# Lab 1 - "Playing with audio slices"

Friday, June 19, 2009
11:37 PM

## PURPOSE

This lab will introduce you to the practice of analyzing, segmenting, feature extracting, and applying basic classifications to audio files. Our future labs will build upon this essential work - but will use more sophisticated training sets, features, and classifiers.

We'll first need to setup some additional Matlab folders, toolboxes, and scripts that we'll use later.

## DIRECTORY

**Go to the folder: /usr/ccrma/courses/mir2009**

## SETUP

1. Launch Matlab
2. Go to the folder: /usr/ccrma/courses/mir2009
3. Download the folder **Toolboxes** to your local Matlab folder and add to your Matlab path (including all subfolders).
4. Copy the folder **/usr/ccrma/courses/mir2009/audio** to the folder **/scratch. (The folder /scratch is actually a folder on your** local **hard drive.) You'll refer to this folder for any audio examples for the course.**

## SECTION 1

Purpose: We'll experiment with the different features for known frames and see if we can build a basic understanding of what they are doing.

**Make sure to save all of your development code in an .m file. You will be building upon and reusing much of this code over the workshop.**

1. Load the audio file **SimpleLoop.wav** into Matlab, storing it in the variable ***x*** and sampling rate in ***fs***.
   ```
   [x,fs]=wavread('SimpleLoop.wav');
   ```
2. You can play the audio file by typing using typing
   ```
   sound(x,fs)
   ```
3. Run an onset detector to determine the approximate onsets in the audio file.
   ```
   [onsets, numonsets] = ccrma_onset_detector(x,fs)
   ```
4. The onset values are displayed in samples. How do you display them in seconds?

One of Matlab's greatest features is its rich and easy visualization functions. Visualizing your data at every possible step in the algorithm development process not only builds a practical understanding of the variables, parameters and results, but it greatly aids debugging.

4. Plot the audio file in a figure window.
   ```
   plot(x)
   ```
5. Now, add a marker showing the position of each onset on top of the waveforms.
   ```
   plot(x); hold on; plot(onsets,0.2,'rx')
   ```
6. Adding text markers to your plots can further aid in debugging or visualizing problems. Label each onset with it's respective onset number with the following simple loop:
   ```
   for i=1:numonsets
     text(onsets(i),0.2,num2str(i));  % num2st converts an number to a string for display purposes
   end
   ```

   Labeling the data is crucial. Add a title and axis to the figures. (**ylabel**, **xlabel**, **title**.)
   ```
   xlabel('seconds')
   ylabel('magnitude')
   ```

```
title('my onset plot')
```

7. Now that we can view the various onsets, try out the onset detector and visualization on a variety of other examples. Continue to load the various audio files and run the onset detector - does it seem like it works well?

**Segmenting audio in Frames**

As we learned in lecture, it's common to chop up the audio into fixed-frames. These frames are then further analyzed, processed, or feature extracted. We're going to analyze the audio in 100 ms frames starting at each onset.

8. Create a loop which carves up the audio in fixed-size frames (100ms), starting at the onsets.
9. Inside of your loop, plot each frame, and play the audio for each frame.

```
% Loop to carve up audio into onset-based frames
frameSize = 0.100 *fs; % sec
for i=1:numonsets
  frames{i}= x(onsets(i):onsets(i)+frameSize);
  figure(1);
  plot(frames{i}); title(['frame ' num2str(i)]);
  sound(frames{i} ,fs);
  pause(0.5)
end
```

**Feature extract your frames**

Create a loop which extracts the Zero Crossing Rate ***for each frame***, and stores it in an array.
Your loop will select 100ms (in samples, this value is = ***fs*** * 0.1) , starting at the onsets, and obtain the number of zero crossings in that frame.

The command **[z] = zcr(x)** returns the number of zero crossings for a vector x.
Don't forget to store the value of **z** in a feature array for each frame.

```
clear features
% Extract Zero Crossing Rate from all frames and store it in "features(i,1)"
for i=1:numonsets
  features(i,1) = zcr(frames{i})
end
```

For SimpleLoop.wav, you should now have a feature array of 5 x 1 - which is the 5 frames (one at each detected onset) and 1 feature (zcr) for each frame.

**Sort the audio file by it's feature array.**

Let's test out how well our features characterize the underlying audio signal.
To build intuition, we're going to sort the feature vector by it's zero crossing rate, from low value to highest value.

10. If we sort and re-play the audio that corresponds with these sorted frames, what do you think it will sound like? (e.g., same order as the loop, reverse order of the loop, snares followed by kicks, quiet notes followed by loud notes, or ??? ) Pause and think about this.

11. Now, we're going to play these sorted audio frames, from lowest to highest. (The pause command will be quite useful here, too.) How does it sound? Does it sort them how you expect them to be sorted?

```
[y,index] = sort(features);

for i=1:numonsets
  sound(frames{index(i)},fs)
  figure(1); plot(frames{index(i)});title(i);
  pause(0.5)
end
```

You'll notice how trivial this drum loop is - but this is a good tip - always use familiar and predictable audio files

when you're developing your algorithms! No sense in testing out overly complex examples at this stage.

## BONUS [only if you finish the other sections]: More Cowbell!

Heuristics are experimentally derived rules; they are commonly used in a variety of applications. In this case, let's create some heuristics to detect when a snare drum is played. You will receive less direction in how to implement your techniques in this and further sections - if you need help, or don't know the proper Matlab commands for what you want to do, don't be afraid to ask.

i. Load the audio file "SimpleLoop.wav".
ii. Plotting the zero crossing rate of this file, can you tell which frames contain snare drums? If so, which ones?

Sonifying events by inserting a special tone or marker into the audio stream is another great way of developing and debugging your algorithms. By listening to the sonified and marked-up audio files, you will be able to easily hear if you are getting good results. Looking at an array of floating points numbers is one thing - but listening to your results can be much more intuitive.

i. Create a loop which implements your heuristics by playing a cowbell at the same time as the snare drum of a given loop. The audio file "cowbell.wav" can be found in the audio files folder.

   Play the mixed audio file with the cowbell inserted on top of the snare drum. (Rock on!)

ii. Now try on 125BOUNC-mono.WAV. Try some other files, too. What do you notice? Does your snare drum heuristic remain accurate the same for all of the other drum loop files?

   Notice what happens when the snares is hit harder, is pitched differently, has variation in its attack tone, has simultaneous hits with other instruments, or unseen events occur!

## BONUS 2 [only if you finish Bonus section 1] : My First Transcription

i. Open up the file zcr.m in the editor and look at how it works. This is one of the simplest of feature extractors.

ii. Using the audio file "SimpleLoop.wav", now create a simple transcription heuristic for determining kick and snare.

iii. Re-synthesize your transcription using some samples. (I placed a "kick.wav" and "snare.wav" in the audio files folder for you.)

iv. Now, try out your transcription with 125BOUNC-mono.WAV. Now try it on some drum loop examples. How well does it hold up?

   Hmm… looks like we'll need some additional intelligence to classify the full range of kicks and snares out there.

## BONUS 3 [the most bonus of all] :

Look at the audio files in the **/scratch** folder - I've added numerous examples of loops, samples, and popular music for your analysis. Experiment with the above techniques and steps using different audio files.

## NEED HELP?

**Loading audio into Matlab**

To load from the command line, you can load use the **wavread** command, such as [x,fs]=wavread('foo.wav').

**Listening to an audio file in Matlab**

sound(x,fs)
To stop listening to an audio file, press Control-C.

Audio snippets less than ~8000 samples will often not play out Matlab. (known bug on Linux machines)

**Tricks of the trade**

Select code in Matlab editor and then press F9. This will execute the currently selected code.
To run a Matlab "cell" (multiline block of code), press Control-Enter with the text cursor in the current cell.

The **clear** command re-initializes a variable. To avoid confusion, you mind find it helpful to clear arrays and structures at the beginning of your scripts.

**Common Errors**
>??? Index exceeds matrix dimensions.
Are you trying to access, display, plot, or play past the end of the file / frame?
For example, if an audio file is 10,000 samples long, make sure that the index is not greater than this maximum value. If the value is > than the length of your file, use an **if** statement to catch the problem.

**Why are the Paste / Save keys different? Why does Paste default to Control-Y?**
On Linux, Matlab defaults to using Emacs key bindings. To fix this, go:
File menu > Preferences > Keyboard
Switch the Editor/Debugger key bindings to "Windows"

**Cowbell**
If you don't understand the joke behind "more cowbell", check out :
http://webfeedcentral.com/2005/01/21/more-cowbell-video/