Problem Set 7: Simulating the Spread of Disease and Virus Population

Handed Out: Lecture 14.
Due: 11:59pm, Lecture 16.

Introduction

In this problem set, 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.

You should submit 3 files for this problem set: your code in **`ps7.py`** `and` **`ps7b.py`**, and a write-up in pdf format called **`writeup.pdf`**. (you may find this online pdf converter useful).

Workload : Please let us know how long you spend on each problem. We want to be careful not to overload you by giving out problems that take longer than we anticipated.

Getting Started

Download: ps7.py

For this lab, you will be using pylab to plot the results of simulation runs, which you should already have from Problem Set 6.

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 because of 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 quickly acquire genetic resistance to therapy. In this problem set, 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 (for example, see this paper, by MIT graduate David Ho). In this problem set, 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 problem set, we'll be looking at a detailed simulation of the spread of a virus within a Person. We've provided you with skeleton code in `ps7.py`.

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.

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

Hint: during debugging, you might want to use `random.seed()` so that your results are reproducible.

The `reproduce` method in `SimpleVirus` should produce an offspring by returning a new instance of `SimpleVirus` with probability:

`self.maxBirthProb` * (1 – `popDensity`)

`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 `SimplePatient` class.

```
class SimpleVirus(object):
    """
    Representation of a simple virus (does not model drug
effects/resistance).
    """

    def __init__(self, maxBirthProb, clearProb):
        """
        Initialize a SimpleVirus instance, saves all parameters as attributes
        of the instance.

        maxBirthProb: Maximum reproduction probability (a float between 0-1)

        clearProb: Maximum clearance probability (a float between 0-1).
        """
        # TODO

    def doesClear(self):
        """
        Stochastically determines whether this virus particle is cleared from
the
        patient's body at a time step.

        returns: True with probability self.clearProb and otherwise returns
False.
        """
        # TODO

    def reproduce(self, popDensity):
        """
        Stochastically determines whether this virus particle reproduces at a
        time step. Called by the update() method in the SimplePatient and
        Patient classes. The virus particle reproduces with probability
        self.maxBirthProb * (1 - popDensity).

        If this virus particle reproduces, then reproduce() creates and
returns
        the instance of the offspring SimpleVirus (which has the same
        maxBirthProb and clearProb values as its parent).

        popDensity: the population density (a float), defined as the current
        virus population divided by the maximum population.

        returns: a new instance of the SimpleVirus class representing the
        offspring of this virus particle. The child should have the same
        maxBirthProb and clearProb values as this virus. Raises a
        NoChildException if this virus particle does not reproduce.
        """
        # TODO
```

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

The `update` method in the `SimplePatient` 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). Then if the virus particle is not cleared, it is considered for reproduction.

(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 and 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.

HINT: Be very wary about mutating an object while iterating over its elements. It is best to avoid this entirely (consider introducing additional "helper" variables).

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

You will test your implementation in problem 2.

```
class SimplePatient(object):
    """
    Representation of a simplified patient. The patient does not take any
drugs
    and his/her virus populations have no drug resistance.
    """

    def __init__(self, viruses, maxPop):
        """
        Initialization function, saves the viruses and maxPop parameters as
        attributes.

        viruses: The list representing the virus population (a list of
        SimpleVirus instances)

        maxPop: The  maximum virus population for this patient (an integer)
        """
        # TODO

    def getTotalPop(self):
        """
        Gets the current total virus population.

        returns: The total virus population (an integer)
        """
        # TODO

    def update(self):
        """
        Update the state of the virus population in this patient for a single
        time step. update() should execute the following steps in this order:

        - Determine whether each virus particle survives and updates the list
          of virus particles accordingly.

        - The current population density is calculated. This population
density
          value is used until the next call to update()

        - Determine whether each virus particle should reproduce and add
          offspring virus particles to the list of viruses in this patient.

        returns: the total virus population at the end of the update (an
        integer)
        """
        # TODO
```

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()`. This method should instantiate a `SimplePatient` and repeatedly call the `update` method to simulate changes in the virus population over time. Save the population values over the course of the simulation and use pylab to plot the virus population as a function of time. Be sure to title and label your plot.

`SimplePatient` should be instantiated with the following parameters:

- `viruses`, a list of 100 `SimpleVirus` instances
- `maxPop`, Maximum Sustainable Virus Population = 1000

Each `SimpleVirus` 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

Fill in the function `simulationWithoutDrug()` that instantiates a patient, simulates changes to the virus population for 300 time steps (i.e. 300 calls to `update`), and plots the virus population as a function of time; that is, the x-axis should correspond the number of elapsed time steps, and y-axis the population of the virus in the patient. Run the simulation many times (without setting the seed). Produce a single plot that is representative of the average case as a result of running the simulation many times. Don't forget to include axes labels, a key for the curves, and a title on your plot.

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

To add the plot to your writeup, click on the disk icon at the bottom of the figure window and save an image file. Use Microsoft Word (or your word processor of choice) to import the image file.

In your writeup, include the plot and answer this question: about how long does it take before the population stops growing?

```
def simulationWithoutDrug():
    """
    Run the simulation and plot the graph for problem 2 (no drugs are used,
    viruses do not have any drug resistance).

    Instantiates a patient, runs a simulation for 300 timesteps, and plots
the
    total virus population as a function of time.
    """
    # TODO
```

HINT: Compared to the previous problem sets, testing your simulation code is more challenging, because the behavior of the code is stochastic, and the expected output is not exactly known. How do you know whether your plots are correct or not? One way to test this is to run the simulation with extreme input values (i.e. initialization parameters), and check that the output matches your intution. For example, if `codemaxBirthProb` 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 simulation 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 simuation 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`.

To be continued in Problem Set 8 …

Problem 3 : Probabilities

This part of the problem set involves some pencil-and-paper exercises, as well as a brief bit of coding. It will help you practice and understand simple probability and statistics. Include the answers for parts 1 and 2 as formulas in your writeup, and the code from part 3 in the file **`ps7b.py`**.

1. You flip a fair coin 3 times, write down the probability of the below events. Assume all sequences are equally likely.
    1. Three heads: A : {H,H,H}
    2. The sequence head, tail, head: A : {H,T,H}
    3. Any sequence with 2 heads and 1 tail
    4. Any sequence where the number of heads is greater than or equal to the number of tails
2. What is the probability of rolling a Yahtzee! on the first roll? That is, what is the probability of rolling five 6-sided dice, and having them all display the same number?
3. Write a Monte Carlo simulation to solve the above problem (the Yahtzee problem), and submit your code as `ps7b.py`.

Hand-In Procedure

1. Save

Save your solutions as **`ps7.py`** and **`ps7b.py`**. Your writeup should be called **`writeup.pdf`**

2. Time and Collaboration Info

At the start of each file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people you collaborated with. For example:

```
# Problem Set 7
# Name: Jane Lee
# Collaborators: John Doe
# Time: 3:30
#
... your code goes here ...
```

3. Sanity checks

After you are done with the problem set, do sanity checks. Run the code and make sure it can be run without errors. You should never submit code that immediately generates an error when run!

Also, make sure that your writeup contains everything we've asked for.