ELEC4400: Homemade Discrete Fourier Transform

Thomas Fuller

July 11, 2018

Introduction

Your own discrete Fourier Transform is unlikely to impress the masses as much as your own Snapchat filter, but it's probably a much more important thing.

Goal

Our goal today is to implement in Python the Discrete Fourier Transform.

$$\hat{U}_n = \sum_{k=0}^{N-1} u_k e^{-j2\pi kn/N}$$

Definitions

$\hat{U}_n$: Big u sub n. Fourier Transform Output. Makes a long sequence $[U_0, U_1, ..., U_{N-1}]$.

u: Little u is the data you want to take the discrete fourier transform of.

u_k: kth element of input array. Another way to phrase it is that it's the k-th element of the thing you're taking the discrete fourier transform of.

N: The number of elements in little u.

n: It's basically the same n as in Fourier Series. The first element of the output sequence $\hat{U}_n$ is

Implementation Notes

1. When you see $\sum_{k=0}^{N-1}$, you just think that you're doing a for loop, and then adding the results of the for loop up every iteration.

2. In the "theoretical world" we have an option of computing an infinite number of "n's". As much as I wish it were true, y'all definitely do not want to spend an infinite amount of time with me though. We're just going to compute N values of U_n and stick them into one array. What this means is that you're going to have a nested for loop structure to your code. One for loop for each output

value. One for loop for each input value. It's gonna be slow. but it's going to be your own homemade creation, it will get you there.

3. In Python, we represent the imaginary component of a complex number by saying, for example "1j". If I have a variable `a`, and I want to make it imaginary, I just do `1j*a`.

Process

1. Open a new script and `import numpy as np`, and `import matplotlib.pyplot as plt`.

2. Create a function called `myOwnDFT`, which accepts an input data `np.array()` called `u`, and returns another `np.array()` called `Un`.

3. Inner for loop: `myOwnDFT` should be a nested for loop structure. The inner for loop runs for k in `np.arange(0,N)`, and performs the summation defined on the top of the lab for a particular value of n.

4. Outer for loop: The outer for loop runs for n in `np.arange(0,N)` and uses `np.append()` to append the result of the inner for loop to Un.

Testing/Questions

1. I want you to show that the DFT works by creating a sine wave $\sin(2\pi f t)$, which has a duration of 1s, and a frequency of 10Hz. Repeat this for 20 Hz, and also 30Hz. Show that the "peaks" of the FFT, have the highest amplitude at the input frequency of the given sine wave. **When you PLOT the fft OUTPUT against FREQUENCY, you need to generate a list of the frequencies, rather than a list of the fft output INDICES.**, so try and think about how you did that in last lab, and apply it here. Plot these three FFT outputs on one figure, with 3 subplots.

2. Import the python "time" module `import time`, and then use `start = time.time()`, and `end = time.time()` to determine how long it takes to run the home made DFT. Do this for an input sinusoid of 200,400,600,800 and 1000 elements in length. Plot the number of elements in the input vector on the horizontal axis, and the number of seconds it took to run the DFT on the vertical axis. (Weird I know), but do this to convince yourself that the algorithm is $O(n^2)$.

3. Use the code from Lab 8 to determine the length of the "kpt.wav" file, and use reasoning from number to try and estimate how long it would take to run our DFT on that data.

4. I've uploaded a portion of the KPT.wav file from last lab for you to use your DFT function on. Please tell me what note was being played, using the

resources from last lab. (Whichever Note it's closest to is the actual note).

What you turn in

1. Submit code, plots, and answers to the questions.