I-CHEP Faculty of Engineering, Ain Shams University

Computer Engineering and Software Systems

ECE- 251 Signals and System Fundamentals

Fall 2024

Team Members: -

Abdelrahman Mostafa Sallam	22P0150
Andrew Rami Rizk Bassily	22P0187
Mohamed Ashraf Mohamed	22P0210
Seif Aly Othman Fahmy	22P0182

Represented to: -

Dr. Micheal Ibrahim

Eng. Yassin Salah

Contribution Sheet:-

Name	ID	Contribution
Mohamed Ashraf Mohamed Ahmed	22P0210	Steps 1-7
Abdelrahman Mostafa Sallam	22P0150	Steps 8-13
Andrew Rami Bassily	22P0187	Steps14- 19 & report
Seif Aly Othman	22P0182	Steps 20-25 & report

Acknowledgment: -

We would like to express our deepest gratitude to Dr. Micheal Ibrahim, whose invaluable guidance and profound knowledge of Signals and Systems fundamentals greatly contributed to the success of our course project. His dedication to teaching and mentorship provided us with the essential tools and insights necessary for tackling the complexities of this subject.

We also extend our heartfelt appreciation to Eng. Yassin Saleh for his constant support, timely feedback, and willingness to clarify our doubts. Their combined assistance was instrumental in helping us navigate through challenges and refine our work.

This project would not have been possible without their unwavering support and encouragement.

Table of Contents: -

1. Initializations and First step: -

Sampling Parameters:

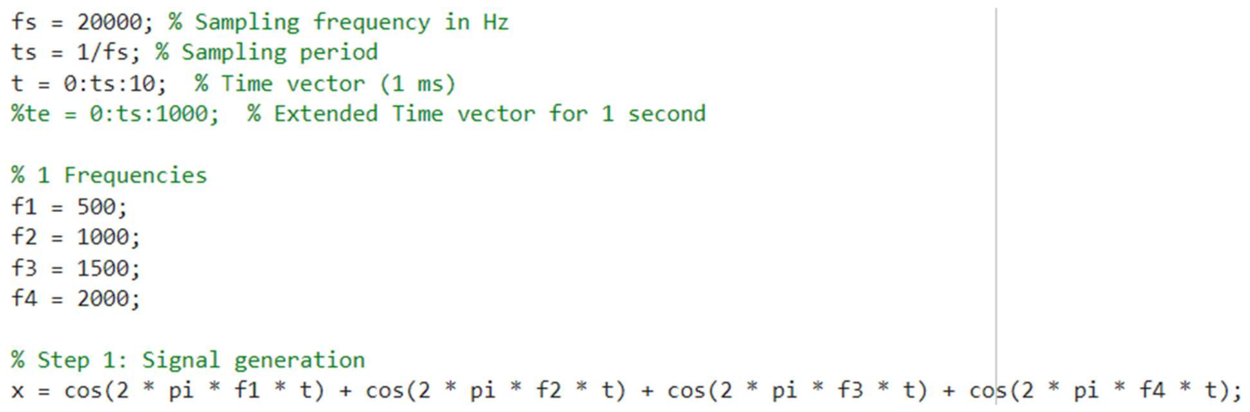
- **fs** = 20000; specifies the sampling frequency at 20 kHz, which is 10 times the maximum frequency we have (f4).
- **ts** = 1/fs; calculates the sampling period.
- **t** = 0:ts:10, Which defines a time vector from 0 to 10 seconds, sampled at the defined period.

Frequencies:

- **f1**, **f2**, **f3**, and **f4** are the frequencies of the four sinusoidal components (500 Hz, 1000 Hz, 1500 Hz, and 2000 Hz).

Signal Generation:

- x = cos (2 * pi * f1 * t) + cos (2 * pi * f2 * t) + cos (2 * pi * f3 * t) + cos (2 * pi * f4 * t), This creates a signal **X** by summing cosine waves with the specified frequencies.



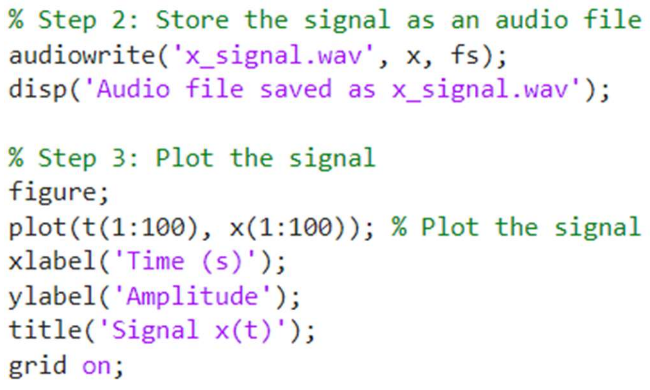
```
fs = 20000; % Sampling frequency in Hz
ts = 1/fs; % Sampling period
t = 0:ts:10;  % Time vector (1 ms)
%te = 0:ts:1000;  % Extended Time vector for 1 second

% 1 Frequencies
f1 = 500;
f2 = 1000;
f3 = 1500;
f4 = 2000;

% Step 1: Signal generation
x = cos(2 * pi * f1 * t) + cos(2 * pi * f2 * t) + cos(2 * pi * f3 * t) + cos(2 * pi * f4 * t);
```

Figure 1 Step1

2. Steps 2 and 3: -

2.1 Step 2: -

In the second step, we save the signal we created as a .wav file and called it **x_signal** using the sampling frequency fs of 20k Hz. After that, we display a message in the command window confirming that the file has been saved using the function disp.



```
% Step 2: Store the signal as an audio file
audiowrite('x_signal.wav', x, fs);
disp('Audio file saved as x_signal.wav');

% Step 3: Plot the signal
figure;
plot(t(1:100), x(1:100)); % Plot the signal
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal x(t)');
grid on;
```

Figure 2 Steps 2 & 3

2.2 Step 3: -

To plot our signal X, we use the plot function with times t (1:100) and x (1:100) to select the first 100 samples of the time vector and signal, respectively. This is done to avoid cluttering the plot when the time vector is large. For the formatting part, xlabel ('Time (s)') labels the X-axis as the time axis using seconds time unit, and ylabel ('Amplitude') to lable the Y-axis as the amplitude. Title ('Signal x(t)') adds a title to describe the plot. Lastly, (grid on) enables a grid for easier interpretation.

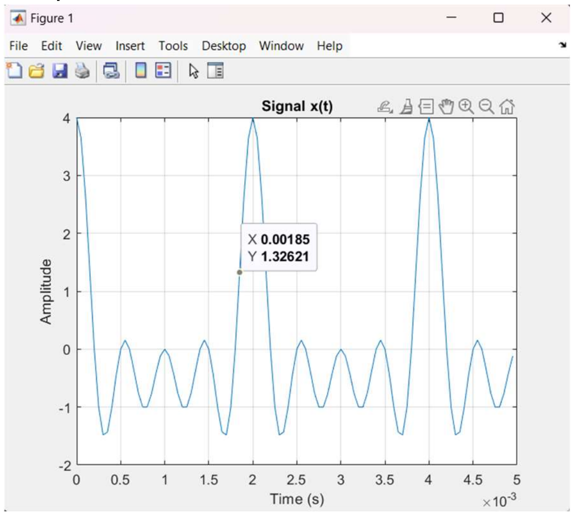


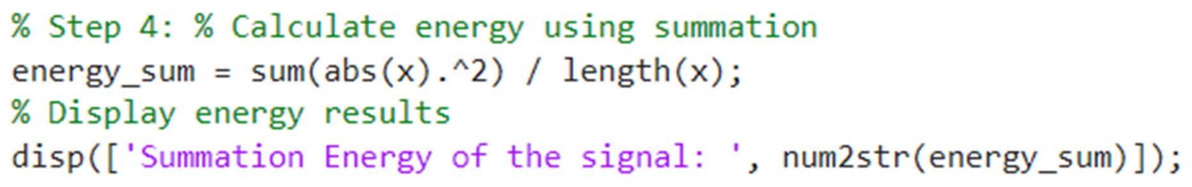
Figure 3 Signal X(t)

3. Step 4: -

When calculating the energy, the result might not have been 100% accurate since reaching such accuracy requires the time period to be taken from negative infinity to infinity which is impossible to calculate on a program, thus a time period from zero to 1000 to be as accurate as possible.

$$\boxed{P = \lim_{L\to\infty}\left[\frac{1}{2L+1}\sum_{n=-L}^{L}|x[n]|^2\right]}$$

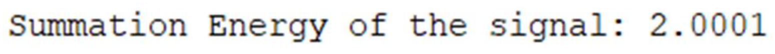
The energy of the signal was calculated using summation to approximate its continuous-time energy. The squared magnitude of each sample in the signal was computed using |x|^2, and the total energy was obtained by summing these squared values. To calculate the average energy per sample, the total energy was divided by the length of the signal. This resulted in an energy value that was displayed using the (disp) function.



```
% Step 4: % Calculate energy using summation
energy_sum = sum(abs(x).^2) / length(x);
% Display energy results
disp(['Summation Energy of the signal: ', num2str(energy_sum)]);
```

Figure 4 Step 4

The previous energy calculations resulted in 2 joules of energy.

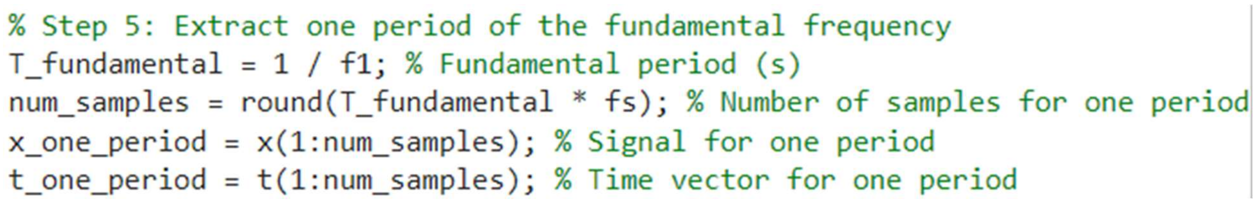


```
Summation Energy of the signal: 2.0001
```

Figure 5 Energy value of x(t)

4. Step 5: -

The code extracts one period of the signal corresponding to the fundamental frequency **f1**. First, the fundamental period, **T fundamental**, is calculated as the reciprocal of the fundamental frequency, **f1**. Given the sampling frequency, **fs**, the number of samples in one period is determined by multiplying **T fundamental** by **fs**, and the result is rounded to the nearest integer to ensure a discrete number of samples. Using this calculated number of samples, the portion of the signal representing one period is extracted and stored in **x_one_period**. Additionally, the corresponding time values for this period are extracted from the time vector and stored in **t_one_period**. This process isolates a single cycle of the fundamental frequency, enabling further analysis or visualization.



```
% Step 5: Extract one period of the fundamental frequency
T_fundamental = 1 / f1; % Fundamental period (s)
num_samples = round(T_fundamental * fs); % Number of samples for one period
x_one_period = x(1:num_samples); % Signal for one period
t_one_period = t(1:num_samples); % Time vector for one period
```

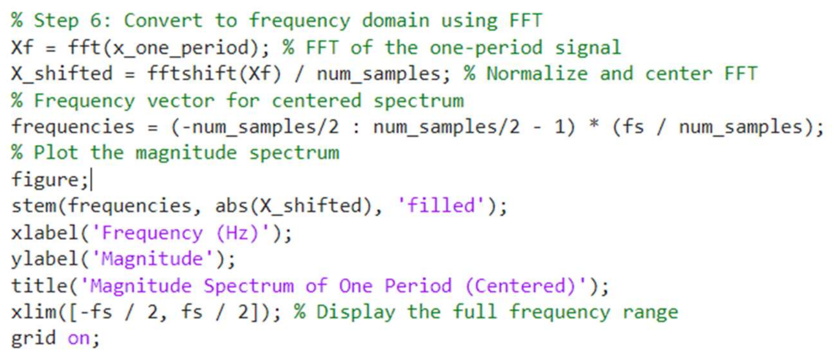
Figure 6 Steps 5

5. Step 6: -

The code begins by calculating the Fast Fourier Transform (FFT) of the one-period signal stored in **x_one_period**. The FFT transforms the signal from the time domain to the frequency domain, providing insight into the signal's frequency components. This operation outputs a complex-valued array, where each element represents the amplitude and phase of a specific frequency component in the signal.

To properly scale and center the FFT for analysis, the result is normalized by dividing it by the number of samples, **num_samples**. This ensures that the magnitudes in the frequency domain correspond to the actual amplitudes of the signal's components. The **fftshift** function is then applied to shift the zero-frequency component to the center of the spectrum. This makes it easier to interpret both the positive and negative frequency components symmetrically.

A frequency vector is constructed to correspond to the centered FFT output. This vector spans from **−fs/2** to **fs/2**, where **fs** is the sampling frequency. The resolution of the frequency vector is determined by the number of samples in the one-period signal, ensuring that each point in the FFT result is associated with a specific frequency.



```
% Step 6: Convert to frequency domain using FFT
Xf = fft(x_one_period); % FFT of the one-period signal
X_shifted = fftshift(Xf) / num_samples; % Normalize and center FFT
% Frequency vector for centered spectrum
frequencies = (-num_samples/2 : num_samples/2 - 1) * (fs / num_samples);
% Plot the magnitude spectrum
figure;
stem(frequencies, abs(X_shifted), 'filled');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum of One Period (Centered)');
xlim([-fs / 2, fs / 2]); % Display the full frequency range
grid on;
```

Figure 7 Step 6

Finally, the magnitude spectrum is plotted using the stem function, which displays the discrete frequency components clearly with filled markers. The x-axis represents the frequency values in Hertz, while the y-axis shows the magnitude of each frequency component. This visualization helps identify the prominent frequencies in the signal and verify its expected frequency components.

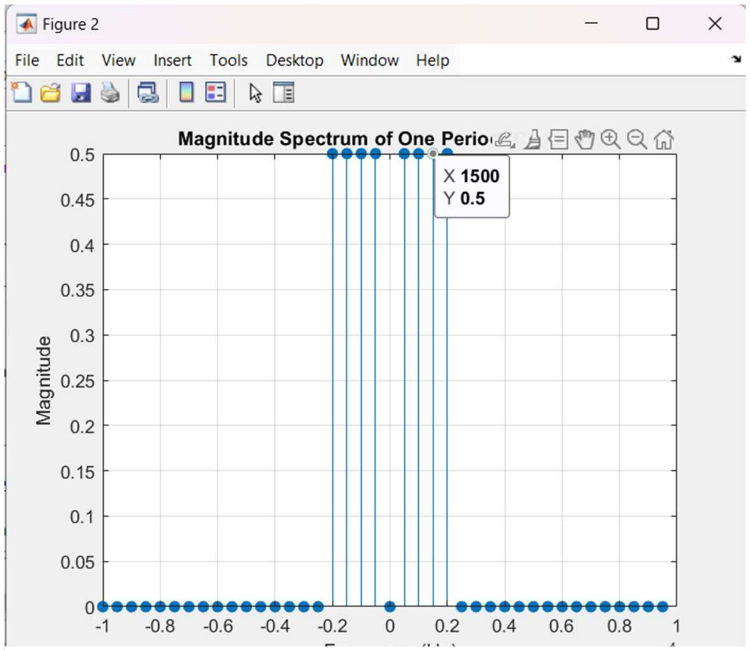
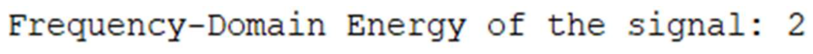


Figure 8 magnitude spectrum of x(t)

6. Steps 7: -

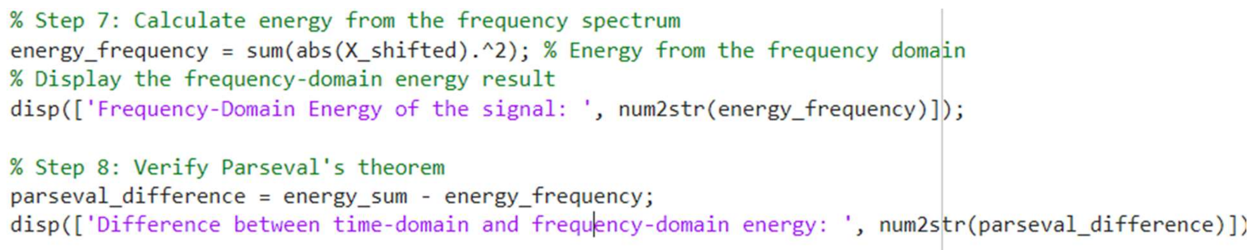
The code calculates the energy of the signal in the frequency domain. This is done by summing the squared magnitudes of the frequency components of the signal, which are contained in the shifted FFT result **X_shifted**. The expression abs(**X_shifted**). ^2 computes the squared magnitude of each complex frequency component in the FFT, representing the energy contribution of each frequency component. By summing these squared magnitudes, the total energy of the signal in the frequency domain is obtained. This energy is then displayed using the (disp) function. This approach follows the principle that the energy in both the time domain and frequency domain should be equal, as stated by Parseval's theorem. However, as previously stated in step 4 Numbers are not 100% accurate, nevertheless, the difference between the two energies is approximately zero verifying Parseval's theorem

$$P_{av} = \frac{1}{T}\int_T |x(t)|^2 \mathrm{dt} = \sum_{k=-\infty}^{\infty} |X[k]|^2$$



```
Frequency-Domain Energy of the signal: 2
```

Figure 9 Computed energy from frequency domain



```
% Step 7: Calculate energy from the frequency spectrum
energy_frequency = sum(abs(X_shifted).^2); % Energy from the frequency domain
% Display the frequency-domain energy result
disp(['Frequency-Domain Energy of the signal: ', num2str(energy_frequency)]);

% Step 8: Verify Parseval's theorem
parseval_difference = energy_sum - energy_frequency;
disp(['Difference between time-domain and frequency-domain energy: ', num2str(parseval_difference)])
```

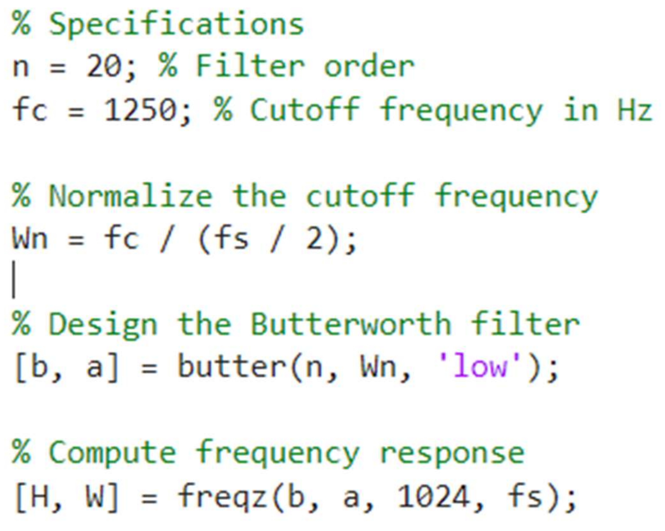
Figure 10 Steps 7 & 8

7. Step 8: -

The filter order (n = 20) is set, determining the steepness of the filter's transition from passband to stopband. A higher order provides a sharper cutoff but may introduce more complexity or delay. Also, the cutoff frequency (fc = 1250) Hz is specified, which defines the point at which the filter begins to attenuate frequencies. Frequencies below the cutoff are passed through, and those above are attenuated.

The cutoff frequency is normalized to the Nyquist frequency (half the sampling rate). This is done by dividing the cutoff frequency **fc** by half the sampling frequency **fs** / 2. The normalized cutoff frequency **Wn** is used for the filter design process. After that, the (butter) function is used to design the Butterworth filter, where **n** is the order and **Wn** is the normalized cutoff frequency. The 'low' argument specifies a low-pass filter, allowing frequencies below the cutoff frequency to pass through and attenuating higher frequencies.

Finally, the (freqz) function computes the frequency response of the designed filter. It takes the filter coefficients b and a (obtained from the butter function) and returns the frequency response H and corresponding frequency vector W. The 1024 argument specifies the number of frequency points for the computation.



```
% Specifications
n = 20; % Filter order
fc = 1250; % Cutoff frequency in Hz

% Normalize the cutoff frequency
Wn = fc / (fs / 2);

% Design the Butterworth filter
[b, a] = butter(n, Wn, 'low');

% Compute frequency response
[H, W] = freqz(b, a, 1024, fs);
```

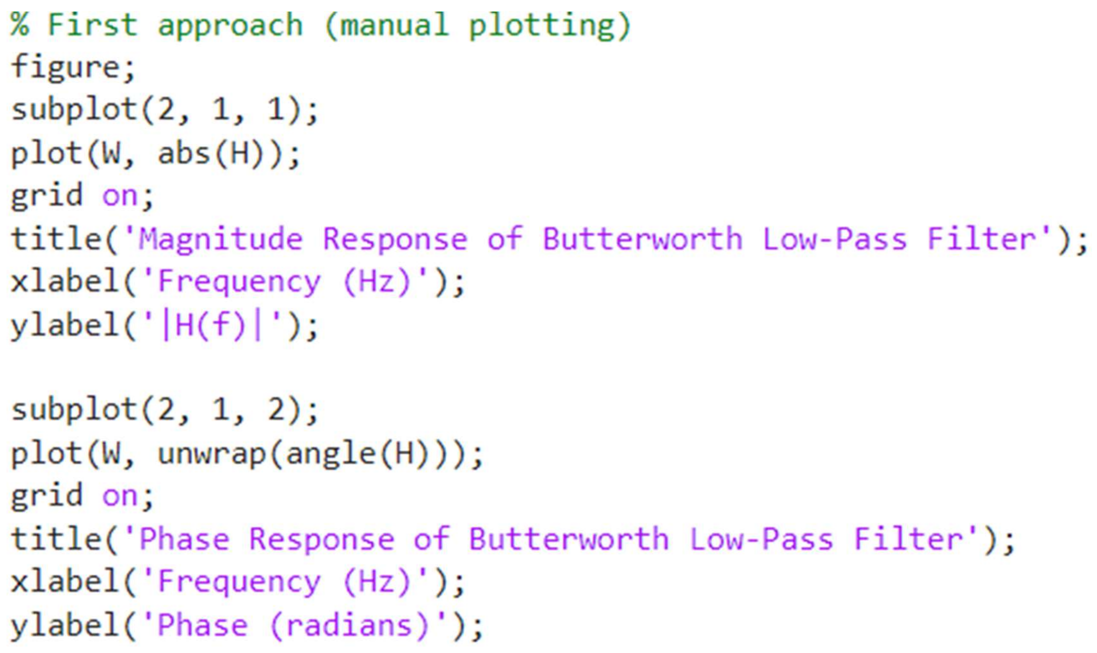
Figure 11Code for low pass filter

8. Step 9: -

The code generates a figure with two subplots, displaying the magnitude and phase responses of the Butterworth low-pass filter. The first subplot, which shows the magnitude response, is created by using the subplot (2, 1, 1) command, dividing the figure into a grid with two rows and one column, and selecting the first plot. The plot (W, abs (H)) function is used to plot the magnitude of the frequency response H against the frequency vector W. The abs(H) function calculates the magnitude of the complex frequency response, and this shows how the filter attenuates different frequency components. The grid on command adds a grid to the plot, making it easier to interpret the frequency response visually. The title 'Magnitude Response of Butterworth Low-Pass Filter' is added to explain the plot, and the x-axis is labeled 'Frequency (Hz)' while the y-axis is labeled '|H(f)|' to represent the magnitude of the filter's response across frequencies

The second subplot, which represents the phase response of the filter, is created by using the subplot (2, 1, 2) command, selecting the second part of the figure grid. The phase response is plotted with plot (W, unwrap (angle (H))), where angle (H) returns the phase angle of the frequency response H, and unwrap is applied to avoid discontinuities in the phase plot. These phase discontinuities occur when the phase jumps suddenly, and the unwrap function smooths out these jumps to provide a continuous phase curve. The grid on command is used again to add a grid to the phase response plot, improving its readability. The title 'Phase Response of Butterworth Low-Pass Filter' is used to describe the subplot, and the x-axis and y-axis are labeled as 'Frequency (Hz)' and 'Phase (radians)', respectively, to indicate the phase shift in radians across the frequency spectrum. The phase response typically shows a gradual shift from 0 radians, indicating the amount of phase distortion introduced by the filter at different frequencies.

Together, these two plots provide a complete picture of the filter's behavior in both the amplitude and phase domains. The magnitude response illustrates how the filter affects the signal's amplitude across various frequencies, while the phase response reveals how the phase of the signal is shifted by the filter. These visualizations are essential for understanding the characteristics of the low-pass Butterworth filter, including how it smoothly attenuates higher frequencies while preserving lower frequencies with minimal phase distortion.



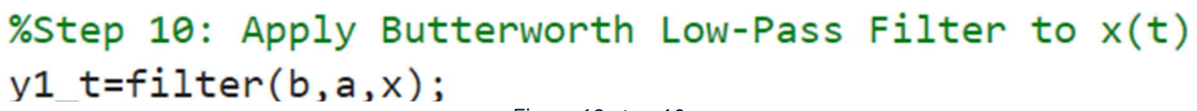
```
% First approach (manual plotting)
figure;
subplot(2, 1, 1);
plot(W, abs(H));
grid on;
title('Magnitude Response of Butterworth Low-Pass Filter');
xlabel('Frequency (Hz)');
ylabel('|H(f)|');

subplot(2, 1, 2);
plot(W, unwrap(angle(H)));
grid on;
title('Phase Response of Butterworth Low-Pass Filter');
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
```

Figure 12 plotting low pass filter

9.Step 10: -

Applying a Butterworth Low-Pass Filter to x(t)

The Butterworth low-pass filter was applied to the input signal x(t) using the MATLAB filter function. The filter coefficients b and a, computed earlier, determine the filter's characteristics. The output, y1(t) is the filtered version of x(t), retaining frequencies below the specified cutoff.



```
%Step 10: Apply Butterworth Low-Pass Filter to x(t)
y1_t=filter(b,a,x);
```

Figure 13 step 10

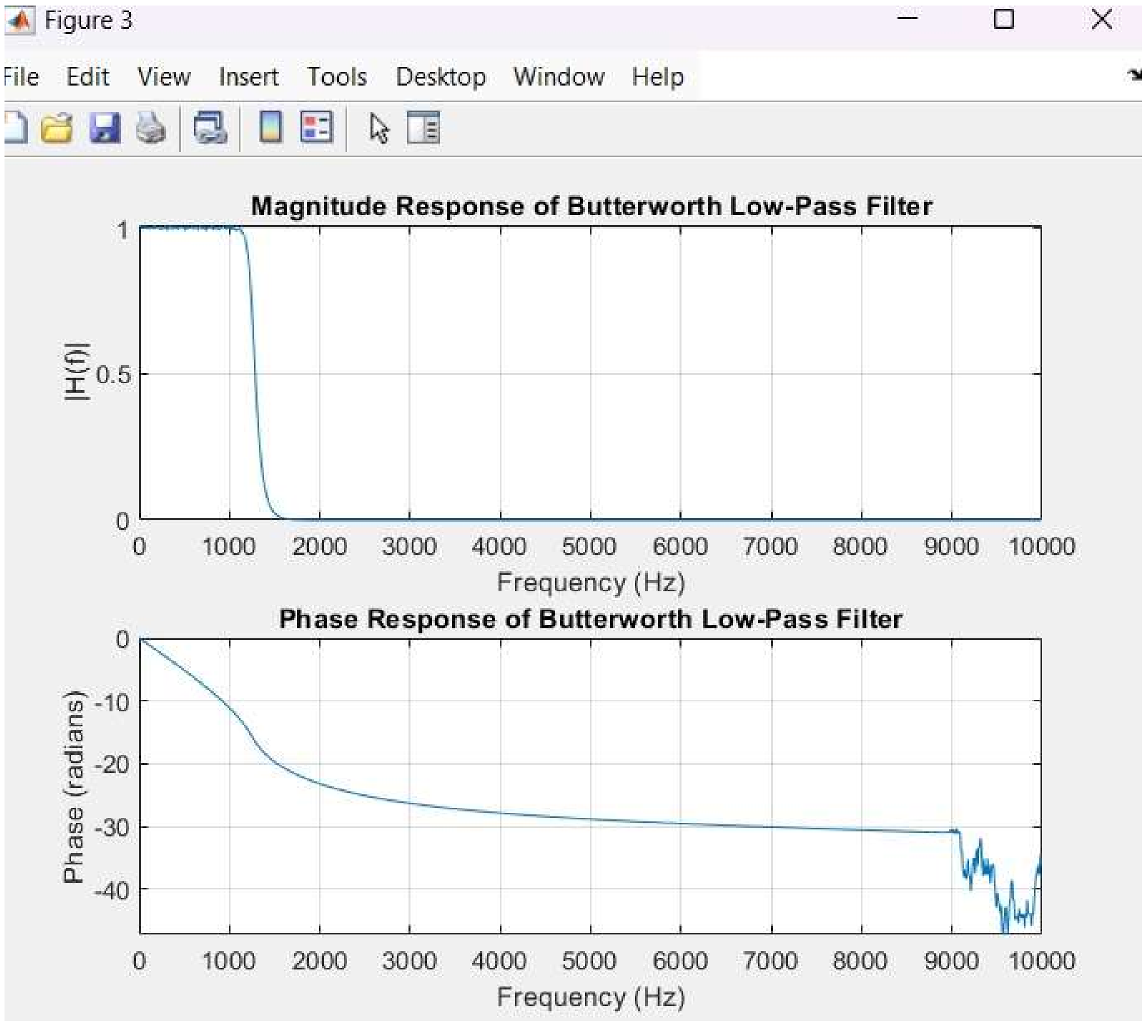


Figure 14 low pass filter on x(t)

10. Step 11: -

From this step forward all methods and functions used will have already been described in this report. The filtered signal y1(t) was saved as an audio file named y1_t_lowpass_filtered_signal.wav using MATLAB's audiowrite function for future analysis or playback.

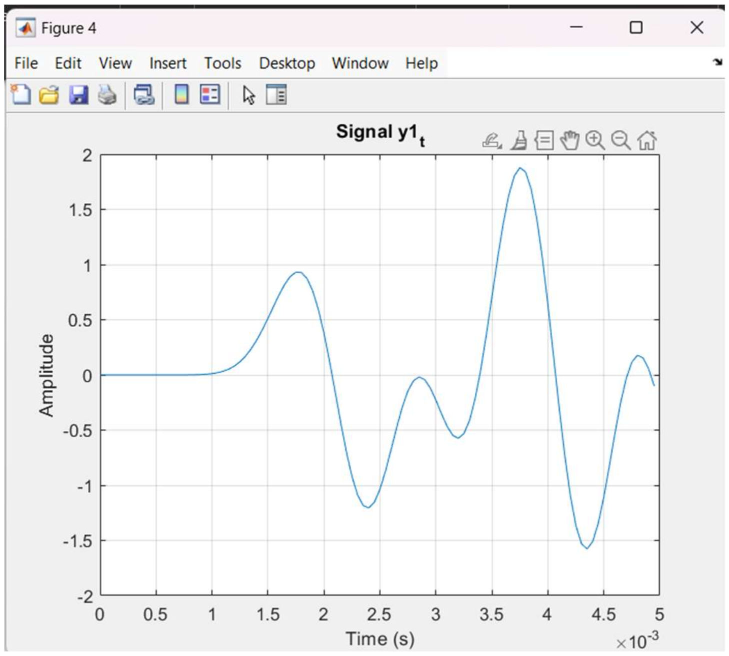
```
%Step 11: Store the signal y2_t as audio file
audiowrite('y1_t_lowpass_filtered_signal.wav', y1_t, fs);
```

Figure 15 step 11

11. Step 12:

A segment of y1(t) was plotted to visualize its amplitude over time. This aids in understanding the temporal behavior of the low pass filtered signal.

```
%step 12: plot y1_t
figure;
plot(t(1:200), y1_t(1:200)); % Plot the signal
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal y_1(t) (Low-Pass Filtered)');
grid on;
```

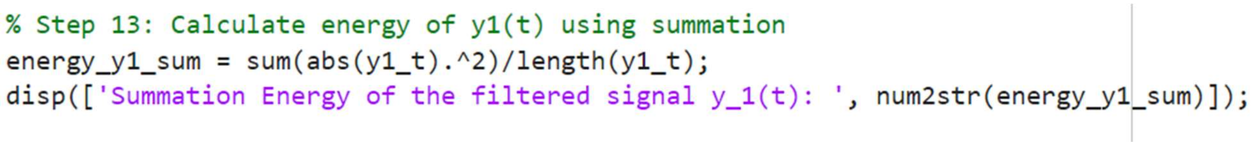


12. Step 13: -

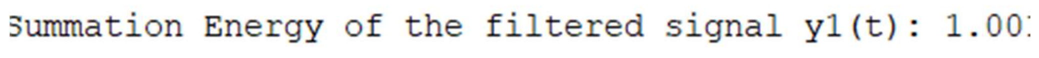
The energy of y1(t) was calculated using the summation formula:

$$E = \frac{1}{N}\sum_{n=1}^{N} |y_1[n]|^2$$

This provides a measure of the signal's strength in the time domain.



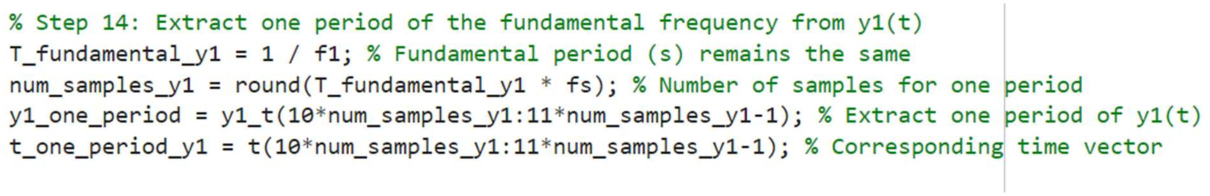
```
% Step 13: Calculate energy of y1(t) using summation
energy_y1_sum = sum(abs(y1_t).^2)/length(y1_t);
disp(['Summation Energy of the filtered signal y_1(t): ', num2str(energy_y1_sum)]);
```



```
Summation Energy of the filtered signal y1(t): 1.00
```

13. Step 14: -

To analyze the periodic nature of y1(t), one period corresponding to the fundamental frequency f1was extracted. This segment was used for further analysis in the frequency domain.



```
% Step 14: Extract one period of the fundamental frequency from y1(t)
T_fundamental_y1 = 1 / f1; % Fundamental period (s) remains the same
num_samples_y1 = round(T_fundamental_y1 * fs); % Number of samples for one period
y1_one_period = y1_t(10*num_samples_y1:11*num_samples_y1-1); % Extract one period of y1(t)
t_one_period_y1 = t(10*num_samples_y1:11*num_samples_y1-1); % Corresponding time vector
```

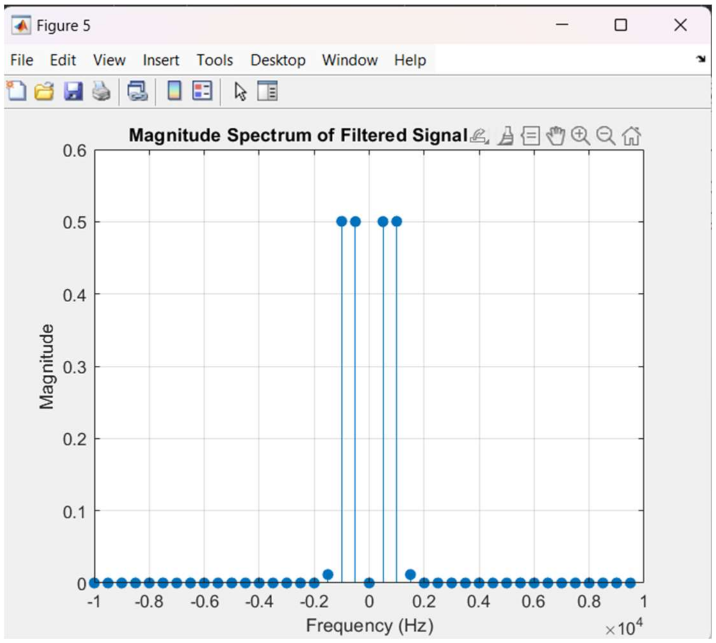
14. Step 15: -

The Fast Fourier Transform (FFT) was applied to the extracted signal period, and the result was normalized and centered using fftshift. The frequency spectrum was plotted to observe the signal's frequency components.

```
% Step 15: Convert y1(t) to the frequency domain using FFT
Yf = fft(y1_one_period);
Y_shifted = fftshift(Yf) / num_samples_y1; % Normalize and center FFT

% Frequency vector for the centered spectrum
frequencies_y1 = (-num_samples_y1/2 : num_samples_y1/2 - 1) * (fs / num_samples_y1);
```

We then plot the magnitude spectrum of one period in a discrete way using the stem() functionality, we then label the x-axis as ‘Magnitude’ and the y-axis as ‘Frequency (Hz)’, using the xlabel and ylabel functionalities.

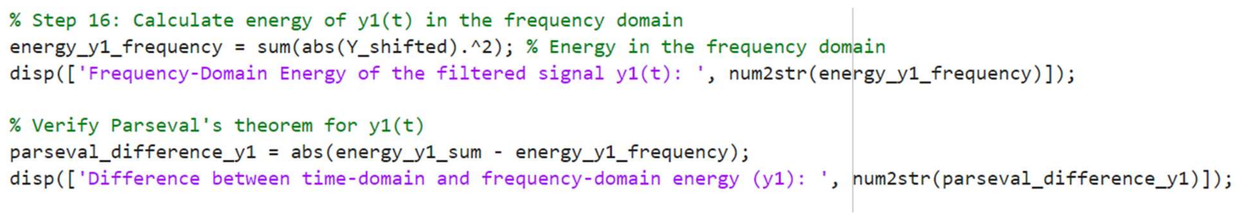
```
% Plot the magnitude spectrum of y1(t)
figure;
stem(frequencies_y1, abs(Y_shifted), 'filled');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum of Filtered Signal y_1(t) (Centered)');
xlim([-fs / 2, fs / 2]);
grid on;
```



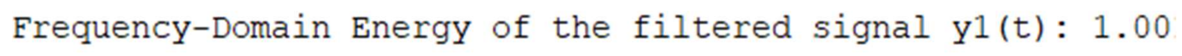
15. Step 16: -

Calculating Frequency-Domain Energy and Verifying Parseval's Theorem

The energy of y1(t) was calculated in the frequency domain using the magnitude of the FFT coefficients. Parseval's theorem was verified by comparing time-domain and frequency-domain energy, confirming energy preservation across domains.



```
% Step 16: Calculate energy of y1(t) in the frequency domain
energy_y1_frequency = sum(abs(Y_shifted).^2); % Energy in the frequency domain
disp(['Frequency-Domain Energy of the filtered signal y1(t): ', num2str(energy_y1_frequency)]);

% Verify Parseval's theorem for y1(t)
parseval_difference_y1 = abs(energy_y1_sum - energy_y1_frequency);
disp(['Difference between time-domain and frequency-domain energy (y1): ', num2str(parseval_difference_y1)]);
```

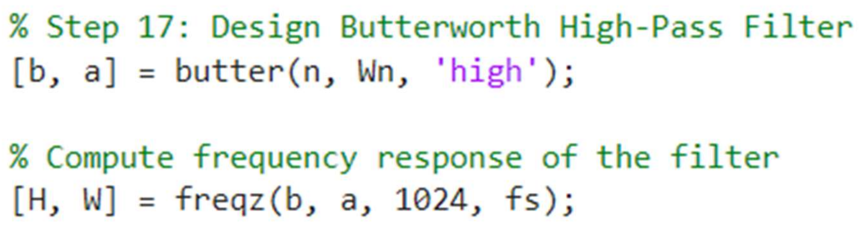


```
Frequency-Domain Energy of the filtered signal y1(t): 1.00
```

16. Steps 17 & 18: -

Step 17:

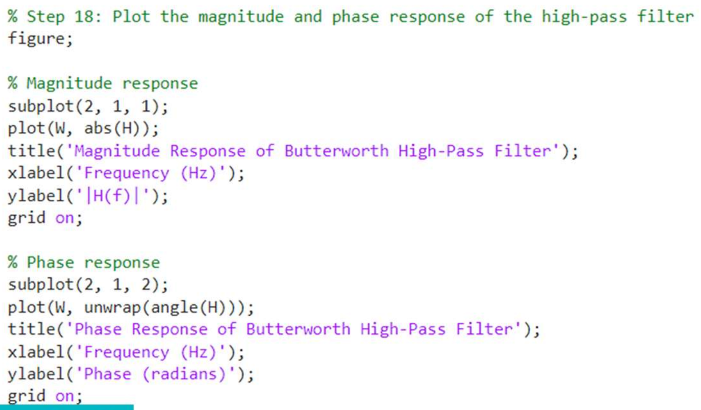
Using the same initializations that were used for the low pass filter, we simply use the function "butter", and specify that it's a "high" in order to clarify its identity as a high pass filter, we then commute the frequency response of the filter using the "freqz()" function.



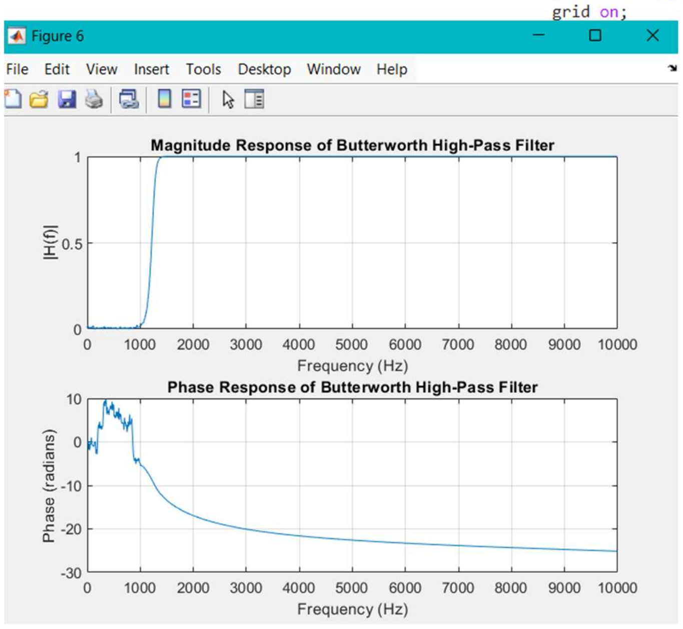
```
% Step 17: Design Butterworth High-Pass Filter
[b, a] = butter(n, Wn, 'high');

% Compute frequency response of the filter
[H, W] = freqz(b, a, 1024, fs);
```

Step 18:

We then plot both the magnitude and phase response of the high pass filter, using the matlab code displayed in the attached screenshot



```
% Step 18: Plot the magnitude and phase response of the high-pass filter
figure;

% Magnitude response
subplot(2, 1, 1);
plot(W, abs(H));
title('Magnitude Response of Butterworth High-Pass Filter');
xlabel('Frequency (Hz)');
ylabel('|H(f)|');
grid on;

% Phase response
subplot(2, 1, 2);
plot(W, unwrap(angle(H)));
title('Phase Response of Butterworth High-Pass Filter');
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
grid on;
```



The code generates a figure with two subplots, displaying the magnitude and phase responses of the Butterworth High-pass filter. The first subplot, which shows the magnitude response, is created by using the subplot (2, 1, 1) command, dividing the figure into a grid with two rows and one column, and selecting the first plot. The plot (W, abs (H)) function is used to plot the magnitude of the frequency response H against the frequency vector W. The abs(H) function calculates the magnitude of the complex frequency response, and this shows how the filter attenuates different frequency components. The grid on command adds a grid to the plot, making it easier to interpret the frequency response visually. The title 'Magnitude Response of Butterworth High-Pass Filter' is added to explain the plot, and the x-axis is labeled 'Frequency (Hz)' while the y-axis is labeled '|H(f)|' to represent the magnitude of the filter's response across frequencies

The second subplot, which represents the phase response of the filter, is created by using the subplot (2, 1, 2) command, selecting the second part of the figure grid. The phase response is plotted with plot (W, unwrap (angle (H))), where angle (H) returns the phase angle of the frequency response H, and unwrap is applied to avoid discontinuities in the phase plot. These phase discontinuities occur when the phase jumps suddenly, and the unwrap function smooths out these jumps to provide a continuous phase curve. The grid on command is used again to add a grid to the phase response plot, improving its readability. The title 'Phase Response of Butterworth High-Pass Filter' is used to describe the subplot, and the x-axis and y-axis are labeled as 'Frequency (Hz)' and 'Phase (radians)', respectively, to indicate the phase shift in radians across the frequency spectrum. The phase response typically shows a gradual shift from 0 radians, indicating the amount of phase distortion introduced by the filter at different frequencies.

17. Steps 19 & 20: -

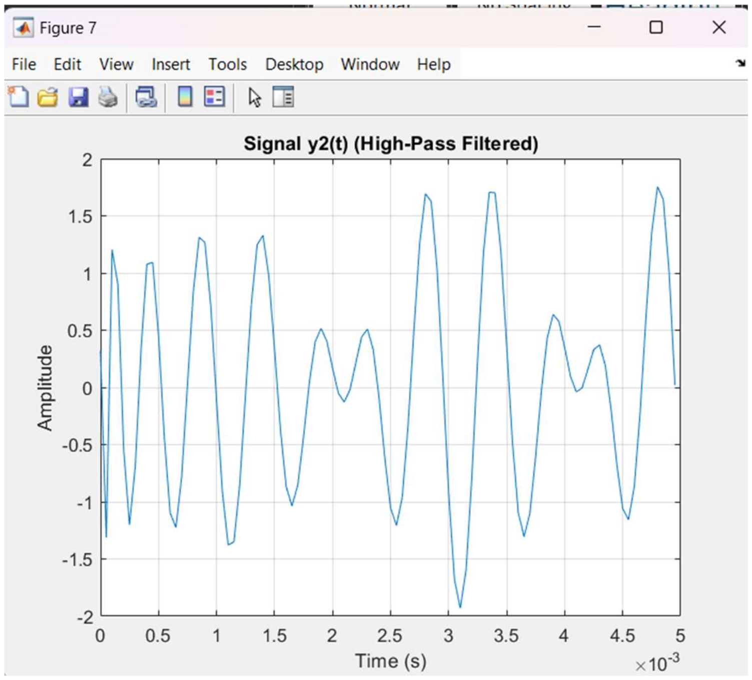
Similar to step 10 here we apply the high pass filter to the original signal x(t) storing it in y2(t) using the filter function by MATLAB. We also store this signal in an audio file to be available to the user for listening.

```
% Step 19: Apply Butterworth High-Pass Filter to x(t)
y2_t = filter(b, a, x); % Apply high-pass filter to x(t)

% Step 20: store the signal y2_t as audio file
audiowrite('y2_t_highpass_filtered_signal.wav', y2_t, fs); % Save as audio file
```

18. Step 21: -

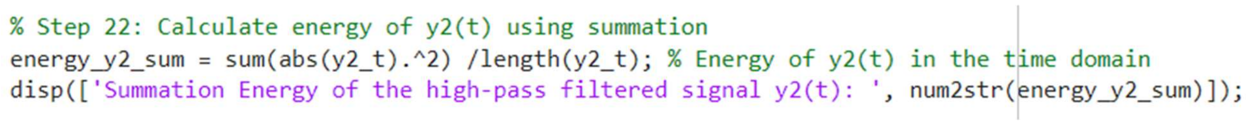
After we have successfully applied the high pass filter, we will display the signal y2(t) using the plot function.

```
% Step 21: Plot y2(t)
figure;
plot(t(1:100), y2_t(1:100)); % Plot the filtered signal
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal y2(t) (High-Pass Filtered)');
grid on;
```

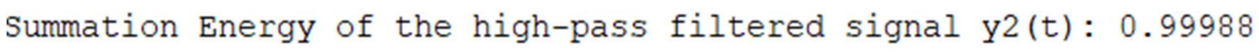


19. Step 22: -

In step 22 we calculate the energy of the signal y2(t) in the time domain by summing the squared magnitudes of the signal's values and then normalizing by the length of the signal, similar to signals x(t) and y1(t).



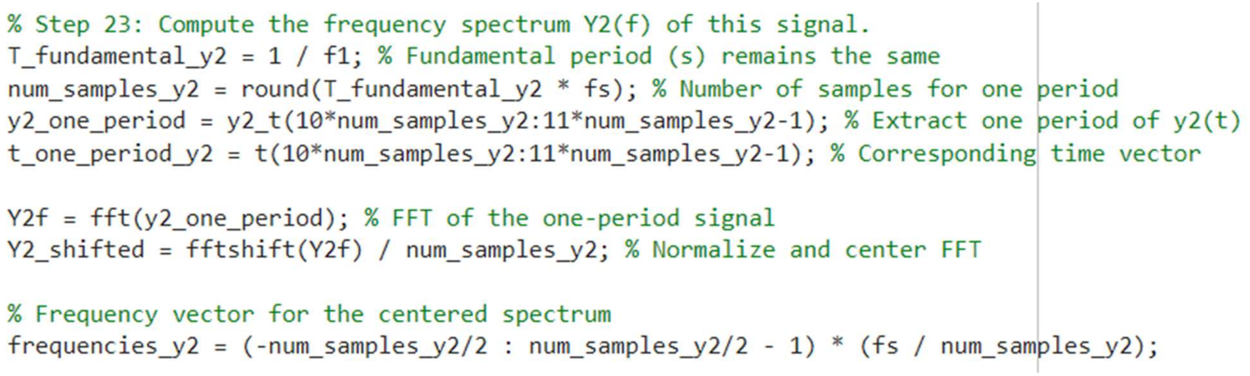
```
% Step 22: Calculate energy of y2(t) using summation
energy_y2_sum = sum(abs(y2_t).^2) /length(y2_t); % Energy of y2(t) in the time domain
disp(['Summation Energy of the high-pass filtered signal y2(t): ', num2str(energy_y2_sum)]);
```



```
Summation Energy of the high-pass filtered signal y2(t): 0.99988
```

20. Step 23: -

This part extracts one period of the high pass filtered signal y2(t) and computes its frequency spectrum using the Fast Fourier Transform (FFT). First, it calculates the fundamental period based on the signal's frequency and determines the number of samples per period based on the sampling frequency. It then extracts a segment of the signal corresponding to one period and applies the FFT to this segment. The result is shifted using fftshift() to center the zero-frequency component and normalized by the number of samples. Finally, a frequency vector is generated to match the frequency bins of the shifted FFT output, providing a frequency-domain representation of the signal.

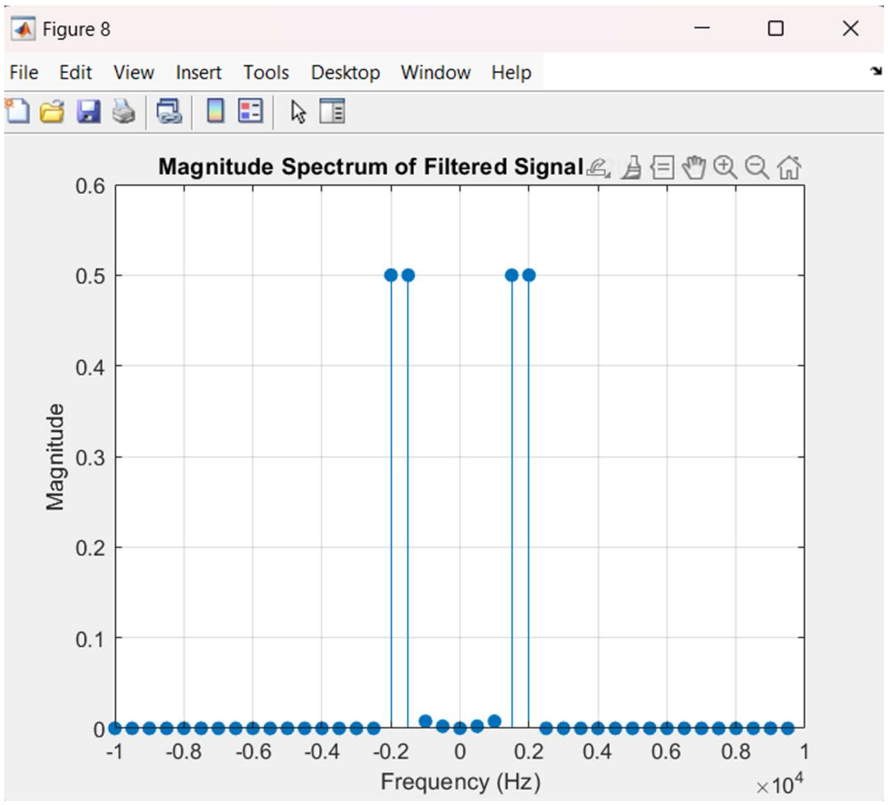


```
% Step 23: Compute the frequency spectrum Y2(f) of this signal.
T_fundamental_y2 = 1 / f1; % Fundamental period (s) remains the same
num_samples_y2 = round(T_fundamental_y2 * fs); % Number of samples for one period
y2_one_period = y2_t(10*num_samples_y2:11*num_samples_y2-1); % Extract one period of y2(t)
t_one_period_y2 = t(10*num_samples_y2:11*num_samples_y2-1); % Corresponding time vector

Y2f = fft(y2_one_period); % FFT of the one-period signal
Y2_shifted = fftshift(Y2f) / num_samples_y2; % Normalize and center FFT

% Frequency vector for the centered spectrum
frequencies_y2 = (-num_samples_y2/2 : num_samples_y2/2 - 1) * (fs / num_samples_y2);
```

21. Step 24: -

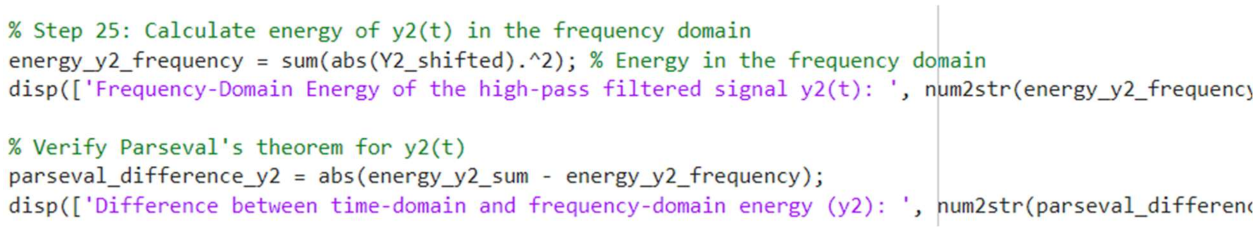
This part displays the extracted impulses in the frequency domain

```
% Step 24: Plot the magnitude spectrum of y2(t)
figure;
stem(frequencies_y2, abs(Y2_shifted), 'filled');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum of Filtered Signal y2(t) (Centered)');
xlim([-fs / 2, fs / 2]); % Display the full frequency range
grid on;
```

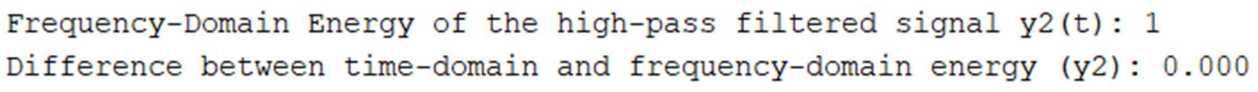


22. Final step: -

This code calculates the energy of the high pass filtered signal y2(t) in both the time and frequency domains and verifies **Parseval's theorem**. First, the energy in the frequency domain is computed by squaring the absolute values of the FFT result Y2_shifted and summing them up. The difference between the energy in the time domain (energy_y2_sum) and the frequency domain (energy_y2_frequency) is then calculated to check if Parseval's theorem holds, which states that the total energy of a signal should be the same in both domains. The small difference between the energies confirms whether the theorem is valid for y2(t).



```
% Step 25: Calculate energy of y2(t) in the frequency domain
energy_y2_frequency = sum(abs(Y2_shifted).^2); % Energy in the frequency domain
disp(['Frequency-Domain Energy of the high-pass filtered signal y2(t): ', num2str(energy_y2_frequency

% Verify Parseval's theorem for y2(t)
parseval_difference_y2 = abs(energy_y2_sum - energy_y2_frequency);
disp(['Difference between time-domain and frequency-domain energy (y2): ', num2str(parseval_differenc
```



```
Frequency-Domain Energy of the high-pass filtered signal y2(t): 1
Difference between time-domain and frequency-domain energy (y2): 0.000
```

Figure 16 Frequency domain energy

23. References: -

1. **Oppenheim, A. V., Willsky, A. S., & Nawab, S. H. (1997).** *Signals and Systems.* Prentice Hall.
2. **Proakis, J. G., & Manolakis, D. G. (2007).** *Digital Signal Processing: Principles, Algorithms, and Applications.* Pearson Education.
3. **Smith, S. W. (1997).** *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing.
4. **Bracewell, R. N. (1999).** *The Fourier Transform and Its Applications.* McGraw-Hill.