CS168 Spring Assignment 6

SUNet ID(s): johngold jsalloum
Name(s): John Gold Justin Salloum
Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Part 1: Fourier Transform and Convolution

(a) Since q is a (100N - 99) tuple (N=6 in this case, so a 501 tuple) according to the definition of f * g, q[150] represents the probability that the sum of 100 die rolls will add up to exactly 151.

(b) Proof:

$$\mathrm{F}(f * g) = \mathrm{F}f^{+} \cdot \mathrm{F}g^{+}$$

$$\sum_{k=0}^{2N-1} \mathrm{F}(f * g)[\mathrm{k}] = \sum_{k=0}^{2N-1} (\mathrm{F}f^{+} \cdot \mathrm{F}g^{+})[\mathrm{k}]$$

$$\mathrm{F}\sum_{k=0}^{2N-1} (f * g)[\mathrm{k}] = \sum_{k=0}^{2N-1} (\mathrm{F}f^{+})[\mathrm{k}] \cdot \sum_{k=0}^{2N-1} (\mathrm{F}g^{+})[\mathrm{k}]$$

$$\mathrm{F}\sum_{k=0}^{2N-1}\sum_{j=0}^{N-1} \mathrm{f[j]g[k\text{-}j]} = \sum_{k=0}^{2N-1} (\mathrm{F}f^{+})[\mathrm{k}] \cdot \sum_{k=0}^{2N-1} (\mathrm{F}g^{+})[\mathrm{k}]$$

$$\mathrm{F}\sum_{k=0}^{2N-1}\sum_{j=0}^{N-1} \mathrm{f[j]g[k\text{-}j]} = \mathrm{F}\sum_{k=0}^{2N-1} (f^{+})[\mathrm{k}] \cdot \mathrm{F}\sum_{k=0}^{2N-1} (g^{+})[\mathrm{k}]$$

$$\mathrm{F}(\sum_{k=0}^{2N-1}\sum_{j=0}^{N-1} \mathrm{f[j]g[k\text{-}j]}) = \mathrm{F}(\sum_{k=0}^{2N-1} (f^{+})[\mathrm{k}] \cdot \sum_{k=0}^{2N-1} (g^{+})[\mathrm{k}])$$

$$\mathrm{F}(\sum_{k=0}^{2N-1}\sum_{j=0}^{N-1} \mathrm{f[j]g[k\text{-}j]}) = \mathrm{F}(\sum_{k=0}^{2N-1} (f^{+})[\mathrm{k}](g^{+})[\mathrm{k}])$$

Since the only non-zero elements of f^{+} and g^{+} are in the indicies less than N-1:

$$\mathrm{F}(\sum_{k=0}^{2N-1}\sum_{j=0}^{N-1} \mathrm{f[j]g[k\text{-}j]}) = \mathrm{F}(\sum_{k=0}^{2N-1}\sum_{j=0}^{N-1} \mathrm{f[j]g[k\text{-}j]})$$

The analogous conclusion is that we can pad the shorter one to be the same length as the longer one.

(c) Code:

```
    #Uses FFT and inverse FFT to multiple 2 arrays of numbers
    #each of those numbers represents a digit
def multiply(x, y):
```

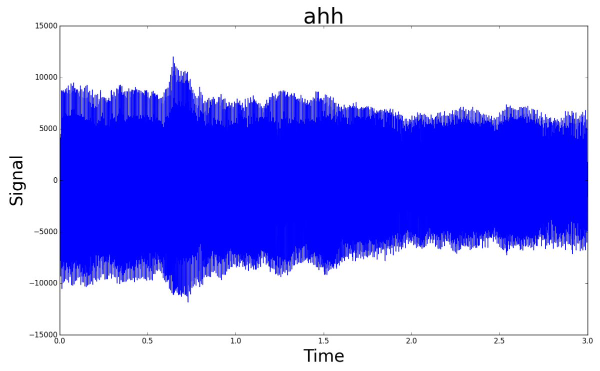
```
#Pad with 0s
two_n = max(len(x), len(y)) * 2
while(len(y) < two_n):
    y.append(0)
while(len(x) < two_n):
    x.append(0)

#Take 'convolution' using FFT/IFFT, then round/cast to ints
conv = np.absolute(np.fft.ifft(np.fft.fft(x)*np.fft.fft(y)))
conv = [int(round(x)) for x in conv]

#convert back to format required by problem
answer = 0
for index, num in enumerate(conv):
    answer += np.power(10, index) * num
return [int(i) for i in str(answer)][::-1]
```

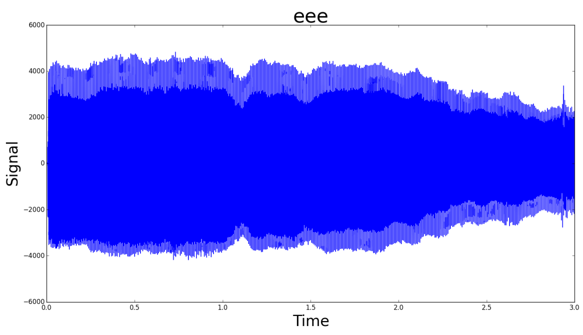
Runtime Analysis: For the naive implementation, this takes O($n * m$) time, WLOG assume m is the length of the longer vector(or integer). However, our method takes 2 FFTs, both of size m, which takes O($mlogm$), and one IFFT, also O($mlogm$). So basically we can go from O(nm) to O($mlogm$), a major improvement.

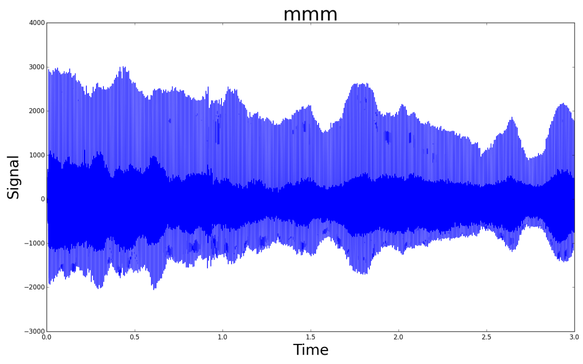
(d) Bonus:

Part 2: The Sound of Fourier Transform

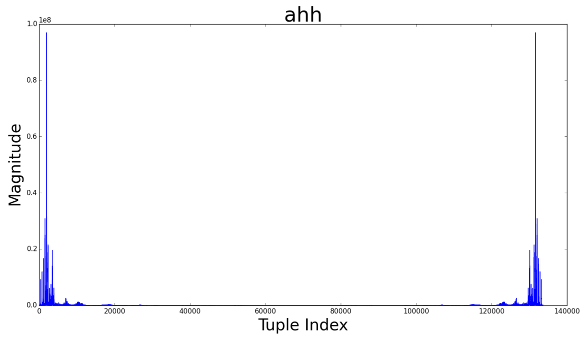
(a) Plots:



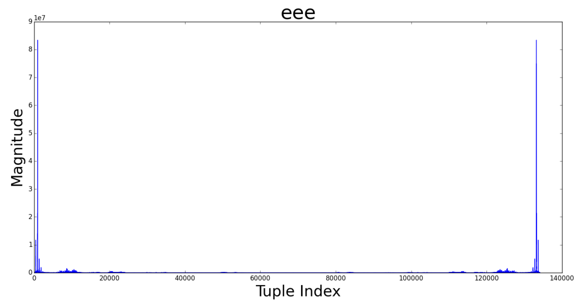




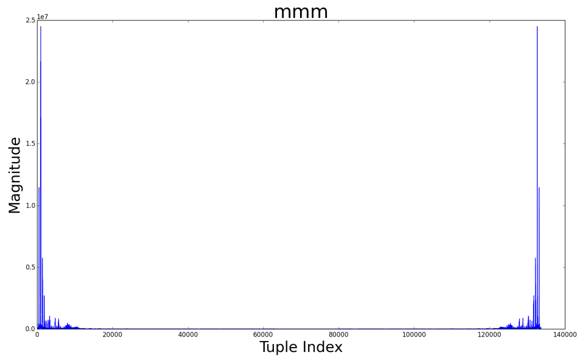
(b) Plots:



The dominant frequency was 648 Hz



The dominant frequency was 303 Hz



The dominant frequency was 315 Hz

(c) Discussion: The main differences in these sounds are the magnitudes. The magnitude of "ahh" is similar to "eee", but 4 times larger than that of the "mmm". Additionally, the dominant frequency of "ahh" was over 600 Hz and "eee" and "mmm" are around 300 Hz. Different vowels/syllables and mouth formations produce a variety of frequencies that resonate differently even when sung at roughly the same pitch.

Part 3: Creating Echoes

(a) Warmup

(b) **f1** is the vector [1, 0, 0..., .5, 0, 0...0, 0] where .5 is at the position .25 * sample rate.

(c) **f2** is the vector [1, 0, ..., 0, 1/3, 0, ..., 0, 1/4, 0, ..., 0, 1/8, 0, ... 0] where 1/3 is at the position (2*18/344)*sampleRate, 1/4 is at the position (2*40/344)*sampleRate, and 1/8 is at the position (2*53/344)*sampleRate.

(d) We can use **f2** to create a filter that takes into account the echo of echos. Let's assume that anything coming off the fourth wall sounds half as loud as it did before hitting the wall. Given **f2**, we can convolve that with [1, 0, 0, 0, 0, .5, 0..0] and that convolved with our original signal will give us the original echo and the echo of echos effect. Since our back is to the wall, assume each of the original echos is heard ever so slightly after the first echo, thus the small shift of the .5.

To get the echo of echo of echos, we do something similar, and that will involve convolving the echo of echos with the original inital echos **f2** to take into account the distances of the 3 walls.

(e) Bonus:

```
    Code Appendix:
    #p6.py

import sys
import scipy.io.wavfile as wavfile
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
from scipy import signal

def main():
    if(sys.argv[1] == '1'):
        x = [0, 1, 2, 3]
        y = [4, 5, 6]
        print multiply(x,y)
    if(sys.argv[1] == '2'):
        part2()
    if(sys.argv[1] == '3'):
        part3()


#More sounds
def part3():
    sampleRate, sentence = get_sound_data("sentence.wav")
    #print sampleRate

    data = sentence * 1.0 / np.max(np.abs(sentence))
    #print data

    conv_filter = np.zeros_like(data)
    conv_filter[0] = 1
    conv_filter[sampleRate*.25] = .5
    #print conv_filter
    #print conv_filter.shape

    conv_filter2 = np.zeros_like(data)
    conv_filter2[0] = .001
    conv_filter2[sampleRate*36/344] = 1/3.0
    conv_filter2[sampleRate*80/344] = 1/4.0
    conv_filter2[sampleRate*106/344] = 1/8.0


    echo1 = signal.fftconvolve(data, conv_filter)
    echo2 = signal.fftconvolve(data, conv_filter2)

    with open("echo1.wav", 'w') as f:
        wavfile.write(f, sampleRate, echo1)
    with open("echo2.wav", 'w') as f:
        wavfile.write(f, sampleRate, echo2)

#Soundssss
def part2():
    data = None
    if(sys.argv[2] == "ahh"):
        data = get_sound_data("ahh.wav")

    elif(sys.argv[2] == "eee"):
        data = get_sound_data("eee.wav")

    elif(sys.argv[2] == "mmm"):
        data = get_sound_data("mmm.wav")


    #Part A
    if(sys.argv[3] == 'a'):
        plt.ylabel('Signal', fontsize = 28)
        plt.xlabel('Time', fontsize = 28)
        time = np.linspace(0, len(data[1])/data[0], num=len(data[1]))
        plt.plot(time, data[1])


    #Part B
    elif(sys.argv[3] == 'b'):
        plt.ylabel('Magnitude', fontsize = 28)
        plt.xlabel('Tuple Index', fontsize = 28)
        print data[1]

        fourier_mag = np.absolute(np.fft.fft(data[1]))
        print fourier_mag

        plt.plot(fourier_mag)

        #Find max frequency
        freqs = np.fft.fftfreq(len(fourier_mag))
        idx = np.argmax(fourier_mag)
        max_freq = freqs[idx]
        freq_in_hz = abs(max_freq * data[0])
        print freq_in_hz


    plt.title(sys.argv[2], fontsize = 36)
    plt.show()


def get_sound_data(filename):
    with open(filename, 'r') as f:
        sampleRate, data = wavfile.read(f)
        if len(data.shape) == 2 and data.shape[1] == 2:
            data = data[:,1]
    return (sampleRate, data)


#Uses FFT and inverse FFT to multiple 2 arrays of numbers
#each of those numbers represents a digit
def multiply(x, y):
    #Pad with 0s
    two_n = max(len(x), len(y)) * 2
    while(len(y) < two_n):
        y.append(0)
    while(len(x) < two_n):
        x.append(0)

    #Take 'convolution' using FFT and IFFT, then round and cast to ints
    conv = np.absolute(np.fft.ifft(np.fft.fft(x) * np.fft.fft(y)))
    conv = [int(round(x)) for x in conv]

    #convert back to format required by problem
    answer = 0
    for index, num in enumerate(conv):
        answer += np.power(10, index) * num
    return [int(i) for i in str(answer)][::-1]


if __name__ == "__main__":
    main()
```