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

Laboratory work nr. 1

Discrete Time Systems.

Noise generation and its distortion.

A elaborat:	**Cristian Brinza** st. gr. FAF-212
A verificat:	**Railean Serghei** lect. univ

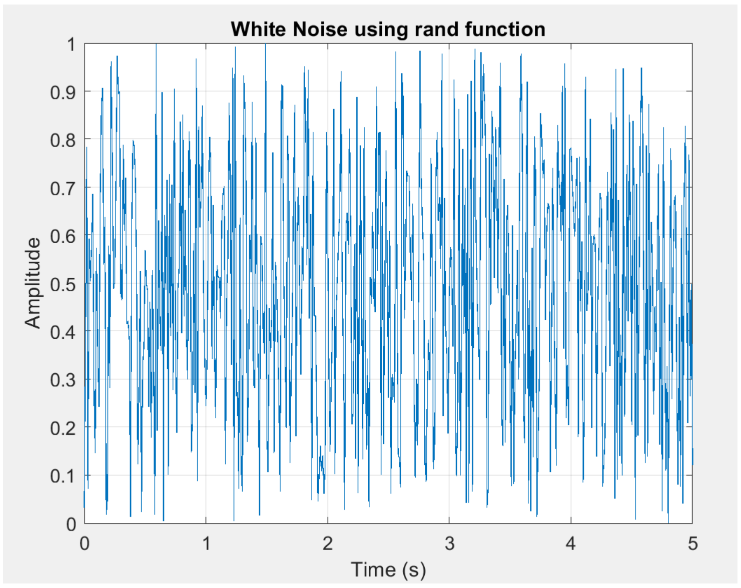
Chișinău, 2024

The purpose of the work: Noise generation and its filtering using the Discrete Time System "M-point Moving Average System"

1. Random process research

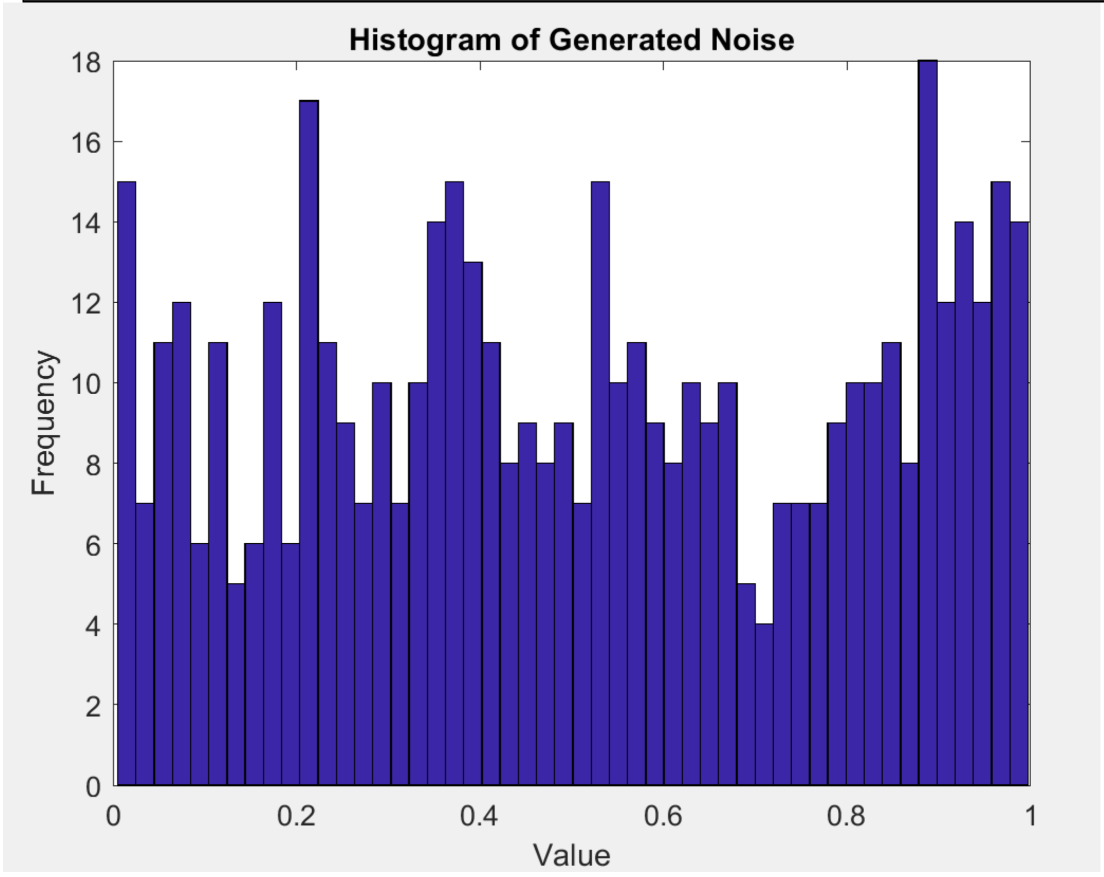
1.1."White" noise with Gaussian representation is made by the rand procedure. Generate a random process as follows:

```
	Ts = 0.01; % Sampling time
	t = 0:Ts:5; % Time vector from 0 to 5 seconds
	x1 = rand(1, length(t)); % Generating white noise with
	uniform distribution
	figure; % Open a new figure window
	plot(t, x1); % Plot the noise
	grid on; % Enable grid
	title('White Noise using rand function'); % Title of the
	plot
	xlabel('Time (s)'); % X-axis label
ylabel('Amplitude'); % Y-axis label
```



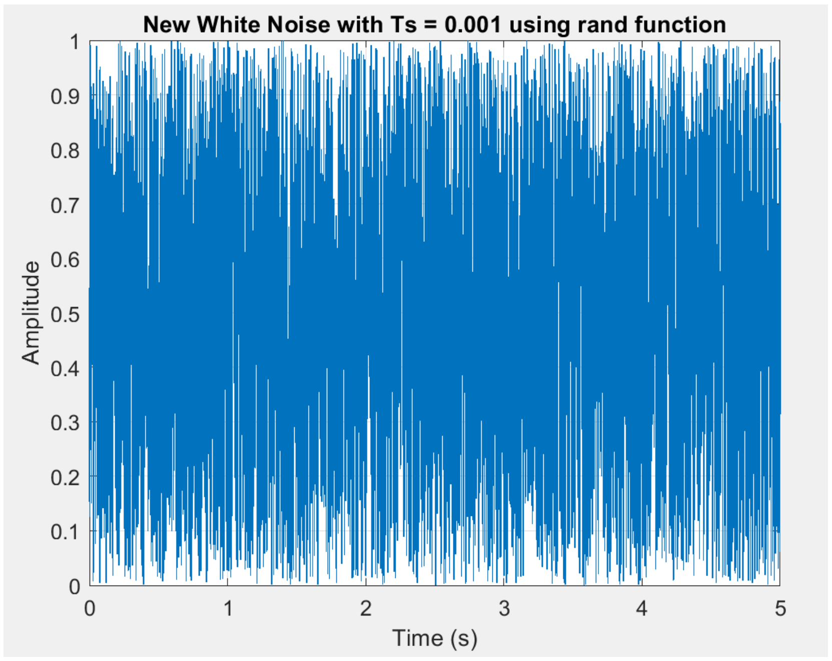
1.2. **P**lot the histogram of the generated noise, replacing the plot function with hist (beforehand change time to 1 and change to place t,x1).

```
      Ts = 0.01; % Sampling time
      t = 0:Ts:5; % Time vector from 0 to 5 seconds
      x1 = rand(1, length(t)); % Generating white noise with
      uniform distribution
      figure; % Open a new figure window
      hist(x1, 50); % Plot histogram of the noise with 50 bins
      title('Histogram of Generated Noise'); % Title of the
      histogram
      xlabel('Value'); % X-axis label
ylabel('Frequency'); % Y-axis label
```



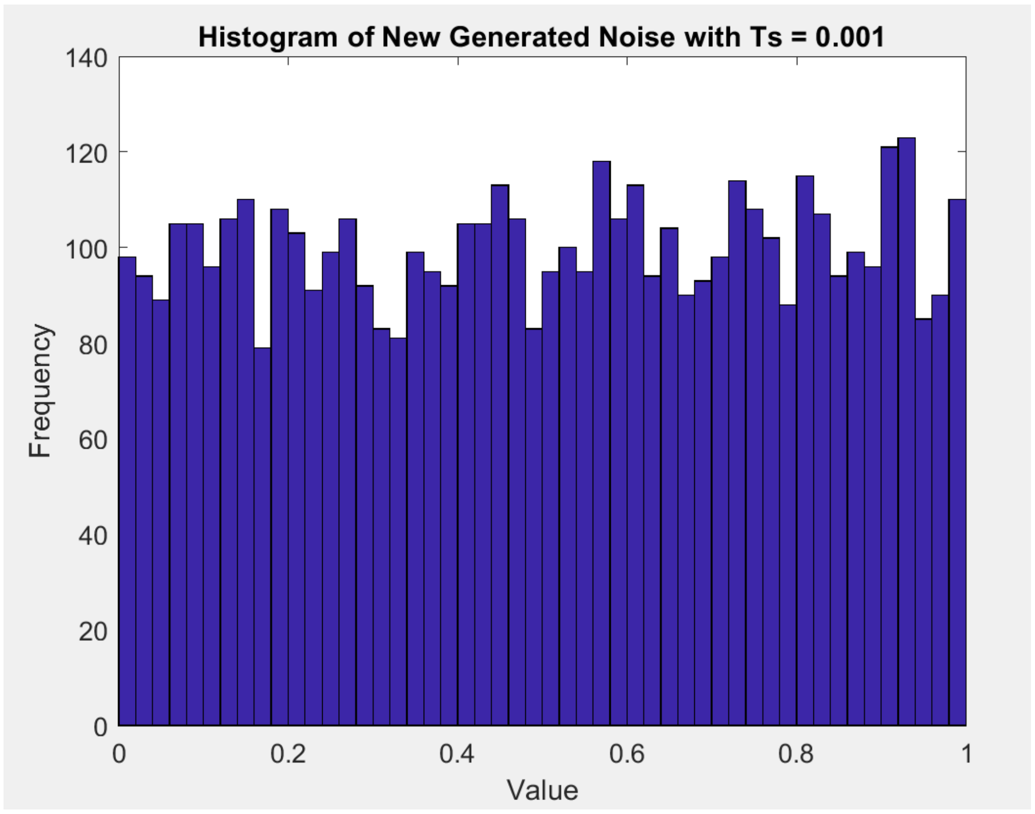
1.3.Repeat p. 1.1 for Ts=0.001 and generate a new noise x2

```
      Ts = 0.001; % New sampling time
      t = 0:Ts:5; % New time vector from 0 to 5 seconds
      x2 = rand(1, length(t)); % Generating new white noise with
      uniform distribution
      figure; % Open a new figure window
      plot(t, x2); % Plot the new noise
      grid on; % Enable grid
      title('New White Noise with Ts = 0.001 using rand
      function'); % Title of the plot
      xlabel('Time (s)'); % X-axis label
ylabel('Amplitude'); % Y-axis label
```



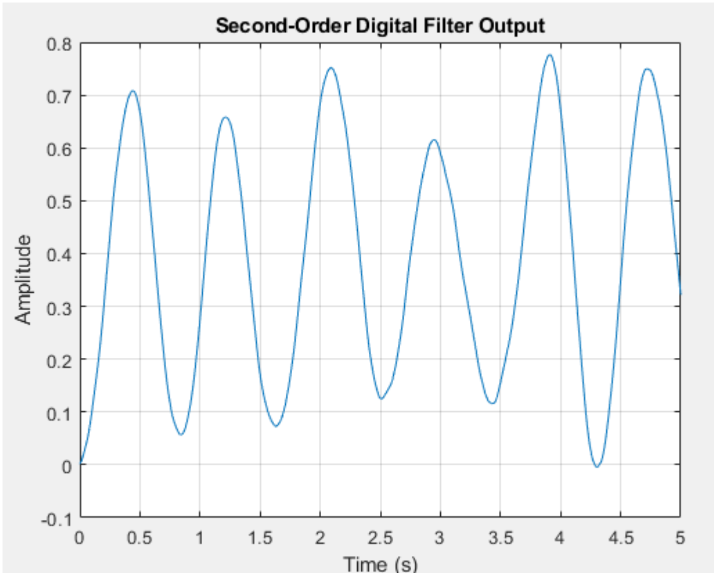
1.4.Represent the histogram of the noise generated x2 in p. 1.3.

```
      Ts = 0.001; % New sampling time
      t = 0:Ts:5; % New time vector from 0 to 5 seconds
      x2 = rand(1, length(t)); % Generating new white noise with
      uniform distribution
      figure; % Open a new figure window
      hist(x2, 50); % Plot histogram of the new noise with 50
      bins
      title('Histogram of New Generated Noise with Ts = 0.001');
      % Title of the histogram
      xlabel('Value'); % X-axis label
ylabel('Frequency'); % Y-axis label
```



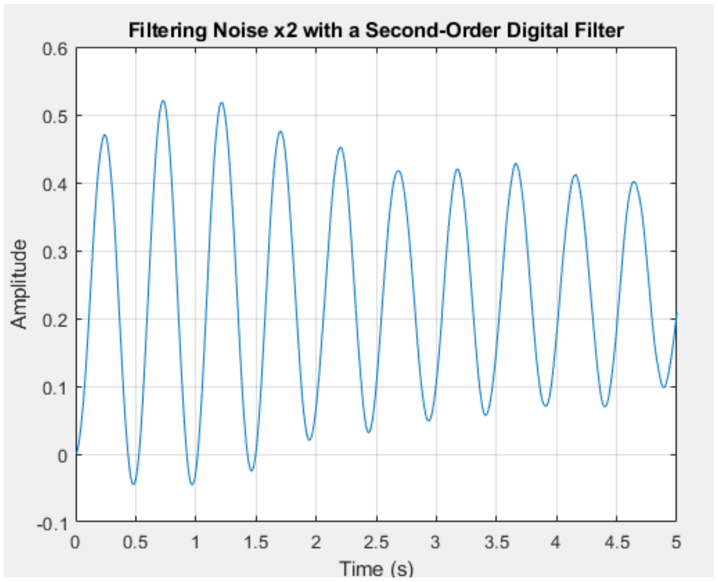
1.5. Design a second-order digital filter with the frequency of natural oscillations 1 Hz, pass through this filter the signal x1 and display the signal at the filter output:

```
Ts = 0.01; % Sampling time
om0 = 2 * pi; % Natural frequency (assumed to be 2*pi for
1 Hz)
dz = 0.005; % Damping ratio
A = 1; % Amplitude
oms = om0 * Ts; % Scaled omega by sampling time
a = [1 + 2*dz*oms + oms^2, -2*(1 - dz*oms), 1]; %
Denominator coefficients
b = [A*oms^2]; % Numerator coefficients, corrected for
filter function
t = 0:Ts:5; % Time vector
x1 = rand(1, length(t)); % Generating white noise
y1 = filter(b, a, x1); % Filtering the noise
figure; % Open a new figure window
plot(t, y1); % Plot the filtered signal
grid on; % Enable grid
title('Second-Order Digital Filter Output'); % Title of
the plot
xlabel('Time (s)'); % X-axis label
ylabel('Amplitude'); % Y-axis label
```



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

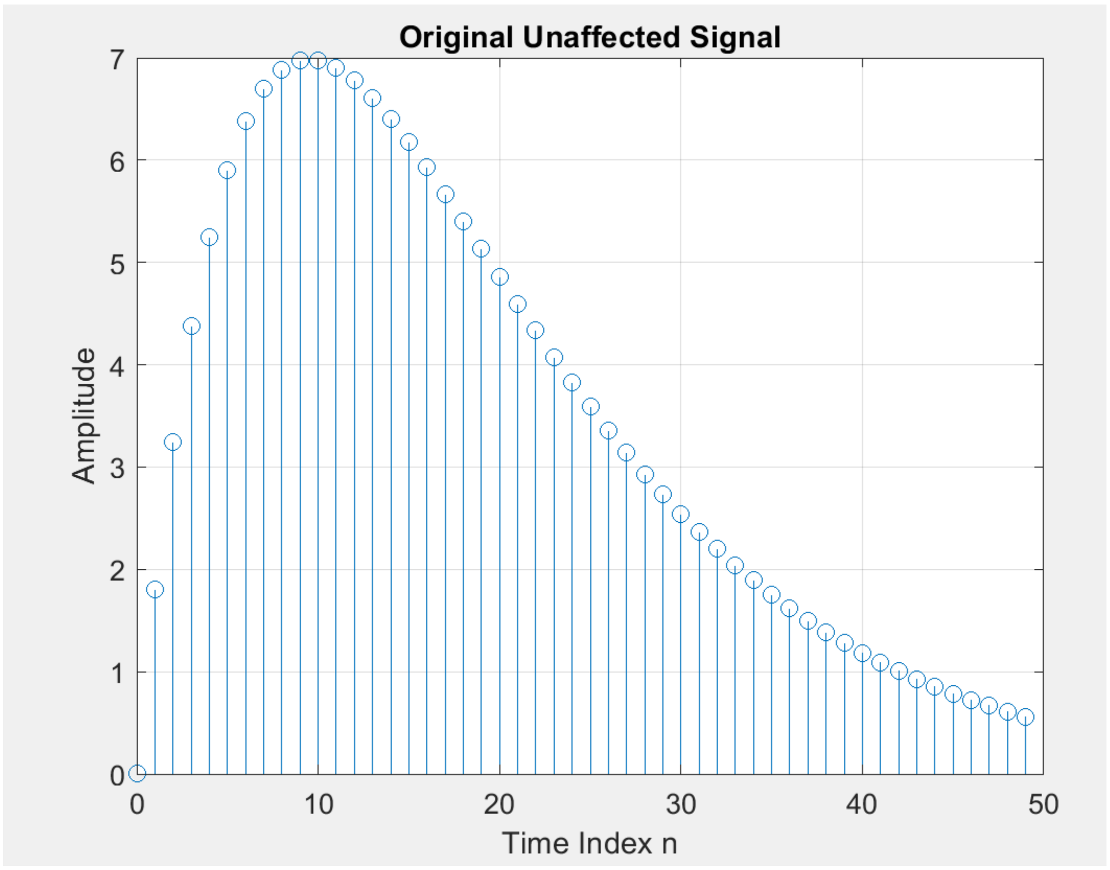
```
      Ts = 0.001; % New sampling time
      om0 = 2 * pi; % Natural frequency (assumed to be 2*pi for
      1 Hz)
      dz = 0.005; % Damping ratio
      A = 1; % Amplitude
      oms = om0 * Ts; % Scaled omega by sampling time
      a = [1 + 2*dz*oms + oms^2, -2*(1 - dz*oms), 1]; %
      Denominator coefficients
      b = [A*2*oms^2]; % Numerator coefficients, corrected for
      filter function
      t = 0:Ts:5; % New time vector
      x2 = rand(1, length(t)); % Generating new white noise with
      updated Ts
      y2 = filter(b, a, x2); % Filtering the new noise
      figure; % Open a new figure window
      plot(t, y2); % Plot the filtered signal
      grid on; % Enable grid
      title('Filtering Noise x2 with a Second-Order Digital
      Filter'); % Title of the plot
      xlabel('Time (s)'); % X-axis label
ylabel('Amplitude'); % Y-axis label
```



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

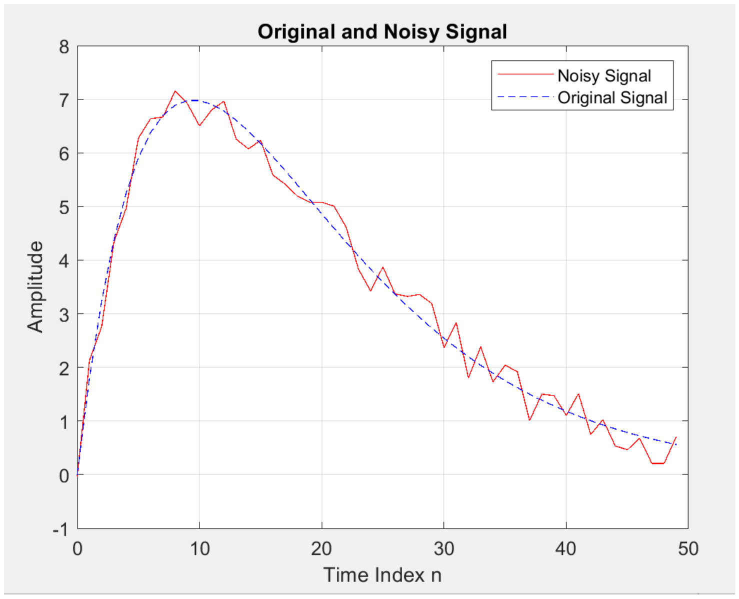
2.1. Generate an original signal affected by noise s(n):

```
      R = 50;
      m = 0:1:R-1;
      s = 2*m.*(0.9.^m);
      figure; % Open a new figure window
      stem(m, s); % Plot the original signal
      grid on; % Enable grid
      title('Original Unaffected Signal');
      xlabel('Time Index n');
ylabel('Amplitude');
```



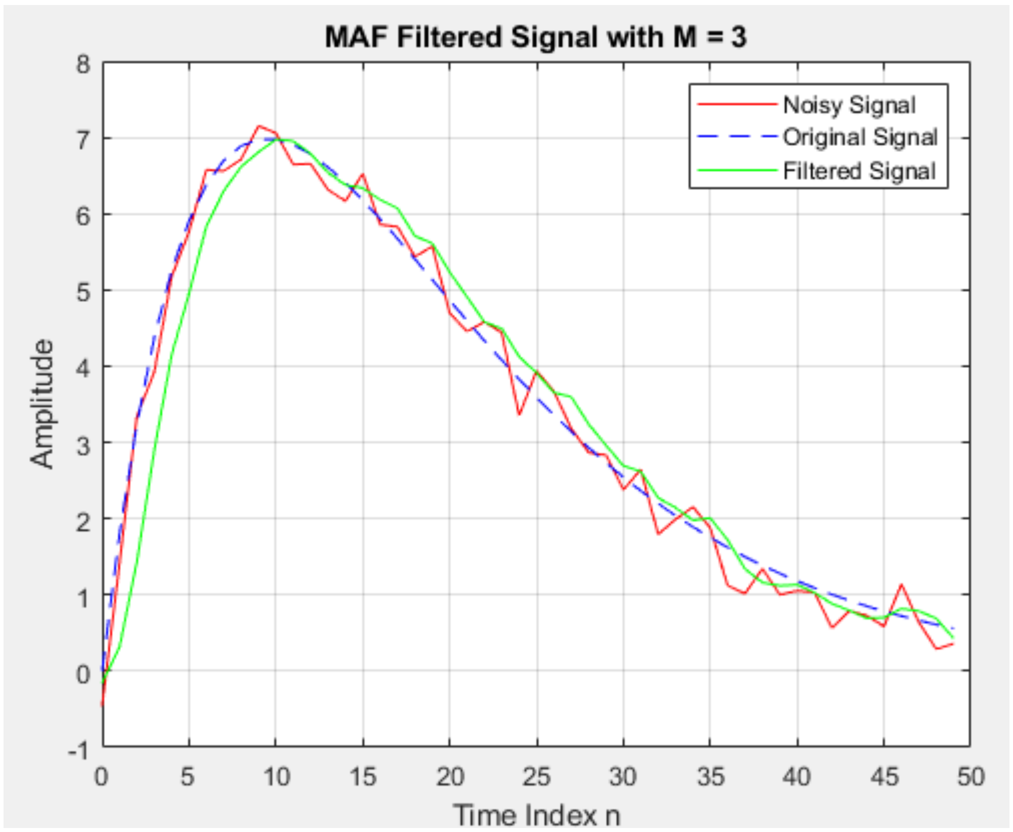
2.2. Generate a noise, using the rand function by adding in p. 2.1 the noise d=rand(1,length(m))-0.5.

```
     d = rand(1, length(m)) - 0.5; % Generating noise
     x = s + d; % Adding noise to the original signal

figure; % Open a new figure window
     plot(m, x, 'r-', m, s, 'b--'); % Plot noisy and original
     signals
     legend('Noisy Signal', 'Original Signal');
     grid on; % Enable grid
     title('Original and Noisy Signal');
     xlabel('Time Index n');
ylabel('Amplitude');
```



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

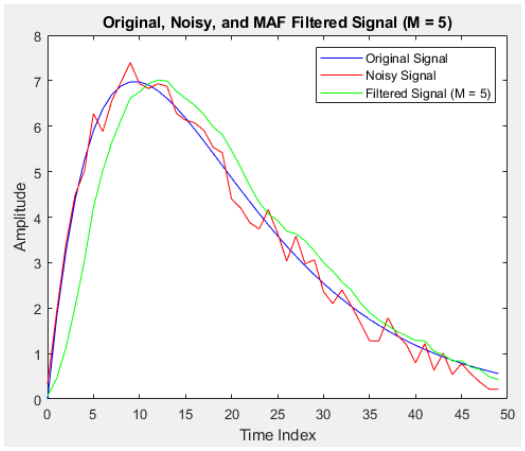
```
      M = 3; % Size of the moving average filter
      b = ones(1, M) / M; % Filter coefficients
      y = filter(b, 1, x); % Applying the MAF filter

      figure; % Open a new figure window
      plot(m, x, 'r-', m, s, 'b--', m, y, 'g-'); % Plot
      original, noisy, and filtered signals
      legend('Noisy Signal', 'Original Signal', 'Filtered
      Signal');
      grid on; % Enable grid
      title(sprintf('MAF Filtered Signal with M = %d', M));
      xlabel('Time Index n');
ylabel('Amplitude');
```

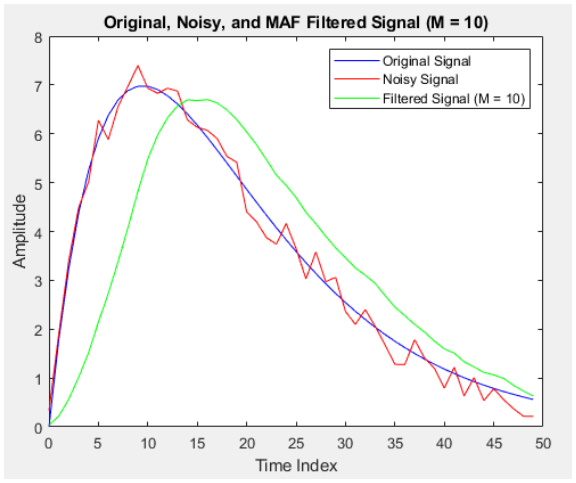


2.4. Represent the sum of these two signals x=s+d and represent the resulting signal x and the initial signal s on a single graph, using the plot function.

```
% Define the original signal
R = 50; % The range for the time index
m = 0:1:(R-1); % Time index
s = 2 * m .* (0.9.^m); % Original signal

% Generate noise and add it to the original signal to
create 'x'
d = rand(1, length(m)) - 0.5; % Noise
x = s + d; % Noisy signal

% Exercise 2.4: Apply MAF with M = 5
M5 = 5; % Filter length for M = 5
b5 = ones(1, M5) / M5; % Filter coefficients for M = 5
y_M5 = filter(b5, 1, x); % Filtering the signal with M = 5

% Plotting the original signal, noisy signal, and filtered
signal for M = 5
figure;
plot(m, s, 'b', m, x, 'r', m, y_M5, 'g');
title('Original, Noisy, and MAF Filtered Signal (M = 5)');
legend('Original Signal', 'Noisy Signal', 'Filtered Signal
(M = 5)');
xlabel('Time Index');
ylabel('Amplitude');

% Exercise 2.4 continuation: Apply MAF with M = 10
M10 = 10; % Filter length for M = 10
b10 = ones(1, M10) / M10; % Filter coefficients for M = 10
y_M10 = filter(b10, 1, x); % Filtering the signal with M =
10

% Plotting the original signal, noisy signal, and filtered
signal for M = 10
figure;
plot(m, s, 'b', m, x, 'r', m, y_M10, 'g');
title('Original, Noisy, and MAF Filtered Signal (M =
10)');
```

```
      legend('Original Signal', 'Noisy Signal', 'Filtered Signal
      (M = 10)');
      xlabel('Time Index');
ylabel('Amplitude');
```

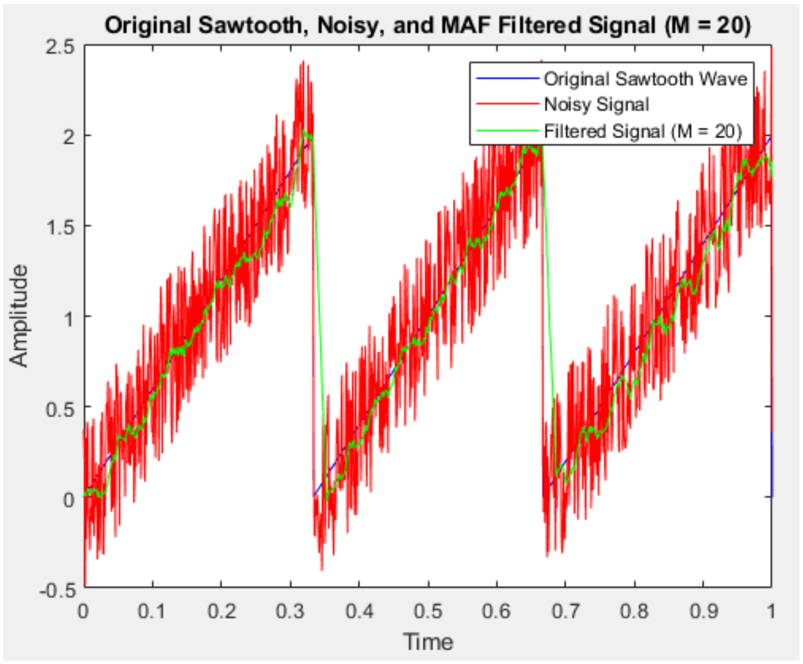




2.5. Design a MAF filter with parameters y=filter(b,1,x) b=ones(M,1)/M and previously specifying M=3 and filter the signal affected by noise. Represent the already filtered signal y and the one affected by the noiseg x, but also the initial one s on a single graph, using the plot function

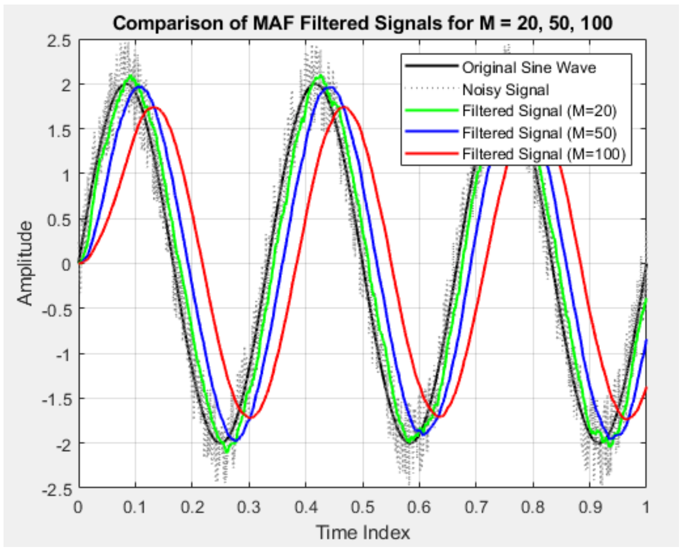
```
      % Parameters for sawtooth wave
      R = 1; % End time
      Fs = 1000; % Sampling frequency
      T = 1/Fs; % Sampling period
      t = 0:T:R; % Time vector

      % Sawtooth parameters
      f = 3; % Frequency of the sawtooth wave
      A = 2; % Amplitude of the sawtooth wave

      % Generate sawtooth wave manually
      sawtooth_wave = mod(t*f, 1) * A; % Simple sawtooth

      % Adding noise
      d = rand(1, length(t)) - 0.5;
      x = sawtooth_wave + d;

      % Apply MAF with M = 20
```

```
    M = 20;
    b = ones(1, M) / M;
    y_M20 = filter(b, 1, x);

    % Plotting
    figure;
    plot(t, sawtooth_wave, 'b', t, x, 'r', t, y_M20, 'g');
    title('Original Sawtooth, Noisy, and MAF Filtered Signal
    (M = 20)');
    legend('Original Sawtooth Wave', 'Noisy Signal', 'Filtered
    Signal (M = 20)');
    xlabel('Time');
ylabel('Amplitude');
```



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

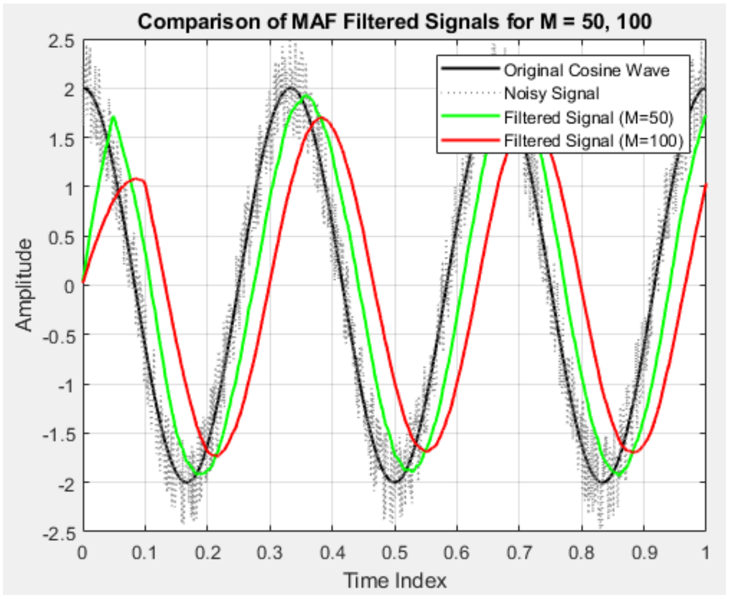
```

    % Duration and time index setup
    R = 1;
    m = 0:0.001:R;
    % Original signal - using a sine wave as a substitute for
    the sawtooth wave
    s = 2 * sin(2 * pi * 3 * m);

    % Generating noise and adding it to the sine wave to
    create a noisy signal
    d = rand(1, length(m)) - 0.5;
    x = s + d;

    % Filter lengths for comparison
    Ms = [20, 50, 100];

    % Preparing for plotting
    figure;
    plot(m, s, 'k', 'LineWidth', 1.5); % Original Signal
    hold on;
    plot(m, x, ':', 'Color', [0.5 0.5 0.5], 'LineWidth', 1); %
    Noisy Signal

    % Colors for different M values
    colors = ['g', 'b', 'r'];

    % Loop through each M value to filter and plot
    for i = 1:length(Ms)
        M = Ms(i); % Current M value
        b = ones(1, M) / M; % Filter coefficients
        y = filter(b, 1, x); % Applying the MAF filter

        % Plotting the filtered signal
        plot(m, y, colors(i), 'LineWidth', 1.5);
    end

    % Enhancing the plot
    legend('Original Sine Wave', 'Noisy Signal', 'Filtered
    Signal (M=20)', 'Filtered Signal (M=50)', 'Filtered Signal
    (M=100)');
    title('Comparison of MAF Filtered Signals for M = 20, 50,
    100');
    xlabel('Time Index');
    ylabel('Amplitude');
    grid on;
hold off;
```



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.

```
% Duration and time index setup
R = 1; % Duration of the signal
m = 0:0.001:R; % Time index with smaller step size

% Alternative signal - using a cosine wave for variation
s = 2 * cos(2 * pi * 3 * m);

% Generating noise and adding it to the cosine wave to
create a noisy signal
d = rand(1, length(m)) - 0.5; % Noise
x = s + d; % Noisy signal

% Filter lengths for comparison
Ms = [50, 100];

% Preparing for plotting
figure;
plot(m, s, 'k', 'LineWidth', 1.5); % Original Signal
hold on;
```

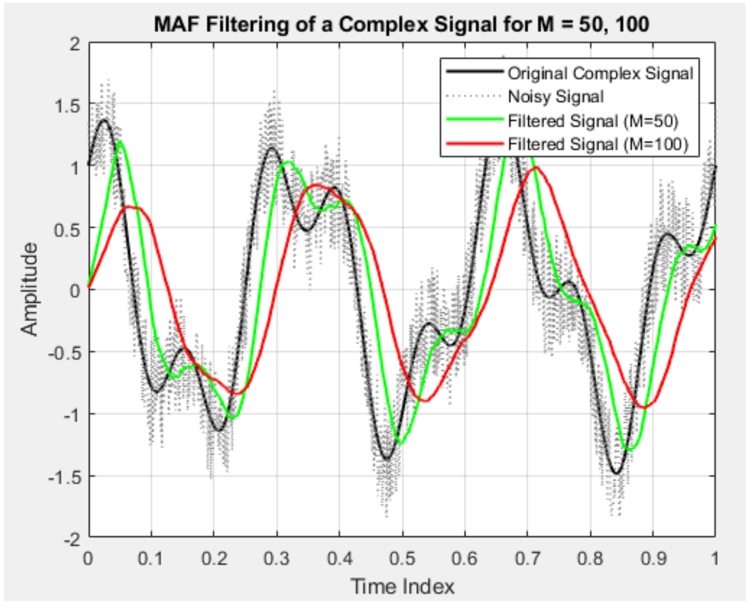
```
     plot(m, x, ':', 'Color', [0.5 0.5 0.5], 'LineWidth', 1); %
     Noisy Signal

     % Colors for different M values
     colors = ['g', 'r']; % Using two colors for M=50 and M=100

     % Loop through each M value to filter and plot
     for i = 1:length(Ms)
         M = Ms(i); % Current M value
         b = ones(1, M) / M; % Filter coefficients
         y = filter(b, 1, x); % Applying the MAF filter

         % Plotting the filtered signal
         plot(m, y, colors(i), 'LineWidth', 1.5);
     end

     % Enhancing the plot
     legend('Original Cosine Wave', 'Noisy Signal', 'Filtered
     Signal (M=50)', 'Filtered Signal (M=100)');
     title('Comparison of MAF Filtered Signals for M = 50,
     100');
     xlabel('Time Index');
     ylabel('Amplitude');
     grid on;
hold off;
```



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

```
% Duration and time index setup
R = 1; % Duration of the signal
m = 0:0.001:R; % Time index with smaller step size

% Generating a more complex signal by combining different
frequencies
s = cos(2 * pi * 3 * m) + 0.5 * sin(2 * pi * 8 * m);

% Adding noise to create a noisy signal
d = rand(1, length(m)) - 0.5; % Noise
x = s + d; % Noisy signal

% Preparing for filtering and plotting
Ms = [50, 100]; % Filter lengths for comparison
colors = ['g', 'r']; % Colors for M=50 and M=100 plots

% Plotting the original and noisy signals
figure;
plot(m, s, 'k', 'LineWidth', 1.5); % Original Signal
hold on;
plot(m, x, ':', 'Color', [0.5 0.5 0.5], 'LineWidth', 1); %
Noisy Signal

% Looping through each M value to filter and plot
for i = 1:length(Ms)
    M = Ms(i); % Current M value
    b = ones(1, M) / M; % Filter coefficients
    y = filter(b, 1, x); % Applying the MAF filter

    % Plotting the filtered signal
    plot(m, y, colors(i), 'LineWidth', 1.5);
end

% Finalizing the plot
legend('Original Complex Signal', 'Noisy Signal',
'Filtered Signal (M=50)', 'Filtered Signal (M=100)');
title('MAF Filtering of a Complex Signal for M = 50,
100');
```

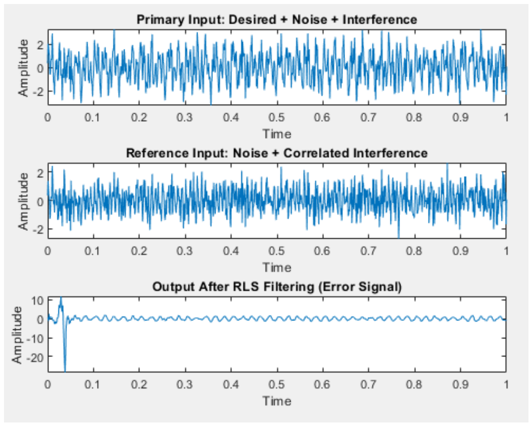
```
      xlabel('Time Index');
      ylabel('Amplitude');
      grid on;
hold off;
```



3.3. Open and launch the Niose canceller (LMS) application. Save the block diagram and the corresponding dependencies. Save informed

```
      % Parameters
      n = 0:0.001:1; % Time vector
      f1 = 50; % Frequency of the desired signal component
      f2 = 120; % Frequency of an undesired signal component
      noisePower = 0.5; % Power of the additive white Gaussian
      noise

      % Generating the signals
      desiredSignal = sin(2*pi*f1*n); % Desired signal component
      noise = sqrt(noisePower) * randn(size(n)); % Additive
      white Gaussian noise
      interferenceSignal = sin(2*pi*f2*n); % Interference signal
      component
      primaryInput = desiredSignal + noise + interferenceSignal;
      % Primary input (desired+noise+interference)
      referenceInput = noise + 0.5*sin(2*pi*f2*n); % Reference
      input (noise+correlated interference)
```

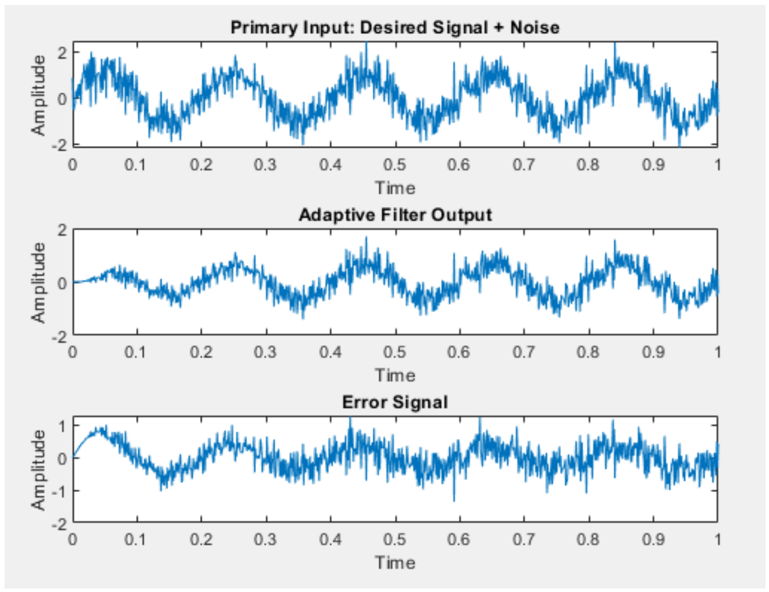
```
% RLS Filter Setup
forgetFactor = 0.99; % Forget factor for the RLS algorithm
order = 32; % Order of the adaptive filter
rlsFilter = dsp.RLSFilter('Length', order,
'ForgettingFactor', forgetFactor);

% Using the RLS filter for noise cancellation
[filteredOutput, err] = rlsFilter(referenceInput',
primaryInput');

% Plotting
figure;
subplot(3,1,1);
plot(n, primaryInput);
title('Primary Input: Desired + Noise + Interference');
xlabel('Time');
ylabel('Amplitude');

subplot(3,1,2);
plot(n, referenceInput);
title('Reference Input: Noise + Correlated Interference');
xlabel('Time');
ylabel('Amplitude');

subplot(3,1,3);
plot(n, err);
title('Output After RLS Filtering (Error Signal)');
xlabel('Time');
ylabel('Amplitude');
```



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.

```
% Parameters
n = 0:0.001:1; % Time vector
desiredSignal = sin(2*pi*5*n); % Desired signal component
noise = 0.5 * randn(size(n)); % Additive white Gaussian
noise
primaryInput = desiredSignal + noise; % Noisy signal
(desired signal + noise)

% Adaptive Filter Setup
coeff = 0; % Initial coefficient (weight)
mu = 0.01; % Learning rate
numIterations = length(n); % Number of iterations equals
the length of the signal
output = zeros(size(primaryInput)); % Initialize the
output of the adaptive filter
error = zeros(size(primaryInput)); % Initialize the error
signal

% Adaptive Filtering Process (Simplified)
for i = 1:numIterations
    % Here, the 'reference input' is the primary input
itself
    output(i) = coeff * primaryInput(i); % Filter output
    error(i) = desiredSignal(i) - output(i); % Error
calculation
    coeff = coeff + mu * error(i) * primaryInput(i); %
Coefficient update (simplified adaptation)
end

% Plotting
figure;
subplot(3,1,1);
plot(n, primaryInput);
title('Primary Input: Desired Signal + Noise');
xlabel('Time');
ylabel('Amplitude');
```

```
subplot(3,1,2);
plot(n, output);
title('Adaptive Filter Output');
xlabel('Time');
ylabel('Amplitude');

subplot(3,1,3);
plot(n, error);
title('Error Signal');
xlabel('Time');
ylabel('Amplitude');
```



Conclusion

The laboratory sessions offered a detailed exploration of digital signal processing (DSP), with an emphasis on discrete-time systems, noise generation and mitigation, and adaptive filtering techniques such as Least Mean Squares (LMS). These sessions involved practical exercises using MATLAB, where you engaged in modifying and enhancing signals through various filtering methods. Specifically, you learned how to diminish or eliminate noise to improve the quality of signals. The labs also delved into the adaptive nature of certain filters, which modify themselves in real-time based on changes in signals or noise. While some of the more complex topics were covered only theoretically due to limitations in the available tools, the combination of theoretical and practical learning provided a solid foundation in DSP. This experience underscored the value of ongoing learning and adaptability in mastering DSP techniques, preparing you for applications in diverse professional settings.