Lab 3: Delay Effect and Circular Buffers

EE 4163 / EL 6183 : Digital Signal Processing Lab

Fall 2015

1 Play a wave file using PyAudio

This demo program shows how to read and play a wave file using PyAudio. It is assumed the wave file is mono (single channel). A program to play a stereo (two channel) wave file is available on the course web page.

```
# play_wav_mono.py

import pyaudio
import wave
import struct
import math

def clip16( x ):
    # Clipping for 16 bits
    if x > 32767:
        x = 32767
    elif x < -32768:
        x = -32768
    else:
        x = x
    return int(x)

gain = 0.5

# wavfile = 'author.wav'
wavfile = 'sin01_mono.wav'
# wavfile = 'sin01_stereo.wav'

print("Play the wave file %s." % wavfile)

wf = wave.open( wavfile, 'rb' )

# Read the wave file properties
num_channels = wf.getnchannels()        # Number of channels
Fs = wf.getframerate()                  # Sampling rate (frames/second)
signal_length  = wf.getnframes()        # Signal length
width = wf.getsampwidth()               # Number of bytes per sample

print("The file has %d channel(s)."            % num_channels)
print("The frame rate is %d frames/second."    % Fs)
print("The file has %d frames."                % signal_length)
print("There are %d bytes per sample."         % width)

```

```python
p = pyaudio.PyAudio()

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

input_string = wf.readframes(1)          # Get first frame

while input_string != '':

    # Convert string to number
    input_tuple = struct.unpack('h', input_string)  # One-element tuple
    input_value = input_tuple[0]                    # Number

    # Compute output value
    output_value = clip16(gain * input_value)    # Number

    # Convert output value to binary string
    output_string = struct.pack('h', output_value)

    # Write output value to audio stream
    stream.write(output_string)

    # Get next frame
    input_string = wf.readframes(1)

print("**** Done ****")

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

Note that, in this example, we use a `while` loop to write a `string` to the `stream`, and inside the loop, we use `unpack()` and `pack()` methods to translate between a binary string and a list of integers. Note that in the `while` loop, we use a gain variable to amplify the signal and a self-defined function `clip16()` to keep the value within the range of a signed 16-bit integer to avoid possible overflow run-time errors. Correspondingly the strings `'h'` (`'hh'` for stereo) are used in the `unpack()` and `pack()` methods to set the encoding format.

Documentation for the `wave` module is at:

https://docs.python.org/2/library/wave.html

1.1 Exercises

1. **Single program for mono and stereo.** Write a single Python program for playing both mono and stereo wave files. The program should determine the number of channels by reading the wave file information.

   (a) Create stereo (two-channel) wave file of your own voice at 16-bits per sample.

   (b) Verify your program can play both mono and stereo wave files encoded with 16-bits per samples.

   (c) Write a program so that it can be used at the command line like

   ```
   >> python mywave_play.py filename.wav
   ```

For this, you will need to import the `sys` module. For example, if the file `demo_sys.py` is:

```
# demo_sys.py
import sys
print 'Argument 0 is', sys.argv[0]
if len(sys.argv) > 1:
    print 'Argument 1 is', sys.argv[1]
```

then at the terminal command line we get:

```
>> python demo_sys.py
Argument 0 is demo_sys.py

>> python demo_sys.py aaa.wav
Argument 0 is demo_sys.py
Argument 1 is aaa.wav
```

2. Write a Python program that can play wave files with 8, 16, or 32 bits per sample.

   (a) Note that `'h'` is used to set the encoding scheme to signed 16-bit integer in `unpack()` and `pack()` methods. What letters should be used for (1) signed 32-bit integer, and (2) unsigned 8-bit integer?

   (b) What are the ranges of signed 32-bit integer and unsigned 8-bit integer? Write your own clipping functions to avoid run-time overflow errors while playing wav files of signed 32-bit integer and unsigned 8-bit integer.

   (c) Modify the example program so that it can play a mono wave files with any of (1) unsigned 8-bit integer, (2) signed 16-bit integer, or (3) signed 32-bit integer.

2 Basic delay effects

Reading: First sections of Chapter 2 of text book.

Several demo programs, shown in class, are available of the course web page. The following demo program implements the basic delay (with no feedback). An important point is the use of a circular buffer in the implementation. The delay and gain parameters are set within the program.

```
# play_delay.py
# Reads a specified wave file (mono) and plays it with a delay.
# This implementation uses a circular buffer.

import pyaudio
import wave
import struct
import math
from myfunctions import clip16

wavfile = "author.wav"
print("Play the wave file %s." % wavfile)

# Open the wave file
wf = wave.open( wavfile, 'rb')

# Read the wave file properties
```

```
num_channels = wf.getnchannels()        # Number of channels
Fs = wf.getframerate()                  # Sampling rate (frames/second)
signal_length  = wf.getnframes()        # Signal length
width = wf.getsampwidth()               # Number of bytes per sample

print("The file has %d channel(s)."            % num_channels)
print("The frame rate is %d frames/second."    % Fs)
print("The file has %d frames."                % signal_length)
print("There are %d bytes per sample."         % width)

# Set parameters of delay system
gain = 1
gain_delay = 0.8
delay_sec = 0.05 # 50 milliseconds
delay_samples = int( math.floor( Fs * delay_sec ) )

print('The delay of {0:.3f} seconds is {1:d} samples.'.format(delay_sec, delay_samples))

# Create a buffer to store past values. Initialize to zero.
buffer = [ 0 for i in range(delay_samples) ]

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


# Get first frame (sample)
input_string = wf.readframes(1)

k = 0       # buffer index (circular index)

print ("**** Playing ****")

while input_string != '':

    # Convert string to number
    input_value = struct.unpack('h', input_string)[0]

    # Compute output value
    output_value = gain * input_value + gain_delay * buffer[k]
    output_value = clip16(output_value)

    # Update buffer
    buffer[k] = input_value
    k = k + 1
    if k >= delay_samples:
        k = 0

    # Convert output value to binary string
    output_string = struct.pack('h', output_value)

    # Write output value to audio stream
    stream.write(output_string)

    # Get next frame (sample)
    input_string = wf.readframes(1)

print("**** Done ****")

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

2.1 Exercises

1. What are the poles of the basic feedforward delay system with a delay of N samples? Are there any parameter settings that make the system unstable? Explain.

2. Experiment with different delay and gain parameters. How do short delays (e.g., less than 50 milliseconds) and long delays (e.g., longer than 0.2 seconds) sound different?

3. Modify the program so that it produces a stereo output, with a different delay in each of the two channels.

4. Note that the provided demo program truncates the output audio before it is finished. It is not noticeable for short delays or for wav files ending with a sufficiently long period of silence, but it is noticeable for long delays for some wave files. Modify the demo program so that the output signal is not truncated at the end (i.e., so that the trailing end of the final echo is played.)

3 Delay with feedback

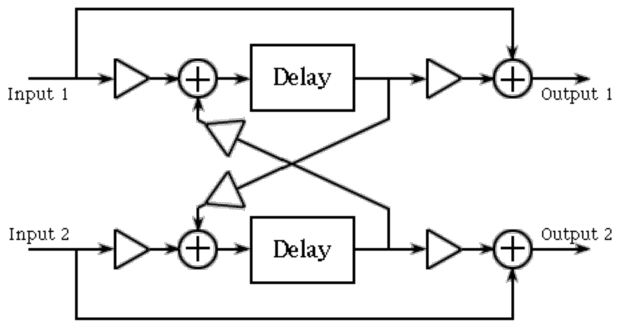
The following demo program implements a delay with a feedback loop. The difference equation of the system is recursive. This demo program implements a circular buffer of minimal length with one index. Other demo programs, available on the course web page, show how to implement a circular buffer of longer length with more than one index. That type of implementation is useful when the delay is time-varying.

```
# play_feedbackdelay1.py
# Reads a specified wave file (mono) and plays it with a delay with feedback.
# This implementation uses a circular buffer (of minimum
# length) and one buffer index.

import pyaudio
import wave
import struct
import math
from myfunctions import clip16

wavfile = "author.wav"
print("Play the wave file %s." % wavfile)

# Open the wave file
wf = wave.open( wavfile, 'rb')

# Read the wave file properties
num_channels = wf.getnchannels()        # Number of channels
Fs = wf.getframerate()                  # Sampling rate (frames/second)
signal_length  = wf.getnframes()        # Signal length
```

```python
width = wf.getsampwidth()                # Number of bytes per sample

print("The file has %d channel(s)."            % num_channels)
print("The frame rate is %d frames/second."    % Fs)
print("The file has %d frames."                % signal_length)
print("There are %d bytes per sample."         % width)

# Set parameters of delay system
Gfb = 1.5        # feed-back gain
Gdp = 0.8        # direct-path gain
Gff = 0.3        # feed-forward gain
# Gff = 0.0          # feed-forward gain (set to zero for no effect)

delay_sec = 0.08 # 50 milliseconds
delay_samples = int( math.floor( Fs * delay_sec ) )

print('The delay of {0:.3f} seconds is {1:d} samples.'.format(delay_sec, delay_samples))

# Create a delay line (buffer) to store past values. Initialize to zero.
buffer_length = delay_samples
buffer = [ 0 for i in range(buffer_length) ]

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

# Get first frame (sample)
input_string = wf.readframes(1)

# Delay line (buffer) index
k = 0

print ("**** Playing ****")

while input_string != '':

    # Convert string to number
    input_value = struct.unpack('h', input_string)[0]

    # Compute output value
    output_value = Gdp * input_value + Gff * buffer[k];

    # Update buffer
    buffer[k] = input_value + Gfb * buffer[k]

    # Increment buffer index
    k = k + 1
    if k == buffer_length:
        # We have reached the end of the buffer. Circle back to front.
        k = 0

    # Clip output value to 16 bits and convert to binary string
    output_string = struct.pack('h', clip16(output_value))

    # Write output value to audio stream
    stream.write(output_string)

    # Get next frame (sample)
    input_string = wf.readframes(1)

print("**** Done ****")

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

3.1 Exercises

1. What are the poles of the delay system with feedback of N samples? Are there any parameter settings that make the system unstable? Explain.
2. The demo programs provide two implementations of the circular buffer: one program uses a minimal length buffer and one buffer index, the other program uses a longer buffer and multiple buffer indices. They give identical output signals, even though the circular buffer is implemented differently. Explain.
3. Implement 'ping-pong' delay described in text book (Figure 2.4 in textbook). (Combine the delay effect with the stereo example.) Use headphones or a pair of stereo speakers to test your program.



4 To Submit

Submit the following Exercises for this Lab:
1.1) Exercise 1
2.1) Exercises 1 and 4
3.1) Exercises 1 and 3
Submit each exercise as a separate attachment to NYU Classes under the 'Lab3' assignment.