EE 419 - Project 3
Discrete Fourier Transform and FFT Signal Processing

Names: Chris Adams & Aiku Shintani **Lab Date: 1/22/19**

Bench #: 9 **Section: 2**

1) Plotting the Magnitude Spectrum of a Discrete-Time Signal

```
[ DFTx, Fd ]=plot_DFT_mag(x,fsample,figure_num);
```

where the input arguments are:
- *x* = time domain samples of a discrete-time sequence
- *fsample* = sampling frequency (samples / second)
- *figure_num* = number of the figure to use for the two plots

and the outputs are:
- *DFTx* = DFT spectrum values (complex) [same # samples as x]
- *Fd* = Digital frequency values for each DFT sample

Design Specifications:

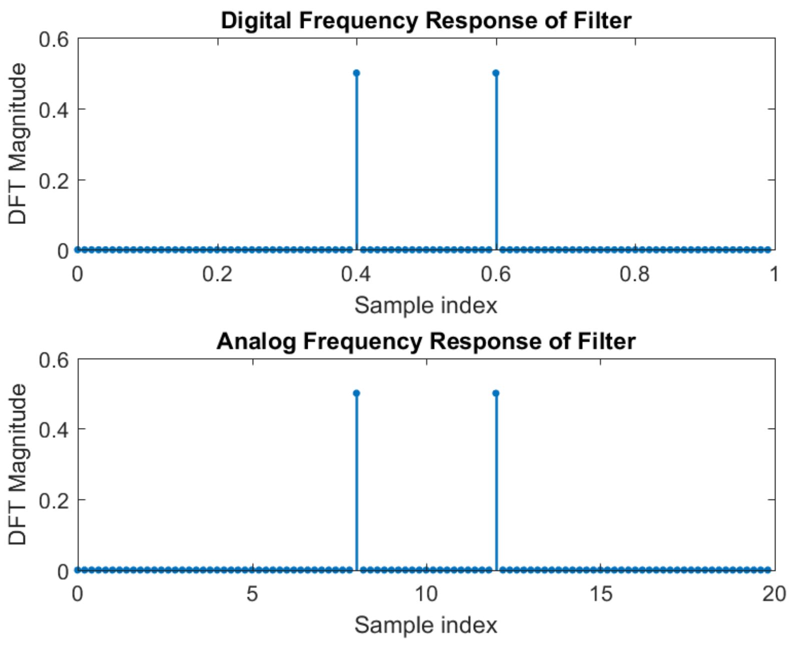
(1): Plot the DFT magnitude vs. **digital** frequency on the upper subplot.

(2): Plot the DFT magnitude vs. equivalent **analog** frequency in the lower subplot.

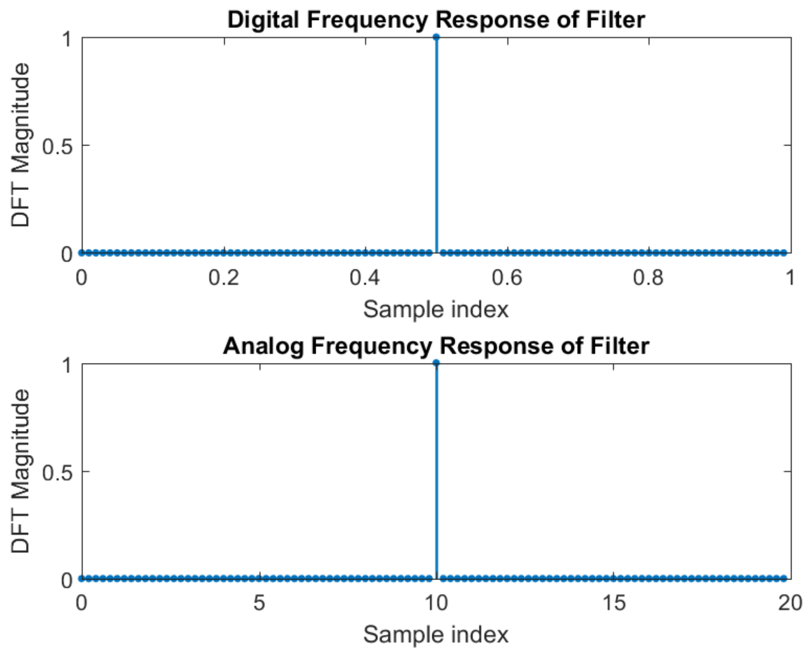
- Use **stem** plots for the Magnitude response, using the '.' marker.
- All plots use a **linear** scale for the **frequency** axis, and for the **magnitude** response
- **Normalize the magnitude responses** by dividing the DFT values by the # of samples.
- **Label** all axes appropriately, and provide a **title** for the each plot.

Test Cases:

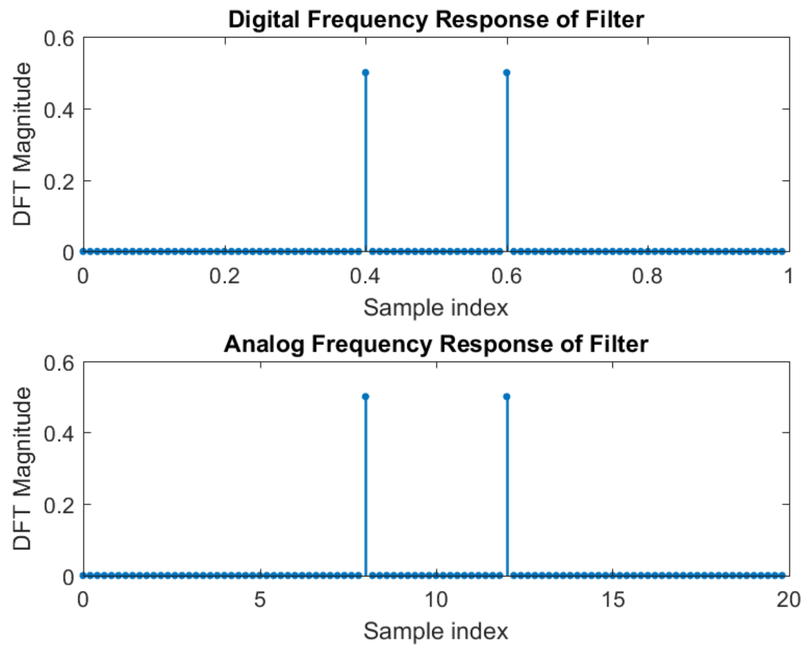
Test Case 1: Cosine function: freq= **8 Hz**, phase=0, amplitude=1, f_{sample}= 20 Hz, # samples=100;



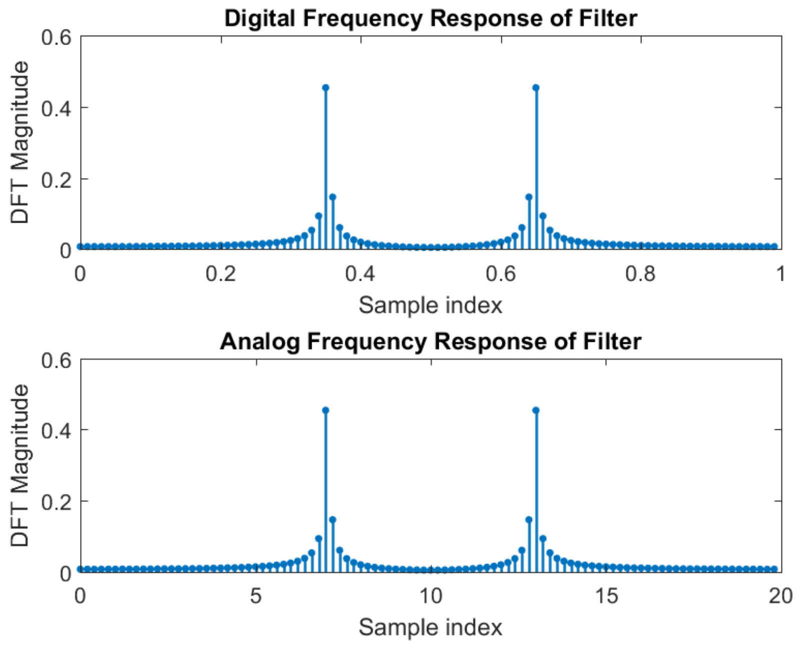
Test Case 2: Cosine function: freq= **10 Hz**, phase=0, amplitude=1, f_{sample}= 20 Hz, # samples=100;



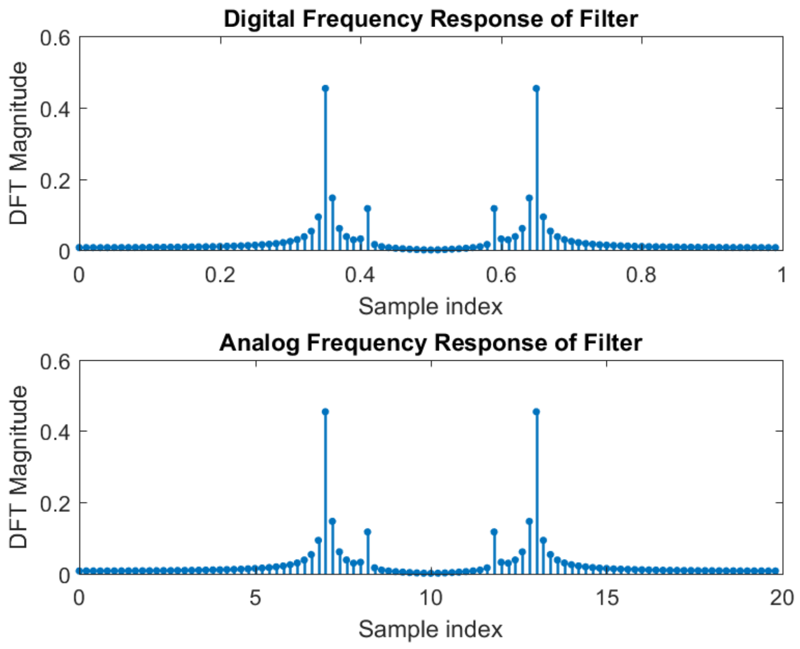
Test Case 3: Cosine function: freq= **12 Hz**, phase=0, amplitude=1, f_{sample}= 20 Hz, # samples=100;



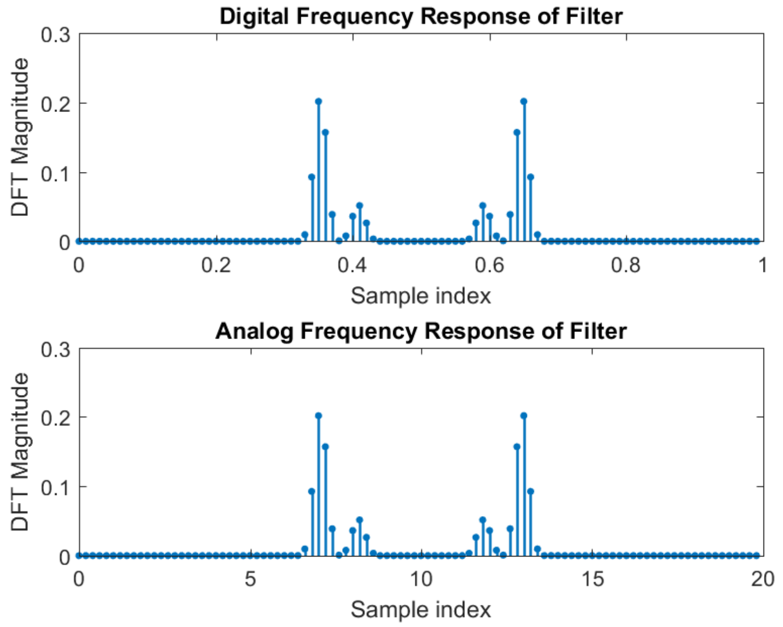
Test Case 4: Cosine function: freq= **7.05 Hz**, phase=0, amplitude=1, f_{sample}= 20 Hz, # samples=100;



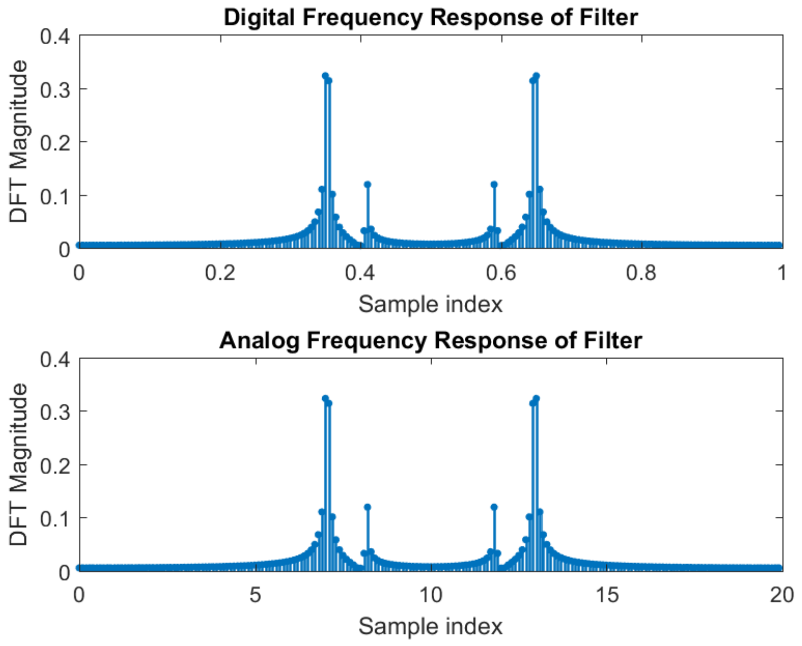
Test Case 5: Cosine function: freq= **7.05 Hz**, phase=0, amplitude=**1**, f_{sample}= 20 Hz, # samples=100 **added to** Cosine function: freq=**8.17** Hz, phase=0, amplitude=**0.25**, f_{sample}= 20 Hz, # samples=100;



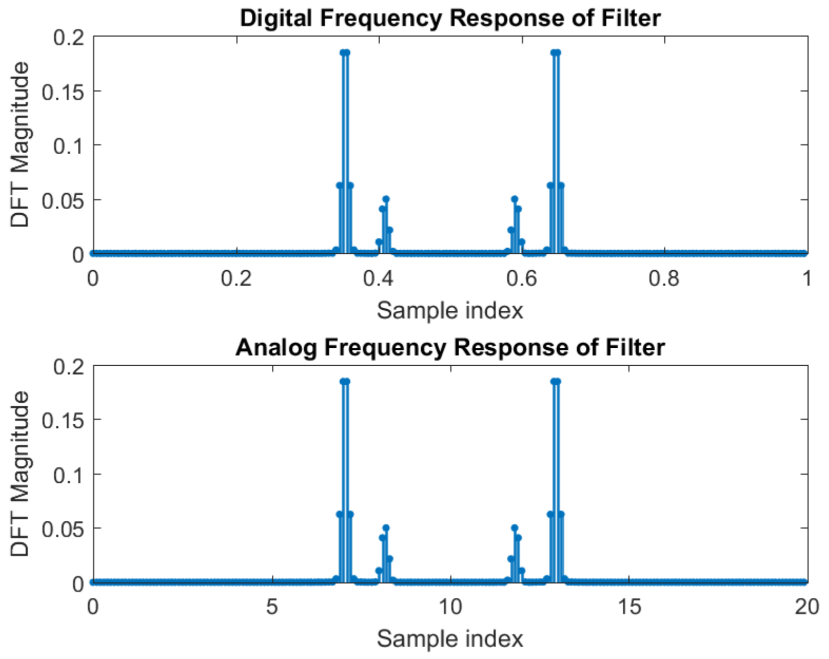
Test Case 6: Same as (5), but multiplied by a **Blackman window** (100 point window).



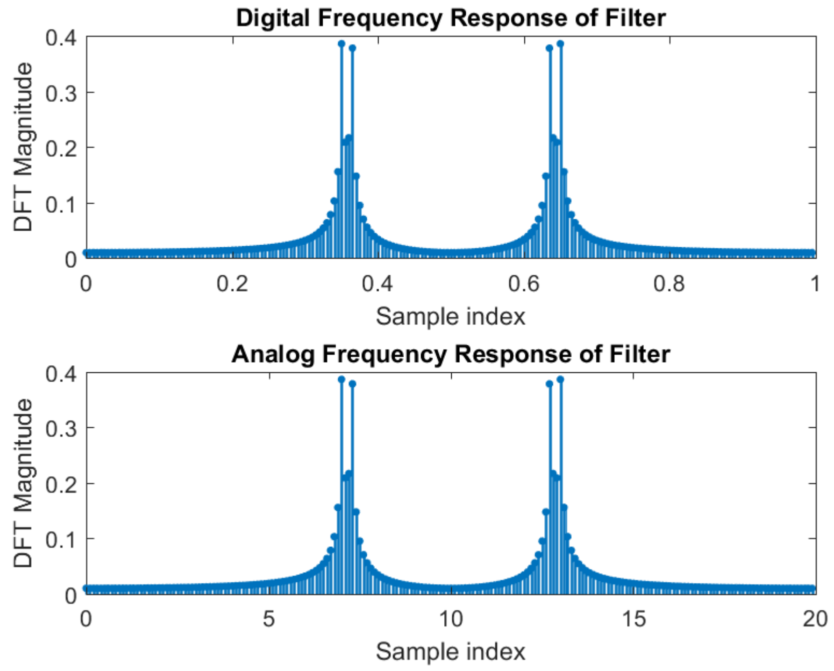
Test Case 7: Same as (5), but # samples = **200**; [2 signals, 200 samples, no window]



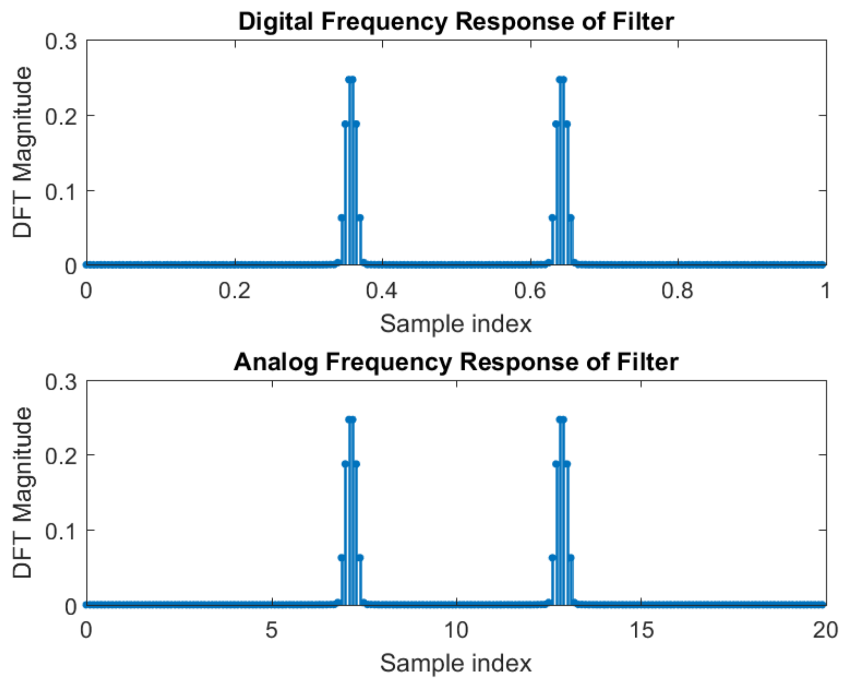
Test Case 8: Same as (6), but # samples = **200**; [2 signals, 200 samples, with window]



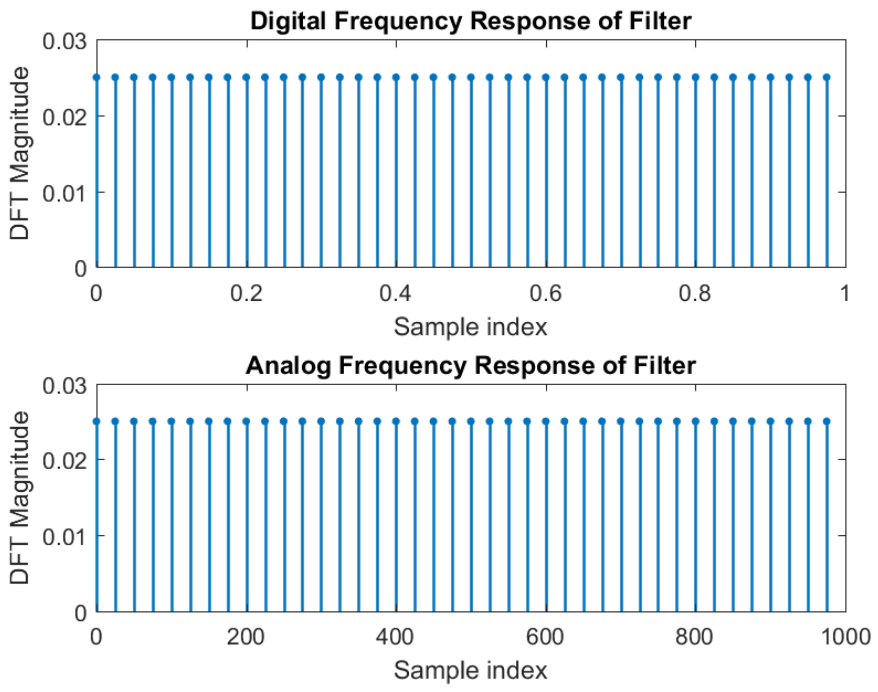
Test Case 9: Cosine function: freq= 7.05 Hz, phase=0, amplitude=1, fsample= 20 Hz, # samples=200 added to Cosine function: freq= **7.25** Hz, phase=0, amplitude=**1**, fsample= 20 Hz, # samples=200;



Test Case 10: Same as (9), but multiplied by a **Blackman window** (200 point window)

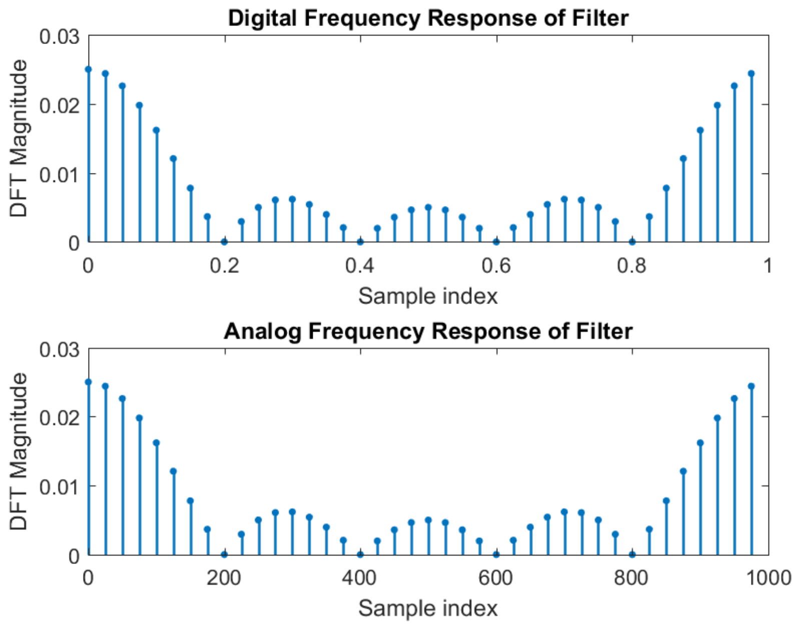


Test Case 11: Unit sample sequence: # samples = 40; fsample= 1 KHz;



Test Case 12: Unit sample response of the filter with difference equation:

y[n] = 0.2x[n]+0.2x[n-1]+0.2x[n-2]+0.2x[n-3]+0.2x[n-4] # samples=40; fsample=1KHz;



Answer the following questions based on your results:

1. Explain the position (frequency) of the cosine function's magnitude response peaks in Test Signal #3.
   In test signal #3, the sampling frequency is not twice the fundamental frequency of the test signal. This results in aliasing as shown in the following calculation: F = 12/20 = 0.6 > 0.5 (aliasing!). The aliasing frequency can be computed using the following calculation: Falias = F – 1 = -0.4. Falias = -0.4 results in a spike at F =0.6 as the negative frequencies are mirrored about F = 0.5. The spike at F = 0.4 also results from this aliasing.

2. Describe and explain the unusual spectrum of Test Signal #4.
   In test signal #4, the fundamental frequency of the cosine function is 7.05Hz while the number of samples is 100. Spectral leakage is introduced since the number of samples is not a perfect multiple of the fundamental frequency. The highest magnitude response is centered around the fundamental frequency as expected, with this leakage on both sides.

3. Compare the spectra of Test Cases #5 & 6. Explain what you observe about the ability to distinguish the spectrum of each signal, and why it is better in one case than the other.
   Test signal #6's DFT was computed using a Blackman window, which effectively reduced the spectral leakage (case #6 > case #5) created by both of the fundamental frequencies of the cosines of Test signal #5. The leakage is visibly decreased, in magnitude, when using the Blackman window.

4. Compare the spectra of Test Cases #5 & 6 to #7 & 8. Describe the effect of increasing the number of true sample values on the spectra.
   Increasing the number of true sample values, from test cases #5 & 6 to #7 &8, effectively improves the frequency resolution. As shown in the generated frequency response plots for test case #7, the spectral leakage around the two fundamental frequencies is present and covers a relatively large bandwidth. Compared to the plots for test case #7, the spectral leakage around the two fundamental frequencies is present but covers a smaller bandwidth. Less bandwidth of noise = better so increasing the number of true samples improves the resolution of the frequency response.

5. Compare the spectra of Test Cases #9 & 10. Explain what you observe about the ability to distinguish the spectrum of each signal, and why it is better in one case than the other.
For test cases #9 & 10, the input test signal includes two cosines with fundamental frequencies that are relatively close in value. When the frequency response is plotted for #9, with no use of the Blackman window, there are clearly four peaks (the biggest) in the response at the frequencies corresponding to the two fundamental frequencies. When the frequency response is plotted for #10, with use of the Blackman window, the magnitude response appears to only have two peaks (the biggest) in the response a frequency corresponding to the average of the two fundamental frequencies. The Blackman window is useful for extraction of spectral noise but this analysis shows its limitations in frequency resolution (case #9 > case #10).

6. Why does the unit sample sequence (Test Signal #11) have the frequency spectrum shown? Is it what you expected?
Yes, the unit sample response of test signal #11 looks as expected. There are 40 unit samples as seen, each with magnitude of 0.025 which is 1/40. The '1' in the numerator is the height of a unit sample and the '40' in the denominator is the number of samples.

7. What filter response characteristic is shown in the DFT results of Test Signal #12 (unit sample response)?
The DFT results of test signal #12 displays the response characteristic of a window function. From $0 \leq F \leq 0.2$***, the main lobe of the response is present. From*** $0.2 \leq F \leq 0.5$***, the side lobes of the response are present. The side lobe attenuation can be computed as follows:*** $-A_{SL} = -20\log(A_1/A_2) = -20\log(0.025/0.006) = -12.396\text{dB}$***. Based off this calculation, the window function can be considered a rectangular one.***

8. Based on the appearance of the DFT results of Test Signal #12, what sort of filter would you say created the tested unit sample response (low-pass, band-pass, high-pass, or band-stop)?
The DFT results of test signal #12 displays a lowpass characteristic. The magnitude response of the filter attenuates as the absolute value of the frequency is increased. This lowpass response is mirrored about $F=0.5$.

2) Fast Convolution Using FFT With Matlab

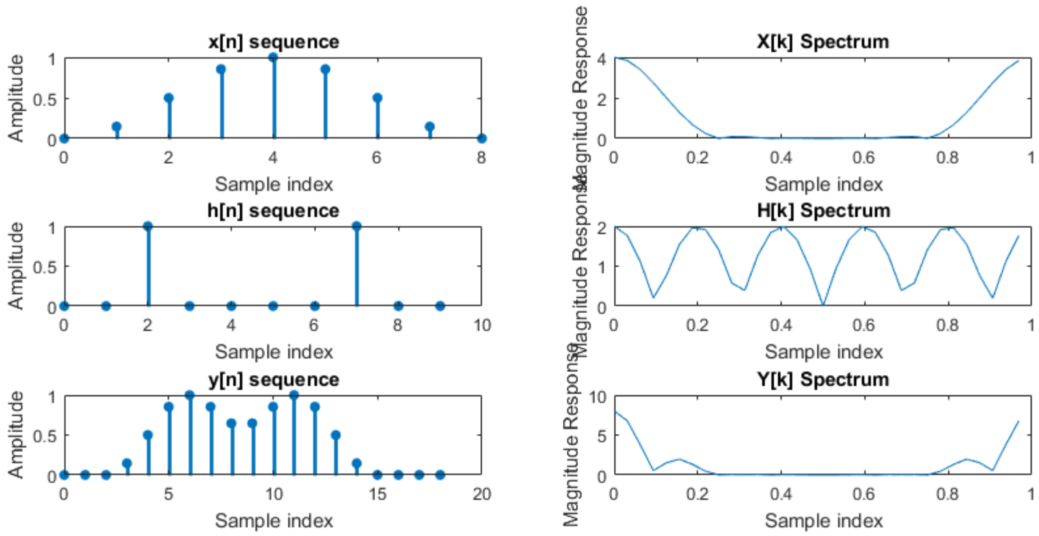
`function yn = fftconv( xn, hn )`

% Function to perform LINEAR CONVOLUTION using Fast Fourier Transforms
where the input arguments are:
xn = time domain samples of a discrete-time input signal
hn = time domain samples of a D-T system Unit Sample Response
and the output is:
yn = time domain samples of a discrete-time system output signal

Design Specifications:

- Use FFTs and inverse FFTs for computations
- Zero pad sequences to the correct length to achieve linear (not circular) convolution
- Use sequence lengths that are powers of 2 for fastest FFT computations
- Truncate convolution results to the correct length
- Plot the signals and DFT magnitude spectrum of each signal in a single figure
- Suppress plotting if the sequence lengths are too long.

Test Case: xn = a 9-point hann window; hn = [0 0 1 0 0 0 0 1 0 0]



M-File Listing:

```
function yn = fftconv(xn, hn)
% This function takes two input arguments: the xn coefficients and unit
% sample response. The function computes the FFT and returns one figure
% with 6 subplots: stem plots of x[n], h[n], and y[n] and plots of the
% magnitude responses for them.

Mh = length(hn);        %compute lengths
Mx = length(xn);
M = Mh + Mx - 1;        %number of samples for fft is sum of lenghts - 1
```

```matlab
n = 2^(nextpow2(M));   %for most efficient fft computation

k_x = [0 :1 : Mx - 1]; %for plotting x[n], x-axis
k_h = [0: 1: Mh - 1];  %for plotting h[n], x-axis
k_y = [0: 1: M - 1];   %for plotting y[n], x-axis

Xk = fft(xn, n);       %fft of x[n]
Hk = fft(hn, n);       %fft of h[n]
Yk = Xk.*Hk;           %convolution in time domain is multiplication in freq.

yn = real(ifft(Yk));   %get good parts of fft of y[n] after doing the inv fft

if (Mh > 1000) or (Mx > 1000) %check to see if excessive # of points to plot
    produce_plots = 0;
else
    %Generate plots
    figure(1)

    %make stem plot of input
    subplot(3, 2, 1)
    stem(k_x, xn, '.', 'MarkerSize', 20, 'Linewidth', 2);
    xlabel('Sample index')
    ylabel('Amplitude')
    title('x[n] sequence')

    %make stem plot of unit sample response
    subplot(3, 2, 3)
    stem(k_h, hn, '.', 'MarkerSize', 20, 'Linewidth', 2);
    xlabel('Sample index')
    ylabel('Amplitude')
    title('h[n] sequence')

    %make stem plot of output
    subplot(3, 2, 5)
    stem(0:M-1, yn(1:M), '.', 'MarkerSize', 20, 'Linewidth', 2);
    xlabel('Sample index')
    ylabel('Amplitude')
    title('h[n] sequence')

    %plot magnitude responses
    %plot Xk_mag
    Fd = (0:(length(Yk)-1))/length(Yk); %compute the sampled digital F, x-axis
    Xk_mag = abs(Xk);

    subplot(3, 2, 2)
    plot(Fd, Xk_mag)
    xlabel('Sample index')
    ylabel('Magnitude Response')
    title('X[k] Spectrum')

    %plot Hk_mag
    Hk_mag = abs(Hk);

    subplot(3, 2, 4)
    plot(Fd, Hk_mag)
    xlabel('Sample index')
    ylabel('Magnitude Response')
    title('H[k] Spectrum')

    %plot Yk_mag
    Yk_mag = abs(Yk);

    subplot(3, 2, 6)
    plot(Fd, Yk_mag)
    xlabel('Sample index')
    ylabel('Magnitude Response')
    title('Y[k] Spectrum')

end
```

3) [Lab] Verify the Multiple Notch 60 Hz Harmonic Filter Design

Design Specifications:

- Eliminate 60 Hz noise and all harmonics
- Multiple Notch Filter with -3dB Widths $\leq$ 6 Hz
- Passband Gain = 1.0 (+/- 2%)
- Sampling Frequency = 360 Hz
- Minimize # of poles and Zeros

Design Description:

a) Difference Equation:

y[n] = 0.905x[n] + 0.905x[n-1] + 0.905x[n-2] + 0.905x[n-3] + 0.905x[n-4] + 0.905x[n-5] – 0.96y[n-1] – 0.9216y[n-2] – 0.8847y[n-3] – 0.8493y[n-4] – 0.8154y[n-5]

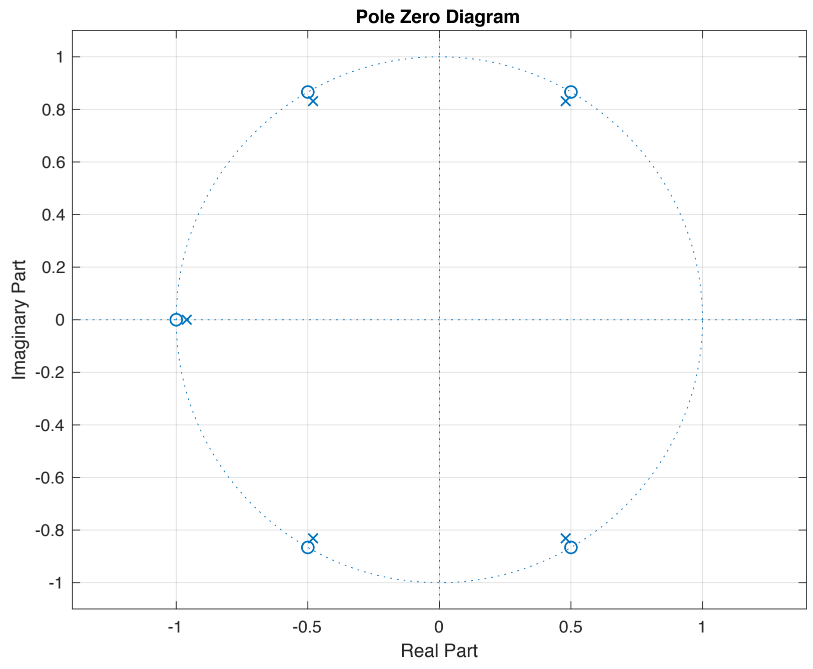
b) Zero locations?

Zeros = [$1e^{\pi j/3}$, $1e^{2\pi j/3}$, $1e^{\pi j}$, $1e^{4\pi j/3}$, $1e^{5\pi j/3}$]

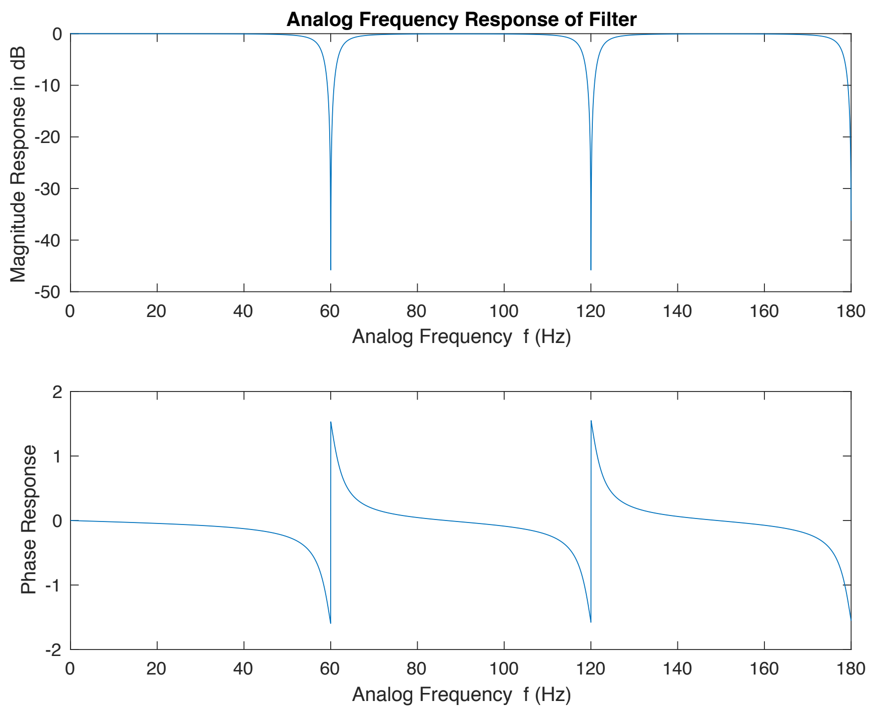
c) Pole locations?

Poles = [$0.96e^{\pi j/3}$, $0.96e^{2\pi j/3}$, $0.96e^{\pi j}$, $0.96e^{4\pi j/3}$, $0.96e^{5\pi j/3}$]

d) Pole / Zero Diagram:

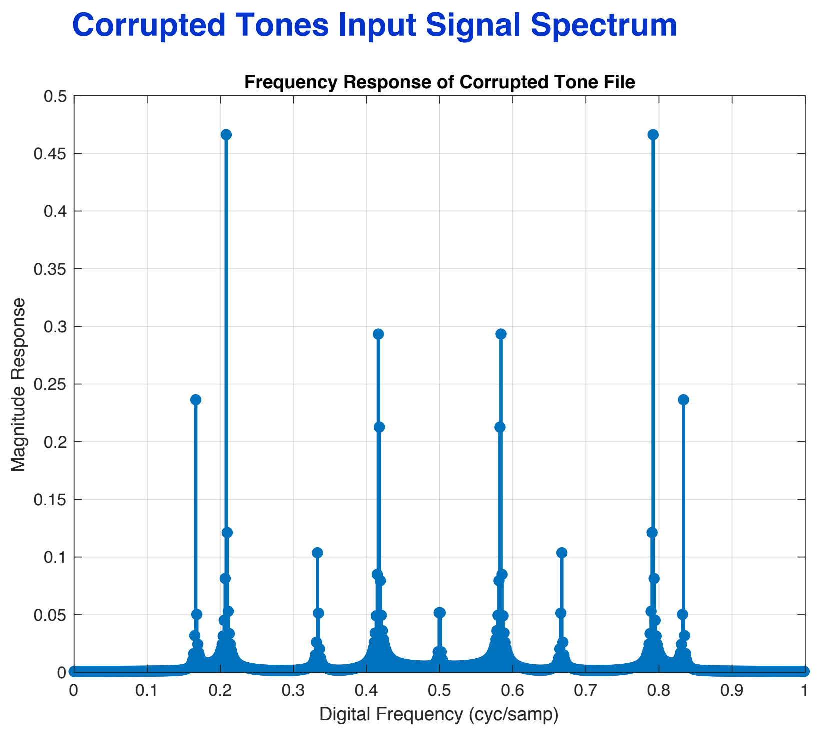


e) Frequency response (vs. Analog Frequency) plot:

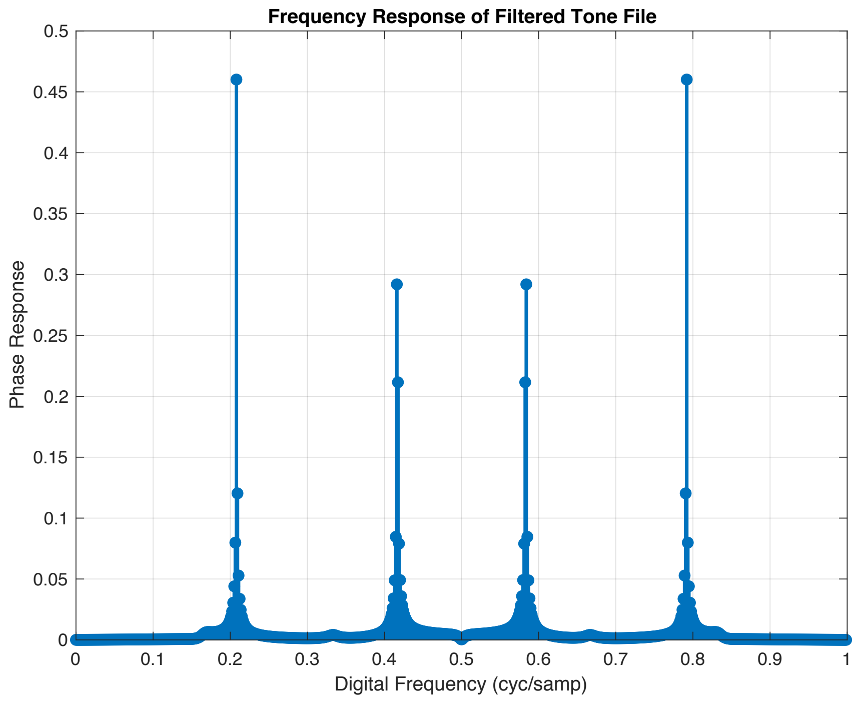


Design Verification Testing

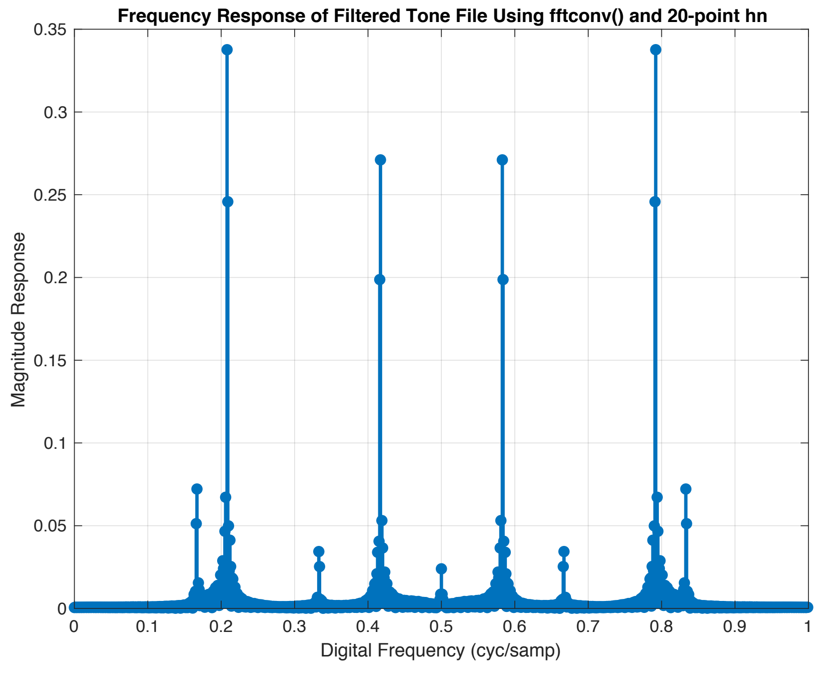
Test Case: corrupted_tones.mat



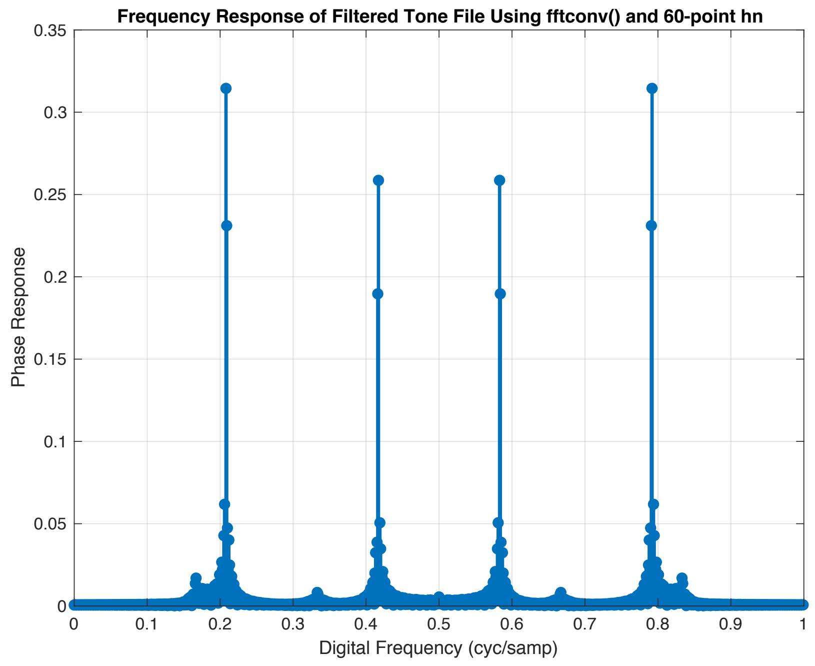
Filter Output Using filter()



Filter Output Using fftconv() and 20-point h[n]



Filter Output Using fftconv() and 60-point h[n]



Test Results Interpretation

- Explain any differences between the time-domain filtering results and each of the two frequency domain filtering tests. Which results were the most accurate? Why were there inaccuracies in some of these results?

  The most accurate filtering is a result of MATLAB's filter() function (time domain filtering) because it is passing the corrupted signal through the difference equation (the most data points). In the case of the fftconv() function created, it takes in a finite length h[n]. In this case, the filter was IIR, which means the h[n] was chopped off at 20 and 60 points. This leads to an inaccurate DFT of h[n] which is why more of the unwanted frequencies are still present in the filtered files (more leakage in 20-point than 60-point). The longer sequence of h[n] used the better the filtered result. Despite the fftconv() being less accurate, it has its advantages of being a faster method of computations by making use of the fft.

Verification Test Code:

```
load corrupted_tones;                     %load corrupted tone file into 'y'
[DFTy, Fdy] = plot_DFT_mag(y, 360, 1);                 %plot the DFT of 'y'
fsample = 360


%diff eq coefficients
Ak = [1 0.96 0.9216 0.8847 0.8493 0.8154];
Bk = [0.905 0.905 0.905 0.905 0.905 0.905];

y_filt = filter(Bk, Ak, y);               %filter corrupted file

[DFTy_filt, Fdy_filt] = plot_DFT_mag(y_filt, 360, 1);  %plot the DFT of
filtered y fsample = 360
```

```
figure
stem(Fdy, DFTy, '.', 'MarkerSize', 20, 'Linewidth', 2) %plot magnitude
response
grid

xlabel('Digital Frequency (cyc/samp)')
ylabel('Magnitude Response')
title('Frequency Response of Corrupted Tone File')

figure
stem(Fdy_filt, DFTy_filt, '.', 'MarkerSize', 20, 'Linewidth', 2) %plot
magnitude response
grid

xlabel('Digital Frequency (cyc/samp)')
ylabel('Phase Response')
title('Frequency Response of Filtered Tone File')

[hn1, n1] = unit_sample_response(Bk, Ak, 20, 1);      %obtain 20 point hn
[hn2, n2] = unit_sample_response(Bk, Ak, 60, 1);      %obtain 60 point hn

yn1 = fftconv(y, hn1);      %convolve corrupted signal with 20 pt hn
yn2 = fftconv(y, hn2);      %convolve corrupted signal with 60 pt hn

[DFTy1, Fdy1] = plot_DFT_mag(yn1, 360, 1);       %plot the DFT of
fftconvolved y fsample = 360
[DFTy2, Fdy2] = plot_DFT_mag(yn2, 360, 1);       %plot the DFT of
fftconvolved y fsample = 360

figure
stem(Fdy1, DFTy1, '.', 'MarkerSize', 20, 'Linewidth', 2) %plot magnitude
response
grid

xlabel('Digital Frequency (cyc/samp)')
ylabel('Magnitude Response')
title('Frequency Response of Filtered Tone File Using fftconv() and 20-point
hn')

figure
stem(Fdy2, DFTy2, '.', 'MarkerSize', 20, 'Linewidth', 2) %plot magnitude
response
grid

xlabel('Digital Frequency (cyc/samp)')
ylabel('Phase Response')
title('Frequency Response of Filtered Tone File Using fftconv() and 60-point
hn')
```

Conclusions:
Summarize one or two learnings about the computation, characteristics, limitations, uses, or proper methods needed to apply the Discrete Fourier Transform or Fast Fourier Transform that this project helped you understand better.

Name: Aiku Shintani

Conclusions: Through completion of this project, the functionality of the DFT and the FFT were confirmed via simulation. One of the fundamental limitations of the DFT is its property of repeating computations. The FFT is essentially a faster version of the DFT which uses an algorithmic approach to improve its efficiency. In order to correctly implement an FFT, the number of samples taken must be an exact power of 2. Matlab provides functions such as "fft()" and "nextpow2()" which allow the user to easily implement the most efficient computation of the DFT, the FFT.

The DFT is very useful in that it was the ability to compute a frequency response for one's input sequence, unit sample sequence, and ultimately the output sequence. In this lab, the input sequence and unit sample sequence were both 'FFT'ed' and then the two were multiplied (convolution in time domain is multiplication in the frequency domain) to obtain the frequency response of the output. The output frequency response was then inverse 'FFT'ed' using the "ifft()" function to obtain the output time domain response. By utilizing the time domain ←→ frequency domain transform properties, the output time domain response was obtained for a given input sequence and unit sample response. This skill can be translated for any combination of input and unit sample sequences and should be considered very useful.

Name: Chris Adams
Conclusions: Upon completing this project, how to implement DFT and FFT in Matlab were much better understood. Through the knowledge of how to use the DFT and FFT, filtering data becomes easier to accomplish (will become important for future projects in digital signal processing). One of the limitations of both the FFT and DFT is that to properly transform the data sequence, the sequence must be zero padded or sampled to the appropriate amount to reduce circular convolution and spectral leakage. The leakage is particularly evident in part 1 with the DFT. One way to help reduce the leakage is by applying a windowing function (blackman window in this experiment) which filters out the unwanted spectral components.

How to use the FFT to filter data quickly is another important take away from this project. While using the FFT to multiply the impulse response by the sampled input in the frequency domain is not necessarily the most accurate, it makes for a quick and reasonably accurate filtered output. This is because you must reduce the impulse response to a finite length even if it's a IIR filter. The longer the sequence is kept, the more accurate the response and slower the computations. It is clear through this project, that there are several tradeoffs when determining when to use the DFT/FFT and how one should go about filtering data.