ECE 345: Linear Systems and Signals — Fall 2020
Professors Sarwate and El Rouayheb — Due: December 17, 2020

# Lab 4

This lab is made up of two mini-labs. See the Canvas page for the lab for details and for file downloads. To turn in this lab, you need to submit a `.zip` file containing the following three files:

- a completed lab report **in PDF format**.
- your code for part 1 of the minilab
- your code for part 2 of the minilab

Make sure your file names are labeled so that we can easily find your material.

Points will be taken off for non-PDF submissions. Why are we being sticklers about formatting? In the real world, your work will also have to fit formatting guidelines. If you bid on a government contract, for example, and don't follow the guidelines, your bid/proposal could be rejected without review. Get in the habit of following the formatting rules!

Along those lines, do not forget to label your plots – it's part of the rubric!

Please make sure Signal Processing Toolbox and DSP System Toolbox are installed in your Matlab. Also, to use the stft() function, the version of Matlab should be at least R2019a.

## 1 Mini-lab 1: Sampling above the Nyquist rate

This exercise illustrates why we need to sample above the Nyquist rate.

(a) Synthesize the following signal for $f_c = 440$ Hz:

$$x(t) = \cos(2\pi f_c t) + \cos(2\pi(1.5f_c)t) + \cos(2\pi(2f_c)t). \tag{1}$$

use a simulation sampling rate of $f_0 = 44000$ Hz. What is the Nyquist rate for this signal? Play the signal using `soundsc`.

(b) We can use the DTFT to approximate the CTFT and the FFT to approximate the DTFT. But to do this we need to be able to represent the frequency axis correctly. MATLAB plots the DTFT for $\omega \in [0, 2\pi]$ which is not the way we have been doing it in class. To compute the DTFT in MATLAB you can use the `fft` and `fftshift` functions.

Use a $N = 2^{20}$ point FFT to approximate the CTFT of x(t). Plot the magnitude as a function of the frequency $f$ in Hz. Note that using `fftshift` the frequencies in the N-point FFT correspond to N evenly-spaced frequencies from $-f_0/2$ to $f_0/2$.

(c) Sample the signal at half the Nyquist rate. You can use the `upsample` and `downsample` functions in MATLAB to create a signal which keeps samples of x(t) at a rate of 880 Hz and inserts an appropriate number of zeros so that the simulation sampling rate is still $f_0$. Plot the original signal from $t = 0$ to $t = 0.05$ using `plot` and the sampled signal using `stem` on the same plot (use `hold on`).

Play your sampled signal using `soundsc` with the sampling rate $f_0$. Can you hear the original signal? How does the plot show that you should not expect to hear the original signal?

(d) Plot the magnitude of the DTFT of the signal from (c) using `fft` and `fftshift`. Note that now your frequency axis should go from $-\pi$ to $\pi$.

(e) Now sample the signal at K = 10 times the Nyquist rate. Play your downsampled signal using `soundsc` with the sampling rate $f_0$. Can you hear the original signal? Plot the DTFT of the resulting downsampled signal. What do you see?

(f) Use `lowpass` to filter the signal from the previous part with cutoff equal to twice the Nyquist rate. Play the resulting signal. How does it compare to the original signal from part (a)?

(g) Repeat the previous two parts with K = 5. What differences do you see and hear?

## 2 Mini-lab 2: Audio manipulation and filtering

This lab uses signal processing techniques to clean up an audio file of someone speaking and (hopefully) make it possible to understand what they are saying.

The original signal $x(t)$ has been affected by three different things. First, there is a reverberation which is formed by convolving $x(t)$ with a decaying periodic series of impulse functions:

$$h(t) = \delta(t) + r\delta(t - D) + r^2\delta(t - 2D) + \ldots = \sum_{k=0}^{\infty} r^k \delta(t - kD). \tag{2}$$

Second, there is a sinusoid that has been added to the signal:

$$s(t) = A\cos(2\pi f_i t) \tag{3}$$

Finally, there is highpass white noise $w(t)$ which has been added to the signal. Overall, the signal is

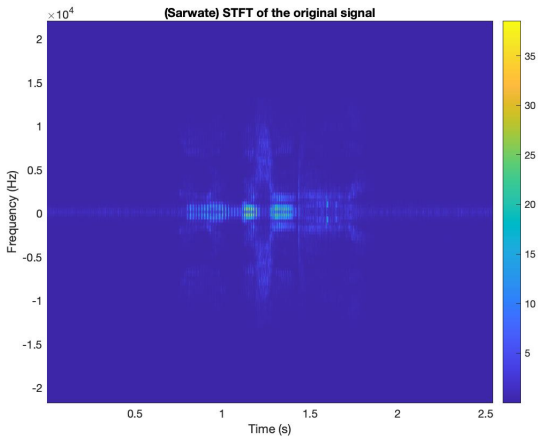
$$z(t) = (x * h)(t) + s(t) + w(t). \tag{4}$$

The Short Time Fourier Transform (STFT) slides a short window across the signal and computes the Fourier transform of the windowed signal at each step. The result is an *image*: the $m$-th vertical slice is the Fourier transform of $z(t)v(t - m\Delta)$, where $v(t - m\Delta)$ is the window centered at $m\Delta$. You can compute and display the STFT of `z` using this code block:

Listing 1: taking the STFT

```
[S,F,T] = stft(z,fs);
pcolor(T,F, abs(S));
title('stft of signal after notch filter')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
shading flat
colorbar
```

The corrupted file is in `mysteryclip.wav`. This is a *spectrogram* of the original signal:




(a) Read in the file `mysteryclip.wav`, extract the left channel, and listen to the clip. Can you tell what is being said? This is our $z(t)$.

(b) First it might help to visualize the signal. Compute the STFT and save the resulting image as a `.jpg`.

(c) Let's get rid of the reverb first. Look at SSTA 8-4.3 and design a DT filter to remove the reverb of the signal. Here $D = 0.1$ seconds and $r = 0.7$. Use the `filter` function to filter $z(t)$. Plot the STFT and save it as a `.jpg`.

(d) The next step is to remove that annoying cosine tone. We can do this using a *notch* filter via the `iirnotch` function. To use this function on a CT signal in Hertz, you need to normalize the frequency. In particular, if you want a notch at $f_0$ Hertz, you need to provide an input $w_0 = \frac{f_0}{f_s/2}$.

First estimate the what frequency you need for your notch by looking at the STFT of the signal. As a hint, the frequency $f_i$ of the cosine is a multiple of 100 Hz.

Design a notch filter using `iirnotch` and use `filter` to remove the tone. Plot the STFT of the result and save it as a `.jpg`.

(e) To get rid of the highpass noise, we need to use a lowpass filter. Use the `lowpass` function to filter out the highpass noise and listen to the result. Adjust your cutoff for the lowpass filter appropriately to make the audio signal understandable. Plot the STFT of the result and save it as a `.jpg`.

(f) What is the person saying in the clip?