ET 204 Final Project

Spring 2020

Background

The logistic map is a deceptively simple difference equation (basically a discrete differential equation) that is famous for showing unbelievably complex behavior. The equation takes a current value of a variable, x_n (the *nth* value), and uses it to calculate the next value, x_{n+1} (the *(n+1)th* value). Here is the relationship:

$$x_{n+1} = x_n * r * (1 - x_n) \quad (1)$$

The equation originally was intended to predict wolf population fluctuations from year to year and introduced by a biologist, Robert May in 1976. See here for tons of interesting info:

https://en.wikipedia.org/wiki/Logistic_map

The ***x***'s are bound between 0 and 1 and represent the population (1 = the absolute maximum number of wolves a particular environment can physically support and 0 = total population collapse) and the parameter ***r*** loosely represents how much food supply the wolves have. As you will see, ***r*** really drives the behavior of the equation.

It works like this. You feed the equation a seed value by setting x_n = any number between 0 and 1. Plug that into the right side of eq (1) to get x_{n+1}. Once you have x_{n+1} you can plug it into the right side of eq (1) in the x_n position to get another x_{n+1} which is in reality x_{n+2}, and so on. Make sense?

For ***r*** < 3, the system will converge to a fixed value (also known as a *fixed point*) after several iterations. For ***r*** > 3, much more interesting things happen. (see the Wikipedia page for details.)

The Assignment

Your assignment is to write a python program which does the following:

- Define a function that will take a current value of ***x*** as its input and returns the new value of ***x*** according to eq (1).
- Ask the user for a seed value (between 0 and 1)
- Ask the user for an ***r*** value (between 0 and 4, and make this a *float*, not an *int*)
- Use a while loop (or other means) to check after every iteration of the equation to see if the new value is within 0.0000001 of the old value. If it is this close, we can assume it has settled onto the fixed point and we can stop feeding it new values. If the system fails to settle onto the fixed point after 20,000 iterations we can assume it's doing something else and we can end the program.
- The system should print out the iteration number, ***n*** as well as the current value of ***x*** at each step.

- For ***r*** values less than 3 you'll see that it may settle onto the fixed point fairly quickly. If it does so, print a message like "found the fixed point in", n, "steps!" or something more clever and entertaining. Print a different message if it doesn't settle on the fixed point and goes all the way to 20,000.
- For ***r*** values greater than 3 you will see a variety of things going on. Play around with it and see what it's doing at various values. For example, try 0.50, 3.30, 3.63, 3.84 and 3.98.
- Submit a cover sheet in addition to your python code for me to test. The cover sheet should describe the behavior you see at the ***r*** values you try out. The code itself should not be all that hard to develop – if you find it hard, you're probably not thinking about it right and you should email me (my code is literally less than 30 lines). The thing that's important here is to write code that is useful to you as you are exploring something new. We have this equation we've probably not seen before, and we want to whip together some code and check out what it's doing. This is an important and valuable skill to develop, and the main point of this project.

Extra Credit:

- For a bonus add the following feature. As you can see from playing around with r values, you can find regions among the chaos where there are 'orbits' that repeat every 3 iterations, 5 iterations, 6 iterations, etc. Modify your code to check for a particular orbit length as specified by the user (or hard-coded into your script).

Extra, Extra Credit:

- For you overachievers out there, modify your code to now step through values of ***r***, only stopping when the specified orbit is detected. (I would suggest NOT printing out every value of x in this case as it really slows things down. Maybe just print the current value of r that is being checked.) Use the plots as found on the Wikipedia page to help you figure out a good starting point for ***r*** and how big of steps you should take to make your search successful.