can lag behind the original signal more.

Exercise 2.8

```
R = 1;
```

```
m = 0:0.001:R;
```

```
s = 2*sawtooth(3*pi*m+pi/6)
```

```
d=rand(1,length(m))-0.5;
```

```
x = s+d;
```

```
M1 = 20;
```

```
b1 = ones(M1, 1)/M1;
```

```
y1 = filter(b1, 1, x);
```

```
M2 = 50;
```

```
b2 = ones(M2, 1)/M2;
```

```
y2 = filter(b2, 1, x);
```

```
M3 = 100;
```

```
b3 = ones(M3, 1)/M3;
```

```
y3 = filter(b3, 1, x);
```

```
plot(m, y1, 'g', 'Linewidth', 1);
```

```
hold on
```

```
plot(m, y2, 'r', 'Linewidth', 1);
```

```
hold on
```

```
plot(m, y3, 'y', 'Linewidth', 1);
```

```
hold on
```

```
plot(m, s, 'r', 'Linewidth', 1);
```

```
hold on
```

```
plot(m, x, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
```

```
xlabel('Indecul de timp n'); ylabel('Amplitudinea')
```

```
title('Comparison of Filtered sounds with sawtooth')
```

legend('Filtered affected signal, M = 20', 'Filtered affected signal, M = 50',
 'Filteredaffectedsignal, M = 100', 'Original affected signal', 'Original signal + noise signal')

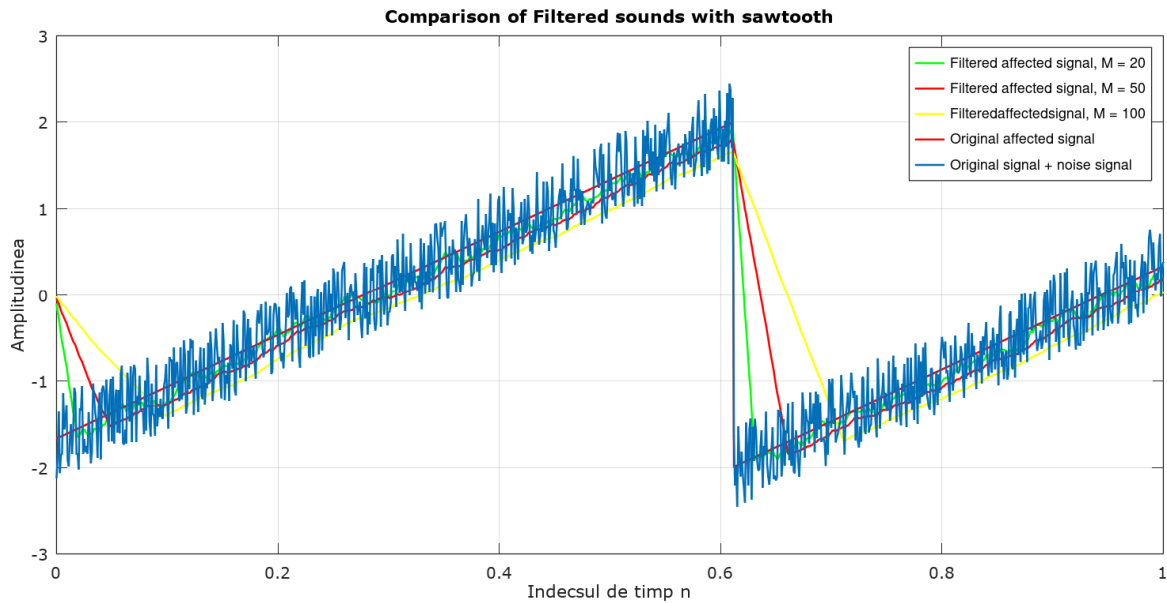


Figure 22: Comparing filtered new signals Exercise 2.8

Sinusoidal function:

```

R = 1;
m = 0:0.001:R;
s = 2*sin(3*pi*m+pi/6)
d=rand(1,length(m))-0.5;
x = s+d;
M1 = 20;
b1 = ones(M1, 1)/M1;
y1 = filter(b1, 1, x);
M2 = 50;
b2 = ones(M2, 1)/M2;
y2 = filter(b2, 1, x);
M3 = 100;
b3 = ones(M3, 1)/M3;
  
```

```

y3 = filter(b3, 1, x);
plot(m, y1, 'g', 'Linewidth', 1);
hold on
plot(m, y2, 'r', 'Linewidth', 1);
hold on
plot(m, y3, 'y', 'Linewidth', 1);
hold on
plot(m, s, 'r', 'Linewidth', 1);
hold on
plot(m, x, 'Linewidth', 1), grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Indecsul de timp n'); ylabel('Amplitudinea')
title('Comparison of Filtered sounds with sawtooth')
legend('Filtered affected signal, M = 20', 'Filtered affected signal, M = 50',
'Filteredaffectedsignal, M = 100', 'Original affected signal', 'Original signal +
noise signal')

```

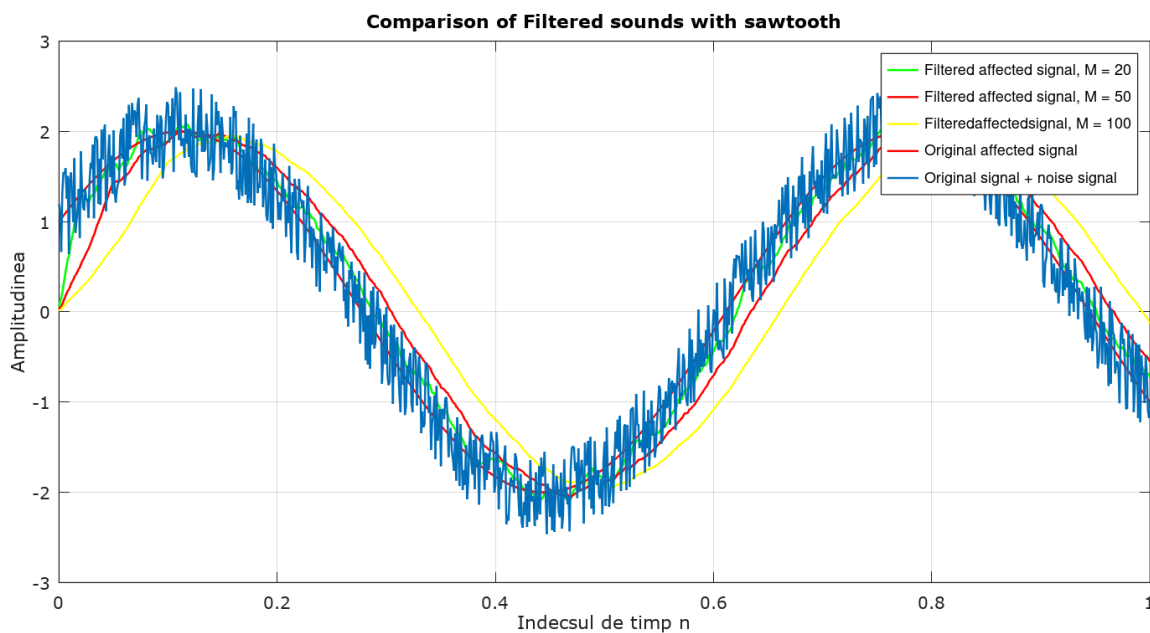


Figure 23: Comparing filtered new signals in a sinusoidal graphic Exercise 2.8

Testing with different filter lengths ($M=20,50,100$) on both sawtooth and sinusoidal signals shows that longer filters give smoother results but can lag behind the original signal more.

Research of LMS and RLS algorithms

Because I use GNU Octave steps in instruction manual are not applicable, therefore I implemented LMS and RLS myself in this editor

pkg load signal

% Parameters

N = 500; % Number of samples

mu = 0.01; % Step size (learning rate)

M = 4; % Number of filter taps (order)

x = randn(N, 1); % White noise input

d = filter([1, 0.5, -0.3, 0.2], 1, x) + 0.1 * randn(N, 1);

w = zeros(M, 1); % Filter coefficients

y = zeros(N, 1); % Filter output

e = zeros(N, 1); % Error signal

for n = M:N

 x_n = x(n:-1:n-M+1); % Input vector

 y(n) = w' * x_n; % Filter output

 e(n) = d(n) - y(n); % Error signal

 w = w + mu * x_n * e(n); % Update weights

end

figure;

subplot(2,1,1);

plot(d, 'b', 'DisplayName', 'Desired Signal');

hold on;

plot(y, 'r', 'DisplayName', 'LMS Output');

```

legend;
title('LMS Adaptive Filter Output');
subplot(2,1,2);
plot(e, 'k');
title('Error Signal');
xlabel('Sample Index');
ylabel('Error');

```

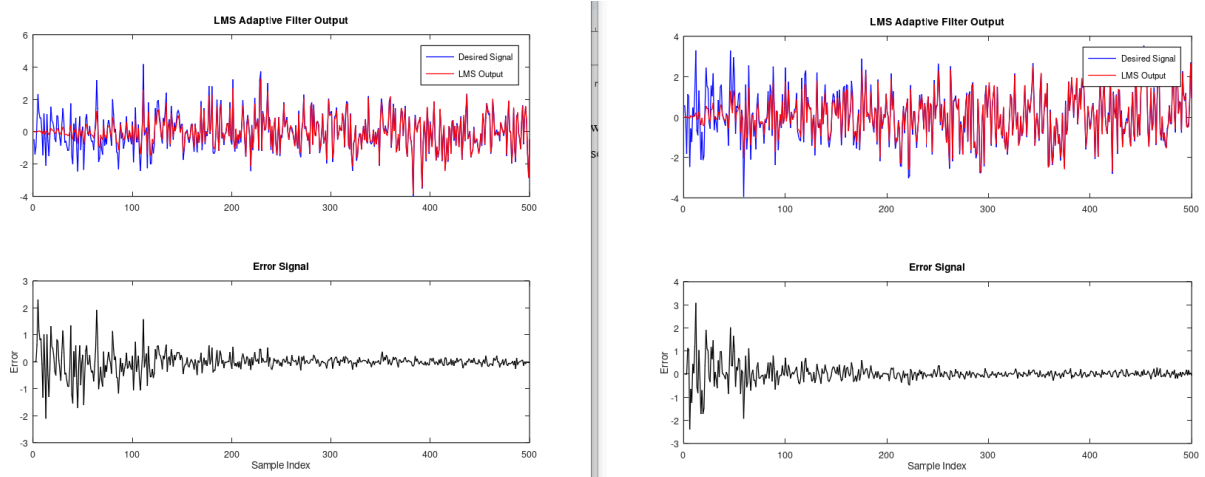


Figure 24: LMS demo

```

lambda = 0.99;    % Forgetting factor

delta = 1;        % Initialization factor

M = 4;            % Number of filter taps (order)

w = zeros(M, 1);  % Filter coefficients

P = eye(M) / delta; % Inverse correlation matrix

y = zeros(N, 1);  % Filter output

e = zeros(N, 1);  % Error signal

for n = M:N

    x_n = x(n:-1:n-M+1); % Input vector

    k = (P * x_n) / (lambda + x_n' * P * x_n); % Gain vector

    y(n) = w' * x_n;      % Filter output

    e(n) = d(n) - y(n);   % Error signal

```

```

w = w + k * e(n);    % Update weights

P = (P - k * x_n' * P) / lambda; % Update inverse correlation matrix

end

figure;

subplot(2,1,1);

plot(d, 'b', 'DisplayName', 'Desired Signal');

hold on;

plot(y, 'r', 'DisplayName', 'RLS Output');

legend;

title('RLS Adaptive Filter Output');

subplot(2,1,2);

plot(e, 'k');

title('Error Signal');

xlabel('Sample Index'); ylabel('Error');

```

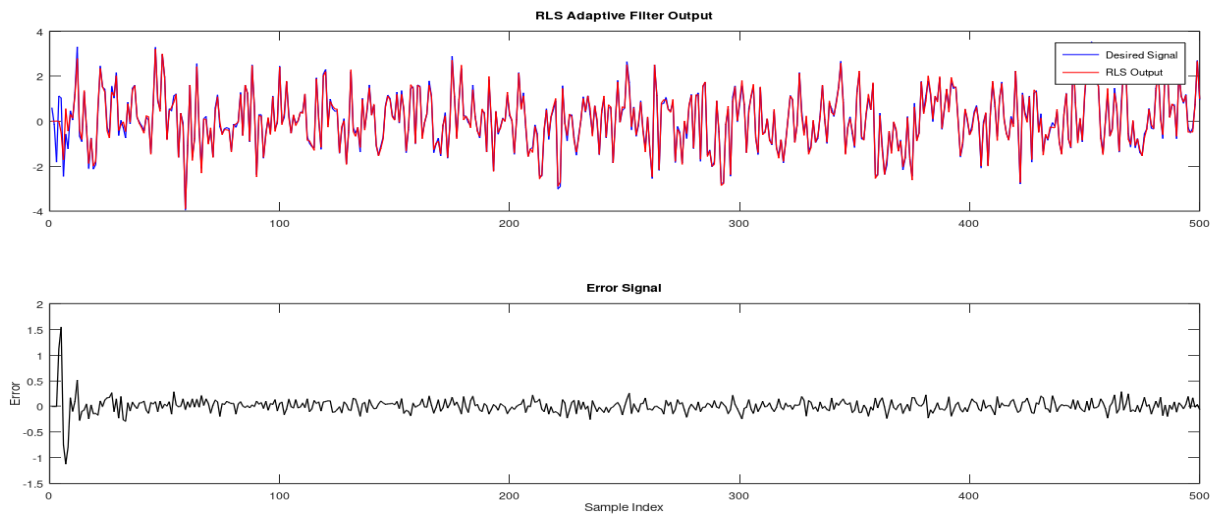


Figure 25: RLS demo

The LMS and RLS adaptive filters both do a good job of tracking the desired signal, but RLS converges faster and shows smaller error overall. This demonstrates why RLS is often preferred when computational resources allow for it.

• Conclusion

This laboratory work provided valuable hands-on experience with discrete-time systems, noise generation, and filtering techniques. Through our experiments, we made several important observations:

1. Random Noise Characteristics:
 - We successfully generated and visualized random noise using different time steps
 - The histograms confirmed uniform distribution of our noise signals
 - Smaller time steps ($T_s = 0.001$) provided more detailed noise representations
2. Filtering Performance:
 - The M-point Moving Average Filter proved effective at noise reduction
 - Larger filter lengths (M) provided smoother outputs but introduced more delay
 - There's a clear trade-off between noise reduction and signal preservation
3. Signal Type Impact:
 - The filter performed well on different signal types (exponential decay, sawtooth, sinusoidal)
 - Each signal type presented unique challenges for filtering
 - The original signal characteristics were generally well-preserved after filtering
4. Adaptive Filtering:
 - Both LMS and RLS algorithms demonstrated effective noise reduction
 - RLS showed faster convergence and better overall performance
 - The adaptive nature of these algorithms made them robust to varying signal conditions
5. Practical Implications:
 - Understanding filter parameter selection (like filter length M) is crucial
 - Real-world applications need to balance noise reduction with signal preservation
 - Different filtering approaches may be optimal for different signal types

This lab work reinforced the importance of signal processing in real-world applications and demonstrated how different filtering techniques can be applied to improve signal quality. The practical experience gained with MATLAB/Octave will be valuable for future signal processing tasks.

- **Control questions**

Block Diagram Representations

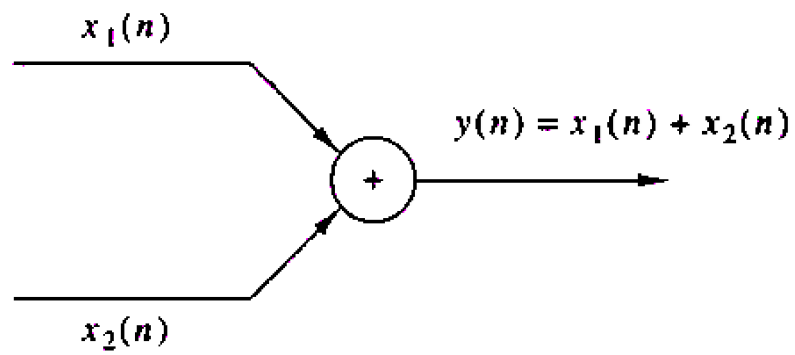


Figure 26: Adder diagram

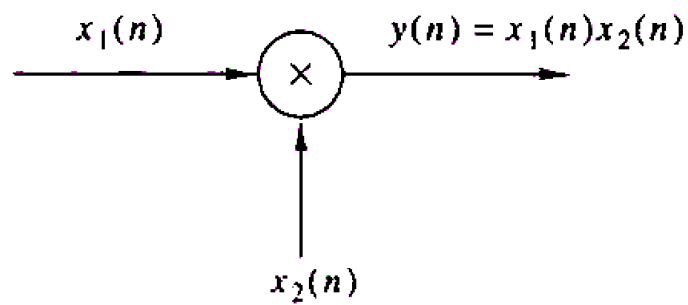


Figure 27: Multiplier diagram

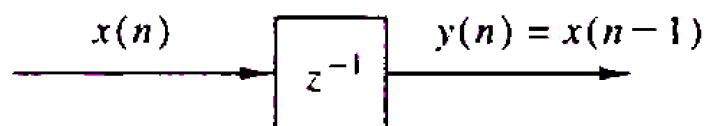


Figure 28: Unit delay diagram

Static vs. Dynamic Systems

- **Static (Memoryless) System:** The output at any given time depends only on the input at that exact moment.
- **Dynamic System:** The output depends on past or future inputs, meaning it has memory.

Time-Variant vs. Time-Invariant Systems

- **Time-Invariant System:** The system's behavior remains unchanged over time; if the input is shifted, the output shifts accordingly.
- **Time-Variant System:** The system's characteristics change over time, meaning a shifted input does not necessarily produce a corresponding shifted output.

Linear vs. Nonlinear Systems

- **Linear System:** Follows the superposition principle—scaling and summing inputs lead to the same scaling and summing of outputs.
- **Nonlinear System:** Does not obey superposition; outputs may behave unpredictably based on input changes.

Recursive vs. Non-Recursive Systems

- **Recursive System:** The current output depends on past outputs and inputs.
- **Non-Recursive System:** The output depends only on present and past inputs, not on past outputs.

Smoothing Filter Algorithm

Smoothing reduces noise in a signal by averaging data points. Peaks due to noise are reduced, and dips are raised, creating a smoother trend.

- **LMS (Least Mean Square):** An adaptive filter that adjusts its coefficients to minimize the difference between the desired and actual signal using a stochastic approach.
- **RLS (Recursive Least Squares):** A more complex adaptive filter that rapidly converges to optimal coefficients using a recursive method, but at a higher computational cost.