CSE 3313 – Homework #9

Assigned: Monday, April 20, 2020
Due: Monday, April 27, 2020 at end of class

Assignment:

1. (will not be graded) We have the transfer function for an analog bandpass filter:

$$H(s) = \frac{2s}{s^2 + 2s + 4}$$

Note that I chose the values of 2 and 4 in the formula, but really these coefficients would be chosen for the characteristics that you want from the filter.

Use the bilinear transformation to produce $H(z)$ and from this the corresponding output difference equation. For this, use the following values

- $f_s = 1000$ Hz
- $\omega_d = 200\pi$
- $c = 1$
- $s = (z-1)/(z+1)$

Be clear what $H(z)$ and $y[n]$ are. As you move through the entire process, round off each result to 2 decimal places (I'm going for simple here, not numerical precision). The output difference equation should be in a form similar to what we have done all semester; don't leave the arithmethic and simplifications for me to do.

2. (100 points) **Application Area: Music Identification**
Purpose: Learn to extract features from the spectrogram of an audio file.

The music identification service Shazam can recognize a song by "listening" to a short piece of it with a user's mobile phone [Wan03]. While the underlying algorithm is more complex than I am describing here, the basic idea is that a spectrogram of the song being played is generated, then the peaks within the spectrogram are used to generate a signature representing the song. This signature is compared to a database of existing song signatures to identify the song being played.

   (a) The course website has a skeleton file called `musicID.py` that you should complete. It calls a function named `classifyMusic()`, which has the signature

```
classifyMusic()
```

You will create `classifyMusic()`. While it's OK to decompose your code into multiple functions, I should be able to call `classifyMusic()` with my own arguments and have everything work.

Do not create any global variables.

(b) On Blackboard is a file, `music_wav.tgz` (warning: it's 145 MB). It contains:

- A `README.txt` file.
- 64 files of the form `song-*.wav`; these are the original songs that you will build a database of signatures from.
- 64 files of the form `test-*.wav`; these are the corrupted versions of the songs that your program should be able to identify.
- `testSong.wav` is a sample test file. To change the test song, copy one of the other `test-*.wav` files to this filename.

(c) Your program should assume that the music files are in the same directory as your program.

(d) Do not read the files with names of the form `test-*.wav`

(e) We are going to use the scipy function `spectrogram()` to analyze each song. In order for us to get the same results, we will call it like this:

```
f, t, Sxx = spectrogram(x, fs=fs, nperseg=fs//2)
```

where

- `f` is an m-element vector of the frequencies evaluated in the signal.
- `t` is an n-element vector of the times at which the signal was evaluated. While a more robust system might use this information, we don't care about the actual times.
- `Sxx` is an $m \times n$ matrix of the amplitudes of each frequency/time pair.

(f) For each of the 64 songs that fit the pattern `song-*.wav` you will need to store its signature and its name. This will be our "database" of songs. To do this without hard-coding the 64 filenames in your program or the number 64, use the `glob` library to build a list of filenames that match the pattern `song-*.wav`.

(g) The signature for each song will be a list of the maximum frequencies within each segment of the spectrogram. Therefore, there will be one value for each of the time segments. For example, if the results of the spectrogram were to produce the plot in

F	0Hz	3	5	**9**
R	10Hz	3	6	4
E	20Hz	**8**	5	7
Q	30Hz	1	**7**	2
		0	1	2
	T	I	M	E

Table 1: sample spectrogram

then the signature would be [20, 30, 0] since the max value in column 0 is 8 (frequency = 20Hz), the max value in column 1 is 7 (frequency = 30Hz), and the max value in column 3 is 9 (frequency = 0Hz).

My signature for `song-beatles.wav` is

```
(148.0, 146.0, 72.0, 72.0, 108.0, 176.0, 146.0, 148.0, 148.0, 72.0,
 72.0, 110.0, 176.0, 148.0, 148.0, 148.0, 72.0, 72.0, 108.0, 176.0,
```

```
148.0, 148.0, 108.0, 112.0, 112.0, 132.0, 134.0, 110.0, 112.0, 112.0,
 56.0, 112.0, 132.0, 134.0, 110.0, 102.0, 98.0, 98.0, 48.0, 48.0,
 48.0, 48.0, 48.0, 50.0, 38.0, 2476.0, 48.0, 52.0, 48.0, 72.0, 72.0,
110.0, 176.0, 176.0, 148.0, 146.0, 74.0, 146.0, 176.0, 176.0, 146.0,
146.0, 72.0, 72.0, 108.0, 176.0, 146.0, 146.0)
```

(h) To test your program, it will read a file called `testSong.wav`, which is just a copy of one of the corrupted songs matching the pattern `test-*.wav`. You will build a signature of this file.
Note: You WILL hard-code the name `testSong.wav` in your function, not the name of any of the specific test files.

(i) The metric that we will use to compare the signatures of the test song and a song in our database is the vector 1-norm. Given a vector $\vec{v} = [v_1, v_2, \ldots, v_n]$, the vector 1-norm is

$$||\vec{v}||_1 = \sum_{i=1}^{n} |v_i| = |v_1| + |v_2| + \cdots + |v_n|$$

The vector 1-norm is also known as the taxicab distance, city-block distance, or Manhattan distance.

The test will be the 1-norm of the difference of the two signatures. That is, if the signature of the test song is $\vec{t}$ and the signature of the song from our database is $\vec{s}$, then you will produce $||\vec{s} - \vec{t}||_1$.

(j) You will use this test to compare the signature of the test sample with the signature of each song in the database. The results for the 5 songs closest to the test song (as measured by out test) will be printed to the screen (i.e., `stdout`) in ascending order by norm value. Each line of output will consist of

- the norm value (to zero decimal places)
- two spaces
- the name of the song from the database used in the test

For example, here are my results when the test file is `test-winds.wav`:

```
10588  song-winds.wav
18788  song-marlene.wav
19922  song-ballad.wav
20026  song-madradeus.wav
20610  song-unpoco.wav
```

You will also display the spectrograms of `testSong.wav` and the first two best matches, each in its own figure window. For this example, the first two best matches are `song-winds.wav` and `song-marlene.wav`. For displaying the spectrograms, use this:

```
matplotlib.pyplot.specgram(x, Fs=fs)
```

It's default color map is better than that of `spectrogram()`.

(k) When uploading, DO NOT include the music files.

General requirements about the Python problems:

a) As a comment in your source code, include your name.

b) The Python program should do the work. Don't perform the calculations and then hard-code the values in the code or look at the data and hard-code to this data unless instructed to do so.

c) The program should not prompt the user for values, read from files unless instructed to do so, or print things not specified to be printed in the requirements.

To submit the Python portion, do the following:

a) Create a directory using your last name, the last 4 digits of your student ID, and the specific homework, with a hyphen between your ID and the homework number. For example, if John Smith has a student ID of 1000123456 and is submitting hw02, his directory would be named **smith3456-hw02**. Use all lowercase and zero-pad the homework number to make it two digits.

   If you have a hyphenated last name or a two-part last name (e.g., Price-Jones or Price Jones), let's discuss what you should do.

b) Place your .py files in this directory.

c) Zip the directory, not just the files within the directory. You must use the zip format and the name of the file (using the example above) will be `smith3456-hw02.zip`.

d) Upload the zip'd file to Canvas.

References

[Wan03] Avery Li-Chun Wang. An industrial-strength audio search algorithm. In *Proceedings of the 4th International Conference on Music Information Retrieval*, 2003.