Lab 5: The Callback Method

EE 4163 / EL 6183 : Digital Signal Processing Lab

Fall 2015

1 Playing random notes

This demo program plays notes randomly and plots the waveform on the screen. This requires the Python library `matplotlib`. If you are unable to install this library, then refer instead to the demo program `play_randomly.py` which does not plot the waveform.

```
# play_randomly_plots.py
"""
PyAudio Example: Generate random pulses and input them to an IIR filter of 2nd order.
It sounds like pings from a Sonar or from a Geiger Counter.
Gerald Schuller, March 2015
Modified - Ivan Selesnick, October 2015
"""

import pyaudio
import struct
import random
from math import sin, cos, pi
from matplotlib import pyplot as plt
from myfunctions import clip16
# import numpy as np

BLOCKSIZE = 1024    # Number of frames per block
WIDTH = 2           # Bytes per sample
CHANNELS = 1
RATE = 8000         # Sampling rate in Hz

# Parameters
T = 10       # Total play time (seconds)
Ta = 0.2     # Decay time (seconds)
f1 = 350     # Frequency (Hz)

# Pole radius and angle
r = 0.01**(1.0/(Ta*RATE))       # 0.01 for 1 percent amplitude
om1 = 2.0 * pi * float(f1)/RATE

# Filter coefficients (second-order IIR)
a1 = -2*r*cos(om1)
a2 = r**2
b0 = sin(om1)

NumBlocks = T * RATE / BLOCKSIZE

y = [0 for i in range(BLOCKSIZE)]

plt.ion()           # Turn on interactive mode so plot gets updated
fig = plt.figure(1)
line, = plt.plot(y)
plt.ylim(-32000, 32000)
```

```
plt.xlim(0, BLOCKSIZE)
plt.xlabel('Time (n)')
plt.show()

# Open the audio output stream
p = pyaudio.PyAudio()
PA_FORMAT = p.get_format_from_width(WIDTH)
stream = p.open(format = PA_FORMAT,
                channels = CHANNELS,
                rate = RATE,
                input = False,
                output = True)

print 'Playing for {0:f} seconds ...'.format(T),

THRESHOLD = 2.5 / RATE          # For a rate of 2.5 impulses per second

# Loop through blocks
for i in range(0, NumBlocks):

    # Do difference equation for block
    for n in range(BLOCKSIZE):

        rand_val = random.random()
        if rand_val < THRESHOLD:
            x = 15000
        else:
            x = 0

        y[n] = b0 * x - a1 * y[n-1] - a2 * y[n-2]
              # What happens when n = 0?
              # In Python negative indices cycle to end, so it works..

        y[n] = clip16(y[n])

    line.set_ydata(y)
    plt.title('Block {0:d}'.format(i))
    plt.draw()

    # y = np.clip(y, -32000, 32000)     # Clipping using numpy if available

    # Convert numeric list to binary string
    data = struct.pack('h' * BLOCKSIZE, *y);

    # Write binary string to audio output stream
    stream.write(data, BLOCKSIZE)

print 'Done.'

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

Exercises

1. **Stereo.** Write a program based on the demo program `play_randomly.py` that produces stereo audio. The start-time should be different in each of the two channels. (The left and right channels can be independent).

2. **Stereo with plotting.** Modify your program for stereo so that it plots both channels of the signal — use a different color for left and right channels. The two waveforms in the plot maybe vertically offset

from on another to improve legibility.

3. **Random frequencies.** The notes produces by the demo program `play_randomly.py` are not very random. Only the start-times of the notes are random. Write a program that generates notes with random frequencies. In your program, you may use a discrete set of frequencies.

2 The Callback method

In previous PyAudio programs we have used `stream.read()` and `stream.write()` to read from the microphone and to write to the speaker. This is known as *blocking* mode.

Another approach is to use the *callback* method. This method does not use `stream.read()` or `stream.write()`. Instead, this method calls a user-defined *callback* function.

This demo program gives a simple example where the input is read from the microphone and written to the speaker. This examples does not implement any effect. In just plays the input from microphone in real-time. To avoid feedback, use headphones or an external microphone (otherwise you may hear an artifact that gets louder and louder).

```
# simple_wire_gain.py
# Play microphone input to speaker using callback function
#
# Like simple_wire.py, but additionally applies a gain

import pyaudio
import struct
import time
from myfunctions import clip16

CHANNELS = 1
RATE = 16000        # frames / second
gain = 4.0

# Define callback function
def my_callback_fun(input_string, block_size, time_info, status):

    N = block_size      # Number of frames

    # Convert string to tuple of numbers
    input_block = struct.unpack('h'*N, input_string)

    # Create output (initialize to zero)
    output_block = [0.0 for n in range(N)]

    for n in range(N):
        output_block[n] = clip16(gain * input_block[n])

    # Convert output values to binary string
    output_string = struct.pack('h'*N, *output_block)

    return (output_string, pyaudio.paContinue)    # Return data and status


# Create audio object
p = pyaudio.PyAudio()

# Open stream using callback
stream = p.open(format = pyaudio.paInt16,       # 16 bits/sample
                channels = CHANNELS,
                rate = RATE,
```

```
                input = True,
                output = True,
                stream_callback = my_callback_fun)

stream.start_stream()

print 'The wire will be on for 6 seconds'
# Keep the stream active for 6 seconds by sleeping here
time.sleep(6)

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

The `start_stream()` function starts the callback function being continuously called when ever a new block of audio becomes available from the input. The callback function will continue to be called until the `steop_stream()` function is invoked. The `sleep()` function keeps the program from proceeding to `stop_stream()` which will stop the callback function.

For more details about PyAudio and its callback:
https://people.csail.mit.edu/hubert/pyaudio/docs/
For more details about `time` module:
https://docs.python.org/2/library/time.html

The demo program `simple_wire_gain_stereo_AM.py` available on the course site shows how an audio effect can be implemented in the callback.

The demo program `record_and_play_vibrato.py` implements the vibrato effect using the callback method.

Exercises

1. Run and verify the demo programs on the course site:

   `simple_wire.py`
   `simple_wire_gain.py`
   `simple_wire_gain_stereo.py`
   `simple_wire_gain_stereo_AM.py`

   Also run and verify the other demo programs on the course site.

2. Modify the demo program `simple_wire_gain.py`. Use a print statement inside the callback function to print out the length of the input string and the value of `block_size`. Since we did not specify the block size, the printed value is the default value.

3. The block size can be set using the `frames_per_buffer` parameter in the `p.open()` statement. (See demo programs on the course site.) Set this value to a value different from the default value and verify by a print statement in the callback function that the input block has the specified length.

4. The demo program `play_randomly.py` does not use the callback method. Rewrite this demo program so it uses the callback method. When opening the audio stream, do not set `frames_per_buffer`. Instead, use the default block size.

5. In the demo program `simple_wire_gain.py`, there is one channel. If `CHANNELS` is set to 2, does the program work? Why or why not?

6. If we set `frame_per_buffer = 1` and `CHANNELS = 2`, then what is the value of `block_size` in the callback function?

7. Write a program to implement the vibrato effect using the callback method. The input should be from the microphone. The output should be to the speakers. (No wave files.) The parameter `frames_per_buffer` should be set to 1024 or left unspecified (default value). (Note that the parameter `frames_per_buffer` is set to 1 in many of the demo files. For this exercise, change this to `frames_per_buffer = 1024` or omit it.)

3 To Submit

Submit the following Exercises for this Lab:

Section 1) Exercise 3

Section 2) Exercises 4 and 7

Note: Submit all files as a **single** zip file to NYU Classes under the 'Lab5' assignment. All scripts, function definitions, wave files, and your written report document should be included in the zip file.

Name your main Python scripts:

`Lab5_Sec1_Ex3_NetID.py`

`Lab5_Sec2_Ex4_NetID.py`

`Lab5_Sec2_Ex7_NetID.py`

Please ensure your submitted codes can run independently of the system (Windows, Linux, etc.) and directory. Do not put absolute paths like `C:\\xxxx.wav` or `\usr\local\xxx` in your code.

4 Stereo

For stereo signals, the stream should contains interlaced samples: L R L R ....

For example, see `simple_wire_gain_stereo_AM.py` on the course site. For a stereo signal to be played correctly, the binary string written to the audio output should have this structure (L R L R ...).

5 NumPy and matplotlib

Depending on your Python installation, these libraries may need to be downloaded an installed on your computer system before you can import them into Python.

- The matplotlib library defines functions for plotting.

  http://matplotlib.org

- The NumPy library defines an array data type and allows for vector operations (instead of loops).

  http://www.numpy.org

  Tutorials about NumPy:

  http://cs231n.github.io/python-numpy-tutorial/

  http://www.python-course.eu/numpy.php