Assignment 4 Python - Predator-prey

Due Feb 11, 2016 by 12pm **Points** 68 **Submitting** an external tool
Available until Feb 13, 2016 at 12pm

This assignment was locked Feb 13, 2016 at 12pm.

In video and class, you have seen that our current implementation of the Lotka-Volterra equations gives inaccurate results. In this assignment, you will implement a more accurate version, implement a variation on this, and discover properties about the resulting data.

- Start by modifying previous solutions to class exercises on the predator-prey problem. You can use either our solutions or your own. Put all of your code in one Python file, and all of your text answers in one text file, and submit both files.
- **Test** (http://codeskulptor.appspot.com/owltest/?urlTests=comp130.comp130_assignment_predator_prey_tests.py&urlPylintConfig=comp130.pylint_config.py)
- Submit with the button at the bottom of the page.

- **(10 points)** Code style
- **(4 points — not automatically tested or graded)** Documentation strings are precise and accurate descriptions of the functions.

Improving the Model's Accuracy

As outlined in the video about the inaccuracy problem, the simplest solution is to generate population data in smaller time steps. Our previous implementation generated a single pair of population counts for each year. Instead, we can generate data for each month, week, day, or whatever.

(30 points correctness) Implement the function `pred_prey_intervals(birth, predation, growth, death, prey, pred, years, intervals)`. It takes the same four annual rates and two initial population counts as we've seen in previous exercises. `years` is the number of years to run the simulation, while `intervals` is the number of intervals that will subdivide each year.

Previously, our function returned just the population data. Now, we will also return the elapsed time for each data point. In particular, it return the original population counts, plus projected populations for each interval within each year. Thus, the length of the returned list should be 1 + `years` × `intervals`. For example, if `years` and `intervals` are both 2, then the returned list should look like `[(0.0, prey, pred), (0.5, …, …), (1.0, …, …), (1.5, …, …), (2.0, …, …)]`, with the appropriate projected population values.

You will need to make three main changes from the in-class version:

- Modify the loop structure so that it loops the appropriate number of times. This can be done with two nested loops or a single loop.
- Modify how the data is put into the resulting list, so that it also contains the elapsed time values.
- Modify how the rates are used. As mentioned above, the input rates are still annual rates. So, you need to divide the annual rates by the number of intervals in each year.

Plotting the Model's Results

Since this new function returns data in a slightly different form than we've previously used, we need to change our plotting functions, as well.

(8 points — not automatically tested or graded) Implement the function `plot_pred_prey(populations, pred_name, prey_name)`. The only change to the version you did in class is that `populations` is now a list of triples.

(8 points — not automatically tested or graded) Write the function `plot_time_populations(populations, pred_name, prey_name)`. This is very similar to `plot_years_populations()` that you wrote in class. However, the `populations` is now a list of triples including time, so the year is no longer needed as a separate argument.

When testing and plotting, we recommend using `years` ≥ 50 to ensure that you have enough data to discover any patterns and `intervals` ≥ 100 to ensure sufficient accuracy.

In principle, we should now fit our improved implementation's results to our original data. From this we would evaluate how well the computed results match real-world data and also experimentally determine the appropriate rates from this fitting process. However, there are enough problems with the data (e.g., disease and weather) that this would be difficult without more statistical experience. Furthermore, the lynx-and-hare data have the unexpected and much-discussed property that they actually fit the model better with the lynx as the prey and the hare as the predator!

So, in light of these problems, we will move on.

A Variant of the Model

The standard Lotka-Volterra model assumes that food for the prey species is essentially unlimited. However, it is more realistic to assume that the environment has some carrying capacity, i.e., there is a maximum prey population that can be supported. You will implement a variation on our model that adds this notion to our model, and then explore how that changes the population dynamics.

We will model the carrying capacity in a non-standard way that is very simple. When calculating the new prey population for a given interval, we will assume that any prey animals above the carrying capacity die of starvation. I.e., instead of all prey going somewhat hungry, we'll assume that some eat their fill while others starve. Note that the predator species is not affected directly by this carrying capacity. The predators are affected only indirectly, in that their food supply (the prey) are affected.

(8 points) Write the function `pred_prey_carrying(birth, predation, growth, death, carry_cap, prey, pred, years, intervals)`. This should be just like `pred_prey()` above, except with the addition of the carrying capacity `carry_cap`.

Do not change or make a new version of `change_prey()`. Instead, modify the prey population calculation that occurs after the call to `change_prey()`.