Computational Thinking and Data Science: Final Project

In this final project, using Python and Pylab you will design and implement a stochastic simulation of patient and virus population dynamics, and reach conclusions about treatment regimens based on the simulation results.

Background: viruses, drug treatments, and computational models

Viruses such as HIV and H1N1 represent a significant challenge to modern medicine. One of the reasons that they are so difficult to treat is their ability to evolve.

As you may know from introductory biology classes, the traits of an organism are determined by its genetic code. When organisms reproduce, their offspring will inherit genetic information from their parent. This genetic information will be modified, either because of mixing of the two parents' genetic information, or through mutations in the genome replication process, thus introducing diversity into a population.

Viruses are no exception. Two characteristics of viruses make them particularly difficult to treat. The first is that their replication mechanism often lacks the error checking mechanisms that are present in more complex organisms. This speeds up the rate of mutation. Secondly, viruses replicate extremely quickly (orders of magnitude faster than humans) – thus, while we may be used to thinking of evolution as a process which occurs over long time scales, populations of viruses can undergo substantial evolutionary changes within a single patient over the course of treatment.

These two characteristics allow a virus population to acquire genetic resistance to therapy quickly. In this project, we will make use of simulations to explore the effect of introducing drugs on the virus population and determine how best to address these treatment challenges within a simplified model.

Computational modeling has played an important role in the study of viruses such as HIV. In this project, we will implement a highly simplified stochastic model of virus population dynamics. Many details have been swept under the rug (host cells are not explicitly modeled and the size of the population is several orders of magnitude less than the size of actual virus populations). Nevertheless, our model exhibits biologically relevant characteristics and will give you a chance to analyze and interpret interesting simulation data.

Spread of a virus in a person

In reality, diseases are caused by viruses and have to be treated with medicine, so in the remainder of this project, we'll be looking at a detailed simulation of the spread of a virus within a person. The file `ps3b.py` reposited in eCampus will be the skeleton of your code.

Problem 1: Implementing a simple simulation (no drug treatment)

We start with a trivial model of the virus population: the patient does not take any drugs and the viruses do not acquire resistance to drugs. We simply model the virus population inside a patient as if it were left untreated.

Class `SimpleVirus`

To implement this model, you will need to fill in the `SimpleVirus` class in the file `ps3b.py`, which maintains the state of a single virus particle. You will implement the methods `__init__`, `getMaxBirthProb`, `getClearProb`, `doesClear`, and `reproduce` according to the specifications. Use `random.random()` for generating random numbers to ensure that your results are consistent with ours.

The `reproduce` method in `SimpleVirus` should produce an offspring by returning a new instance of `SimpleVirus` with probability: `self.maxBirthProb * (1 - popDensity)`. This method raises a `NoChildException` if the virus particle does not reproduce.

`self.maxBirthProb` is the birth rate under optimal conditions (the virus population is negligible relative to the available host cells so there is ample nourishment available).
`popDensity` is defined as the ratio of the current virus population to the maximum virus population for a patient and should be calculated in the `update` method of the `Patient` class.

Class `Patient`

You will also need to implement the `Patient` class in `ps3b.py`, which maintains the state of a virus population associated with a patient.

The `update` method in the `Patient` class is the inner loop of the simulation. It modifies the state of the virus population for a single time step and returns the total virus population at the end of the time step. At every time step of the simulation, each virus particle has a fixed probability of being cleared (eliminated from the patient's body). If the virus particle is not cleared, it is considered for reproduction. The virus population should never exceed `maxPop`;

if you utilize the population density correctly, you shouldn't need to provide an explicit check for this.

Unlike the clearance probability, which is constant, the probability of a virus particle reproducing is a function of the virus population. With a larger virus population, there are fewer resources in the patient's body to facilitate reproduction, and the probability of reproduction will be lower. One way to think of this limitation is to consider that virus particles need to make use of a patient's cells to reproduce; they cannot reproduce on their own. As the virus population increases, there will be fewer available host cells for viruses to utilize for reproduction.

To summarize, `update` should first decide which virus particles are cleared and which survive by making use of the `doesClear` method of each `SimpleVirus` instance, then update the collection of `SimpleVirus` instances accordingly. With the surviving `SimpleVirus` instances, update should then call the `reproduce` method for each virus particle. Based on the population density of the surviving `SimpleVirus` instances, `reproduce` should either return a new instance of `SimpleVirus` representing the offspring of the virus particle, or raise a `NoChildException` indicating that the virus particle does not reproduce during the current time step. The `update` method should update the attributes of the patient appropriately under either of these conditions. After iterating through all the virus particles, the `update` method returns the number of virus particles in the patient at the end of the time step.

Note that the mapping between time steps and actual time will vary depending on the type of virus being considered, but for this project, think of a time step as a simulated hour of time.

Problem 2: Running and analyzing a simple simulation (no drug treatment)

You should start by understanding the population dynamics before introducing any drug.

Fill in the function `simulationWithoutDrug(numViruses, maxPop, maxBirthProb, clearProb, numTrials)` in `ps3b.py` that instantiates a `Patient`, simulates changes to the virus population for 300 time steps (i.e., 300 calls to `update`), and plots the average size of the virus population as a function of time; that is, the x-axis should correspond to the number of elapsed time steps, and the y-axis should correspond to the average size of the virus population in the patient. The population at time = 0 is the population after the first call to `update`.

Run the simulation for `numTrials` trials, where `numTrials` in this case can be up to 100 trials. Use `pylab` to produce a plot (with a single curve) that displays the average result of running the simulation for many trials. Make sure you run enough trials so that the resulting plot does not change much in terms of shape and time steps taken for the average size of the virus population to become stable. Don't forget to include axes labels, a legend for the curve, and a title on your plot.

You should call `simulationWithoutDrug` with the following parameters:

- `numViruses` = 100
- `maxPop` (maximum sustainable virus population) = 1000
- `maxBirthProb` (maximum reproduction probability for a virus particle) = 0.1
- `clearProb` (maximum clearance probability for a virus particle) = 0.05

Thus, your simulation should be instantiating one `Patient` with a list of 100 `SimpleVirus` instances. Each `SimpleVirus` instance in the viruses list should be initialized with the proper values for `maxBirthProb` and `clearProb`.

Consult reference documentation on pylab as reference. Scroll down on the page to find a list of all the plotting commands in `pylab`.

For testing, we have provided the .pyc (compiled Python) files for the completed `Patient` and `SimpleVirus` classes (and for Problem 4, the `ResistantVirus` and `TreatedPatient` classes) that you can use to confirm that your code is generating the correct results during simulation.

If you comment out your versions of these classes in `ps3b.py`, and add the following import statements to the top of your file, you can run the simulation using our pre-compiled implementation of these classes to make sure you are obtaining the correct results. This is a good way to test if you've implemented these classes correctly. Make sure to comment out the import statement and uncomment your implementations before moving to Problem 3.

```
from ps3b_precompiled_27 import *
```

Problem 3: Implemening a simulation with drugs

In this problem, we consider the effects of both administering drugs to the patient and the ability of virus particle offsprings to inherit or mutate genetic traits that confer drug resistance. As the virus population reproduces, mutations will occur in the virus offspring, adding genetic diversity to the virus population. Some virus particles gain favorable mutations that confer resistance to drugs.

Class `ResistantVirus`

In order to model this effect, we need you to implement a subclass of `SimpleVirus` called `ResistantVirus` in `ps3b.py`. `ResistantVirus` maintains the state of a virus particle's drug resistances, and accounts for the inheritance of drug resistance traits to offspring.

Class `TreatedPatient`

We also need a representation for a patient that accounts for the use of drug treatments and manages a collection of `ResistantVirus` instances. For this, please implement the `TreatedPatient` class, which is a subclass of `Patient`. `TreatedPatient` must make use of the new methods in `ResistantVirus` and maintain the list of drugs that are administered to the patient.

Drugs are given to the patient using the `TreatedPatient` class's `addPrescription()` method. What happens when a drug is introduced? The drugs we consider do not directly kill virus particles lacking resistance to the drug, but prevent those virus particles from reproducing (much like actual drugs used to treat HIV). Virus particles with resistance to the drug continue to reproduce normally.

Problem 4: Running and analyzing a simulation with a drug

In this problem, we will use the implementation you made for Problem 3 to run a simulation. You will create a `TreatedPatient` instance with the following parameters, then run the simulation:

- `viruses`, a list of 100 `ResistantVirus` instances
- `maxPop`, maximum sustainable virus population = 1000

Each `ResistantVirus` instance in the viruses list should be initialized with the following parameters:

- `maxBirthProb`, maximum reproduction probability for a virus particle = 0.1
- `clearProb`, maximum clearance probability for a virus particle = 0.05

- `resistances`, The virus's genetic resistance to drugs in the experiment = `{'guttagonol': False}`

- `mutProb`, probability of a mutation in a virus particle's offspring = 0.005

Run a simulation that consists of 150 time steps, followed by the addition of the drug, guttagonol, followed by another 150 time steps. You should make use of the function `simulationWithDrug(numViruses, maxPop, maxBirthProb, clearProb, resistances, mutProb, numTrials)`. As with problem 2, perform up to 100 trials and make sure that your results are repeatable and representative.

Create one plot that records both the average total virus population and the average population of guttagonol-resistant virus particles over time. (A few good questions to consider as you look at your plots are: What trends do you observe? Are the trends consistent with your intuition?)

Again, as in Problem 2, you can use the provided `.pyc` file to check that your implementation of the TreatedPatient and ResistantVirus classes work as expected.