Lab 5 Information

- The material in this section is informational. Please read through the section as it helps you work on the lab exercises in the next section. There may be code examples in this informational section. You are welcome to copy-and-paste them to MATLAB to run the code, but no submission is needed on any test run.

Plotting Frequency Response in MATLAB

One can easily plot the freqeuncy response $H(e^{j\hat{\omega}})$ of any LTI system in MATLAB. The important thing to remember is that $H(e^{j\hat{\omega}})$ is a complex-valued function of $\hat{\omega}$; hence we need to use two plots to completely describe $H(e^{j\hat{\omega}})$. Typically, we plot the magnitude response $\left|H(e^{j\hat{\omega}})\right|$ and the phase response $\angle H(e^{j\hat{\omega}})$ versus $\hat{\omega}$. Note also that $H(e^{j\hat{\omega}})$ is periodic in $\hat{\omega}$ with period 2π; thus we only need to plot one period of $H(e^{j\hat{\omega}})$. Typically, we plot from $\hat{\omega} = -\pi$ to π.

For an example, consider the first difference filter specified by the difference equation:

$$y[n] = x[n] - x[n-1].$$

We have worked out in the lecture videos that its frequency response is

$$H(e^{j\hat{\omega}}) = 1 - e^{-j\hat{\omega}}.$$

To plot the magnitude and phase response, we need to first create a frequency vector:

```
w = -pi:pi/1000:pi;
```

The increment in `w` above specifies how fine we evaluate $H(e^{j\hat{\omega}})$ between $\hat{\omega} = -\pi$ to π. Then we can plot the responses using the following piece of code:

```
H = 1-exp(-1j*w); % FREQUENCY RESPONSE

figure;
subplot(2, 1, 1)
plot(w/pi, abs(H));
grid on;
title('Magnitude Response')
xlabel('Normalized Radian Frequency (\times \pi rad/sample)');
ylabel('Amplitude');
subplot(2, 1, 2)
plot(w/pi, angle(H)/pi);
grid on;
title('Phase Response')
xlabel('Normalized Radian Frequency (\times \pi rad/sample)');
ylabel('Phase (\times \pi rad)');
```

It is sometimes more desirable to plot the magnitude response in dB scale. To do so, we can replace the corresponding lines of code above by

```
plot(w/pi, 20*log10(abs(H)));
ylabel('Amplitude (dB)');
```

The MATLAB function `freqz` also calculates and plots the frequency response of an FIR filter (can do actually beyond FIR filters). You may use it to check your frequency response calculation and plots. Use the `help` command to learn more about the `freqz` function.

Lab 5 Exercises

- Unless stated otherwise, you must submit your solutions to all the lab exercises in this section.
- Your laboratory solutions should be submitted on Canvas as a single PDF. The simplest way is to put your codes and answers for all the lab exercises in a single MATLAB Publisher script and use `%%` to separate the codes and answers for different exercises into different sections as described in the information section of Lab 1.

Exercise 5.1: *(Bandpass Filter)*

This exercise shows you how to plot the frequency response (magnitude and phase) of a bandpass FIR filter and apply it to a sum of sinusoids.

(a) Given an FIR filter of order M specified by the difference equation:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]. \tag{1}$$

Analytically show that the frequency response of the FIR filter is

$$H(e^{j\hat{\omega}}) = \sum_{k=0}^{M} b_k \, e^{-j\hat{\omega}k}. \tag{2}$$

Create a new function `H = freq_resp(b, w)` where the input vector `b` is a vector of filter coefficients (`b` is also the impulse response vector of the FIR filter, why?), input vector `w` is a vector of angular frequencies, and output vector `H` is the complex-valued frequency response given in (2). Hint: The most straightforward approach is to use a for-loop to calculate each term (for each k) in the sum of (2).

(b) Consider the bandpass FIR filter of order M (length $L = M + 1$) whose FIR filter taps are given by

$$b_k = A\cos(\hat{\omega}_0 k), \tag{3}$$

for $k = 0, 1, \ldots, M$, where $\hat{\omega}_0$ is a normalized radian frequency parameter (between $-\pi$ and π) that specifies the center of the filter's passband and A is a gain parameter chosen such that the maximum value of $\left|H(e^{j\hat{\omega}})\right|$ is set to 1.

The following steps show you how to choose A given specific values of $\hat{\omega}_0$ and L. Write a MATLAB function `b = gen_filter(w0, L)` to carry out the steps, where `w0` inputs $\hat{\omega}_0$, `L` specifies the length of the filter, and `b` is an output vector that contains all the filter taps:

(i) Generate an `L`-length vector `b_unit_gain` that contains the filter taps by setting $A = 1$ in (3). Notice that the values of $\hat{\omega}_0$ and L are specified by the input arguments of your function.

(ii) Apply your function

```
H = freq_resp(b_unit_gain, w);
```

in (a) to calculate the frequency response of the FIR filter with taps specified by `b_unit_gain`.

(iii) Find the maximum magnitude of `H` by

```
max_mag = max(abs(H));
```

(iv) The value of A that normalizes the maximum magnitude response to unity is the reciprocal of `max_mag`. That is, we can set the normalized tap vector

```
b = b_unit_gain / max_mag;
```

(c) Set $\hat{\omega}_0 = \frac{\pi}{4}$ and $L = 20$ for the rest of this exercise. Apply your function `gen_filter` to obtain the filter tap vector `b`. Use your function `freq_resp` to calculate the frequency response `H` of the FIR filter `b`. Plot the magnitude response in linear scale and the phase response of the filter using `subplot` as shown before. Check to make sure that the maximum value of the magnitude response is normalized to 1, i.e., your function `gen_filter` correctly does its job. Determine the normalized radian frequency at which the magnitude response reaches its maximum value 1. (If you want to verify that your `freq_resp` is correctly implemented, you may check its output against the one obtained by using the MATLAB bulit-in function `freqz`.)

(d) The following discrete-time signal

$$x[n] = 2 + \cos\left(\frac{\pi}{4}n\right) + \cos\left(\frac{3\pi}{4}n + \frac{\pi}{2}\right) \tag{4}$$

is entered into the FIR filter that you generated in (c) to obtain the output signal $y[n]$. Analytically find the expression of $y[n]$. You may use the plots of the magnitude and phase response in (c) to help you obtain the expression.

(e) Generate $x[n]$ in MATLAB for $n = 0, 1, \ldots, 79$. Put the resulting signal vector in `x`. Use the MATLAB function `filter` to apply the FIR filter `b` in (c) on `x` to get the output vector `y`. (You may also use `conv` to apply the filter. However the output vector `y` will then be longer than the input `x`. Do you know by how much? You should be able to answer by referring back to Lab 4.)

Use `subplot` and `stem` to plot `x` and `y` one on top of the other over the range of discrete time $n = 0, 1, \ldots, 79$. Label the horizontal axis "Samples" and vertical axis "x[n]" or "y[n]". If both your expression in (d) and your application of the FIR filter to `x` here are correct, you may expect the plot of the output signal `y` to match the expression of $y[n]$ obtained in (d) over the whole time range of $n = 0, 1, \ldots, 79$. Is that what you observe? If not, provide the range of n over which the plot of `y` matches the expression of $y[n]$ and the range over which they don't match. Explain the cause for any differences.

Exercise 5.2: *(Octave Filters)*

Recall in Lab 2, we introduced the idea of music synthesis based on playing keys on a piano keyboard. Each set of 12 consecutive keys is called an *octave*. The following table (in MATLAB comments) gives the key numbers from octave 2 to octave 7:

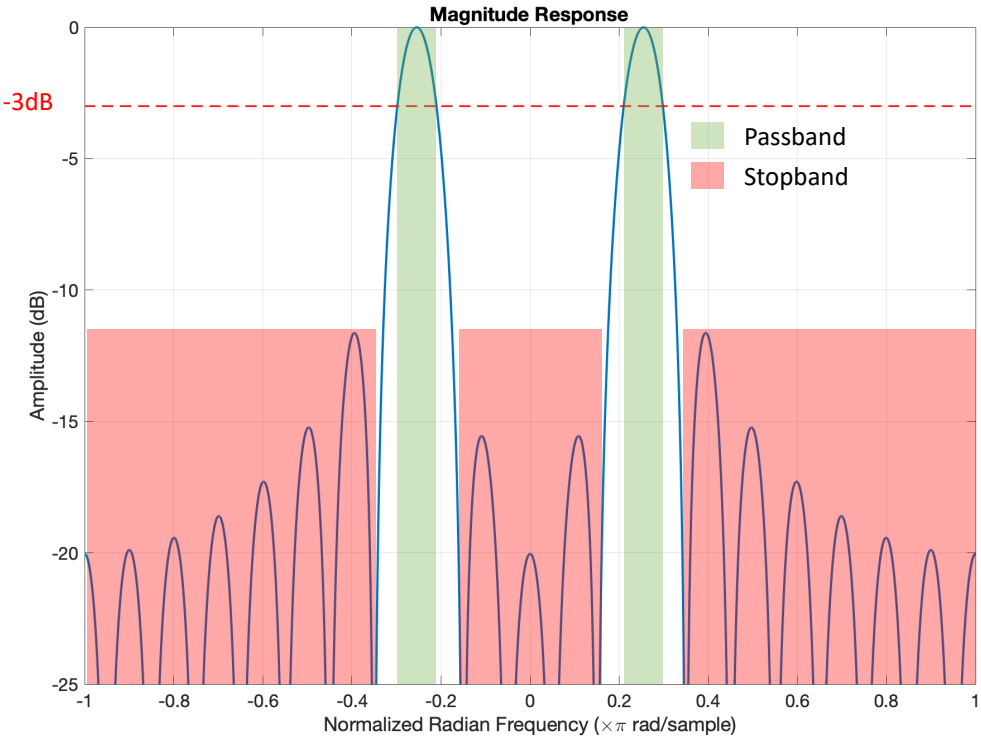
```
% Octave           :      2      3      4      5      6      7
% Start key #      :     36     48     60     72     84     96
% End key #        :     47     59     71     83     95    107
% Start freq (Hz) :   65.4  130.8  261.6  523.3 1046.5 2093.0
% End freq (Hz)    :  123.5  246.9  493.9  987.8 1987.5 3951.1
% Center freq (Hz):   94.4  188.9  379.2  755.5 1551.0 3022.0
```

Recall that each key corresponds to a note of a specific fundamental frequency given by the following formula:

$$f = 440 \cdot 2^{(k-69)/12}.$$

Using this formula, we can calculate the note frequencies of the start and end keys of the octaves shown in the table above. We may treat the frequencies of the start note and end note as the start frequency and end frequency of the octave, respectively. Further, the mid point between the start and end frequency is the *center frequency* of the octave.

In this exercise, we use the bandpass filter prototype in (3) to design filters whose *passbands* coincide with the octave bands. To illustrate how we are to design the filter, let us plot below the magnitude response in dB scale of the bandpass filter in Exercise 5.1(c). Clearly, we see that the



magnitude response peaks at $\hat{\omega} = \frac{\pi}{4}$ (and $\hat{\omega} = -\frac{\pi}{4}$, why this too?) with the maximum value of 0 dB (why?). Draw a horizontal line at the value of -3 dB (the broken red line shown in the figure), we can work out the range of normalized radian frequency over which the value of the magnitude

response is above -3 dB. This range of frequency is called the *passband* of the filter, shown by the green-shaded region in the figure. The -3 dB threshold is an arbitrary but common choice. The red-shaded region, over which the magnitude response is very small, is called the *stopband* of the filter. The unshaded frequency range is called the *transition band*. For the purpose of this exercise, we care only about the passband.

Because of symmetry, we may focus only on the positive-frequency passband. In particular, we want to determine its start frequency, end frequency, and its width (called the *bandwidth* of the filter). We can measure all these values from the plot of the magnitude response. Nevertheless, we can determine them more accurately using the MATLAB function `find` (do a `help` to learn the function better):

(i) Assume that the frequency response of the filter is generated by

```
w = -pi:pi/1000:pi;
b = gen_filter(pi/4, 20);
H = freq_resp(b, w);
```

(ii) Find the normalized radian frequencies in `w` over which the value of magnitude response is at least as large as 0.7071 (-3 dB). This set of frequencies constitutes the passband of the filter:

```
index_passband = find(abs(H) >= 0.7071);
w_passband = w(index_passband);
```

(iii) The minimum and maximum nonnegative normalized radian frequency in `w_passband` give us the start and end frequency of the passband, from which we can also calculated the bandwidth of the filter:

```
w_passband_positive = w_passband(find(w_passband >= 0));
start_freq = min(w_passband_positive)
end_freq = max(w_passband_positive)
bandwidth = end_freq - start_freq
```

Note that the accuracy of the frequency and bandwidth values obtained using this procedure depends on the resolution in the normalized frequency vector `w`. You may improve the accuracy by using a normalized frequency vector with a finer resolution (how?).

Below are the lab assignments for which you must submit your code and solutions:

(a) Write a MATLAB function

```
[b, f_start, f_end, bw] = gen_filter_w_info(w0, L);
```

to generate the bandpass filter prototype given in (3) and calculate the start frequency, end frequency and bandwidth of the filter. All the output frequency and bandwidth variables should be in the unit of rad/sample.

(b) Use your function `gen_filter_w_info` to calculate the start and end frequency of the passband and the bandwidth of the FIR filter with parameters $\hat{\omega}_0 = \frac{\pi}{4}$ and $L = 20$. Redo the calculation for the parameters $\hat{\omega}_0 = \frac{\pi}{4}$ and $L = 40$. Can you deduce a general rule that tells us approximately by how much the bandwidth increases when the length L halves?

(c) Assume that the sampling frequency $f_s = 11025$ Hz is employed for the rest of this exercise. Convert the start, end, and center frequencies of the six octaves in the table given above from cyclic frequency in Hz to normalized radian frequency in rad/sample. You may copy the table (as MATLAB comments) and add three more rows for the corresponding start, end, and center frequency in rad/sample.

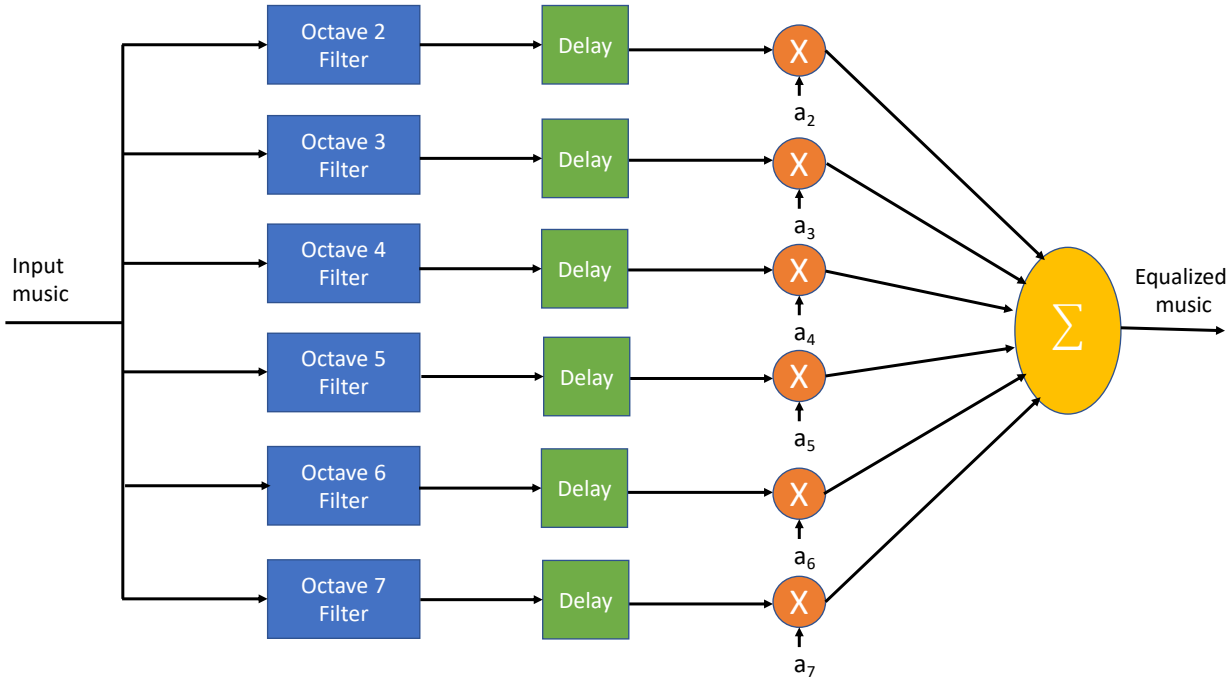
(d) Use your function `gen_filter_w_info` repeatedly to design bandpass filters `b2`, `b3`, `b4`, `b5`, `b6` and `b7` so that the passbands of the filters match the corresponding octave bands. For convenience later, you may also want to put the resulting `b` vectors together in a matrix, in which each row (or column) gives the filter tap vector for an octave.

When designing an octave filter, you should clearly choose the input parameter `w0` to be the normalized radian center frequency of the octave. To match the start and end frequency, you need to try different choices of `L`. Note that you may not be able to get exact matches. Choose the value of `L` that gives the start and end frequency of the filter closest to those of the octave. You may do this by trial and error (perhaps using the relation that you discover in (c)), or you may use a for-loop to enumerate all reasonable choices of `L` and then select the best `L`. In any case, report all intermediate results as well as the final results of your design process for each of the 6 filters. Plot the magnitude response of your final design for each of the 6 octave filters.

(e) Use `audioread` to load the included WAV file `x-file.wav`. Apply your octave 3 filter to the sound signal. Save the filter output to the WAV file `x-file-octave3.wav` and submit it with your report. Describe what you hear after the filtering by your octave 3 filter. Repeat using your octave 7 filter instead (save to the WAV file `x-file-octave7.wav`).

Exercise 5.3 (Extra credits: +20 points):

This exercise shows you how to use your octave filters in Lab Exercise 5.2 to build a simple music *equalizer*, which you may use to emphasize or suppress the sound components in different octaves. The idea of this simple equalizer is best described by the following block diagram:



A music signal is filtered by each of the six octave filters. The octave filters introduce different amounts of delay to the input music signal. The amount of delay introduced by an octave filter depends on the length of the octave filter. For our purpose here, we can approximate the amount of delay by $\lceil \frac{L-1}{2} \rceil$, where L is the length of the octave filter. Because the six octave filters are of different lengths, their output signals experience different amounts of delay. To compensate for the different delays, we need to further delay the octave filters' output signals by different amounts. For example, let $L_2 > L_3 > \cdots > L_7$ be the lengths of the octave filters. Then the corresponding delays experienced by the input signal will be $\lceil \frac{L_2-1}{2} \rceil > \lceil \frac{L_3-1}{2} \rceil > \cdots > \lceil \frac{L_7-1}{2} \rceil$. We may compensate by further delaying the output of the octave k filter by $\lceil \frac{L_2-1}{2} \rceil - \left\lceil \frac{L_k-1}{2} \right\rceil$, for $k = 2, 3, \ldots, 7$. The delayed filter output signals are then amplified or attenuated according the gain values $a_2, a_3, \ldots, a_7$ for the six octaves. The scaled outputs from the octave filters are then superimposed to recreate the equalized music. Hence the equalization effect is specified by the choices of the gain values $a_2, a_3, \ldots, a_7$.

Write a MATLAB function to implement the block diagram above. For simplicity, you may fix the sampling frequency $f_s = 11025$ Hz so that you can use the octave filters that you design in Exercise 5.2. If you choose to allow users to specify the value of f_s, then you will need to design the octave filters (choose $\hat{\omega}_0$ and L for each of them) on the fly according to the given value of f_s. That is considerably more difficult. Your MATLAB function should be of the form `y = equalize(x, a)` (or `y = equalize(x, a, fs)` if you allow users to specify f_s), where `x` is the input music signal, `a` is a vector of 6 elements specifying the gains $a_2, a_3, \ldots, a_7$ **in dB**, and `y` is the equalized output music. Apply your `equalize` function to the `x-file` music with one choice of `a` that emphasizes

the bass but suppresses the treble and one choice that emphasizes treble but suppresses the bass. Save the equalized music to WAV files `x-file-bass.wav` and `x-file-treble.wav`, respectively. Remember to report your choice of `a` for each case.