Lab 5

Learning Objectives:

I. **What characters do I type next?** *How do I interact with data in Python and then display this data?* This lab will focus on using NumPy to be able to mathematically manipulate data.

II. **Let me Google that for you.** *How do I use function 'X'?* This lab will require use of many different functions from libraries. One of the most useful skills a programmer can have is the ability to read documentation and understand how to use certain codes. Thus for this lab we want you to look up documentation on functions and figure out how to use them to help you get the results you want.

While you work is about learning to read documentation and apply it to your code.
Part 1 is a warmup for you to play around with some basic but extremely widely used NumPy functions.
Parts 2 and 3 help you practice with more complex functionality in NumPy, especially as used for data analysis.

While You Work: Google Documentation

It's hard to remember every little detail of a function, so it's useful to be able to look them up. When you need to learn or relearn how to use it up, try Google first. If you figure out that certain search terms work better than others, take note of that for future reference.

Part 0: Get the Lab from Github

Today's lab has starter code. Click on the link https://classroom.github.com/a/1I09lQG8; the lab is then located at https://github.com/physics91si/spring19-lab05-*username*.

Part 1: Playing with simple arrays

This part is for you to start playing with NumPy and matplotlib. Open file lab05.ipynb and look at Part 1. Notice that we already included the relevant import statements so that you can immediately begin to use functions from the NumPy library. (Note that we imported it as np so you can access its functions by typing np.*function()* rather than the full library name.)

What's the name of the NumPy function for $sinx$? Make an array of x-values from 0 to 2π with a spacing of 0.01 using the np.arange function. Use it and the already implemented plotting function plot_fn() to plot $sinx$ on the interval $[0, 2\pi]$. Does it look like what you expect?

Integrate your sin(x) function over the same range using integrate(y, dx) and compare to analytically known results (the integral of $sinx$ from 0 to 2π should give 0). How close can you get? How does it depend on the spacing dx?

Instead of using your integrate(y, dx) function, find and use a NumPy function that also numerically integrates your function. Which function is it and how do the results compare?

Now make an array of $sinx$ generated from x-values from 0 to 6π with a spacing of 0.01. Can you find all the local maxima and minima of the $sinx$ array? Where are they and how do the values compare to what you expect? *Hint: the functions np.diff and np.sign may be helpful.*

Part 2: Data Analysis

Look at part 2 of lab05.ipynb. You'll notice that it contains a function noisy_packet(). This function creates a Gaussian wave packet with some simulated experimental error. This function accepts four arguments: an array of x-values, a wavenumber k (for the sinusoid $sin(kx)$), a standard deviation σ (sigma, for the Gaussian), and a parameter noise_amplitude (how much simulated error you want). It produces an array of y-values describing a Gaussian wave packet with those parameters. **Your task is to clean a lot of the noise out of this function using a Fourier transform.**

- **Wave packet:** a wave packet is a function that is sinusoidal within a small range and pretty much zero everywhere else. It is created by multiplying a sinusoid by an "envelope": some decaying function—in this case a Gaussian, $e^{-x^2/2\sigma^2}$, where σ is the standard deviation—which selects the portion of the sinusoid to keep.
- **Gaussian noise:** To simulate experimental error, this returned noisy packet has Gaussian noise. This means each "measured" y-value deviates from the true (wave packet) y-value by some amount, and that the deviations are Gaussian distributed: it is very likely that most deviations will be close to 0, but there's a small probability that any particular deviation could be very large (positive or negative). The Gaussian noise from this function has standard deviation noise_amplitude.
- **Fourier transform:** Conceptually, the Fourier transform accepts a function of x (e.g. a wave packet) and returns a function of k, which tells you how strong the mode with wavenumber k is in the provided function. Its inverse takes the function of k and returns the original function.

Now write code in a cell below the defined functions. Call the noisy_packet function with different parameters and plot the resulting noisy packets. Produce a noisy wave packet with a noise_amplitude of 0.2, a wavenumber of 5 and a standard deviation of 1. What do you expect

it to look like? Plot it; does it look like what you expect?

We'll now do some simple data cleansing to get rid of the noise. Fill in the function clean_data with the following steps.

- First take the Fourier transform of your noisy data to get the transformed data and the frequencies (*Hint: look at* numpy.fft.rfft *and/or* numpy.fft.rfftfreq).
- Implemented for you is a way to zero out the transformed data corresponding to noisy high frequency components (a "low-pass filter" - when applied, only low frequencies survive). Use the low-pass filter, Inverse Fourier transform the result using numpy.fft.irfft, then generate your "clean" data.
    - *How does the low-pass filter work? Essentially, the output of the fourier transform np.fft.rfft is represented by an array where each index corresponds to a fourier component, the first being the lowest frequency, and the last being the highest frequency. Thus the filtering can be applied based on indices. If you want to see the frequencies for yourself, poke around with np.fft.rfftfreq.*
- Plot the cleaned data along with the noisy data using plot_fn(). Have you successfully cleaned up the wavepacket?

Part 3: If You Have Time

Generate a clean wavepacket using the function from Part 2, but this time use the np.ma module to mask everywhere where the absolute value of the wavepacket exceeds 0.5. *(Hint: np.where is very useful for generating the mask of a np.ma.masked_array)* Plot this masked wavepacket and then see what you get. Now repeat the steps in Part 2 with the masked data. Have you successfully cleaned up the wavepacket? Why or why not?

Part 4: Submit Your Lab

In case you forgot -- save your work in the Jupyter notebook, then shut it down and enter the commands (from Terminal/Anaconda Prompt/whichever command line you're using):

1. git commit lab05.ipynb -a -m "<insert commit message here>"
2. git push