Module 11 – numpy and matplotlib

Instructions

1) If not already open, from the Anaconda Navigator Home screen, launch Sypder

2) In Spyder, go to “File”, then “Open”, then navigate to the location where you saved: “11_Numpy_Matplotlib.py”, and select it to open the script file.

3) Introduction

For many of you, these two modules will be essential to your research workflows. This will give you an introduction of how to load these and start working with them in your code. Both do come along with the Anaconda Distribution, so you shouldn’t need to import them.

Frequently, you’ll see examples just like Line 10, where **numpy** is imported as **np**, which is then used to reference all of the allowable methods/operations and properties associated with numpy. This is a helpful package for computational methods and techniques, and is part of the larger scipy package (which of course also comes with the Anaconda Distribution). More can be found here: https://numpy.org/.

Matplotlib is something you will largely use to produce graphs, charts, plots, etc., and offers a good deal of flexibility in how you can do so. It also offers opportunities to produce animated or interactive visualizations. You can find out more here: https://matplotlib.org/ .

Given what they each do, it is not surprising to find that they often are used together.

4) Working with Arrays in numpy

Arrays are the primary object that numpy uses. **Run Line 10 first**. You can initiate arrays as shown on Lines 13 and 15 in the script file. The outer brackets define the full extent of the matrix, while the inner brackets define the n number of rows that your matrix has. The length of the rows represents the number of columns.

Once these two 2x2 matrices are initiated when you **run Lines 13-17 together**, the basic matrix algebra operations can now be summoned without needing to type **‘np’** in the code to do it. **Run lines 19 and 20**, which show the output of matrix addition and subtraction, respectively, and the results (should) check out. One thing to watch for, though: careful using the ‘*’ operator for matrix multiplication. **Run Line 21 and Line 23**, which produce very different results. The former simply multiplies each element in A by the element in the corresponding position in B. For matrix multiplication, call the **np.dot()** method, as on Line 23. If you need a refresher on matrix multiplication, the comments on Line 24 through 26 walk through how those numbers came about.

That’s not to say that numpy won’t help catch other issues. **Running Lines 28 and 29** will show you that trying to add these two matrices with different dimensions creates some issues. D and E each have at least one similar dimension to A, so adding either of them to A in the console will provide a valid input, once you **run Lines 32 and 33**

When working with your own datasets, it may be helpful to translate data read from an input file into an array.

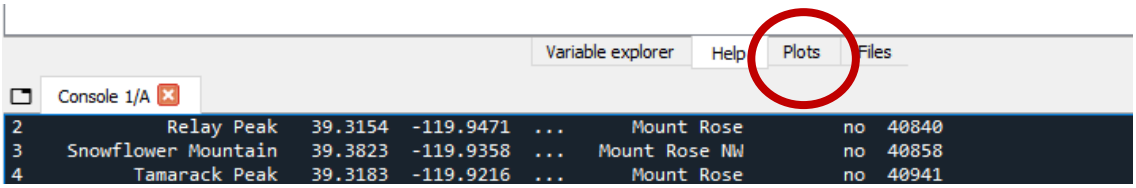
Activity - If you have a dataset (such as a .csv file) handy, try reading it in using either the basic file reading operations from the previous module, or pandas. Convert some of your data to an array, and try out a few basic operations.

5) Working with matplotlib to make static plots

The examples that follow introduce you to pyplot, but feel free to explore around matplotlib for other kind of visualizations you can apply to your data and workflows. Lines 39 and 40 point you to the source of the examples that follow, too.

My code comments describe what some of the characters mean in some of the lines of code (such as Line 50, Line 59, and Line 64). You saw variations of some of these lines of code in the previous module. Make sure you **run Line 41 first** to import the right module.

Then, 1) **run Lines 43-46 together**, 2) **Lines 50-54 together**, 3) **Lines 57-60 together**, and then 4) **Lines 64-66 together**. As in the prior module, to find the output of the plots, make sure you click on the "Plots" tab above the console (as below).



Two minor things to point out. Line 51 shows you how to tailor/modify the axis values to your liking, and the values appear in the list in this order (x-min, x-max, y-min, y-max).

Line 57 uses the numpy method **arange,** which sets min, max, and interval values (respectively).

Experiment, modify, try different visualizations, and I strongly encourage you to enter your own data for plots as you play around with these examples after running through them the first time.

This completes the structured part of Module 11. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.