CS168, Spring 2021

# Mini-Project #8

Due by 11:59 PM on **Tuesday**, May 25th.

## Instructions

- You can work in groups of up to four students. If you work in a group, please submit one assignment via Gradescope (with all group members' names).
- Detailed submission instructions can be found on the course website (`https://web.stanford.edu/class/cs168`) under "Coursework - Assignments" section.
- Use 12pt or higher font for your writeup.
- Make sure the plots you submit are easy to read at a normal zoom level.
- If you've 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. You do not need to mark the pages having the code when you mark pages containing answers to questions on Gradescope.
- Code marked as Deliverable should be pasted into the relevant section. Keep variable names consistent with those used in the problem statement, and with general conventions. No need to include import statements and other scaffolding, if it is clear from context. Use the `verbatim` environment to paste code in LaTeX.

```
def example():
     print "Your code should be formatted like this."
```

- **Reminder:** No late assignments will be accepted, but we will drop your lowest assignment grade.

1. (2 points) Recall that your "final miniproject" is due before the end of the quarter. Please do start thinking about it. To get the 2 points for this part, please spend at least 5 minutes thinking about it, and write a very quick summary of where you are: e.g. "I'm deciding between a research miniproject on generalization in deep networks, and doing an 'alternate' version of the SVD miniproject that has a significant component on tensor methods". If you've already finished your final miniproject, just say so : )

2. **Fourier Transforms.** Recall that the discrete Fourier transform of a length $N$ vector/signal $\mathbf{f}$ is a length $N$ complex-valued vector/signal $\mathbf{F}$ whose $m$th coordinate is defined to be

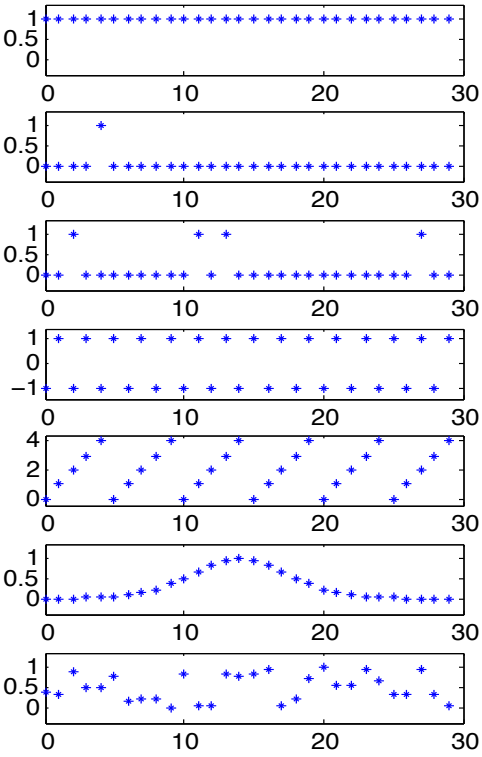
$$\mathbf{F}[m] = \sum_{j=0}^{N-1} \mathbf{f}[j] e^{-2\pi i m j/N}, \ m = 0, 1, \ldots, N-1$$

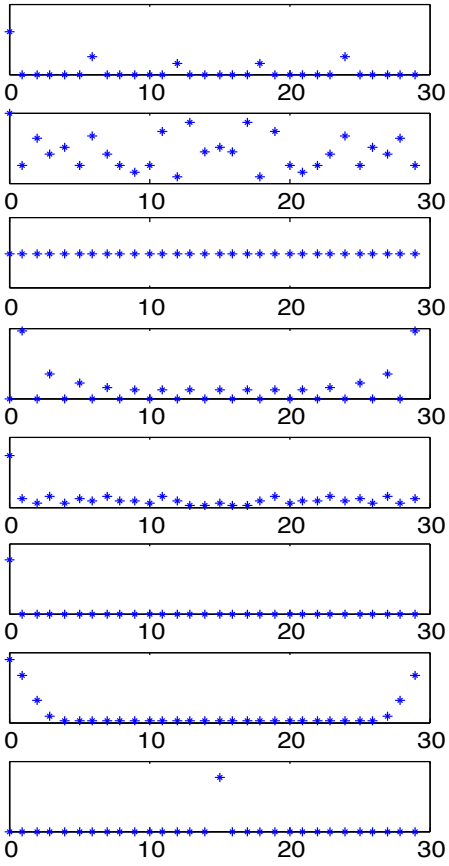
We denote the Fourier transform of a vector $\mathbf{f}$ by $\mathcal{F}\mathbf{f}$, and use $\mathcal{F}^{-1}$ to denote the inverse transformation. Hence for $\mathbf{F}$ as defined above, we have $\mathbf{F} = \mathcal{F}\mathbf{f}$ and $\mathbf{f} = \mathcal{F}^{-1}\mathbf{F}$.

(14 points) For each vector/signal depicted in the left column, 1) draw a line connecting it with the plot on the right of the magnitude of its (discrete) Fourier transform, and 2) write one sentence justifying your answer in a high-level intuitive way (e.g. "The Fourier transform of random noise is usual BLAH, and the plot at left looks like random noise and the plot at right looks like BLAH."). So that you need

to look at the shapes of the plots, the scale on the y-axis of the plots in the right column have been removed.

**Deliverables:** For each of the 7 plots on the left: an index of the corresponding plot on the right, and one sentence explanation.



3. **Convolution.**

A concept that is closely related to the discrete Fourier transform 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]

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

(a) (4 points) Given a six-sided die, let $p$ be a 6-tuple where $p[i]$ denotes the probability that $i+1$ appears. Let $q = \underbrace{p * p * \cdots * p}_{100 \text{ times}}$. What event occurs with probability $q[150]$? [Assume zero-index vectors.]

(b) (4 points) Given an $N$-tuple $\mathbf{f}$, define $\mathbf{f}^+$ to be the $2N$-tuple obtained by padding $\mathbf{f}$ with zeros:

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

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. [You should not just quote the lecture notes—you must write out the calculation explicitly using the definition of the Fourier transform.] 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) (5 points) 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 $3,210 \times 654 = 2,099,340$. Using your code, what is the product of the following two numbers:

$$x = 12345678901234567890, \text{ and } y = 98765432109876543210987654321 0?$$

[For this part, please embed your *actual* code in the solution, instead of including it in the appendix. Do not directly use convolution functions in your code.]

(d) (4 points) Compare the asymptotic run-time of this multiplication algorithm 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.

(e) (Bonus: 2 points) 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 answers for (c); discussion for (d) and (e).

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

4. **The Sound of Fourier Transform.** In this part, we explore how Fourier transform can help us understand human sounds. Three years ago, the Laurel/Yanny audio clip ignited the internet. Some people hear the clip play as "Laurel" and some hear it as "Yanny". Several news outlets posted explanations of the different perceptions, and made tools that let you perceive both interpretations. In this problem, you'll load the Laurel/Yanny audio clip, and process it in several ways that accentuate the "Laurel" or "Yanny" perceptions.

   It is easy to load an audio file, such as the LaurelYanny.wav file we provide, into a Matab or Python numpy array. In both cases, you also need to keep track of the "sample rate" of the file, which corresponds to the number of datapoints per second. In Matlab, you can do this by:

```
[data, sampleRate] = audioread('laurel_yanny.wav')
```

   If you're working with Python, you can load a `.wav` file as a numpy array in the following way:

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

with open("laurel_yanny.wav", "rb") as f:
  sampleRate, data = wavfile.read(f)
```

   You can also listen to the sound represented by an array. The Matlab command `soundsc(data, sampleRate)` will play the waveform, sampled at a rate of *sampleRate* over your computer speakers. 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))*32767).astype(np.int16)
with open("output.wav", "wb") as f:
  wavfile.write(f, sampleRate, data)
```

   **Exercises:**

   (a) (0 points) Download and play the laurel/yanny recording. What do you hear? Most people hear "Laurel" or "Yanny"/"Yarry" or both.

   (b) (1 point) Load the signal into an array using the instructions above and plot the waveform/signal that this 43,008 point array represents. (You will have a plot where the $x$-axis of the plot is time, and the $y$-axis is the physical position/displacement of the speaker cone.)

   (c) (3 point) Take the discrete Fourier transform of the signal, and plot the real number corresponding to the magnitude/amplitude of each coefficient of the Fourier transform. Describe what you see in words. Why should you expect this?

   (d) (5 points) Plot the "spectrogram" of the signal, which illustrates how the amount of high/low frequencies change over the course of the audio clip. Specifically, chop the signal up into 500-sample long blocks, and compute the Fourier transform for each chunk. Plot a heatmap (in Python, plt.imshow() with cmap='hot'; in Matlab, imagesc()) where the x axis corresponds to the index of the chunk, and the y values corresponds to frequencies, and the value at location $x, y$ corresponds to the magnitude of the $y$th Fourier coefficient for the $x$th chunk. Since frequencies beyond the 80th Fourier coefficient are too high for humans to hear, the $y$-axis should just go from 1 to 80. Turn in this spectogram—you should see some curious structure. (If the contrast is too low to see clearly, take the logarithm of each entry before plotting it....)

   (e) (4 points) You will now change the signal to sound more like "Laurel" or, more like "Yanny". To start, take the original audio, and try 1) zeroing out all "high" frequencies—frequencies above a certain threshold $t$, and 2) zeroing out all frequencies below $t$. [This can be accomplished by multiplying the Fourier transform of the signal by the thresholding function, then taking the inverse Fourier transform.] Is there a threshold $t$ such that the "low" frequency version sounds like "Laurel" and the "high" frequency version sounds like "Yanny/Yarry" to you? If so, state what this frequency is.

(f) (4 points) Try changing the playback speed (by multiplying the "sampleRate" by values in the range 0.25 to 2. What effects do you observe?

(g) (5 point bonus) Can you find other ways of bringing out the Laurel or Yanny in the original audio? You can experiment with modifying the Fourier transform in more complicated ways, including fancier scalings, as well as things like translating the Fourier transform (i.e. appending zeros to the beginning, or removing the first set of values), etc. Discuss any especially effective approaches that you discovered.

(h) (**10 point bonus**) Can you create a new, distinct Laurel/Yanny style illusion? One natural research question is the extent to which this Laurel/Yanny phenomenon is an isolated instance. Can you create a different audio signal where one portion of the population perceives one word, and another portion of the population perceives a different word? What fraction of words can be turned into this sort of phenomenon? To show that this question is not impossible, in "WorldsYikes.zip", we have several audio signals that we constructed which can be perceived as either the word *worlds* or the word *yikes* depending on the playback speed and/or damping of the high frequencies (try playing "worldsYikes1.wav", "worldsYikes2.wav",...,"worldsYikes10.wav" until you perceive both "worlds" and "yikes"). For any example you find, include at least 3 or 4 versions of the audio clip so that we can hear both/all perceptions. Upload these files to YouTube and include the links in your solution.

Grading note: This bonus part will be graded in an all-or-nothing manner with no partial credit. If you end up constructing at least one good example (of comparable quality to the laurel/yanny or worlds/yikes examples) and describe your approach, you will get full credit and your example(s) will be presented to the class.

**Deliverables:** Plots for (b) and (c); description of plot and explanation for (c); image of spectrogram for (d); discussion for (e) and (f) , discussion, possibly including plots for (g); code, description of how to generate the illusions, and Youtube link for (h).