A1: Python and sounds

Course on Audio Signal Processing for Music Applications

Introduction

Welcome to the course! This first programming assignment is for you to get familiar with the assignment submission system and to test out some very basic audio operations using Python. There are four parts in this assignment: 1) Reading an audio file, 2) Basic operations with audio, 3) Python array indexing, and 4) Downsampling audio - Changing the sampling rate. The Part-4 of this assignment is optional and will not contribute towards your final grade.

Before attempting the assignment, please go through the general guidelines for all the programming assignments. You can find them in 'Programming assignment-Guidelines.pdf' in the assignment folder (A1) and on the *Instructions* tab of programming assignment 1 in Coursera platform.

Should you have any questions or doubts about this assignment, please go the *Discussions* tab of programming assignment 1 in Coursera platform. There you can find questions, discussion and clarification about this assignment. If you do not find the answer that you are looking for, you can start a new thread.

Relevant concepts

Python: Python is a powerful and easy to learn programming language, which is used in a wide variety of application areas. We will use python in all our programming assignments and in this first assignment you will start learning about it by performing some basic operations with sound files. For a quick introduction to python watch the first programming lecture of the first week of the course.

Wav file: The wav file format is a lossless format to store sounds on a hard drive. Each audio sample is stored as a 16 bit integer number (sometimes also as 24 bit integer or 32 bit float). In this course we will work with only one type of audio files. All the sound files we use in the assignments should be wav files that are mono (one channel), in which the samples are stored in 16 bits, and that use (most of the time) the sampling rate of 44100 Hz. Once read into python, the samples will be converted to floating point values with a range from -1 to 1, resulting in a one-dimensional array of floating point values.

A1-Part-1: Reading a wav audio file (*3 points*)

Complete the function `readAudio(inputFile)` in the file `A1Part1.py` so that it reads an audio file and returns 10 consecutive samples of the file starting from the 50001th sample. This means that the output should exactly contain the 50001th sample to the 50010th sample (10 samples).

The input to the function is the file name (including the path) and the output should be a numpy array containing 10 samples.

If you use the wavread function from the utilFunctions module the input samples will be automatically converted to floating point numbers with a range from -1 to 1, which is what we want.

Remember that in python, the index of the first sample of an array is 0 and not 1.

If you run your code using piano.wav as the input, the function should return the following numpy array with 10 samples: array([-0.06213569, -0.04541154, -0.02734458, -0.0093997 , 0.00769066, 0.02319407, 0.03503525, 0.04309214, 0.04626606, 0.0441908], dtype=float32).

```
def readAudio(inputFile):
    """
    Input:
        inputFile: the path to the wav file
    Output:
        The function should return a numpy array that
        contains 10 samples of the audio.
    """
    ## Your code here
```

A1-Part-2: Basic operations with audio (*3 points*)

Complete the function `minMaxAudio(inputFile)` in the file `A1Part2.py` so that it reads an audio file and returns the minimum and the maximum values of the audio samples in that file.

The input to the function is the wav file name (including the path) and the output should be two floating point values returned as a tuple.

If you run your code using oboe-A4.wav as the input, the function should return the following output: (-0.83486432, 0.56501967)

```
def minMaxAudio(inputFile):
    """
    Input:
        inputFile: file path to the wav file
    Output:
        A tuple of the minimum and the maximum value of the audio
        samples, like: (min_val, max_val)
    """
    ## Your code here
```

A1-Part-3: Python array indexing (*4 points*)

Complete the function `hopSamples(x,M)` in the file `A1Part3.py` so that given a numpy array x, the function returns every Mth element in x, starting from the first element.

The input arguments to this function are a numpy array x and a positive integer M such that M is less than the number of elements in x. The output of this function should be a numpy array.

If you run your code with x = np.arange(10) and M = 2, the function should return the following output: array([0, 2, 4, 6, 8]).

```
def hopSamples(x,M):
    """
    Inputs:
        x: input numpy array
        M: hop size (positive integer)
    Output:
        A numpy array containing every Mth element in x, starting
        from the first element in x.
    """
    ## Your code here
```

A1-Part-4: Downsampling audio - Changing the sampling rate (*Optional*)

One of the required processes to represent a signal inside a computer is *sampling*. The sampling rate is the number of samples obtained in one second when sampling a continuous *analog* signal to a discrete *digital* signal. As mentioned earlier, most of the time we will be working with wav audio files that have a sampling rate of 44100 Hz, which is a typical value. For some applications, changing the sampling rate of an audio signal can be necessary. This optional part shows how to do this, from a higher sampling rate to a lower one.

Complete the function `downsampleAudio(inputFile,M)` in the file `A1Part4.py` so that given an audio file, it applies downsampling by a factor of M and create a wav audio file <input_name>_downsampled.wav at a lower sampling rate.

In Part1 you learned how to read a wav file and the function from Part3 can be used to perform the downsampling of a signal contained in an array. To create a wav audio file from an array, you can use the wavwrite function from the utilFunctions module. Be careful with the sampling rate parameter since it should be different from that of the original audio.

You can test your code using the file 'vibraphone-C6.wav' and a downsampling factor of M=16. Listen to the 'vibraphone-C6_downsampled.wav' sound. What happened to the signal? How could we avoid damaging the signal when downsampling it? You can find some related information in `https://en.wikipedia.org/wiki/Decimation_%28signal_processing%29`.

```
def downsampleAudio(inputFile, M):
```

```
"""
Inputs:
    inputFile: file name of the wav file (including path)
    M: downsampling factor (positive integer)
"""
## Your code here
```

Grading

Only the first three parts of this assignment are graded and the fourth part is optional. The total points for this assignment is 10.