[Connor Dupuis]

Table of Contents

[Friday 1:55pm] - [28944] - [Naoki Sawahashi]

QUESTION 1 COMMENTING

```
% DO NOT REMOVE THE LINE BELOW
clear; close all; clc;
```

QUESTION 2 Thinking in Three Domains 1

```
% LOAD AUDIO
[x, fs] = audioread('music.wav');

% DEFINE AXES
w = -pi:pi/8000:pi-pi/8000;
```

```
t = 1/fs:1/fs:length(x)/fs;

% DEFINE POLES

mypoles = [ ...
        0.8*exp(1j*0.05*pi) ...
        0.8*exp(-1j*0.05*pi) ...
        0.85*exp(1j*0.06*pi) ...
        0.85*exp(-1j*0.06*pi) ...
        0.85*exp(1j*0.08*pi) ...
        0.85*exp(-1j*0.08*pi) ...
        0.8*exp(1j*0.09*pi) ...
        0.8*exp(-1j*0.09*pi) ...
        0.9*exp(1j*0.07*pi) ...
        0.9*exp(-1j*0.07*pi) ...
         ...
         ];

% DEFINE ZEROS
myzeros = [ ...
        0.99*exp(1j*0.15*pi) ...
        0.99*exp(-1j*0.15*pi) ...
        0.99*exp(1j*0.225*pi) ...
        0.99*exp(-1j*0.225*pi) ...
        1*exp(1j*0.3*pi) ...
        1*exp(-1j*0.3*pi) ...
        0.96*exp(1j*0.4*pi) ...
        0.96*exp(-1j*0.4*pi) ...
        0.90*exp(1j*0.6*pi) ...
        0.90*exp(-1j*0.6*pi) ...
        0.85*exp(1j*0.4*pi*2) ...
        0.85*exp(-1j*0.4*pi*2) ...
         ...
         ];

% CONVERT POLES AND ZEROS INTO COEFFICIENTS
[b,a] = pz2ba(mypoles,myzeros);
y = filter(b,a,x);

X = DTFT(x,w);
Y = DTFT(y,w);
```

PLAY MUSIC

```
disp('Playing Original Music ... ')
soundsc(x, fs)
pause(length(x)/fs*1.1)

disp('Playing Filtered Music ... ')
soundsc(y, fs)
```

Playing Original Music ...
Playing Filtered Music ...

2 (a) Answer question

```
% I created a low pass filter because we only want to pass the bass
 which
% has a lower frequency than the other instruments.
```

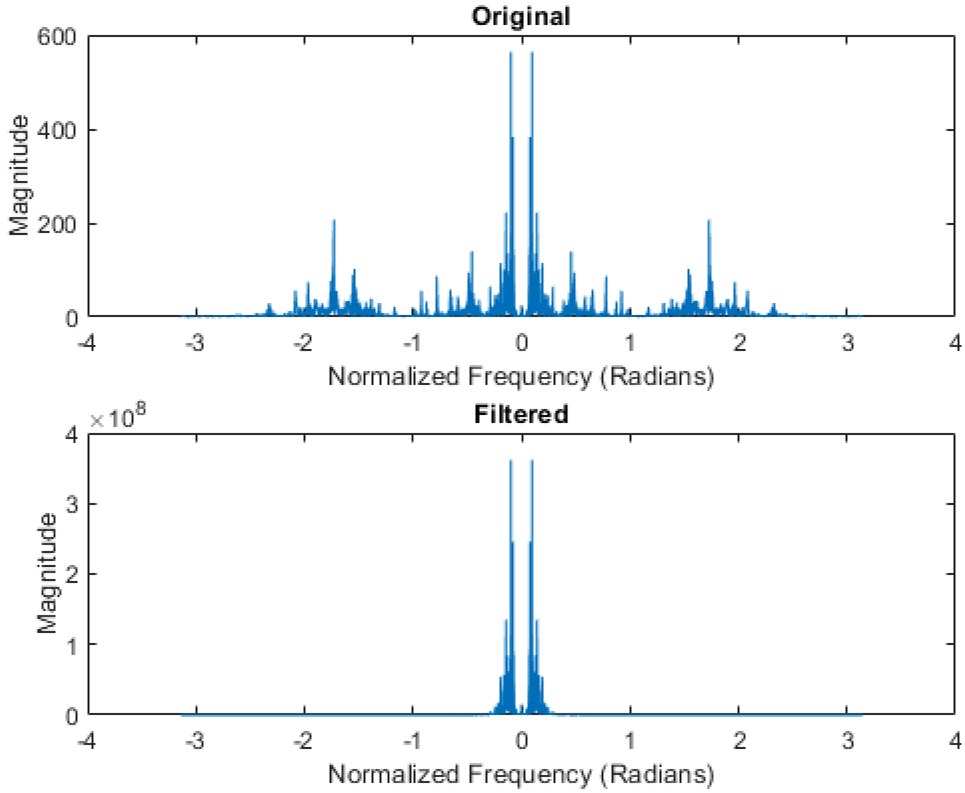
2 (b) Answer question

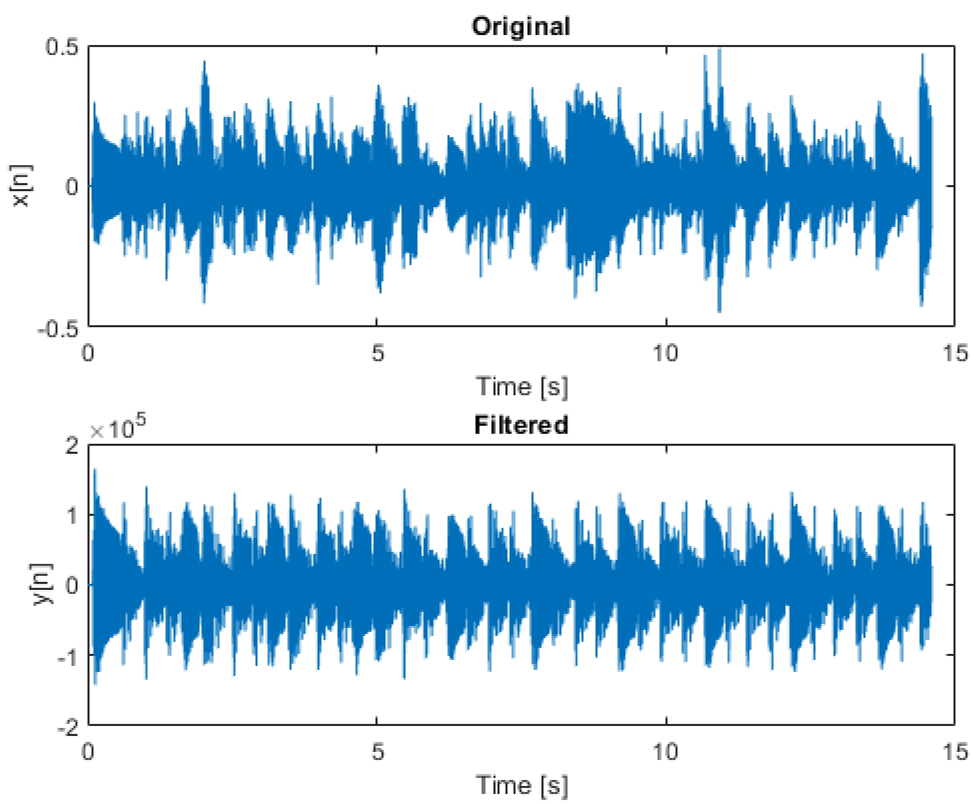
```
% I created a IIR filter because they are simpler to make, faster, and
 in
% this case would acheive a similar result.
```

2 (c) Plot inputs and outputs

```
% PLOT FREQEUNCY DOMAIN AND
figure
subplot(211)
plot(w,abs(X))
xlabel('Normalized Frequency (Radians)')
ylabel('Magnitude')
title('Original')
subplot(212)
plot(w,abs(Y))
xlabel('Normalized Frequency (Radians)')
ylabel('Magnitude')
title('Filtered')

% PLOT TIME DOMAIN
figure
subplot(211)
plot(t,x)
xlabel('Time [s]')
ylabel('x[n]')
title('Original')
subplot(212)
plot(t,y)
xlabel('Time [s]')
ylabel('y[n]')
title('Filtered')
```


Original
Magnitude
600
400
200
0
-4
-3
-2
-1
0
1
2
3
4
Normalized Frequency (Radians)
Filtered
×10^8
Magnitude
Normalized Frequency (Radians)


Original
x[n]
0.5
0
-0.5
0
5
10
15
Time [s]
Filtered
×10^5
y[n]
Time [s]

2 (d) Answer question

```
% The time domains show the output at a specific time. In the filtered
% song, the time domain shows when the bass is being played. The
 frequency domain
% shows us the magnituide of specific frequencies of the instruments
 in the
% song. The filtered plot shows that only frequencies in the low range
% are being outputted.
```

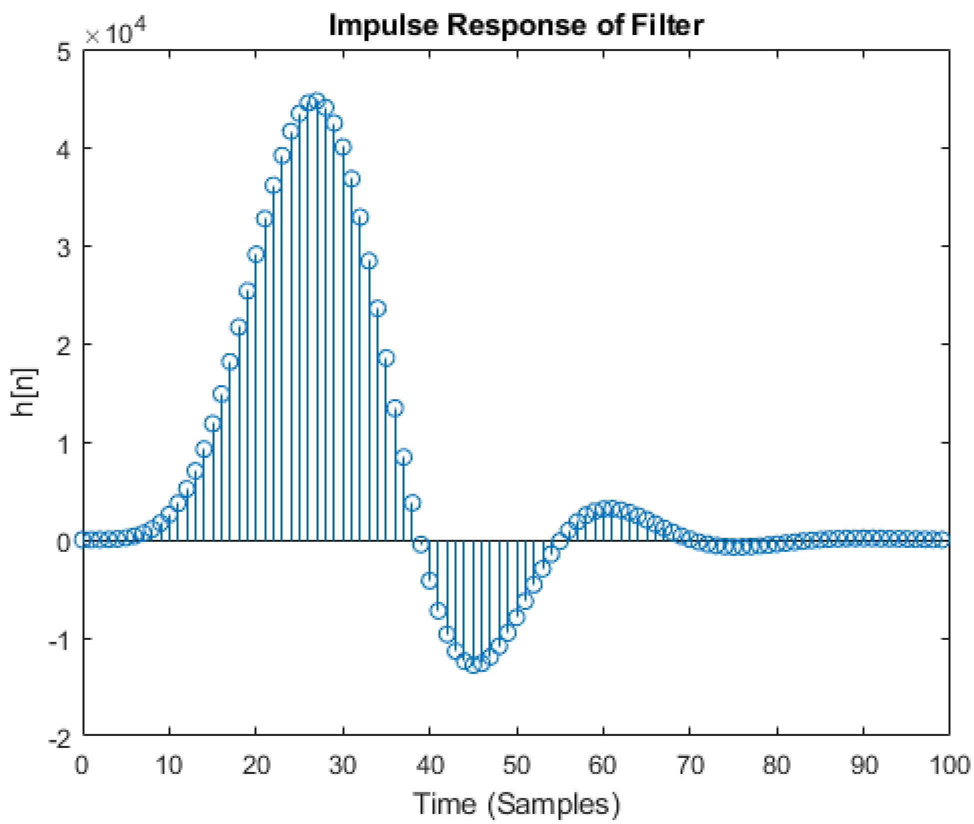
2 (e) Plot impulse response

```
N = 100;
n = 0:(N-1);

x_i = zeros(N,1);
x_i(1) = 1;

y_i = filter(b,a,x_i);

figure
stem(n,y_i)
xlabel('Time (Samples)')
ylabel('h[n]')
title('Impulse Response of Filter')
```



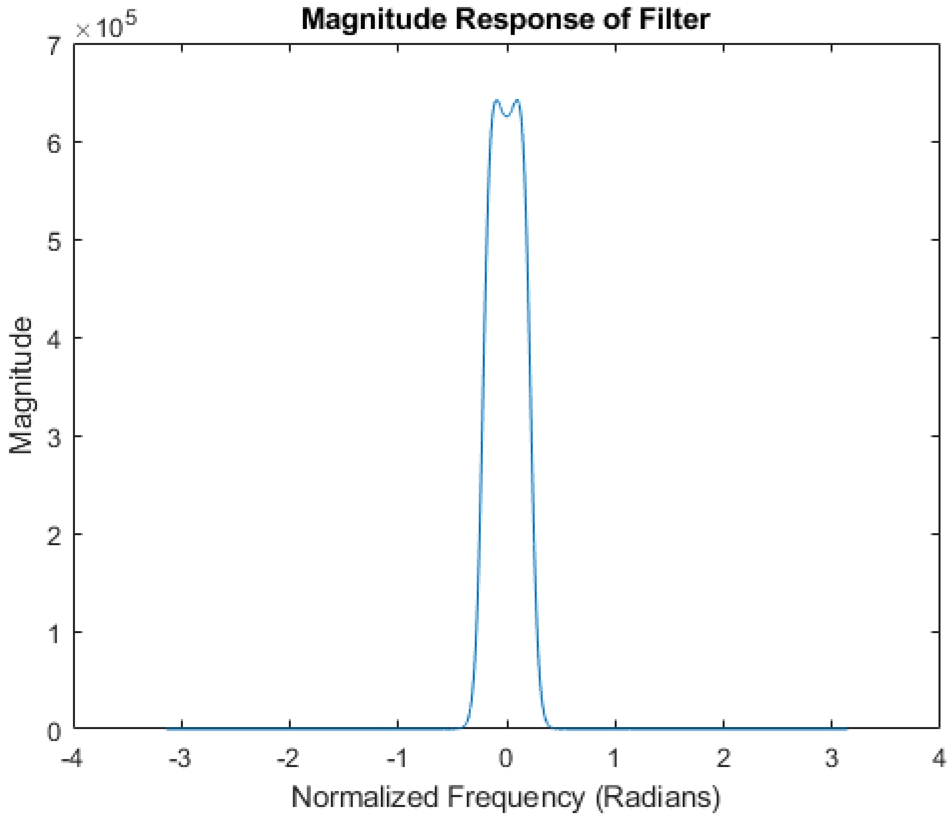
2 (f) Answer question

```
% Due to the fact I am using a IIR, the impulse response never
 converges at
% zero. This plot confirms that we are using an IIR filter.
```

2 (g) Plot freqeuency response

```
H1 = DTFT(y_i,w);

figure
plot(w,abs(H1))
title('Magnitude Response of Filter')
xlabel('Normalized Frequency (Radians)')
ylabel('Magnitude')
```

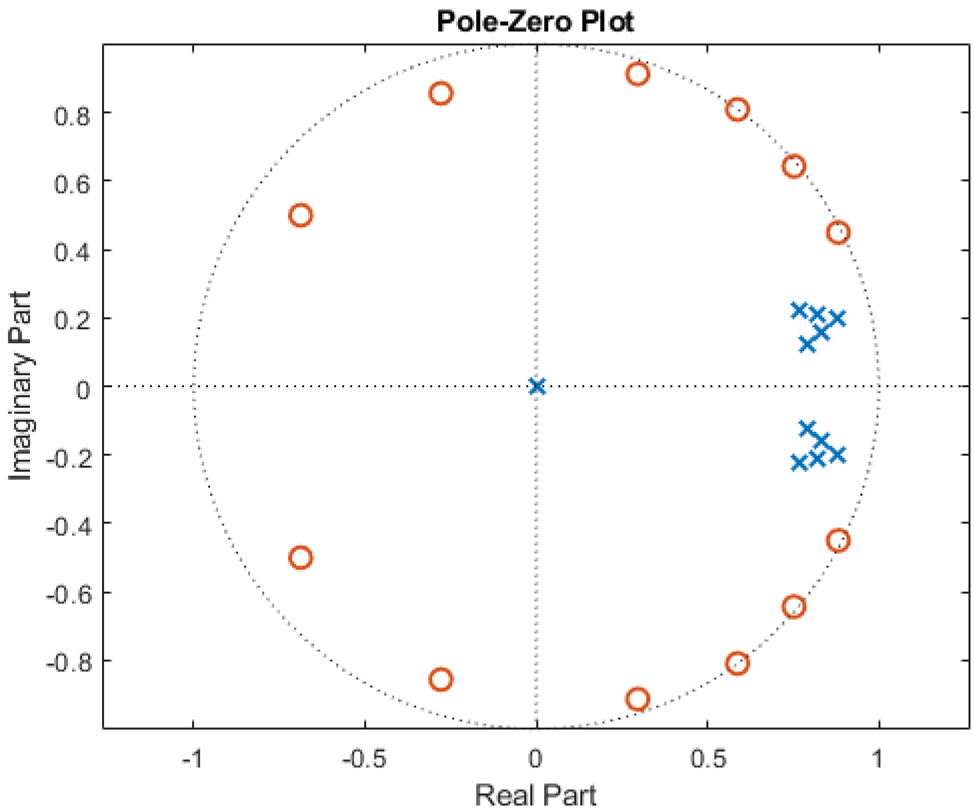


2 (h) Answer question

```
% The filtered magnitude frequency response plot shows what
 frequencies our
% filter will pass, which in our case in the lows.
```

2 (i) Plot pole-zero plot

```
% PLOT POLE-ZERO PLOT
figure
pzplot(b,a)
axis equal;
```



2 (j) Answer question

```
% The pole-zero plot shows what frequencies we are boosting (the
 poles) and what
% frequencies we are attenuating (the zeros). For our filter, we are
% trying to attentuate all except the lows.

scaled = y/max(abs(y));
audiowrite('bass.wav', scaled, fs);
```

QUESTION 3 Thinking in Three Domains 2

```
% DEFINE POLES
mypoles = [ ...
        0.65*exp(1j*0.20*pi) ...
        0.65*exp(-1j*0.20*pi) ...
        0.65*exp(1j*0.22*pi) ...
        0.65*exp(-1j*0.22*pi) ...
```

```
            0.6*exp(1j*0.23*pi) ...
            0.6*exp(-1j*0.23*pi) ...
            0.6*exp(1j*0.19*pi) ...
            0.6*exp(-1j*0.19*pi) ...
            0.7*exp(1j*0.21*pi) ...
            0.7*exp(-1j*0.21*pi) ...
             ...
             ];

% DEFINE ZEROS
myzeros = [ ...
            1 ...
            0.99*exp(1j*0.03*pi) ...
            0.99*exp(-1j*0.03*pi)...
            0.99*exp(1j*0.05*pi) ...
            0.99*exp(-1j*0.05*pi) ...
            0.99*exp(1j*0.15*pi) ...
            0.99*exp(-1j*0.15*pi) ...
            0.6*exp(1j*0.25*pi) ...
            0.6*exp(-1j*0.25*pi)...
            0.85*exp(1j*0.35*pi) ...
            0.85*exp(-1j*0.35*pi)...
            0.96*exp(1j*0.4*pi) ...
            0.96*exp(-1j*0.4*pi) ...
            0.96*exp(1j*0.5*pi) ...
            0.96*exp(-1j*0.5*pi) ...
            0.96*exp(1j*0.55*pi) ...
            0.96*exp(-1j*0.55*pi) ...
            0.90*exp(1j*0.66*pi) ...
            0.90*exp(-1j*0.66*pi) ...
            0.90*exp(1j*0.70*pi) ...
            0.90*exp(-1j*0.70*pi) ...
            0.90*exp(1j*0.4*pi*2) ...
            0.90*exp(-1j*0.4*pi*2) ...
            0.80*-1 ...
            ...
             ];

% CONVERT POLES AND ZEROS INTO COEFFICIENTS
[b,a] = pz2ba(mypoles,myzeros);

% FILTER INPUT
y = filter(b,a,x);

X = DTFT(x,w);
Y = DTFT(y,w);
```

PLAY MUSIC

```
disp('Playing Original Music ... ')
soundsc(x, fs)
pause(length(x)/fs*1.1)
```

```
disp('Playing Filtered Music ... ')
soundsc(y, fs)
```

```
Playing Original Music ...
Playing Filtered Music ...
```

3 (a) Answer question

```
% I created a band pass filter because we only want to pass the
 mandolin which
% has a frequency centered around 505 Hz,or 0.210pi normalized
 frequency.
```

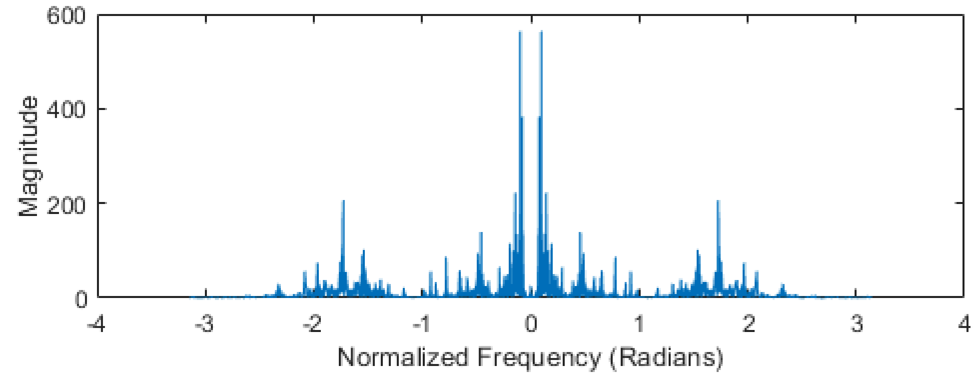
3 (b) Answer question

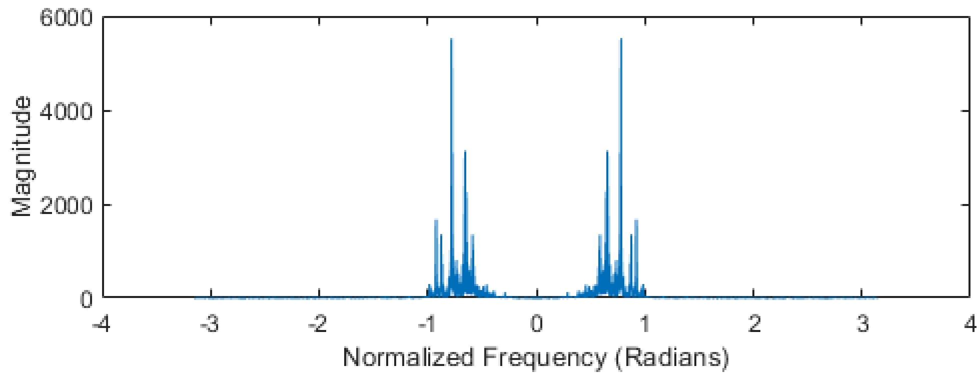
```
% I created a IIR filter because they are simpler to make, faster, and
 in
% this case would acheive a similar result.
```

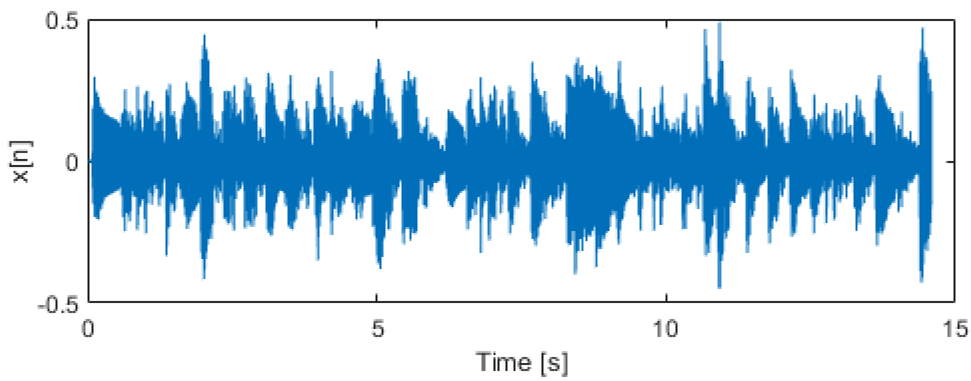
3 (c) Plot inputs and outputs

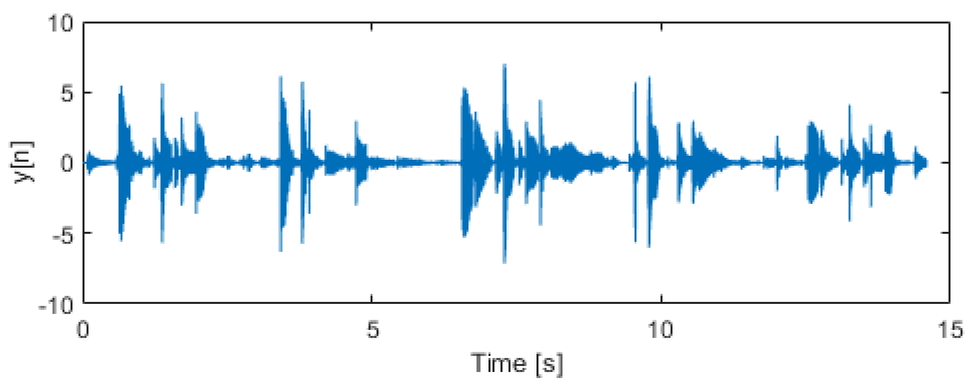
```
% PLOT FREQEUNCY DOMAIN AND
figure
subplot(211)
plot(w,abs(X))
xlabel('Normalized Frequency (Radians)')
ylabel('Magnitude')
subplot(212)
plot(w,abs(Y))
xlabel('Normalized Frequency (Radians)')
ylabel('Magnitude')

% PLOT TIME DOMAIN
figure
subplot(211)
plot(t,x)
xlabel('Time [s]')
ylabel('x[n]')
subplot(212)
plot(t,y)
xlabel('Time [s]')
ylabel('y[n]')
```


Magnitude
600
400
200
0
-4
-3
-2
-1
0
1
2
3
4
Normalized Frequency (Radians)


Magnitude
6000
4000
2000
0
-4
-3
-2
-1
0
1
2
3
4
Normalized Frequency (Radians)


x[n]
0.5
0
-0.5
0
5
10
15
Time [s]


y[n]
10
5
0
-5
-10
0
5
10
15
Time [s]

3 (d) Answer question

```
% The time domains show the output at a specific time. In the filtered
% song, the time domain shows when the mandolin is being played. The
 frequency domain
% shows us the magnituide of specific frequencies of the instruments
 in the
% song. The filtered plot shows that only frequencies roughly around
% the mandolin are being outputted.
```

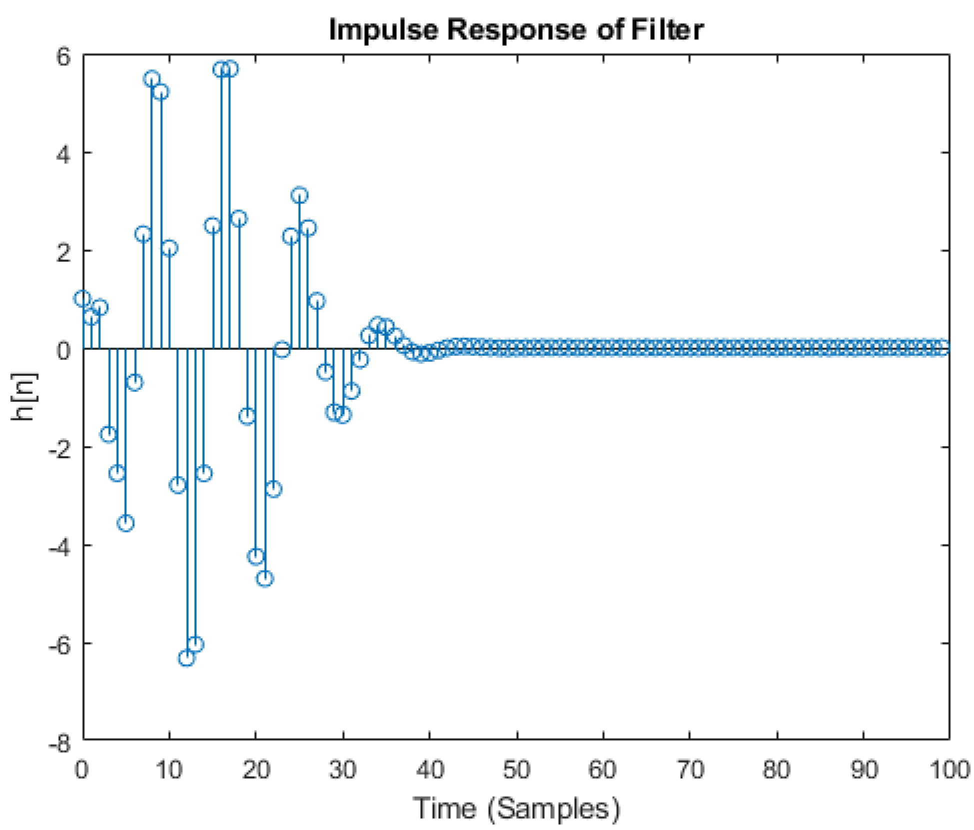
3 (e) Plot impulse response

```
N = 100;
n = 0:(N-1);

x_i = zeros(N,1);
x_i(1) = 1;

y_i = filter(b,a,x_i);

figure
stem(n,y_i)
xlabel('Time (Samples)')
ylabel('h[n]')
title('Impulse Response of Filter')
```



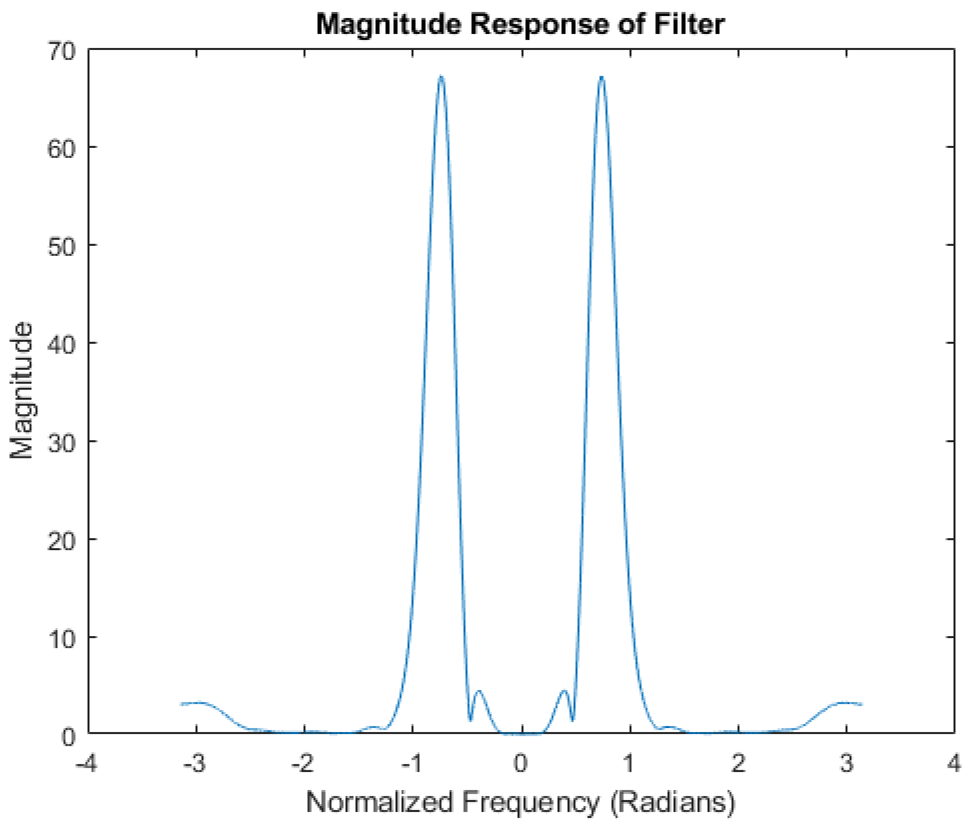
3 (f) Answer question

```
% Due to the fact I am using a IIR, the impulse response never
 converges at
% zero. This plot confirms that we are using an IIR filter.
```

3 (g) Plot freqeuency response

```
H1 = DTFT(y_i,w);

figure
plot(w,abs(H1))
title('Magnitude Response of Filter')
xlabel('Normalized Frequency (Radians)')
ylabel('Magnitude')
```

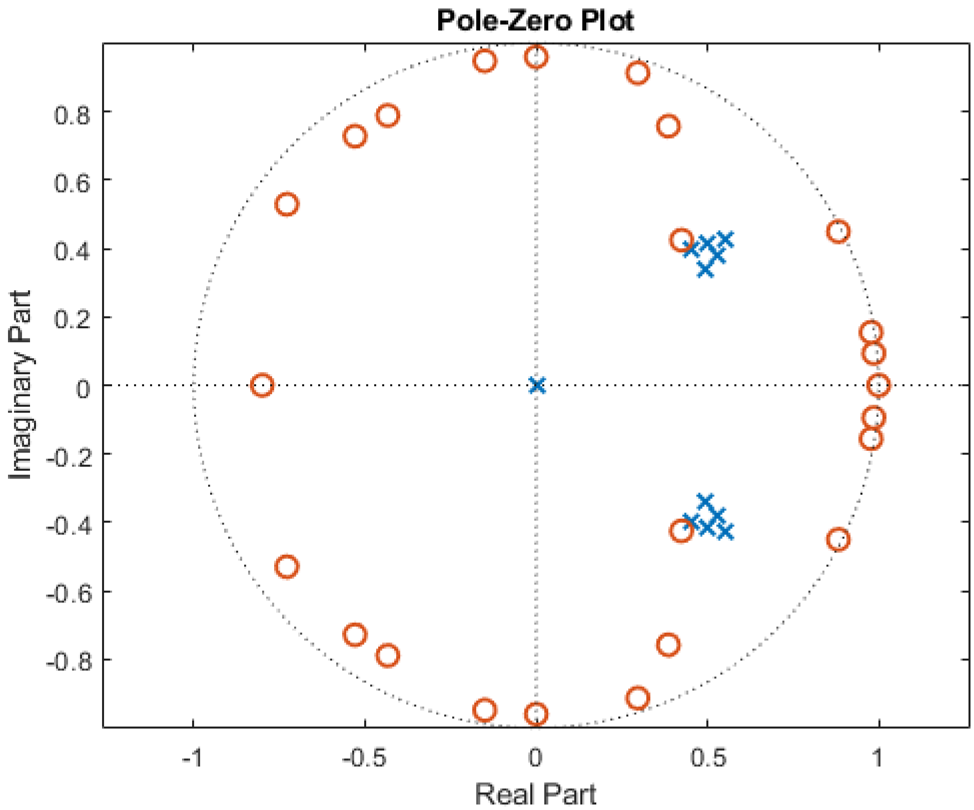


3 (h) Answer question

```
% The filtered magnitude frequency response plot shows what
 frequencies our
% filter will pass, which in our case are around the mandolin.
```

3 (i) Plot pole-zero plot

```
% PLOT POLE-ZERO PLOT
figure
pzplot(b,a)
axis equal;
```



3 (j) Answer question

```
% The pole-zero plot shows what frequencies we are boosting (the
 poles) and what
% frequencies we are attenuating (the zeros). For our filter, we are
% trying to attentuate all except the range of 0.20pi - 0.23pi
 (normalized frequency).

scaled = y/max(abs(y));
audiowrite('mandolin.wav', scaled, fs);
```

ALL FUNCTIONS SUPPORTING THIS CODE % %

```
function pzplot(b,a)
% PZPLOT(B,A)  plots the pole-zero plot for the filter described by
```

```
% vectors A and B.  The filter is a "Direct Form II Transposed"
% implementation of the standard difference equation:
%
%    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
%                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
%

    % MODIFY THE POLYNOMIALS TO FIND THE ROOTS
    b1 = zeros(max(length(a),length(b)),1); % Need to add zeros to get
 the right roots
    a1 = zeros(max(length(a),length(b)),1); % Need to add zeros to get
 the right roots
    b1(1:length(b)) = b;    % New a with all values
    a1(1:length(a)) = a;    % New a with all values

    % FIND THE ROOTS OF EACH POLYNOMIAL AND PLOT THE LOCATIONS OF THE
 ROOTS
    h1 = plot(real(roots(a1)), imag(roots(a1)));
    hold on;
    h2 = plot(real(roots(b1)), imag(roots(b1)));
    hold off;

    % DRAW THE UNIT CIRCLE
    circle(0,0,1)

    % MAKE THE POLES AND ZEROS X's AND O's

 set(h1, 'LineStyle', 'none', 'Marker', 'x', 'MarkerFaceColor','none', 'linewidth'
 1.5, 'markersize', 8);

 set(h2, 'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor','none', 'linewidth'
 1.5, 'markersize', 8);
    axis equal;

    % DRAW VERTICAL AND HORIZONTAL LINES
    xminmax = xlim();
    yminmax = ylim();
    line([xminmax(1) xminmax(2)],[0 0], 'linestyle', ':', 'linewidth',
 0.5, 'color', [1 1 1]*.1)
    line([0 0],[yminmax(1) yminmax(2)], 'linestyle', ':', 'linewidth',
 0.5, 'color', [1 1 1]*.1)

    % ADD LABELS AND TITLE
    xlabel('Real Part')
    ylabel('Imaginary Part')
    title('Pole-Zero Plot')

end


function circle(x,y,r)
% CIRCLE(X,Y,R)  draws a circle with horizontal center X, vertical
 center
% Y, and radius R.
```

```
%

    % ANGLES TO DRAW
    ang=0:0.01:2*pi;

    % DEFINE LOCATIONS OF CIRCLE
    xp=r*cos(ang);
    yp=r*sin(ang);

    % PLOT CIRCLE
    hold on;
    plot(x+xp,y+yp, ':', 'linewidth', 0.5, 'color', [1 1 1]*.1);
    hold off;

end



function H = DTFT(x,w)
% DTFT(X,W)  compute the Discrete-time Fourier Transform of signal X
% acroess frequencies defined by W.

    H = zeros(length(w),1);
    for nn = 1:length(x)
        H = H + x(nn).*exp(-1j*w.'*(nn-1));
    end

end


function xs = shift(x, s)
% SHIFT(x, s) shifts signal x by s such that the output can be defined
 by
% xs[n] = x[n - s]

    % INITIALIZE THE OUTPUT
    xs = zeros(length(x), 1);

    for n = 1:length(x)
        % CHECK IF THE SHIFT IS OUT OF BOUNDS FOR THIS SAMPLE
        if n-s > 0 && n-s < length(x)
            % SHIFT DATA
            xs(n) = x(n-s);
        end
    end

end



function [b,a] = pz2ba(p,z)
% PZ2BA(P,Z)  Converts poles P and zeros Z to filter coefficients
%             B and A
%
```

```
% Filter coefficients are defined by:
%    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
%                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
%

    % CONVERT ROOTS (POLES AND ZEROS) INTO POLYNOMIALS
    b = poly(z);
    a = poly(p);

end


function [p,z] = ba2pz(b,a)
% BA2PZ(B,A)  Converts filter coefficients B and A into poles P and
 zeros Z
%
% Filter coefficients are defined by:
%    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
%                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
%

    % MODIFY THE POLYNOMIALS TO FIND THE ROOTS
    b1 = zeros(max(length(a),length(b)),1); % Need to add zeros to get
 the right roots
    a1 = zeros(max(length(a),length(b)),1); % Need to add zeros to get
 the right roots
    b1(1:length(b)) = b;    % New a with all values
    a1(1:length(a)) = a;    % New a with all values

    % FIND THE ROOTS OF EACH POLYNOMIAL AND PLOT THE LOCATIONS OF THE
 ROOTS
    p = real(roots(a1))+1j*imag(roots(a1));
    z = real(roots(b1))+1j*imag(roots(b1));

end
```

Published with MATLAB® R2020a