CS168, Spring 2015

Mini-Project #6

Due by 11:59 PM on Tuesday, May 12th.

Instructions

- You can work individually or with one partner. If you work in a pair, both partners will receive the same grade.
- If you have written code to solve a certain part of a problem, or if the part explicitly asks you to implement an algorithm, you must also include the code in your pdf submission. Unless otherwise stated, the code for all parts should go in an appendix. This is a change from the first few assignments, where the code was embedded in each part.
- Make sure plots you submit are easy to read at a normal zoom level.
- Detailed submission instruction can be found on the course website (`http://cs168.stanford.edu`) under the Coursework - Assignment section. If you work in pairs, only one member should submit all of the relevant files.
- **Reminder:** No late assignments will be accepted, but we will drop your lowest mini-project grade when calculating your final grade.

Part 1: Fourier Transform and Convolution

Let $\mathbf{f} = (\mathbf{f}[0], \mathbf{f}[1], \ldots, \mathbf{f}[N-1])$ be an N-tuple. The *discrete Fourier transform* (DFT) of $\mathbf{f}$ is the N-tuple $\mathbf{F} = (\mathbf{F}[0], \mathbf{F}[1], \ldots, \mathbf{F}[N-1])$ defined by

$$\mathbf{F}[m] = \sum_{j=0}^{N-1} \mathbf{f}[j] e^{-2\pi i m j/N}, \ m = 0, 1, \ldots, N-1$$

For notational convenience, we denote $\mathcal{F}$ to be the DFT operator (and $\mathcal{F}^{-1}$ be the inverse operator), so that $\mathbf{F} = \mathcal{F}\mathbf{f}$ (and $\mathbf{f} = \mathcal{F}^{-1}\mathbf{F}$).

A concept that is closely related to DFT is *(linear) convolution*. Given a N-tuple $\mathbf{f}$ and a M-tuple $\mathbf{g}$, define their convolution $\mathbf{f} * \mathbf{g}$ to be a $(M+N-1)$ tuple: [1]

$$(\mathbf{f} * \mathbf{g})[k] = \sum_{j=0}^{N-1} \mathbf{f}[j]\mathbf{g}[k-j]$$

Exercises:

(a) Given a six-sided die, let p be a 6-tuple where $p[i]$ denotes probability that $i+1$ appears. Let $q = \underbrace{p * p * \cdots * p}_{100 \text{ times}}$. What event occurs with probability $q[150]$?

(b) Given an N-tuple $\mathbf{f}$, define $\mathbf{f}^+$ to be the $(2N-1)$-tuple obtained by padding $\mathbf{f}$ with zeros:

$$\mathbf{f}^+[i] = \begin{cases} \mathbf{f}[i] & \text{if } 0 \le i \le N-1 \\ 0 & \text{if } N \le i \le 2N-2 \end{cases}$$

[1] We define $\mathbf{g}[k-j] = 0$ when $k-j < 0$ or $k-j \ge M$.

Verify that convolution is multiplication under the Fourier transform: for any two N-tuples $\mathbf{f}$ and $\mathbf{g}$, show that

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

where $\cdot$ denotes element-wise multiplication. Suggest an implementation of convolution using the Fast Fourier Transform and its inverse transform. What is the analogous conclusion you can make when the two tuples have different lengths?

(c) Using the Fast Fourier Transform (and its inverse transform), write a method $multiply(x, y)$ that takes in two arrays of digits, each representing an integer (lower indices represent lower digits), and return an array that represents the product of the two integers. For example, the output of $[0, 1, 2, 3]$ and $[4, 5, 6]$ should be $[0, 4, 3, 9, 9, 0, 2]$, representing the fact that $3,210 \times 654 = 2,099,340$. Compare its asymptotic run-time to that of the naive grade-school integer multiplication algorithm. Feel free to refer to the runtime of the Fast Fourier Transform algorithm that we discussed in class. [For this part, please embed your code in the solution, instead of including it in the appendix. Do not directly use convolution functions in your code.]

(d) Bonus: For your implementation for part (c), roughly how large can the inputs be for the method to return accurate results? Justify your response via experiments and a discussion of the Fourier transform and the known limitations of your program environment (i.e. number of significant bits stored, etc.).

Deliverables: answer for (a); calculation, suggestion, and analogous conclusion for (b); code and discussion for (c); discussion for (d).

Part 2: The Sound of Fourier Transform

In this part, we explore how Fourier transform can help us understand human sounds. You'll need to record yourself, and transform the audio signal into an array which you can analyze. You'll also need to know the sample rate of the recordings. In Matlab, you can do this by:

```
recObj = audiorecorder; % recorder configuration
recordblocking(recObj, 3); % records 3 seconds of audio
data = getaudiodata(recObj); % the vector representing the audio
sampleRate = 8000; % default sample rate
```

If you're working with Python, first save your recording as a `.wav` file, and then load it into Python: [2]

```
import scipy.io.wavfile as wavfile
import numpy as np

with open("audio.wav", "r") as f:
  sampleRate, data = wavfile.read(f)
  if len(data.shape) == 2 and data.shape[1] == 2:
      data = data[:,1] # remove the second channel if exists
```

Exercises:

(a) Record yourself making the following sounds for at least three seconds each:[3]

- The Dentist Check: open your mouth wide and make an "Ahhhhhh" sound.
- The E: make the sound of the letter "Eeeeeee."
- The Skeptic: close you mouth and make a "Mmmmmm" sound.

For each sound, plot the signal against time, and zoom in to the point where you can see the waveforms.

[2] There seems to be a bug in the way the **scipy.io.wavfile** package processes some PPC **.wav** files. If **read()** throws an exception in your code, try to save the file in a different architecture, or use the Python default **wave** package.

[3] With your best effort, try to make it consistent in volume and pitch, but don't worry about it too much.

(b) Take the Fourier Transform of each signal. Plot the **magnitude** of the elements in the Fourier Transform against the tuple index. You should get three (almost) symmetric plots. What is the dominant frequency (the frequency with the largest magnitude) of each sound? [Hint: you'll need to use the sample rate to calculate the dominant frequencies.]

(c) Making references to the Fourier transforms of the signals, what are the main differences between these sounds? Why do you think the transforms of the "Ahhhh" sound and the "Mmmmm" sound differ in this way?

Deliverables: Plots for (a); plots and calculation for (b); discussion for (c).

Part 3: Creating Echoes

In this part, you will try to create echoes and other transforms of your own recording. In addition to recording yourself, you'll need to be able to listen to the sound represented by an array. The Matlab command `soundsc(data, sampleRate)` allows you to do that; if you're using Python, you'll need to save the signal array as a `.wav` file and then listen to the file:

```
data = data * 1.0 / np.max(np.abs(data)) # scaling
with open("output.wav", "w") as f:
  wavfile.write(f, sampleRate, data)
```

Exercises:

(a) (Warm-up, do not submit.) Record yourself saying a sentence, such as "Fast Fourier Transform is magical." Denote the signal by $\mathbf{f}$.

(b) We will now add an echo: what filter (i.e. vector) $\mathbf{f_1}$ has the property that $\mathbf{f} * \mathbf{f_1}$ consists of the original recording, over-layed with an echo of delay 0.25 seconds at half the volume? Make sure you listen to the signal—it should sound like the original recording but with an echo that is half as loud as the main recording.

(c) Now imagine you are in a large room with sound-reflecting walls (maybe a cathedral): suppose you are up against one of the walls, and there are 3 other walls, at distances 18 meters, 40 meters, and 53 meters. Assume the closest wall sends back a reflection that is 1/3rd as loud as the original sound, the second wall's reflection is 1/4th as loud, and the furthest wall is 1/8th as loud. Assuming that the speed of sound is 344 meters per second, what filter $\mathbf{f_2}$ would you use to simulate the echoes created by the walls?

(d) Now, recall that your back is against the fourth wall–we should also take into account the fact that the echoes will themselves bounce of the wall and have their own echoes. How would you use $\mathbf{f_2}$ to create the effect of the echoes of echoes? [Hint: define a filter in terms of $\mathbf{f_2}$ that you would convolve the original signal by to obtain the desired signal with the echoes of the echoes.] What about taking into account the echoes of the echoes of the echoes? Listen to the recording, and play around with the distances, and dampening factors.

(e) Bonus: In echoes, higher frequencies are often dampened more than lower frequencies. How do you create a signal that addresses this effect?

Deliverables: $\mathbf{f_1}$ and your derivation for (b); $\mathbf{f_2}$ and your derivation for (c); discussion for (d); discussion for (e).