Lab 2: Getting Started with PyAudio

EE 4163 / EL 6183 : Digital Signal Processing Lab

Fall 2015

1 Audio using a recursive filter

In this section, we illustrate the use of a second-order recursive filter to generate a decaying sinusoidal signal. This section is in Matlab.

For example, suppose sampling rate is $f_s = 8$ kHz and we wish to generate a decaying sinusoidal signal with a frequency of $f_1 = 400$ Hz. A recursive filter to produces this frequency has the zero-pole diagram shown in Figure 1(a). We have set the pole radius to $r = 0.999$. To generate the sinusoidal signal, we drive the recursive filter with the impulse,

$$x(n) = \begin{cases} 1, & n = 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

To simulate this system and input signal in Matlab, we generate a one-dimensional array with first element being 1 and all other elements being 0. In Matlab, we write

```
N = 4000;
x = zeros(N, 1);          % all zeros
x(1) = 1;           % first value is 1
```

The recursive filter can be implemented by

```
Fs = 8000;              % sampling frequency (sample/second)
F1 = 400;               % frequency (cycles/second)
f1 = F1/Fs;             % normalized frequency (cycles/sample)
om1 = 2 * pi * f1;
r = 0.999;
a = [1 -2*r*cos(om1) r^2];
b = 1;
y = filter(b, a, x);
```

Then we obtain the output signal as Figure 2.

Assignment 1

1. Write a Matlab script to make a recursive filter as in Figure 1. Use `filter()` to generate the sinusoid signal as in Figure 2. Use `soundsc()` to listen to the output signal.

2. Use Matlab to plot the first period (20 points) to verify the frequency of the signal. How can the frequency be inferred from a plot of these points? Also plot the Fourier spectrum to verify the frequency of the signal.

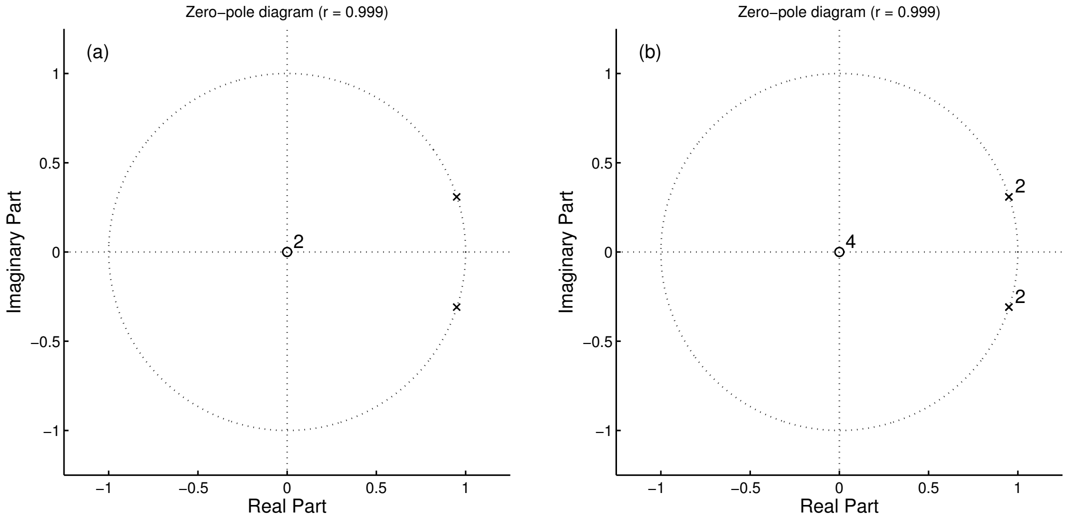

Figure 1: Zero-pole diagrams. (a) Second order system. (b) Fourth order system.

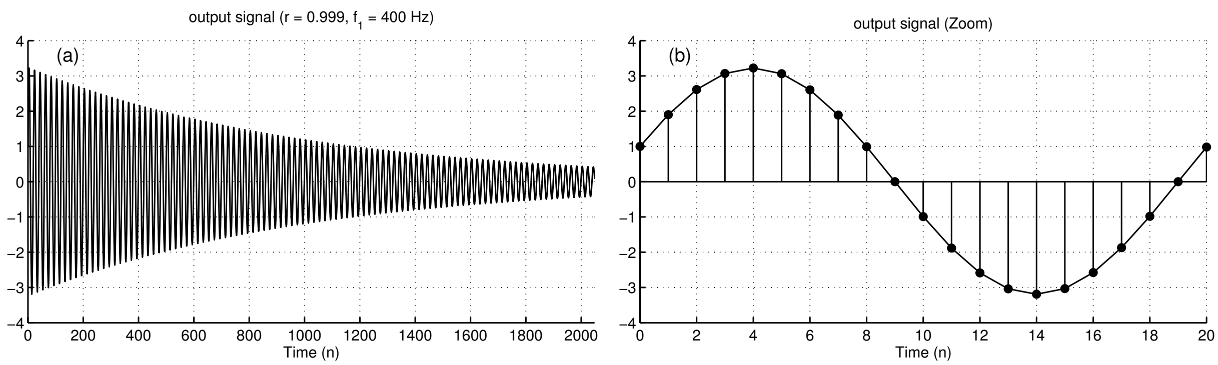

Figure 2: Output signal and its detail.

3. Write a Matlab script to read one of your wav files from Lab 1. Use the signal from the wav file as the input signal to the filter in Figure 1(a). Compare

```
soundsc(input, fs)
```

and

```
soundsc(output, fs)
```

Describe what you hear from the filtering. (The sampling rate should be determined by your recording).

4. Use the second-order system to create a fourth-order system with double poles as in Figure 1(b). This is equivalent to concatenating two identical second-order systems.

5. How does the pole radius (r) affect the amplitude profile of impulse response (What is the mathematical relationship of r and the time for the amplitude profile to decaty to 1% of its original value.)

2 Install PyAudio

2.1 Windows Users

The installation file of PyAudio is available at:

- `PyAudio` for Windows: `http://people.csail.mit.edu/hubert/pyaudio/`

Windows users need to download the version of PyAudio corresponding to the version of Python on their computer. In this course, we are using only `Python 2.7.x` with `PyAudio for Python 2.7`.

On the webpage of PyAudio as shown in Figure 3, select the third link (`PyAudio for Python 2.7`) assuming this version of Python has been installed on your computer (as in Lab 1).

Figure 3: Downloadable files for Windows.

Note that PyAudio only supports 32-bit Python. The version of Windows (64-bit or 32-bit) is irrelevant.

In the installation process, the PyAudio installer will search for default directory of Python (e.g., `C:/Python27/`). The installation program will run only when the correct version of Python can be found in the correct directory (see Figure 4).

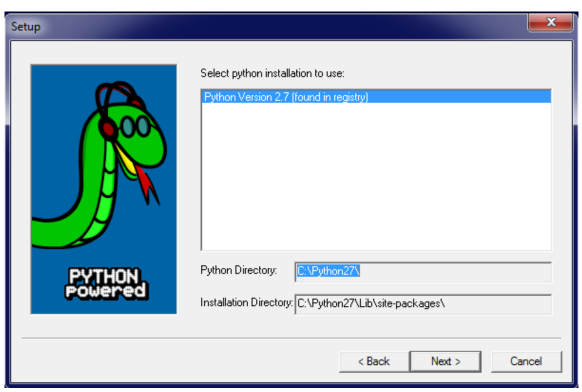

Figure 4: PyAudio installation.

2.2 Mac OS X Users

For Mac users, there is only one file to download on the webpage. It is a `*.dmg` file, which should be easily installed without any configurations.

We use Python 2.x for this course, so there is no need to configure MacPython 3.3 on your Mac.

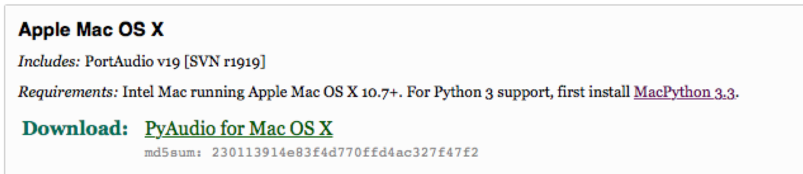

Figure 5: Downloadable files for Mac OS X.

2.3 Ubuntu Users

Ubuntu or other Debian based Linux users may need to download the installation file through the link on the webpage as shown Figure 6 dependent on the system version (i386 or amd64).

In Ubuntu, the downloaded package can be installed automatically by `Ubuntu Software Center`.

Figure 6: Downloadable files for Ubuntu.

Assignment 2

1. Verify your computer has Python 2.7.x with PyAudio installed (32-bit Python for Windows Users). (Type `'import pyaudio'` in Python on the interactive command line. Verify that the system gives no error).

3 PyAudio

In this section, we start to use PyAudio.

3.1 Use PyAudio Module

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.

For more details see:

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

To use PyAudio, the user needs to import the `pyaudio` module:

```
import pyaudio  # PyAudio module
```

Then the user creates an instance of `PyAudio`:

```
p = pyaudio.PyAudio()
```

To create a stream, some variables of `stream` should be defined and assigned with specific values.

```
format_pyaudio_int = pyaudio.paInt16    # 16 bits per sample
NUM_CHANNEL_INT = 1                     # mono
FS = 16000                              # Sampling rate of 16K samples per second
LENGTH_BUFFER_INT = 2                   # Number of frames as a block
```

Then, a stream can be defined:

```
stream = p.open(format=format_pyaudio_int,  # format (intager) by pyaudio
                channels=NUM_CHANNEL_INT,   # number of channels (1 for mono)
                rate=FS,                    # sampling rate (samples per second)
                input=False,                # stream of input  (True or False)
                output=True,                # stream of output (True or False)
                frames_per_buffer=LENGTH_BUFFER_INT) # samples per buffer
```

Note that `pyaudio.paInt16` is a value specified in PyAudio for 16-bit encoding format (recall Lab 1). Then the `format` variable required as an input of `open()` is an integer defined within `pyaudio` module, where the specific values can be found by

```
print pyaudio.paInt16
```

or using `paFloat32, paInt32, paInt24, paInt16, paInt8, paUInt8, paCustomFormat`, which is dependent on applications. It should be noted that when the `WIDTH` (bytes per sample) is known, the `format` can be obtained by

```
pyaudio.get_format_from_width(WIDTH)
```

Assignment 3

1. Verify that with `WIDTH = 2`,

   ```
   print pyaudio.paInt16
   ```

   and

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

   give the same result. Why does encoding (or digitizing) a sample into a 16-bit signed integer have a width of 2?

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

3.2 Code Example

This program can be used to test the installation of Python and PyAudio: `filtering_paInt16_a.py`

```python
from math import cos, pi
import pyaudio
import struct

# 16 bit/sample

# Fs : Sampling frequency (samples/second)
Fs = 8000
# Try Fs = 16000 and 32000

T = 1        # T : Duration of audio to play (seconds)
N = T*Fs     # N : Number of samples to play

# Difference equation coefficients
a1 = -1.8999
a2 = 0.9977

# Initialization
y1 = 0.0
y2 = 0.0
gain = 10000.0

p = pyaudio.PyAudio()
stream = p.open(format = pyaudio.paInt16,
                channels = 1,
                rate = Fs,
                input = False,
                output = True,
                frames_per_buffer = 1)

for n in range(0, N):

    # Use impulse as input signal
    if n == 0:
        x0 = 1.0
    else:
        x0 = 0.0

    # Difference equation
    y0 = x0 - a1 * y1 - a2 * y2

    # Delays
    y2 = y1
    y1 = y0

    # Output
    out = gain * y0
    str_out = struct.pack('h', out)     # 'h' for 16 bits
    stream.write(str_out, 1)

print("* done *")

stream.stop_stream()
stream.close()
p.terminate()
```

This code uses a block scheme to produce input samples with a `for` loop. In this loop, the program `pack` converts the output sample to a string. Then the string is written to the `stream` in order to produce sound.

`PyAudio` can only read and write binary strings to a stream. Hence, we use `pack()` to convert a number to a string so that it can be written to the `stream`. In the example code, the format `'h'` indicates signed 16-bit integer (`short` in the C language).

This conversion is needed because `PyAudio` can only read and write the digitized audio samples as binary strings. A binary string can be represented compactly in terms of hexadecimal symbols (e.g., `"\x00"`). For example, if the format is a signed 16-bit integer, then `PyAudio` reads and writes value '0' as a `string` `"\x00\x00"`, and value '1' as a `string` `"\x01\x00"`, etc., where the lower byte (8 bits) is on the left and high byte (8 bits) is on the right. More specifically, `"\x01\x00"` is the binary number

$$\texttt{1 as an integer} \longleftrightarrow \underbrace{\underbrace{\texttt{"\textbackslash x01}}_{\text{low}} \ \underbrace{\texttt{\textbackslash x00"}}_{\text{high}}}_{\text{2 bytes in hex}} \longleftrightarrow \underbrace{\underbrace{\texttt{0000 0000}}_{\text{high}}\underbrace{\texttt{0000 0001}}_{\text{low}}}_{\text{two bytes in binary}} \tag{2}$$

The method `unpack()` in the `struct` module can convert the string back into numbers.

The scheme of converting between number and binary string is controlled by the format mark, e.g. `'h'`. The format `'h'` means converts between signed 16-bits integers and binary strings. For the details about `unpack()` and `pack()`, see the documentation of the `struct` module:

`struct`: https://docs.python.org/2/library/struct.html

Assignment 4

1. What hexadecimal symbol string gives `-1` as a signed 16-bit integer using `unpack()` command?

2. What hexadecimal symbol string gives `256` as a signed 16-bit integer using `unpack()` command?

3. What hexadecimal symbol strings give the above numbers as signed 32-bit integers using `unpack()` command?

4. Change the value of `gain` to a larger number (e.g. 400000, 800000 or even larger). What happens? What error do you get?

5. In `filtering_paInt16_a.py`, how should you set the gain to ensure the peak amplitude does exceed the maximum allowed value of $2^{15} - 1$?

6. Modify `filtering_paInt16_a.py` to avoid run-time overflow errors even if `gain` is very high. Do this by inserting an if statement to verify if the sample value is in the allowed range; and if not, to set the sample value to its maximum (positive or negative) allowed value, before writing the sample value 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?

7. Implement the if statement by defining a python function with `def`. Put your function into its own module (file). Then `import` your module and use your function to modify the code example so that `gain` can be an arbitrary floating-point value without leading to a run-time error.

8. Write a version of `filtering_paInt16_a.py` using 8 bits/sample. You may use either `paInt8` or `paUInt8` as the PyAudio format. Ensure run-time overflow errors do not occur.

9. Write a version of `filtering_paInt16_a.py` that applies the filter twice. The filter will be a fourth-order filter, but it should be implemented as two second-order filters in cascade.

10. Describe how to design two second-order filters (with same resonant frequency f_1) so that the rise-time and decay-time of the impulse response can be specified? (The two filters will have different pole radii.) Given an example of the design in Matlab and its real-time implementation in Python/PyAudio.

4 To Submit

What to submit for this Lab:

Assignment 1: nothing to submit.

Assignment 2: nothing to submit.

Assignment 3: Your answers to questions 1 and 2.

Assignment 4: Your answers to questions 1 through 5 (no code). Your code for questions 6 through 9. Your mathematical derivation and code for question 10.