In this assignment, we used Python and Pylab to design and implement a stochastic simulation of patient and virus population dynamics, and reach conclusions about treatment regimens based on the simulation results. In biology class, you learn that traits of an organism are determined by its genetic code. When organisms reproduce, their offspring will inherit genetic info. from their parent. Their genetic info. will be modified due to mixing or 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 2 characteristics allow a virus population to acquire genetic resistance to therapy quickly. In this problem set, we make use of simulations to explore the effect of drugs on the virus population and determine how best to address the treatment challenges in a simplified model.

In this problem, we implemented a highly simplified stochastic model of virus population dynamics.

Many details were 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 a chance to analyze and interpret interesting simulation data.

In reality, diseases are caused by viruses and have to be treated with medicine, so in this problem set we looked at a detailed simulation of the spread of a virus within a person. Skeleton starter project code was provided by the professors.

We first implemented a simple simulation with no drug treatment. We started with a trivial model of the virus population. The patient does not take any drugs and the viruses do not acquire any resistance to drugs. We simply modeled the virus population inside a patient as if it were left untreated.

The SimpleVirusClass maintains the state of a single virus particle. We implemented the \_\_init\_\_, doesClear and reproduce methods. We used random.random for generating random numbers to ensure the results are consistent.

The reproduce method in SimpleVirus should produce an offspring by returning a new instance of SimpleVirus with probability: self.maxBirthProb \* (1 – popDensity). A NoChildException is raised 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 and should be calculated in the update method of the Patient class.

The Patient class maintains the state of the 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 clearance probability, which is constant, the probability of a virus particle reproducing is a funciton 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 on their own. As the virus population increases, there will be fewer available host cells for viruses to utilize for reproduction.

To summarize, update had to 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. A time step is simulated as an hour of time in this problem set.

We next ran and analyzed a simple simulation with no drug treatment. This was done because you should understand the population dynamics before introducing any drug. We had to fill in a function: simulationWithoutDrug(numViruses, maxPop, maxBirthProb, clearProb, numTrials) that instantiates a Patient, simulates changes to the virus population for 300 time steps (ie, 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 simulation was run for numTrials, where numTrials in this case can be up to 100 trials. Pylab was used to produce a plot with a single curve that displays the average result of running the simulation for many trials. You have to run for enough trials such 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. We had to include axes labels, a key for the curve, and a title on the plot.

We used the following parameters for simulationWithoutDrug: 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 this simulation instantiated 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.

The graph had to contain one line that represented the average of many different trials. One way of computing the average involves holding all of your data in one list, with one element for each of the 300 time steps, and adding to each data point during each trial. Then, at the end, each element of the list is divided by the total number of trials, thus taking the average of each element of the list.

Testing simulation code is more challenging because the behavior of the code is stochastic, and the expected output is not exactly known. One way to know whether your plots are correct or not is to run the simulation with extreme input values (i.e. initialization parameters), and check that the output matches your intuition. For example, if maxBirthProb is set to 0.99 instead of 0.1, then you would expect that the virus population rapidly increases over a short period of time. Similarly, if you run your simulations with clearProb = 0.99 and maxBirthProb = 0.1, then you should see the virus population quickly decreasing within a small number of steps. You can also try to vary the input values, and check whether the output plots change as you expect. For example, if you run multiple simulation runs, each time increasing maxBirthProb, the curves in the successive plots should show an “upward” trend, since the virus will reproduce faster with a higher maxBirthProb. A good question we had to answer when we examined our plots was: about how long does it take before the population stops growing, such as about 50 time-steps, 100 time-steps, 150, 200, and 250 time-steps.

We then had to implement a simulation with a drug. We had to consider the effects of both administering drugs to the patient and the ability of the 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.

We next implemented a ResistantVirus class as a subclass of SimpleVirus. ResistantVirus maintains the state of a virus particles drug resistances, and accounts for the inheritance of drug resistance traits to offspring.

We next implemented the TreatedPatient class as a subclass of Patient. It accounted for a patient's use of drug treatments and manages a collection of ResistantVirus instances. TreatedPatient must make use of the new methods in ResistantVirus and maintain a 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 the drugs are introduced? The drugs we consider do not directly kill virus particles lacking resistance to the drug, but prevent those virus particles from reproducing. Virus particles with resistance to the drug continue to reproduce normally.

We then ran and analyzed a simulation with a drug. We created a TreatedPatient class with the following parameters: viruses = a list of 100 ResistantVirus instances, maxPop = maximum sustainable virus population = 1000. Each ResistantVirus instance in the viruses list were 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 = {guttanol: False}, mutProb = probability of a mutation in a virus particle's offspring = 0.005.

We ran a simulation that consisted of 150 time-steps, followed by the addition of the drug, guttanol, followed by another 150 time-steps. The function was: simulationWithDrug(numViruses, maxPop, maxBirthProb, clearProb, resistances, mutProb, numTrials). We performed up to 100 trials to make sure our results were repeatable and representative.

We created a plot that recorded both the average total virus population and the average population of guttanol-resistant virus particles over time.

Good questions to ask ourselves when examing our plots were: What trends do you observe? Are the trends consistent with your intuition?