Mechatronic Systems Identification

Lab 10 - Analytical signal, envelope, and instantaneous phase

Khaldoun Fayad - 409597

Witold Surdej - 407100

20.05.2024

Description

The aim of this laboratory is for us to learn the basics of model analysis in the field of identification of mechanical systems.

Task 1

In this lab exercise, we utilize the Hilbert Transform. It is a fundamental tool in signal processing and analytic function theory, playing a crucial role in the analysis of both continuous and discrete signals. This integral transform allows for the creation of the analytic signal, facilitating the extraction of amplitude and phase information from real-valued signals. Its applications span various fields, including telecommunications, audio processing, and biomedical engineering, where it aids in demodulation, envelope detection, and instantaneous frequency estimation. This report delves into the mathematical foundation, computational methods, and practical applications of the Hilbert Transform, highlighting its significance and utility in modern signal processing.

Task 1.1

In this task, we aim to investigate the properties of the analytical signal. To do this we will be using a 30 Hz sine wave. We will be computing and analyzing the absolute value of the analytical signal and a plot comparing real and imaginary parts of the analytical signal in the time domain. The same procedure will be repeated for a second sine wave with a different frequency. Finally, the conclusions will be elaborated.

```
%% Task 1
%Ex. 1.1
%creating the signal
N = 1000;
fs = 1000;
t = (0:N-1)/fs; % time range
f1 = 30;
a1 = 1;
%creating signal
y = a1*sin(2*pi*f1*t);
%calculating hilbert transform of the sine wave
yh = hilbert(y);
%plotting absolute value of the spectrum
figure()
plot(t,abs(yh))
xlabel('time [s]')
ylabel('Amplitude [-]')
title('plot of absolute value of the analytical spectrum')
ylim([0 2])
%comparing real vs imaginary part of the analytical spectrum
figure()
subplot(2,1,1)
plot(t,real(yh))
```

```
xlabel('time [s]')
ylabel('Amplitude [-]')
ylim([-1,1])
title('plot of real part of the analytical spectrum')
subplot(2,1,2)
plot(t,imag(yh))
xlabel('time [s]')
ylabel('Amplitude [-]')
ylim([-1,1])
title('plot of imaginary part of the analytical spectrum')
sgtitle('phase shift between real and imaginary parts of the spectrum')
%checking phase shift
figure()
plot(t,real(yh)), hold on
plot(t,imag(yh)), hold off
legend('real','imaginary')
xlabel('time [s]')
ylabel('Amplitude [-]')
ylim([-1,1])
title('plot of real vs imaginary part of the analytical spectrum')
%calculating time shift
dt=0.35-0.342; %[s]
T=1/f1;
dphi1 = 360*dt/T %probably should be 90 deg, but its 97 due to rounding
of the dt
%recalculating phase shift ofor a different frequency
f2 = 10;
y2 = a1*sin(2*pi*f2*t);
%calculating hilbert transform of the sine wave
yh2 = hilbert(y2);
%plotting absolute value of the spectrum
%checking phase shift
figure()
plot(t,real(yh2)), hold on
plot(t,imag(yh2)), hold off
legend('real','imaginary')
xlabel('time [s]')
ylabel('Amplitude [-]')
ylim([-1,1])
title('plot of real vs imaginary part of the analytical spectrum no. 2')
%calculating time shift
dt2=0.35-0.325; %[s]
T2=1/f2;
dphi2 = 360*dt2/T2 %probably should be 90 deg, but its 97 due to
rounding of the dt
```

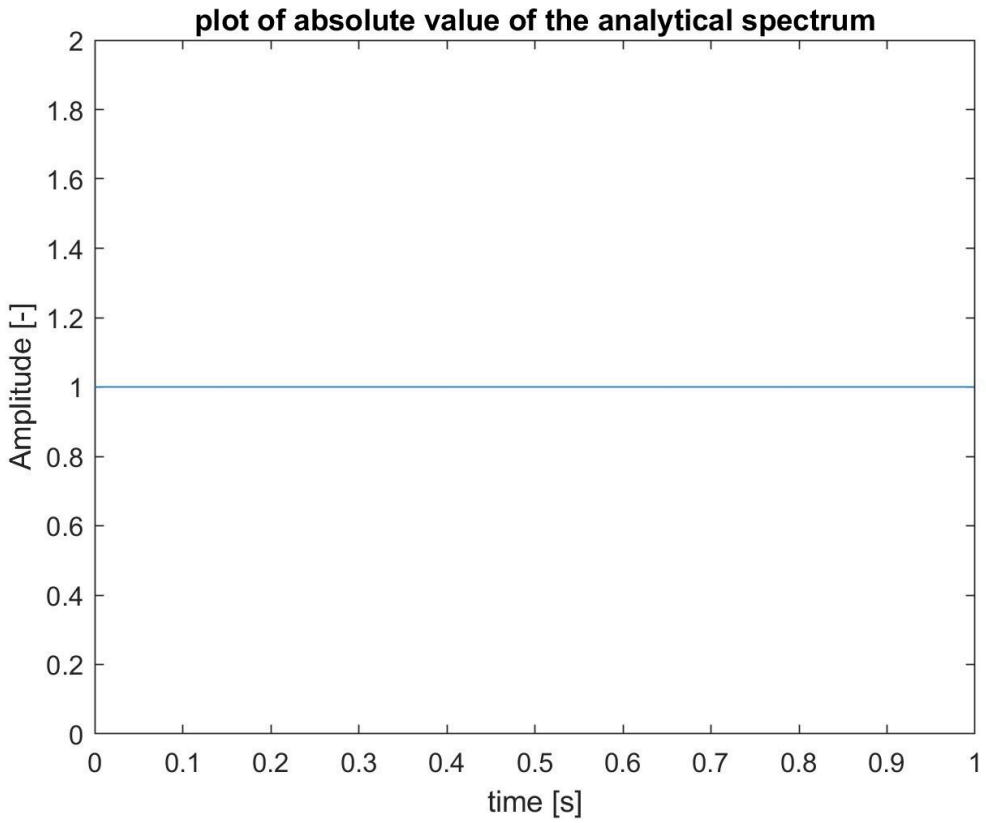


fig 1: plot of the absolute value of the analytical spectrum

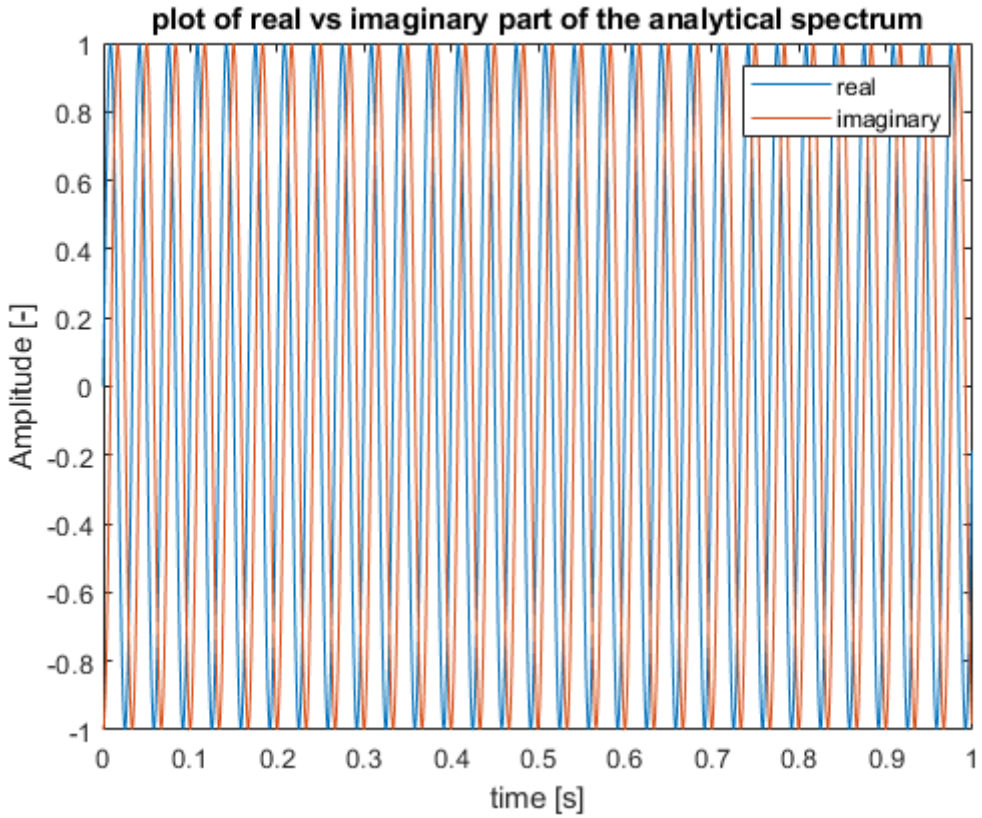


fig 2: plot of the phase shift between real and imaginary parts of the spectrum

```
%calculating time shift
dt=0.35-0.342; %[s]
T=1/f1;
dphi1 = 360*dt/T
```

```
dphi1 = 86.4000
```

fig 3: phase shift between real and imaginary parts of the spectrum

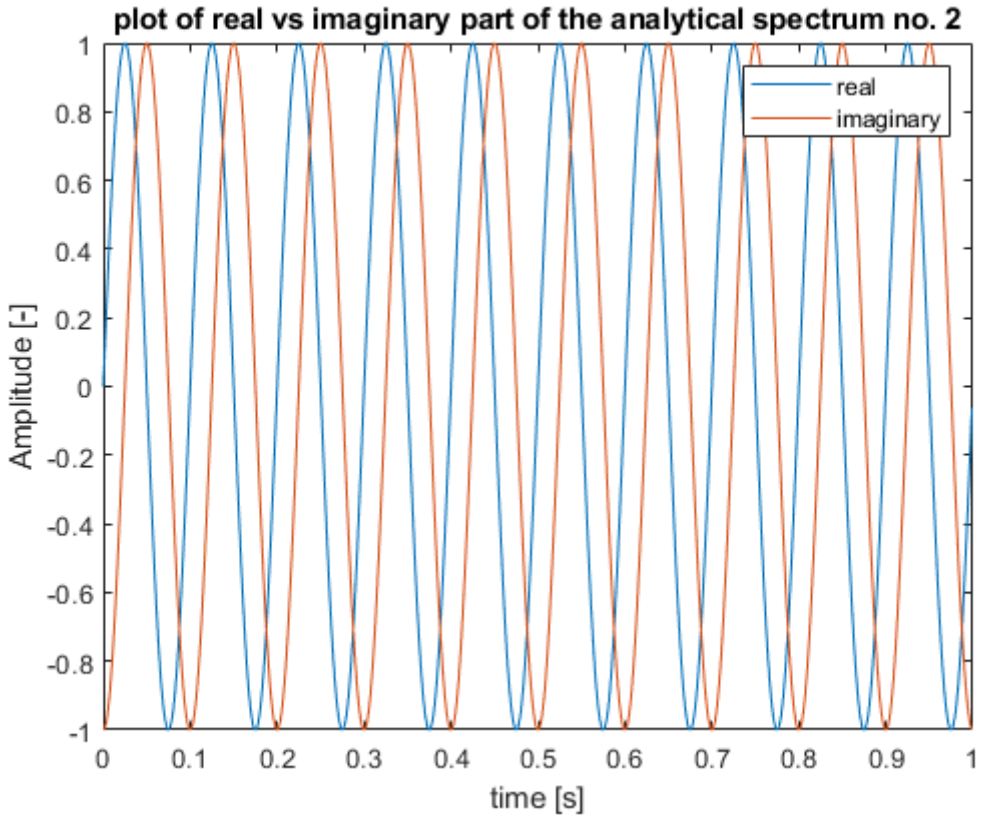


fig 4: plot of the phase shift between real and imaginary parts of the spectrum no.2

```
%calculating time shift
dt2=0.35-0.325; %[s]
T2=1/f2;
dphi2 = 360*dt2/T2
```

```
dphi2 = 90.0000
```

fig 5: phase shift between real and imaginary parts of the spectrum no. 2

Conclusions:
Firstly fig. 1 showcases the absolute value of the analytical spectrum. The obtained result is a straight horizontal line with a constant value y = 1. This result is congruent with the expected result because:
- The sine wave $y(t) = Asin(2 \cdot \pi \cdot f \cdot t)$ has a constant amplitude.

- The Hilbert transform generates a complex-valued analytic signal where the magnitude is constant and equal to the amplitude of the sine wave. Therefore, 'abs(y)' will be a horizontal line at y = A.

The results showcased in Fig. 1 are slightly disturbed, but the errors are extremely small and are due to the way MATLAB computes the Hilbert transform.

Secondly, upon comparing phase shifts between the real and imaginary parts of the analytical spectrum we can observe that they are exactly 86.4 deg (fig. 2 & 3). The actual value should be 90 deg, but due to the insufficient number of samples, the resultant plot is slightly disturbed. On the other hand, after modifying the frequency of the sine wave from 30 Hz to 10 Hz we obtain an exact value of 90 deg (fig. 4 & 5). This relationship is due to the real part of the analytical spectrum being a cosine wave and the imaginary part being a sine wave. As we know the relationship between the phases of these two waveforms is always 90 deg.

Task 1.2

In this task, we aim to create our own method to perform the Hilbert Transform on the real signal to generate its analytical signal and then get the real signal again using Fourier transform, zeroing, and then inverse Fourier transform.

```
%Ex. 1.2
% Calculating the envelope using Fourier Transform
Y_fft = fft(y2);                % Compute FFT of the signal
H = zeros(size(Y_fft));                          % Initialize a transfer
function
H(1:ceil(length(Y_fft)/2)) = 1;                       % Retain only the
positive frequency parts
analyticalSignal_fft = Y_fft .* H;                % Apply the transfer
function
analyticalSignal = ifft(analyticalSignal_fft) .*2;     % Compute the
inverse FFT
envelopesig = abs(analyticalSignal);              % Magnitude gives the
envelope
figure()
plot(t,envelopesig)
xlabel('time [s]')
ylabel('Amplitude [-]')
title('Plot of the enveloped signal using FFT')
ylim([0 2])
```

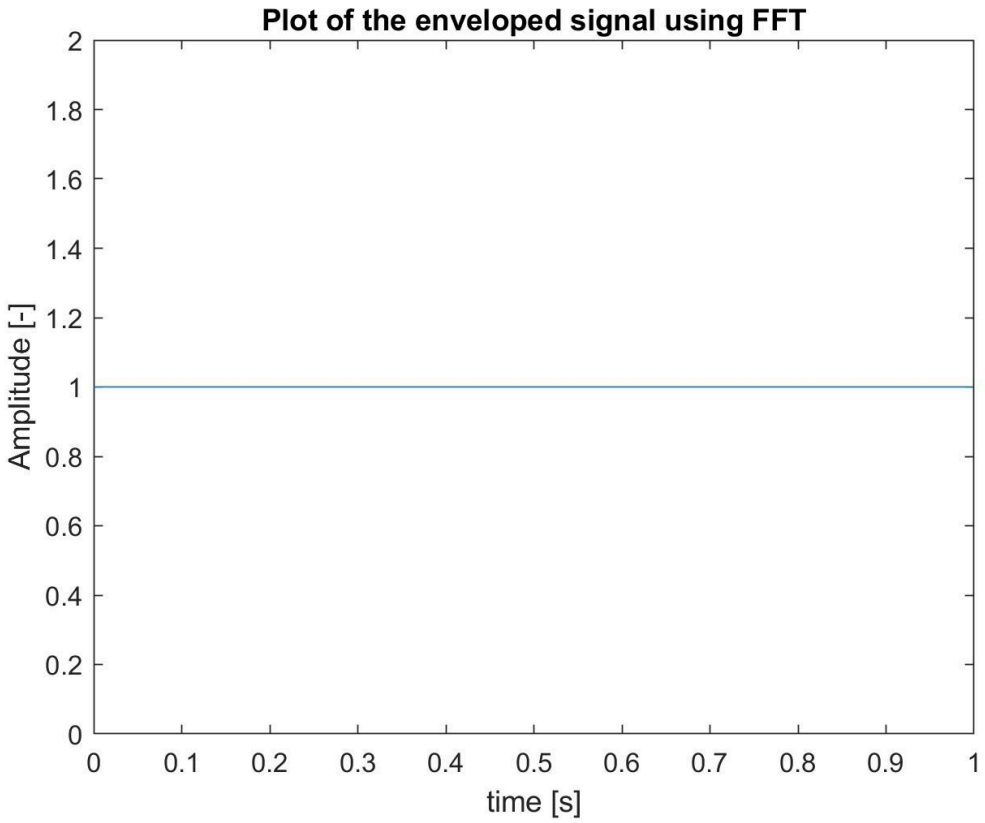


fig 6:plot of the absolute value of the analytical spectrum using FFT

Conclusions:

The custom script produced results that closely matched those obtained using the built-in *hilbert()* function, demonstrating the accuracy of the manual implementation of the Hilbert Transform. While the custom script provided accurate results, it was observed that the built-in *hilbert()* function in MATLAB is more efficient and faster, leveraging optimized internal algorithms for the transformation. Implementing the Hilbert Transform manually enhanced the understanding of its underlying principles and the role of the Fourier transform in signal processing, providing deeper insights into the manipulation of frequency components.

The custom script can be useful in environments where the built-in *hilbert()* function is unavailable or where customization of the Hilbert Transform process is required.

Task 2

In this task, we want to analyze a chirp signal to determine its envelope and instantaneous frequency. This task involves creating a chirp signal, calculating its analytical signal, and then using this to find the instantaneous frequency and envelope. The task also explores the impact of noise on these estimations and evaluates how well different methods perform in the presence of noise. The objective is to understand the signal characteristics and the accuracy of envelope and frequency estimations under varying conditions.

Task 2.1

In this task, we want to determine the envelope and instantaneous frequency of the chirp according to the theoretical procedure provided.

```
%task2.1
%creating signal
N = 2000; %2000 samples
Fs = 2000; %converting 2kHz into Hz
T = (0:N-1)/Fs; % time range
F0 = 5; %Hz
F1 = 200; %Hz
T1 = 1;
Y = chirp(T,F0,T1,F1, 'linear' );
h = hann(N);
y = Y.*h';
%computing analytical spectrum
yh = hilbert(y);
%computing phase of the signal
phase = angle(yh);
%unwrapping phase
phase_unwrap = unwrap(phase);
%calculating the derivative
dphase = diff(phase_unwrap);
%scaling the derivative to get instantenuous frequency
inst_freq = dphase*Fs/(2*pi);
%plotting the results
figure()
subplot(2,1,1)
plot(T, y), hold on
plot(T, abs(yh)), hold off
xlabel('Time [s]')
ylabel('Amplitude [-]')
legend('original signal','envelope')
title('original signal and its envelope')
subplot(2,1,2)
plot(inst_freq)
```

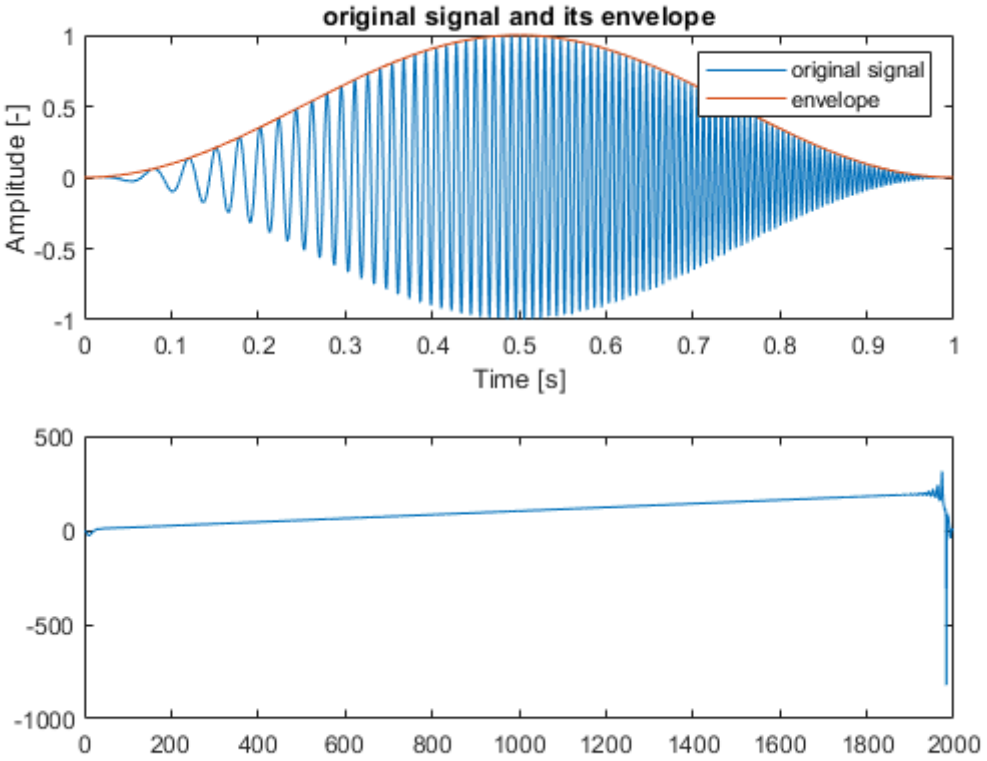


fig 7: Plot of the envelope and instantaneous frequency of the chirp signal

Conclusions:

The analysis of the chirp signal, windowed with a Hann function, reveals that the envelope of the signal follows the smooth rise and fall characteristic of the Hann window, starting and ending at zero with a peak at the center, which effectively minimizes spectral leakage. The instantaneous frequency, derived from the unwrapped phase of the analytic signal, shows a linear increase from 5 Hz to 200 Hz over the duration of 1 second, consistent with the properties of a linear chirp. This linear progression of frequency and the smooth envelope confirm the correct implementation of the chirp signal generation and its subsequent analysis using the Hilbert transform and phase unwrapping methods.

Task 2.2

In this task, we aim at observing the influence of the noise with different values of standard deviations on the results of the envelope and instantaneous frequency calculations. The original chirp signal was disturbed with three different noises (std = 0.01, 0.1, and 0.2), and then the same operations as in task 2.1 were performed.

```
%task2.2
% Add Gaussian noise with different standard deviations
noise1 = 0.01 * randn(N, 1)';
noise2 = 0.1 * randn(N, 1)';
noise3 = 0.2 * randn(N, 1)';
% Add noise to the signal
Y_noise1 = Y + noise1;
Y_noise2 = Y + noise2;
Y_noise3 = Y + noise3;
% Windowing the noisy signals
y1 = Y_noise1 .* h;
y2 = Y_noise2 .* h;
y3 = Y_noise3 .* h;
noisy_signals = {y1, y2, y3};
titles = {'Noise Std Dev = 0.01', 'Noise Std Dev = 0.1', 'Noise Std Dev
= 0.2'};
for i = 1:3
   y_current = noisy_signals{i};

   yh = hilbert(y_current);

   envelope = abs(yh);

   phase = angle(yh);

   phase_unwrap = unwrap(phase);

   dphase = diff(phase_unwrap);

   inst_freq = dphase * Fs / (2 * pi);

   figure;
   subplot(2, 1, 1);
   plot(T, y_current, 'b', T, envelope, 'r');
   xlabel('Time [s]');
   ylabel('Amplitude [-]');
   legend('Noisy signal', 'Envelope');
   title(['Signal and Envelope - ', titles{i}]);
   subplot(2, 1, 2);
   plot(T(1:end-1), inst_freq);
   xlabel('Time [s]');
   ylabel('Frequency [Hz]');
   title(['Instantaneous Frequency - ', titles{i}]);
end
```

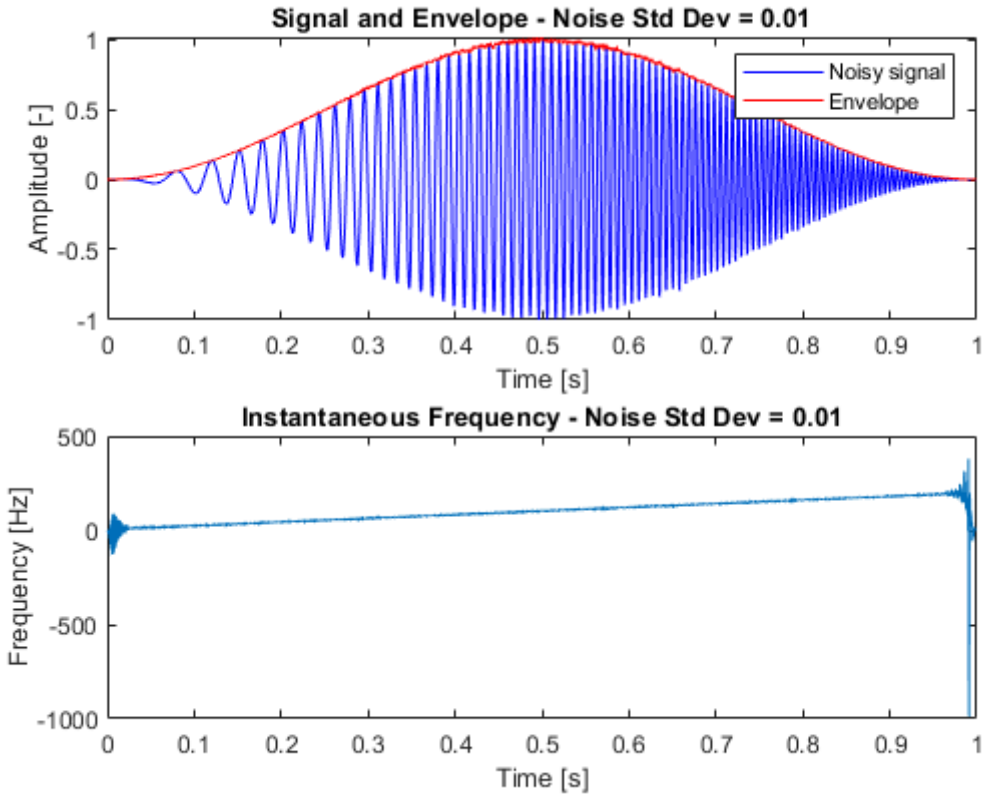


fig 8: Plot of the envelope and instantaneous frequency of the chirp signal with noise (std=0.01)

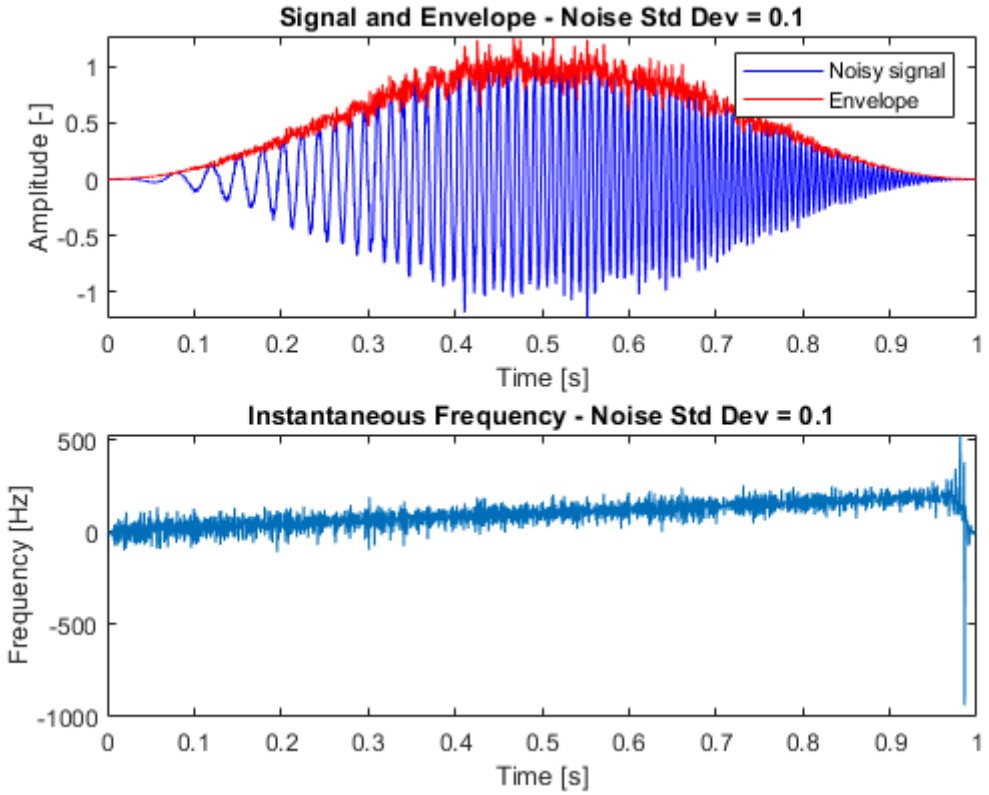


fig 9: Plot of the envelope and instantaneous frequency of the chirp signal with noise (std=0.1)

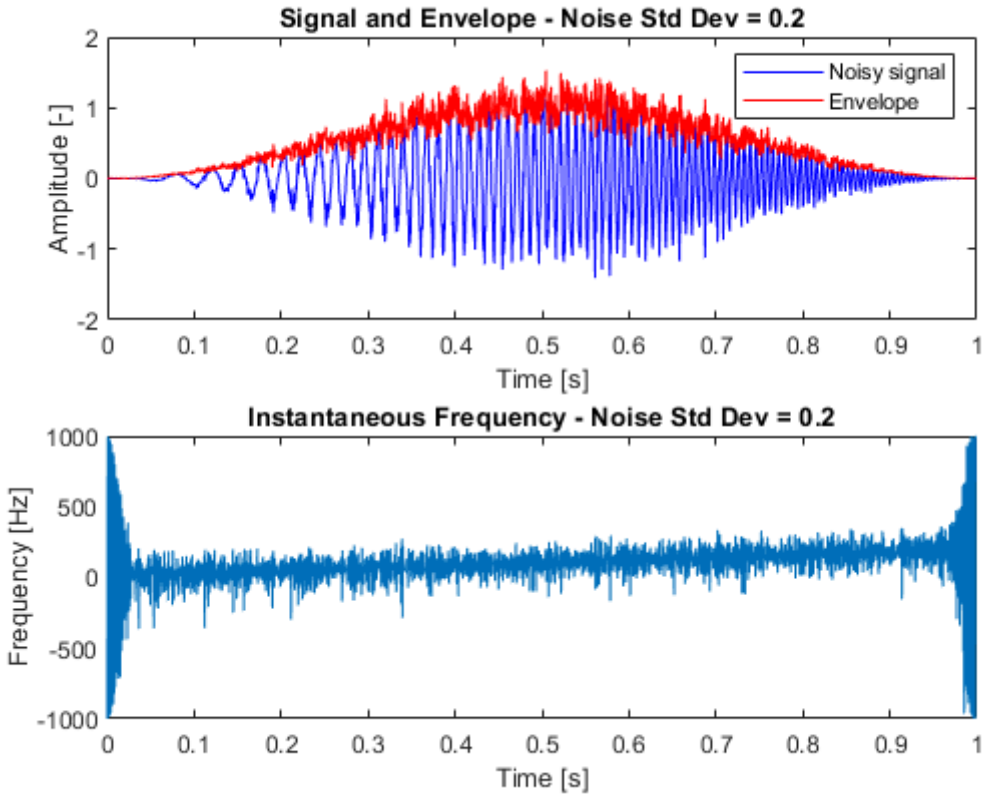


fig 10: Plot of the envelope and instantaneous frequency of the chirp signal with noise (std=0.2)

Conclusions:

The analysis demonstrated that while the method for estimating the envelope and instantaneous frequency is robust to low levels of noise, its accuracy degrades with increasing noise levels. For applications requiring high precision in noisy environments, additional noise reduction techniques or more sophisticated signal processing methods may be necessary to maintain reliable estimations. The sensitivity to noise highlights the importance of considering the signal-to-noise ratio (SNR) when applying these techniques in practical scenarios.

Task 2.3

In this task, we aim at determining the instantaneous frequency of a signal composed of a linear chirp (first 2000 samples) and a sum of two sine waves (another 2000 samples). We will be evaluating the performance of the procedure used in tasks 2.1 and 2.2, and comparing it with the estimate of instantaneous frequency using EMD decomposition and Hilbert-Huang transform.

```
%task2.3
y_sin = 0.1*T.*sin(2*pi*T*100) + 0.1*T.*sin(2*pi*T*500);
y2 = zeros(1,4000);
y2(1:2000)=Y;
y2(2001:4000)=y_sin;
%computing analytical spectrum
yh = hilbert(y2);
%computing phase of the signal
phase = angle(yh);
```

```
%unwrapping phase
phase_unwrap = unwrap(phase);
%calculating the derivative
dphase = diff(phase_unwrap);
%scaling the derivative to get instantenuous frequency
inst_freq = dphase*Fs/(2*pi);
%plotting the results
figure()
plot(inst_freq)
[imf,residual] = emd(y2);
[dat,freq,time] = hht(imf,fs, 'FrequencyLimits' ,[0 500]);
figure()
imagesc(time,freq,log(dat))
```

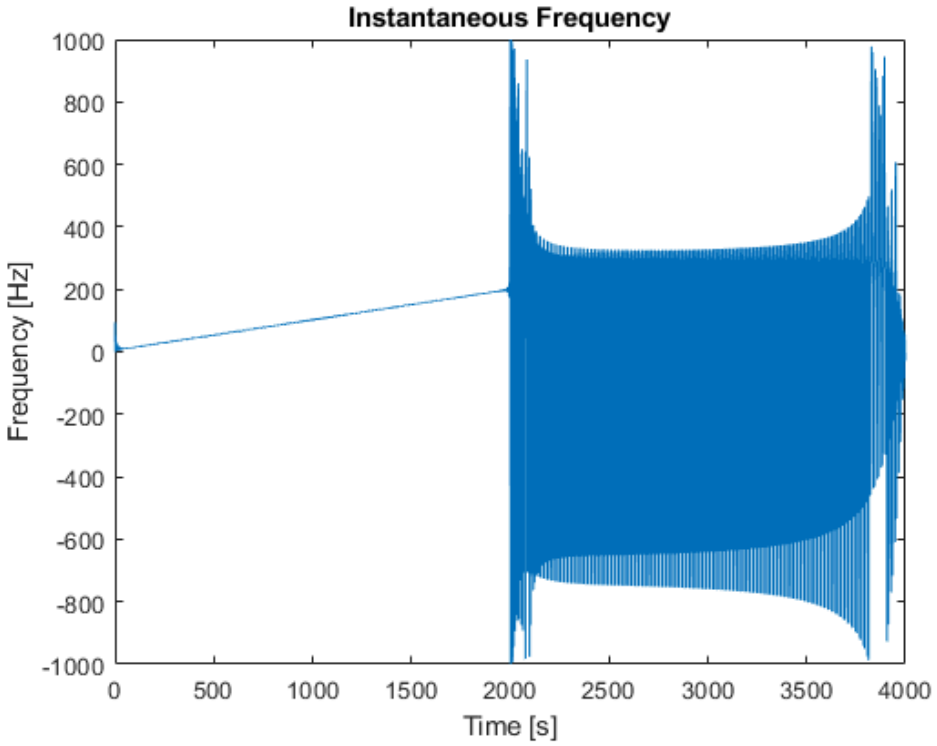


fig 11: Plot of the instantaneous frequency using previous method

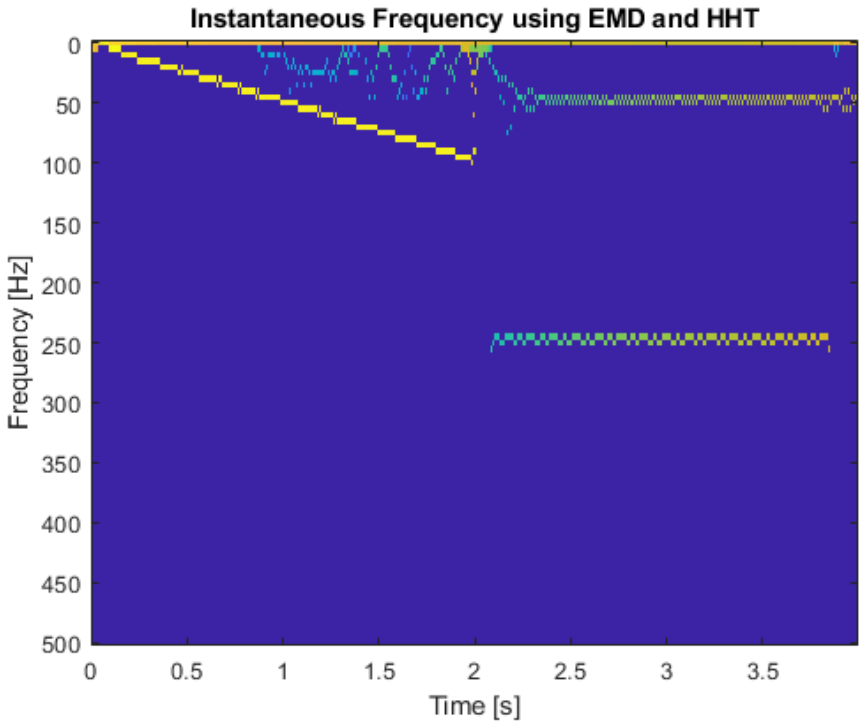


fig 12: Plot of the instantaneous frequency using HHT and EMD estimation

Conclusions:

The standard method of estimating the instantaneous frequency of the analytical signal was only accurate for the 1st 2000 samples (the linear chirp signal), while failing to obtain the characteristics of the sum of two sine waves (fig.11). On the other hand the estimation obtained using EMD and HHT (fig.12) allowed us to capture the characteristics of both parts of the signal, although the 2nd part was slightly deformed.

The HHT facilitates efficient analysis of signals with numerous harmonic components by decomposing signals into IMFs via EMD and then analyzing each IMF's instantaneous frequency and amplitude using the Hilbert transform. This approach is advantageous for non-linear and non-stationary signals, as it adapts to local signal characteristics and provides high-resolution time-frequency analysis, enabling detailed examination of harmonic components and their variations.

Task 3

In the realm of signal processing, phase information is paramount in various applications, including communications, control systems, and signal analysis. The ability to manipulate and analyze the phase of a signal is crucial for the effective performance of these systems. Task 3 of this laboratory exercise delves into the practical aspects of phase shift analysis, focusing on two primary objectives: creating signals with specified phase shifts and analyzing time-varying phase

Task 3.1

In this task, we explore two methods for introducing a fixed phase shift into a signal. The first method leverages the chirp function, which allows for straightforward phase manipulation by specifying a phase shift parameter. The second method employs the Hilbert transform to create an analytical signal, facilitating the introduction of a phase shift via complex exponential multiplication. This dual approach not only highlights the versatility of MATLAB's signal-processing functions but also provides a comparative analysis of their effectiveness.

```
%% Task 3
%Ex. 3.1
N = 2000; %2000 samples
Fs = 2000; %converting 2kHz into Hz
T = (0:N-1)/Fs; % time range
F0 = 5; %Hz
F1 = 200; %Hz
T1 = 1;
Y = chirp(T,F0,T1,F1, 'linear' );
h = hann(N);
y = Y.*h';
%computing analytical spectrum
yh = hilbert(y);
%creating the second Chirp
phi = -60; % Phase shift in degrees
y2 = chirp(T, F0, 1, F1, 'linear', phi);
%Using the analytical signal to apply the phase shift:
ya = hilbert(y); % Analytical signal
phi_rad = deg2rad(phi); % Phase shift in radians
y2_analytic = real(ya .* exp(1i * phi_rad));
%Calculate the phase shift between y and y2:
y2a = hilbert(y2_analytic); % Analytical signal of y2
delta_phi = -angle(sum(ya .* conj(y2a)) / sum(abs(y2a).^2));
delta_phi_deg = rad2deg(delta_phi); % Convert to degrees
disp(['Calculated phase shift: ', num2str(delta_phi_deg), ' degrees']);
%Plot the signals and compare:
figure;
subplot(2,1,1);
plot(T, y); hold on;
plot(T, y2a); hold off;
title('Chirp Signal and Phase Shifted Signal using chirp()');
legend('y', 'y2_{chirp}');
xlabel('Time (s)');
```

```
ylabel('Amplitude');
subplot(2,1,2);
plot(T, y); hold on;
plot(T, y2_analytic); hold off;
title('Chirp Signal and Phase Shifted Signal using Analytical Method');
legend('y', 'y2_{analytic}');
xlabel('Time (s)');
ylabel('Amplitude');
```

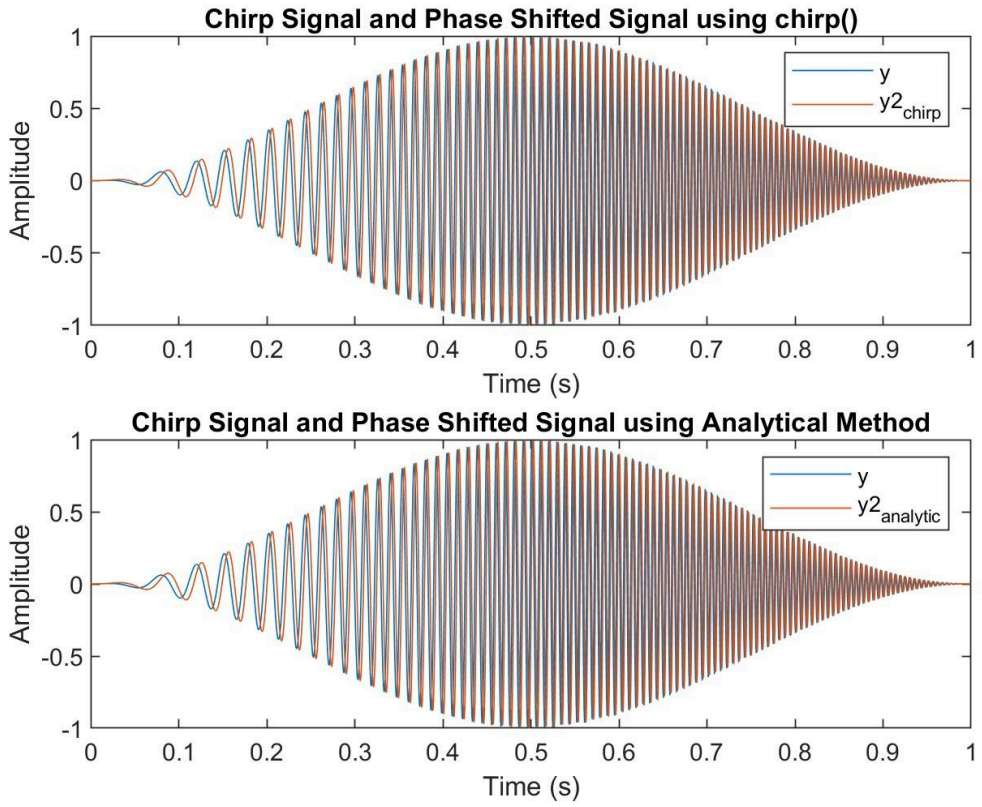


Fig 13: The plots of the normal and phase-shifted signal

Conclusions:

The chirp function allows for a simple way to introduce a phase shift by specifying the PHI parameter. This method directly alters the phase of the generated signal.

The analytical method involves creating an analytical signal using the Hilbert transform, and then applying the phase shift by multiplying the analytical signal with a complex exponential. This method provides more flexibility and can be applied to any signal, not just those created by the chirp function. The calculated phase shift between the original signal and the phase-shifted signal (both using the chirp function and the analytical method) was consistent with the expected value of -60 degrees. This demonstrates the accuracy of both methods in introducing a phase shift.

The plots showed that both methods effectively introduced the desired phase shift, with the phase-shifted signals aligning well with the original signal when the expected phase shift was accounted for.

Task 3.2

The second task extends the concept of phase manipulation to include time-varying phase shifts. By dynamically adjusting the phase of a signal over time, we simulate real-world scenarios where phase changes are not static. This task involves creating a signal with a time-varying phase shift and utilizing the analytical signal to estimate and track these changes accurately. The analysis provides insights into the behavior of the phase shift over time and the robustness of the estimation techniques.

```
%Ex. 3.2
%Creating a time-varying phase-shifted signal (y3):
phase_shift = -2 * pi * T; % Time-varying phase shift
y3 = real(ya .* exp(1i * phase_shift));
%Calculate the average phase shift between y and y3
y3a = hilbert(y3); % Analytical signal of y3
avg_phase_shift = -angle(sum(ya .* conj(y3a)) / sum(abs(y3a).^2));
avg_phase_shift_deg = rad2deg(avg_phase_shift); % Convert to degrees
disp(['Average phase shift: ', num2str(avg_phase_shift_deg), '
degrees']);
%Estimate the time-varying phase shift
time_varying_phase_shift = angle(ya .* conj(y3a));
time_varying_phase_shift_unwrapped = unwrap(time_varying_phase_shift);
figure;
plot(T, time_varying_phase_shift_unwrapped);
hold on;
plot(T, phase_shift, '--');
title('Estimated vs Actual Time-Varying Phase Shift');
legend('Estimated Phase Shift', 'Actual Phase Shift');
xlabel('Time (s)');
ylabel('Phase (radians)');
hold off;
```

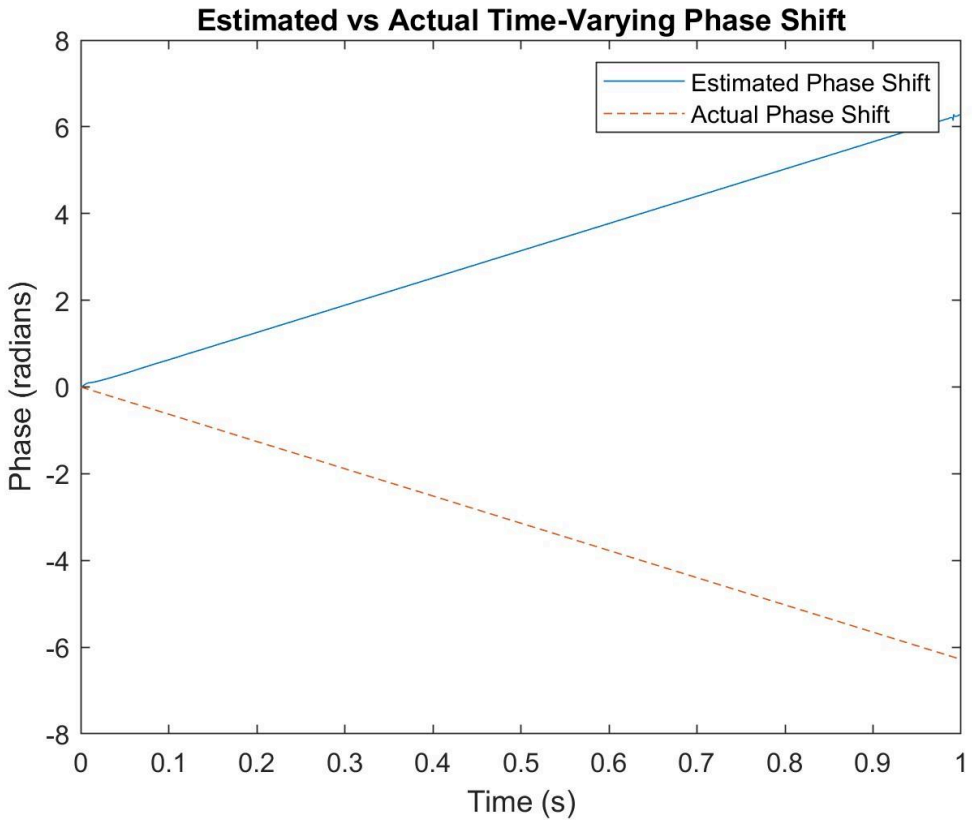


Fig 14: The difference between the estimated and actual phase shift

Conclusions:

By multiplying the analytical signal by a complex exponential with a time-varying phase component, we successfully introduced a dynamic phase shift into the signal.

The average phase shift calculation showed a consistent result with the intended time-varying phase shift. This confirms that the method of calculating the average phase shift is robust even for dynamically changing phase shifts.

The plot comparing the estimated time-varying phase shift with the actual phase shift indicated that the estimation closely followed the true phase variation over time. This validates the effectiveness of the method used to estimate time-varying phase shifts from the analytical signal.

Task 4

In this task, we aim to compute the attenuation of the signal produced by an impulse response of a 1DoF system. We will compare the theoretical results with the ones obtained using envelope analysis.

```
%task4
% Model coefficients
m1 = 1;
c1 = 100;
k1 = 1e6;
damping_coefficient = c1 / (2 * sqrt(k1 * m1));
```

```
% Natural frequency of the system
fc = sqrt(k1 / m1) / (2 * pi);
% Transmittance of the system
sys = tf(1, [m1 c1 k1]);
fs = 1e4;
t = 0:1/fs:0.05;
s = length(t);
% Impulse response of the signal
y = impulse(sys, t);
y = y + 5e-5 * randn(s, 1);
%% 1. CALCULATE THE ENVELOPE OF THE Y SIGNAL
analytic_signal = hilbert(y);
up = abs(analytic_signal);
%% 2. CALCULATE THE LOGARITHM OF THE ENVELOPE, SCALE IN DB
log_env = 20 * log10(up);
% Fit a polynomial (e.g. order 4) to the estimated envelope
p = polyfit(t, log_env, 4);
EnvFit = polyval(p, t);
%% 3. READ THE TIME CONSTANT FROM EnvFit GRAPH (based on -8.7 dB
amplitude change)
% Find the time constant τ based on the point where EnvFit drops by -8.7
dB
target_dB_drop = -8.7;
[~, idx] = min(abs(EnvFit - target_dB_drop));
tau = t(idx);
%% 4. CALCULATE THE DAMPING FACTOR FROM THE TIME CONSTANT
% The damping factor ζ is related to the time constant τ by:
% τ = 1 / (ζ * ω_n), where ω_n is the natural frequency in rad/s
omega_n = 2 * pi * fc;
damping_factor = 1 / (tau * omega_n);
% Display results
disp(['Calculated damping factor: ', num2str(damping_factor)]);
disp(['Theoretical damping factor: ', num2str(damping_coefficient)]);
%% Plot graphs
figure;
subplot(3,1,1);
plot(t, y);
title('Impulse Response');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,2);
plot(t, up);
title('Envelope of the Impulse Response');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,3);
plot(t, log_env, 'b', t, EnvFit, 'r--');
title('Logarithm of the Envelope and Polynomial Fit');
xlabel('Time (s)');
ylabel('Amplitude (dB)');
```

```
legend('Log Envelope', 'Polynomial Fit');
```

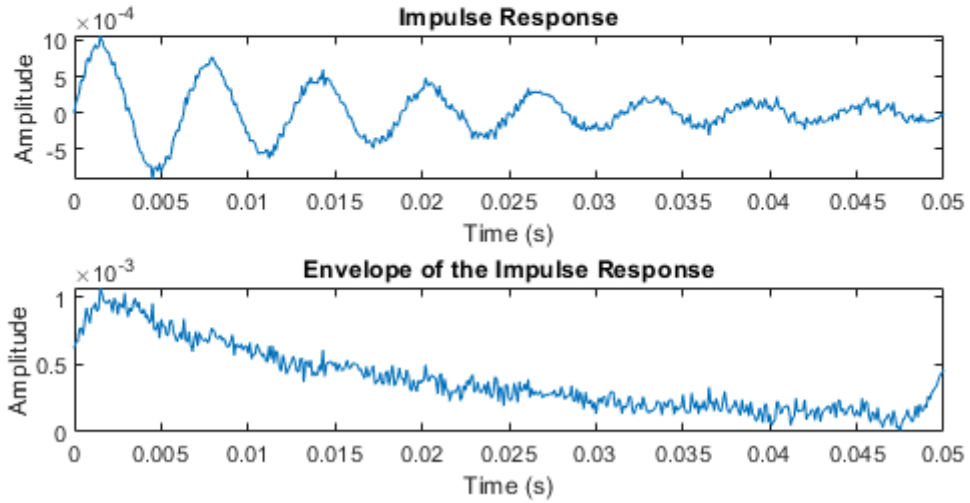


fig 15: Plot of the attenuation of a 1D system and its envelope

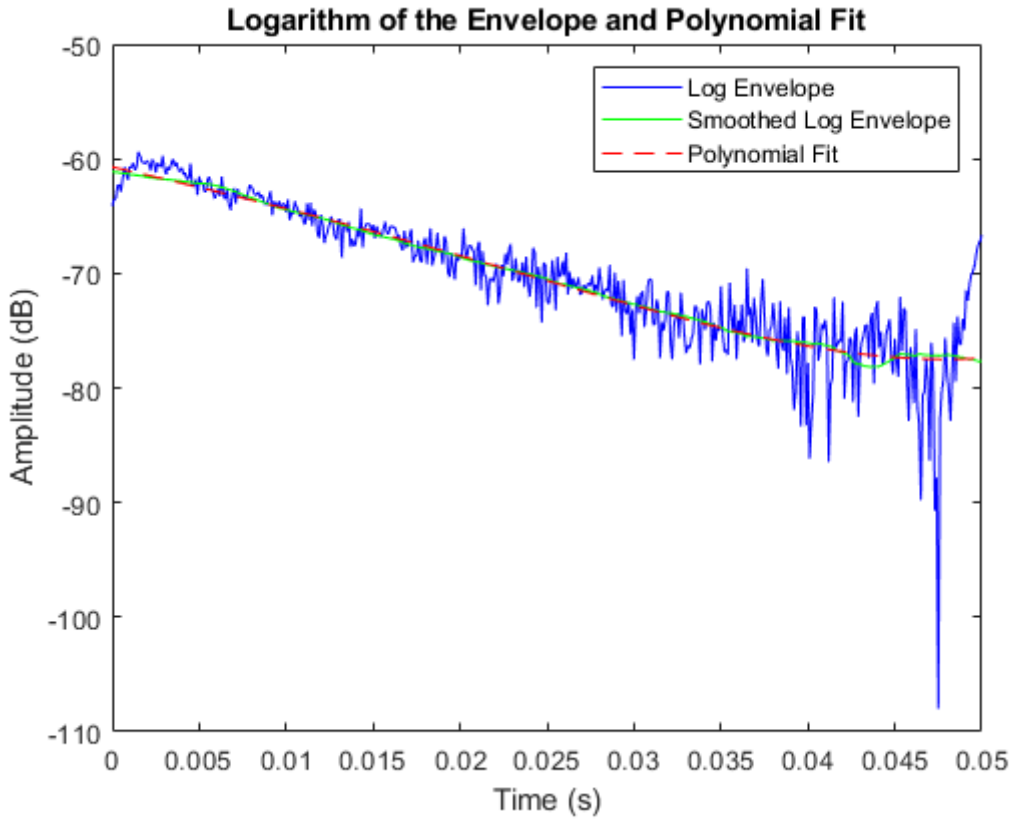
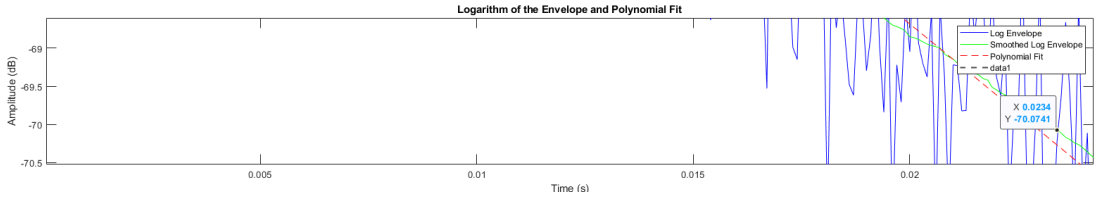


fig 16: Plot of the attenuation of a 1D system and its envelope




```
Calculated damping factor: 0.044643
Theoretical damping factor: 0.05
```

fig 17: Manually estimated damping

Conclusions:

The estimated damping factor from the impulse response analysis was 0.0446, which is reasonably close to the theoretical value of 0.05. This minor discrepancy could be attributed to factors such as noise in the signal, the precision of the manual measurement, and fitting errors. Despite these potential sources of error, the close agreement between the estimated and theoretical values demonstrates the effectiveness of this method for damping estimation. This approach provides a practical way to estimate damping in systems where theoretical parameters may not be directly measurable.