# **Digital Signal Processing Lab Report**

Contents

[**Digital Signal Processing Lab Report** 1](#_Toc149577377)

[Question 1: 2](#_Toc149577378)

[Question 2 ( Part a): 4](#_Toc149577379)

[Question 2 ( Part b): 6](#_Toc149577380)

[Question 3 (part a): 8](#_Toc149577381)

[Question 3 (part b): 10](#_Toc149577382)

[Question 3 (part c): 12](#_Toc149577383)

[Question 4 (part a): 15](#_Toc149577384)

[Question 4 (part b): 18](#_Toc149577385)

# Question 1:

**Code:**

% Parameters

Amplitude = 5; % Amplitude

Period = 40e-3; % Period in seconds

timeVector = linspace(0, Period, 1000); % Time vector

FourierTerms = 50; % Number of terms in the Fourier series

% Initialize the waves

y\_squareWave = zeros(size(timeVector));

y\_pulseWave = zeros(size(timeVector));

y\_triangleWave = zeros(size(timeVector));

% Generate the Fourier series

for harmonic = 1:2:FourierTerms\*2

% Square wave

Bn\_squareWave = (2\*Amplitude/pi) \* (1/harmonic); % Fourier coefficient

y\_squareWave = y\_squareWave + Bn\_squareWave \* sin(2\*pi\*harmonic\*timeVector/Period); % Add each term to the waveform

% Pulse wave (different coefficients)

Bn\_pulseWave = (2\*Amplitude/pi) \* (1 - (-1)^harmonic) / (pi \* harmonic); % Fourier coefficient

y\_pulseWave = y\_pulseWave + Bn\_pulseWave \* sin(2\*pi\*harmonic\*timeVector/Period); % Add each term to the waveform

% Triangle wave

Bn\_triangleWave = (8\*Amplitude/(pi^2\*harmonic^2)) \* ((-1)^((harmonic-1)/2)); % Fourier coefficient

y\_triangleWave = y\_triangleWave + Bn\_triangleWave \* sin(2\*pi\*harmonic\*timeVector/Period); % Add each term to the waveform

end

% Plot the waveforms

figure;

subplot(3,1,1);

plot(timeVector, y\_squareWave, 'r');

title('Fourier Square Wave');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3,1,2);

plot(timeVector, y\_pulseWave, 'g');

title('Fourier Pulse Wave');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3,1,3);

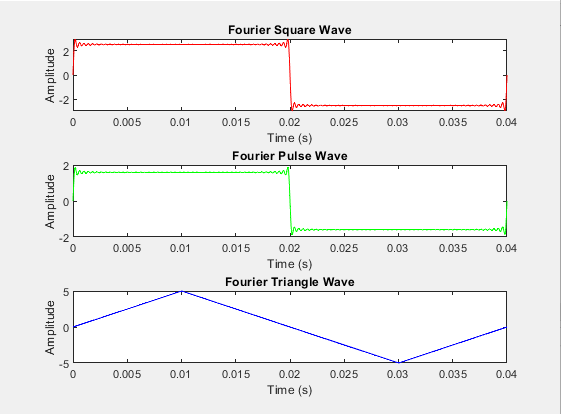
plot(timeVector, y\_triangleWave, 'b');

title('Fourier Triangle Wave');

xlabel('Time (s)');

ylabel('Amplitude');

**Output:**

****

# Question 2 ( Part a):

**Code:**

clc

clear

close all

samplingRate = 600; % sampling frequency

sampleCount = 8192; % number of samples

duration = sampleCount/samplingRate; % total duration

timeVector = 0:1/samplingRate:0.5; % time vector

Amplitude = 5; % Peak-to-peak amplitude of the square wave

squareWaveFrequency = 10;

% Generate a square wave with odd harmonics

squareWaveSignal = Amplitude \* square(2 \* pi \* squareWaveFrequency \* timeVector);

for harmonicIndex = 1:2:11

squareWaveSignal = squareWaveSignal + 1/harmonicIndex\*sin(2\*pi\*harmonicIndex\*timeVector);

end

% Generate random noise

randomNoise = randn(size(timeVector));

% Add noise to the square wave

noisySquareWaveSignal = squareWaveSignal;

% Normalize to RMS amplitude of 1

noisySquareWaveSignal = noisySquareWaveSignal + .1\*randomNoise;

% Plot the noisy square wave

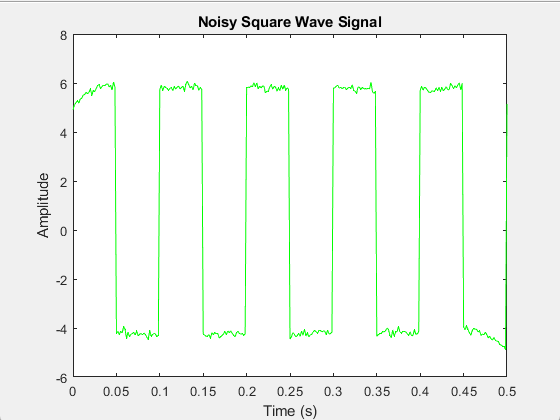
plot(timeVector, noisySquareWaveSignal,'g');

title('Noisy Square Wave Signal');

xlabel('Time (s)');

ylabel('Amplitude');

**Output:**

****

# Question 2 ( Part b):

**Code**

clc

clear

close all

samplingFrequency = 600; % sampling frequency

numberOfSamples = 8192; % number of samples

totalDuration = numberOfSamples/samplingFrequency; % total duration

timeVector = 0:1/samplingFrequency:0.5; % time vector

peakAmplitude = 5; % Peak-to-peak amplitude of the square wave

squareWaveFrequency = 10;

% Generate a square wave with odd harmonics

squareWaveSignal = peakAmplitude \* square(2 \* pi \* squareWaveFrequency \* timeVector);

for harmonicIndex = 1:2:11

squareWaveSignal = squareWaveSignal + 1/harmonicIndex\*sin(2\*pi\*harmonicIndex\*timeVector);

end

% Generate random noise

randomNoise = randn(size(timeVector));

% Add noise to the square wave

noisySquareWaveSignal = squareWaveSignal;

% Normalize to RMS amplitude of 1

noisySquareWaveSignal = noisySquareWaveSignal + .1\*randomNoise;

% Compute the FFT of the noisy square wave

FFTResult = fft(noisySquareWaveSignal, numberOfSamples);

frequencyVector = samplingFrequency/2 \* linspace(0, 1, numberOfSamples/2 + 1);

% Calculate the log magnitude in dB

logMagnitudeResult = 20 \* log10(2/numberOfSamples \* abs(FFTResult(1:numberOfSamples/2 + 1)));

% Plot the time-domain signal

figure;

subplot(2, 1, 1);

plot(timeVector, noisySquareWaveSignal, 'r');

title('Noisy Square Wave in Time Domain');

xlabel('Time (s)');

ylabel('Amplitude');

grid on;

% Plot the log magnitude single-sided frequency plot

subplot(2, 1, 2);

plot(frequencyVector, logMagnitudeResult, 'b');

title('Single-Sided Frequency Plot in dB');

xlabel('Frequency (Hz)');

ylabel('Magnitude (dB)');

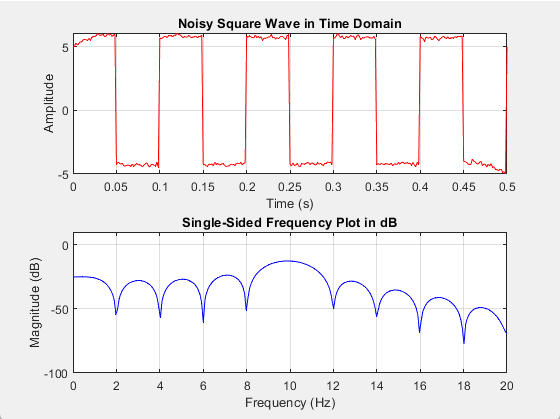
grid on;

% Customize the frequency plot for a square wave

axis([0 20 -100 10]);

% Adjust the axis limits and other settings as needed for your specific requirements.

**Output:**

****

# Question 3 (part a):

**Code:**

clc

clear

close all

samplingRate = 600; % sampling frequency

sampleCount = 8192; % number of samples

totalTime = sampleCount/samplingRate; % total duration

timeVector = 0:1/samplingRate:.5; % time vector

peakAmplitude = 5; % Peak-to-peak amplitude of the square wave

squareWaveFreq = 10;

% Generate a square wave with odd harmonics

squareWaveSignal = peakAmplitude \* square(2 \* pi \* squareWaveFreq \* timeVector);

for harmonicIndex = 1:2:11

squareWaveSignal = squareWaveSignal + 1/harmonicIndex\*sin(2\*pi\*harmonicIndex\*timeVector);

end

% Generate random noise

randomNoise = randn(size(timeVector));

% Add noise to the square wave

noisySquareWaveSignal = squareWaveSignal;

% Normalize to RMS amplitude of 1

noisySquareWaveSignal = noisySquareWaveSignal + .1\*randomNoise;

% Define the filter specifications

centerFrequencies = [25, 75, 125, 175, 225, 275]; % in Hz

bandWidth = 3; % in Hz

passBandRipple = 0.1; % in dB

stopBandAttenuation = 60; % in dB

% Initialize the filtered signal as a zero array

filteredSignal = zeros(size(noisySquareWaveSignal));

sumOfSignals=zeros(size(noisySquareWaveSignal));

k=1;

% Loop over each centre frequency

figure;

for freqCenter = centerFrequencies

% Define the passband and stopband frequencies

Wp = [(freqCenter - bandWidth) (freqCenter + bandWidth)] / (samplingRate / 2);

Ws = [(freqCenter - 2\*bandWidth) (freqCenter + 2\*bandWidth)] / (samplingRate / 2);

% Use the ellip function to design an Elliptic filter

[n,Wn] = ellipord(Wp, Ws, passBandRipple, stopBandAttenuation);

[b,a] = ellip(n, passBandRipple, stopBandAttenuation, Wn);

% Apply the filter to the noisy signal and add it to the filtered signal

filteredSignal = filter(b, a, noisySquareWaveSignal);

sumOfSignals=sumOfSignals+filteredSignal;

subplot(6,1,k);

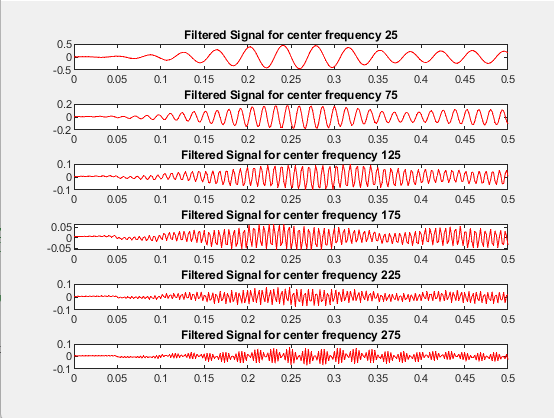
plot(timeVector, filteredSignal,'r');

title('Filtered Signal for center frequency '+string(centerFrequencies(k)) );

k=k+1;

end

**Output:**

****

# Question 3 (part b):

**Code:**

clc

clear

close all

samplingRate = 600; % sampling frequency

sampleCount = 8192; % number of samples

totalTime = sampleCount/samplingRate; % total duration

timeVector = 0:1/samplingRate:.5; % time vector

peakAmplitude = 5; % Peak-to-peak amplitude of the square wave

squareWaveFreq = 10;

% Generate a square wave with odd harmonics

squareWaveSignal = peakAmplitude \* square(2 \* pi \* squareWaveFreq \* timeVector);

for harmonicIndex = 1:2:11

squareWaveSignal = squareWaveSignal + 1/harmonicIndex\*sin(2\*pi\*harmonicIndex\*timeVector);

end

% Generate random noise

randomNoise = randn(size(timeVector));

% Add noise to the square wave

noisySquareWaveSignal = squareWaveSignal;

% Normalize to RMS amplitude of 1

noisySquareWaveSignal = noisySquareWaveSignal + .1\*randomNoise;

% Define the filter specifications

centerFrequencies = [25, 75, 125, 175, 225, 275]; % in Hz

bandWidth = 3; % in Hz

passBandRipple = 0.1; % in dB

stopBandAttenuation = 60; % in dB

% Initialize the filtered signal as a zero array

filteredSignal = zeros(size(noisySquareWaveSignal));

sumOfSignals=zeros(size(noisySquareWaveSignal));

k=1;

% Loop over each centre frequency

figure;

for freqCenter = centerFrequencies

% Define the passband and stopband frequencies

Wp = [(freqCenter - bandWidth) (freqCenter + bandWidth)] / (samplingRate / 2);

Ws = [(freqCenter - 2\*bandWidth) (freqCenter + 2\*bandWidth)] / (samplingRate / 2);

% Use the ellip function to design an Elliptic filter

[n,Wn] = ellipord(Wp, Ws, passBandRipple, stopBandAttenuation);

[b,a] = ellip(n, passBandRipple, stopBandAttenuation, Wn);

% Apply the filter to the noisy signal and add it to the filtered signal

filteredSignal = filter(b, a, noisySquareWaveSignal);

sumOfSignals=sumOfSignals+filteredSignal;

subplot(6,1,k);

plot(timeVector, filteredSignal,'r');

title('Filtered Signal for center frequency '+string(centerFrequencies(k)) );

k=k+1;

end

% Plot the original and filtered signals for comparison

figure;

subplot(2,1,1);

plot(timeVector, noisySquareWaveSignal,'b');

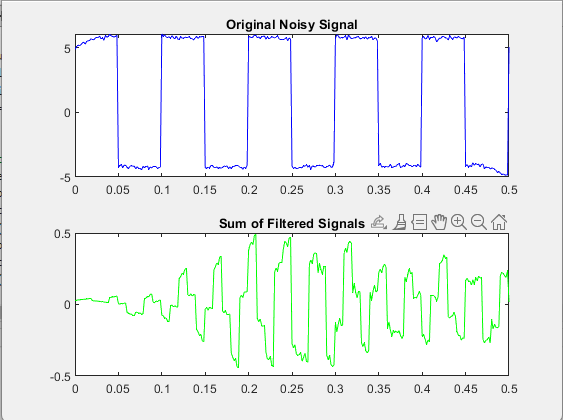
title('Original Noisy Signal');

subplot(2,1,2);

plot(timeVector, sumOfSignals,'g');

title('Sum of Filtered Signals');

**Output:**

****

# Question 3 (part c):

**Code:**

clc

clear

close all

% Define parameters

samplingFreq = 600; % Sampling frequency

numOfSamples = 8192; % Number of samples

totalTime = numOfSamples/samplingFreq; % Total duration

timeVec = 0:1/samplingFreq:1; % Time vector

peakAmp = 5; % Peak-to-peak amplitude of the square wave

freqSquare = 10;

% Generate a square wave with odd harmonics

squareWaveSignal = peakAmp \* square(2 \* pi \* freqSquare \* timeVec);

for harmonicIndex = 1:2:11

squareWaveSignal = squareWaveSignal + 1/harmonicIndex\*sin(2\*pi\*harmonicIndex\*timeVec);

end

% Generate random noise

randomNoise = randn(size(timeVec));

% Add noise to the square wave

noisySquareWaveSignal = squareWaveSignal;

% Normalize to RMS amplitude of 1

noisySquareWaveSignal = noisySquareWaveSignal + 0.1 \* randomNoise;

% Define the filter specifications

centerFreqs = [25, 75, 125, 175, 225, 275]; % in Hz

bandWidth = 3; % in Hz

passBandRipple = 0.1; % in dB

stopBandAttenuation = 60; % in dB

% Initialize the filtered signal as a zero array

filteredSignals = zeros(length(centerFreqs), length(noisySquareWaveSignal));

% Loop over each center frequency

for i = 1:length(centerFreqs)

freqCenter = centerFreqs(i);

% Define the passband and stopband frequencies

Wp = [(freqCenter - bandWidth) (freqCenter + bandWidth)] / (samplingFreq / 2);

Ws = [(freqCenter - 2\*bandWidth) (freqCenter + 2\*bandWidth)] / (samplingFreq / 2);

% Use the ellip function to design an Elliptic filter

[n, Wn] = ellipord(Wp, Ws, passBandRipple, stopBandAttenuation);

[b, a] = ellip(n, passBandRipple, stopBandAttenuation, Wn);

% Apply the filter to the noisy signal

filteredSignals(i, :) = filter(b, a, noisySquareWaveSignal);

end

% Combine the filtered signals

outputSignal = sum(filteredSignals);

% Plot the original noisy signal and the filtered signal

figure;

subplot(2, 1, 1);

plot(timeVec, noisySquareWaveSignal,'r');

title('Original Noisy Signal');

xlabel('Time (s)');

ylabel('Amplitude');

grid on;

subplot(2, 1, 2);

plot(timeVec, outputSignal,'b');

title('Filtered Signal');

xlabel('Time (s)');

ylabel('Amplitude');

grid on;

% Compute the FFT of the filtered signal

Y = fft(outputSignal, numOfSamples);

frequenciesVector = samplingFreq/2 \* linspace(0, 1, numOfSamples/2 + 1);

% Calculate the log magnitude in dB

logMagnitudeResult = 20 \* log10(2/numOfSamples \* abs(Y(1:numOfSamples/2 + 1)));

% Plot the log magnitude single-sided frequency plot

figure;

plot(frequenciesVector, logMagnitudeResult,'g');

title('Log Magnitude (in dB) Single-Sided Frequency Plot');

xlabel('Frequency (Hz)');

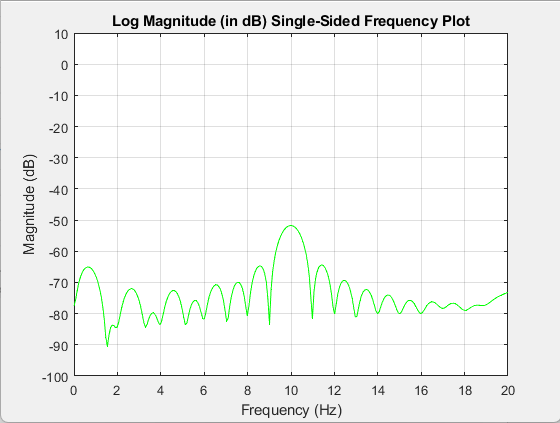
ylabel('Magnitude (dB)');

grid on;

% Customize the frequency plot for a square wave

axis([0 20 -100 10]);

**Output:**



# Question 4 (part a):

**Code:**

clc

clear

close all

% Define parameters

sampleFreq = 600; % Sampling frequency

numSamples = 8192; % Number of samples

totalDuration = numSamples/sampleFreq; % Total duration

timeVec = 0:1/sampleFreq:1; % Time vector

peakAmp = 5; % Peak-to-peak amplitude of the square wave

freqSquareWave = 10;

% Generate a square wave with odd harmonics

squareWaveSignal = peakAmp \* square(2 \* pi \* freqSquareWave \* timeVec);

for harmonicIndex = 1:2:11

squareWaveSignal = squareWaveSignal + 1/harmonicIndex\*sin(2\*pi\*harmonicIndex\*timeVec);

end

% Generate random noise

randomNoise = randn(size(timeVec));

% Add noise to the square wave

noisySquareWaveSignal = squareWaveSignal;

% Normalize to RMS amplitude of 1

noisySquareWaveSignal = noisySquareWaveSignal + 0.1 \* randomNoise;

% Define the filter specifications

centerFreqs = [25, 75, 125, 175, 225, 275]; % in Hz

bandWidth = 3; % in Hz

passBandRipple = 0.1; % in dB

stopBandAttenuation = 60; % in dB

% Initialize the filtered signal as a zero array

filteredSignalsArray = zeros(length(centerFreqs), length(noisySquareWaveSignal));

% Loop over each center frequency

for i = 1:length(centerFreqs)

freqCenter = centerFreqs(i);

% Define the passband and stopband frequencies

Wp = [(freqCenter - bandWidth) (freqCenter + bandWidth)] / (sampleFreq / 2);

Ws = [(freqCenter - 2\*bandWidth) (freqCenter + 2\*bandWidth)] / (sampleFreq / 2);

% Use the ellip function to design an Elliptic filter

[n, Wn] = ellipord(Wp, Ws, passBandRipple, stopBandAttenuation);

[b, a] = ellip(n, passBandRipple, stopBandAttenuation, Wn);

% Apply the filter to the noisy signal

filteredSignalsArray(i, :) = filter(b, a, noisySquareWaveSignal);

end

% Combine the filtered signals

combinedFilteredSignals = sum(filteredSignalsArray);

f7thHarmonic = 7\*freqSquareWave; % Frequency of the seventh harmonic

BWwidth = 8; % Bandwidth

woNormalizedFreq = f7thHarmonic/(sampleFreq/2); % Normalized frequency

BWwidthNormalizedBandwidth = BWwidth/(sampleFreq/2); % Normalized bandwidth

[b,a] = iirnotch(woNormalizedFreq, BWwidthNormalizedBandwidth); % Design notch filter

filteredSignalFinalOutput = filter(b, a, combinedFilteredSignals); % Apply filter to remove seventh harmonic

% Plot the original and filtered signals for visualization

timeVectorPlotting = (0:length(combinedFilteredSignals) - 1) / sampleFreq;

figure;

subplot(2, 1, 1);

plot(timeVectorPlotting, combinedFilteredSignals,'r');

title('Original Combined Filtered Signals');

xlabel('Time (s)');

ylabel('Amplitude');

grid on;

subplot(2, 1, 2);

plot(timeVectorPlotting, filteredSignalFinalOutput,'b');

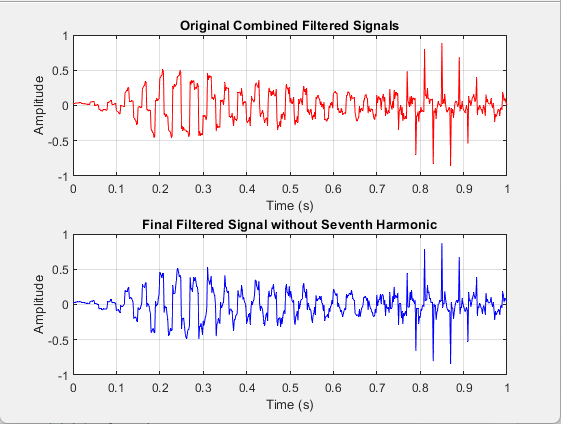
title('Final Filtered Signal without Seventh Harmonic');

xlabel('Time (s)');

ylabel('Amplitude');

grid on;

**Output:**

****

# Question 4 (part b):

**Code:**

% Clear workspace

clear

close all

% Define parameters

samplingFreq = 600; % Sampling frequency

numSamples = 8192; % Number of samples

totalDuration = numSamples/samplingFreq; % Total duration

timeVector = 0:1/samplingFreq:1; % Time vector

amplitude = 5; % Peak-to-peak amplitude of the square wave

freqSquare = 10;

% Generate a square wave with odd harmonics

generatedSquareWave = amplitude \* square(2 \* pi \* freqSquare \* timeVector);

for index = 1:2:11

generatedSquareWave = generatedSquareWave + 1/index\*sin(2\*pi\*index\*timeVector);

end

% Generate random noise

randomNoise = randn(size(timeVector));

% Add noise to the square wave

noisyGeneratedSquareWave = generatedSquareWave;

% Normalize to RMS amplitude of 1

noisyGeneratedSquareWave = noisyGeneratedSquareWave + 0.1 \* randomNoise;

% Define the filter specifications

centerFreqs = [25, 75, 125, 175, 225, 275]; % in Hz

bandWidth = 3; % in Hz

passbandRipple = 0.1; % in dB

stopbandAttenuation = 60; % in dB

% Initialize the filtered signal as a zero array

filteredSignals = zeros(length(centerFreqs), length(noisyGeneratedSquareWave));

% Loop over each center frequency

for index = 1:length(centerFreqs)

freq = centerFreqs(index);

% Define the passband and stopband frequencies

passBandFreqs = [(freq - bandWidth) (freq + bandWidth)] / (samplingFreq / 2);

stopBandFreqs = [(freq - 2\*bandWidth) (freq + 2\*bandWidth)] / (samplingFreq / 2);

% Use the ellip function to design an Elliptic filter

[filterOrder, cutoffFreq] = ellipord(passBandFreqs, stopBandFreqs, passbandRipple, stopbandAttenuation);

[filterB, filterA] = ellip(filterOrder, passbandRipple, stopbandAttenuation, cutoffFreq);

% Apply the filter to the noisy signal

filteredSignals(index, :) = filter(filterB, filterA, noisyGeneratedSquareWave);

end

% Combine the filtered signals

combinedSignal = sum(filteredSignals);

seventhHarmonicFreq = 7\*freqSquare; % Frequency of the seventh harmonic

notchBandwidth = 8; % Bandwidth

normalizedFreq = seventhHarmonicFreq/(samplingFreq/2); % Normalized frequency

normalizedBandwidth = notchBandwidth/(samplingFreq/2); % Normalized bandwidth

[notchFilterB,notchFilterA] = iirnotch(normalizedFreq, normalizedBandwidth); % Design notch filter

filteredSignalFinal = filter(notchFilterB, notchFilterA, combinedSignal); % Apply filter to remove seventh harmonic

% Plot the original and filtered signals for visualization

timeVectorPlotting = (0:length(combinedSignal) - 1) / samplingFreq;

figure

subplot(2, 1, 1);

plot(timeVectorPlotting, combinedSignal,'r'); % Red color for original signal plot.

grid on

title('Combined Signal');

subplot(2, 1, 2);

plot(timeVectorPlotting, filteredSignalFinal,'b'); % Blue color for filtered signal plot.

grid on

title('Final Filtered Signal without Seventh Harmonic');

% Magnitude plot with green color.

figure

fvtool(notchFilterB,notchFilterA)

**Output:**

**A graph of a graph

Description automatically generated**

**Effectiveness of Notch Filters:**

Notch filters, also known as band-stop filters or frequency rejection filters, are crucial in audio engineering and signal processing. These electronic filters are engineered to selectively attenuate or reject specific frequencies, enabling the elimination of undesired noise or interference from an audio signal. Here are some additional points about the effectiveness of notch filters:

1. **Noise Mitigation:** Notch filters play a pivotal role in audio applications where they mitigate unwanted noise such as hiss, hum, or buzz. They can also enhance the performance of other filter types by reducing the amount of out-of-band energy that they need to process.
2. **Targeted Attenuation:** Notch filters can selectively attenuate a narrow range of frequencies. This ability allows them to effectively eliminate hum, noise, or interference, resulting in a cleaner and more pleasurable auditory experience.
3. **Signal Enhancement:** In many scenarios, it’s necessary to remove a narrowband signal without altering band energy. This can be accomplished by routing the signals through a notch filter.
4. **Versatile Applications:** Due to their ability to eliminate specific frequencies from a signal, notch filters find applications in various fields.
5. **Improved EEG Signal Quality:** The application of a notch filter at the specific frequency of power line hum can significantly reduce or eliminate unwanted noise, thereby enhancing the quality of the EEG signal.
6. **Audio Processing:** In audio processing, notch filters can be used to eliminate unwanted feedback or resonance. This can improve the quality of audio recordings and live performances.
7. **Adaptability:** Notch filters can be designed to target any frequency, making them highly adaptable to a variety of applications. This adaptability extends to their ability to handle different types of noise, from low-frequency hums to high-frequency hisses.