# Lab 2: Getting Started with PyAudio

DSP Lab (EE 4163 / EL 6183)

Last Edit: Thursday 14th September, 2017

## 1 PyAudio

### 1.1 Installing PyAudio

The PyAuido Module is available here:

https://people.csail.mit.edu/hubert/pyaudio/

https://people.csail.mit.edu/hubert/pyaudio/docs/

There are instructions on installing yAudio on the PyAudio web page.

To verify that PyAudio is installed, type `import pyaudio` in Python on the interactive command line.

Run the demo program `filter_16.py` to test that PyAudio works.

### 1.2 PyAudio constants

The PyAudio module includes two classes: `PyAudio` and `Stream`. `PyAudio` handles the linking of the program to the hardware. `Stream` is an I/O stream dependent on a specific `PyAudio` object, the input and/or output data of which the program can change in order to obtain a sound effect.

See the demo file `filter_16.py`.

The constant `pyaudio.paInt16` indicates the 16-bit encoding format in PyAudio. Other constants include `paInt32, paInt24, paInt16, paInt8`. These constants are just integer values defined in PyAudio.

```
1 |     print pyaudio.paInt16
```

When the `WIDTH` (bytes per sample) is known, then the format can be specified by

```
1 | pyaudio.get_format_from_width(WIDTH)
```

## 2 Demo files

This lab assignment corresponds to several demo programs.

`make_filter_01.m`
`make_filter_02.m`
`make_filter_03.m`

`filter_cat.m`
`filter_16.py`
`filter_32.py`
`filter_16_r.py`
`filter_16_T.py`
`filter_twice.py`
`filter_gui_example_ver1.m`
`filter_gui_example_ver2.m`

# 3 Exercises

1. Verify that the following gives the same result.

```
print pyaudio.paInt16
```

   and

```
WIDTH = 2
print pyaudio.get_format_from_width(WIDTH)
```

   Why do 16-bit signed integers have a width of 2?

2. What is `WIDTH` when the PyAudio format is `paInt8`?

3. In `filter_16.py`, setting the value of `gain` to a very large value (e.g. 100000, 1000000) results in an error. Why? How can this error be avoided? How should the gain be set to ensure signal values do not exceed the maximum allowed value of $2^{15} - 1$ ?

4. Modify `filter_16.py` to avoid run-time overflow errors even if `gain` is very high, by clipping the signal as necessary. To do this, insert an if statement to verify that the sample value is in the allowed range. If it is not, then set the value to its maximum (positive or negative) allowed value, before writing it to the audio stream. Test your program by setting the gain to a high value. What effect does this have on the sound produced by the program? SUBMIT

5. The demo program `filter_16.py` implements a filter with transfer function SUBMIT

$$H(z) = \frac{1}{1 + a_1\, z^{-1} + a_2\, z^{-2}}. \tag{1}$$

   Modify the filter in the program so that the transfer function is

$$H(z) = \frac{b_0 + b_1\, z^{-1} + b_2\, z^{-2}}{1 + a_1\, z^{-1} + a_2\, z^{-2}} \tag{2}$$

   where $B(z) = b_0 + b_1\, z^{-1} + b_2\, z^{-2}$ is set so that the impulse response is exactly

$$h(n) = r^n \cos(\omega_1 n)\, u(n) \tag{3}$$

   where $u(n)$ is the unit step function. Note that the first value of this impulse response is 1, and that all other values are less than 1. That is $h(n) \leq 1$ for all $n$. To find $b_n$,

you may consult a table of Z-transforms! How should `gain` be set to ensure the impulse response does not exceed the maximum allowed value of $2^{15} - 1$ ?

Write a Matlab program that calculates the impulse response using the `filter` function and plot the impulse response, to verify that the initial value is 1.

Implement the filter in real-time using Python/PyAudio (modify the demo program `filter_16.py`).

6. Modify `filter_16.py` so that it produces a stereo signal with a different frequency in left and right channels. Use headphones to verify the stereo effect. SUBMIT

7. Write a version of `filter_16.py` using 8 bits/sample. You may use `paInt8` as the PyAudio format. Ensure run-time overflow errors do not occur.

8. The demo program `filter_twice.py` applies the same second-order filter twice in cascade. The result is a fourth-order filter. Modify this program so that the rise-time and decay-time of the impulse response are different. The two second-order filters should have the same resonant frequency $f_1$, but they should have different pole radii. Design the two filters so that the impulse response has a short rise-time and slow decay-time. Implement the filter in both Matlab and in real-time in Python/PyAudio. Use Matlab to save a plot of the impulse response and a WAV file of the sound. Confirm the sound produced by your Python program is the same as the WAV file you save in Matlab.

9. Modify the Matlab demo program `filter_cat.m` to use different filters.

   (a) A higher-order Butterworth band-pass filter.

   (b) A Chebyshev Type II band-pass filter (use `cheby2` instead of `butter` in Matlab to design the filter coefficients).

   (c) An elliptic band-pass filter (use `ellip` in Matlab to design the filter coefficients).

   (d) A Butterworth band-stop filter (instead of a band-pass filter). SUBMIT (d)

   Make plots showing the filters and input/output signals, as in the demo file. Comment on your observations.

10. Matlab Graphical User Interface (GUI). Write a Matlab GUI that allows the user to control the cut-off frequency of a low-pass filter. The GUI should have a slider for the cut-off frequency. The GUI should display the SUBMIT

    - impulse response
    - frequency response (magnitude)

    These plots should update as the user adjusts the slider.