Lab 6: FFT / Complex AM / Room Response

EE 4163 / EL 6183 : Digital Signal Processing Lab

Fall 2015

1 FFT

Several demo programs show how to use the FFT in Python using the NumPy library. The following program shows the 'live' spectrum of audio read from the microphone.

```
# plot_micinput_spectrum.py

"""
Using Pyaudio, get audio input and plot real-time FFT of blocks.
Ivan Selesnick, October 2015
Based on program by Gerald Schuller
"""

import pyaudio
import struct
import numpy as np
from matplotlib import pyplot as plt

plt.ion()           # Turn on interactive mode so plot gets updated

WIDTH = 2           # bytes per sample
CHANNELS = 1        # mono
RATE = 16000        # Sampling rate (samples/second)
BLOCKSIZE = 1024
DURATION = 10       # Duration in seconds

NumBlocks = int( DURATION * RATE / BLOCKSIZE )

print 'BLOCKSIZE =', BLOCKSIZE
print 'NumBlocks =', NumBlocks
print 'Running for ', DURATION, 'seconds...'

# Initialize plot window:
plt.figure(1)
plt.ylim(0, 10*RATE)

# plt.xlim(0, BLOCKSIZE/2.0)         # set x-axis limits
# plt.xlabel('Frequency (k)')
# f = np.linspace(0, BLOCKSIZE-1, BLOCKSIZE)

# # Time axis in units of milliseconds:
plt.xlim(0, RATE/2.0)         # set x-axis limits
plt.xlabel('Frequency (Hz)')
f = [n*float(RATE/BLOCKSIZE) for n in range(BLOCKSIZE)]

```

```
line, = plt.plot([], [], color = 'blue')  # Create empty line
line.set_xdata(f)                         # x-data of plot (frequency)

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

for i in range(0, NumBlocks):
    input_string = stream.read(BLOCKSIZE)                    # Read audio input stream
    input_tuple = struct.unpack('h'*BLOCKSIZE, input_string)  # Convert
    X = np.fft.fft(input_tuple)
    line.set_ydata(abs(X))                                   # Update y-data of plot
    plt.draw()
# plt.close()

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

print '* Done'
```

Exercises

1. **NumPy.** Install the NumPy library and run the demo program `plot_micinput_spectrum.py`. Experiment with changing the block size and and sampling rate. How do these parameters affect the spectrum?

2. **Logarithmic scale.** Modify the above program so that the spectrum is displayed on a log-log scale. A logarithmic scale helps to show a wider range of values (on a linear scale, small values can be hard to see.) For the vertical log scale, use dB (i.e., $20 \log_{10} |X|$).

   Your program should also print one of the spectra, displayed during the running of the program, to a pdf file.

2 Complex Amplitude Modulation

In past demo and lab assignments, we uses amplitude modulation (AM) to affect a speech signal. This method computes the output signal as

$$y(t) = x(t) \cos(2\pi f_1 t) \tag{1}$$

This AM method can lead to overlapping of the spectrum with itself. To shift a speech signal to a higher frequency without the spectrum overlapping itself (as in Figure 1), we can use complex AM. This method was shown in class and in the Matlab demo programs (in the folder `complexAM_demos`) on the course web page.

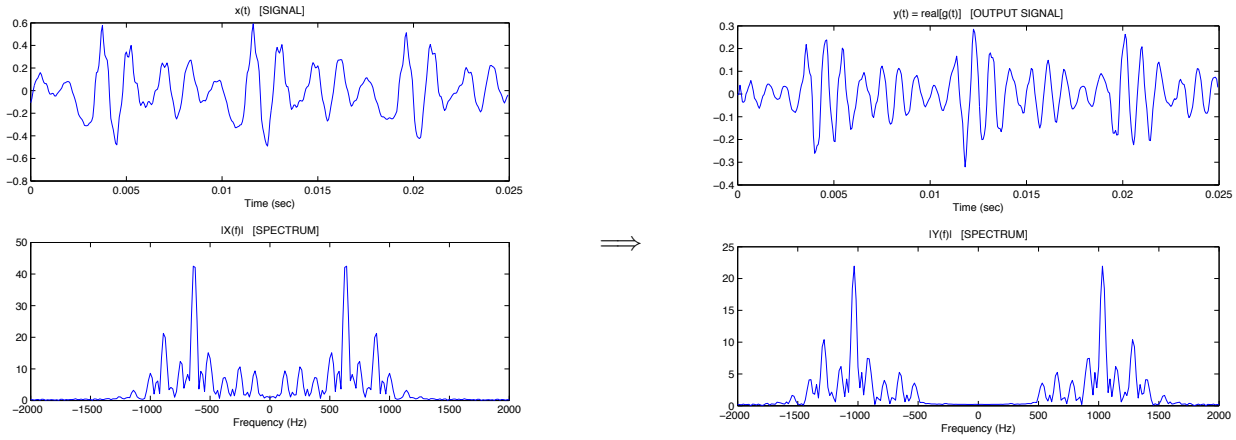

Figure 1: Complex AM

Exercises

1. Using an input wave file of your choice, use Matlab to implement complex AM (like in the Matlab demo file).

2. Use Python to obtain the same result as in Matlab using the same input wave file as in the preceding part. Your Python output should be the same as your Matlab output.

3. **Real-time complex AM.** Implement real-time complex AM in Python with PyAudio. Your program should take the microphone signal as input and produce an output audio signal (on speakers or headphones). Compare the sound with the simple AM method implemented earlier in the course (equation (1)). It is best to use headphones or earbuds because the speakers on a laptop might not be sufficiently good quality to hear the difference properly.

3 Room Response

The physical environment (room, concert hall, outdoors, etc.) can have a significant impact on the perceptual quality of sound. One way to measure the acoustic effect of a room or other environment is by the room impulse response (or 'room response'). One simple approach to approximately measure the room response is by popping a balloon or generating another form of acoustic impulse.[1]

Exercises

1. Write a Python program that records the microphone input to a wave file. The recording should start only when the input signal exceeds a prescribed threshold value. The recording should last for a prescribed time duration.

   Use the preceding program to measure the room response by popping a balloon and recording the sound it produces to a wave file. Print a plot of the room response to a pdf file. You should measure

[1] A more accurate robust way to measure the room response is to use a frequency sweep.

the room responds in at least two different acoustic environments. (One may be outdoors.) Comment on your observations.

It may be useful to use a high sampling rate for this purpose. What is the highest sampling rate your system allows? Common sampling rates are: 44.1 KHz, 48 KHz, and 96 KHz.

4 To Submit

Submit the following Exercises for this Lab:

Section 1) Exercise 2. Include a pdf file of a plot of a spectrum.

Section 2) Exercise 3.

Section 3) Exercise 1. Submit plots (in pdf file format) of the room response (at least two).

Note: Submit all files as a **single** zip file to NYU Classes under the 'Lab6' 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_NetID.py`

`Lab5_Sec2_NetID.py`

`Lab5_Sec3_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.