Final Project

Introduction to Computing for Biologists

Spring 2020

Due by 5 PM on May 8, 2020

Context

You are a conservation biologist in charge of managing habitats for a protected reserve. Your reserve includes several distinct patches of intact forest that harbor populations of the endangered Warbling Babbler. You are worried about the long-term fate of these populations if the habitats remain disconnected with limited movement of individuals between them (i.e., limited migration between the populations), so you'd like to use funds from your budget to establish habitat corridors that will allow individuals to move between the patches. However, you also know that the populations have some phenotypic differences (like color) and you are wondering how the frequencies of the phenotypes, and the overall population sizes, will change in the populations once they are connected.

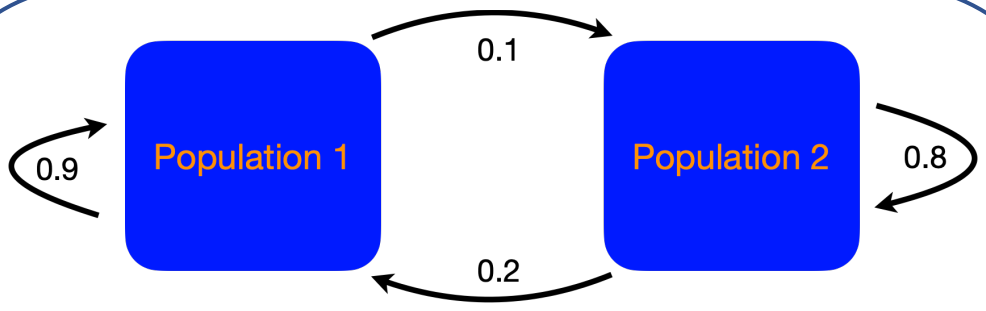
Simulation Model

To explore the consequences of establishing habitat corridors between the patches, you decide to build a Python framework that will allow you to simulate the movement of individuals through time (simulated as discrete steps) and track what happens to the populations. Your framework will include three types of classes: an **Individual** class, a **Population** class, and a **Landscape** class. Individuals exist in populations, and populations exist in a landscape. In each time step of your simulation, individuals can stay in the population where they started, or move to a new population. The probabilities that individuals stay or leave are stored in a table called a dispersal matrix (more detail below).

Individuals need to (at a minimum) have a phenotype attribute, populations need to (at a minimum) have a list of **Individual**s, and **Landscape**s need to (at a minimum) have a list of **Population**s and a dispersal matrix. The dispersal matrix consists of the probabilities that an individual disperses between populations, or stays in place, each time step. For Warbling Babblers, these probabilities correspond to their probability of moving in a week (Fig. 1). If an individual moves from one population to another, you should remove it from the list of individuals in the population where it started, and add it to the list of individuals in the population where it went. There

are multiple ways to use the probabilities in the dispersal matrix to decide if an individual moves in each time step, but one function you may find helpful is *random.choices(…)* from the *random* module.

Population constructors should create a new list for individuals in that population, and **Population**s should have methods to add and remove individuals, as well as to calculate and print the frequency of phenotypes among individuals. **Landscape** constructors should create a new list for the populations in that landscape, and **Landscape**s should have a method that uses the probabilities in the dispersal matrix to determine if an individual moves or stays each time step. Either the **Population** or **Landscape** constructors should set the starting population sizes. The total number of individuals across all populations should stay constant, but individual populations may change size.



From \ To	1	2
1	0.9	0.1
2	0.2	0.8

Fig. 1 – The dispersal matrix (bottom) contains the probabilities that an individual Warbling Babbler in one of the populations moves or stays in that population each week. The probabilities in each row of the dispersal matrix must sum to 1.0. For this example, an individual in population 1 has probability 0.9 of staying in population 1 each week and population 0.1 of dispersing to population 2. If the matrix is symmetric, the probabilities of staying and going are the same, regardless of an individual's population of origin. This example corresponds to the simplest case of two populations, but the same principles apply to three or more populations.

All three classes may contain other attributes or methods as you see fit! Some elements of this framework are similar to recent assignments, so you may find inspiration in the code you wrote earlier. One additional piece of advice is to test each class's constructor and methods as you write them.

When simulations begin, all the individuals in a population should have the same phenotype. As individuals move back and forth between populations through the time steps (weeks), you will want to track the trajectory of phenotype frequencies and population sizes. To do this, you will need to write (at least one) function to create these plots. This function can either be a method of one of your classes, or it can be independent. I recommend overlaying phenotype frequencies or population sizes for different populations on a single plot for easy comparison. For example, you might create one plot for phenotype frequencies that shows three different lines for three different populations, and another, similar plot for population sizes.

Expectations and Guidance

I expect and hope that you will work with others on this project, but given the challenging circumstances of remote classwork you are not required to do so. If you do work with others, *all of the code should still be your own*. As with written assignments for any class, no two people's code should be identical and you should not copy and paste. Plagiarism rules apply.

To demonstrate your progress and ability to use git, I expect to see a regular commit history for your project. *All of the code should not show up suddenly at once in a single commit.* You should begin by writing out what you think the structure of your code should look like using comments and pseudocode. Your first commit should demonstrate this pseudocode structure. You can, of course, modify and change this structure as you work, if needed.

Your code does not necessarily need to be very long. Well-structured code is usually shorter than less structured code.

You can write your code and run your simulations in a single Jupyter notebook, or using some combination of Python and bash scripts. Either way, make sure I can follow the structure of your code, the simulations you ran, and the answers to the questions.

You do not need to submit a pull request to submit your final project. Instead, create a new repository on your own and send me the link to your repository to submit your assignment.

While I intended to have you present on your final projects during our allotted final exam time, I think that will be too challenging right now. Instead, you have all of finals week to work on your final projects. They are **due next Friday, May 8th at 5 PM**. Please email me a link to your repository by then.

Potential Pitfall

Keeping track of individuals that move between populations will require appending and removing individuals from lists. However (**warning!**) you should *not* modify a list that you are looping through inside of the loop itself. Otherwise, you will run into unexpected behavior. Here's an example:

```
test = [1,2,3,4,5,6]
for t in test:
    print(t)
    test.remove(test[0])
```

What happens in this case?

So, if you use for loops to iterate through a list of individuals in a population, do not add or remove individuals from the list until your for loop is complete.

Questions

The purpose of writing this code is to answer questions that you, as a conservation biologist, need answered. Therefore, once your code is working, you should run simulations under different conditions in order to answer the questions below. It is up to you to determine how many and what kinds of simulations you need to run in order to answer these questions.

How do the frequencies of phenotypes change in each population week-by-week as individuals move?

What effect does an overall increase in the rate of movement (migration) have?
What happens to both phenotype frequencies and population sizes when movement probabilities are not symmetric (individuals have a higher probability of moving from population A to population B, than they do of moving from population B to population A)?
What effect does changing the starting population sizes have on the trajectory of phenotype frequencies? In other words, what might happen if you connect big habitat patches versus small habitat patches?

Grading Rubric

40% - Accuracy of Code and Simulation
25% - Commenting and Cleanliness of Code
25% - Use of Code to Answer Questions
10% - Commit History that Shows Project Progress (start with comments/pseudocode)