ECEn 380 — Laboratory Assignment 5
Audio Response for Real-World LTI Systems and Aliasing from Undersampling

Lab Schedule and Submission Requirements:

You will have two lab periods to complete this laboratory assignment. The lab exercises are organized into 2 main tasks: 1) applying real-world system impulse responses to audio signals, and 2) an exercise in aliasing due to undersampling. Work through the exercises and neatly track your progress in your lab book. Save all of your MATLAB code, and answer all questions in your lab book. A complete lab report is due on the following dates by the beginning of your assigned lab session that day:

- Sections 1 and 3: Monday, November 16
- Section 2: Friday, November 13
- Section 4: Tuesday, November 10

The report must consist of the following to be deemed complete: 1) a detailed log of all laboratory exercises including derivations, code listings (which should be relatively short for this lab, so please include them), code output with well-labeled figures, answers to all questions posed in the lab, and thoughtful responses and/or descriptions of lab book content; 2) a summary paragraph of the exercises you completed in the lab; and 3) a conclusions paragraph identifying important principles from the lab and discussing what you have learned. This lab really should be doable in less time than we're giving you, but we hope you have fun with it and try a lot of variants. Also, we're not giving you axes labels in the code we provide, but any good engineer should label all of their axes in plots, so don't forget to do that.

Objectives:

The first task of this lab seeks to drive home some of the concepts that we've been learning throughout the semester, including the power and simplicity of signal filtering with linear, time-invariant (LTI) systems. In the first task, you will record some audio at a sampling rate of 96,000 samples/second. You'll convolve the signal with several different impulse responses (also sampled at 96 kHz), yielding a filtered audio signal that you can listen to. (Remember that LTI systems are typically frequency-selective filters, which simply apply a gain and phase shift to each input frequency.) The results can be quite interesting.

The second task in the lab seeks to make the concept of aliasing that we've recently learned about a bit more real. You will generate tones at several different frequencies, and then sample the signals below the Nyquist rate for those frequencies and observe the aliasing. You will describe what is taking place in the Fourier domain.

Prelab:

Before showing up to your designated lab section, complete the following tasks:

1. read this entire lab assignment and **highlight important information that may be hard to find later**;
2. read the help documents for MATLAB functions: `audiorecorder`, `record`, `recordblocking`, `getaudiodata`, `audioplayer`, `play`, `disp`, `conv`, and `freqz`;
3. be familiar with concepts from Section 8.2-1, particularly sampling and aliasing;

4. work and understand Example 8.4 on pp. 795–796 from L&G in your lab book;
5. **bring your breadboard, collection of circuit components, and textbook** with you to lab!

Task 1—LTI Filtering to Model a Listening Environment for an Audio Signal

In this task you will record an audio signal in MATLAB (sampled at a rate of 96 kHz), and then filter the audio signal through several different LTI systems. We will be working in the time domain in this task, and filtering by an LTI system will be accomplished by convolving the input signal (your audio recording) with the impulse response of the LTI system.

Procedure

This lab will be conducted using MATLAB. Launch MATLAB, making sure to launch it locally (not through the Citrix service) since we will be interfacing with the audio hardware of your lab computer.

1. Familiarize yourself with the `audiorecorder` documentation in MATLAB. The `audiorecorder` object is MATLAB's mechanism for recording audio. Before you record audio, you need to determine:

   (a) **What sampling rate do you wish to use?** The sampling rates you can choose are actually sound card dependent, but most sound cards will support sampling at 8000, 11025, 22050, 44100, 48000, and 96000 Hz. The sound card will automatically send the incoming signal through an anti-aliasing filter prior to sampling based on the sample rate you choose.

   In your lab book, record what you think an appropriate corner frequency would be for the anti-aliasing filter in the sound card for each of the sampling frequencies given above while assuming you have a perfect, ideal anti-alias filter with a zero-width transition band for each of the sample frequencies. Is this a realistic assumption? Why or why not?

   (b) **How many bits do you wish to use to record each sample?** Most sound cards will typically support 8, 16, or 24 bits per sample.

   In your lab book, describe what the advantages and disadvantages are of using 8 bits to record each sample vs. 24 bits.

   (c) **How many sound channels will you be recording?** This will typically be 1 for mono or 2 for stereo.

   In this lab, we're going to record audio sampled at a rate of 96 kHz, recorded with 24 bits per sample, and we're going to only record in mono (a single channel). The sampling rate and bits per sample are really overkill for what we are trying to accomplish, but it will ensure that we aren't losing high frequencies in our sound, and that the recording is of very good quality. However, make sure that your recordings are fairly short in order to keep the processing time for the convolution reasonable and to keep our sound array data sizes reasonable.

2. Use the following code to record a 5 second sample of you speaking into the microphone:

```
recObj = audiorecorder(96000, 24, 1); % (sampling rate, bits/sample, channels)
disp('Start speaking.'); % This just outputs text
```

```
recordblocking(recObj, 5); % This does the recording/sampling, for 5 seconds
disp('End of Recording.');
```

You can now use the `audiorecorder` object directly to play back what was recorded as follows:

```
play(recObj); % This directly plays back the recording using the recording sampling
rate
```

Right now, the sampled data from your recording is stored in `recObj`, the `audiorecorder` object you created. You can get the data out so you can directly manipulate it like this:

```
myRecording = getaudiodata(recObj); % Grab the sampled data from recObj
```

Now the sampled data is stored in the array `myRecording`. Use the `whos` command to look at the size of `myRecording`.

Explain in your lab book why the array `myRecording` is the length that it is.

Now that we have the sampled data in an array, we can plot it:

```
plot(myRecording);
```

We can also do other things to it, as we'll see in subsequent sections.

3. Familiarize yourself with the `audioplayer` documentation in MATLAB. The `audioplayer` object is MATLAB's mechanism for playing audio. While the `audiorecorder` object we created in the last section allowed us to play back what was recorded, it didn't allow us to manipulate the data before playing it back. It also didn't allow us to specify a different sampling rate for playback than the one with which it was recorded.

   The `audioplayer` object takes two parameters upon creation. The first is an array of data samples, and the second is the sampling rate the D/A converter should use to reconstruct the signal. Try the following code:

   ```
   player = audioplayer(myRecording, 96000); % 2nd param plays back at 96,000 samples/s
   playblocking(player); % Actually plays the sound in object player back
   ```

   Now try the two lines of code above, but using playback frequencies of 70000 samples/s and 120000 samples/s. Explain in your lab book why we hear what we hear in the above experiments.

4. Now we're going to manually create an impulse response that passes the audio signal, but also has a delayed echo of the audio signal at a lower amplitude 0.5 seconds later. Let's make our impulse response 2 seconds long. At a sampling rate of 96,000 samples/s, that means our impulse response will have to have 96,000 $\times$ 2 = 192,000 samples. First we'll create an array that is 192,000 samples long of all zeros:

   ```
   impulse_echo = zeros(192000,1);
   ```

   Now let's put a delta at the beginning of the impulse response, and another delta (but a smaller one) after 0.5 seconds. Remember that this is a discrete-time signal in MATLAB, so we're just going to use 1's for our deltas (really a Kronecker delta):

   ```
   impulse_echo(1) = 1; % A delta at the beginning of our impulse response
   impulse_echo(48000) = 0.5; % A smaller (softer) one after 0.5 seconds (48,000 samples)
   ```

   Add a third even softer echo at 1 second, and then play the impulse response out through the speakers using the following code:

   ```
   player = audioplayer(impulse_echo, 96000);
   playblocking(player);
   ```

Describe what you hear in your lab book.

5. Now that we have both a recorded signal (`myRecording`) and an impulse response (`impulse_echo`), let's see what happens to your recorded signal `myRecording` when you pass it through an LTI system with impulse response `impulse_echo`. To do this, we simply convolve `myRecording` with `impulse_echo` using the `conv` command in MATLAB as follows:

   ```
   myRecording_echo = conv(myRecording, impulse_echo); % This will be a bit slow...
   ```

   Now play the filtered signal `myRecording_echo` back:

   ```
   player = audioplayer(myRecording_echo, 96000);
   playblocking(player);
   ```

   You can also try playing it back at different frequencies if you like. Playing it a bit slower will allow you to really hear what is happening. Record in your lab book what you hear, and explain why you hear what you hear.

6. You can download a few different impulse responses (courtesy of the Centre for Digital Music, Queen Mary, University of London) from the Lab Materials section on the Content tab in Learning Suite. These impulse responses are also 2 seconds long and sampled at 96 kHz. One is the acoustic impulse response of the Great Hall at University of London, one of the Octagon (another cool room) at the University of London, and one from a regular classroom at University of London. You can read more about them here: http://isophonics.net/content/room-impulse-response-data-set

   The data is also described in this publication:

   Stewart, Rebecca and Sandler, Mark. "Database of Omnidirectional and B-Format Impulse Responses," in Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP 2010), Dallas, Texas, March 2010.

   The impulse responses are stored as `.wav` files. You can read them into arrays in MATLAB with the following code:

   ```
   impulse_great_hall = audioread('great_hall.wav');
   impulse_octagon = audioread('octagon.wav');
   impulse_classroom = audioread('classroom.wav');
   ```

   Once you have loaded each of these impulse responses into MATLAB, listen to them using an `audioplayer` object. Describe what you hear for each impulse response in your lab book.

   You can use the `subplot` command in MATLAB to create a single plot with three panes showing all three impulse responses as follows:

   ```
   figure;
   subplot(3,1,1);
   plot(impulse_great_hall);
   subplot(3,1,2);
   plot(impulse_octagon);
   subplot(3,1,3);
   plot(impulse_classroom);
   ```

7. We assume you've been writing all your code in a `.m` file (a script). To finish up this task, make a new "section" of your script file. Sections of MATLAB files can be run individually, and often provide a nice way to organize your code. Each "section" begins with a `%%` and at least one space, after which you can add a comment like `Task 1 sign off section`. In this new section of code perform the following steps:

   (a) Have the script prompt the user to record 5 seconds of audio.

(b) Filter the recorded signal by convolving the original signal with each of the impulse responses (the one you created and the three from the `.wav` files).

(c) Play back each of the filtered signals.

Task #1 Pass-Off: Submit your script and documentation from your lab book of its outputs and results for pass off to the TAs via Piazza for this section of the lab.

8. **Extra Credit (10% of one lab assignment, counted only on the Lab portion of your grade). Do NOT attempt this until you have completed all other tasks!:** Try recording an impulse response of your own by producing a short audio "impulse" of some kind and recording the following sound for roughly two seconds. Note that technically, the impulse itself is not part of the impulse response. It is the "input" to the system, while the echos and reverberations are the "output." However, the `.wav` files we used still had the impulse, so it's okay if your impulse response has it too. Brainstorm how to produce the impulse, and feel free to get creative. Get this impulse response into MATLAB, convolve your voice recording with it, and see what you get. You could use any number of methods for recording the sound: a handheld device, a laptop, or you can ask the TAs to check you out one of the small USB audio recorders we have in the lab (although, I'm not really sure where these are, so you might try to use your own technology instead).

Task 2—Aliasing from Undersampling

In this task you will generate simple sinusoidal signals at different frequencies using the signal generator, and then sample them below the Nyquist rate to observe how the signals alias to lower frequencies. Unfortunately (at least for this experiment) the sound cards in your lab computers automatically send any input signal that is being sampled through a high quality, high order anti-aliasing filter prior to sampling. The corner frequency of the anti-aliasing filter is set based on the sampling frequency that is chosen.

For example, in Task 1 you used the `audiorecorder` object in MATLAB to sample an audio signal at a frequency of 96,000 samples/s. That means that the audio signal was **first** passed through an anti-aliasing filter with a corner frequency of less than 48,000 Hz (actually near 35,000 Hz) prior to being sampled. This filtered out any frequencies over 35,000 Hz; those frequencies would have aliased back into the sampled signal if the anti-aliasing filter were not used.

So how can we possibly observe aliasing using the sampling capabilities of the computer's sound card, when it automatically filters out any signals that might alias (given the sampling rate) before it samples? For example, suppose that we want to undersample a tone (sinusoidal signal) at 15 kHz so that we can observe aliasing. The Nyquist rate for this signal is 30,000 samples/s, so we would need to sample below that frequency to observe aliasing. But if we set the sound card to sample below that frequency, the 15 kHz signal will be filtered out before it gets sampled!

The solution is to sample the 15 kHz signal at a higher rate (in our case, we'll sample at 48,000 samples/s) so that it doesn't get filtered out by the anti-aliasing filter in the sound card. Then, after we have the samples in MATLAB, we delete the samples that we don't need to get down to a lower effective sampling rate. For example, if we wanted to see how the 15 kHz signal would alias if we sampled at a rate of 8,000 samples/s (way below the Nyquist rate of 30,000 samples/s), we would simply keep every 6th sample from our MATLAB vector that was sampled at 48,000 samples/s.

The process of throwing out the samples we don't need, to get a lower effective sampling rate is often called "decimation" or "downsampling" of the signal. However, sometimes the signal is digitally low-pass filtered prior to decimation to eliminate aliasing at the new lower sampling frequency, which is not what we want to do. Don't use the MATLAB `decimate` command for this lab, as it actually applies a digital anti-aliasing filter to the higher-sampling-rate data prior to decimating it.

Procedure

1. Suppose that you sampled a 15 kHz signal at a sampling frequency of 8,000 samples/s without using an anti-aliasing filter to band-limit the signal prior to sampling. Assuming you reconstruct the signal with the same sampling frequency of 8,000 samples/s, follow the technique that you used in Example 8.4 from L&G in the prelab to identify the new frequency of the 15 kHz signal after aliasing.

2. Now let's verify that your mathematical answer is correct.

   (a) Create a 15 kHz sinusoidal signal. You can either get this from the function generator, or, if you want to be a bit more adventuresome (and you want to play around with listening to the signal before you sample it), download a dog whistle app to your smartphone. Note that 15 kHz will be above the hearing range of many of you.

   (b) Sample 5 seconds of the signal at 48,000 samples/s using the `audiorecorder` technique you learned in Task 1. **Make sure to set the sampling rate to 48,000 samples/s, and not the 96,000 samples/s we used before.**

   (c) Play the full signal that you recorded back using either of the techniques you learned last week. Can you hear the 15 kHz signal? Some cheap speakers and headphones will not reproduce this frequency. Remember that the 15 kHz signal didn't get filtered out by the sound card's anti-aliasing filter because we sampled at 48,000 Hz, which is well above the Nyquist rate of 30,000 Hz for this signal. **Note:** If you are using an audioplayer object to play this signal back, make sure you set it to reconstruct at 48,000 samples/s!

   (d) Now we are going to simulate sampling at a rate of 8,000 samples/s by only keeping every 6th sample in our sampled signal. Remember from Lab #4 that an 8 kHz sampling frequency is a very common one for simple low-quality audio signal sampling, like in your cell phone. You don't really need to preserve frequencies above about 3 kHz to have perfectly understandable voice recordings, so a sampling frequency of 8 kHz works just fine for simple voice applications. Create a new vector in MATLAB that only keeps every 6th sample of your original sampled signal.

   MATLAB tip: If the original sampled signal is stored in MATLAB vector `my48kHzRecording`, you can create a new vector that just keeps every 6th sample with the following code:

   ```
   my8kHzRecording = my48kHzRecording(1:6:length(my48kHzRecording));
   ```

   (e) Now play back your undersampled version. This time, make sure you set the `audioplayer` object to use a reconstruction frequency of 8,000 samples/s! Otherwise, your results won't make sense.

   Describe what you hear in your lab book. Does the result match what you predicted in the previous section? Let's plot the spectrum of your signal to see whether it really ended up where you predicted. There are lots of ways to do this, but this code will suffice for today:

   ```
   fs = 8000; % sampling frequency
   ww = -pi:0.0001:pi; % identify which frequencies you'd like to test
   XX = freqz(my8kHzRecording,1,ww); % calculate the frequency response
   ff = ww*fs/(2*pi); % convert the frequency vector to Hz
   figure
   plot(ff,(20*log10(abs(XX)))); % plot the magnitude response in dB
   ```

3. Repeat your experiment from step 2, but this time use a tone at 14 kHz instead of 15 kHz. In your lab book, predict the frequency you will hear after undersampling and reconstruction. Does the tone you hear match your prediction? Plot the magnitude response as before to make sure.

4. Repeat the step 2 experiment again, but instead of leaving the tone at 15 kHz, sweep it down in frequency from 20 kHz to 100 Hz during the 5 seconds (or longer) recording interval. If you are clever you can set up this sweep to run automatically on the function generator, or you can manually tune the frequency with the control knob in steps of 100 Hz.

   Describe in your lab book what you hear when playing back the full sample rate recording, and the decimated recording. What is unusual about it? Is this what you expected, or should have expected?

5. Now that you've demonstrated aliasing due to undersampling on a very simple tone, let's try it on something more complicated. Try recording 5 seconds of music from an MP3 player in two different ways:

   (a) Sample the music at 48,000 samples/s, preserve only every **6th sample**, and play it.

   (b) Now **actually** directly sample a 5 second clip of the music, setting the sampling frequency of the sound card to 8,000 samples/s, and play it back.

   Describe in your lab notebook any differences that you hear in the two cases. Neither one of these audio signals will sound perfect, since we are sampling them at such a low frequency and thus only preserving frequencies below 4 kHz. However, one of the signals will be corrupted even more by aliasing. Describe with sketches in the frequency domain what is happening in each case.

6. The two-stage filter that we designed in Lab #4 can function very well as an anti-aliasing filter for a sampling rate of 8 kHz. Rebuild it if you don't still have it on hand.

   (a) Repeat step (2) above, but now run the 15 kHz signal through your own anti-aliasing filter before feeding it into the sound card for sampling at 48 kHz. See if your anti-aliasing filter eliminates the aliased tone you heard! Repeat step 4 by recording the sweep signal through your anti alias filter. What changed?

   (b) Using your MATLAB plot from Lab #4 of the designed frequency response for your 4th order filter, find the expected attenuation of the part of the signal that would alias to 3 kHz when sampling at 8 ksamp/s. First find the higher frequency that would alias to 3 kHz by the sampling. Then from your magnitude response plot, determine how much this frequency would be attenuated (in dB) by the anti-aliasing filter prior to sampling.

Task #2 Pass-Off: Submit your scripts and documentation from your lab book of its outputs and results for pass off to the TAs via Piazza for this section of the lab.

Before turning in your lab report, complete all lab entries thoroughly and include all required work as stated in the beginning of this lab assignment (2nd paragraph).