Filter Design Project

ELC 423: Digital Signal Processing Lecture

Matthew Bohr

Dr. Ambrose Adegbege

December 10, 2022

1. Introduction and Background

When designing a digital filter, it is useful to start by designing an analog filter and then ultimately transition to the discrete-time domain. Luckily, MATLAB has a variety of functions and toolboxes that greatly simplify this process.

For this project, the students were tasked with designing a digital filter with the following specifications:

- Input signal sampled at a uniform rate of 1 KHz.
- Frequencies above 420 Hz must be suppressed with a transition bandwidth of 20 Hz i.e. (410 Hz – 430 Hz). Filter must have a linear phase with an equiripple passband with a maximum ripple of 0.01 and an equiripple stopband with a maximum ripple of 0.1.
- Frequencies below 50 Hz must be suppressed with a stopband attenuation of 50 dB, a maximum passband ripple of 2 dB, a stopband edge frequency of 40 Hz, and a passband edge frequency of 60 Hz.

2. Design Approach

The specifications above indicate the need for a bandpass filter (BPF). The second bullet demands the filter to suppress the components of the input signal above 420 Hz, characteristic of a lowpass filter (LPF), and to suppress the components below 50 Hz, characteristic of a highpass filter (HPF). Cascading these two structures of course produces a BPF. With this in mind, our BPF filter in MATLAB was designed as the cascade of a LPF and a HPF that satisfy the specifications above.

The specifications were all assigned to MATLAB variables (lines 5-15). The LPF was designed first by using the *firpmord()*, *firpm()*, and *dfilt.dffir()* MATLAB functions from the Communications Toolbox. The LPF passband frequency, stopband frequency, passband ripple, and stopband ripple are passed into the *firpmord()* function on line 20 to produce important characteristics about the LPF such as the order and the normalized frequency band edges. The *firpm()* function on line 22 takes in these parameters to produce a vector of LPF coefficients, which is the argument to *dfilt.dffir()* on line 24 to produce a filter object called *LPF_Filter*. Lines 29-41 show the exact same procedure for the HPF to create *HPF_Filter*, with two additional expressions needed on lines 31-32 to compute the passband and stopband ripples in the linear scale since they were given in dB for the highpass portion.

The *dfilt.cascade()* on line 46 cascades the *LPF_Filter* with *HPF_Filter* to yield a bandpass filter called *BPF_Filter*. The *BPF_Filter* object is passed into *freqz()* and *phasez()* and stored in *BPF_mag* and *BPF_phase* on lines 48 and 50 respectively to produce the magnitude and phase response of the filter. Moreover, the *impz()* function on line 52 produces the impulse response of the filter as well.

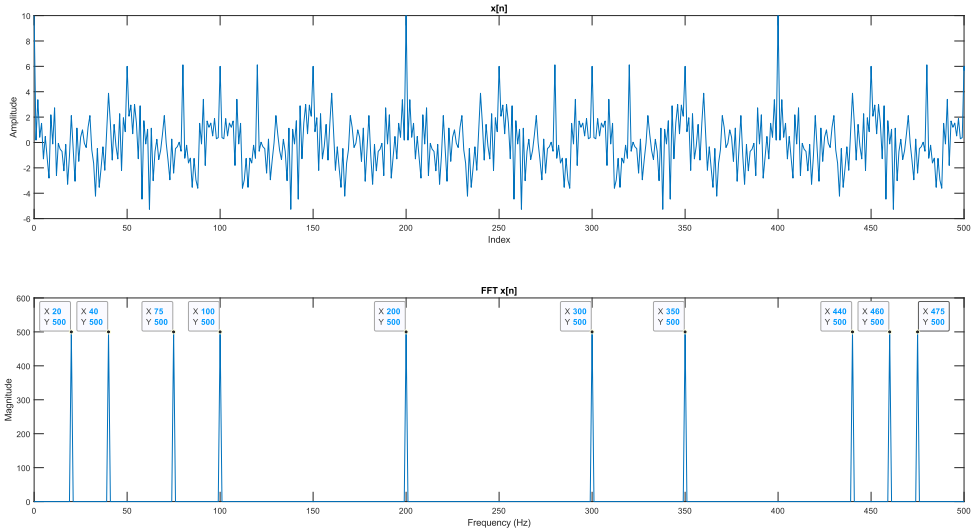
With the BPF designed, the next step was to use it to filter some test input signal. The input signal is created on line 59 and it consists of overlapping cosine waves sampled at 1 kHz of the following frequencies: 20 Hz, 40 Hz, 75 Hz, 100 Hz, 200 Hz, 300 Hz, 350 Hz, 440 Hz, 460 Hz, and 475 Hz. The FFT of the input is computed on line 60. The *filter()* function on line 62 filters the input signal through *BPF_Filter* to produce the output signal, upon which the FFT is computed on line 63. The final step is to graph everything which is was done on lines 66-81.

3. Results

3.1. Part A: Input Signal and its Frequency Spectrum.

Figure 1 shows a graph of 500 samples from the input signal as well as the first 500 samples from its 1000-point FFT. The spikes of the FFT occur at the frequencies mentioned above and are all the same amplitude. This makes sense this the input signal was composed of the superposition of cosines of equal magnitude at these frequencies.

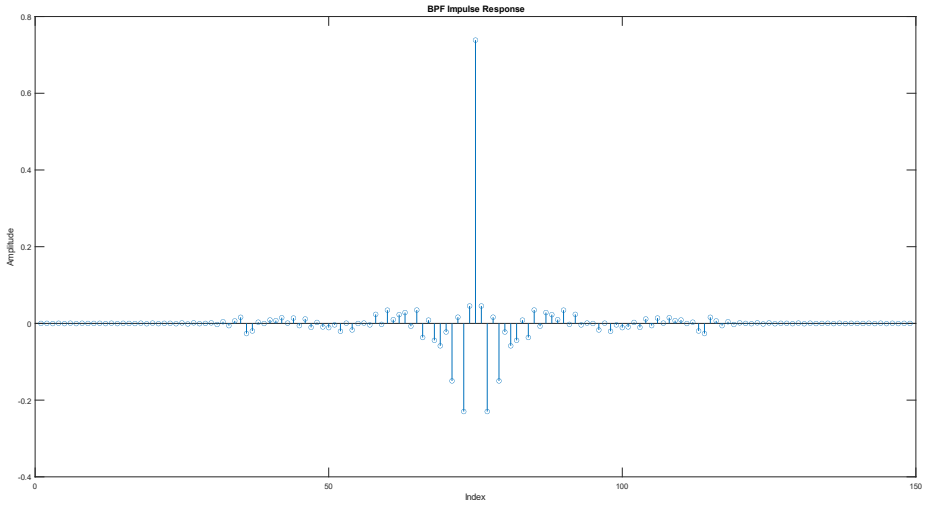
Figure 1: Input Signal and its FFT



3.2. Part B: Impulse, Magnitude, and Phase Response of BPF

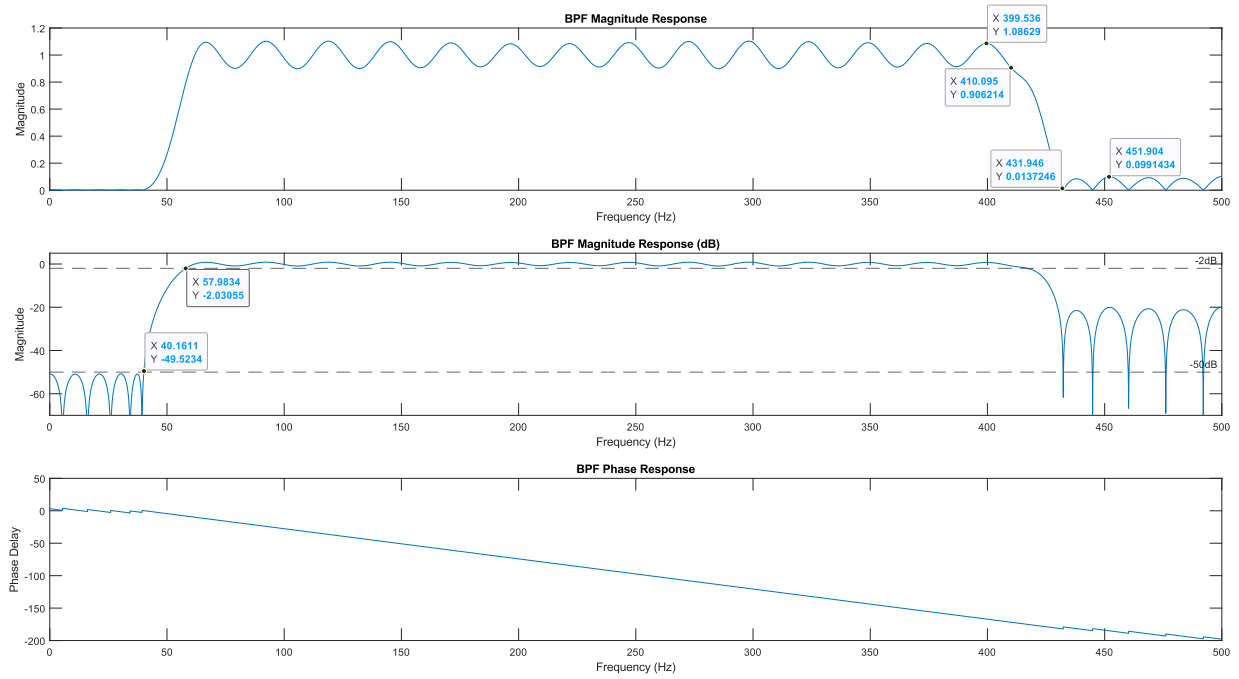
Figure 2 shows the impulse response of our BPF. It resembles a windowed sinc function which is what one would expect from the impulse response of a BPF.

Figure 2: BPF Impulse Response



To further verify the functionality of the BPF, the frequency response was graphed and is shown in Figure 3. Recall that the LPF portion was to have passband ripple of 0.01 and stopband ripple of 0.1. In the linear magnitude response, the stopband ripple seems to stay right below 0.1 which aligns with our filter design. However, the passband ripple reaches a value of roughly 0.09 which is greater than the 0.01 passband rippled specified by the design criteria. Nonetheless, the transition is roughly from 410 Hz to 431 Hz which closely aligns with the design criteria.

Figure 3: Magnitude (linear), Magnitude (dB), and Phase Responses

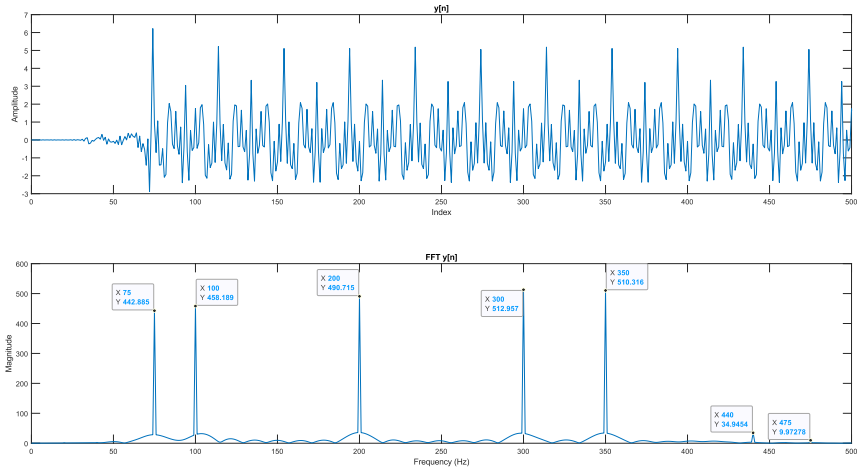


For the HPF portion, a maximum stopband attenuation of 50 dB and a maximum passband ripple of 2 dB were desired. Notice how in the second graph of Figure 3, both these criteria are satisfied since the HPF stopband ripple never goes above 50 dB and the passband ripple never goes below 2 dB. Therefore, the HPF filter does a great job attenuating the low frequency components while not distorting the frequency band of interest. Moreover, the graph also indicates that the HPF stopband edge frequency is 40 Hz and the HPF passband edge frequency is 57 Hz, closely aligning with the 40 Hz and 60 Hz that we were aiming for. The phase response shown in the third graph is clearly linear thereby satisfying the linear-phase criteria of the BPF.

3.3. Part C: Output Signal and its Frequency Spectrum

Lastly, the output signal and its frequency content were graphed in Figure 4.

Figure 4: Filtered Output Signal and its FFT



The output signal looks similar to the input, but there are very obvious differences. This is because the BPF attenuated some of the low and high frequency components while keeping the mid-level frequency components. Therefore, the output resembles the input because it has some, not all, of the original information still within it. This is reaffirmed in the plot of its FFT in which the 25 Hz, 40Hz, 440 Hz, 460 Hz, and 475 Hz magnitudes are much lower than the magnitudes of the other frequency components. The BPF needed to suppress frequencies below 50 Hz and above 420 Hz, and expectedly, our implementation does just that. All the frequencies within this range experience very little attenuation since they are in the band of frequencies that pass through the filter nearly unaffected. However, the linear phase response indicates that the delay of the output decreases linearly with frequency in this band as indicated in Figure 3.

4. Conclusion

This project was important as it provided students with insight into how to design a filter to satisfy a set of design considerations. This greatly supplemented the work done in lab which primarily focuses on filter implementation as opposed to filter design. Although the goal of the project was to design a BPF, students also gained a deeper understanding into LPFs, HPFs, and cascaded filters, thereby widening the scope of this project. Students designed and implemented their own BPF and analyzed how it attenuated certain frequency components of the sampled input signal. Moreover, they learned how filter parameters impacted the filter's frequency response. Overall, the skills and knowledge students learned from this project will create a useful set of tools that they can take with them into the workforce or graduate school.

5. Appendix

Bohr_Matthew_DSPDesign.m

```
clear
close all

%----------------filter paramters---------------%
Ft = 1000;                  % sampling freq

Fp_low = 410;               % lowpass passband freq
Fs_low = 430;               % lowpass stopband freq
delta_p_low = 0.01;         % lowpass passband ripple
delta_s_low = 0.1;          % lowpass stopband ripple

Fp_high = 60;               % highpass passband freq
Fs_high = 40;               % highpass stopband freq
As_high = 50;               % highpass stopband attenuation
Ap_high = 2;                % highpass passband ripple


%------------------computing lowpass filter paramters---------------------%
% use firpmord to get order N_low and normalized edge frequencies Fo_low
[N_low, Fo_low, mo_low, W_low] = firpmord([Fp_low Fs_low],[1 0],[delta_p_low delta_s_low],Ft);
% use firmp function to generate LPF coeffs using result from firpmord
LPF_coeffs = firpm(N_low, Fo_low, mo_low, W_low);
% use dfilt.dffir to construct digital filter object from LPF_coeffs
LPF_filter = dfilt.dffir(LPF_coeffs);
% storing frequency response of LPF_filter in LPF_mag
LPF_mag = freqz(LPF_filter);


%------------------computing highpass filter paramters--------------------%
% computing ripples in linear scale
delta_p_high = (10^(Ap_high/20)-1) / (10^(Ap_high/20)+1);
delta_s_high = (1+delta_p_high) / (10^(As_high/20));

% use firpmord to get order N_high and normalized edge frequencies Fo_high
```

```
[N_high, Fc_high, mo_high, W_high] = firpmord([Fs_high Fp_high],[0 1],[delta_s_high delta_p_high],Ft);
% use firmp function to generate HPF coeffs using result from firpmord
HPF_coeffs = firpm(N_high, Fc_high, mo_high, W_high);
% use dfilt.dffir to construct digital filter object from HPF_coeffs
HPF_filter = dfilt.dffir(HPF_coeffs);
% storing magnitude response of HPF_filter in HPF_mag
HPF_mag = freqz(HPF_filter);


%------------------computing bandpass filter responses--------------------%
% cascade LPF_filter and HPF_filter using dfilt.cascade() function
BPF_filter = dfilt.cascade(LPF_filter, HPF_filter);
% storing magnitude response of BPF_filter in BPF_mag
BPF_mag = freqz(BPF_filter);
% storing phase response of BPF_filter in BPF_phase
BPF_phase = phasez(BPF_filter);
% storing impulse response of BPF_filter in BPF_impulse
BPF_impulse = impz(BPF_filter);


%--------------------generating and filtering input-----------------------%
n = 0:Ft-1;                    % using 1000 samples for signal and FFT
k = 1:8192;
k1 = k * (500/8192);             % normalizing n between 0 and 1 for FFT plots
x = cos(2*pi*(20/Ft)*n) + cos(2*pi*(40/Ft)*n) + cos(2*pi*(75/Ft)*n) + cos(2*pi*(100/Ft)*n) +
cos(2*pi*(200/Ft)*n) + cos(2*pi*(300/Ft)*n) + cos(2*pi*(350/Ft)*n) + cos(2*pi*(440/Ft)*n) +
cos(2*pi*(460/Ft)*n) + cos(2*pi*(475/Ft)*n);
x_fft = fft(x);                % FFT of input signal

y = filter(BPF_filter, x);     % filtering x through BPF_filter and storing in y
y_fft = fft(y);                % FFT of filtered output


%---------------------------generating plots-------------------------------%
% figure 1: plot of input signal and its frequency content
figure(1); subplot(2,1,1); plot(n,x); title('x[n]'); xlim([0 500]), xlabel('Index'); ylabel('Amplitude');
           subplot(2,1,2); plot(n, abs(x_fft)); title('FFT x[n]'); xlim([0 500]), xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

```
% figure2 : impulse response of BPF
figure(2); stem(BPF_impulse); title('BPF Impulse Response'), xlabel('Index'); ylabel('Amplitude');

% figure 3: BPF magnitude response (linear and dB) and BPF phase response
figure(3); subplot(3,1,1); plot(k1, abs(BPF_mag)); title('BPF Magnitude Response'); xlim([0 500]);
xlabel('Frequency (Hz)'); ylabel('Magnitude');
          subplot(3,1,2); plot(k1, mag2db(abs(BPF_mag))); hold on; yline(-2, '--', '-2dB'); hold on;
yline(-50, '--', '-50dB'); hold off; xlim([0 500]); ylim([-70 5]); title('BPF Magnitude Response (dB)');
xlabel('Frequency (Hz)'); ylabel('Magnitude');
          subplot(3,1,3); plot(k1, BPF_phase); title('BPF Phase Response'); xlim([0 500]);
xlabel('Frequency (Hz)'); ylabel('Phase Delay');

% figure 4: plot of output signal and its frequency content
figure(4); subplot(2,1,1); plot(n, y); title('y[n]'); xlim([0 500]), xlabel('Index'); ylabel('Amplitude');
          subplot(2,1,2); plot(n, abs(y_fft)); title('FFT y[n]'); xlim([0 500]), xlabel('Frequency (Hz)');
ylabel('Magnitude');
```