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

Individual Work Nr. 3

FREQUENCY DOMAIN ANALYSIS OF SIGNALS USING MATLAB

Elaborated: **Cristian Brinza**
st. gr. FAF-212

Verified: **Railean Serghei**
lect. univ

Chișinău, 2024

Purpose of the laboratory work:

Analysis of the frequency domain of the sinusoidal signal using MATLAB.

Theory:

When discretizing a continuous signal $f(t)f(t)$ within any interval TT seconds, we generate a sequence of values from the original signal: $fk=f(kT)fk=f(kT)$.

In MATLAB, vector notation typically starts with index 1: $x(1),x(2),\ldots x(1),x(2),\ldots$. However, signal notation commonly begins with zero: $g0,g1,\ldots g0,g1,\ldots$, and extends to include negative values like $h-2,h-1,h0,\ldots h-2,h-1,h0,\ldots$. If signals gg and hh each consist of 10 values, their corresponding vectors will also contain 10 values. For instance, vector $g\mathbf{g}$ will hold values from 0 to 9, and vector $h\mathbf{h}$ will encompass values from -2 to 7.

The frequency domain of signals can be depicted using complex values, which represent the sinusoids constituting the signal.

The Discrete Fourier Transform (DFT) algorithm serves to convert a digital signal from the time domain into a series of points in the frequency domain. The input for the DFT algorithm comprises a set of NN values from the time domain $[fk][fk]$.

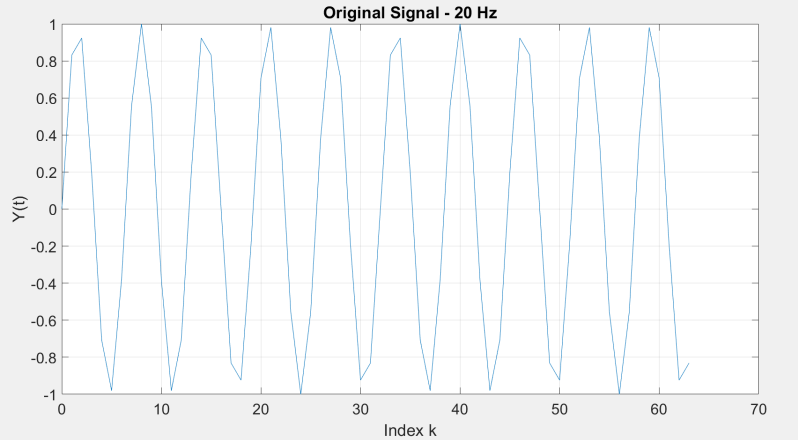
The algorithm computes a set of NN complex values $[Fk][Fk]$ that convey information about the frequency domain. When NN is a power of 2 ($N=2MN=2M$), the Fast Fourier Transform (FFT) is utilized.

Implementation:

1. **FOURIER TRANSFORMATION**:

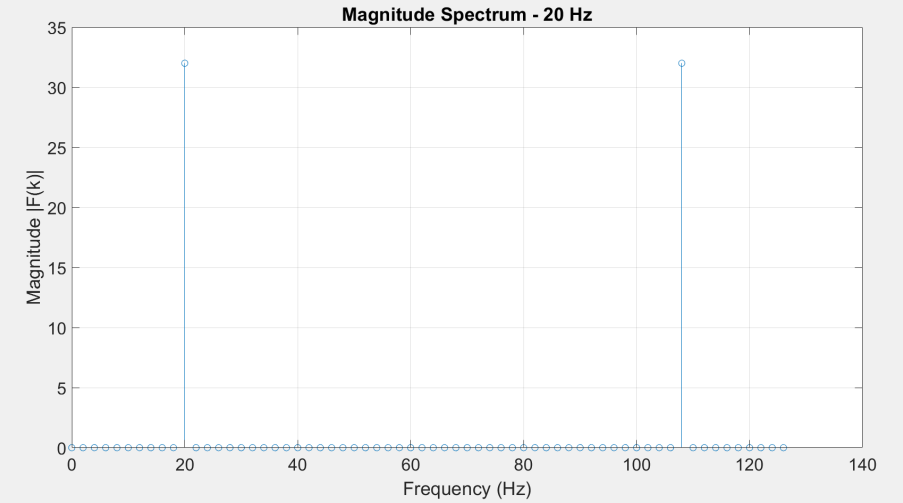
   1. ***Generate a discrete signal containing 64 samples.***

```
% Define parameters
N = 64;
T = 1/128;
k = 0:N-1;

% Generate the sine wave signal
f = sin(2*pi*20*k*T);

% Plot the signal
figure;
plot(k, f);
grid on;set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Index k');ylabel('Y(t)');
title('Original Signal - 20 Hz');
```



2. ***To determine the Fk that corresponds to the 20 Hz frequency, the step in Hz between the points in the frequency domain is modeled, which is 1/NT or 2 Hz. Now the 20 Hz component will appear at F10 (k=10 in the frequency domain). To visualize perform the Fourrier transform using F=fft(f), then display the modulus dependence of the Fourrier transform using plot(k,abs(F)).***

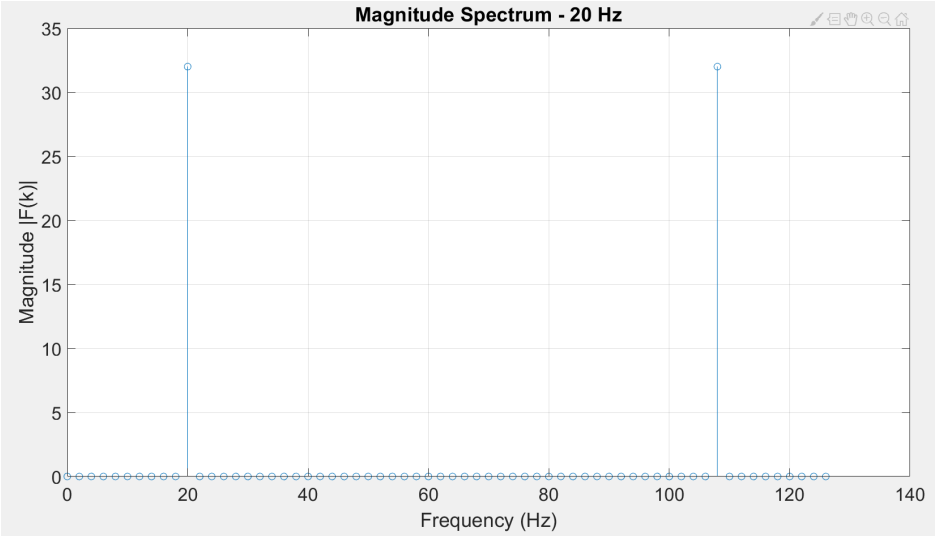
```
% Compute the Fourier Transform of the signal
F = fft(f);

% Frequency axis
frequency = (0:N-1)*(1/(N*T));

% Plot the magnitude of the Fourier Transform
figure;
stem(frequency, abs(F));
grid on;
set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Frequency (Hz)');
ylabel('Magnitude |F(k)|');
title('Magnitude Spectrum - 20 Hz');
```



The significant component at 20 Hz will appear at index 10 (k=10) in the frequency domain because the frequency resolution is 1/(N*T) = 2 Hz (from N=64 and T=1/128). The signal component at 20 Hz also appears at F54 due to the symmetry of the FFT and the sampling theorem.

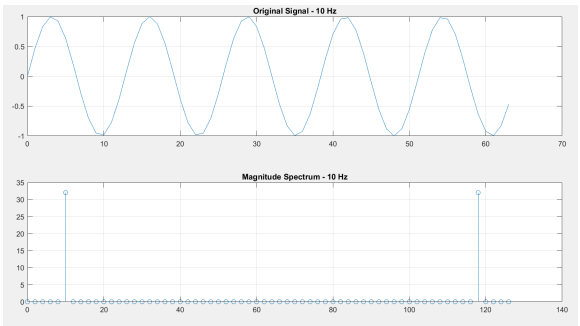
To confirm this periodicity, repeat p. 1.2. for the signal frequency f 108 Hz.

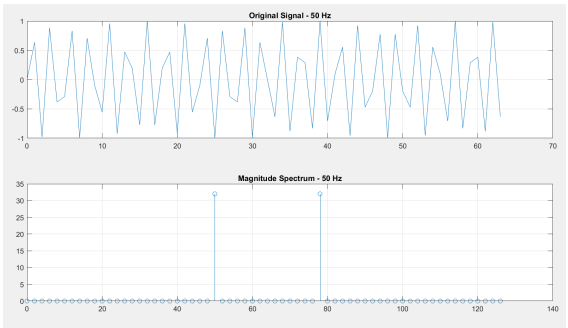
```
% Compute the Fourier Transform of the signal
F = fft(f);

% Frequency axis
frequency = (0:N-1)*(1/(N*T));

% Plot the magnitude of the Fourier Transform
figure;
stem(frequency, abs(F));
grid on;
set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Frequency (Hz)');
ylabel('Magnitude |F(k)|');
title('Magnitude Spectrum - 20 Hz');
```

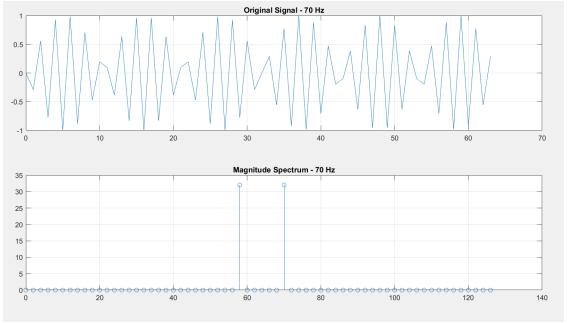


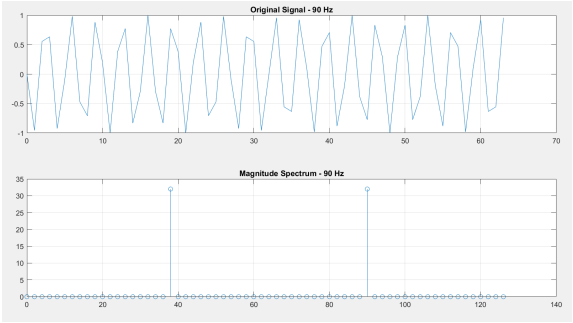
The procedure suggests repeating the analysis for a signal frequency of 108 Hz to observe the effect of frequency aliasing due to the symmetry of the FFT in representing frequencies above half the sampling rate (Nyquist frequency).

```
% Define signal frequencies for analysis
frequencies = [10, 50, 70, 90];
for freq = frequencies
    % Generate the sine wave signal
    f_var = sin(2*pi*freq*k*T);

    % Compute its Fourier Transform
    F_var = fft(f_var);

    % Plot each signal and its frequency spectrum
    figure;
    subplot(2,1,1);
    plot(k, f_var);
    grid on;
    title(['Original Signal - ', num2str(freq), ' Hz']);

    subplot(2,1,2);
    stem(frequency, abs(F_var));
    grid on;
    title(['Magnitude Spectrum - ', num2str(freq), ' Hz']);
end
```





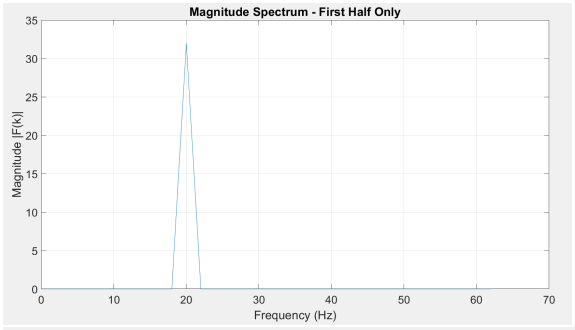


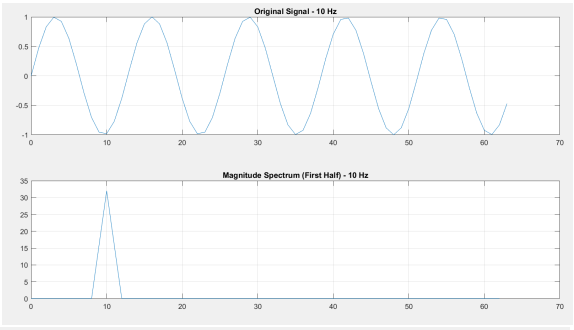


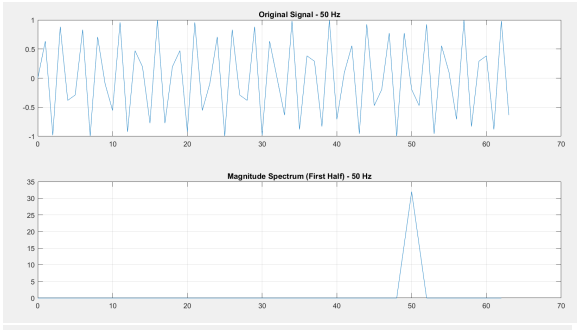
3. ***It is generally recommended to draw only half of the modulus values. Likewise, it is more convenient to present the x-axis in Hz than the index k. For this use the representation of the parameter k in Hz hertz=k*(1/(N*T)) and display only half of the obtained spectrum plot(hertz(1:N/2),magF(1:N/2)), preventive specifying magF=abs(F).***

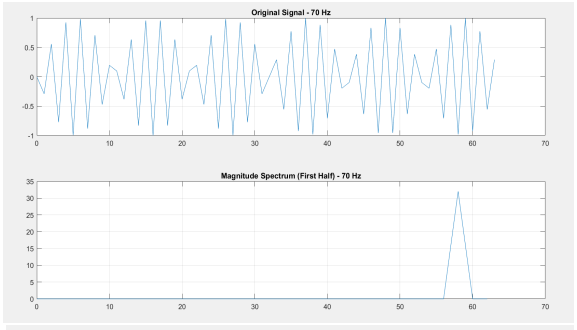
```
% Define parameters
N = 64;
T = 1/128;
k = 0:N-1;

% Generate the sine wave signal at 20 Hz
f = sin(2*pi*20*k*T);

% Compute the Fourier Transform of the signal
F = fft(f);

% Compute magnitude of the Fourier Transform
magF = abs(F);

% Frequency axis in Hertz
hertz = k * (1 / (N * T));

% Plot the first half of the magnitude spectrum
figure;
plot(hertz(1:N/2), magF(1:N/2)); % Only plotting the first half
grid on;
set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Frequency (Hz)');
ylabel('Magnitude |F(k)|');
title('Magnitude Spectrum - First Half Only');

% Analysis repeated for different frequencies if necessary
frequencies = [10, 50, 70, 90, 108]; % Includes an additional 108 Hz for
the previous note
for freq = frequencies
    % Generate the sine wave signal
    f_var = sin(2*pi*freq*k*T);

    % Compute its Fourier Transform
    F_var = fft(f_var);
```

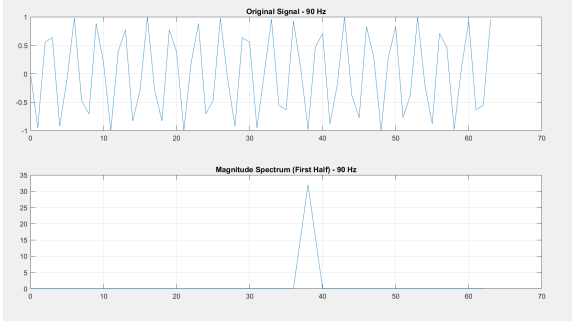
```
    % Calculate magnitude
    magF_var = abs(F_var);

    % Plot each signal and its frequency spectrum for the first half
only
    figure;
    subplot(2,1,1);
    plot(k, f_var);
    grid on;
    title(['Original Signal - ', num2str(freq), ' Hz']);

    subplot(2,1,2);
    plot(hertz(1:N/2), magF_var(1:N/2));
    grid on;
    title(['Magnitude Spectrum (First Half) - ', num2str(freq), ' Hz']);
end
```

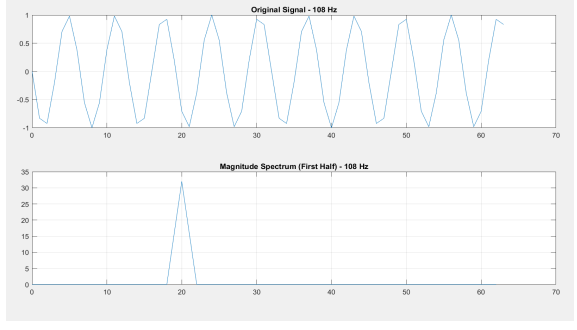








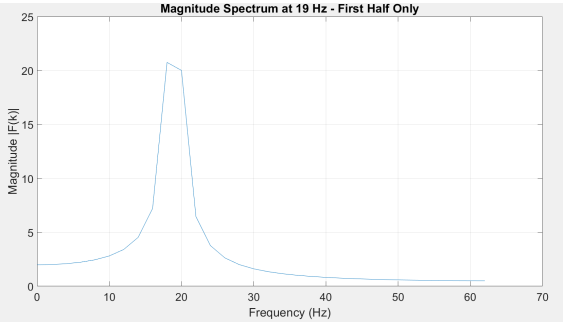




The plots display the magnitude spectrum of the Fourier Transform, focusing on the first half which represents the unique frequency components of the signal due to the symmetry in the FFT of real-valued signals. The primary peak observed in each plot corresponds to the frequency of the sinusoidal signal being analyzed (20 Hz, 10 Hz, 50 Hz, 70 Hz, 90 Hz, and 108 Hz), clearly showing how each sinusoid's frequency content is distinctly captured and visualized through the Fourier Transform.

4. ***Suppose that the sinusoidal frequency was 19 Hz (instead of 20). The step of increasing Fk values in Hz is 2 Hz, so the sinusoid will appear at Fk where k=9.5. But k is integral and there are no values at F9.5. In this case the sinusoid will appear at the neighboring values F9 and F10. To illustrate this, repeat point 1.3 by changing the frequency of the signal f to 19 Hz.***

```
% Define parameters
N = 64;
T = 1/128;
k = 0:N-1;

% Generate the sine wave signal at 19 Hz
f = sin(2*pi*19*k*T);

% Compute the Fourier Transform of the signal
F = fft(f);

% Compute magnitude of the Fourier Transform
magF = abs(F);

% Frequency axis in Hertz
hertz = k * (1 / (N * T));

% Plot the first half of the magnitude spectrum
figure;
plot(hertz(1:N/2), magF(1:N/2)); % Only plotting the first half
grid on;
set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16);
xlabel('Frequency (Hz)');
ylabel('Magnitude |F(k)|');
title('Magnitude Spectrum at 19 Hz - First Half Only');
```

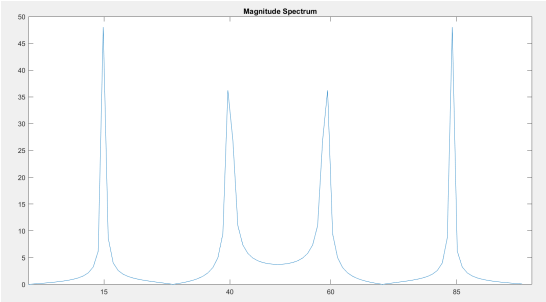


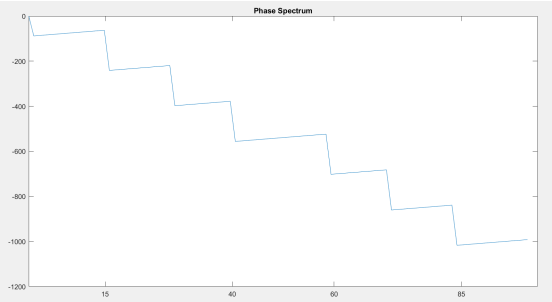
The spectrum shows spectral leakage around the frequencies 18 Hz and 20 Hz, as the 19 Hz signal frequency does not align perfectly with the DFT bins, which are spaced at 2 Hz. This results in the signal energy being distributed primarily between the nearest bins (F9 and F10) rather than being concentrated at a single bin, illustrating the effect of frequency misalignment in the FFT.

2. **DETERMINING THE MODE AND PHASE OF THE FOURIER TRANSFORM**:

1. Create a signal by the sum of two sinusoids and display the spectrum of the modulus and the spectrum of the phase of the Fourrier transform.

```
% Time vector
t = (0:1/99:1); % Defines a time vector from 0 to 1 with a
step of 1/99

% Signal creation
x = sin(2*pi*15*t) + sin(2*pi*40*t); % Sum of two
sinusoids at 15 Hz and 40 Hz

% Discrete Fourier Transform (DFT)
y = fft(x); % DFT of the signal

% Magnitude and phase
m = abs(y); % Magnitude of the Fourier coefficients
p = unwrap(angle(y)); % Unwrapped phase of the Fourier
coefficients

% Frequency vector
f = (0:length(y)-1)*99/length(y); % Frequency vector for
plotting

% Plot Magnitude Spectrum
figure;
plot(f, m);
title('Magnitude Spectrum');
set(gca, 'XTick', [15 40 60 85]); % Set x-axis ticks for
clearer visualization

% Plot Phase Spectrum
figure;
plot(f, p*180/pi); % Convert phase from radians to degrees
title('Phase Spectrum');
set(gca, 'XTick', [15 40 60 85]); % Set x-axis ticks for
clearer visualization
```



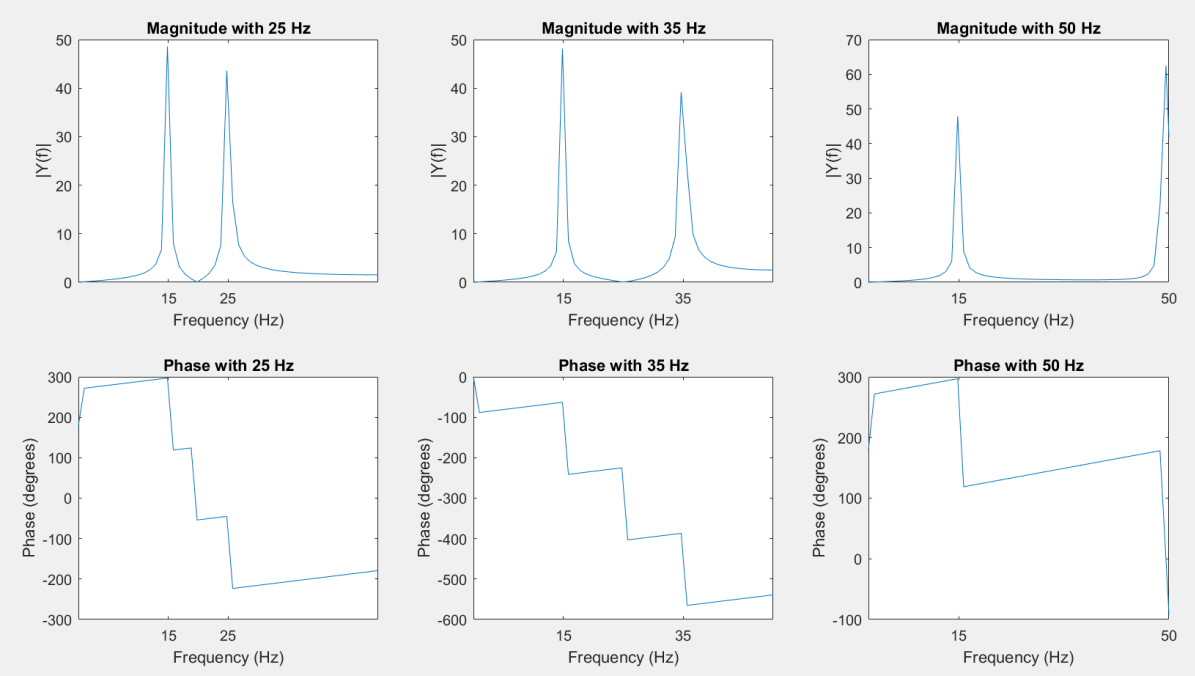


This code plots the magnitude and phase spectrum of a signal composed of two sinusoids. The magnitude spectrum will have peaks at 15 Hz and 40 Hz, corresponding to the frequencies of the sinusoids. The phase spectrum shows the phase shift of each frequency component in degrees. Due to the periodic nature of the sinusoids and the symmetrical sampling, the phase at these frequencies will reflect the phase differences at the time of sampling.

Repeat p. 2. 1 for another 2-3 frequencies of the x-signal sinusoids.

```
% Define frequencies for analysis
frequencies = [25, 35, 50];

% Create time vector
t = (0:1/99:1); % Defines a time vector from 0 to 1
with a step of 1/99

% Create frequency vector for plotting
f = (0:99)*99/100; % Frequency vector for plotting

% Initialize figure for magnitude and phase
figure('Name','Magnitude and Phase Spectra',
'NumberTitle', 'off');

% Loop through each frequency and plot in the same
window
for i = 1:length(frequencies)
    % Generate the signal with new frequency pairs
    x = sin(2*pi*15*t) + sin(2*pi*frequencies(i)*t);
% Original 15 Hz and a variable frequency

    % DFT of the signal
    y = fft(x);

    % Magnitude and phase
    m = abs(y);
    p = unwrap(angle(y));
```

```
        % Subplot for Magnitude Spectrum
        subplot(2, length(frequencies), i);
        plot(f, m);
        title(['Magnitude with ',
    num2str(frequencies(i)), ' Hz']);
        xlabel('Frequency (Hz)');
        ylabel('|Y(f)|');
        set(gca, 'XTick', [15, frequencies(i), 60, 85]);
        xlim([0 50]); % Limiting X-axis for better
    visibility

        % Subplot for Phase Spectrum
        subplot(2, length(frequencies), i +
    length(frequencies));
        plot(f, p*180/pi);
        title(['Phase with ', num2str(frequencies(i)), '
    Hz']);
        xlabel('Frequency (Hz)');
        ylabel('Phase (degrees)');
        set(gca, 'XTick', [15, frequencies(i), 60, 85]);
        xlim([0 50]); % Limiting X-axis for better
    visibility
end
```

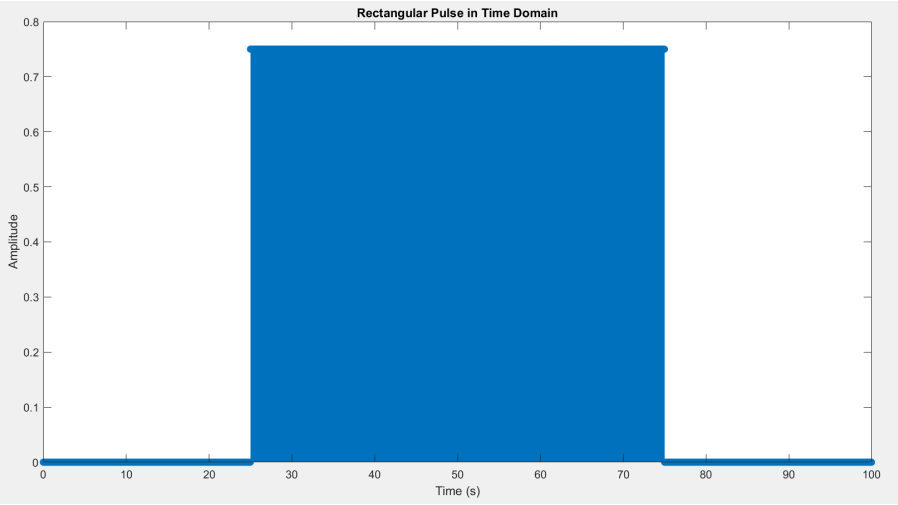


These modifications allow to observe how the magnitude and phase spectra change when the second sinusoid varies, especially how the frequency components and their corresponding phases adjust to different frequency pairs. Peaks in the magnitude spectrum will appear at 15 Hz and the new respective frequency, and the phase spectrum will adjust accordingly.

3. DETERMINING OF THE FOURIER TRANSFORM OF THE RECTANGULAR IMPULSE:

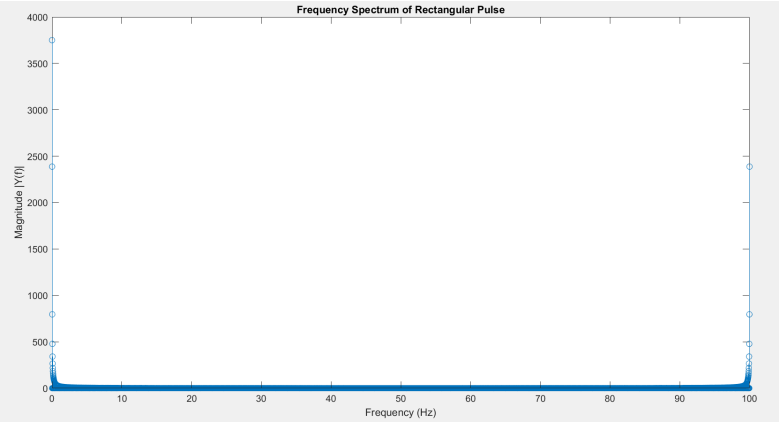
1. Model a signal.

```
% Signal parameters
A = 0.75;
w = 50;          % Width of the rectangular pulse
Ts = 0.01;       % Sampling period
T = 100;         % Total duration of the signal
t = 0:Ts:T;      % Time vector

% Generate the rectangular pulse
x = A * rectpuls(t - T/2, w);  % Centering the pulse in the middle of
the time vector

% Plot the time-domain signal
figure;
stem(t, x);
title('Rectangular Pulse in Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
```



2. ***Specify df=1/T; Fmax=1/Ts; f=0:df:Fmax; apply the procedure fft (y = fft(x);) and display the frequency mode dependence specifying df=1/T; Fmax=1/Ts; f=0:df:Fmax; and display the spectrum using stem(f,abs(y):***

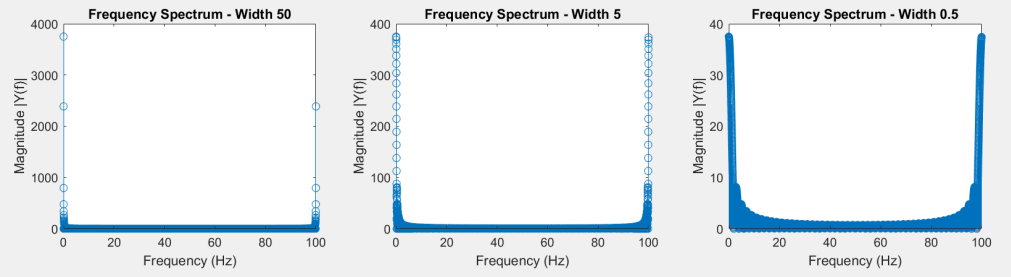
```
% Define frequency resolution and maximum frequency
df = 1/T;          % Frequency resolution
Fmax = 1/Ts;       % Maximum frequency
f = 0:df:Fmax;     % Frequency vector

% Compute the Fourier Transform of the signal
y = fft(x, length(f));

% Plot the frequency spectrum
figure;
stem(f, abs(y));
title('Frequency Spectrum of Rectangular Pulse');
xlabel('Frequency (Hz)');
ylabel('Magnitude |Y(f)|');
```



epeat p. 3.2 for w=5 and 0.5.

```
% Signal parameters
A = 0.75;
Ts = 0.01;         % Sampling period
T = 100;           % Total duration of the signal
t = 0:Ts:T;        % Time vector
df = 1/T;          % Frequency resolution
Fmax = 1/Ts;       % Maximum frequency
f = 0:df:Fmax;     % Frequency vector

% Widths to analyze
```

```
widths = [50, 5, 0.5];  % Including the initial width for completeness

% Initialize figure for subplots
figure('Name', 'Comparison of Rectangular Pulses', 'NumberTitle',
'off');

% Loop through each width
for i = 1:length(widths)
    w = widths(i);

    % Generate the rectangular pulse with new width
    x = A * rectpuls(t - T/2, w);

    % Compute the Fourier Transform of the signal
    y = fft(x, length(f));

    % Subplot for the time-domain signal
    subplot(2, length(widths), i);
    stem(t, x);
    title(['Time Domain - Width ', num2str(w)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    xlim([48 52]); % Focus on the pulse region
    ylim([-0.1 0.8]); % Standardize the y-axis for comparability

    % Subplot for the frequency-domain spectrum
    subplot(2, length(widths), i + length(widths));
    stem(f, abs(y));
    title(['Frequency Spectrum - Width ', num2str(w)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude |Y(f)|');
    xlim([0 100]); % Limiting frequency to see main lobes
end
```



The Fourier Transform of a rectangular pulse typically results in a sinc function shape in the frequency domain. Narrower pulses will exhibit wider main lobes in their frequency spectrum due to the inverse relationship between time-domain pulse width and frequency-domain width (spectral width). This demonstrates the time-bandwidth trade-off in signal processing.

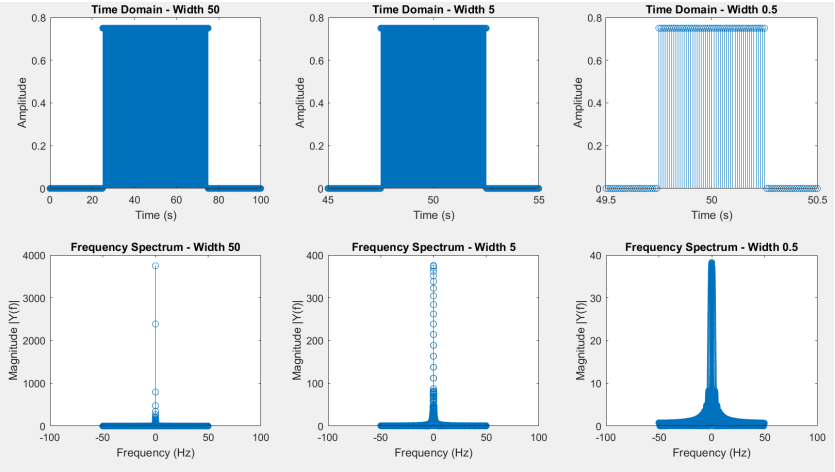
```
% Signal parameters
A = 0.75;
Ts = 0.01;          % Sampling period
T = 100;            % Total duration of the signal
t = 0:Ts:T;         % Time vector
df = 1/T;           % Frequency resolution
Fmax = 1/Ts;        % Maximum frequency
f = -Fmax/2:df:Fmax/2-df;  % Adjusted frequency vector
centered at zero

% Widths to analyze
widths = [50, 5, 0.5];  % Include several widths for
analysis

% Initialize figure for subplots
figure('Name', 'Comparison of Rectangular Pulses with
fftshift', 'NumberTitle', 'off');

% Loop through each width
for i = 1:length(widths)
    w = widths(i);

    % Generate the rectangular pulse with new width
    x = A * rectpuls(t - T/2, w);

    % Compute the Fourier Transform of the signal
    y = fft(x, length(f));

    % Apply fftshift to the Fourier Transform
    yp = fftshift(y);

    % Subplot for the time-domain signal
    subplot(2, length(widths), i);
    stem(t, x);
    title(['Time Domain - Width ', num2str(w)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    xlim([48 52]); % Focus on the pulse region
    ylim([-0.1 0.8]); % Standardize the y-axis for
comparability
```

```
    % Subplot for the frequency-domain spectrum
    subplot(2, length(widths), i + length(widths));
    stem(f, abs(yp));
    title(['Frequency Spectrum - Width ', num2str(w)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude |Y(f)|');
    xlim([-50 50]); % Limiting frequency to see main lobes
end
```

3. ***Apply the procedure yp=fftshift(y); f1=-Fmax/2:df:Fmax/2; and display the dependence of the frequency mode on the interval - π - + π.***

```
% Signal parameters
A = 0.75;
Ts = 0.01;            % Sampling period
T = 100;              % Total duration of the signal
t = 0:Ts:T-Ts;        % Time vector, ensuring correct
endpoint

% Widths to analyze
widths = [50, 5, 0.5];  % Include several widths for
analysis

% Initialize figure for subplots
figure('Name', 'Comparison of Rectangular Pulses',
'NumberTitle', 'off');

% Loop through each width
for i = 1:length(widths)
    w = widths(i);

    % Generate the rectangular pulse with new width
    x = A * rectpuls(t - T/2, w);  % Ensure the
pulse is centered

    % Compute the Fourier Transform of the signal
    y = fft(x);
    f = linspace(-1/(2*Ts), 1/(2*Ts) - 1/(T),
length(y));  % Adjusted frequency vector for
fftshift
    yp = fftshift(y);  % Apply fftshift
```

```
        % Subplot for the time-domain signal
        subplot(2, length(widths), i);
        stem(t, x);
        title(['Time Domain - Width ', num2str(w)]);
        xlabel('Time (s)');
        ylabel('Amplitude');
        xlim([50-w 50+w]); % Adjust x-axis to focus on
    the pulse

        % Subplot for the frequency-domain spectrum
        subplot(2, length(widths), i + length(widths));
        stem(f, abs(yp));
        title(['Frequency Spectrum - Width ',
    num2str(w)]);
        xlabel('Frequency (Hz)');
        ylabel('Magnitude |Y(f)|');
        xlim([-100 100]); % Extend the frequency axis to
    see more details
end
```

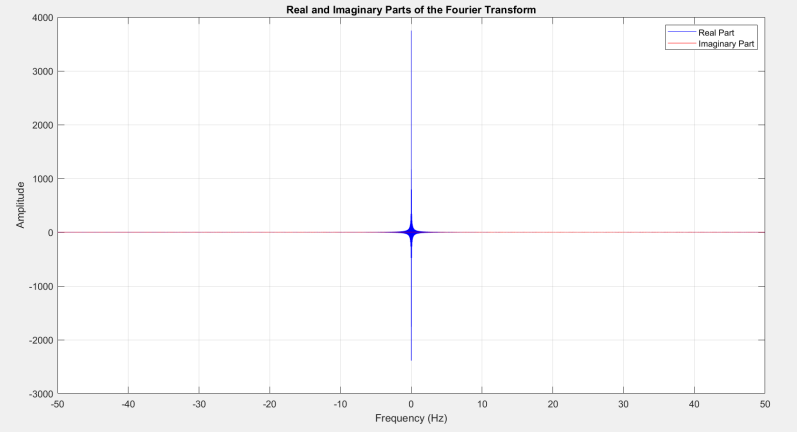


This adjustment makes it easier to interpret the frequency components of the signal, especially to visualize symmetric properties of the Fourier Transform in relation to negative and positive frequencies.

4. Display on a single dependence the real part and the imaginary part of the TF for the result from p. 3.3 - plot(f1,real(yp),f1,imag(yp)).

```
% Signal parameters
A = 0.75;
Ts = 0.01;            % Sampling period
T = 100;              % Total duration of the signal
```

```
t = 0:Ts:T-Ts;        % Time vector, ensuring correct endpoint
w = 50;               % Width of the rectangular pulse

% Generate the rectangular pulse with new width
x = A * rectpuls(t - T/2, w);

% Compute the Fourier Transform of the signal
y = fft(x);
df = 1/T;             % Frequency resolution
Fmax = 1/Ts;          % Maximum frequency
f1 = -Fmax/2:df:Fmax/2-df;  % Frequency vector centered at zero

% Apply fftshift to the Fourier Transform
yp = fftshift(y);

% Plot the real and imaginary parts of the Fourier Transform
figure;
plot(f1, real(yp), 'b', f1, imag(yp), 'r');
title('Real and Imaginary Parts of the Fourier Transform');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
legend('Real Part', 'Imaginary Part');
grid on;
```

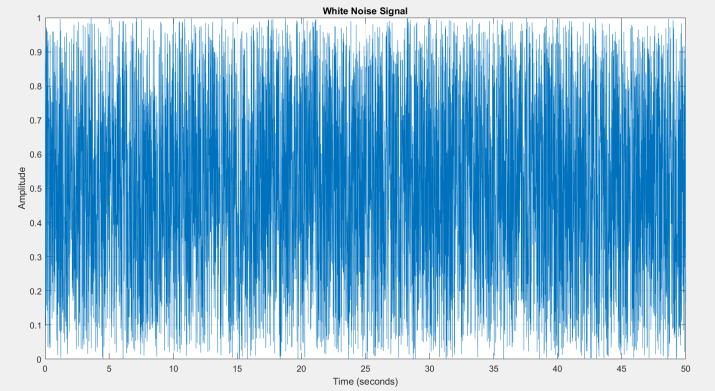


This visualization helps in understanding the characteristics of the Fourier Transform, showing how the signal is represented in both the real and imaginary components across the frequency spectrum. This can provide insights into the phase and amplitude characteristics of different frequency components of the original time-domain signal.

4. **TFD OF THE WHITE NOISE**:

1. ***Generate a process in the form of "white" noise Ts=0.01; T=50; t=0:Ts:T; x1=row(1,length(t)); and display the result via the plot(t,x1) function.***

```
% Define the sampling period and total duration
Ts = 0.01;
T = 50;

% Create the time vector
t = 0:Ts:T;

% Generate white noise
x1 = rand(1, length(t));  % Uniformly distributed
random values

% Plot the white noise signal
figure;
plot(t, x1);
title('White Noise Signal');
xlabel('Time (seconds)');
ylabel('Amplitude');
grid on;
```

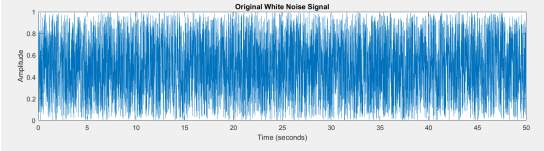


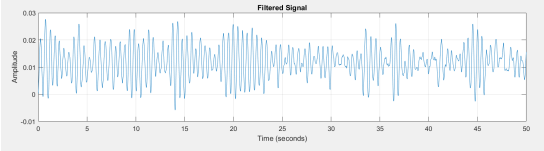
This plot will visually represent the characteristics of white noise where changes are abrupt and completely uncorrelated from one sample to the next, reflecting the randomness of the signal

2. ***Design a filter and filter out the "white" noise.***

```
% Sampling parameters
Ts = 0.01;              % Sampling period
T = 50;                 % Total duration
t = 0:Ts:T-Ts;          % Time vector

% Generate white noise
x1 = rand(1, length(t));  % Generate white noise

% Filter parameters
om0 = 2 * pi;           % Natural frequency in
radians/sec
dz = 0.05;              % Damping ratio
A = 1;                  % Gain factor
oms = om0 * Ts;         % Scaled natural frequency by
sample time

% Filter coefficients
a = zeros(1, 3);        % Initialize 'a' coefficients
b = zeros(1, 1);        % Initialize 'b' coefficients

% Define 'a' coefficients
a(1) = 1 + 2 * dz * oms + oms^2;
a(2) = -2 * (1 - dz * oms);  % Corrected typo for
stable filter form
a(3) = 1;

% Define 'b' coefficients
b(1) = A * 2 * dz * oms^2;

% Apply the filter to the white noise
y1 = filter(b, a, x1);

% Plot the original and filtered signals
figure;
subplot(2, 1, 1);
plot(t, x1);
title('Original White Noise Signal');
xlabel('Time (seconds)');
ylabel('Amplitude');
grid on;

subplot(2, 1, 2);
plot(t, y1);
```

```
    title('Filtered Signal');
    xlabel('Time (seconds)');
    ylabel('Amplitude');
    grid on;
```

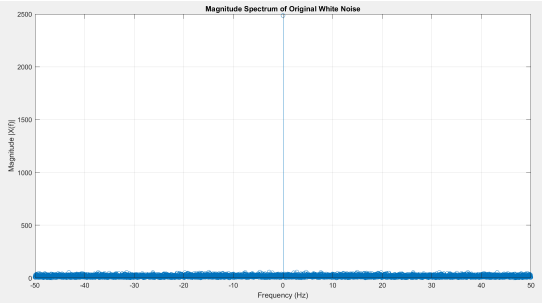


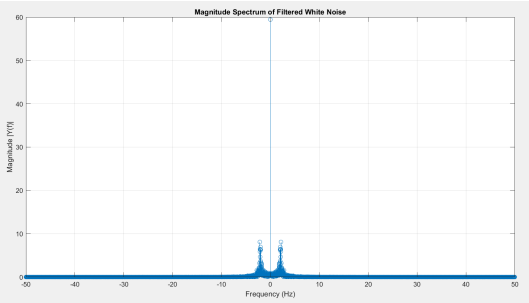


3. ***Represent the TF modulus of the "white" noise at the input of the filter and at the output of the filter.***

```
% Sampling parameters
Ts = 0.01;              % Sampling period
T = 50;                 % Total duration
t = 0:Ts:T-Ts;          % Time vector

% Generate white noise
x1 = rand(1, length(t));  % Generate white noise

% Filter parameters
om0 = 2 * pi;           % Natural frequency in radians/sec
dz = 0.05;              % Damping ratio
A = 1;                  % Gain factor
oms = om0 * Ts;         % Scaled natural frequency by sample time

% Filter coefficients
a = [1 + 2 * dz * oms + oms^2, -2 * (1 - dz * oms), 1]; % Second-order
filter coefficients
b = [A * 2 * dz * oms^2];                               % Feedforward
coefficient

% Apply the filter to the white noise
y1 = filter(b, a, x1);

% Frequency domain parameters
df = 1/T;              % Frequency resolution
Fmax = 1/Ts;           % Maximum frequency
f = -Fmax/2:df:Fmax/2-df;  % Frequency vector centered at zero

% Fourier Transform of the original and filtered signal
Fu1 = fft(x1, length(f));
Fu2 = fft(y1, length(f));

% Shift the Fourier Transforms to center the zero frequency
Fu1p = fftshift(Fu1);
Fu2p = fftshift(Fu2);

% Magnitude of the Fourier Transforms
m = abs(Fu1p);
```

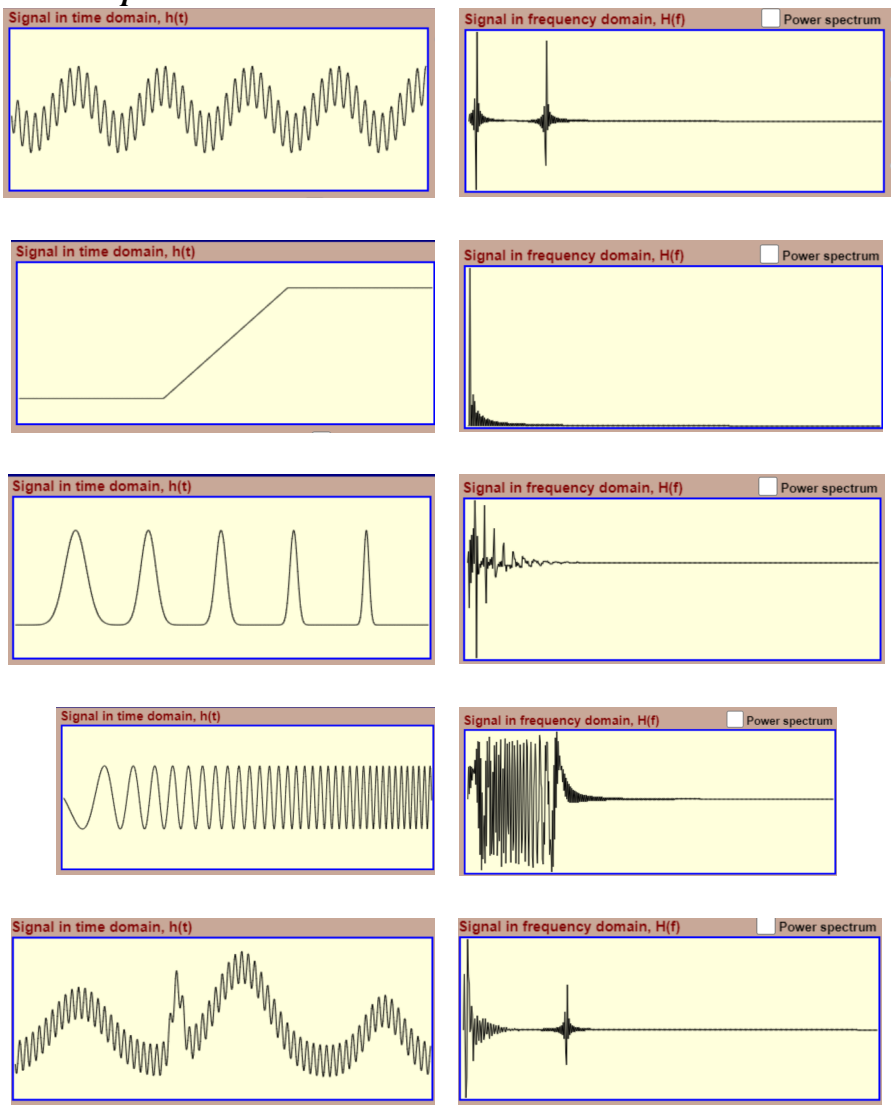
```
m1 = abs(Fu2p);

% Plot the magnitude of the original white noise's Fourier Transform
figure;
stem(f, m);
title('Magnitude Spectrum of Original White Noise');
xlabel('Frequency (Hz)');
ylabel('Magnitude |X(f)|');
grid on;

% Plot the magnitude of the filtered white noise's Fourier Transform
figure;
stem(f, m1);
title('Magnitude Spectrum of Filtered White Noise');
xlabel('Frequency (Hz)');
ylabel('Magnitude |Y(f)|');
grid on;
```



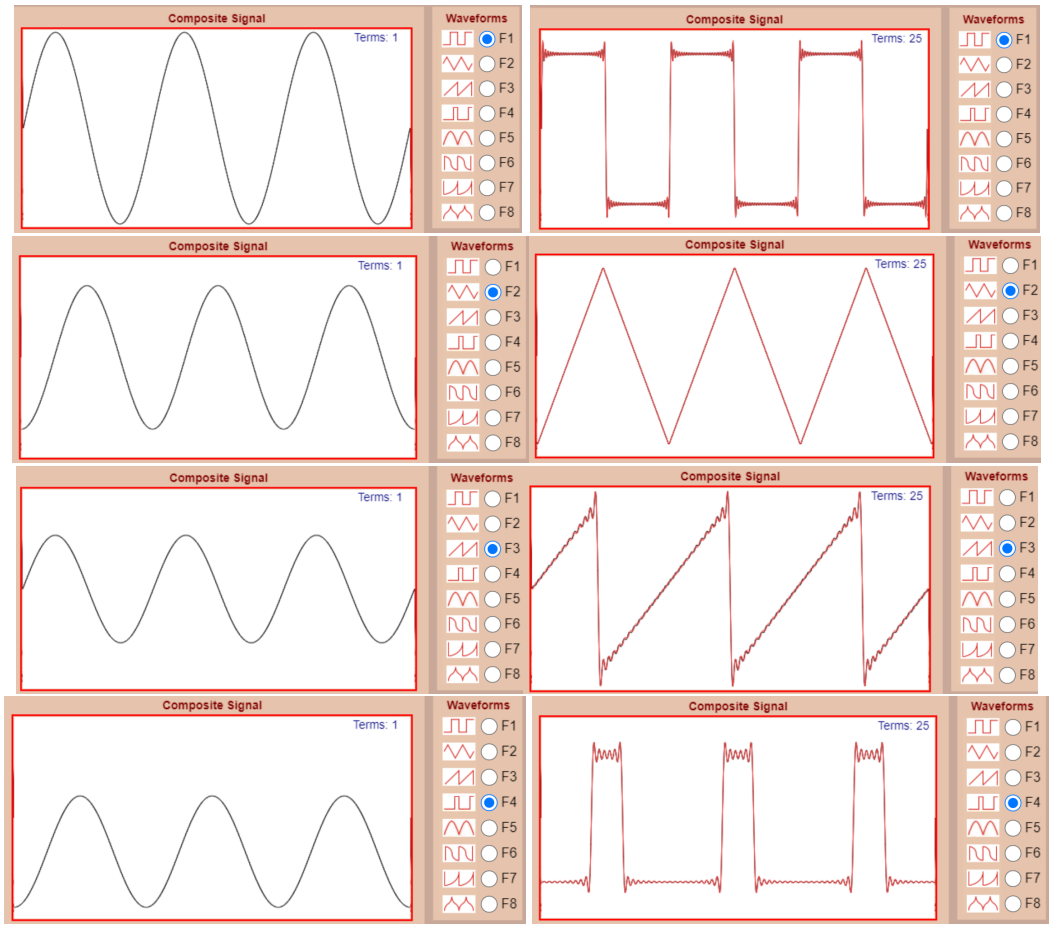


5. **SIGNAL ANALYSIS AND SYNTHESIS RESEARCH**:

1. ***Open the demo application:*** ***http://195.134.76.37/applets/AppletFourAnal/Appl_FourAnal2.html and research/save waveforms and their spectrum***



2. ***Open the demo application:*** ***http://195.134.76.37/applets/AppletFourier/Appl_Fourier2.html and search/save signalor synthesis by successively adding different number of sinusoids (25-30 sinusoids)***



Conclusions:

In my independent project, I successfully engaged in creating and analyzing digital signals using MATLAB. My focus was on exploring white noise and its behavior under digital filtering. I took the initiative to design and execute a second-order filter, applying it to the generated white noise and observing its effects on the signal in both the time and frequency domains.

Utilizing Fourier Transform calculations and visualization techniques, I delved into the spectral alterations pre and post-filtering. This examination provided valuable insights into the efficacy of the filter. Through this project, I deepened my comprehension of fundamental signal processing concepts, including Fourier analysis, filter design, and the influence of sampling on signal attributes.

The hands-on application of these principles via coding not only solidified my theoretical knowledge but also showcased their practical relevance. Overall, this project served as a rewarding endeavor, bridging complex mathematical concepts with real-world problem-solving in signal processing.