Skill-04-ReadWriteFiles-Student

September 12, 2024

1 Skill Homework #4

1.1 # Reading & Writing Data Files

Run the cell below to read in the `numpy` module and recreate our arrays from the last sesssion to get started.

[1]:
```
import numpy as np
time = np.linspace(0,5,6)
xPos1 = 1 + 2 * time
xPos2 = 2 * time**2 # remember ** is raise to a power.
```

---

2 Writing Data to Files

The `numpy` module includes functions to help reading and writing data to files. Sometimes they are useful, but sometimes it is easier to code the file input and output manually. You will see either way has details that you have to close attention to.

A file format I strongly recommend is a Comma Separate Value (CSV) file with comment lines begining with a pound sign, `#`. This file format is common and it can be opened and read by Microsoft Excel and LibreOffice Calc programs. Many other program produce and accept this format. Here is what it might look like for our little time and position data set.

```
# Data from run 1
# 2015-01-06 20:53
# This is a final comment
# Time, x1(m), x2(m)
0.000,1.000,0.000
1.000,3.000,2.000
2.000,5.000,8.000
3.000,7.000,18.000
4.000,9.000,32.000
5.000,11.000,50.000
```

Note that you should include meaningful comments about the data **in the data file itself**! Here are ProfHuster's words of wisdom (actually requirements) for data file comments. 1. You should alway include the date and time the data was created in a comment. 2. You should have a comment

right before the first data point with a brief column label including units. 3. You should have at least one comment describing the data.

I also strongly recommend that the *name* of the data file contain at least the data and a sequence number. For example I am going to create a data file named `2017-01-07-Run1.csv`. If you write start the file name with the date in this format `YYYY-MM-DD`, then when the files are in alphabetical order, the are also in chronological order. When you are working on an experiment over several weeks this is a huge help!

2.1 Using `np.savetxt` to Write Data Files

The basic `savetxt` function has the usage `np.savetxt(fileName, myArray)`. Your filename must be a string. Use `.csv` as the extension or last four characters of your file name. The second argument is an array to save.

Here is the first crisis. One and only one array can be saved using this function. However, your data usually has at least two data array to save, like `time` and `position`!

Solution: create a temporary array that glues together your data arrays together.

You can use the basic `np.array` function to do this. Remember that you can make a list where the items of the list are themselves arrays. Sweet. Make this list then call `np.array` to convert it back to an array.

Go ahead and try this using the arrays `time`, `xPos1`, `xPos2`, and print the results out.

[ ]:

OK. You should have gotten output like this:

```
[[  0.   1.   2.   3.   4.   5.]
 [  1.   3.   5.   7.   9.  11.]
 [  0.   2.   8.  18.  32.  50.]]
```

If you did not, then try the command `print(np.array([time, xPos1, xPos2]))`

Problem: this array is "sideways" or transposed. If you wrote this new array out, all of the times would be in the first line of the file, then all of `xPos`, then all of `xPos2`. The file would only have three lines of data in it. You want it the other way with the first line having a `time`, a `xPos1`, and a `xPos2`.

Easy. Every `array` has a built in transpose operation you can use just by adding a `.T` at the end of the array. In the cell below print out the temporary data array in the correct orientation. It should start like

```
[[  0.   1.   0.]
 [  1.   3.   2.]
```

[ ]:

You are almost there. Next you need to know how to put commas as the delimiter (the character that separates the number in a line) in the file.

Python has what are known as `keyword arguments` in functions. These have a default value, so if you are ok with the defulat value, you can leave the argument out. The function `savetxt` has a default delimiter of a space, " ". Since you want a comma, after the filename and the array, include `delimiter=','`.

When `savetxt` writes out arrays of floats, it write 18 significant figures for each number. This makes the file hard to read. If you know about how many significant figures your data has, you can supply a *format* for your data. For this example, two decimal places are fine. The format for this in python is `%.2f`. The `%` starts the format statement, the `f` makes it a floating point formatted output, and `.2` says format the data with two decimal places. You need another keyword argument to the function `savetxt` that looks like this: `fmt='%.3f'`.

Finally, you have to give lines of comments at the beginning of you file. The function `savetxt` conveniently has this feature with the keyword argument `header=`. You do not have to start the header lines with the comment character `#` because that is the default character `savetxt` adds to the lines in the header. Since you want more than one comment line in your header, the easy way is to define a string variable with several lines of comments in it. This is easy to do with the triple quote. Here is what I used to define the variable `headers`. Note the backslash continues the line of code:

```
headers = \
"""Data from run 1
2015-01-06 20:53
This is a final comment
Time, x1(m), x2(m)"""
```

BTW, I learned *all* of this about the function `savetext` by typing `help(savetxt)`

I am going to give you the command to write your data file. Copy, paste, and run the following in the cell below and run it.

```
headers = \
"""Data from run 1
2015-01-06 20:53
This is a final comment
Time, x1(m), x2(m)"""

np.savetxt("positionData.csv", np.array([time, xPos1, xPos2]).T,\
  fmt='%.3f', delimiter=',', header=headers)
```

[ ]:

Now you should have your data file in the folder where jupyter is running. Check it. It should have your comments, then the data in three columns separated by commas.

You can open files in jupyter by click on the tab `Home`, then clicking on the file name. Open `positionData.csv` from jupyter and check that its contents looks like the file at the top of this notebook.

Whew! Next is reading a data file.

---

3 Reading a Data File

Creating a data file is useless if you don't know who to read it. Fortunately `numpy` has a function for reading data files named `genfromtxt`. The basic usage is `myArray = np.genfromtxt(fileName)`. Note that I like to first make a variable with the file's name, then use the variable in `genfromtxt`. One reason I do this is I can use that variable in the title if I make a plot. Then I know where the data came from that made the plot.

Another cool jupyter hint: When making the variable `fileName`, after you type `"`, press `<tab>` and jupyter will tab complete to help you find a file in your working folder.

Run this code in the cell below, but put your file name in the variable string.

```
fileName = "your file name"
myArray = np.genfromtxt(fileName, delimiter=',')
print(myArray)
```

Do this. You should get your data in three columns.

[ ]:

You are almost there for what you need. You actually want to read the data into named arrays. The following sample gives you your time array:

```
t1 = myArray[:,0]
```

Reminder, in the index brackets, the `:` means *all elements* and `0` means *of column 0*, the first column.

In the cell below, create the new variables `t1`, `yPos1`, and `yPos2` from `myArray`. (If you mess up, you may have to re-execute the previous cell that calls `genfromtxt`.) Print the variables out to verify you have grabbed them correctly.

[ ]:

3.1 Printing the Comments

A lot of times I would like to see the comments at the top of the file. You can just open a CSV file with any text editor, but you have to be careful because most operating systems want to open CSV files with Excel of another spreadsheet program. In both Windows and Mac OS X, if you navigate to the file and right-click, you can choose `Notepad` or `TextEdit` to open the file. Then you can read the first few lines.

The code below has a programming approach. The first line is a `for` loop over all of the lines in the file, the second line picks the lines starting with the comment character.

```
for line in open("2015-01-06-Run1.csv", 'r').readlines():
    if line[0] == '#':
        print(line, end='')
```

(The argument `end=''` keeps the print function from adding an extra line to the output.)

Run the code in the cell below.

[ ]:

4 Conclusion

There are many more ways to read and write data files from python, but these methods will go a long way for you.