Practical Work No. 2
Learning Preliminary Processing of One-Dimensional Signals in MATLAB/Python Programming Environment

Objective

The aim of this practical work is to study basic operations of preliminary processing of one-dimensional signals. Students will learn how to generate, visualize, and analyze signals, as well as apply fundamental preprocessing techniques such as normalization, filtering, and noise removal in MATLAB and Python.

1.1 Generate a Simple Sine Signal

- A sine wave is a periodic function described by the equation:

$$x(t) = A \cdot \sin(2\pi f t + \phi)$$

where:

- A → Amplitude (height of the wave)
- f→ Frequency (number of cycles per second, in Hz)
- t → Time variable
- ϕ → Phase shift (shifting the wave left or right)

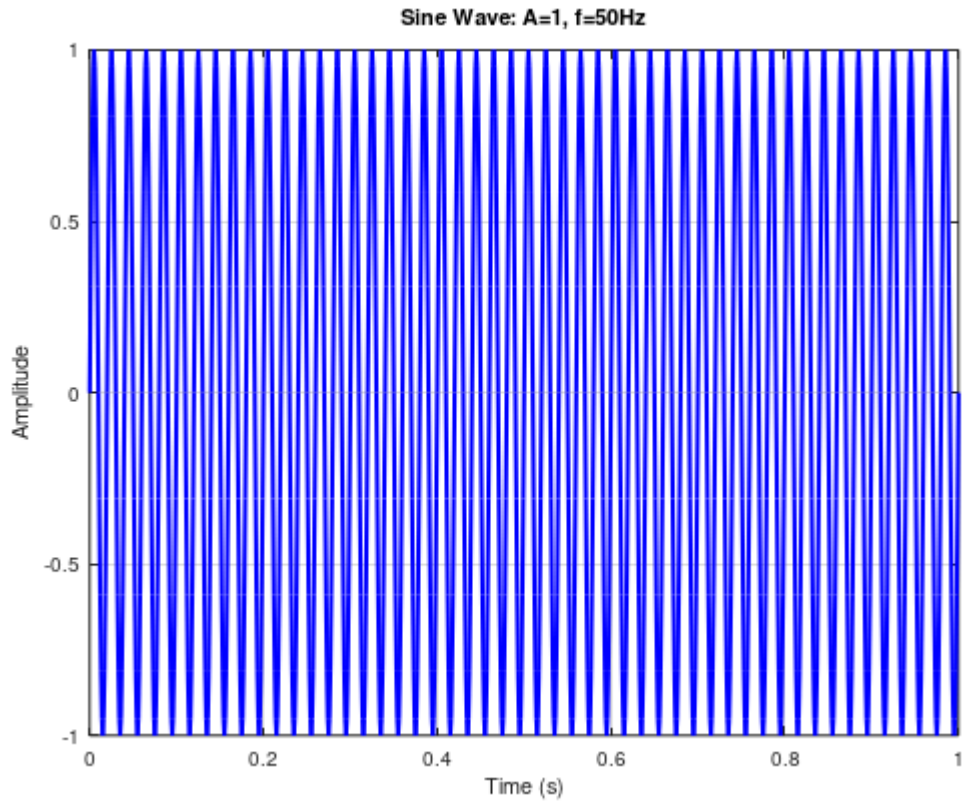
Example:

- Amplitude A=1
- Frequency f=50 Hz
- Phase ϕ=0
- Sampling frequency F_s=1000 Hz
- Duration T=1 s

This will generate a clean sine wave oscillating 50 times per second.

Matlab code :

```
% Practical Work 2 - Task 1.1
% Generate a Simple Sine Signal

% Parameters
A = 1;          % Amplitude
f = 50;         % Frequency (Hz)
phi = 0;        % Phase (radians)
Fs = 1000;      % Sampling frequency (Hz)
T = 1;          % Duration (seconds)

% Time vector
t = 0:1/Fs:T;   % From 0 to 1 second with step 1/Fs

% Generate sine signal
x = A * sin(2*pi*f*t + phi);

% Plot signal
figure;
plot(t, x, 'b', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Amplitude');
title('Sine Wave: A=1, f=50Hz');
grid on;
```



1.2 Generate a Simple Cosine Signal

- A cosine wave is similar to a sine wave but shifted in phase by 90̊

$$y(t) = A \cdot \cos(2\pi f t + \phi)$$

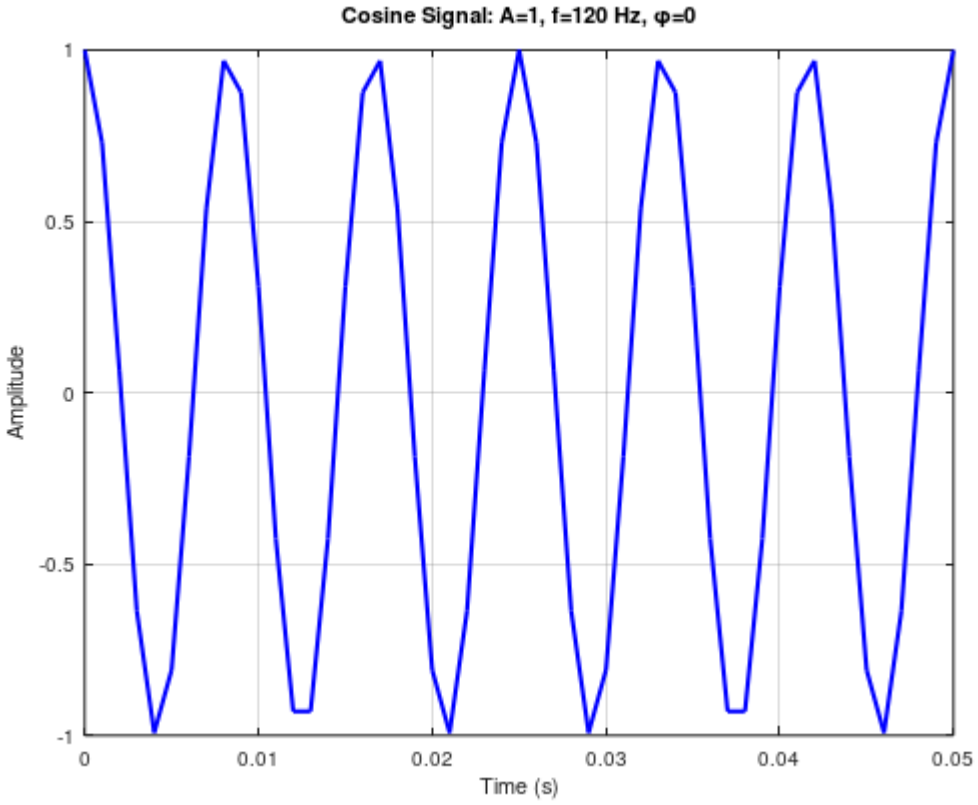
Example:

- Amplitude A=1
- Frequency f=120 Hz
- Phase ϕ=0

This wave oscillates faster than the sine wave because its frequency is higher.

Matlab code:

```
% Parameters
A = 1;              % Amplitude
f = 120;            % Frequency in Hz
phi = 0;            % Phase in radians
fs = 1000;          % Sampling frequency (Hz)
t = 0:1/fs:0.05;    % Time vector (0 to 0.05 s)

% Generate cosine signal
x = A * cos(2*pi*f*t + phi);

% Plot
figure;
plot(t, x, 'b', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Amplitude');
title('Cosine Signal: A=1, f=120 Hz, \phi=0');
grid on;
```



1.3 Create a Composite Signal

- A composite signal can be formed by summing multiple signals.
- Here, we combine the sine and cosine signals:

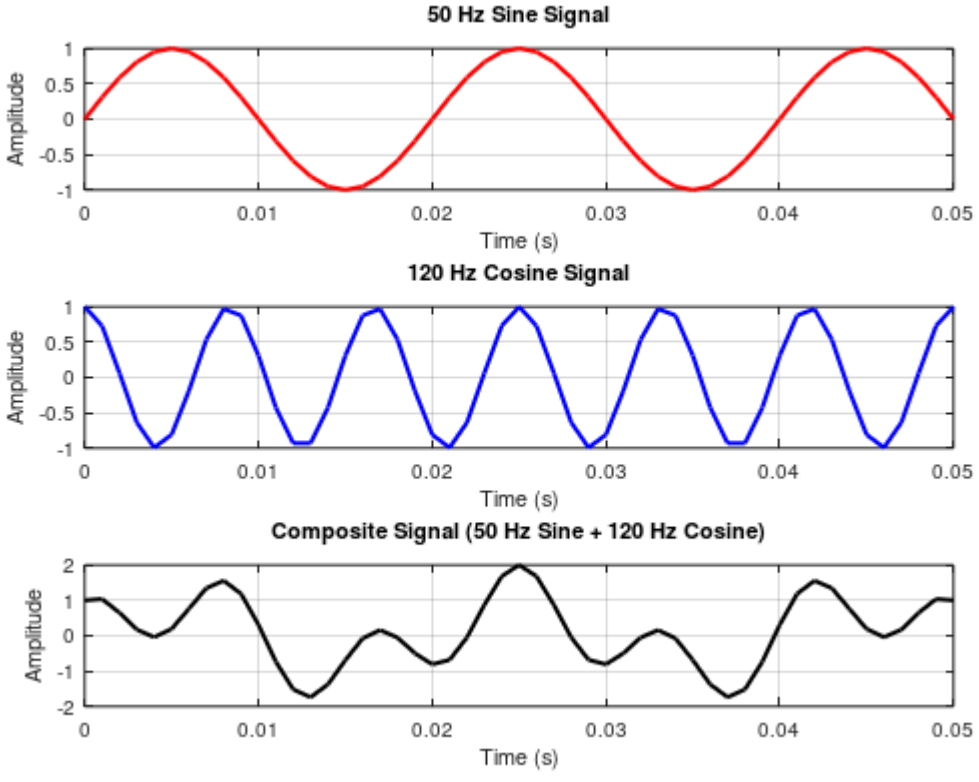
$$s(t) = x(t) + y(t)$$

This means the resulting signal contains **two frequency components**: one at 50 Hz (from sine) and another at 120 Hz (from cosine).

Composite signals are very important in signal processing because real-world signals (like audio, ECG, or vibration data) often consist of many frequency components combined together.

```
% Parameters
A1 = 1;          % Amplitude of sine wave
f1 = 50;         % Frequency of sine wave (Hz)
phi1 = 0;        % Phase of sine wave (rad)

A2 = 1;          % Amplitude of cosine wave
f2 = 120;        % Frequency of cosine wave (Hz)
phi2 = 0;        % Phase of cosine wave (rad)

fs = 1000;       % Sampling frequency (Hz)
t = 0:1/fs:0.05;   % Time vector (0 to 0.05 s)
```

```
% Generate individual signals
x1 = A1 * sin(2*pi*f1*t + phi1);   % 50 Hz sine
x2 = A2 * cos(2*pi*f2*t + phi2);   % 120 Hz cosine

% Composite signal
x = x1 + x2;

% Plot
figure;
subplot(3,1,1);
plot(t, x1, 'r', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title('50 Hz Sine Signal');
grid on;

subplot(3,1,2);
plot(t, x2, 'b', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title('120 Hz Cosine Signal');
grid on;

subplot(3,1,3);
plot(t, x, 'k', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title('Composite Signal (50 Hz Sine + 120 Hz Cosine)');
grid on;
```



1.4 Visualization of Signals

Visualization helps us **understand** the properties of the signal (amplitude, frequency, periodicity). In MATLAB or Python, we use plotting functions to display:

1. **Sine signal plot** → shows a smooth oscillating curve with a lower frequency.
2. **Cosine signal plot** → shows a faster oscillation due to its higher frequency.
3. **Composite signal plot** → displays the combination of both signals. You will notice a more complex waveform, because it includes both low- and high-frequency components.

For better visualization:

- The x-axis → represents **time (seconds)**
- The y-axis → represents **signal amplitude**
- Each signal can be plotted in separate figures or together in subplots for comparison.

2. Signal Normalization

Normalization is a preprocessing step used to **scale a signal** to a specific range, typically [−1,1] or [0,1].
This is important because signals often have different amplitudes, and comparing or processing them without normalization can cause errors.

- **General formula for normalization (range [−1,1]):**

$$x_{norm}(t) = \frac{x(t)}{\max(|x(t)|)}$$

This ensures that the **maximum amplitude becomes 1** and the minimum becomes -1.

Why normalization is important:

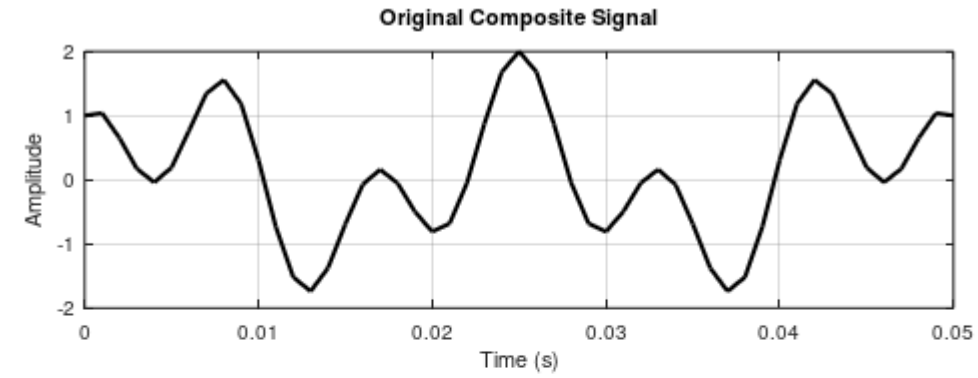
- Prevents distortion when combining multiple signals.
- Ensures numerical stability in algorithms (like FFT, filtering).
- Makes visualization easier since signals fit within the same scale.

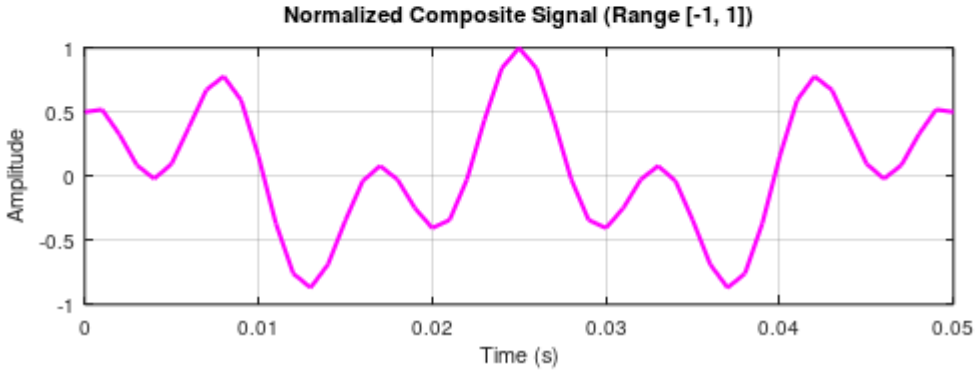
Example in practice:

- If the maximum amplitude of the composite signal is 2.5, after normalization, the signal values will lie between -1 and +1.

Matlab Code

```
% Composite signal (from earlier example)
A1 = 1; f1 = 50; phi1 = 0;
A2 = 1; f2 = 120; phi2 = 0;
fs = 1000;
t = 0:1/fs:0.05;

x1 = A1 * sin(2*pi*f1*t + phi1);   % 50 Hz sine
x2 = A2 * cos(2*pi*f2*t + phi2);   % 120 Hz cosine
x = x1 + x2;                       % Composite signal

% Normalization (range [-1, 1])
x_norm = x / max(abs(x));

% Plot original vs normalized
figure;
subplot(2,1,1);
plot(t, x, 'k', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title('Original Composite Signal');
grid on;

subplot(2,1,2);
plot(t, x_norm, 'm', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title('Normalized Composite Signal (Range [-1, 1])');
grid on;
```





3. Adding Noise

Real-world signals are rarely clean — they are usually corrupted by **noise**. Noise can come from environment, sensors, or transmission.
In this task, we add **Gaussian noise** (random values following a normal distribution) to simulate real-world conditions.

- **Noisy signal model:**

$$s_{noisy}(t) = s(t) + n(t)$$

where:

- s(t)= original signal
- n(t) = random noise (usually with mean 0 and small variance)

Types of noise:

- **Gaussian noise** (most common) – used here.
- **Impulse noise** (sudden spikes).
- **Uniform noise** (random flat distribution).

Why add noise in experiments?

- To test how robust preprocessing and filtering techniques are.
- To simulate real-world conditions like audio signals recorded in a noisy environment.

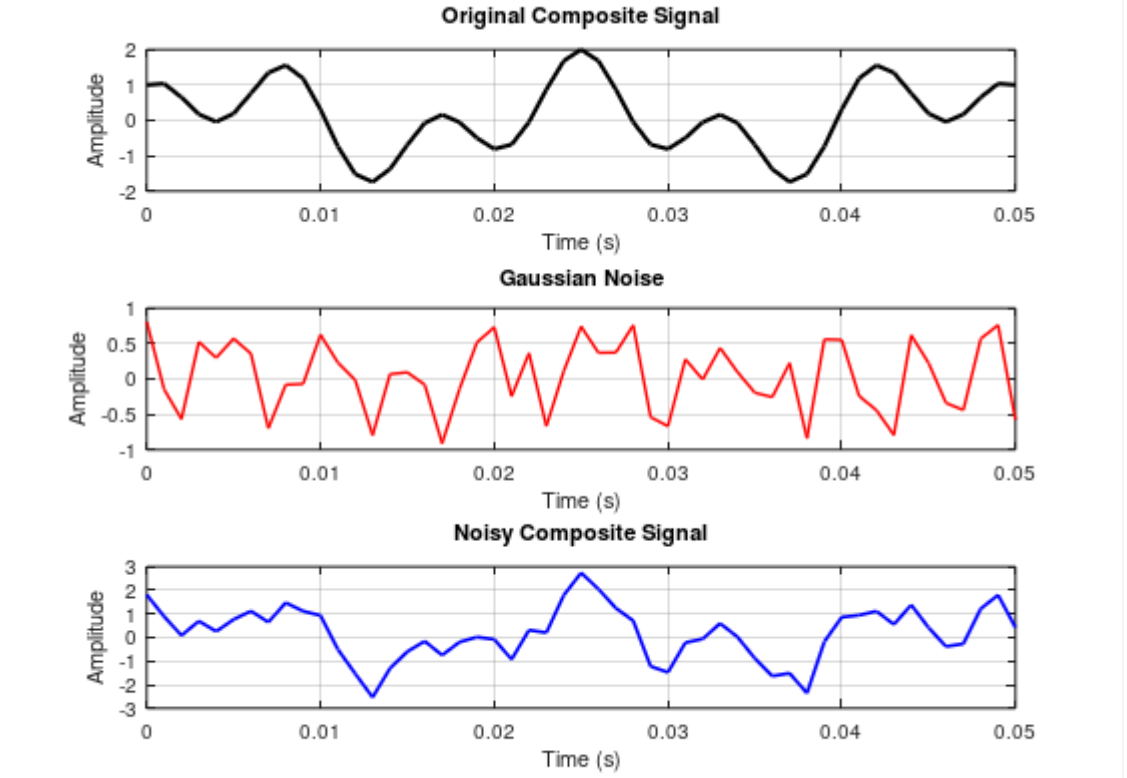
Visualization:

- A clean sine/cosine wave looks smooth.
- After adding noise, the signal becomes jagged, but the main waveform is still visible.

Matlab code:

```
% Parameters
A1 = 1; f1 = 50; phi1 = 0;
A2 = 1; f2 = 120; phi2 = 0;
fs = 1000;
t = 0:1/fs:0.05;

% Original composite signal
x1 = A1 * sin(2*pi*f1*t + phi1);   % 50 Hz sine
x2 = A2 * cos(2*pi*f2*t + phi2);   % 120 Hz cosine
x = x1 + x2;

% Add Gaussian noise
noise_level = 0.5;                 % Adjust noise strength (variance)
n = noise_level * randn(size(t));   % Gaussian noise ~ N(0, σ²)
x_noisy = x + n;

% Plot signals
figure;
subplot(3,1,1);
plot(t, x, 'k', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title('Original Composite Signal');
grid on;

subplot(3,1,2);
plot(t, n, 'r', 'LineWidth', 1);
xlabel('Time (s)'); ylabel('Amplitude');
title('Gaussian Noise');
grid on;

subplot(3,1,3);
plot(t, x_noisy, 'b', 'LineWidth', 1.2);
xlabel('Time (s)'); ylabel('Amplitude');
title('Noisy Composite Signal');
```

grid on;



4. Filtering

Filtering is the process of **removing unwanted components** from a signal.
In this task, we will use two common approaches:

4.1 Moving Average Filter

- One of the simplest filters.
- It replaces each data point with the average of its neighbors within a window.

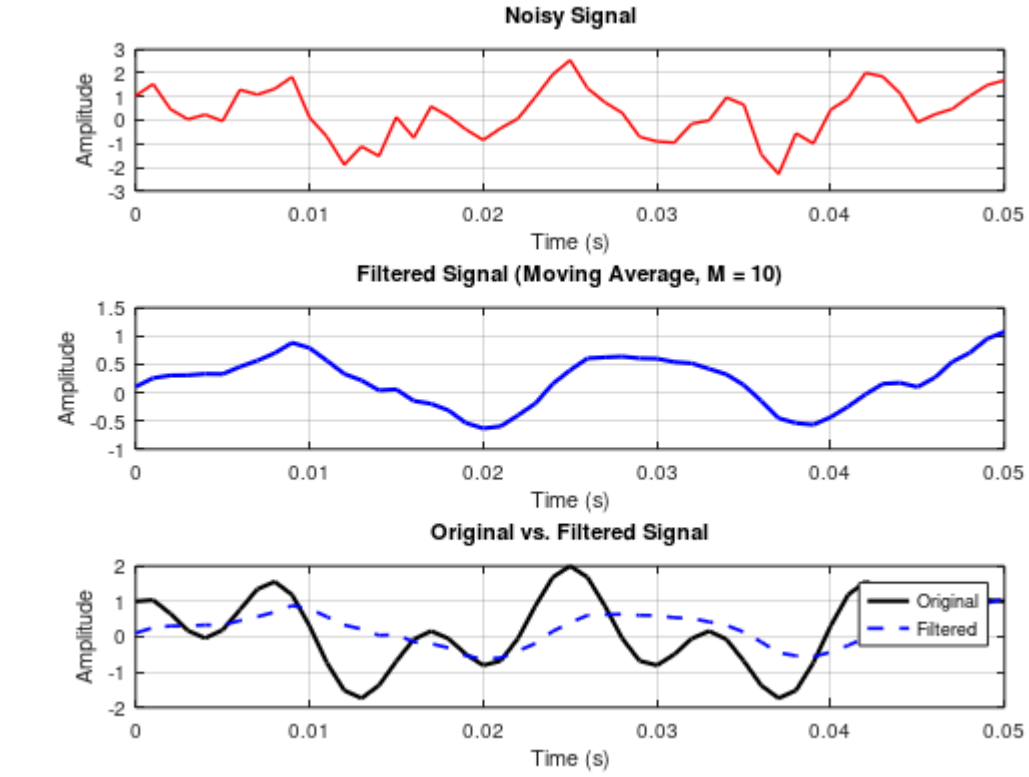
$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

where M is the window size.

- **Effect:** Smooths the signal and reduces random noise.
- **Drawback:** May also reduce sharp changes in the signal.

Matlab: code:

```
% Parameters (same as before)
A1 = 1; f1 = 50; phi1 = 0;
A2 = 1; f2 = 120; phi2 = 0;
fs = 1000;
t = 0:1/fs:0.05;

% Original composite signal
x1 = A1 * sin(2*pi*f1*t + phi1);   % 50 Hz sine
x2 = A2 * cos(2*pi*f2*t + phi2);   % 120 Hz cosine
x = x1 + x2;

% Add Gaussian noise
noise_level = 0.5;
n = noise_level * randn(size(t));
x_noisy = x + n;

% ---- Moving Average Filter ----
M = 10;                          % Window size
b = (1/M) * ones(1, M);              % Filter coefficients
x_filtered = filter(b, 1, x_noisy);   % Apply moving average

% Plot results
figure;
subplot(3,1,1);
plot(t, x_noisy, 'r', 'LineWidth', 1);
xlabel('Time (s)'); ylabel('Amplitude');
title('Noisy Signal');
grid on;

subplot(3,1,2);
plot(t, x_filtered, 'b', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
title(['Filtered Signal (Moving Average, M = ' num2str(M) ')']);
grid on;

subplot(3,1,3);
plot(t, x, 'k', 'LineWidth', 1.5); hold on;
plot(t, x_filtered, 'b--', 'LineWidth', 1.2);
xlabel('Time (s)'); ylabel('Amplitude');
title('Original vs. Filtered Signal');
legend('Original', 'Filtered');
grid on;
```



Summary of Tasks

- **Signal Generation:** Created sine and cosine waves, and a composite signal.
- **Normalization:** Scaled signals into a fixed range for fair comparison.
- **Adding Noise:** Simulated real-world scenarios by introducing Gaussian noise.
- **Filtering:** Used moving average and low-pass filtering to clean signals.
- **Analysis (FFT):** Compared original, noisy, and filtered signals in the frequency domain.

Generate a Signal

1.A wave is a periodic function described by the equation:

$$X(t)= 2A*\cos(phi*f*t+\phi)$$

- Amplitude A=1

- Frequency f=50 Hz
- Phase ϕ=0
- Sampling frequency F_s=1000 Hz
- Duration T=1 s

2.A wave is a periodic function described by the equation:

X(t)= 2A*sin(phi*f*t+ϕ)

- Amplitude A=1
- Frequency f=4 Hz
- Phase ϕ=0
- Sampling frequency F_s=1000 Hz
- Duration T=0.5 s

3.A wave is a periodic function described by the equation:

X(t)= A*cos(phi*f*t+2*ϕ)

- Amplitude A=1
- Frequency f=10 Hz
- Phase ϕ=0
- Sampling frequency F_s=1000 Hz
- Duration T=0.4 s

4.A wave is a periodic function described by the equation:

X(t)= A*sin(phi*f*t+ϕ)

- Amplitude A=1
- Frequency f=3 Hz

- Phase ϕ=1
- Sampling frequency F_s=1000 Hz
- Duration T=0.5 s