CS 591 Computational Audio

Final Project

Topic: Tempo Analysis

Name: Laixian WAN

Login ID: wanlx@bu.edu

**1) Files in this zipped directory**

-Project.docx: This is a short writeup about the whole final project.

-audioUtilities.py: A collection of the most important algorithms used throughout the class.

-project.py: The project itself. This docx file will briefly go through this python script and explain how it does(doesn't) work/how do I approach the problem/ideas in this script.

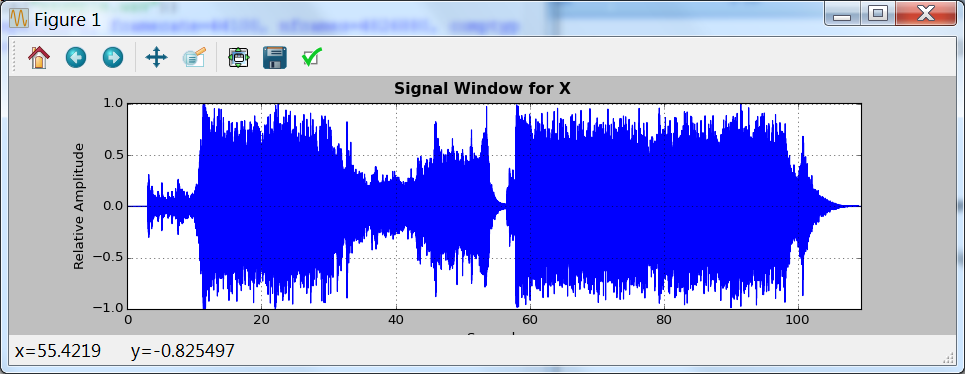
-deep blue sunrise.wav/Oriens.wav: two audio files. I will run the python script on those two wave files as examples of how my program works.

**2) Introduction/About the topic I picked for the final project**

Tracking the tempo of a musical recording is very challenging but also very interesting. Artists have thousands of techniques to express their understanding of the music throughout the performance, and changing the tempo is one of them. The tempo of a song/piece can change many times in different ways, and as a programmer, I would like to challenge myself by trying to find a single algorithm which can track the tempo of any music pieces. Of course my attempt is not very successful for the time being, but I do think it is a good try and valuable experience. I believe a program like this can not only satisfy my own curiosity but also benefits musicians and recorders. I spent years in practicing piano and keeping track of the tempo has always been a big challenge. It will be excellent if a real-time computer program which can constantly keeps track of my performance and gives very detailed feedbacks exist, and I definitely will spend more time on this project even out the CS591 class.

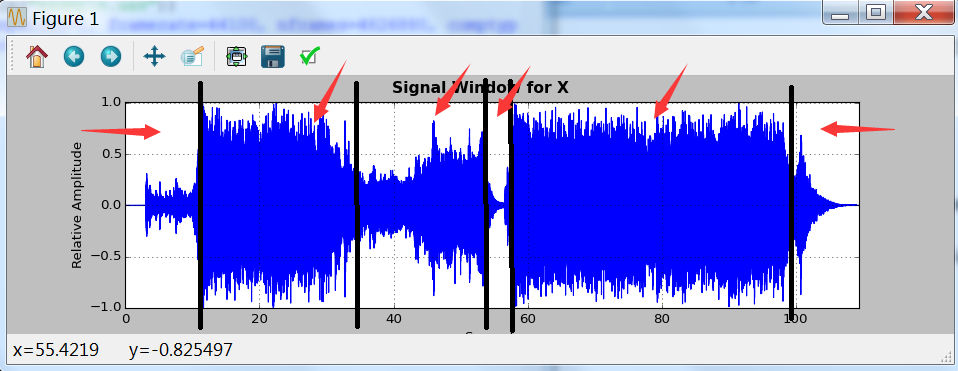
**3) Overall ideas/structure of the program**

The main idea for this project is to track the tempo by energy, or amplitude. For example, if we take a look at the following signal:



what can we say about the tempo of this signal?

Clearly we can first divide the signal to different section just by looking at it. Sections with bigger relative amplitudes have higher chance to be the climax of the music(thus corresponding to a higher tempo), and vice versa:



Section A: intro of the music(very low amplitude) should correspond to a slow tempo.

Section B: looks like the first climax of the music so it should have a fast tempo.

Section C: another smooth part of the music. It might be a bridge connecting two climax sections of the music(B and E) so it might have a slower tempo compare to section B and E.

Section D: musician halt a little bit here. Tempo must drop a lot too.

Section E: another climax, same as section B and should correspond to a fast tempo.

Section F: the music fade out so the music should correspond to a slow tempo.

This is how we human determine the tempo if we can only look at the signal. Computer, just like us, can also only read a bunch of numbers and determine the tempo from there. My program does exactly the same thing: (a) divides a signal to different parts and (b) determine the tempo of each part.

There are two main functions in my project.py. Function split(Wave) will take a signal and returns a list called "marked" which contains several timing points which divided the signal to different parts, and function detectBPM will evaluate each part and return the tempo(in BPM). The structure of the program looks like the following:

(a) user call the function tempoAnalysis() with the file they want to analyze.

(b) tempoAnalysis will read the wave by readWaveFile function and then pass the signal to split(Wave)

(c) split function returns timing points back to tempoAnalysis, and tempoAnalysis will call detectBPM with each parts.

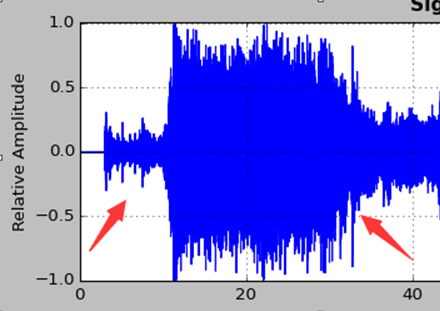
(d) detectBPM returns the BPM of each parts to tempoAnalysis.

(e) tempoAanlysis gather the results and plot them into a graph.

**4) details about split(Wave) and detectBPM(Wave)**

-detectBPM(Wave): detectBPM is generally the improved version of the program we wrote for hw08. Instead of taking a file, detectBPM take a fraction of a music, or in general any signal and return its BPM. However the basic idea is still the same: it first finds the rectification, calculates the difference, then finds the "beat spectrum" via realFFT(), and finally gives the result by looking at the peak.

-split(Wave): The way how I split the signal is looking at the "difference", similar to how we find the difference in the detectBPM function. The reason is that when a music meets its climax, amplitudes various greatly, resulting in huge difference among waves, while in clam sections, amplitudes generally stay in certain ranges and there won't be any "sharp edges":

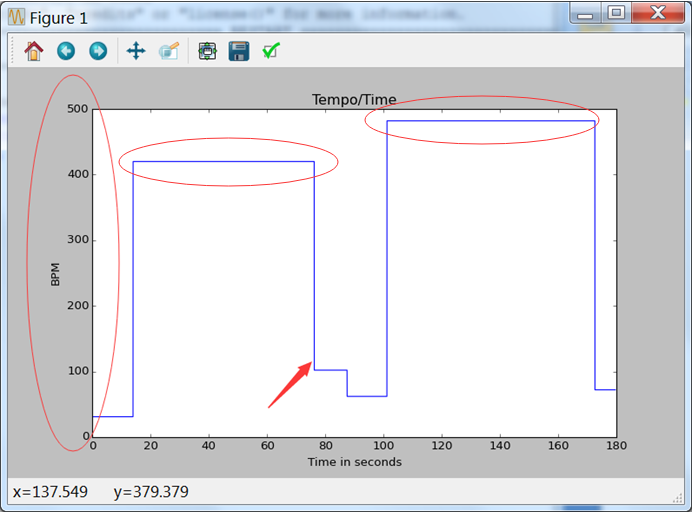


clam section vs. climax section: you can see the wave is very smooth in the clam section, while in the climax, there are many sharp edges, and there are huge difference among peaks.

The job of split function is to find timing points (peaks with huge differences), and then eliminate unnecessary timing points to find the boundary of each parts. For example in the above signal, split function will first find a few timing points around 10 seconds(where intro ends and the music proceed to its first climax) and then finds a bunch timing points until 35 seconds, where the first climax end. Then the function will eliminate all the timing points between 10~35 seconds(by a constant c decided by the programmer), and return the rest timing points which divide the music to 3 parts: 0~10s(first timing point), 10~35s(first and second timing points) and 35s~(after the second timing point).

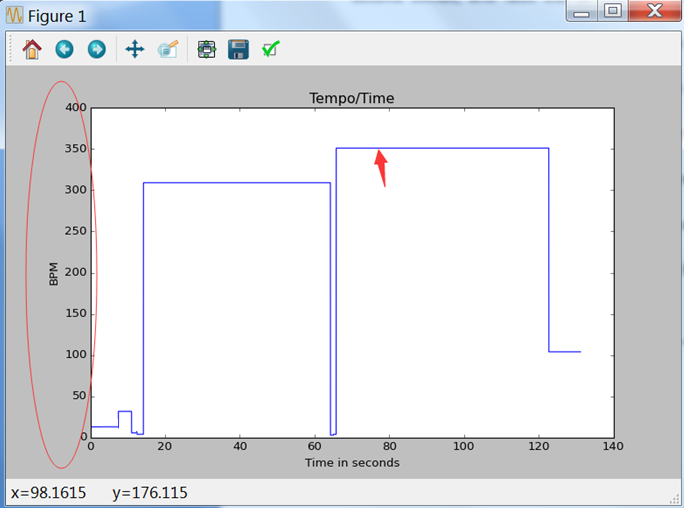
**5) Examples: deep blue sunrise.wav and Oriens.wav: success and problems of the program.**

(a) deep blue sunrise.wav:



My program is quite successful in several aspects: it finds out the intro part very well(0~16/17s) and identifies a slow tempo. Then the music reaches its first climax, last for a while until 90s, followed by a short rest for about 10s and then goes to its second climax, and fade out at the end. However there are two huge obvious problems: (a) (arrow) my program finds a timing point at around 75s, which shouldn't exist. The music is still in the climax at that time and the second issue (b) (circle) is my detectBPM function. This function works pretty well in homework 8 and it finds out the BPM for the simple sine wave in the homework, however it totally fails for some real music, for example deep blue sunrise.wav. The real BPM is actually 1/4 the BPM shown in the graph, and two climaxes of the music should have the same BPM.

(b) Oriens.wav



This example is pretty much the same as the previous one. The BPM detection problem still exist(circle), and even tho the program spot a "rest period" in the middle, it miscalculate the timing point where that period should end(it should actually end at around 80s, arrow). But one thing that really surprised me is my program catch a very small speed up at the intro(around 15/16s).

**6) Future directions**

Playing with my project for about two weeks is a very valuable experience to me. I enjoyed a lot and I think my effort pay off. I really would like to spend more time on making the detectBPM function works for real music and trying to make split function more accurate. I will definitely pay attention to my plan next time: I think I was way too ambitious. After doing this project for more than two weeks, I realized that tempo analysis for real music is a very huge and complicated topic: real signal is not even close to any simple wave we studied in class. Analyzing how they behaves and find a way to split them is much harder than I thought, and impossible to be fully done in just two weeks. But I am still satisfy with what I have done so far and I will not stop right here. I will definitely spend more time on this interesting topic.

Laixian Wan

5/2/16