Lab 3 Part 2

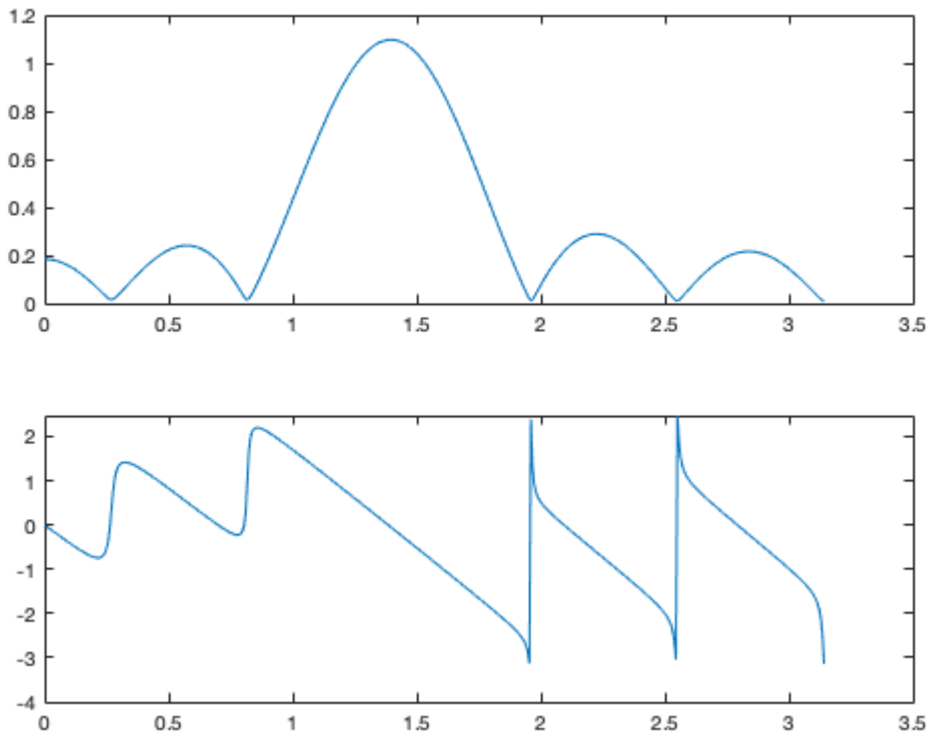
Contents

2.1 - Simple Bandpass Filter

```
clear
```

1. Generate a bandpass filter that will pass a frequency component at w = 0.44pi, with L = 10. Submit necessary code and plots of magnitude and phase response. Measure the gain of the filter (i.e., the magnitude) at the three frequencies of interest: w = 0.3pi , w = 0.44pi and w = 0.7pi.

```
% generate the impulse response, h. centered at wc
L = 10;
wc = 0.44*pi;
n = (0:1:(L-1));
h = 2/L*cos(wc*n);

% generate the frequency response from the impulse response and plot
b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);
subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle
```



```
clf
```

sample 1 = 0.3*pi

```
% find the index that 0.3pi occurs at
index1 = find(abs(ww - 0.3*pi) < 0.001);
abs(H(index1))
```

```
ans =

    0.2836
```

sample 2 = 0.44*pi

```
% find the index that 0.44pi occurs at
index2 = find(abs(ww - 0.44*pi) < 0.001);
abs(H(index2))
```

```
ans =

    1.0961
```

sample 3 = 0.7*pi

```
% find the index that 0.7pi occurs at
index3 = find(abs(ww - 0.7*pi) < 0.001);
abs(H(index3))
```

```
ans =

    0.2861
```

2.

Question 1: For the L = 10 bandpass filter from part (a), determine the passband width (for passband defined above). Repeat the plot for L = 20 and L = 40, and explain how the width of the passband is related to filter length L (i.e., what happens when L is doubled or halved).

For L = 10, the below result is the width of the passband in samples

```
% find all indexes where the magnitude of the frequency response is above
% the cutoff defined in the lab document
passband = find(abs(H) > 0.707);
length(passband)
```

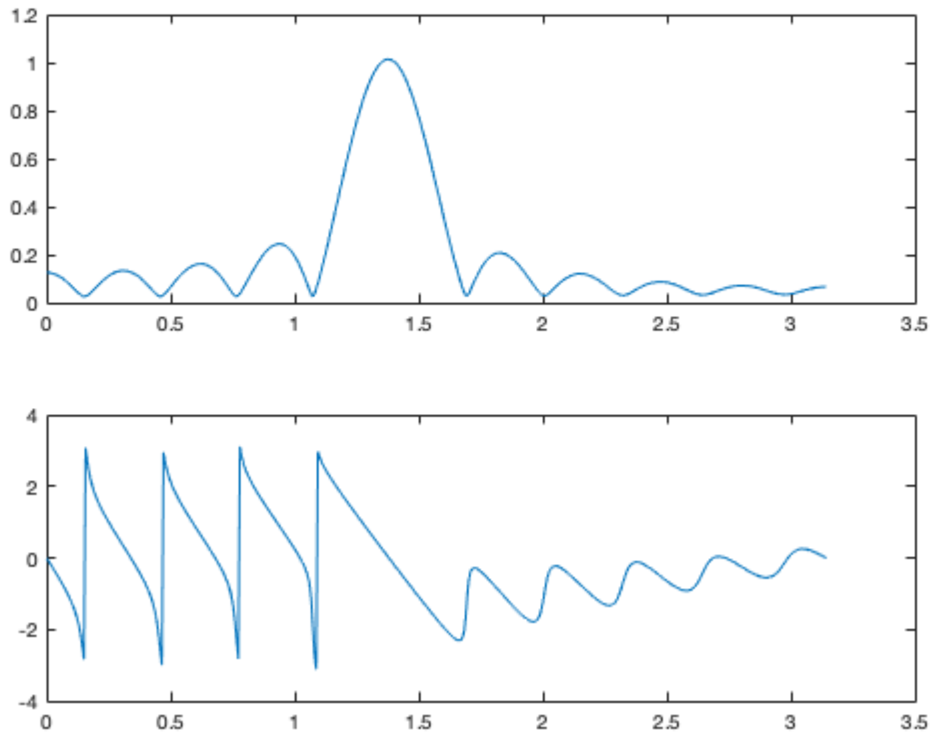
```
ans =

    91
```

For L = 20, the below result is the plot of the frequency response and the width of the passband in samples

```
clf

% generate the bandpass filter's impulse response, h
L = 20;
wc = 0.44*pi;
n = (0:1:(L-1));
h = 2/L*cos(wc*n);

% find the frequency response from the impulse response and plot it
b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);
subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle

% find all indexes where the magnitude of the frequency response is above
% the cutoff defined in the lab document
passband = find(abs(H) > 0.707);
```

```
length(passband)
```

```
ans =

    45
```

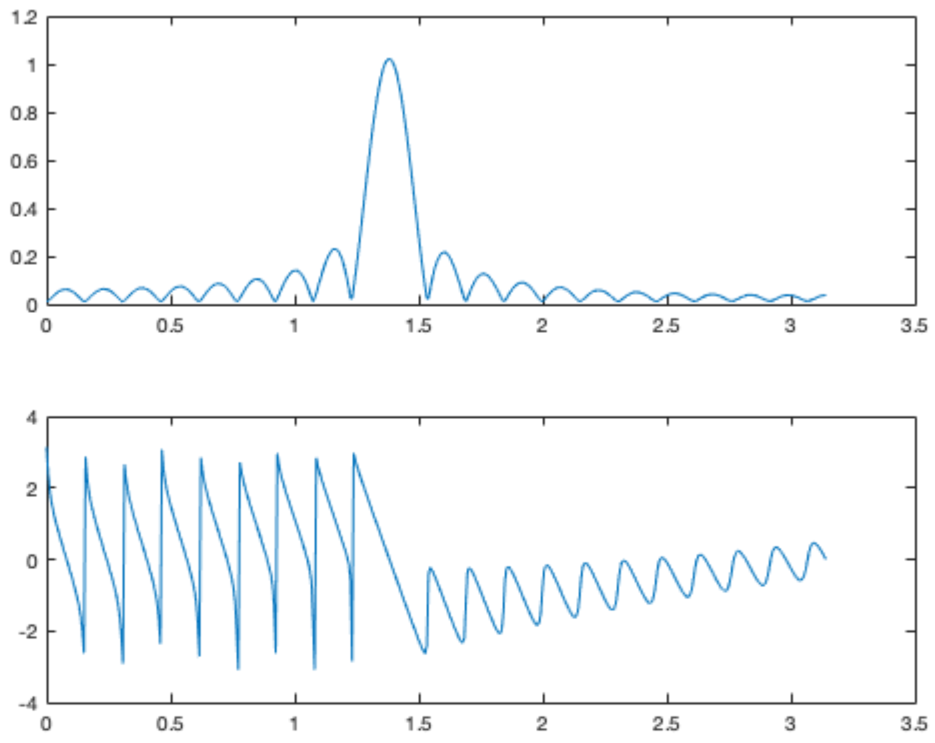


For L = 40, the below result is the plot of the frequency response and the width of the passband in samples

```
clf

% generate the bandpass filter's impulse response, h
L = 40;
n = (0:1:(L-1));
h = 2/L*cos(wc*n);

% find the frequency response from the impulse response and plot it
b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);
subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle

% find all indexes where the magnitude of the frequency response is above
% the cutoff defined in the lab document
```

```
passband = find(abs(H) > 0.707);
length(passband)
```

```
ans =

    23
```



As the length of the filter is doubled, the length of the passband is halved. Thus, they are inversely proportional.

3.

Question 2: Comment on the selectivity of the L = 10 bandpass filter. In other words, which frequencies are "passed by the filter" and which are "nulled by the filter". Use the frequency response to explain how the filter can pass one component at w = 0.44pi, while reducing or rejecting the others at w = 0.3pi and w = 0.7pi.

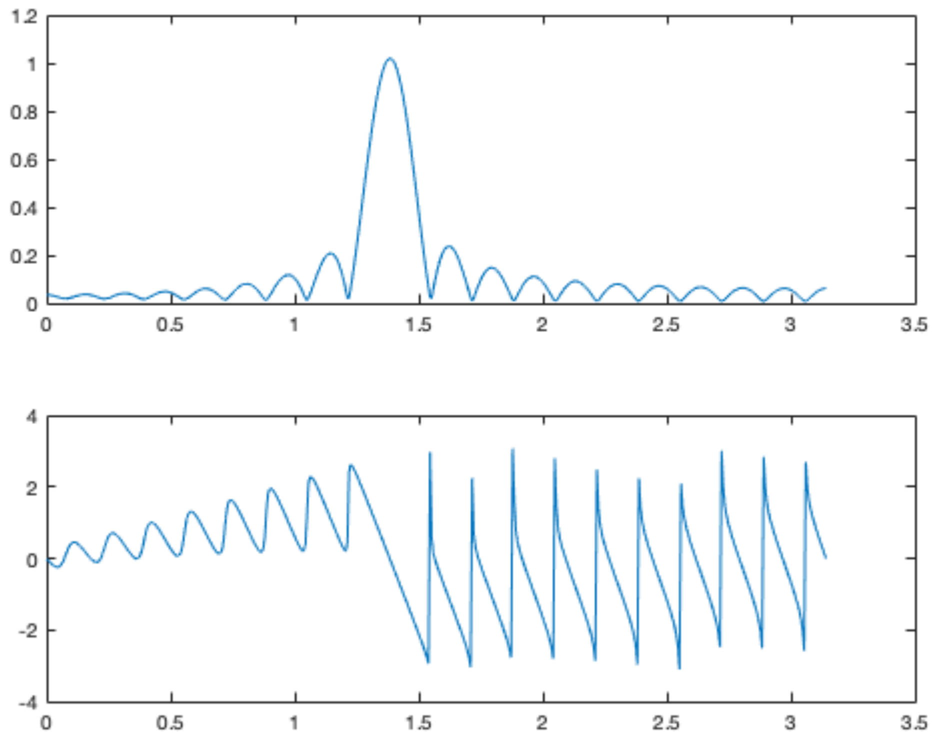
The L = 10 bandpass filter passes frequencies that are close to 0.44pi and reduces and rejects filters that are closer to 0.3pi or 0.7pi. The reason that a bandpass filter might pass a particular frequency is because the magnitude of the frequency response of said filter is closer to 1 when the frequency is closer to the frequency that it is designed for. When the frequency is further away from the frequency that the filter is designed for, the magnitude of the frequency response is closer to 0, so these frequencies are reduced. As the length of the filter increases, there is more cancellation on the frequencies that are farther from the frequency that the filter was designed for and the band of accepted frequencies shrinks.

4. Generate a bandpass filter that will pass the frequency component at w = 0.44pi, but now make the filter length (L) long enough so that it will also greatly reduce frequency components at (or near) w = 0.3pi and w = 0.7pi. Submit code and frequency response plot. Question 3: Determine the smallest value of L so that the following conditions both hold: - Any

frequency component satisfying abs(w) <= 0.3pi will be reduced by a factor of 10 or more. - Any frequency component satisfying 0.7pi <= abs(w) <= pi will be reduced by a factor of 10 or more.

```
% define the variables that do not change in the loop body
wc = 0.44*pi;
b = 1;
ww = 0:pi/500:pi;
index3 = find(ww < 0.3*pi);
index7 = find(ww > 0.7*pi);

% loop until we find a filter that matches the given criteria
for L=10:100

    % generate the impulse response and the frequency response
    n = (0:1:(L-1));
    h = 2/L*cos(wc*n);
    H = freekz(h,b,ww);

    % check to see if the magnitude of the elements in the stop band is
    % less than or equal 0.1 for every element in the passband
    mag3 = find(abs(H(index3)) > 0.1);
    mag7 = find(abs(H(index7)) > 0.1);

    % the above condition is true when both of these array are empty
    if (isempty(mag3) && isempty(mag7))
        break
    end

end

% plot the magnitude and angle response of the latest filter that we
% generated (the first filter that fulfilled the requirements specified)
subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle

% print the length L of the first filter that fulfilled the requirements
L %#ok<NOPTS>
```

```
L =

    37
```

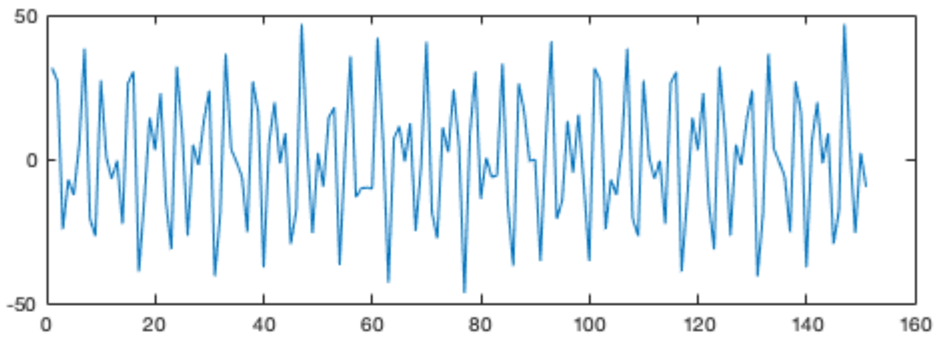


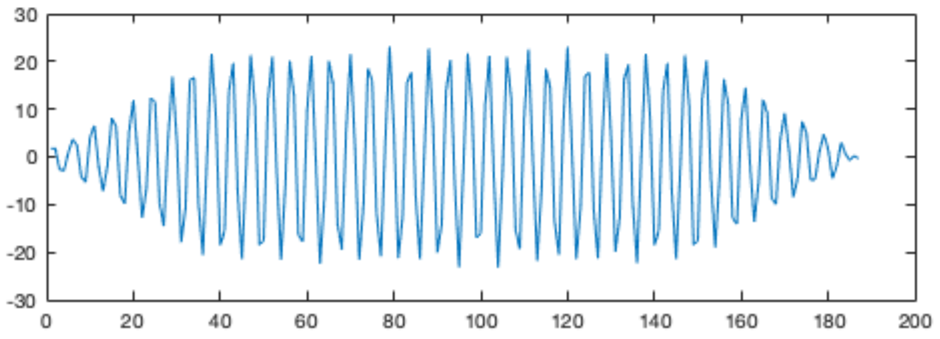
The number printed to the console above is the smallest length L that fulfills the conditions requested.

5.

Question 4: Use the filter from previous part to filter the "sum of 3 sinusoids" signal from Section 1.3.3. Make a plot of 100 points (over time) of the input and output signals, and explain how the filter has reduced or removed two of the three sinusoidal components.

```
% generate the input signal from 0 -> 150
n = 0:1:150;
xx = 5*cos(0.3*pi*n) + 22*cos(0.44*pi*n - pi/3) + 22*cos(0.7*pi*n - pi/4);

% generate the output signal by convolving the input signal with the
% impulse response
yy = conv(xx, h);

clf

% plot the input vs the output
subplot(2,1,1), plot(xx); %-- input
subplot(2,1,2), plot(yy); %-- output
```





You can see that the filter has greatly reduced the sinusoids that have frequencies of 0.3pi and 0.7pi. The sinusoid that has a frequency of 0.44pi, 22*cos(0.44*pi*n - pi/3), dominates the output. This is because the filter was designed to pass sinusoides of frequencies ~0.44pi and greatly reduce sinusoids of frequencies less than 0.3pi or greater than 0.7pi.

6.

Question 5: Using the frequency response for the filter from part 2.1.4, explain how H(e^jw) can be used to determine the relative size of each sinusoidal component in the output signal. In other words, write a sentence or two that connect a mathematical description of the output signal to the values that can be obtained from the frequency response plot.

Point 1: From observing the frequency response plot (specifically, the magnitude response plot), one can observe that for frequencies greater than 0.7pi and less than 0.3pi, they are greatly reduced (by a factor of 10 or more).

Point 2: When you pass a sinusoid through a filter, you should evaluate your frequency response function at the frequency of the sinusoid that is passing through your filter. Then, take the magnitude of this result and multiply it to the amplitude of that particular sinusiod.

What we find when we combine points 1 and 2 is the result that we are multiplying numbers < 0.1 to any frequency greater than 0.7pi or less than 0.3pi and multiplying numbers that are ~1 to frequencies that are ~0.44pi. This means that the output signal will be dominated by those sinusiods with frequencies ~0.44pi.

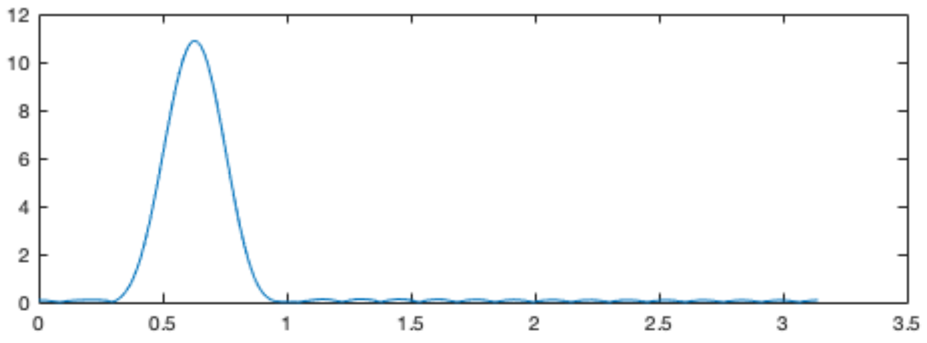
2.2 - A Better BPF

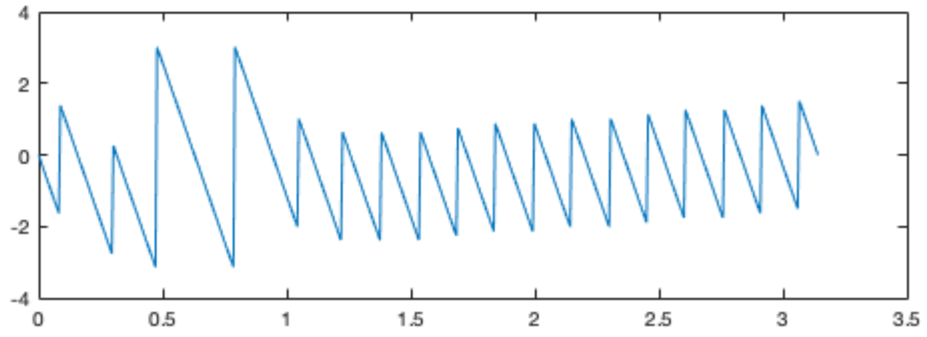
1.

Generate a Hamming bandpass filter that will pass a frequency component at w = 0.2pi. Make the filter length L=41. Make a plot of the frequency response magnitude and phase. Measure the response of the filter (magnitude and phase) at the following frequencies of interest: w = 0.1pi, 0.25pi, 0.4pi, 0.5pi, 0.75pi. Summarize the values in a table.

```
% generate the hamming bandpass filter
L = 41;
wc = 0.2*pi;
n = 0:1:(L-1);
h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));

clf

% plot the magnitude and phase response of the frequency response of the
% filter
b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);
subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle

% generate the table and fill out the first column (the frequencies of
% interest)
r = 5;
c = 3;
wtable = zeros(r,c);
wtable(1,1) = 0.1*pi;
wtable(2,1) = 0.25*pi;
wtable(3,1) = 0.4*pi;
wtable(4,1) = 0.5*pi;
wtable(5,1) = 0.75*pi;

% fill out the other two columns, row by row, of the table by using the
% first column
for i=1:r
    w = wtable(i,1);
    Hindex = find(abs(ww - w) < 0.001);
    wtable(i,2) = abs(H(Hindex));
    wtable(i,3) = angle(H(Hindex));
end

% print the table to the console
table = array2table(wtable,'VariableNames',{'w','Magnitude','Phase'});
table %#ok<NOPTS>
```

```
table =

  5×3 table

       w       Magnitude       Phase    
    _______    _________    ___________

    0.31416      0.08       -7.7954e-15
     0.7854      4.52           -3.1416
     1.2566      0.08       -3.1759e-15
     1.5708      0.08        2.7756e-15
     2.3562      0.08          -1.84e-16
```





2.

Use the plot of the frequency response for the length-41 bandpass filter from the previous section, and determine the passband width using the 50% level to define the pass band.

```
% find the max value and assign it to pm, then find all indexes that are
% greater than pm / 2. this is your passband.
[pm, ~] = max(abs(H));
passband = find(abs(H) > pm/2);
length(passband)
```

```
ans =

    45
```

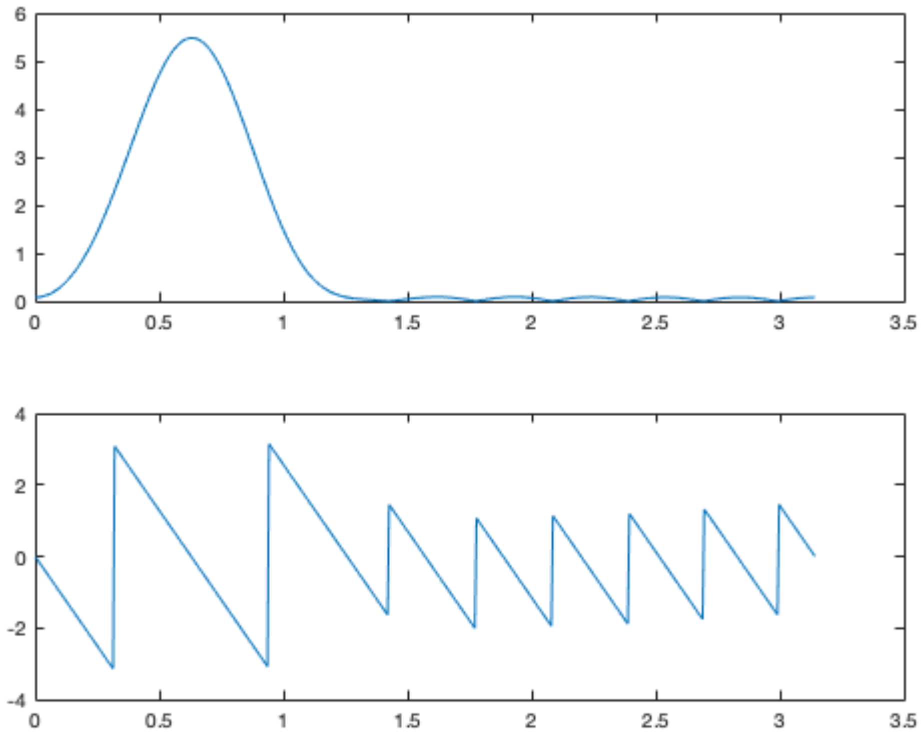
Make two other plots of BPFs for L = 21 and L = 81, and measure the passband width in both.

L = 21

```
clf

L = 21;
wc = 0.2*pi;
```

```
n = 0:1:(L-1);
h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));

b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);

subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle

[pm, ~] = max(abs(H));
passband = find(abs(H) > pm/2);
length(passband)
```

```
ans =

    88
```



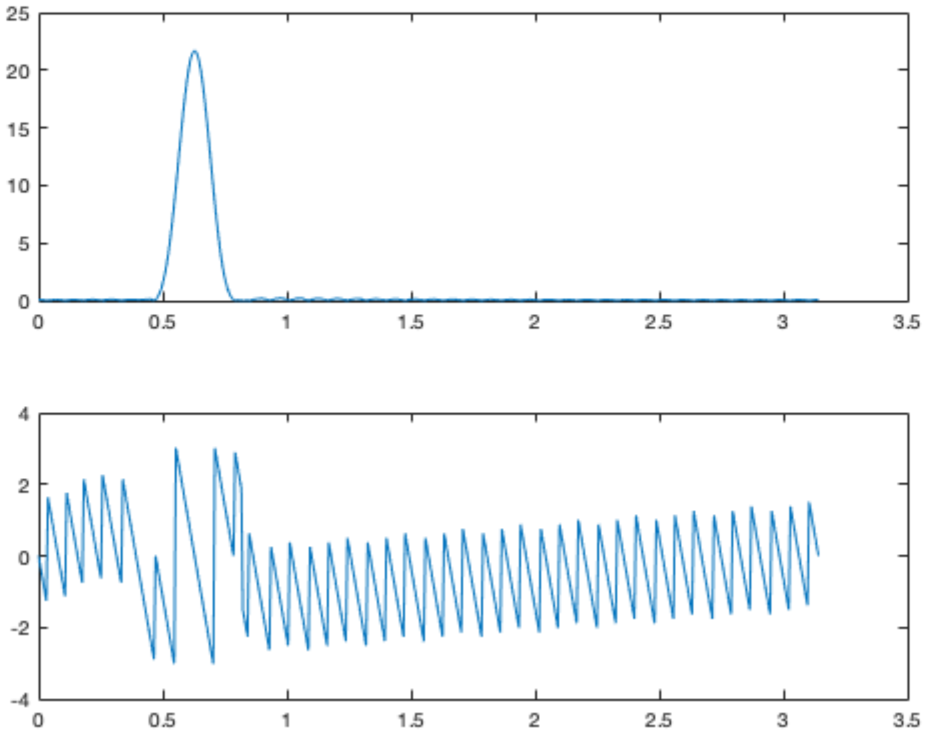
The passband width of the filter with length 21 is seen above.

L = 81

```
clf
```

```
L = 81;
wc = 0.2*pi;
n = 0:1:(L-1);
h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));

b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);

subplot(2,1,1), plot(ww,abs(H)); %-- magnitude
subplot(2,1,2), plot(ww,angle(H)); %-- angle

[pm, pmi] = max(abs(H));
passband = find(abs(H) > pm/2);
length(passband)
```

```
ans =

    23
```



The passband width of the filter with length 81 is seen above.

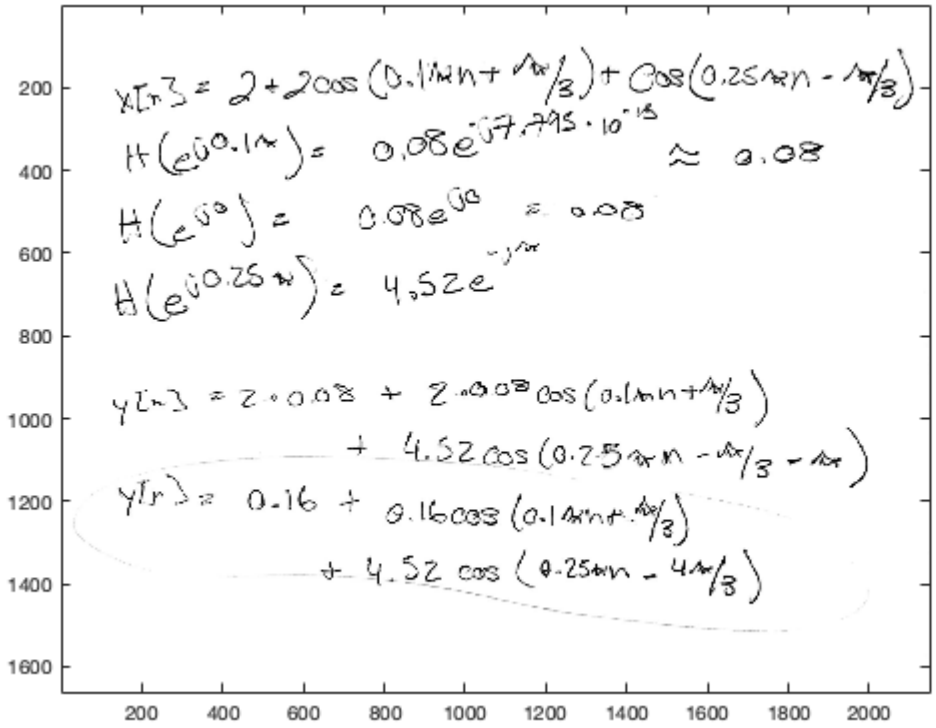
Question 6: Explain how the width of the passband is related to filter length L, i.e., what happens when L is (approximately) doubled or halved.

As was the case with the simple bandpass filter, when you double the length of the filter the passband width is approximately halved.

3.

Question 7: Determine (by hand) the formula for the output signal.

```
clf

img = imread("2.2.3.jpg");
image(img);
```



$$x[n] = 2 + 2\cos(0.1\pi n + \pi/3) + \cos(0.25\pi n - \pi/3)$$

$$H(e^{j0.1\pi}) = 0.08e^{j7.795\cdot 10^{-15}} \approx 0.08$$

$$H(e^{j0}) = 0.08e^{j0} = 0.08$$

$$H(e^{j0.25\pi}) = 4.52e^{-j\pi}$$

$$y[n] = 2\cdot 0.08 + 2\cdot 0.08\cos(0.1\pi n + \pi/3) + 4.52\cos(0.25\pi n - \pi/3 - \pi)$$

$$y[n] = 0.16 + 0.16\cos(0.1\pi n + \pi/3) + 4.52\cos(0.25\pi n - 4\pi/3)$$

Comment on the relative amplitudes of the three signal components in the output signal, observing whether or not they were in the passband or topband of the filter.

The amplitudes of the DC component and the sinusoidal component with frequency 0.1pi were greatly reduced (by a factor of 0.08) while the amplitude of the sinusoidal component with frequency 0.25pi was greatly amplified. This is because the first two components were in the stopband while the third component was in the passband.

4.

Use the frequency response (and passband width) of the length-41 bandpass filter to explain how the filter is able to pass the components at w = +-0.25pi, while reducing or rejecting others.

When observing the plot of the magnitude response - you can see that frequencies that are within a narrow range near 0.2pi are amplified. Since 0.25 is very close to the center frequency of this passband filter, it is withing the passband of the filter and is amplified. The other two components of the previous question were outside of the passband and were thus rejected.

2.3 - Octave Filtering

Question 8: Below, we provide you with the lower edge, higher edge, and center frequencies of all octave (in key number and Hz). Based on this, calculate the same parameters in terms of normalized radian frequency, and submit a table consists of those values.

```
clf

% generate the base table that was given in the lab document

r = 5;
c = 5;
octave_table = zeros(r,c);

octave_table(1,1) = 16;
octave_table(1,2) = 28;
octave_table(1,3) = 40;
octave_table(1,4) = 52;
octave_table(1,5) = 64;

octave_table(2,1) = 65.4;
octave_table(2,2) = 130.8;
octave_table(2,3) = 261.6;
octave_table(2,4) = 523.25;
octave_table(2,5) = 1046.5;

octave_table(3,1) = 27;
octave_table(3,2) = 39;
octave_table(3,3) = 51;
octave_table(3,4) = 63;
octave_table(3,5) = 75;

octave_table(4,1) = 123.5;
octave_table(4,2) = 246.9;
octave_table(4,3) = 493.9;
octave_table(4,4) = 987.8;
octave_table(4,5) = 1978.5;

octave_table(5,1) = 94.4;
octave_table(5,2) = 188.85;
octave_table(5,3) = 379.24;
octave_table(5,4) = 755.51;
octave_table(5,5) = 1551;

% print this table to the console

array2table(octave_table, ...
            'VariableNames', ...
                {'Two', ...
                 'Three', ...
                 'Four', ...
                 'Five', ...
```

```
                  'Six'}, ...
             'RowNames', ...
                 {'Lower Edge (key #)', ...
                  'Lower Edge (Hz)', ...
                  'Higher Edge (key #)', ...
                  'Higher Edge (Hz)', ...
                  'Center (Hz)'})

radian_octave_table = octave_table;
fs = 8000;

% generate the radian octave table using the octave table
for i = 1:r
    if i == 1 || i == 3
        continue
    end
    for j = 1:c
        radian_octave_table(i,j) = 2 * pi * octave_table(i,j) / fs;
    end
end

% print it to the console

array2table(radian_octave_table, ...
             'VariableNames', ...
                 {'Two', ...
                  'Three', ...
                  'Four', ...
                  'Five', ...
                  'Six'}, ...
             'RowNames', ...
                 {'Lower Edge (key #)', ...
                  'Lower Edge (Radians)', ...
                  'Higher Edge (key #)', ...
                  'Higher Edge (Radians)', ...
                  'Center (Radians)'})
```

```
ans =

  5×5 table

                            Two      Three      Four      Five       Six
                           _____    ______    ______    ______    ______

    Lower Edge (key #)        16        28        40        52        64
    Lower Edge (Hz)         65.4     130.8     261.6    523.25    1046.5
    Higher Edge (key #)       27        39        51        63        75
    Higher Edge (Hz)       123.5     246.9     493.9     987.8    1978.5
    Center (Hz)             94.4    188.85    379.24    755.51      1551


ans =
  5×5 table

                                Two          Three        Four         Five          Six
                              ________     _______      _______      _______      _______

    Lower Edge (key #)              16          28           40           52           64
    Lower Edge (Radians)      0.051365     0.10273      0.20546      0.41096      0.82192
    Higher Edge (key #)             27          39           51           63           75
    Higher Edge (Radians)     0.096997     0.19391      0.38791      0.77582       1.5539
    Center (Radians)          0.074142     0.14832      0.29785      0.59338       1.2182
```

2.4 - Bandpass Filter Bank Design

1.

```
L = 21;
wc = 0.2*pi;
n = 0:1:(L-1);
h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));

b = 1;
ww = 0:pi/500:pi;
H = freekz(h,b,ww);

[pm, ~] = max(abs(H));
B = 1/pm;
```

```
h = h * B;
H = freekz(h,b,ww);

max(abs(H))
```

```
ans =

     1
```

We can create scale the bandpass filter with a coefficient B. This B is the inverse of the largest value that is found in the magnitude of the frequency response. When we scale the impulse response by this value, we get a filter that has a maximum magnitude of 1.

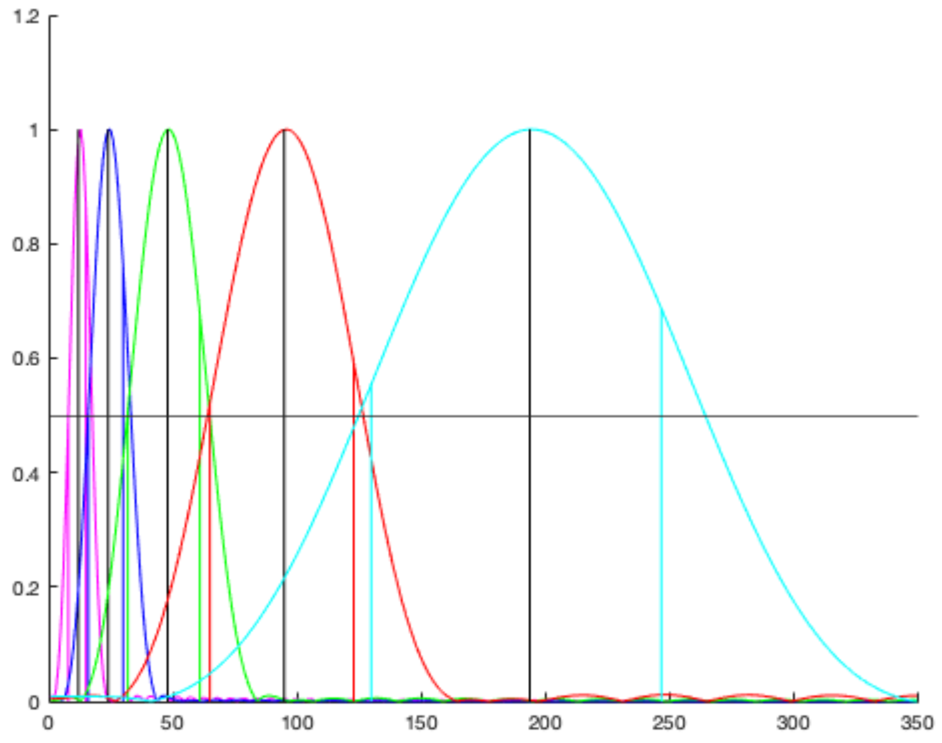
2.

```
b = 1;
step = pi/500;
ww = 0:step:pi;

filter_table = cell(c,1);

for ci = 1:c

    wc = radian_octave_table(5,ci);
    lower_edge = radian_octave_table(2,ci);
    index_low = floor(lower_edge/(step));
    higher_edge = radian_octave_table(4,ci);
    index_high = floor(higher_edge/(step));

    % loop until we find a filter that matches the given criteria
    for L=200:-1:10

        % generate the impulse response and the frequency response
        n = (0:1:(L-1));
        h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));
        H = freekz(h,b,ww);

        % scale the filter back to a maximum value of 1
        [pm, ~] = max(abs(H));
        B = 1/pm;
        h = h * B;
        H = freekz(h,b,ww);

        % check to see if we have encountered a pass that just barely
        % passes the frequencies that we want to pass
        if (abs(H(index_low)) >= 0.5 && abs(H(index_high)) >= 0.5)
            break
        end
    end
    filter_table{ci} = h;
end
```

3.

```
column_colors = ["m", "b", "g", "r", "c", "k", "y"];

hold
yline(0.5);
for ci=1:c
    color = column_colors(ci);

    h = filter_table{ci};
    H = freekz(h,b,ww);
    mag = abs(H);

    center = radian_octave_table(5,ci);
    center_index = ceil(center/(step));

    lower_edge = radian_octave_table(2,ci);
    index_low = floor(lower_edge/(step));
    higher_edge = radian_octave_table(4,ci);
    index_high = floor(higher_edge/(step));

    plot(mag(1:350), color);
    plot([center_index center_index],[0 1], "k");
    plot([index_low index_low],[0 mag(index_low)], color);
    plot([index_high index_high],[0 mag(index_high)], color);
end
hold
```

```
Current plot held
Current plot released
```



Question 9: Comment on the selectivity of the bandpass filters, i.e., use the frequency response to explain how the filter passes one octave while rejecting the others. Are the filter?s passbands narrow enough so that only one octave lies in the passband and the others are in the stopband?

Observe the previous plots. You can see that there are 5 bandpass filters, one for each of the octaves. Note that they are all scaled so that they have a maximum value of 1. This maximum value of 1 occurs at the center frequency of each filter, and is denoted by a black line for each filter. In addition to these centerlines, there are also two colored vertical lines for each filter, representing the lower and the higher edges of the octave. Notice that these lines intersect with their respective octave before the horizontal black line at y = 0.5. This indicates that the lower and upper edges of the octave are indeed contained within the pass band and NOT the stopband.

Each filter passes one octave as they are each centered on the center frequency and contain both the upper and lower edges of the octave in the pass band (above 0.5).

To determine if there is overlap in the passbands, one needs to find a colored vertical line that, when extended, would intersect with a different colored magnitude plot that is ABOVE the pass band (the horizontal black line). These filters overlap slightly, even thouh they met the specifications requested. Unfortunately, due to the proximity of the octaves, it is not possible to design Hamming bandpass filters that do not overlap in these conditions.

2.5 - Equalizer

1.

```
[xx, fs] = audioread("x-file.wav");
```

2. Use fs obtained in previous section to re-calculate the normalized center frequencies. Submit a table similar to section 2.2 with this sampling frequency.

```
radian_octave_table = octave_table;

% generate the radian octave table using the octave table and fs
for i = 1:r
    if i == 1 || i == 3
        continue
    end
    for j = 1:c
        radian_octave_table(i,j) = 2 * pi * octave_table(i,j) / fs;
    end
end

array2table(radian_octave_table, ...
            'VariableNames', ...
                {'Two', ...
                 'Three', ...
                 'Four', ...
                 'Five', ...
                 'Six'}, ...
            'RowNames', ...
                {'Lower Edge (key #)', ...
                 'Lower Edge (Radians)', ...
                 'Higher Edge (key #)', ...
                 'Higher Edge (Radians)', ...
                 'Center (Radians)'})
```

```
ans =

  5×5 table

                                Two         Three        Four        Five        Six
                              ________    ________    ________    _______    _______

    Lower Edge (key #)              16          28          40         52         64
    Lower Edge (Radians)      0.018636    0.037272    0.074543     0.1491     0.2982
    Higher Edge (key #)             27          39          51         63         75
    Higher Edge (Radians)     0.035192    0.070355     0.14074    0.28148    0.56378
    Center (Radians)          0.026899    0.053813     0.10807    0.21528    0.44196
```

With the same code you used in previous section to generate the five octave filters, modify your filters to satisfies these filter lengths (use a loop):

```
b = 1;
step = pi/500;
ww = 0:step:pi;

filter_table = cell(c,1);
l_table = [256, 128, 64, 32, 16];

for ci = 1:c
```

```
    wc = radian_octave_table(5,ci);
    lower_edge = radian_octave_table(2,ci);
    index_low = floor(lower_edge/(step));
    higher_edge = radian_octave_table(4,ci);
    index_high = floor(higher_edge/(step));

    L = l_table(ci);

    % generate the impulse response and the frequency response
    n = (0:1:(L-1));
    h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));
    H = freekz(h,b,ww);

    % scale the filter back to a maximum value of 1
    [pm, ~] = max(abs(H));
    B = 1/pm;
    h = h * B;
    H = freekz(h,b,ww);

    filter_table{ci} = h;
end
```

3.

Since different filter lengths would results in different output length, matrices of different sizes cannot be summed. Hence, modify your script so that all five filters have the same length as the longest filter. Put the relevant filter coefficients exactly in the middle of the filter and pad empty indexes on both sides with zeros.

```
clf

max_length = 256;
for ci=2:c
    filter = filter_table{ci};
    offset = floor((max_length - length(filter))/2);
    padded_filter = zeros(1,max_length);
    padded_filter(offset:(offset+length(filter)-1)) = filter;
    filter_table{ci} = padded_filter;
end
```

4.

```
type octaveEq
```

```
function [y] = octaveEQ(xx , eq , fs)
%OCTAVEEQ an equalizer function with modified gain for notes in each octave

    l_table = [256, 128, 64, 32, 16];
    radian_octave_table = 2 * pi * [94.4, 188.85, 379.24, 755.51, 1551] / fs;

    c = length(radian_octave_table);
    ww = 0:pi/500:pi;

    filter_table = cell(c,1);

    % generate the five filters and store in the filter table
    for ci = 1:c

        % get the properties of the cith filter table
        wc = radian_octave_table(ci);
        L = l_table(ci);

        % generate the impulse response and the frequency response
        n = (0:1:(L-1));
        h = (0.54 - 0.46*cos(2*pi*n/(L-1))).*cos(wc*(n-(L-1)/2));
        H = freekz(h,1,ww);
```

```
            % scale the filter proportional to eq
            [pm, ~] = max(abs(H));
            B = 1/pm;
            h = h * B * 10^(eq(ci)/20);

            filter_table{ci} = h;
        end

        % pad each filter to the maximum filter size and update in the filter
        % table
        max_length = max(l_table);
        for ci=2:c
            filter = filter_table{ci};
            offset = floor((max_length - length(filter))/2);
            padded_filter = zeros(1,max_length);
            padded_filter(offset:(offset+length(filter)-1)) = filter;
            filter_table{ci} = padded_filter;
        end

        % sum each filter into h_net
        h_net = zeros(1,max_length);
        for ci=1:c
            h_net = h_net + filter_table{ci};
        end

        % convolve to get the output
        y = conv(xx,h_net);
    end
```

5. Turn your script into a function that takes in the input vector xx , scale vector eq , sampling frequency fs and return output vector y.

```
eq = [20, 50, 70, 0, 0];
y = octaveEQ(xx, eq, fs);

soundsc(y,fs);
audiowrite("y.wav", y/max(y), fs)
```

```
Warning: Data clipped when writing file.
```

Published with MATLAB® R2019a