Lab 6: Introduction to Frequency Response

1.1 Frequency Response of the Four-Point Averager

Before beginning your work on this section, make sure that you understand the concept of Frequency Response. Information on the topic can be found in the course lecture videos and in section "A.1 Frequency Response in Matlab" found in the appendix of this document.

(a) Using the Euler formulas, show by hand that the frequency response for the 4-point running average operator is given by

$$H(e^{j\hat{\omega}}) = \frac{2\cos(0.5\,\hat{\omega}) + 2\cos(1.5\,\hat{\omega})}{4} e^{-j1.5\hat{\omega}}$$

(b) Implement this equation directly in MATLAB and plot the magnitude and phase response of this filter. For $\hat{\omega}$, use a vector that includes 400 samples between $-\pi$ and π. Since the frequency response is a complex-valued quantity, use `abs()` and `angle()` to extract the magnitude and phase of the frequency response for plotting.

(c) The function `freqz()` (as described in Section A.1) evaluates the frequency response for all frequencies in the vector `ww` by using the *summation formula* instead of the formula from part (a). Use `freqz()` in MATLAB to plot the magnitude and phase of $H(e^{j\hat{\omega}})$ versus $\hat{\omega}$. The plots from this section should look the same as the ones from part b.

Hint: The filter coefficient vector for the 4-point averager can be defined by

```
bk = 1/4*ones(1,4);
```

1.2 The MATLAB `find()` function

As part of signal processing functions, we often need to perform logical tests, such as with `if` statements. MATLAB's `find()` function allows logical tests to be done in a vectorizable form. It returns a vector of indices of all the vector elements for which the logical test is true. For example, in the following code, `find()` returns the indices of all the numbers that "round" to 3:

```
xx = 1.4:0.33:5;
jkl = find(round(xx)==3);
xx(jkl)
```

So, the `xx(jkl)` command returns the elements of the `xx` vector with indices found by the `find()`. Note that the argument of the `find()` function can be any logical expression. `find()` returns a list of indices where such logical condition is true.

Suppose that you have a frequency response:

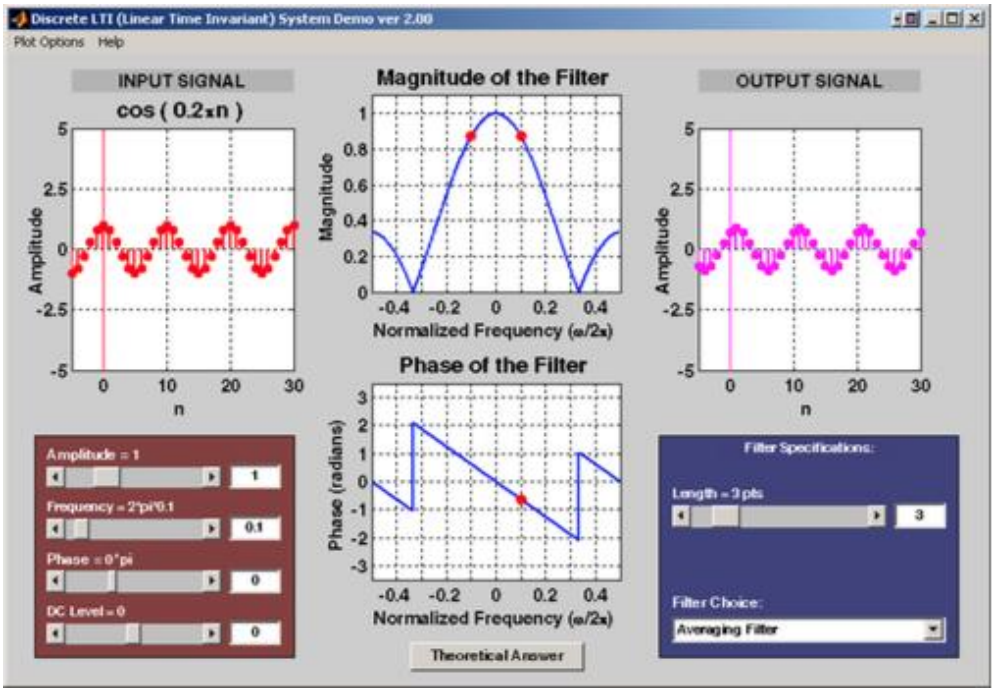
```
ww = -pi:(pi/500):pi;
HH = freqz(1/4*ones(1,4), 1, ww);
```

Using find, display the list of frequencies where `HH` is approximately zero. Note that since there might be round-off error in calculating `HH`, the logical test should probably be a test for those indices where the magnitude (absolute value in MATLAB) of `HH` is less than some small number, e.g., 1×10^{-6}. Does this match the frequency response that you plotted for the 4-point average?

1.3 LTI Frequency Response Demo

To illustrate the effects of filters on signals, use the SPFirst GUI accessed by the command:

```
dltidemo
```



The `dltidemo` shown above illustrates the "sinusoid-IN gives sinusoid-OUT" property of discrete-time LTI systems. In this demo, you can change the amplitude, phase and frequency of an input sinusoid, $x[n]$, and you can also change the digital filter that processes the signal. Then, the GUI will show the output signal, $y[n]$.

(a) Set the input to $x[n] = 1.5\cos(0.1\pi(n-4))$, set the digital filter to be a 9-point averager and determine the formula for the output signal, written in the form

$$y[n] = A\cos(\hat{\omega}_0(n-n_d))$$

(b) Using n_d for $y[n]$ and the fact that the input signal had a peak at $n = 4$, determine the amount of delay through the filter. In other words, how much has the peak of the cosine wave shifted?

(c) Determine the length of the averaging filter needed to make the output zero, i.e., $y[n] = 0$. Using the GUI, show that you have the correct filter to zero the output.
Hint: If the length is more than 15, you will have to enter the "Filter Specifications".

(d) When the output is zero, the filter acts as a *Nulling Filter*, because it eliminates the input at $\hat{\omega} = 0.1\pi$. Which other frequencies $\hat{\omega}$ are also nulled?

1.4 Nulling Filter:

This lab introduces two practical filters: *bandpass filters* and *nulling filters*. Bandpass filters can be used to detect and extract information from sinusoidal signals, e.g., tones in a touch-tone telephone dialer. Nulling filters can be used to remove sinusoidal interference, e.g., jamming signals in a radar. You should learn how to characterize a filter by knowing how it reacts to different frequency components in the input.

In this country, electrical power lines operate at $120Hz$. Most of it can be rectified. But in many cases, we wish to null this frequency. Nulling filters completely eliminate some frequency component. The simplest possible general nulling filter can have as few as three coefficients. If $\hat{\omega}_n$ is the desired nulling frequency, then the following length-3 FIR filter

$$y[n] = x[n] - 2\cos(\hat{\omega}n)\,x[n-1] + x[n-2]$$

will have a zero in its frequency response at $\hat{\omega} = \hat{\omega}_n$. For example, a filter designed to completely eliminate signals of the form $Ae^{j0.5\pi n}$ would have the following coefficients because we would pick the desired nulling frequency to be $\hat{\omega}_n = 0.5\pi$. We would pick our coefficients to be:

$$b0 = 1, b1 = -2\cos(0.5\pi) = 0, b2 = 1$$

(a) Design a filtering system that consists of the **cascade** of two FIR nulling filters that will eliminate the following input frequencies: $\hat{\omega}_n = 0.44\pi$, and $\hat{\omega}_n = 0.7\pi$. For this part, derive the filter coefficients of both nulling filters. Submit necessary code, and plots of the magnitude and phase responses of both filters, along with the cascaded system.

(b) Generate an input signal $x[n]$ that is the sum of three sinusoids:

$$x[n] = 5\cos(0.3\pi n) + 22\cos\left(0.44\pi n - \frac{\pi}{3}\right) + 22\cos(0.7\pi n - \frac{\pi}{4})$$

Make the input signal 150 samples long over the range $0 \le n \le 149$. Submit your code to generate this signal.

(c) Using `firfilt()` or `conv()`, filter the sum-of-three-sinusoids signal $x[n]$ through the filters designed in part (a). Submit the MATLAB code that you wrote to implement the cascade of two FIR filters and the command to filter the signal.

(d) Make a plot of the **first 50 points** of the output signal. Determine (by hand) the exact mathematical formula (magnitude, phase and frequency) for the output signal for $n \geq 5$.

(e) Plot the mathematical formula determined in (d) with MATLAB and compare it to the plot obtained in (d) to show that it matches the filter output from `firfilt()` over the range $5 \leq n \leq 50$.

Explain why the output signal is different for the first few points. How many "start-up" points are found? How is this number related to the lengths of the filters designed in part (a)?

Hint: consider the length of a single FIR filter that is equivalent to the cascade of two length-3 FIRs.

Lab Part Two

2.1 Simple Bandpass Filter:

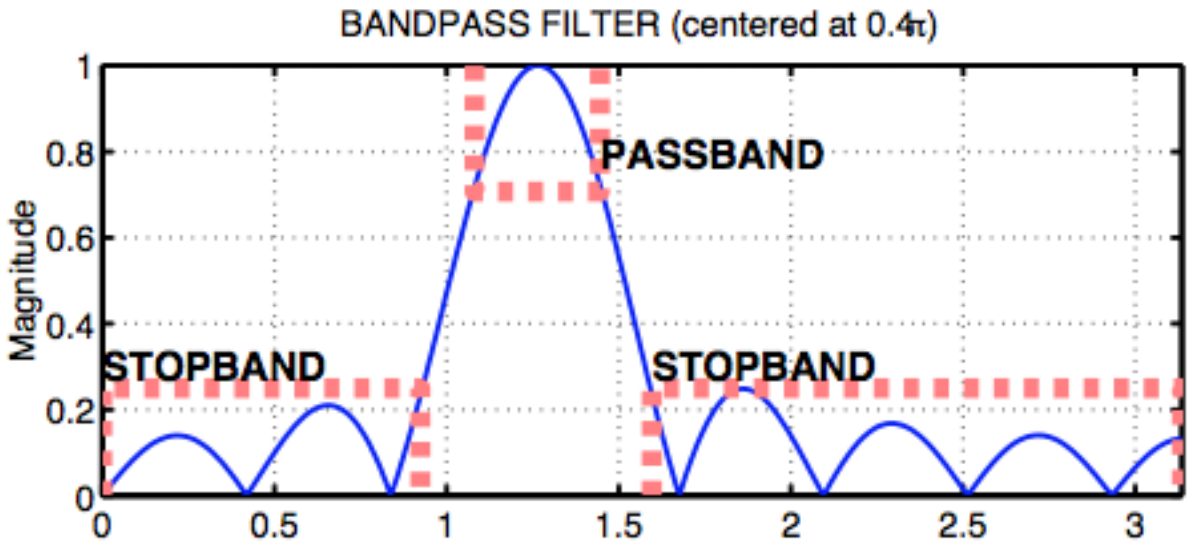
The L-point averaging filter covered earlier is a lowpass filter centered around zero with passband width inversely proportional to L. It is also possible to create a filter whose passband is centered on some frequency other than zero. One simple way to do this is to define the impulse response of an L-point FIR as:

$$h[n] = \frac{2}{L}\cos(\hat{\omega}_c n)$$

$$0 \leq n < L$$

Where L is the filter length, and $\hat{\omega}_c$ is the center frequency that defines the frequency location of the passband. For example, we would pick $\hat{\omega}_c = 0.44\pi$ if we want the peak of the filter's passband to be centered at 0.44π (therefore allowing signals of frequency $0.44\,\pi$ to "pass through" the filter).

(a) Generate a bandpass filter that will pass a frequency component at $\hat{\omega} = 0.44\pi$, 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: $\hat{\omega} = 0.3\pi$, $\hat{\omega} = 0.44\pi$ and $\hat{\omega} = 0.7\pi$.



The *passband* of the BPF filter is defined by the region of the frequency response where $\left|H\left(e^{j\hat{\omega}}\right)\right|$ is close to its maximum value. If we define the maximum to be H_{max}, then the passband width can be defined as the length of the frequency region where the ratio $\frac{\left|H\left(e^{j\hat{\omega}}\right)\right|}{H_{max}}$ is greater than $\frac{1}{\sqrt{2}} = 0.707$, or $3dB$(decibels) below 1. You can use MATLAB's `find()` function to locate those frequencies where the magnitude satisfies $\left|H\left(e^{j\hat{\omega}}\right)\right| \geq 0.707 H_{max}$.

The *stopband* of the BPF filter is defined by the region of the frequency response where $|H(e^{j\hat{\omega}})|$ is close to zero. In this case, we will define the stopband as the region where $|H(e^{j\hat{\omega}})|$ is less than 25%of the maximum, $|H(e^{j\hat{\omega}})| < 0.25H_{max}$.

(b) 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).

(c) 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 $\hat{\omega} = 0.44\pi$, while reducing or rejecting the others at $\hat{\omega} = 0.3\pi$ and $\hat{\omega} = 0.7\pi$.

(d) Generate a bandpass filter that will pass the frequency component at $\hat{\omega} = 0.44\pi$, but now make the filter length (L) long enough so that it will also *greatly* reduce frequency components at (or near) $\hat{\omega} = 0.3\pi$ and $\hat{\omega} = 0.7\pi$. Submit code and frequency response plot. Determine the *smallest* value of L so that the following conditions both hold:

- Any frequency component satisfying $|\hat{\omega}| \leq 0.3\pi$will be reduced by a factor of 10 or more.
- Any frequency component satisfying$0.7\pi \leq |\hat{\omega}| \leq \pi$ will be reduced by a factor of 10 or more.

(e) Use the filter from the previous part to filter the "sum of 3 sinusoids" signal from Section 2.1. 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.

Using the frequency response for the filter from part (d), explain how $H(e^{j\hat{\omega}})$ 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.

2.1 A Better BPF:

It is possible to get better performance in the frequency response by modifying the filter coefficients slightly. One easy way is to use a "Hamming Window." In this case, the impulse response for a length-L bandpass filter would be given as:

$$h[n] = \left(0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right)\right)\cos\left(\hat{\omega}_c\left(n - \frac{L-1}{2}\right)\right)$$

$$n = 0, 1, 2, \ldots, L-1$$

Where $\hat{\omega}_c$ is the desired center frequency for the BPF. As before, the filter length L determines the passband width, although the Hamming BPF tends to be a little wider than the BPF in the previous section. The big advantage of the Hamming BPF is that its stopband has ripples that are very small (usually less than 0.01). If you are curious, use zooming to figure out the height of the ripples in the stopband when you plot the frequency response.

(a) Generate a Hamming bandpass filter that will pass a frequency component at $\hat{\omega} = 0.2\pi$. 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: $\hat{\omega} = \{0, 0.1\pi, 0.25\pi, 0.4\pi, 0.5\pi, 0.75\pi\}$. Summarize the values in a table.

Hint: use MATLAB's `freqz()` function to calculate these values, or the `find()` function to extract this information from the vector that produce the plot.

(b) The passband of the BPF filter is defined by the region of the frequency response where $|H(e^{j\hat{\omega}})|$ is close to its maximum value of one. In this case, the passband width is defined as the length of the frequency region where $|H(e^{j\hat{\omega}})|$is greater than 50% of the peak magnitude value. The stopband of the BPF filter is defined by the region of the frequency response where $|H(e^{j\hat{\omega}})|$is close to zero.

Use the plot of the frequency response for the length-41 bandpass filter from part (a), and determine the passband width using the 50% level to define the pass band. Make two other plots of BPFs for $L = 21$and $L = 81$, and measure the passband width in both. Then explain how the width of the passband is related to filter length L, i.e., what happens when L is (approximately) doubled or halved.

(c) If the input signal to the length-41 FIR BPF is:

$$x[n] = 2 + 2\cos\left(0.1\pi n + \frac{\pi}{3}\right) + \cos\left(0.25\pi n - \frac{\pi}{3}\right)$$

Determine (by hand) the formula for the output signal.

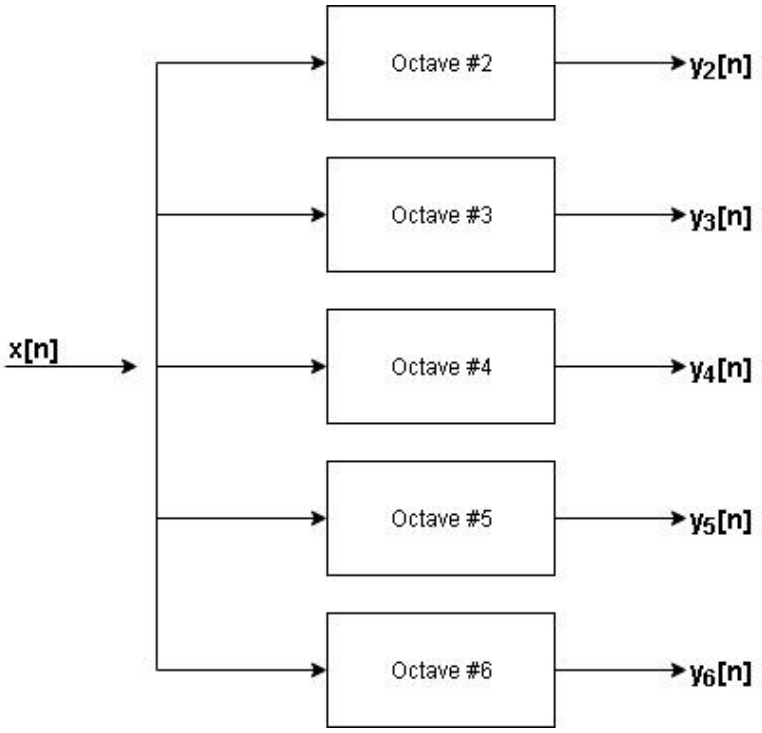
Hint: use the magnitude and phase measurements from part (a).

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

(d) 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 $\hat{\omega} = \pm 0.25\pi$, while reducing or rejecting others.

2.2 Octave Filtering

It is possible to decode piano signals or music in general into octaves using a simple FIR filter bank. The filter bank below consists of five bandpass filters which correspond to octave 2 to 6. The input signal for all the filters is the same.



The octave filtering system needs a set of bandpass filters (BPFs) to isolate individual frequency bands (i.e., the octaves). To make the whole system work, you will have to find out the parameters for the bandpass filters.

The bandpass filter specs are determined by the frequencies of different octaves on the piano. The standard notation is to start an octave at the "C" key. For example, octave #4 starts at "middle-C" which is key #40 and extends up to key #51; the fifth octave starts at key #52 and extends to key #63, and so on.

In this implementation, we want to design filters that will isolate five different octaves: octaves #2 through #6. Octave #2 starts with key #16, octave #3 starts at key #28, and octave #6 starts at key #64 (the last key in octave #6 is key #75).

In order to design the BPFs, we need the upper and lower frequencies for each octave in $\hat{\omega}$. We can convert key number to frequency in hertz, and then use the sampling frequency to convert analog frequency to $\hat{\omega}$. For this lab, assume that the sampling frequency is $f_s = 8000\ Hz$.

Below, we provide you with the lower edge, higher edge, and center frequencies of all octave (in Hz). Based on this, calculate the same parameters in terms of normalized radiant frequency $\hat{\omega}$, and submit a table consists of those values.

Octave	2	3	4	5	6
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	1511

2.3 Bandpass Filter Bank Design

The FIR filters that will be used in the filter bank should be constructed according to the Hamming impulse responses described in Section 2.1(*A Better BPF*). These "Hamming" BPFs require two parameters: the length L and the center frequency $\hat{\omega}_c$. Use the previous table of piano octave frequencies to define the passbands. Furthermore, the filters should be scaled so that their maximum magnitude at the center frequency of the passband is equal to one. This can be done by introducing a third parameter β that will scale all the filter coefficients.

$$h[n] = \beta\left(0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right)\right)\cos\left(\hat{\omega}_c\left(n - \frac{L-1}{2}\right)\right)$$

$$n = 0, 1, 2, \ldots, L-1$$

The constant β gives flexibility for scaling the filter's gain to meet a constraint such as making the maximum value of the frequency response equal to one. The bandwidth of the bandpass filter is controlled by L; the larger the value of L, the narrower the bandwidth.

(a) Devise a strategy for picking the constant β so that the maximum value of the frequency response will be equal to one. Write the one or two lines of MATLAB code that will do this scaling operation in general. We will use the following approach:

Numerical: use MATLAB's `max()` function to measure the peak value of the unscaled frequency response, and then have MATLAB compute β to scale the peak to be one.

(b) You must design five separate bandpass filters for the filter bank system. Each filter has a different length and a different center frequency. The lengths have to be

different because the bandwidths of the octaves are different. In fact, the bandwidth of each octave is twice that of the next lower octave.

For each filter, determine the length L that is required to get the correct bandwidth. Use the bandedges determined in Section 2.3. This filter design process will be trial-and-error, so each time you change L you will have to make a frequency response plot (magnitude only) to see if the filter is correct.

(c) Generate the five (scaled) bandpass filters. Plot the magnitude of the frequency responses all together on one plot (the range $0 \le \hat{\omega} \le \pi$ is sufficient because $|H(e^{j\hat{\omega}})|$is symmetric). Indicate the locations of each of the center frequencies of the five octaves on this plot and illustrate that the passbands cover the separate octaves.

Hint: use the `hold` command and markers as you did in Section 1.2.

As help for the previous parts, here are some comments: The passband of the BPF filter is defined by the region of $\hat{\omega}$ where $|H(e^{j\hat{\omega}})|$ is close to one. In this Lab, the passband width is defined as the length of the frequency region where $|H(e^{j\hat{\omega}})|$ is greater than 0.5 of its maximum value.

Filter Design Specifications: For each octave, choose L so that the entire octave of frequencies lies within the passband of the BPF.

The stopband of the BPF filter is defined by the region of $\hat{\omega}$ where $|H(e^{j\hat{\omega}})|$is close to zero. In this case, we can define the stopband as the region where $|H(e^{j\hat{\omega}})|$ is less than 0.01of its maximum value, because the Hamming filters have excellent stopbands. Notice, however, that the stopbands do not eliminate all the other octaves because the stopband does not start where the passband ends. There is a small "transition region" between the pass and stop bands where the frequency response is small, but not as small as in the stopband.

(d) Use the `zoom on` command to examine the frequency response over the frequency domain where the octaves lie. Comment on the selectivity of the bandpass filters, i.e., use the frequency response (passbands and stopbands) 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?

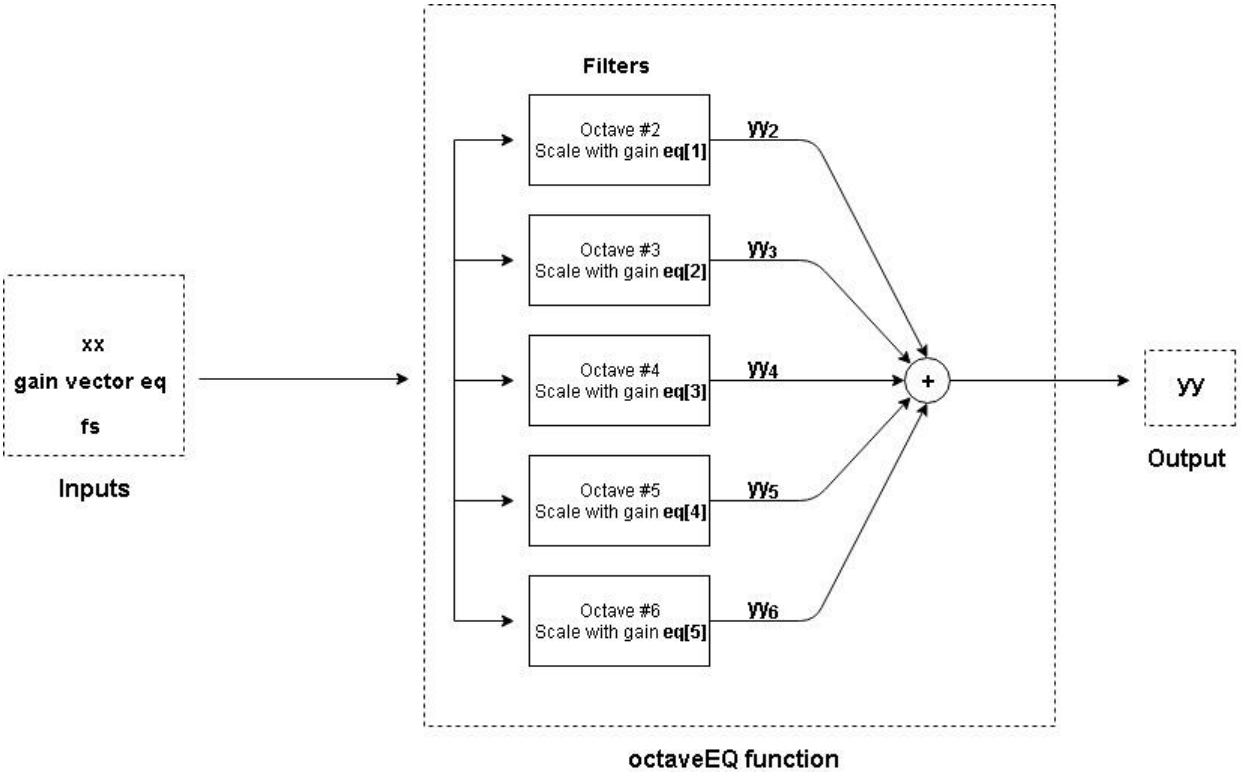
2.5 Equalizer

An interesting application of octave filtering is making an equalizer. In music, songs can have many notes that spans many octaves. What if we want to amplify bass notes without affecting notes from other octaves? What if we want to attenuate high notes while amplifying bass notes? In these situations, equalizer becomes extremely helpful.

The equalizer takes in the song, then uses octave filter banks to extract notes from the same octave. Once extracted, notes from the same octave will be scaled with desired gain factor.

Finally, the song will be reconstructed by summing all the notes back together. Using this method, we can modify music to our preference.

In this portion of the lab, you will make an equalizer consists of 5 octave filters that have been generated in previous section (octave #2 to octave #6). The goal of this section is to create an equalizer function called **octaveEQ** that takes in song vector **X, gain vector, and sampling frequency** then output new song vector **Y** with modified gain for notes in each octave. Here's the flowchart:



(a) From **x-file.wav**, **use audioread() function to obtain the vector and sampling frequency of your input**. Note: use the [Y, FS]=audioread(FILENAME) iteration of this function.

(b) The center frequency of each octave (Hz) are tabled from section 2.2. **Use fs obtained in part a to re-calculate the normalized center frequencies. Submit a table similar to section 2.2 with this sampling frequency.** With the same code you used in previous section to generate the five octave filters, **modify your filters to satisfies these filter lengths using a loop:**

Octave	#2	#3	#4	#5	#6
Filter length	256	128	64	32	16

(c) Since different filter lengths would results in different output length, and as we know, 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.**

(d) **Turn your script into a function that takes in the input vector xx, scale vector eq, sampling frequency fs and return output vector yy.** Your function must:

- Create a vector of center frequency based on the inputted sampling frequency fs.
- Create five octave filters of the same length as the longest filter.
- Scale each filter with a factor of $10^{(eq(i)/20)}$ where eq is the scale vector.
- Sum up all five filter coefficients.
- Use convolution to obtain yy.

(e) Test your function with input vector and sampling frequency obtained in part a. **Use scale vector eq = [20, 50, 70, 0, 0]. Plot magnitude response of your overall filter from -pi to pi. Play the output vector yy to a TA for check-off.**

Lab 6 Part I Appendix

A.1 Frequency Response in Matlab

Frequency response information indicates *how* a filter affects a given input signal: the frequency-dependent output or *response* of a filter depends on the frequency $\hat{\omega}$ of the complex sinusoid input $e^{j\hat{\omega}n}$. Finding the frequency response of a filter is relatively straightforward if we know the impulse response of the filter. For example, take the 2-point averaging filter

$$y[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-1]$$

If we input a general complex exponential, of the following form:

$$x[n] = Ae^{j(\hat{\omega}n+\varphi)}$$

We would observe the following output:

$$y[n] = \frac{1}{2}Ae^{j(\hat{\omega}n+\varphi)} + \frac{1}{2}Ae^{j(\hat{\omega}(n-1)+\varphi)} = Ae^{j(\hat{\omega}n+\varphi)}\left\{\frac{1}{2} + \frac{1}{2}e^{-j\hat{\omega}}\right\} = Ae^{j(\hat{\omega}n+\varphi)} \cdot H\left(e^{j\hat{\omega}}\right)$$

The output is thus the original signal multiplied by a frequency-dependent function, $H\left(e^{j\hat{\omega}}\right)$ – this frequency-dependent function **is the frequency response.** This frequency response, then, for a general FIR linear time-invariant system is of the form:

$$H\left(e^{j\hat{\omega}}\right) = \sum_{k=0}^{M} b_k e^{-j\hat{\omega}k}$$

In the example at the start of this section, $M = 1$, $b_0 = \frac{1}{2}$, and $b_1 = \frac{1}{2}$.Note that frequency response is always periodic in 2π. So, the following frequency responses:

$$H_1\left(e^{j\hat{\omega}}\right) = e^{-j\hat{\omega}} \text{ and } H_2\left(e^{j\hat{\omega}}\right) = e^{-j(\hat{\omega}+2\pi)}$$

both yield the same frequency response (as would any multiple of 2π, because $e^{-j2\pi} = 1$).

MATLAB has a built-in function called `freqz()` to compute the frequency response of a discrete-time LTI system. The following MATLAB statements give an example of how to use `freqz()`.

```
bb = [0.5, 0.5];          %<-- filter coefficients
ww = -pi:(pi/100):pi;     %<-- omega hat
HH = freqz(bb, 1, ww);
subplot(2, 1, 1);
plot(ww, abs(HH))
subplot(2, 1, 2);
plot(ww, angle(HH))
xlabel('Normalized Radian Frequency')
```

When run, this code will display two graphs: the *magnitude* and *phase* of the frequency response for the specified FIR filter.

Note: For FIR filters, the second argument must equal 1 (an explanation of why is deferred to future lab documents). Also, the frequency vector `ww` should cover a length of 2π radians so as to observe the frequency response of all possible signal frequencies for a given sampling frequency.

Lab 6 Part 2 Appendix

A.2 Frequency Response of FIR Filters

The response of a digital filter, or output relative to amplitude of a sampled complex sinusoid input $e^{j\hat{\omega}n}$, depends on the normalized radian frequency $\hat{\omega}$. Often a filter is described solely by how it affects different input frequencies - this is called the frequency response. The frequency response of an FIR linear time-invariant system is

$$H(e^{j\hat{\omega}}) = \sum_{k=0}^{M} b_k e^{-j\hat{\omega}k}$$

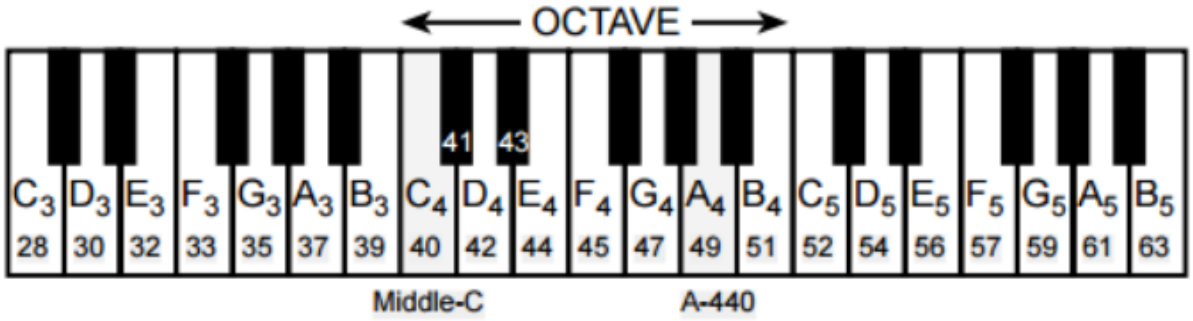
MATLAB has a built-in function for computing the frequency response of a discrete-time LTI system. The following MATLAB statements show how to use `freqz()` to compute and plot the magnitude (absolute value) of the frequency response of an L-point averaging system as a function of $\hat{\omega}$ in the range $-\pi \leq \hat{\omega} \leq \pi$:

```
bb = ones(1, L)/L;     %<-- filter coefficients
ww = -pi:(pi/100):pi;   %<-- omega hat
HH = freqz(bb, 1, ww);  %<-- freekz.m is an alternative
subplot(2, 1, 1);
plot(ww, abs(HH))
subplot(2, 1, 2);
plot(ww, angle(HH))
xlabel('Normalized Radian Frequency')
```

For FIR filters, the second argument of `freqz()` must always be equal to 1, as FIR filters have no feedback. The frequency vector `ww` should cover the interval $-\pi \leq \hat{\omega} \leq \pi$ for $\hat{\omega}$, and its spacing must be fine enough to give a smooth curve for $H(e^{j\hat{\omega}})$.

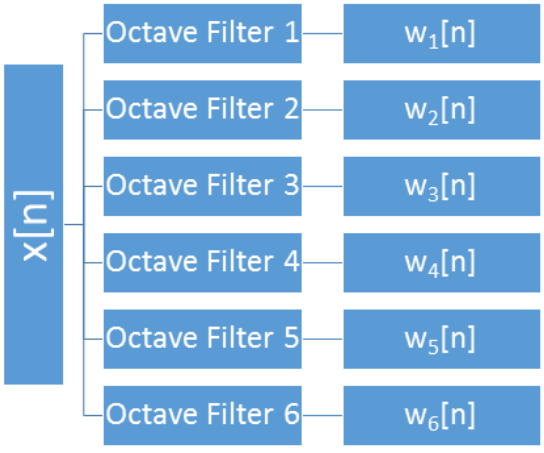
A.3 Piano Notes

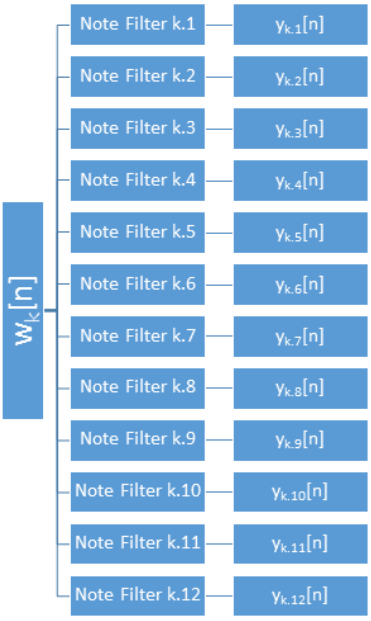
A piano keyboard consists of 88 keys grouped into octaves. Each octave contains 12 notes, the notes in one octave being twice the frequency of the notes in the next lower octave.



Frequencies of the notes are defined by setting the frequency of one note and referring all other frequencies to that note. The reference note is the A above middle-C, which is usually called A-440 (or A4) because its frequency is 440 Hz and it is in the fourth octave. Each octave contains 12 notes (5 black keys and 7 white) and the ratio between the frequencies of the notes is constant between successive notes. As a result, this ratio must be $2^{\frac{1}{12}}$. Since middle C is 9 keys below A-440, its frequency is approximately 261 Hz. Consult Lab 3 for more details.

If we want to produce a system capable of writing music directly from a recorded signal $x(t)$, we need to analyze the frequency content of the signal. One way to do this analysis is to use a set of bandpass FIR filters, each one having its passband designed for one note on the keyboard. This would require a very large number of filters. Another way to do the analysis would be to use a two stage approach. First, we would use a set of bandpass FIR filters where each passband would pass the frequencies in one octave. Then these "octave filters" would be followed by more precise bandpass filters (BPFs) that would determine which key inside the octave is being played. The goal of this lab will be to produce a working set of "octave filters."





A.4 Signal Concatenation

In a previous lab, a very long music signal was created by joining together many sinusoids. When two signals are played one after the other, the composite signal is created by the operation of concatenation. In MATLAB, this can be done by making each signal a row vector, and then using the matrix building notation:

```
xx = [xx, xxnew];
```

where `xxnew` is the sub-signal being appended. The length of the new signal is equal to the sum of the lengths of the two signals `xx`and `xxnew`. A third signal could be added later on by concatenating it to `xx`. However, this becomes an inefficient procedure if the signal length gets to be very large. The reason is that MATLAB must re-allocate the memory space for `xx`every time a new sub-signal is appended via concatenation.

For example, if the length `xx` were being extended from 400,000 to 401,000, then a clean section of memory consisting of 401,000 elements would have to be allocated, 400,000 signal elements would have to be copied, and only then would the new elements be added. While this is not as big a deal for short signals, it is very inefficient for long signals.

An alternative is to pre-allocate storage for the complete signal vector, but this can only be done if the final length is known ahead of time. In other words, your final signal variable and its length would have to be known and defined in MATLAB before processing.