ESE 150 – Lab 06: Perceptual Coding

LAB 06

In this 2-week lab we will do the following:

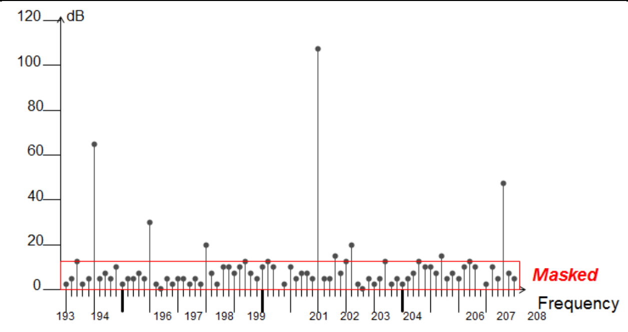
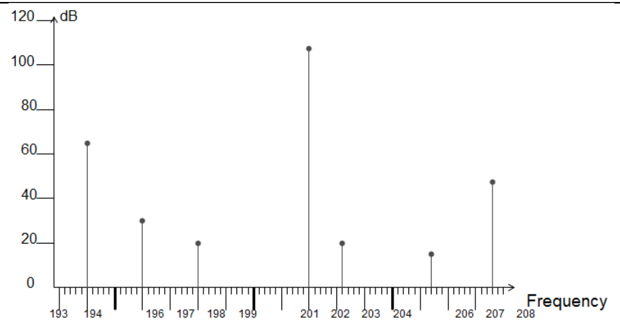
1. Use Matlab to sample three audio files in .WAV format (2/26)
2. Plot the samples in time and frequency domain before any compression (2/26)
3. Analyze the DFT of the samples and drop masked signals
    a. Manually (2/26)
    b. Write a function to drop masked signals (3/12)
4. Perform Huffman Encoding before and after dropping samples (3/12)
5. Analyze efficiency of your compression algorithm (3/12)

You will work in the same teams for the entire lab (both weeks).

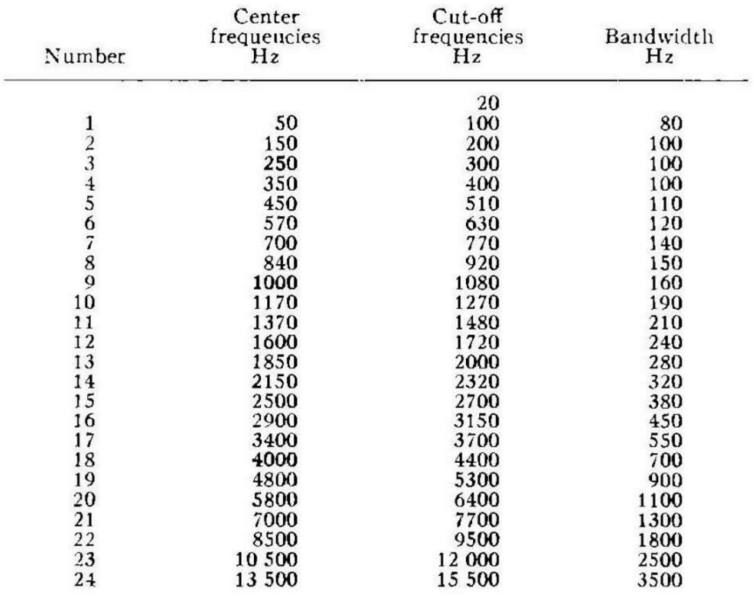
Background:

In this lab, you will apply perceptual coding. Recall that perceptual coding is using your knowledge of psychoacoustics (how human's interpret sound) to encode your audio data in a more succinct way. In lab 5 you performed some classic psychoacoustic experiments. One in particular is frequency masking.

Recall that frequency masking is when one tone is so loud that it "drowns out" other tones of the same or nearby frequencies? The pictures below show an example of that happening. Notice that some tones are so loud that other tones that are not as loud are "masked." In the picture on the right, we've used that principal to "drop" or zero-out certain tones. Since they are "masked" no one will be able to hear them (thanks to psychoacoustics), so instead of storing these tones, we drop them (or zero-them-out) to make it so there is less data to store. It's a form of lossy compression that is used in the MP3 algorithm.

Dropping Samples using psychoacoustic analysis	
	
BEFORE psychoacoustic analysis is performed	AFTER psychoacoustic analysis is performed

Remember also that masking can be different for different frequency bins? Recall the table below that shows the 24 critical band frequency bins human hearing is divided into.



Number	Center frequencies Hz	Cut-off frequencies Hz	Bandwidth Hz
		20	
1	50	100	80
2	150	200	100
3	250	300	100
4	350	400	100
5	450	510	110
6	570	630	120
7	700	770	140
8	840	920	150
9	1000	1080	160
10	1170	1270	190
11	1370	1480	210
12	1600	1720	240
13	1850	2000	280
14	2150	2320	320
15	2500	2700	380
16	2900	3150	450
17	3400	3700	550
18	4000	4400	700
19	4800	5300	900
20	5800	6400	1100
21	7000	7700	1300
22	8500	9500	1800
23	10 500	12 000	2500
24	13 500	15 500	3500

Today in lab, you'll take the time-sampled audio and go through it frequency-bin by frequency-bin and perform your own psychoacoustic analysis. You will look for signals that are masked. If they are masked, you will drop them (or zero-them-out). Then you'll convert your frequency domain data back to the time domain and listen to your modified audio file to see if you can notice the difference! The more tones you drop due to masking, the smaller your audio file will become.

Once you've completed the task of dropping masked tones, you'll run the Huffman encoding algorithm on it and see just how much "redundancy" is now in your audio file that can be reduced with standard compression; further reducing your audio sample's file size. By applying a lossy and lossless compression scheme to your original audio file, you'll have gone through a very similar process as to what is done in an MP3 file! Today we'll just concentrate on frequency masking, but other tricks (like temporal masking) are used in the true MP3 algorithm. But this will give us a good sense of what MP3 is doing and how it achieves the compression ratios that it boasts!

Prelab

- Read through lab
- Questions:
    - Thoroughly explain why and how zeroing out (setting values to 0) frequencies in a signal will lead to compression during Huffman encoding?
    - Particularly, if we zero out 50% of the frequencies, what will happen to the efficiency of the compression? How about 90%?
    - Finally, is there a better alternative to zeroing out values? Why would the alternative method result in better compression rates?
- Bring headphones to lab

Lab Procedure – Lab Section 1 – Using Matlab to Sample .WAV files

- In this section of the lab, we'll import a .WAV files directly into Matlab! This will give you "perfect" PCM audio data that you can then perform a psychoacoustic analysis on.
    - a .WAV file contains time-sampled audio similar to what you collected in lab on the arduino in previous weeks
- We will plot the time and frequency representations of the sound in the .WAV file. This section uses 'song 1' as a reference to verify your plot_time() and plot_dft() functions.
- This section serves as preparation for creating the basic functions needed in the following sections.

1. Download the 3 .WAV files from last lab and put them in a directory where you are running Matlab:
    a. For example, create a directory called: c:\temp\lab6
        i. If you can't create that folder on our lab computers, you can choose any other location you'd like, as long as you know exactly where it is!
    b. Download the .WAV files into c:\temp\lab6
    c. Start Matlab
    d. In Matlab's command window, type:
    cd c:\temp\lab6
2. Import the .WAV files directly into matlab:
    a. In Matlab's command window type:
    [samples_time, samp_rate]=audioread('song1_300-600Hz.wav')
    b. This has imported the PCM quality .WAV file into a matrix called: samples_time
    c. It has also determined the sampling rate that the .WAV file was created with
    d. Look carefully at the value of "samp_rate", is it what you expect for a .WAV file?
3. Plot in the time domain:
    a. The function "audioread" brings the samples into Matlab
    b. Create a function "plot_time":

    ```
    function [samples_time] = plot_time (samples_time, samp_rate)
    ```

    This should
        i. Plot your converted samples in the time-domain
        ii. Return the samples in voltage form with a range of 0 to 5 volts.
    c. Then, call your function: plot_time() and have it plot your imported .WAV file (submit this labeled plot)
4. In Lab 4 you created a function called "plot_dft". Modify it a bit to have the following arguments and returns:

    ```
    function [samples_freq] = plot_dft (samples_time, samp_rate)
    ```

    a. Input: samples_time – should be the output of your "plot_time" function

   b. Input: samp_rate – should take in the sampling rate: e.g.: 10000
   c. Output: samples_freq – should be the unmodified output of the fft() function
      i. Recall, you take the absolute value of it, don't return that version! That's only the magnitude, not the phase information.
   d. The function itself should plot only the valid frequencies: ½ samp rate
   e. Important note: Use the stem() function instead of plot(). The stem() function plots the discrete sequence of your input as stems that extend from the x-axis. The inputs of the stem() function are identical to those of the plot() function.
5. Plot in the frequency domain:
   a. Use your "plot_dft()" function:
      ```
      samples_freq = plot_dft (samples_time, samp_rate)
      ```
   b. If all your functions are working properly, do you see the two spikes you expect?
   c. Modify your function to also return the corresponding magnitudes to sample_freq. Make sure the indices of sample_freq and magnitudes are perfectly matched.
6. Listen to the audio just to make sure…
   a. Plug your headphones into the lab PC
   b. Use Matlab to perform a D2A on the 10-bit data! Type the following:
      ```
      soundsc(samples_time, samp_rate)
      ```
      Put your headphone on and listen! Does it sound right??
   c. **Important note**: You can input either the raw signal or your voltage output from Step 3. The soundsc() function is programmed to auto-format the input with values from -1 to 1. As a result, the function can take in your resulting voltage output from Step 3. Also, the raw sound files are already formatted with values from -1 to 1.
   d. Verify that this step is working before proceeding
7. Turn both of these graphs in with your report!
   a. Remember to only show a few cycles in the time domain (zoom in!)
   b. Place data labels on the frequency spikes in the frequency domain plot
   c. Make sure the 2 functions are correct before continuing.

Lab Section 2 – Performing a psychoacoustic analysis

- In this section you'll actually examine the data in the frequency domain and look for samples to drop!
- For this section, the removal of frequencies from certain sections of the song will be done manually. This will serve as a basis for writing the actual algorithm in the next section.

1. In Section 1, you imported "song1" which is a test file to make sure your plot_dft() function is working
2. Use the steps from Section 1 to import "song2" into Matlab. Use your plot_dft() function to perform a dft on the entirety of song 2. This will serve as a visual to show what frequencies are present in the signal and the corresponding magnitudes. Plot the frequencies and listen to the audio before continuing
    a. You'll also need to save this plot for your lab submission
3. In this step, you will be performing psychoacoustic analysis on a smaller section of song 2.
    a. Use code to isolate a consecutive sequence of 500,000 samples starting from the 100,000th sample of song 2. What is the duration for the 500,000 samples?
        i. Call this extracted sequence `samples_time_extract`
    b. Use your plot_dft() function to obtain a frequency plot for the 500,000 samples.
        i. Save this plot for your lab submission.
    c. Do you see any masking opportunities?
        i. Zoom in on the 2nd critical band. Refer to the critical band frequency bin chart in the introduction section to determine the lower and upper frequencies for the 2nd critical band.
        ii. Are there any frequencies that can be dropped from the 2nd critical band?
    d. Remember, you are looking at the "magnitude" or absolute value of the frequency domain data
        i. BUT, recall that your function: plot_dft() actually returns the complete frequency domain data
    e. For tones that are being masked, zero-out those samples in the matrix that was returned by your plot_dft() function (meaning, set them to zero). Use the magnitudes to determine a threshold for dropping frequencies. Try to drop 50% of the samples.
    f. You've done this for just the 2nd frequency bin. Repeat parts c-e for critical bands 3-8.
    g. Let's see if deletion affected the quality of the sound.
4. Convert your modified frequency-domain data back to the time domain
    a. As you'll recall, there is an "inverse" DFT function in mathematics and in Matlab as well
    b. Assuming you've made a matrix called: `samples_freq_extracted_dropped` that contains the frequency domain data, but with the samples that were masked set to zero, convert it back to the time-domain by typing:

       ```
       samples_time_dropped = ifft(samples_freq_extracted_dropped)
       ```
    c. What you'll get back in "samples_time_dropped" will be your audio file but back in the time domain. Matlab has performed the inverse DFT for you
    d. Play this back the two sets of samples (i.e. before and after dropping; `samples_time_extracted, samples_time_dropped`), can you hear a difference? Did the dropping of the masked signals change anything?

e. Show your TA your plot showing the tones dropped and your before and after frequency samples and answer some questions. This is your Exit Checkoff for February 26th lab. Time permitting, continue working on Section 3.

Lab Section 3 - Automating psychoacoustic analysis

- In the previous section, you only dropped frequencies from a small selection of critical bands for a small portion of the sound wave. The goal of this section is to drop frequencies from all bandwidths for the entirety of the song.

1. Remove masked frequencies as in Section 2 but in an automated way.
    a. Visually inspecting the fft() created matrix is going to be laborious
    b. You can automate this process in Matlab
    c. You will need to create a separate Matlab function that contains the logic in your plot_dft() function and analyzes the transformed signal for masking, frequency-bin by frequency bin
    d. Call this function "drop_samples" and give it the following ins/outs

```
   function [samples_time_dropped] = drop_samples ( samples_time,
window_size )
```

    e. Here's an overview of the logic:
        i. The input variable 'window_size' will control how many samples are allocated to each window. We are using the 'window' variable to isolate smaller sections of the signal, similar to Section 2.
        ii. Use the logic from your plot_dft() function to obtain the fourier transform of each windowed section.
        iii. For each transformed section of the song, go through all 24 bandwidths to delete masked frequencies based on a set threshold and the magnitudes.
        iv. Have your function also print out the compression ratio:
            1. Calculate your "compression" ratio using the initial number of samples and how many samples you will be keeping. Include this in your report.
        v. Think of an efficient way to store the modified windowed sections so you can later take the inverse fourier transform.
        vi. Play back the modified sound as in Section 2. Can you hear a difference? Did the dropping of the masked signals change anything?
        vii. For Trial 1, set window as 1024 samples and a threshold to remove 50% of the frequencies. For Trial 2, change the window to 2048 samples and a threshold to remove 50% of the frequencies. For Trial 3, set window as 1024 and set a threshold to remove 70% of the frequencies. For Trial 4, set window as 1024 and set a threshold to remove 90% of the frequencies.
        viii. Continue to experiment with different window sizes and thresholds. Try to obtain as much compression as possible while maintaining sound quality.
    f. This will take your time. You'll have to think through the code on your own, and you want to spend some time experimenting with code, the window size, the thresholds, and perhaps other parameters you identify.
    (This is why we're giving you the time between labs and a second lab session to do this.)

2. Once you are done, apply your “drop_samples” function to song3, report compression ratios, and listen to the original and masked versions.
    a. Remember to include all necessary screenshots

Post Lab Questions:

1. Why is it inaccurate to remove frequencies directly from the fourier of the entire song? For example, why is it wrong to take the fourier transform of the entire song and remove the 500,000 frequencies with the lowest magnitudes? (500,000 was arbitrarily chosen)
2. Provide 2 qualitative examples to why the method mentioned in the previous question is incorrect. In other words, what situations in a song could easily highlight the inaccuracy of the previous method?
3. How did changing the window size affect the resulting quality of ‘sampled_time_dropped’ (Trial 1 vs Trial 2)? Changing the threshold (Trial 1 vs Trials 3 & 4)?
4. Compare the time & frequency plots of your original sound wave and modified sound wave.

Lab Section 4 – Using Huffman Encoding

- In this section of the lab, you'll use your Huffman function from your previous labs to compare the compression ratios before and after dropping samples.

1. Look at the original files and determine how many samples are in the .WAV files. Include this in a table for your report.
2. Run your Huffman code on the ORIGINAL time-domain samples
   a. Save the matrix as a ".MAT" file
3. Next, run your Huffman code on the dropped time-domain samples
   a. Save this output as a ".MAT" file
4. How much better did you do?
5. Calculate the compression ratio for before and after dropping the samples. What do you notice? Has the ratio improved after dropping samples? Is this as expected?
   a. Show your calculations in the lab report.
6. Produce a table

Song	Original Size	# frequencies dropped	Huffman compression Ratio before dropping	Huffman compression Ratio after dropping

HOW TO TURN IN THE LAB

- There will be no weekly lab writeup for either lab session.
- Include answers to post lab questions in Section 3 and table in Section 4.
- Be sure to include all necessary plots.
- Each student will produce an individual formal lab report that is due Sunday, March 18th.
- See course web page for Formal Lab Report specification and expectations.
- See the specific rubric on campus for the formal writeup for this lab.
- Upload a PDF document to canvas containing for formal lab report.