ECE-GY 6183: DSP Lab Summer 2020

Midterm Exam

Policy:

This exam is open text book and open course notes. You may use the documentation of Python, Pyaudio, and any other libraries. You may not ask anyone for help.

You may consult textbooks and online ‘textbook’ resources on the topic of DSP (e.g., Matlab Signal Processing ToolBox documentation).

In your answer to the exam, you must list any resources you consult (other than the course text book, course notes on NYU Classes, Python documentation).

You should not search the web for answers (you should not ‘Google’ for answers).

The work you do must be entirely your own.

Supplemental file: audio_signal.wav

1) [60%] Write a program in Python and Pyaudio to generate an audio signal like the provided signal. The approach is to first generate white noise, and then filter the white noise using a time-varying band-pass filter. The center frequency of the band-pass filter varies as a function of time. Your program should play the audio on the computer loudspeaker as it generates the signal, and it should also write the signal to a wave file.

- You should submit your Python code, and the wave file it produces.

- With your answer, **write an explanation** of the steps you took to decide how to reproduce the provided sound in “audio_signal.wav”. (For example, how did you determine the characteristics of the band-pass filter from the the provided signal?).

- Comment your code, or credit will be subtracted.

- Your code should be free of compile and runtime errors.

- Your output signal does not have to be exactly the same as the provided signal. In fact, it should not be exactly the same, because it is generated by a random input signal.

- You may find it helpful to look at the frequency spectrum of segments of the provided audio signal, or a live spectrum.

2) [30%] Using Python and TKinter, write a graphical user interface (GUI) to generate signals similar to the provided signal. The GUI should have sliders to control the rate at which the signal changes and the band-width of the band-pass filter in real-time. The goal is that the GUI has low latency, so that the signal changes smoothly and with minimal delay as the user adjusts the sliders.

3) [10%] In the course, we saw that when we implement a difference equation in real-time in Python using the **lfilter** function (scipy.signal.lfilter) with block processing, it is important to correctly set the initial states of the filter for each block to avoid error artifacts due to block processing. This requires getting the output states at the end of each block.

If H1(z) and H2(z) are two LTI systems, then so is the cascade (series) connection H(z) = H1(z) H2(z). Suppose H1 is implemented as the difference equation as: **lfilter(b1, a1, x, zi = …),** and H2 is implemented as **lfilter(b2, a2, x, zi = …).** Describe how you would go about implementing the total system H(z) in real-time with block processing so that the initial states are correctly set so that no error artifacts arise. Do any problems arise? What kinds of issues do you need to be aware of?