MATLAB Assignment 5

Spring 2019, Section B

In this homework, we will go through a few advanced data structures in MATLAB, such as cell arrays, classes, chars, as well as dataset-importing. You will be importing in the fisheriris dataset and encapsulating it in classes. After obtaining all the data, you will be required to write some methods for the class you just created. Also note that some operations will require you to do some research, but not too much!

Please submit this homework as *.m* files, with suppressed output. Remember that all lectures and homeworks may be found at *github.com/guybaryosef/ECE210-materials*. This homework is due at 11:59PM on March 20th to guybymatlab@gmail.com.

1. OOPs I did it again

a. Load in the fisheriris dataset with the command ***load fisheriris***. You should obtain a 150×4 matrix called *meas*, as well as a 150×1 cell array called *species* in your workspace. This is a very popular dataset, and you can quickly google ***fisherisis*** to see what the columns of meas represent.

b. Create a class called *Flower* in its own *.m* file. In your *Flower* class, you should have the following properties:

- petalWidth (double)
- petalLength(double)
- sepalWidth(double)
- sepalLength(double)
- species(char).

The ith species corresponds to the ith row of information in *meas*. Note that you do not need to specify the data type of the properties when you are declaring a class - the following properties are just here to impress on you what type of properties you would be expecting.

c. Create a constructor for your class. It should take in 4 doubles and a char array corresponding, in order, to the five properties of your class.

d. Now, import the entries from *meas* and *species* into MATLAB and store them in a 150×1 cell array of *Flower* instances. You can either use a for loop to import the entries, or use a more elegant way to make all entries in one line (this would require some research on object formation in MATLAB). Either way is OK. However, note that the names of the species are stored as cell arrays; make sure to extract them from the cell as a char and remove trailing white spaces

before storing them in the *Flower* instances. There is a single function that will do this, which you should be able to find with a quick google search.

e. Create a method called *getSLength* for the *Flower* object, which will return the sepal length of the object. Use this on the 25th Flower in your cell array, and check (using ==) that this is the correct value by comparing it to the corresponding entry in *meas*.

f. Create another method called *report* for the *Flower* object, which will print out a statement on the command window and report the details about the *Flower* object. This function does not need to return anything. It should print out a statement of the following form if the flower is 5.1 cm in sepal length, 3.5 cm in sepal width, 1.4 cm in petal length, 0.2 cm in petalWidth, and the species is setosa, for example:

> "The length and width of its sepal are 5.1 cm and 3. 5cm respectively, while the length and width of its petal are 1.4cm and 0.2cm respectively. It belongs to the setosa species."

Bonus (These two questions below are worth 1 extra point):

2. As computer programmers, when we see a number such as 1.9999999 as the output of some function, we are quick to assume that the output is really 2 and that this is simply the effects of quantization. However this is not always the case- there are some numbers, sometimes coined as *near integers*, that closely, but not exactly, resemble integers.

Although this could often lead to an obscure and difficult distinction in computer programming, in MATLAB we can use the symbolic toolbox to find the exact values of such numbers. Actually, if you want to be technical about it, you will never find the exact value of these numbers, but rather will get as close to the exact value as you wish.

Take as an example Ramanujan's Constant, $e^{(\pi\sqrt{163})}$.

a First try to calculate its value using regular variables (i.e. a *double*).

b Next use the symbolic toolbox's ***sym*** function to get a symbolic representation of this number. Then use ***vpa*** to get a 30 and 100 digit representation of this number.

c Create a table comparing these 3 values.

3. Lets try a different, but equally interesting, experiment. Although it is believed that as the decimal digits of π go into infinity each digit appears equally often, no such proof exists. Let us see if we can garner some intuition on this:

Use ***vpa*** to get the first 10001 digits of π, convert the resulting number into a *char*, and then iterate through it, keeping tally of the amount of times each digit appears. Finally, create a histogram of the frequency of each digit.