9.35 Sensation And Perception

Spring 2009

9.35 Spring 2009
Problem Set 6 – Sound and Hearing
100 Points

Note: This problem set will be due in two stages. Problem 1 (building the spectrogram) will be due first (simply submit your m-file), and a correct spectrogram function will be released immediately for you to use for the rest of the pset (should your function not be complete). The remainder of the pset will be due on May 8th, which is the last day allowable by institute policy. Due to these constraints we will be unable to accept any late submissions.

NB: The most recent version(s) of Matlab may be able to read a number of different audio file formats using mmread, but up to version 7.4, Matlab can only read WAV files (with the function wavread). If your version cannot read a file, you should convert it to WAV using any method of your choice. For instance, QuicktimePro can convert between formats, or there may be a free command line utility on Linux or OSX computers. A number of free converter utilities are also available. I successfully used the program 'Audio/Video to EXE' from Download.com

Audio files in Matlab are nothing more than a vector (or matrix with 2 columns) that represent sound pressure levels over time. If your sound is stereo, it will have 2 columns, and you should take the average to convert it to mono.

We recommend not using headphones for this problem set, in case it gets too loud!

1) Write a function to display the spectrogram of a sound. **MAKE SURE** that you get this part right, since the rest of this problem set will depend on it! To check your work, you can use to function spectrogram, but it won't do everything for you.

   a. (10 Points) A simple spectrogram is just a plot of the Fourier amplitudes over time. To do this, you can take a 'window' of sound by simply extracting a temporal subset of the sound signal, computing the FFT, storing the output in a matrix, and proceeding to the next window in time. The pseudocode might look something like this (where a timestep is set as ½ the window size):

```
Spct = zeros(#frequencies, #timesteps)
for every timestep t in mysound
   Spct(:, t) = abs(fft(mysound( (t-win/2):(t+win/2) )))
end
imagesc(Spct)
```

But, there are several important things you must consider. First, the window size is directly related to the frequencies that will be represented in the spectrogram. Assuming that human hearing drops off at 20 Hz, what is the equation that will give you window size, dependent on the sampling rate of your sound, so that your window captures a full cycle at 20 Hz? What is the minimum sampling rate that will accurately detect sound at 20 kHz, the high limit of human hearing?

b. (20 Points) The second point to note is that the output of the fft is not in the desired format. For instance, recall that FFT returns both positive and negative phase components, so be sure to crop the negatives out of Spectrogram. Furthermore, indexing in the Spct matrix created in the pseudocode above will not correspond to temporal frequencies (e.g., Hz = cycles/second), but instead to window-frequencies (cycles/window). You will thus need to translate indices into Hz, and crop from the spectrogram all power below 20 Hz and all power above 20 kHz. (Note that displaying frequencies linearly between 20 and 20K Hz will obscure a lot of the detail at low frequencies, so you can crop the spectrum to contain just 20 Hz to 5 kHz if you wish). You may also want to scale the output logarithmically (i.e., imagesc(log(Spct))) so you can better see strong and weak frequency components together.

Implement the function mySpectrogram(y, Fs) for a mono- sound vector y and sampling rate Fs, using the window size that allows you to estimate power at frequencies as low as 20 Hz. Make sure that the spectrogram accurately labels time on the x-axis and frequency on the y-axis. Also use the command colorbar to indicate the absolute magnitudes. You can test your function on a sound of your choice, or simply add a few sine waves together to make sure the power shows up in the correct place.

2) Several properties of sound have similar counterparts in vision, but they behave in many different ways. For instance:

a. (10 Points) In both vision and audition, energy is generated by some source, and is then modified by the environment before it hits our sensory organs. In vision, one simple part of this interaction is expressed as L=I*R, and we primarily care about R (the reflectance of objects in the world, which modifies the energy coming from the light source). What are the counterparts to 'illumination' and ‘reflectance’ for acoustic signals? Is the counterpart to 'reflectance' in audition still the most important (perceptually)? Why or why not?

b. (10 Points) What is the perceptual difference between the wavelength of light and the wavelength of sound? Does sound have 'color,' and does our auditory system have anything like 'trichromaticity?' Are there any sound 'metamers?'

3) Download a sample of your favorite musical instrument from the webpage:

http://theremin.music.uiowa.edu/MIS.html

(5 Points) Also, download the Balloon Pop sample. **Plot the spectrograms for both**. You can play them in Matlab with the commands sound or soundsc (make sure to send the right Fs, which you get when you load the file with wavread).

a. (10 Points) Discuss how the spectrogram describes the sounds you hear, and how it differentiates between the two sounds.

b. (10 Points) What aspects of the sounds do the spectrograms *not* capture? If you have two spectrograms that appear identical, are the sounds necessarily the same?

c. (15 Points) You can 'build' a synthetic sound by creating a vector of zeros, and adding sine waves to it of the appropriate amplitudes and frequencies, then playing back the vector using sound. The spectrograms should give you all the information you need to do this. Try to replicate the sound of your favorite instrument using just a handful of sine wave components (no more than, say, 10). How many components to you need? How good is the recreation, especially as you take away components?

   Plot the spectrogram of several seconds of your best synthetic sound, next to the spectrogram of the real instrument.

   Note that since we are only adding sine waves in the range of 20 Hz to 20 kHz, we can't recreate the temporal aspects of an instrument (e.g., attack, decay, etc., without a LOT more manual work!), so this will work best for instruments with long, steady sounds (like a flute or trumpet) but not so well for instruments like drums. Constant regions in the spectrogram will indicate a good sound to replicate.

4) (5 Points) Choose your favorite song, or a wicked guitar solo, and plot the spectrogram for the best ~10 seconds of it. Describe how the spectrogram relates to the music. (Make sure it's not so long that when you submit, it all looks scrunched up in your pdf!)

5) (5 Points) Examine the provided sounds e1.wav and s1.wav. Which one is better, and most importantly, why?