Investigating Artificial Life in Python

Marius Furter

Submission Deadline: December 10th, 2021

Abstract

In this project you will design, perform, evaluate, and document biological experiments in a virtual setting. Concretely, you will code a class of artificial organisms that live and compete within a virtual environment. In the simplest case, the organisms will simply eat food and replicate, but there are many other features you will be asked to model. You will then study the behavior of these organisms in controlled experiments that you design. The data resulting from these experiments will then serve as a basis for some basic statistical analysis. You will then document your findings in a mini-paper that mirrors a scientific report.

Contents

1 Introduction

1.1 Goal of the Project

The main goal of this project is to become familiar with the natural science research workflow in a controlled setting. This includes how to use python in this context. If you are are studying a natural science, this will benefit you once you are expected to do this in real life. If you are studying mathematics, this will help you gain insight into how scientific research is performed, which may help you if you ever want to develop mathematics that can be applied to this setting.

1.2 What is Expected?

The result of completing this project will be a report whose structure will mirror that of a scientific publication. It will detail what you did, and describe and discuss the interesting results that you discovered. You can find the specifics of how to write the report in Section 7. In particular you are expected to:

- Perform and understand the main tasks outlined in these instructions.
- Come up with your own ideas for extending the basic framework outlined here.
- Implement your ideas in python, learning new concepts when necessary.
- Document your work in a reader-friendly manner.

1.3 Structure of the Project

This project is divided into 4 parts:

1. Experimental Setup: Here you will code the **Organism** and **Environment** classes that will be the objects of study in your virtual experiments.
2. Data Collection: Here you will run the experiments you have designed and export the data.
3. Data Analysis: Here you will find interesting features in the data you have gathered and describe them using statistical techniques.
4. Writing the report: Here you will document the most interesting features you discovered in steps 1.-3.

In practice, you will do steps 1.-3. multiple times, since each analysis of an experiment will lead to further questions you can address in future experiments. You should make sure that during this process you save your work in such a way that makes it easy for you to write the report in the end. You should also adequately document all the code you write during the project.

1.4 Basic vs. Advanced Features

To help you to understand the instructions, some features will be declared *basic*, and others *advanced*. The *basic* features are those that are minimally required for you to follow along. Full code examples for the *basic* feature are available upon request. In contrast, the *advanced* features are not required in order to work through the instructions. They will be described in less detail and require more thought on your part. You can also come up with your own *advanced* features. A complete project will need to implement all the *basic* features, and a selection of the *advanced* ones.

2 Experimental Setup

In this section, we implement the classes **Organism** and **Environment** whose instances will be the object of study of our virtual experiments.

2.1 Basic Organisms

The basic organisms will resemble single celled species. We will think of them as existing in discrete time, where during each time-step some of their attributes will change. They will be modeled using the following data:

(i) Species name (string): This will help identify organisms of a specific type.

(ii) Size (float): This describe the total amount of 'stuff' inside the organism.

(iii) Uptake rate (float): This is the amount of stuff / food an organism can 'eat' in one time-step.

(iv) Metabolic rate (float): This is the amount of stuff an organism needs to consume per time-step in order to survive.

(v) Division threshold (float): Once an organism reaches this size, it will divide into two daughter organisms

(vi) Alive (bool): This tells us whether the organism is alive or not.

Task: It's Alive!

In this task we will create the basic **Organism** class.

(i) Create a class called **Organism**

(ii) Implement the `__init__` method. The organism should initialize with the following attributes:

- `species`, default to 'unknown'
- `size`
- `division_threshold`
- `uptake_rate`
- `metabolic_rate`

(iii) Implement a method called `update()` which simulates the passing of one time-step. `update()` should take a parameter called `available_food` and return a boolean indicating whether the organism is ready to divide, along with as a float recording how much food the organism ate. Calling `update()` should perform the following:

- set a variable `uptake` to the minimum of `available_food` and `uptake_rate`
- If the organism is alive, increase its size by `uptake`.
- If the organism is alive, decrease its size by `metabolic_rate`
- If the organism's size is above `division_threshold`, return `True, uptake`
- Otherwise, return `True, uptake`

(iv) If no value for `available_food` is specified, `uptake` should just be the uptake rate of the organism.

We will now test the class we have just implemented.

Task: Specimen Zero

We are now ready to create our first instances of **Organism** and see how it behaves.

(i) Create an instance of **Organism** called `organism1` with the following parameters:

- `species = "Specimen Zero"`
- `size = 100`
- `division_threshold = 200`
- `uptake_rate = 5`
- `metabolic_rate = 2`

(ii) Run `organism1.update()` and then print `organism1.size`. Is the result what you expect?

(iii) Let's try to starve the organism. Run `organism1.update(0)` enough times for the organism to die. Verify this by checking the organism's attributes.

2.1.1 Variable Rates

So far each organism has a fixed uptake rate and metabolic rate, specified at its creation. Realistically, both uptake and metabolism will depend on the size of the organism. In this section, we will include this feature in our model.

Biologically, it makes sense to suppose that the uptake rate will be proportional to an organism's surface area, while its metabolic rate will be proportional to its volume. Assuming spherical organisms and that the `size` parameter is proportional to volume, this translates into the following formulas:

$$\text{uptake\_rate} = u * \text{size}^{2/3}$$

$$\text{metabolic\_rate} = m * \text{size}$$

where u and m are constants.

We will now implement a new version of the **Organism** class that can handle variable rates.

Task: Slow Grow

To implement variable rates we must change the **Organism** class to accept functions as `uptake_rate` and `metabolic_rate`. These functions will take `size` as an input and return a number.

- Modify the **Organism** class so that `uptake_rate` and `metabolic_rate` are functions which are evaluated at `size`. (Hint: Any place `uptake_rate` occurs, replace it with `uptake_rate(self.size)`).
- Create an instance of the new **Organism** class with

$$\text{uptake\_rate} = 1/4 * \text{size}^{2/3}$$

$$\text{metabolic\_rate} = 1/50 * \text{size}$$

  Hint: You can use anonymous functions during instantiation:

```
    uptake_rate = lambda x : 1/4 * x**(2/3)
```

- Create an instance of the new **Organism** class with

$$\text{uptake\_rate} = 5$$

$$\text{metabolic\_rate} = 2$$

  Hint: You can use anonymous functions during instantiation:

```
    uptake_rate = lambda x : 5
```

- Run 10^3 organism updates of both instances. Plot their size. What do you observe? Can you explain this behavior?

Following task *Slow Grow* yields the organism growth curves of Figure 1

You can experiment with other rate laws as well. For this it may be helpful to fix a specific geometry of the organism and deduce what laws make sense. For example, if the organisms are 2-dimensional disks, their uptake would be proportional to their 1-dimensional boundary and their metabolism would be proportional to their area. How would these disk organisms fare against their spherical cousins?

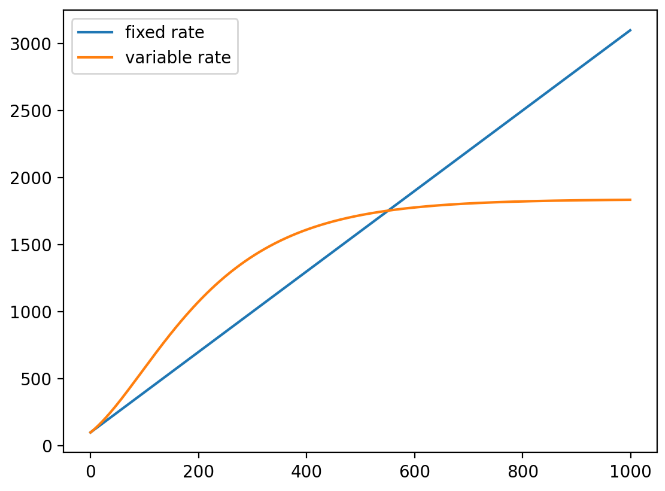

Figure 1: Comparison of organism growth given variable or fixed uptake and metabolism.

2.2 Basic Environments

The **Environment** class will provide a place for our organisms to live. An environment will contain food and a population of organisms, possibly from different species. It will have its own update method. During each update some amount of food will be replenished.

Task: Hello World!

In this task we will create the basic **Environment** class.

(i) Create a class called **Environment**.

(ii) Implement the `__init__` method. The environment should initialize with the following attributes:

- `food` (float): The amount food initially present in the environment.
- `refill_rate` (float): The amount of food added during each time-step.
- `population` (list of Organisms): A list containing all the organisms present in the environment.

(iii) Implement an `update()` method that

- Randomly shuffles a copy of the population list. (Hint: Use the shuffle method in the *random* library)
- For each organism in the shuffled list:
  * Update the organism.
  * Reduce `food` by the amount the organism ate.
  * If the organism returns the division signal `True`, remove the organism from `population`, and add two new instances of **Organism** to the `population` list. These instances should have the same parameters as the original organism, except for `size`, which should be half that of the original organism.
- After all organisms have been updated, increase `food` by the `refill_rate`.
- Counts the total number of divisions that occur in an update cycle, and returns that value.

We now test the **Environment** class we implemented above.

Task: Divide and Conquer

Let's test the **Environment** class.

(i) Create an instance `organism1` of **Organism** as above.

(ii) Create and instance `environment` of **Environment** with

- `food = 10**3`
- `refill_rate = 100`
- `population = [organism1]`.

(iii) Run `environment.update()` 100 times and determine the size of the resulting population.

(iv) Plot the population size against update steps for $5 * 10^3$ updates. What do you observe?

(v) In a separate plot, plot the number of divisions that occur against update steps for $5 * 10^3$ updates. What do you observe?

(vi) Now set `refill_rate = 10**3` and repeat (iv)-(v). How do these results compare to the previous ones?

(vii) Finally set `food = 0` and repeat (iv)-(v). How can you explain the resulting behavior?

A typical example of population growth starting from a single organism can be seen in Figure 2.

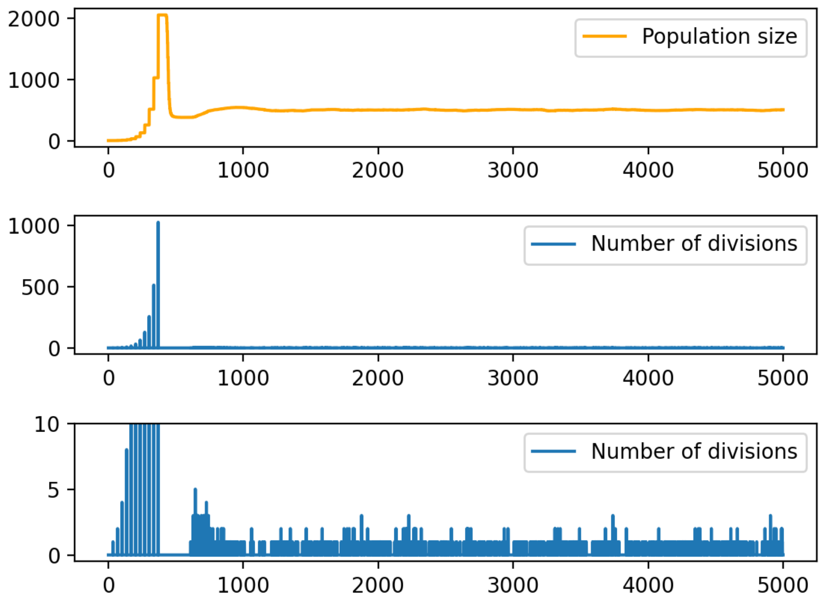

Figure 2: The results of running $5 * 10^3$ updates with `food = 0` and `refill_rate = 10**3`. We observe an overshoot phase followed by a steady state. The organisms divide continually at a low rate in steady state.

3 Natural Selection

In this section we will discover what happens when multiple organisms of different types live in the same environment. Because the food supply is limited and the order in which the organisms eat is randomized, we would expect different organisms to compete with one another for food. When the organisms have different parameters, we can imagine some organisms being fitter than other. These should then win out during the competition. As a result the fitter organisms should increase in number, and the less fit ones decrease. Let's see how this plays out in our virtual setting:

Task: Pick me!

We will now create two different species of organism and see how they interact.

(i) Create an instance `organism_active` of **Organism** with

- `size = 100`
- `division_threshold = 200`
- `uptake_rate = 5`
- `metabolic_rate = 2`

(ii) Create an instance `organism_passive` of **Organism** with

- `size = 100`
- `division_threshold = 200`
- `uptake_rate = 4`
- `metabolic_rate = 1`

(iii) Run $2 * 10^3$ updates of an environment containing 1 `organism_active` and 1 `organism_passive`. Plot the total population, number of `organism_active`, and the number of `organism_passive`. What dou you observe? Can you give an explanation for what is happening?

The results from implementing *Pick me!* can be seen in Figure 3. The more active organism almost immediately dies out once the initial food reserve is used up. If we set a much smaller difference of 0.05 between the parameters of the species, we get the growth curves of Figure 4. Here it is more clearly visible how the passive organism outcompetes the active one over time.

The fact that selection is happening will allow us to perform a type of artificial evolution, provided that we add a source of variation into the mix. This will be the content of several of the *advanced* features presented in the next section.

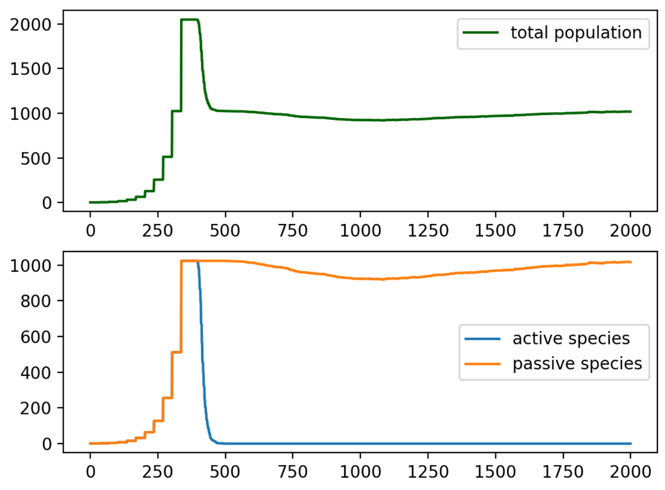

Figure 3: Total, active, and passive organism population resulting from running $2 * 10^3$ updates starting with a population of `[organism_active, organism_passive]`.

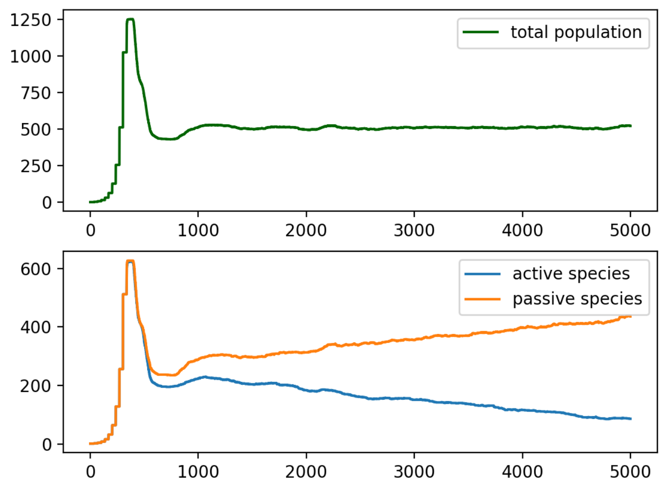

Figure 4: Total, active, and passive organism population resulting from running $5 * 10^3$ updates starting with a population of `[organism_active, organism_passive]`, where the uptake rate of the passive organisms is 4.95 and the metabolic rate is 1.95.

4 Advanced Features

4.1 Advanced Organisms

4.1.1 Metabolite Production

Biological organisms not only take up food, they also excrete various metabolites. These can be waste products, but sometimes they also serve a specific function such as signalling.

Using this as inspiration, we could modify our organisms to also excrete some metabolite at each time step. This could be implemented by adding an `excretion_rate` attribute to the **Organism** class and returning the amount of excreted metabolite in the `update()` method.

Furthermore, it would be desirable to track the total amount of metabolite that has been accumulated in the environment. Doing so would require adding a further attribute to the **Environment** which is incremented each time an organism updates.

Once the accumulated metabolite is tracked one could modify an organism's behavior based on this. For instance, one could suppose that the metabolite is toxic to some organisms and slows their growth rate. This could involve making an organisms `uptake_rate` be inverse proportional to the cumulative amount of metabolite. Alternatively, one could just kill an organism once a certain level of metabolite is reached.

On the other hand, one could imagine that a given type of organism could feed off the metabolite of another. This would mean creating two species, one which lives off the byproducts of the other. Such systems seem like they would display interesting behavior.

4.2 Advanced Environments

4.2.1 Space and Movement

In the *basic* environment, all organisms coexist in the same space. A natural extension would be to divide an environment up into different regions and have every organism only inhabit one region. Another way of thinking about this is that you track where a given organism is living.

The simplest way to implement this idea would be to just have multiple instances of the **Environment** class running at the same time, for example by making a list of instances. This idea could be formalized as a new class whose main attribute is a list of **Environment** instances, and whose update method updates each instance in the list. Interaction between the different environments could the be implemented in this class. For example, one could enable migration of an organism from on instance to another.

An alternative approach would be to introduce coordinates for each organism in an environment. Moreover, one could also introduce coordinates for food sources in the environment and only allow organisms to eat if they are close enough. It would also be natural to allow the organisms to move in this framework. Organisms could then display different patterns of movement, depending on their species. This second approach seems like it would be much more challenging to implement than the first.

4.2.2 Mutation

When two organisms reproduce asexually, there are random mutation that occur. Most of the time these don't affect the viability of the offspring. Sometimes, however, they increase or decrease viability. This leads to a source of variation that can be selected for by natural selection.

In our setting, we could introduce random mutations anytime an organism divides. Recall that the division process is handled by the **Environment** class: Each time an organism's update method returns the divide signal `True`, we remove that organism from the population and replace it with two new daughter instances that have half its size. Critically, all other parameters are left unchanged in this process. To implement random mutation, we therefore have to randomly perturb the parameters in these daughter organisms.

To achieve this, you will probably want to use one of the methods in the *random* library. For example, `random.random()` returns a random number between 0 and 1. This can then be used to add or subtract a random amount from a parameter during division.

Once mutation is implemented you can experiment with different amounts of perturbation and see whether you can observe any selection happening. If a given value of a parameter has a better chance of survival we would expect organisms with that value to be enriched in the population after running the environment for some time.

Note that in order for the mutation mechanism to work, the organisms must be dividing. Recall that we implemented a method to track division rates in the *basic* **Environment** class.

For reading out the variants that are present in the population after running the environment, you can filter all organisms for a specific set of parameter values. Alternatively, you could implement a way of changing the organism's `species` attribute during mutation to reflect its evolutionary history. For instance, you could have `species` be a list tracking all mutations an organism has undergone.

4.2.3 Genetic Exchange

Many microorganisms have ways of passively exchanging genetic information. Many bacteria are able to take up plasmids carrying specific genes. Once taken up, the bacteria will replicate the plasmid as if it were its own genome. This mechanism allows these organisms to quickly adapt to new surroundings. For example, only a single bacteria needs to discover how to become resistent to a given antibiotic and it can then passively share the result with its neighbors.

In terms of our framework, genetic exchange would involve organisms sharing their parameter sets with each other. One could implement this by giving the environment an attribute that can record sets of parameter values. Organisms might then access this attribute and use it to somehow alter their parameter set. Conversely, organisms could deposit their parameter set into the attribute.

Regardless of how this is implemented it would be interesting for the organisms to be able to share their genetic information. If this process happens in an undirected manner, the more successful organisms which are more abundant will automatically deposit more information, so that the external gene-pool will become preferentially enriched with beneficial genes.

4.2.4 Sexual Reproduction

Along similar lines as above one could also think about implementing directed exchange of genetic information between organisms. To achieve this one would have to code a way for two organisms to join together to produce offspring. For example, whenever an organism is ready to divide, it could wait to be paired up with another that is also ready. The result of two organisms reproducing in this way could be daughter organisms that have some combination of the attributes of the parents. For example, you could select parameters at random from both of the parents. Alternatively, you could also average values, although I would suspect this will lead to less interesting results.

4.2.5 Genetic Drift

Genetic drift occurs when there is random sampling from a population (see wikipedia for a detailed explanation). For example, if we repeatedly select a tenth of the population at random for survival each generation and then let them reproduce, certain species may accumulate in frequency and others may die out completely solely due to this random sampling process (in contrast to selection due to their fitness). Genetic drift in our artificial setting could be studied by implementing such a random sampling process. For instance, you could have the environment discard 90% of the organisms in the population at random at regular time intervals. In between the population could be allowed to recover. If you started with multiple species, you could study the resulting distribution of species after such an experiment.

4.2.6 Climate

You could implement further attributes into the environment to simulate further aspects. For instance, you could model the temperature. These attributes could then influence the organisms. Perhaps certain organisms have a favorite temperature to grow at and modulate their uptake accordingly. Perhaps the temperature affects how much food is replenished during each update cycle. Perhaps the temperature increases based on the number of organisms in the population or based on the concentration of a certain metabolite. Feel free to come up with your own ideas here.

4.2.7 Multiple Food Sources

One way to generalize the environment is to model multiple food sources. Different species could have different preferences concerning food. For example, a species might live off food a, but can also eat food b if necessary. However, the species might only receive half the nutritional value from food b. These types of behaviors would also pair nicely with the metabolite production discussed above in Section 4.1.1.

4.2.8 Predation

Rather than just having different food sources, one could have a given species of organism be the food of a species of predators. For this you would need to implement a way for an organism to eat another. This is probably best handled in the **Environment** class: If the species you are updating is a predator, kill one of the prey organism and feed the predator a corresponding amount of food in its update method. One could also give the predator a random chance of catching its prey which could be modulated by the predator's effectiveness at hunting and the prey's ability to escape. Perhaps smaller prey have an easier time escaping. Perhaps prey species can increase their ability to escape at the cost of increasing their metabolic rate. There are many avenues to explore here.

5 Data Collection

Once you have discovered some interesting behavior it's time to collect a bunch of data related to it. In the simplest case, this will involve running an experiment multiple times, perhaps with different parameters, and collecting the resulting data. As an illustration, we will gather population growth data for a set of species with different parameters.

5.1 Example: Population Growth

Let's consider three species a, b, c given as follows from the basic **Organism** class:

```
a = Organism(species = "a", size = 100, division_threshold=200,
     uptake_rate=1,metabolic_rate=0.5)
b = Organism(species = "b", size = 100, division_threshold=200,
     uptake_rate=1.5,metabolic_rate=0.5)
c = Organism(species = "c", size = 100, division_threshold=200,
     uptake_rate=2,metabolic_rate=0.5)
```

For each species we create an environment only containing a single copy of that species:

```
environment_a = Environment(food = 0, refill_rate=10*3,
    population=[a])
environment_b = Environment(food = 0, refill_rate=10*3,
    population=[b])
environment_c = Environment(food = 0, refill_rate=10*3,
    population=[c])
```

Next we update each environment for $5 * 10^3$ steps and record the population size at each time point. A plot of the results can be seen in Figure 5.

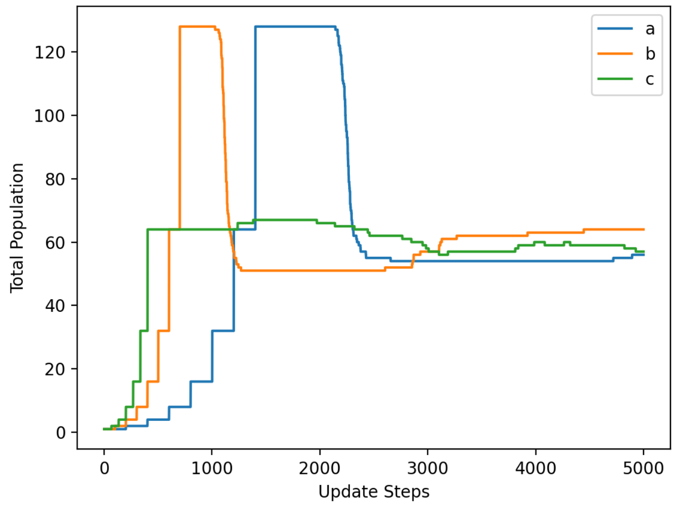

Figure 5: Population growth for for species a, b, c where the uptake rate of a is 1, of b is 1.5, and of c is 2. We observe that the population size increased more rapidly for species with higher uptake rate. The steady state population is comparable for all three species.

Since the results of our experiments depend on randomness, it its desirable to perform each experiment multiple times. For example, we redo the experiment above, but this time in triplicate. The three results for each organism can be seen in Figure 6.

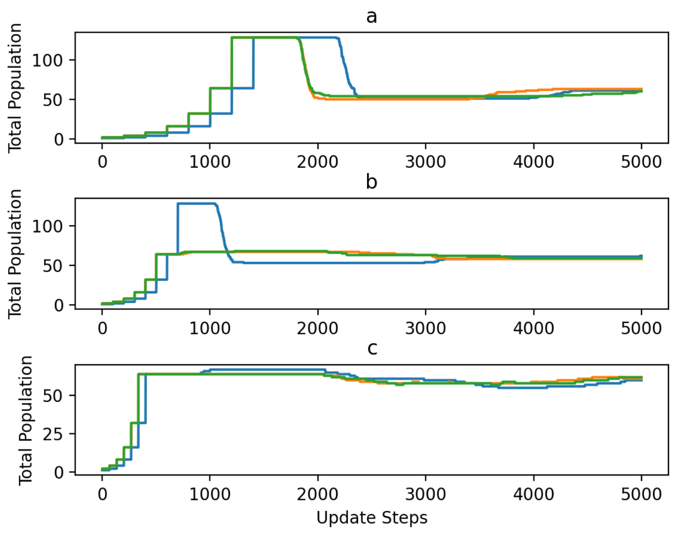

Figure 6: The results of running each experiment in triplicate showing population growth over time for *a* (top), *b* (middle), and *c* (bottom). We see there is some variation between individual trials.

As a final step of the data collection process, we export the data to a file. You should write you own function that opens / creates a file and writes the data to it in a standardized format. The official documentation for reading and writing files can be found here. For example, you can separate each value with commas.

The head of the resulting file could like this:

```
Time-step,Population a trial 1,Population a trial 2,Population
    a trial 3,Population b trial 1,Population b trial 2,
    Population b trial 3,Population c trial 1,Population c
    trial 2,Population c trial 3
0,1,1,1,1,1,1,1,1,1
1,1,1,1,1,2,1,1,1,2
2,1,2,2,1,2,2,1,2,2
3,1,2,2,1,2,2,1,2,2
4,1,2,2,1,2,2,1,2,2
5,1,2,2,1,2,2,1,2,2
6,1,2,2,1,2,2,1,2,2
7,1,2,2,1,2,2,1,2,2
8,1,2,2,1,2,2,1,2,2
```

6 Data Analysis

Now that we have performed some experiments and gathered data it's time to analyze the results. For this you should only rely on the data you have exported in the previous section. If you want to perform follow-up experiments, that's fine, but make sure you don't perform a new experiment each time you redo your analysis.

The following section outlines many methods you could use for analyzing your data. *You are not expected to fully understand any of the methods.* The goal is simply to have some fun exploring your data in various ways. For your report it is sufficient to simply present your results in a reader-friendly way, for example plots. If your journey into exploring data analysis techniques yields something interesting, you can include it, but it will not be expected. Just see this as an open-ended part of the project that you can explore on your own. Knowledge of any of these standard techniques will certainly benefit you in future.

Data analysis can be divided into exploratory data analysis (EDA) and confirmatory data analysis (CDA). The goal of EDA is to find interesting patterns in your data and to summarize these patterns using appropriate measures, while in CDA you start with a hypothesis and see whether it holds in your data set. It is important to differentiate these two approaches as mixing them can lead to methodological errors. For example, if you run different hypothesis tests until you find a hypothesis that yields a significant results, this finding carries much less statistical weight then if you start with a fixed hypothesis and simply keep or reject it. It is important to remember that every statistical test carries certain independence assumptions within it (i.e your hypothesis should not be influenced by the data), and the results will no longer hold if those assumptions are violated. You can read the excellent wikipedia article on data dredging for more information on this phenomenon. It is important that you become aware of this, so that you do not fall prey to it in your own research.

6.1 Visualizing Data

Perhaps the simplest way to understand your data is to represent it visually. We have already been plotting out data to see whats going on. However, if we find a particular feature of interest, it may be worth making visualizations that highlight it. This article provides a short guide to the various plotting options in python. I would recommend sticking to a single plotting library. You can then find more information by looking at the documentation of the library of your choice.

6.2 Descriptive Statistics

Descriptive statistics summarize certain aspects of your data in a single number. Continuing our example from Section 5.1 above, we might wonder how much the results of a fixed experiment vary between trials. One statistic that summarizes this is the empirical variance given by $S^2 = \sum(x_i - \bar{x})^2/(n-1)$, where x_i are the measured sample values, $\bar{x}$ is the average over all sample values, and n is the number of samples. As an example, let's consider the population of organisms a at time 4. In the exported data we see that in trial 1 the population was 1, and in trials 2-3 it was 3. Hence the average value is $(1 + 2 + 2)/3 = 5/3$ and the empirical variance is

$$1/2 * [(1 - 5/3)^2 + (2 - 5/3)^2 + (2 - 5/3)^2] = 1/3.$$

You can find more descriptive statistics along with how to use them in python by consulting the internet, for example here.

6.3 Hypothesis Tests

Hypothesis test offer a way to determine whether you should retain or reject a certain hypothesis based on your data. You can find out about hypothesis test by reading the following wikipedia article. This article shows you how to perform the most common hypothesis tests in Python.

6.4 Model Fitting

Sometimes it is useful to approximate the data in a scatter plot by fitting a mathematical function to it. For example, linear regression finds a straight line through your data that minimizes the square errors of the y coordinates. Alternatively, one can ask the for the squares of the shortest distance between a data point and the curve to be minimal. This is called *least squares* method and can be applied to curves that are not necessarily straight lines. A tutorial of how to apply least squares in Python can be found here.

7 Writing the Report

The most important thing about your report is that it is easy to follow what you did, what results you found, and what you think they mean. Imagine that your reader is someone you care about and that you want to show them what you have been up to during your project.

Don't underestimate the time it takes to write a nice report. It may seem like a hassle, but learning scientific writing will probably benefit you more than most other things. Eventually, many of you will be in a situation where you are excited about something you discovered and want to communicate those results to others. However, if your writing is incoherent, hard to follow, or sloppy, very few people will go through the trouble of reading it. There are always more interesting papers than any reader has time for, so you really have to sell a prospective reader on why they should care about yours. Moreover, you should aim to remove any obstacles for a reader to make it through your paper without giving up and moving on to something else.

Concretely, we will be following the structure of a typical scientific publication in the life sciences. This means the report will be divided into the following sections:

1. Introduction
2. Material and Methods
3. Results
4. Discussion

A very nice overview article of how to write a scientific publication can be found here. Please read this before proceeding.

As you may have noticed, not all of the points in the article will apply to your project. In particular, you are not expected to situate this project within existing research. Moreover, because the project is based on code rather than physical experiments, the material and methods section will have a slightly different character. I will briefly outline what each section should look like, focusing on the deviations from the guide you read above.

7.1 Abstract

Your abstract should be one paragraph including a description of your most important results with minimal interpretation. It should convey to the reader what they can expect to find in your report. Write the abstract last, once you have completed the rest of your report.

7.2 Introduction

Usually an introduction would explain how your project fits into the broader picture of previous research. As this does not really apply to us, you can briefly explain here what the goals of your project were and why you focused on these goals. For example, "We decided to focus on communication between organism, so we modeled ...". The introduction should simply provide the context for what is to come. It is best to write this after you have already written the rest of the report.

7.3 Materials and Methods

This section will deviate most from what is described in the guide you read above because we are working with code. The materials and methods section should objectively state what you did to get your results in such a way that anyone who reads your report could exactly replicate what you did. This means you should for example state that you are using Python 3, and even give the exact version number. To preserve readability, *do not simply paste raw source code into the material and methods section.* Instead, put your commented source code in an appendix if you want it printed in your report, or alternatively list the source files as supplemental data. You can then reference the code by line number in the materials and methods section. For illustration purposes, you may inline small snippets of code according to your needs, but make sure that everything you expect the reader to go over while reading serves a clear purpose and fits into the narrative flow. For example, you could write:

> We implemented a class called “Organism” that has attributed which track ‘species’, ‘size’, and ‘metabolic rate’. (my_code.py lines 10-23).

In addition to describing in general how you arrived at your code and results, you should always comment your code in such a way that a third party is able to (re)use it without issue. Think of the materials and methods as telling the reader what you did broadly, while the code comments give the reader the specifics of how the code works in detail.

You should write this section first. If in doubt about how to write the materials and methods, consult the internet for best practices on writing papers based on code.

7.4 Results

This section is the most straightforward. It can be written after the material and methods. It should simply document the results you obtained from the virtual experiments you performed without yet interpreting the results. You don’t need to include all the results you obtained, only the most interesting ones that you wish to discuss later in the discussion section.

The results should be presented in a reader-friendly format. For example, they can be summarized in an appropriate plot. If you wish to share numerical results, for example those you get from you statistical analyses, you can summarize them in a table. Make sure you include informative captions for your figures and tables. Even a reader who has not read the main text should be able to determine what a figure is displaying by reading the caption.

However, *do no simply paste your raw results into a table.* The results you present should already be summarized for the reader’s convenience. You can include your raw data in a supplementary file that you can reference if the need arises.

7.5 Discussion

In this section you will discuss what you think your results mean. You should write this after having written the results section. You can be creative here as long as your reasoning is sound. The discussion section should communicate to the reader what you have learned from the experiments you described. If your results pose further questions that may be answered in future research, discuss that here.