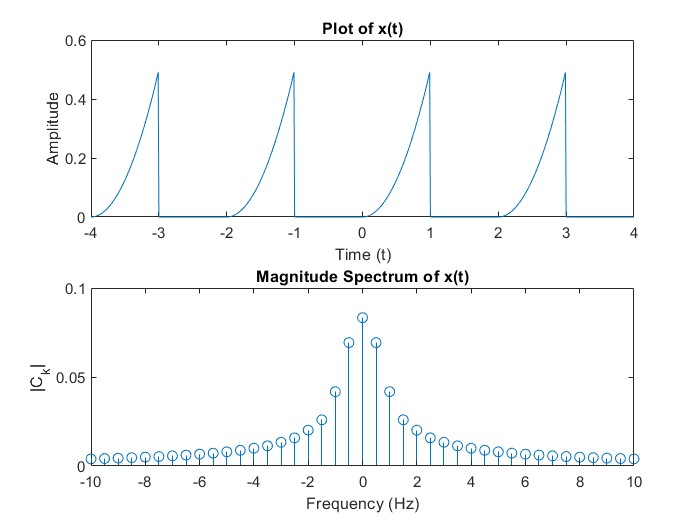
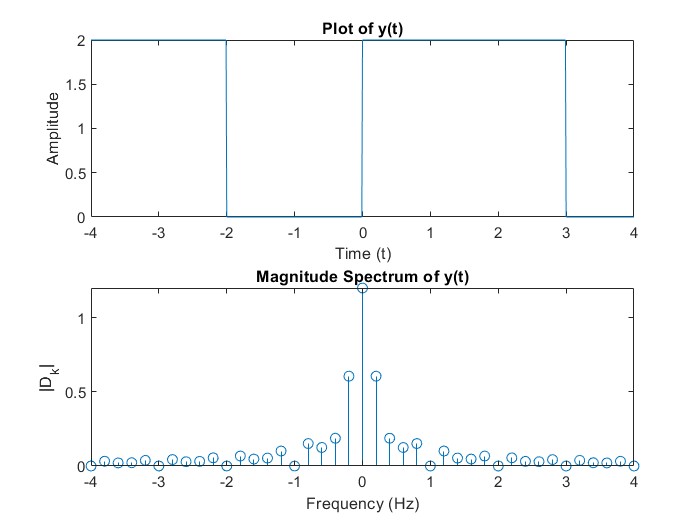
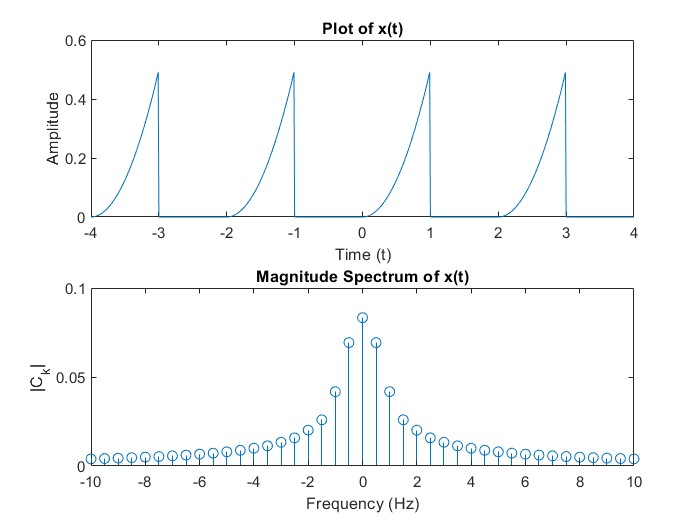
SSRP Assignment 2

**Question1**.  
plots:







Explanation of the code:  
In this task, I worked on analyzing periodic signals and their Fourier series representations. The problem involved three signals: x(t), y(t), and z(t), where I performed the following steps:

Part (a): Plotting x(t) and its Magnitude Spectrum

The first part involved plotting the parabolic periodic function x(t) with a period (T\_x = 2\). The function was defined as:

X(t)=t^2/2 for 0<=t<1 and 0 otherwise

To define this function in MATLAB, I used the `mod` function to repeat the pattern of x(t) over multiple periods. The Fourier series coefficients for x(t) were computed using the integral of the function multiplied by \(e^{-j k \omega\_0 t}\), where \(\omega\_0\) is the fundamental angular frequency. I prealloated the Fourier coefficients array `coeff\_x` and calculated both the DC component and the non-zero coefficients using the `integral` built-in function in MATLAB.

The magnitude spectrum of x(t)was then plotted using the `stem` function, which showed the magnitudes of the Fourier coefficients at different frequencies. The frequency axis was scaled based on the fundamental frequency f=1/Tx, and I plotted the coefficients for the first 20 harmonics .

Part (b): Magnitudes of 3f, 5f, and 7f Components

After plotting the spectrum, I focused on the magnitudes of the components at the frequencies 3f\_0,5f\_0, and 7f\_0, where(f\_0 is the fundamental frequency of the signal. Using the indices corresponding to these frequencies, I displayed their magnitudes from the Fourier coefficients array. The result showed the relative strength of these harmonic components in the signal.

Part (c): Plotting y(t) and Magnitude Spectrum

For the second signal y(t), a rectangular pulse with period T\_y = 5was defined as:

Y(t)=2 for 0<=t<3

Y(t)=0 otherwise

I used the `mod` function again to repeat the pulse over multiple periods. The Fourier coefficients for

Part (d): Plotting =z(t) = x(t) + y(t)and its Magnitude Spectrum

The final part involved calculating the sum of the two signals, z(t) = x(t) + y(t). I then calculated the Fourier coefficients for z(t) by simply adding the corresponding coefficients of x(t) and y(t). The magnitude spectrum of z(t) was plotted, and it was shown that the magnitude spectrum of z(t)was indeed the sum of the individual magnitude spectra of (x(t) and y(t).

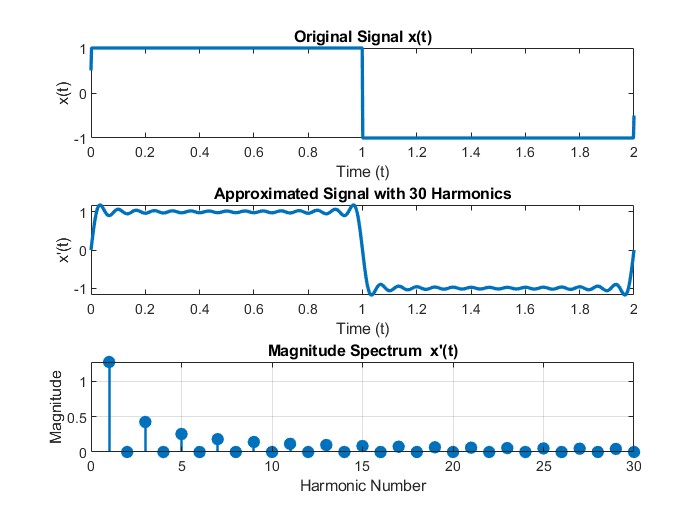
Results:

- The magnitude spectra for both x(t) and y(t) were successfully computed, and their individual contributions to the sum z(t)were verified.

- The specific magnitudes of the harmonics at 3f, 5f, and 7f for x(t) were displayed, giving insight into the frequency components that make up the signal.

- By summing the Fourier series of x(t) and y(t), the spectrum of z(t)was obtained and confirmed to be the sum of the individual spectra, as expected.

This approach provided a clear understanding of how periodic signals can be represented in terms of their Fourier series and how their frequency components relate to the overall signal.

**Question2.**plot:  
 Explanation of the Code:

1. Defined the Period and Time Vector: I set the period `T0` to 2 seconds, which is the given period of the signal. I created a time vector `t` that spans from 0 to `T0`, with 1000 points in between, to get a smooth plot.

2. Defined x(t)Using Unit Step Functions: I created the signal x(t) based on the given expression ( u(t) - 2u(t-1) + u(t-2) ). I used MATLAB's `heaviside` function, which represents the unit step function. This step function combination forms the desired pulse within each period.

3. Set the Number of Harmonics: I set `N = 30` as per the question, meaning I wanted to approximate the signal using 30 harmonics.

4. Initialized Arrays for Approximation and Magnitude Spectrum: I created *`x\_approx*` to store the approximated signal x'(t) and `*magnitude\_spectrum*` to store the magnitudes of each Fourier coefficient.

5. Calculated Fourier Coefficients and Built the Approximation:

- I used a loop over the 30 harmonics. Inside the loop, I calculated each Fourier coefficient “c\_k” by performing an integration using `*trapz`,* with `*exp(-1j \* 2 \* pi \* k / T0 \* t)*` as the Fourier exponential term.

- I added the real part of each harmonic term to `x\_approx` to construct the signal approximation x'(t).

- I stored the magnitude of each c\_k in *`magnitude\_spectrum*` to plot the magnitude line spectrum later.

6. Plotted the Original and Approximated Signals:

- I used *`subplot*` to arrange the plots in the same figure.

- First, I plotted the original x(t)for reference.

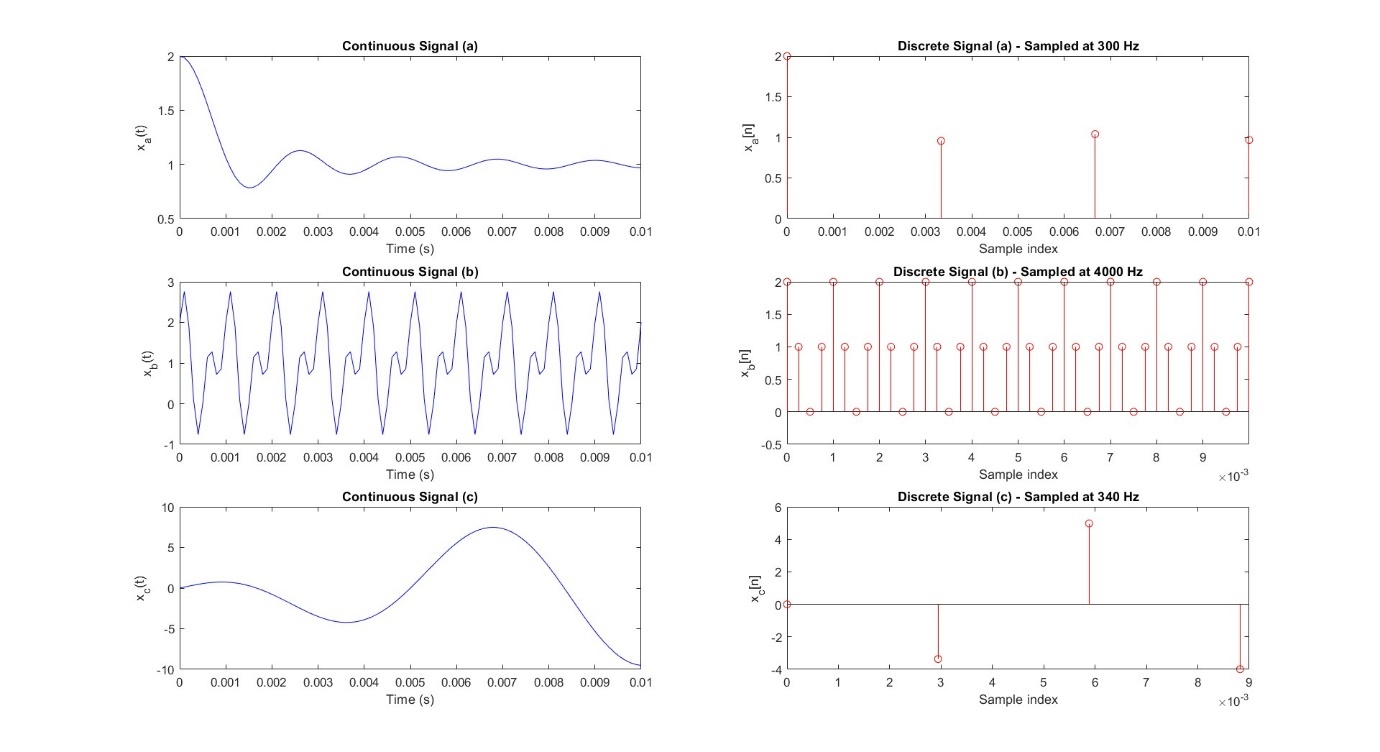
- Then, I plotted x’(t), the 30-harmonic approximation of the original signal.

7. Plotted the Magnitude Spectrum of x'(t) :

- Finally, I used `*stem*` to plot the `*magnitude\_spectrum*` for each harmonic. This shows the magnitude of each harmonic component, providing a clear line spectrum for ( x'(t) .

This approach gave me both the time-domain approximation and the frequency-domain (magnitude) information for the periodic signal.

**Question3.**  
Plot:

  
 Explanation of the Code:

1. Defined Time Variables and Sampling Frequencies:

- I created a continuous-time vector `t` from 0 to 0.01 seconds with a small step size for smooth plotting of the continuous signals.

2. Signal (a): (1 + \text{sinc}(300 \pi t)\):

- The maximum frequency component in this signal is 300 Hz.

- To avoid aliasing, I used a sampling frequency `*fs\_a`* of 300 Hz.

- Discrete-Time Signal: Using *`n\_a*`, the discrete-time sample points based on the chosen sampling interval 1/fs\_a = 1/300, I defined *`x\_a\_disc`* as the discrete version of signal (a).

3. Signal (b): (1 + cos(2000 \*pi \*t) + sin(4000\*pie\*t)):

- The highest frequency component in this signal is 2000 Hz.

- I used a sampling frequency `*fs\_b`* of 4000 Hz to meet the Nyquist criterion.

- Discrete-Time Signal: Using `*n\_b`* as the discrete sample points, I defined `*x\_b\_disc*` as the discrete version of signal (b).

4. Signal (c): 10sin(40\*pi\*t) cos(300\*pi\* t)

- Maximum Frequency Calculation: Expanding the signal, the maximum frequency component is found to be 170 Hz.

- Sampling Frequency: I used a sampling frequency `*fs\_c`* of 340 Hz.

- Discrete-Time Signal: Using `*n\_c`* as the sample points, I defined *`x\_c\_disc*` as the discrete version of signal (c).

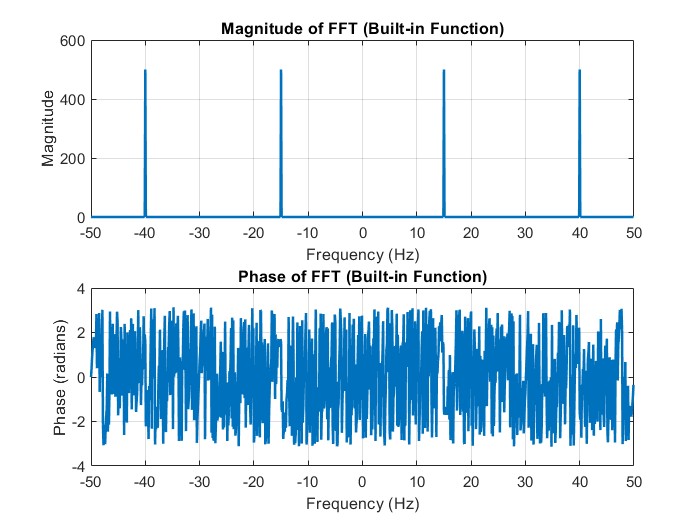
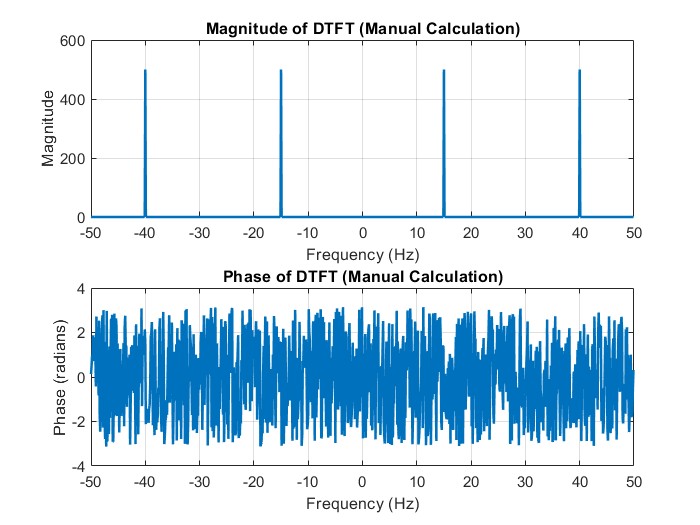
5. Plotting the Continuous and Discrete Signals:

- For each signal (a), (b), and (c), I created a pair of subplots.

- Continuous Signal Plot: I plotted *`x\_a\_cont`, `x\_b\_cont`, and `x\_c\_cont*` for the original continuous-time signals.

- Discrete Signal Plot: I used `stem` to plot `*x\_a\_disc`, `x\_b\_disc`, and `x\_c\_disc*` to show the sampled signals.

This setup allowed me to see the continuous and discrete versions of each signal side by side, verifying that the sampling frequency chosen in each case prevents aliasing.

**Question4**.  
plots:  
 

Explanation of the code:

In this exercise, I computed the Discrete-Time Fourier Transform (DTFT) of the given signal manually. I compared the results with the built-in Fast Fourier Transform (FFT) function in MATLAB.

1. Signal Definition and Time Vector:

First, I defined a time vector `*t = 0:1/100:10-1/100`,* which represents a sampling frequency of 100 Hz. I created a signal `*x = sin(2\*pi\*15\*t) + sin(2\*pi\*40\*t)`* composed of two sinusoidal components with 15 Hz and 40 Hz frequencies.

2. Manual Computation of DTFT:

I manually computed the DTFT of the signal using the following approach:

- I initialized a variable `*DTFT\_manual`* to store the results of the DTFT.

- I defined a frequency vector `*freq\_vector = (-num\_samples/2:num\_samples/2-1)\*(100/num\_samples)`* for plotting the frequency bins.

- Then, I looped through each frequency *bin `freq\_bin`* and time sample *`time\_sample`* and applied the DTFT formula:

DTFT(f) = sum\_{n=0}^{N-1} x(n) e^{-j2\pi f n / N} to compute the Fourier coefficients.

3. Shifted DTFT:

After computing the DTFT manually, I used the `*fftshift`* function to shift the zero-frequency component to the centre of the plot for better visualization. This shifted DTFT is stored in *`DTFT\_manual\_shifted`.*

4. Plotting Magnitude and Phase of DTFT:

I plotted the magnitude and phase of the manually computed DTFT in two separate subplots:

- The magnitude plot was generated using *`abs(DTFT\_manual\_shifted)`,* and the phase plot used `*angle(DTFT\_manual\_shifted)`.*

These plots showed how the signal's frequency content is distributed across the frequency bins.

5. Comparison with Built-in FFT Function:

Next, I used MATLAB's built-in FFT function to calculate the Fourier transform of the signal. I used the command *`FFT\_result = fft(signal)`* to compute the FFT and then shifted the zero-frequency component using `*fftshift(FFT\_result)`* for comparison.

6. Plotting FFT Results:

I plotted the magnitude and phase of the FFT result in separate subplots:

- The magnitude was calculated with `*abs(FFT\_result\_shifted)`.*

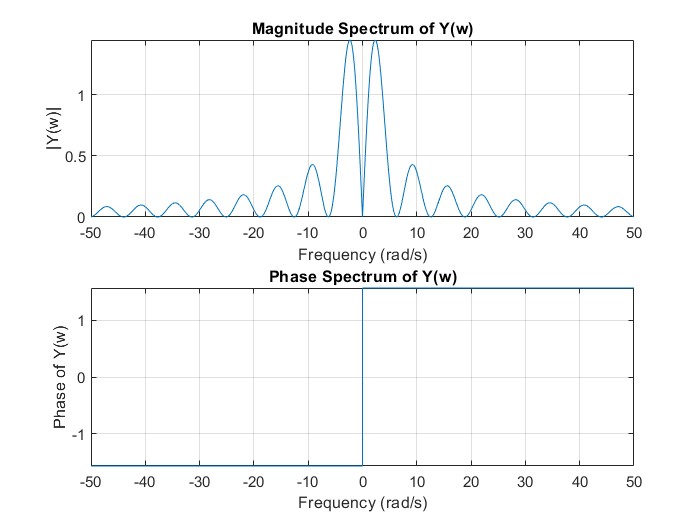
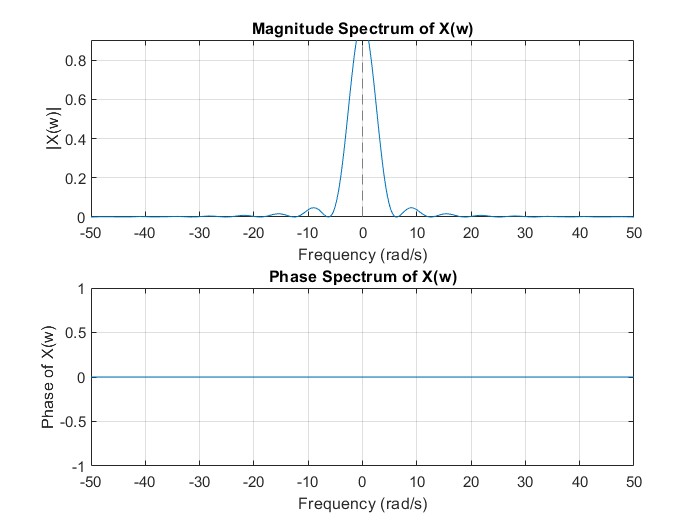
- The phase was obtained using *`angle(FFT\_result\_shifted)`.*

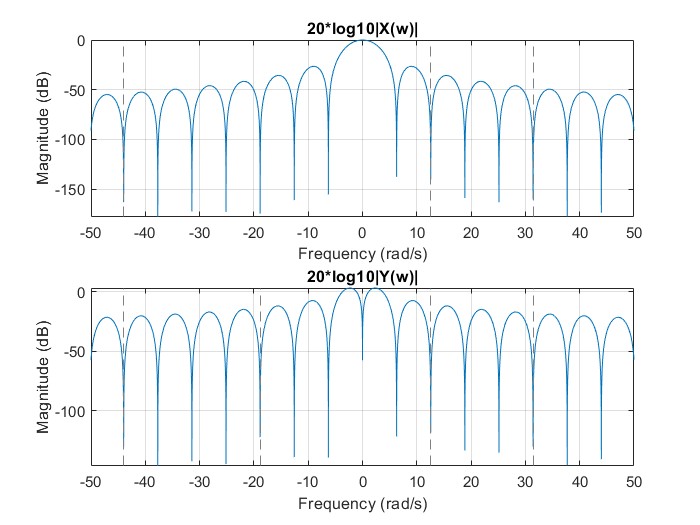
7. Comparison Analysis:

The plots of the magnitude and phase from both the manual DTFT and the built-in FFT were almost identical, which is expected, as the FFT is an efficient implementation of the DTFT for discrete-time signals. Manual computation used a summation approach, whereas FFT used a more optimized algorithm. Both methods gave the same frequency spectrum for the signal.

In conclusion, I verified that the manual DTFT calculation aligns with the result from MATLAB’s FFT function. The results showed clear peaks at the frequencies of the sinusoidal components (15 Hz and 40 Hz), confirming the correctness of the Fourier transform.

**Question 5**.  
plots:





Explanation for code:

The signal x(t) = r(t+1) - 2r(t) + r(t-1) , where r(t) is the ramp function. I used MATLAB to compute the Fourier Transforms of \( x(t) \) and its derivative \( y(t) \), and then analyzed their frequency spectra.

1. Signal Definition:

I defined the ramp function r(t) using the Heaviside step function, which allowed me to construct the signal x(t) as a combination of shifted ramp functions. This formed a piecewise function where the ramps were shifted by 1 and -1 units, and appropriately scaled.

2. \*\*Fourier Transform of x(t) :

Using MATLAB’s fourier function, I computed the Fourier Transform of x(t) denoted as X(w). The result provided the frequency-domain representation of x(t), which showed the distribution of its frequency components.

3. Derivative of x(t) to Get y(t) :

The signal x(t) is the derivative of x(t). Differentiation typically increases high-frequency components, so I computed this derivative using MATLAB’s diff function. This produced a new signal y(t) whose frequency content was expected to have more high-frequency components copared to x(t).

4. Fourier Transform of y(t):

After obtaining y(t), I calculated its Fourier Transform, denoted as Y(w) , to analyze how the frequency components of x(t) evolved when differentiated.

5. Magnitude and Phase Spectra:

To better understand the frequency content of both signals, I computed the magnitude and phase spectra of X(w) and Y(w) using MATLAB’s abs and angle functions. These spectra were plotted to observe the real and imaginary components of the signals.

6. Smoothness Analysis:

To evaluate the smoothness of the signals, I plotted the 20\*log10 of the magnitudes of both X(w) and Y(w). This log-scale plot highlighted the frequency components of both signals, allowing me to visually compare the smoothness. As expected, the derivative of the signal introduced more high-frequency components, making y(t) less smooth than x(t).

Results:

- The magnitude and phase plots showed that X(w) was primarily real-valued, which is typical for signals derived from ramp functions. On the other hand, Y(w) showed some imaginary components due to the differentiation operation.

- From the smoothness analysis using the 20 log10 magnitude plots, I found that Y(w) exhibited higher frequency components, indicating that the derivative operation generated more high-frequency content and possible discontinuities.

Conclusion:

The results showed that differentiation of a signal as in the case of y(t) tends to introduce higher frequencies, making the signal less smooth compared to the original signal. This supports the general observation that differentiation can lead to the generation of high-frequency components and potential discontinuities in a signal.

**X(w)** is **real**, as it is the Fourier transform of a combination of ramp functions, which are real-valued.

**Y(w)** is **complex** (it contains both real and imaginary parts) due to the differentiation operation in the time domain.