Problem 1a: Intro / practice on working with sound files

Pertinent readings for Problem #1a:

The Pokemon theme song:

My recording was taken from an actual recording session by the original guy who sang the title song ! Go to:

https://www.youtube.com/watch?v=fCkeLBGSINs

A brief introduction to vocals and spectrograms:

This is a great webpage !! I would definitely take a look at this first before you tackle the readings in Cohen & Rangayyan below:

http://vocped.ianhowell.net/a-spectrogram-primer-for-singers/

Our main goals: Create spectrograms and do some easy filtering opertaions !

The first of our 2 main objectives for this problem is to generate the “non-overlapping” spectrogram that you see in Figure 1 (and you should check out the fully-annotated, zoomed-in version of the same spectrogram on Blackboard before you start).

$Sound\ file$: $Pokeman.wav$

$Total\ duration\ of\ sound$: $132\ seconds$

$Data\ point\ density$: $f_{sample} = 44{,}100\ Hz$ $\left(\begin{array}{c} This\ means\ 1\ second\ worth\ of\ sound \\ is\ equal\ to\ 44{,}100\ data\ points\ !! \end{array}\right)$

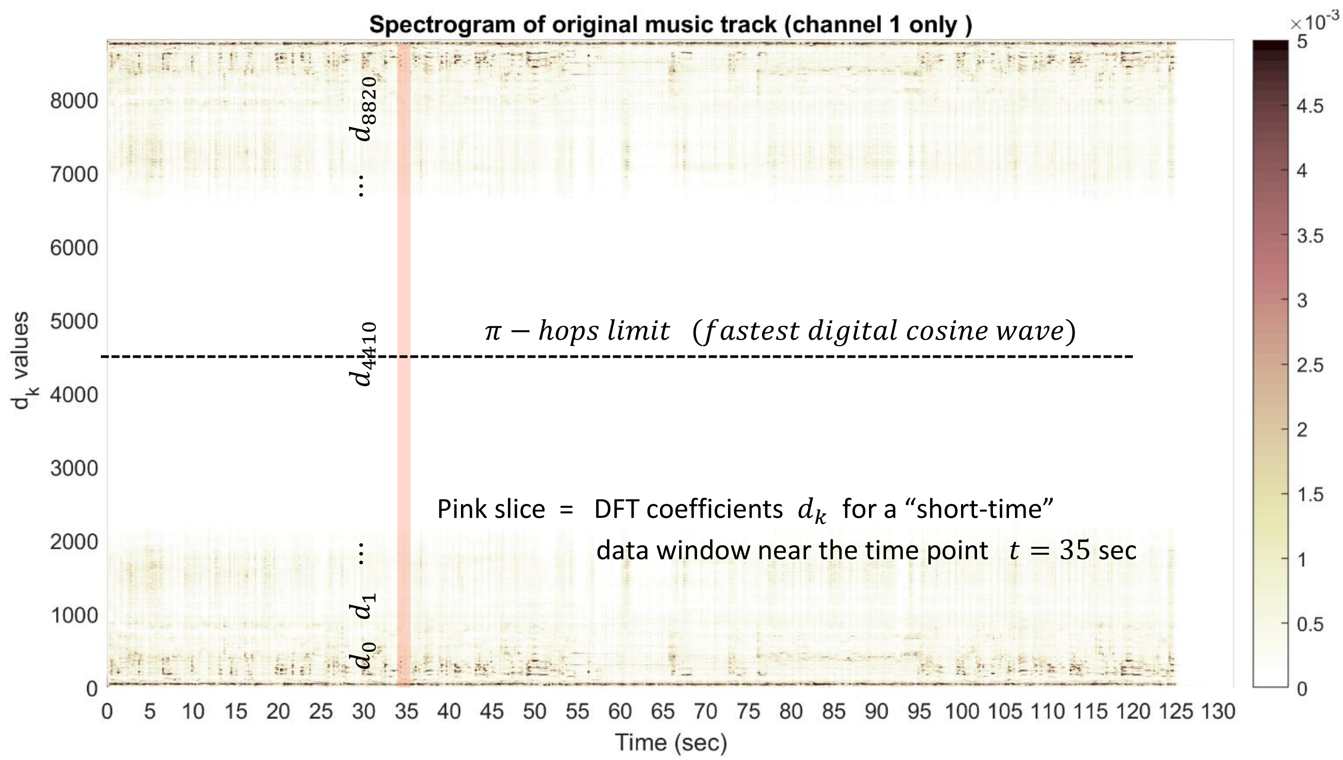


Figure 1: A spectrogram of *pokemon.wav*, where the frequency content (vertical axis) within lots of “short-time” window slices our data are plotted agains the time point of interest.

Before you code your own spectrograms, we first need to go over how matlab read + write audio files. Let's do a quick exploration !

Loading audio files:

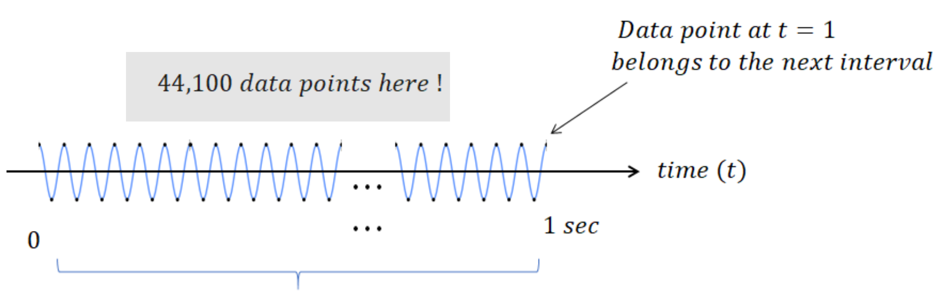
1. Load the audio file *Pokemon.wav* using:

```
% -- Define sound file name
input_filename = {   'Pokemon.wav'    };

% -- Read the sound file…  The "char" command converts the string so that matlab can use it to find the files
%
%    x  =  The audio data  (intensity values0
%    Fs =  The sampling frequency  (f_sample)
[x, Fs] = audioread(char(input_filename));
```

$$Note:\ The\ industry\ default\ is:\quad \boxed{F_s \;=\; f_{sample} \;=\; 44.1\ kHz} \;=\; \frac{44{,}100\ \ data\ points}{1\ second\ interval}$$

Let's think why this is the industry standard !



$The\ fastest, non-aliased\ cosine\ audio\ signal$
$you\ are\ able\ to\ sample\ correctly\ is:$

$$\begin{aligned} cos\big(2\pi\ f_{pi-hops}\big)\ , \qquad where\ f_{pi-hops} &= \frac{1}{2}\ as\ fast\ as\ f_{sample} = 44100\ Hz \\ &= \frac{1}{2}\, f_{sample} \\ &= 22100\ Hz \end{aligned}$$

$(at\ the\ threshold\ of\ human\ hearing\,!)$

Figure 2: The industry default of $f_{sample} = 44.1\ kHz$ has to do with the fact that the limit of human hearing is at around 20 kHz ! The fact that you need $f_{sample} = 2 * f_{pi-hops}$ for non-aliasing sampling runs is called the *Nyquist criterion !*

Separate the left / right channels before data processing:

Most audio files have 2 columns of matching data: One for the left-speaker, and the other for the right speaker ! Therefore, your variable x actually contains 2 column vectors:

$$x = \left[\begin{array}{c|c} \begin{matrix} channel\ 1 \\ data \\ \\ (left\ speaker) \end{matrix} & \begin{matrix} channel\ 2 \\ data \\ \\ (right\ speaker) \end{matrix} \end{array} \right]$$

2. Separate the 2 channels by doing something like this:

```
ch1_raw  =  x(:,1);
ch2_raw  =  x(:,2);
```

Simple practice on editing channels 1 + 2:

After separating the channels, you are usually ready to perform tasks such as:

a) Windowing of data, and then
b) Performing DFTs on your window data
c) Storing your DFT d_k coefficients for that windowed data in a matrix

But here, let's just practice on a simple manipulation: You are gonna make the audio silent from $t = 1\ to\ \ 2\ seconds$ by setting those data points equal to zero ! A schematic of this task can be seen in Figure 3:

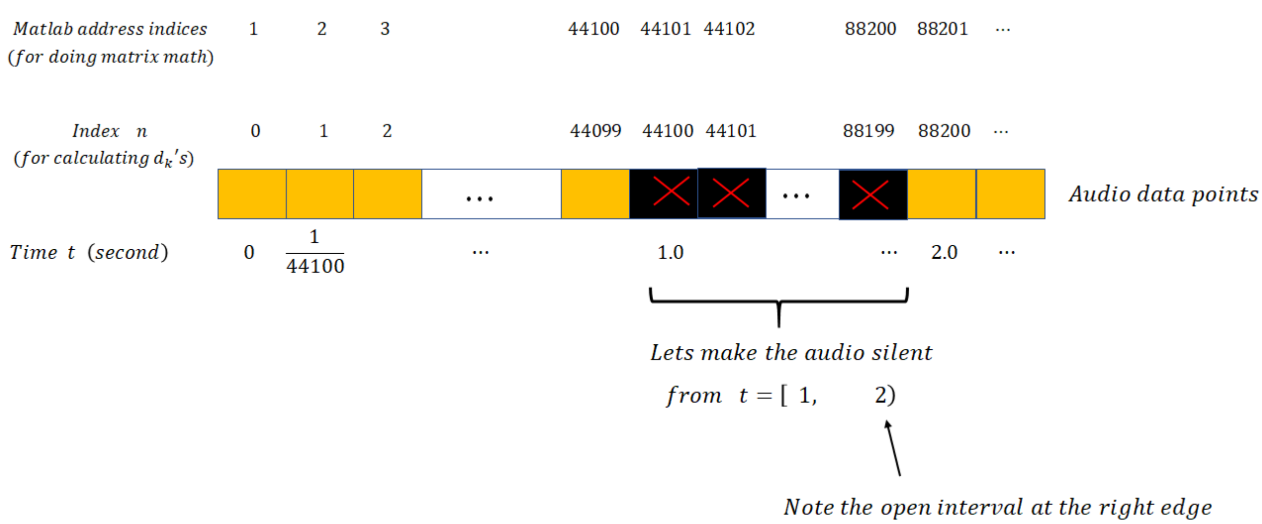


Figure 3: The math you have to keep track of when you're working with a long audio data stream.

3. Silence both channels between t = [1, 2)

(Remember: To do matrix math, we need to think about matlab indices!)

```
ch1_raw(44101 : 1 : 88200)  =  0;
ch2_raw(44101 : 1 : 88200)  =  0;
```

Write our data to an output audio file   /   play your music !

Once you're done processing both channels 1 and 2, you can store both of them as column vectors in a giant matrix called "y." Then, we will use the *audiowrite* function to make "y" into a *wav* file.

4. Package channels 1 and 2 into "y," and write the result into a *wav* file:

```
y = [ch1_raw  ch2_raw];
audiowrite('Pokemon_silent_1to2sec.wav',  y, Fs)
```

Open *Pokemon_silent_1to2sec.wav* using your Windows Media Player (or whatever software you use for listening to music), and you'll be able to hear a 1-second "gap of silence" right after the first word "challenge" has been sung !!   =)

Ok - I think now you're ready to tackle *Pokemon.wav* for real !!   =)