6.S083 Problem set 1: Logistic map

In this problem set, we will study the behaviour of an apparently simple discrete-time dynamics, the **logistic map** given by

$$f_r(x) = r\,x\,(1-x).$$

Here, x is a real number between 0 and 1. There is a single parameter r that ranges between 0 and 4.

This was made famous by a few papers in the 1970s, starting with this one by Robert May in *Nature*. It models, for example, the dynamics of an insect population after n breeding seasons.

The behaviour you will observe has been shown to be **universal**, and actually occurs in a wide range of models of physical, chemical, biological, etc. systems.

Instructions:

Send a `.jl` file or a Jupyter notebook to `sandersd@mit.edu`. Deadline: October 30 at 1pm.

Getting help

Type `?round` in the REPL to get help on the function `round`. In Juno, put the cursor on a word and type `Ctrl-J Ctrl-D` (or `Command-J Command-D` on Mac) to open that function's docs in the documentation pane.

Exercises

Exercise 1: Calculating orbits

1. Write a function `logistic` that takes two parameters, `r` and `x`, and returns $f_r(x)$.
2. Write another method of `logistic` that takes only `r`, and returns an anonymous function mapping x to $f_r(x)$.
3. Check using the `methods` function that the generic function `logistic` indeed has 2 methods.
4. Write a function `orbit` that takes a function `f`, an initial condition `x0`, and a number of iterates, `N`. It should return the **orbit** of x_0, i.e. a `Vector` containing the initial condition and the first `N` iterates x_1, x_2, ..., x_N of the discrete-time dynamical system

$$x_{n+1} = f(x_n)$$

5. Write a method for `orbit` that takes keyword arguments for x_0 and N, so that it is clear which argument is which.
6. Use `orbit` to calculate the orbit of the logistic map with $r = 0.9$ for 100 steps, starting from $x_0 = 0.7$. Examine the data by eye. Is it converging to a fixed point? If so, where is that fixed point? (See lecture 2 slides for the definition of a **fixed point**.)
7. Repeat this for $r = 1.5$.
8. Keep increasing r gradually. For what value of r does this behaviour start to change? What is the new behaviour?

Exercise 2: Plotting orbits

1. Make a function `plot_orbit!` that plots the orbit of a map f with initial condition x_0 and time N, as a function of time n. Draw the orbit using both lines and points. Use `plot!` so that this function adds to a pre-existing figure. You should add an optional argument `label` for a label for the plot.
2. Create an empty figure using `p = plot()`. Use a loop to plot orbits of the logistic map with $x_0 = 0.7$ and 5 values of r between 0.9 and the value you found in exercise 1.7, using the `range` function. Use the `round` function to make the labels nicer (e.g. with 3 digits). Evaluate `p` at the end to display the plot, or use `plot!()`.

   You can change the y-range of your graph using the `ylim=(a, b)` option to `plot` if you need more space on the graph.

   Save the graph as a PDF called `orbits.pdf` using the `savefig` function.

Exercise 3: Visualizing orbits interactively

1. Write a function `visualize_orbit` that takes a function f and makes an interactive visualization of its orbit. Assume that f takes a single parameter p. Allow your visualization to vary x_0, p and the number of iterates N.
2. Use your visualization to fix x_0 and vary r in the logistic map. Search (visually) for values of r where qualitative changes in behaviour occur. Where are they and what happens? What do you see for values of r near to 4?

Exercise 4: Summarizing the behaviour

1. We can summarize the behaviour of the map for all values of r using a *bifurcation diagram*, as follows.

   For 200 values of r between 2 and 4, calculate an orbit of length 500. (You may fix $x_0.)

Plot the final 100 iterates vertically, with r horizontal. To do so, make a vector that stores all of the x_n you have calculated, and another vector that stores the corresponding values of r.

You can select the last 100 elements of a `Vector` as `v[end-100:end]`.

What do you observe? How does this agree with what you found in the previous exercises?

2. Turn this into a function `bifurcation_diagram`, where you can specify the range of r values and how many points to plot at each r.

3. If your computer is fast enough, turn this into an interactive visualization, where you can vary the minimum and maximum values of r that are calculated.

   Is there more interesting behaviour that occurs closer to $r = 4$?

Exercise 5: Benchmarking

1. Write the `orbit` function in Python. Time the code for both Python and Julia to simulate the logistic map for a time 10^6. To make it possibly fairer, make a new version of the function that just returns the final position, and does not save the intermediate positions.

   You can use `%timeit` in IPython / Spyder, and `@btime` from the `BenchmarkTools.jl` package in Julia.

2. How many times faster is Julia than Python for this calculation?

Exercise 6: Understanding the dynamics So far we have observed some qualitative changes in the dynamics. We can understand the change in behaviour visually by drawing the dynamics in a **cobweb plot**, as follows.

1. Make a function `cobweb` that takes `r` and `x0` as parameters. It should make the following figure for the logistic map.

   (a) First, plot the function $f_r(x)$ for x between 0 and 1, and the function $y = x$ using a dashed line (keyword argument `ls=:dash` in `plot`).

   (b) Use `orbit` to calculate iterates starting at `x0` and with the given value of `r`.

   (c) Draw a graphical representation of the trajectory (a "cobweb plot") as follows, as a sequence of horizontal and vertical lines. Start at the point $(x_0, 0)$ and "connect" it to (x_0, x_1) and then to (x_1, x_1). For each $n \geq 0$, connect (x_n, x_n) with (x_n, x_{n+1}), and then this with (x_{n+1}, x_{n+1}).

   To do this, make a single vector of all the x coordinates and another with all the y coordinates. Recall that the function `plot(xs, ys)` *automatically* "connects" the data that it is passed in `xs` and `ys`, so you just need to put the points in the correct order.

Use `aspect_ratio=1` to get the plot to look correct.

2. Make an interactive visualization with a fixed x_0 and varying r starting at 0.9. Describe what happens as you gradually increase r to the value that you found at the end of exercise 1.