PROGRAMMING ASSIGNMENT 1

Due: Thursday, September 3 by 11am. Upload *one* Jupyter notebook to Gradescope.

1. PROJECTILE MOTION

Write a program that plots the trajectory of a projectile launched (without air resistance) at ground level with an initial speed v_0 and a launch angle θ_0 (measured from the ground). The program must have the following features:

- v_0 and θ_0 are clearly defined in the program.
- the time t and the coordinates $x(t)$ and $y(t)$ are stored in arrays.
- the timestep Δt is defined early in the program.
- *either* a `for` or a `while` loop is used to compute the elements of the arrays.

Your program must output graphs for four different input values of v_0 and θ_0 (that range from 10° to 80°), as well as a cell where you describe in a few (complete, grammatically correct) sentences the trends that you observe when varying v_0 and θ_0. Note that you are plotting the analytic solution here – no numerical technique needs to be used to solve this problem, but you should still use the basic code structure outlined above.

Grading Rubric: Program has required features described in prompt (25%), graphs show correct behavior and are easy to interpret (25%), written response is factually correct (25%) and uses proper grammar and spelling (15%), code compiles without errors and reproduces all output (10%)

2. PRACTICE WITH LOOPS

Write a program that does the following:

- Creates an array of length N, where N is a user-defined input value ($N \approx 100$ is fine).
- Initialize the first element of this array to 0.01.
- Let's assume this array is called `x`. Starting with element 1 and continuing up to element $N-1$, apply the following rule to your array `x`:

```
x[n+1] = μ * x[n] * (1-x[n])
```

  where μ is a user-defined parameter that is defined early in the program.

Your program must output several graphs that can be used to answer the following questions (you must also include a cell or cells answering these questions in a few sentences):

- Set $\mu = 0.5, 1, 2, 2.5, 3$. What happens to the graph of x_n vs. n?
- What happens when $\mu = 3.2$?
- What happens when $\mu = 3.5, 3.6, 3.7, 3.8$?
- Set $\mu = 2$. Does the behavior of the plot change if you initialize the first element of your array to a different value (between 0 and 1)?

Grading Rubric: Program has required features described in prompt (25%), graphs show correct behavior and are easy to interpret (25%), written response is factually correct (25%) and uses proper grammar and spelling (15%), code compiles without errors and reproduces all output (10%)

3. Code Efficiency

Suppose we have a polynomial of the form

$$p(x) = \sum_{j=0}^{n} c_j x^j$$

where the term c_j represents the polynomial coefficients. There are two ways that we could evaluate and sum the n terms in this series:

- Directly: starting with the zero term and working upward to term n. In code, this would look something like

```
p = 0
for j in range(n+1):
    p = p + c[j] * x**j
```

  where `c[j]` is an element of an array containing the polynomial coefficients.

- Backwards: starting with term n and working *down* to term zero. In code, this second approach looks like

```
p = 0
for i in range(n+1):
    p = p*x + c[n-i]
```

Your task is to evaluate which approach is faster and *why* it is faster. Use the polynomial

$$p(x) = \sum_{j=1}^{n} \frac{1}{j} x^j$$

and compute the sum of this expression for 10^4 evenly-spaced values from $x = 5 \times 10^{-5}$ up to $x = 0.5$ and store the values in an array. How long does each method above take to run when $n = 1, 2, 3, \dots, 8$. *Hint: look up the Python module "time".*

Your code must print out a table of the execution time of each method for each value of n, as well as a cell where you describe in a few (complete, grammatically correct) sentences *which* method is faster and *why*. **Important note!** Saying "method x is faster because it took less time to execute" **is not an answer for the *why* component of this question** (it is just restating the *which* component of the question). *What is the computational difference* between the two procedures that produces the difference in execution time?

Grading Rubric: Program has required features described in prompt (25%), tabular output shows correct behavior and is easy to interpret (25%), written response is factually correct (25%) and uses proper grammar and spelling (15%), code compiles without errors and reproduces all output (10%)

4. Radioactive Decay

Retrieve the file `counts.dat` from Blackboard and make sure it is saved in the same directory where you are executing your Jupyter notebook (or else keep track of your directory structure, so you can tell your code where to look for the file). This file contains the count rates, as a function of time, for a hypothetical sample of ^{99m}Tc, an excited state of a radioactive isotope that is commonly used in certain medical tests. The first column in this file is time (in seconds), and the second column is the gamma ray count rate (number of counts per second).

The count rate of detected gamma rays per second days with time, following the functional form

$$C(t) = C_0 e^{-t/\tau}$$

where C_0 is the initial count rate and τ is a time constant that you must measure by fitting a function to the data. Note that there are two ways you could fit a function here: by directly fitting an exponentially decaying function to the provided arrays, or by fitting a *line* to the data on a log-log plot (exponential functions in linear space look like straight lines in log space). Which approach you use is up to you: fitting an exponential function is more direct but will require more care in dealing with your fit parameters (C_0 and τ), while fitting a line requires some manipulation of the data up-front but the resulting slope of that line is related to τ.

You may find the SciPy module `curve_fit` helpful for this problem.

Your program must read the columns from `counts.dat` into separate arrays and output a plot of count rate vs. time. Your fit to the data should be superimposed on the data (in a different color, plotting symbol, line style – it is up to you, but the observations and model fit should be *clearly* distinguished from one another). Your plot must additionally include a legend for easy identification of the observational data and model fit and display your result for τ. *Important hint*: `log` and `log10` are not the same thing! One is the *natural* logarithm and the other is the base-10 logarithm – make sure you are using the correct one!

Grading Rubric: Program has required features described in prompt (25%), graphs show correct behavior and are easy to interpret (25%), written response is factually correct (25%) and uses proper grammar and spelling (15%), code compiles without errors and reproduces all output (10%)

5. Using Subplots

It is often very useful to show multiple plots side-by-side or with shared axes, instead of generating multiple plots or cramming multiple lines or data sets into a single plot. The Matplotlib gallery has a subplot demo (https://matplotlib.org/examples/pylab_examples/subplots_demo.html) that illustrates how to do exactly this. Using this demo as a guide, create your own graphic that depicts the function $f(x) = x \sin\left(\frac{1}{2}x\right)$, as well as the first and second derivatives ($f'(x)$ and $f''(x)$, respectively). You must show your work in deriving the first and second derivative. You must create at least two different versions of the same figure, using two different applications of the subplots routine, and describe why you chose to format your plots the way that you did. Note that, in addition to getting your functions correct, the most of points for this problem will be awarded based on the clarity of the plots you generate.

Grading Rubric: Program has required features described in prompt (20%), work is shown (and is correct) in deriving derivatives (20%), graphs show correct behavior and are easy to interpret (20%), written response is well justified (20%) and uses proper grammar and spelling (10%), code compiles without errors and reproduces all output (10%)