Koby Miller Lab8

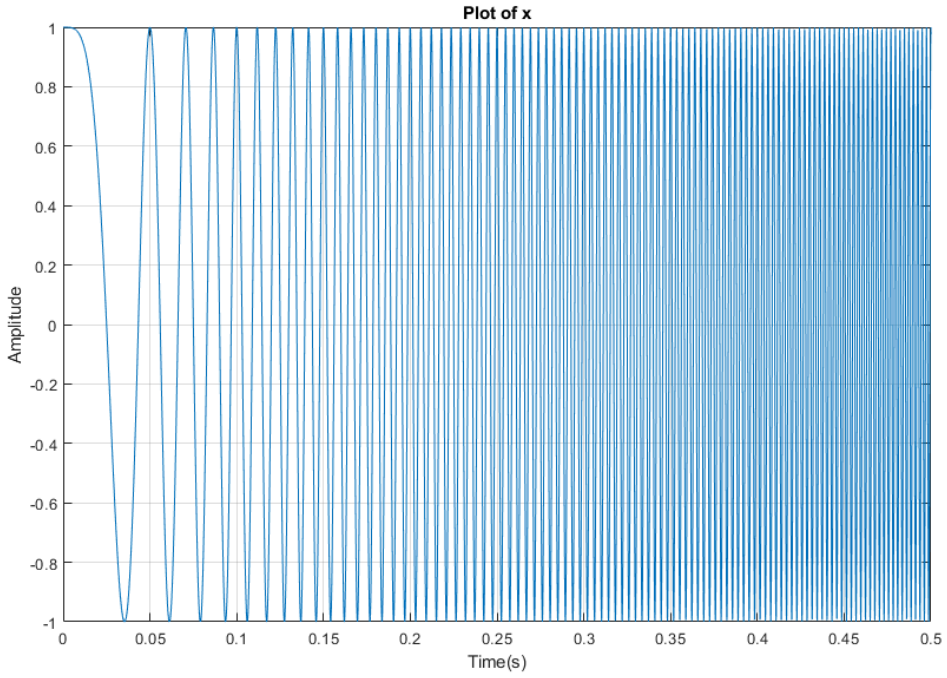
Table of Contents

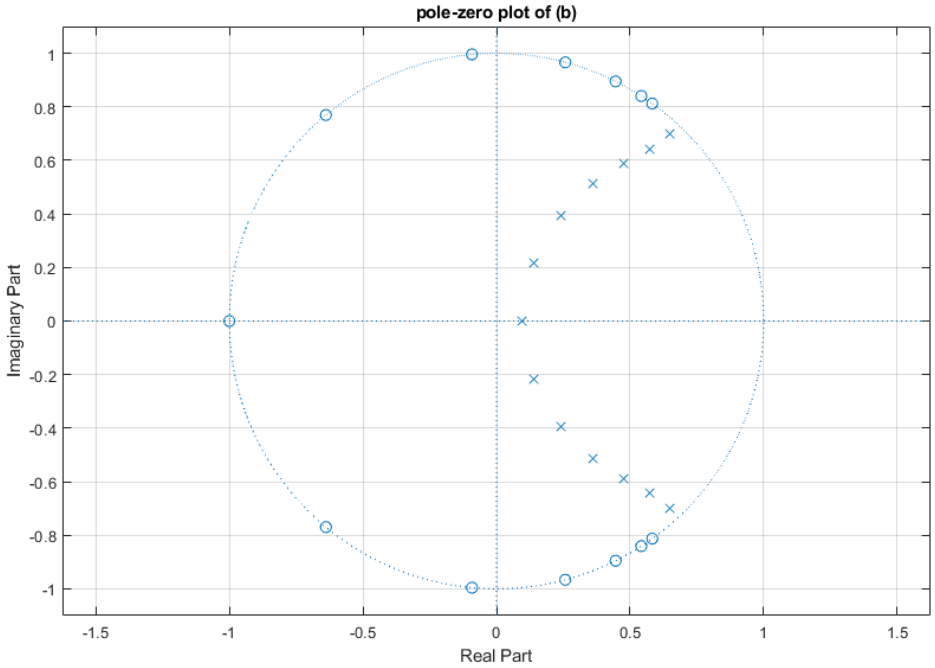
Exercise 8.1

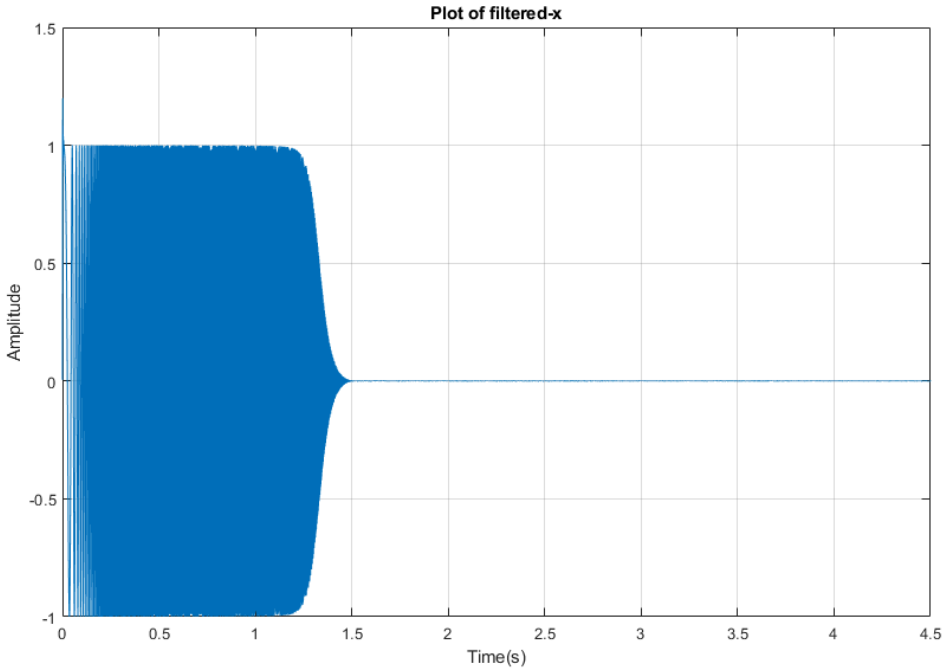
```
%(a)

fs = 8000;
tt = 0:1/fs:4.5;
x = cos(2*pi/20*fs*tt.^2);

figure(1);
plot(tt, x);
xlim([0,.5]);
xlabel('Time(s)');
ylabel('Amplitude');
title('Plot of x');
grid on;

x_scaled = x/max(abs(x));
%soundsc(x_scaled, fs);

% The signal sounds starts at a lower pitch and constantly rises in
% pitch for 4.5 seconds. It is hard to see unless you zoom into the
% plot at different times, but as time goes on, the period of the
% signal decreases. This means that the frequency is rising which
% makes sense because the pitch is rising.



%(b)
% The order of the resulting IIR filter is 13.
% The transfer function is:
% y[n] - 5.1523y[n-1] + 13.6722y[n-2] - 23.6126y[n-3] + 29.2649y[n-4]
% - 27.1830y[n-5] + 19.3691y[n-6] - 10.6593y[n-7] + 4.5197y[n-8]
% - 1.4521y[n-9] + 0.3437y[n-10] - 0.0563y[n-11] + 0.0058y[n-12]
% - 0.0003y[n-13] =
% 0.0033x[n] - 0.0040x[n-1] + 0.0119x[n-2] - 0.0057x[n-3]
% + 0.0140x[n-4] + 0.0013x[n-5] + 0.0090x[n-6] + 0.0090x[n-7]
% + 0.0013x[n-8] + 0.0140x[n-9] - 0.0057x[n-10] + 0.0119x[n-11]
% - 0.0040x[n-12] + 0.0033x[n-13]
```

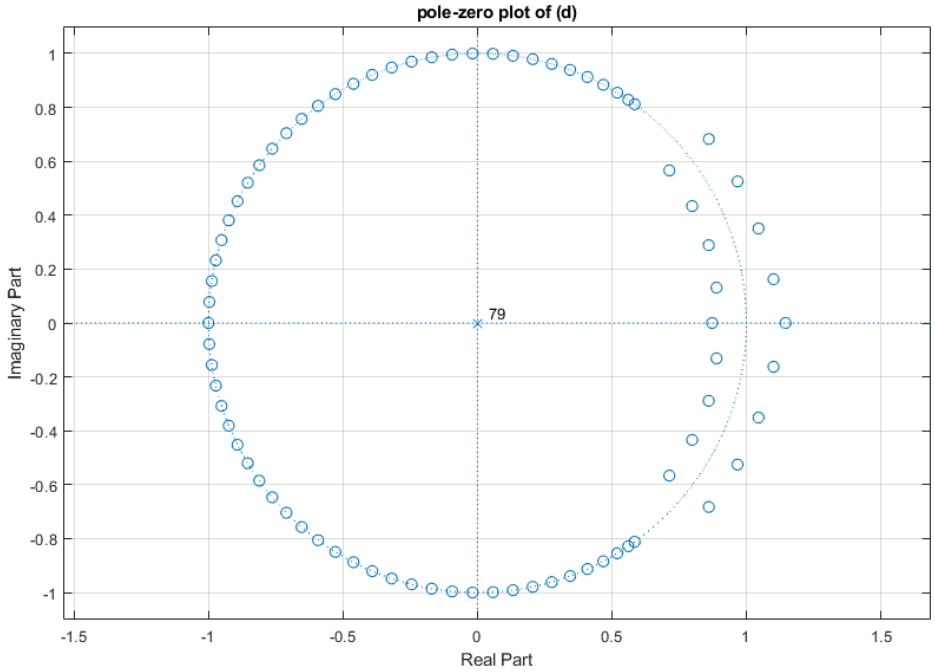
```
% basically: a_k*y[n-k] = b_k*x[n-k] for k = 0 to 13
% b = numerator
% a = denominator

% <<MagnitudeResponse.PNG>>

figure(2);
zplane(b,a);
title('pole-zero plot of (b)');
grid on;


%(c)
filtered_x = filter(b,a,x);

figure(3);
plot(tt, filtered_x);
xlim([0,4.5]);
xlabel('Time(s)');
ylabel('Amplitude');
title('Plot of filtered-x');
grid on;

filtered_x_scaled = filtered_x/max(abs(filtered_x));
%soundsc(filtered_x_scaled, fs);

% After listening to filtered_x_scaled, you can hear that the
% higher pitched notes are gone. Basically everything after 1.5
% seconds has been filtered out. This is where there were higher
% frequencies and since we had a lowpass filter, we kept the lower
% frequencies


%(d)
% part (b)
% The order of the resulting IIR filter is 79.
% The transfer function is really long. Just do the same as before.

% <<MagnitudeResponseD.PNG>>

figure(4);
zplane(FIR);
grid on;
title('pole-zero plot of (d)');

% part (c)

filtered_x_d = filter(FIR,1,x);
%soundsc(filtered_x_d, fs);

figure(5);
plot(tt, filtered_x_d);
xlim([0,4.5]);
```

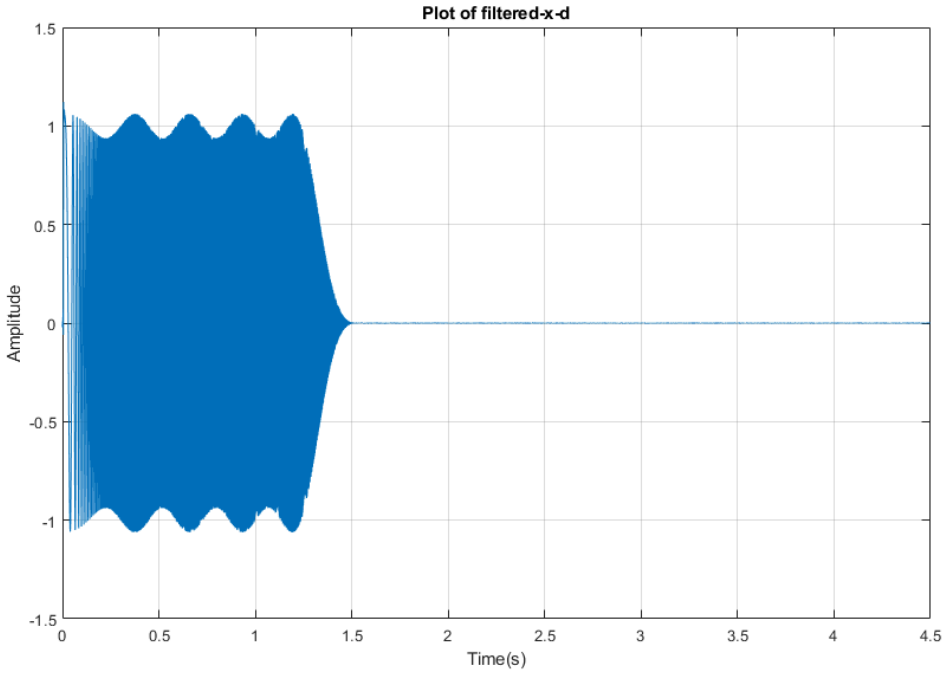
```
xlabel('Time(s)');
ylabel('Amplitude');
title('Plot of filtered-x-d');
grid on;

filtered_x_d_scaled = filtered_x_d/max(abs(filtered_x_d));
%soundsc(filtered_x_d_scaled, fs);

% The FIR filter is more complex to implement. It has a way larger
% transfer function. The resulting plots are very similar but you can
% see that the FIR filter is sort of 'wavy'. I honestly can't really
% hear the difference between the resulting filters. I would hope
% that the FIR filter would sound better to balance out it being
% more complext to implement.
```




pole-zero plot of (b)
Imaginary Part
Real Part
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.5
-1
-0.5
0
0.5
1
1.5


Plot of filtered-x
Amplitude
Time(s)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5



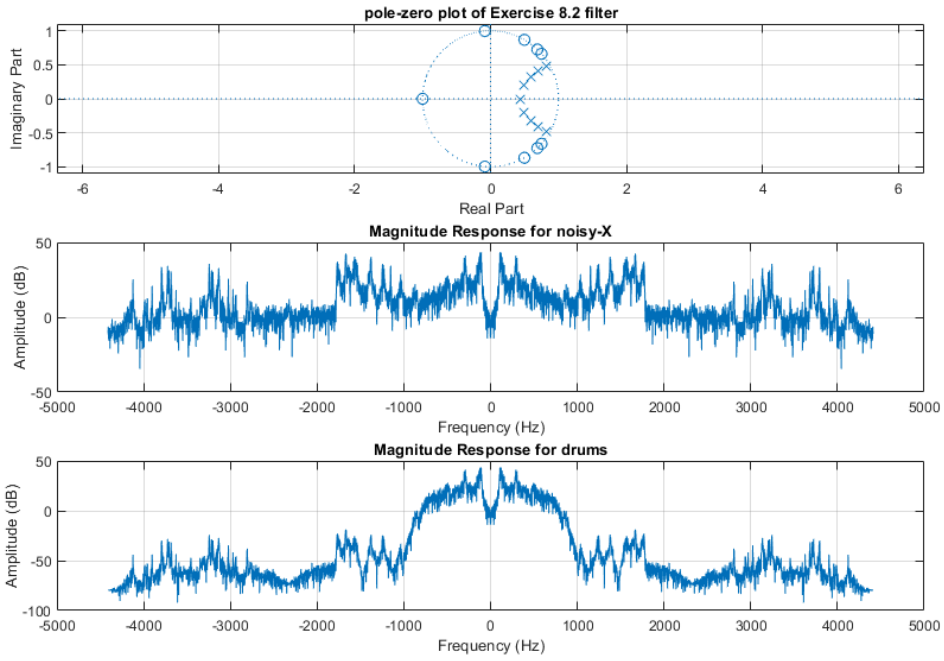


Exercise 8.2

```
[noisy_orig, fs] = audioread('noisy_drum_flute.wav');
noisy_orig = noisy_orig.';
%soundsc(noisy_orig, fs);
```

```
w = -pi:pi/2000:pi;

noisy_X = dtft(noisy_orig, w); %w is still 2000

figure(6);
subplot(3,1,1)
zplane(num,den);
grid on;
title('pole-zero plot of Exercise 8.2 filter');

subplot(3,1,2)
plot(w*fs/(2*pi), 20*log10(abs(noisy_X)));
grid on;
title('Magnitude Response for noisy-X');
xlabel('Frequency (Hz)');
ylabel('Amplitude (dB)');

drums = filter(num,den,noisy_orig);
drums_scaled = drums/max(abs(drums));
%soundsc(drums_scaled, fs);

drums_x = dtft(drums, w);

subplot(3,1,3)
plot(w*fs/(2*pi), 20*log10(abs(drums_x)));
grid on;
title('Magnitude Response for drums');
xlabel('Frequency (Hz)');
ylabel('Amplitude (dB)');

%soundsc(drums_scaled, fs);
%audiowrite('drum.wav', drums_scaled, fs);

% I designed a low-pass filter because I wanted to keep the low
% frequency drums and take out the flute/interference. Also because
% I decided to test the filter we made in the begining of this lab
% and it was already working pretty well. I got most of the drums out
% but if I try to tune it any more, then the lower flute frequencies
% will be filtered out.
%
% I created an IIR filter because it wanted us to minimize the order,
% and as we saw in the beginning, the IIR filter had a smaller order.
% This filter has an ordre of
%
% For the transfer function, we can do the same thing as in the first
% part, but now our order is 9: a_k*y[n-k] = b_k*x[n-k] for k = 0 to 9
%
% You can see the differences between the input and output DTFTs in
% figure 6. The input has frequencies from 0Hz to about 4500Hz that
% we can hear, but after going through the filter, we can only hear
% frequencies up until almost 1000Hz. Every other frequency is
% shifted down 60dB. Any higher frequencies would include the flute
% or the interference.
```

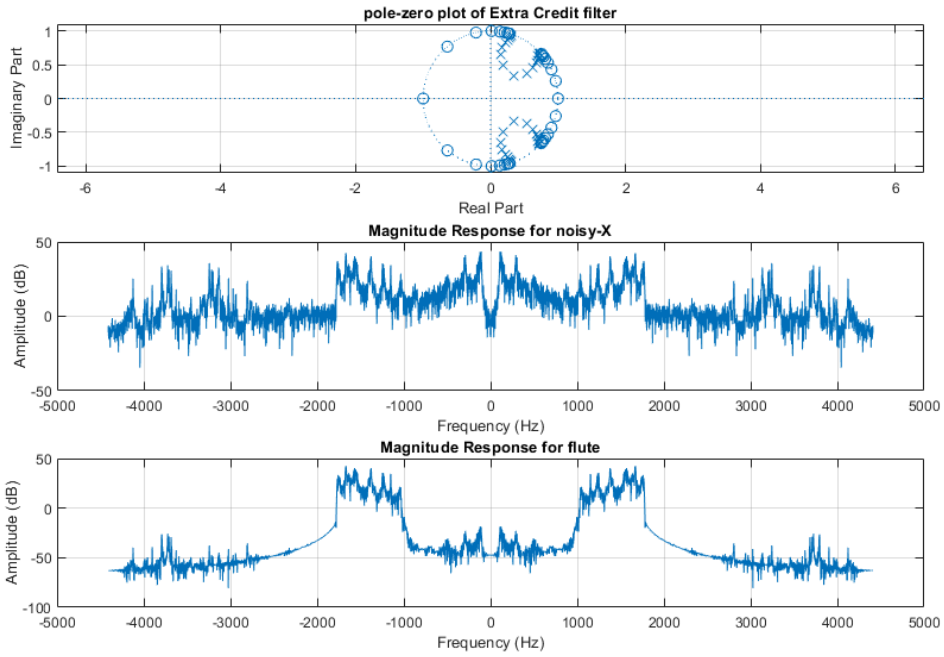
```
% <<MagnitudeResponseDrums.PNG>>
% <<PhaseResponseDrums.PNG>>
%
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
```



Extra Credit

```
figure(7);
subplot(3,1,1)
zplane(numerator,denominator);
grid on;
title('pole-zero plot of Extra Credit filter');

subplot(3,1,2)
plot(w*fs/(2*pi), 20*log10(abs(noisy_X)));
```

```
grid on;
title('Magnitude Response for noisy-X');
xlabel('Frequency (Hz)');
ylabel('Amplitude (dB)');

flute = filter(numerator,denominator,noisy_orig);
flute_scaled = flute/max(abs(flute));
%soundsc(drums_scaled, fs);

flute_x = dtft(flute, w);

subplot(3,1,3)
plot(w*fs/(2*pi), 20*log10(abs(flute_x)));
grid on;
title('Magnitude Response for flute');
xlabel('Frequency (Hz)');
ylabel('Amplitude (dB)');

%soundsc(flute_scaled, fs);
%audiowrite('flute.wav', flute_scaled, fs);

% I designed a Bandpass filter because I needed to filter out the low
% frequency drums and the high frequency interference.
%
% I used an IIR filter again for the same reason. The order is
% smaller. The order of this filter is 34.
%
% You can see the differences between the input and output DTFTs in
% figure 7. The input has frequencies from 0Hz to about 4500Hz that
% we can hear, but after going through the filter, we can only hear
% frequencies between around 1000Hz to 1800Hz. Lower frequencies
% would include the drums and higher frequencies would include
% the interference.

% <<MagnitudeResponseFlute.PNG>>
% <<PhaseResponseFlute.PNG>>
%
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
            %
```



My Functions

```
function X = dtft(x, w)
    X = 0;

    for a = 1 : length(x)
       X = X + (x(1,a) * exp(-1j*w*a-1));
    end

end
```

My Filter Settings

Published with MATLAB® R2020a