Problem Set 8: Simulating the Spread of Disease and Virus Population (continued from Problem Set 7)

Handed Out: Lecture 16
Due: 11:59pm, Lecture 18.

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 2 files for this problem set: your code in `ps8.py` and a write-up in pdf format called `ps8_writeup.pdf`. (you may find this online pdf converter useful). **Your writeup should only be a couple sentences per problem. We are not looking for long expositions. Just include the requested graphs and answer the questions completely and concisely.**

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

For this lab, you will be using the classes from Problem Set 7. Feel free to use your own implementations from Problem Set 7, or you may use the compiled version provided in the download above.

Problem 1: Implementing a Simulation With Drugs

For this part, use the skeleton code provided in `ps8.py`.

In this problem, we consider the effects of both administering drugs to the patient and the ability of virus particle offspring 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.

In order to model this effect, we introduce a subclass of `SimpleVirus` called `ResistantVirus`. `ResistantVirus` maintains the state of a virus particle's drug resistances, and account for the inheritance of drug resistance traits to offspring. Implement the `ResistantVirus` class.

You will test your implementation in problem 2.

```
class ResistantVirus(SimpleVirus):
    """
```

```
    Representation of a virus which can have drug resistance.
    """

    def __init__(self, maxBirthProb, clearProb, resistances, mutProb):
        """
        Initialize a ResistantVirus 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).

        resistances: A dictionary of drug names (strings) mapping to the
state
        of this virus particle's resistance (either True or False) to each
drug.
        e.g. {'guttagonol':False, 'grimpex',False}, means that this virus
        particle is resistant to neither guttagonol nor grimpex.

        mutProb: Mutation probability for this virus particle (a float). This
is
        the probability of the offspring acquiring or losing resistance to a
drug.
        """
        # TODO

    def isResistantTo(self, drug):
        """
        Get the state of this virus particle's resistance to a drug. This
method
        is called by getResistPop() in Patient to determine how many virus
        particles have resistance to a drug.

        drug: The drug (a string)

        returns: True if this virus instance is resistant to the drug, False
        otherwise.
        """
        # TODO

    def reproduce(self, popDensity, activeDrugs):
        """
        Stochastically determines whether this virus particle reproduces at a
        time step. Called by the update() method in the Patient class.

        A virus particle will only reproduce if it is resistant to ALL the
drugs
        in the activeDrugs list. For example, if there are 2 drugs in the
        activeDrugs list, and the virus particle is resistant to 1 or no
drugs,
        then it will NOT reproduce.

        Hence, if the virus is resistant to all drugs
        in activeDrugs, then the virus reproduces with probability:

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

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

        For each drug resistance trait of the virus (i.e. each key of
        self.resistances), the offspring has probability 1-mutProb of
        inheriting that resistance trait from the parent, and probability
        mutProb of switching that resistance trait in the offspring.

        For example, if a virus particle is resistant to guttagonol but not
        grimpex, and `self.mutProb` is 0.1, then there is a 10% chance that
        that the offspring will lose resistance to guttagonol and a 90%
        chance that the offspring will be resistant to guttagonol.
        There is also a 10% chance that the offspring will gain resistance to
        grimpex and a 90% chance that the offspring will not be resistant to
        grimpex.

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

        activeDrugs: a list of the drug names acting on this virus particle
        (a list of strings).

        returns: a new instance of the ResistantVirus 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
```

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 we introduce the `Patient` class, which is a subclass of `SimplePatient`. `Patient` 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 Patient 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. Implement the `Patient` class.

```
class Patient(SimplePatient):
    """
    Representation of a patient. The patient is able to take drugs and
his/her
    virus population can acquire resistance to the drugs he/she takes.
    """

    def __init__(self, viruses, maxPop):
        """
        Initialization function, saves the viruses and maxPop parameters as
        attributes. Also initializes the list of drugs being administered
```

```
        (which should initially include no drugs).

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

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

    def addPrescription(self, newDrug):
        """
        Administer a drug to this patient. After a prescription is added, the
        drug acts on the virus population for all subsequent time steps. If
the
        newDrug is already prescribed to this patient, the method has no
effect.

        newDrug: The name of the drug to administer to the patient (a
string).

        postcondition: The list of drugs being administered to a patient is
updated
        """
        # TODO

    def getPrescriptions(self):
        """
        Returns the drugs that are being administered to this patient.

        returns: The list of drug names (strings) being administered to this
        patient.
        """
        # TODO

    def getResistPop(self, drugResist):
        """
        Get the population of virus particles resistant to the drugs listed
in
        drugResist.

        drugResist: Which drug resistances to include in the population (a
list
        of strings - e.g. ['guttagonol'] or ['guttagonol', 'grimpex'])

        returns: The population of viruses (an integer) with resistances to
all
        drugs in the drugResist list.
        """
        # TODO

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

        - Determine whether each virus particle survives and update 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.
          The listof drugs being administered should be accounted for in the
          determination of whether each virus particle reproduces.

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

Problem 2: Running and Analyzing a Simulation with a Drug

In this problem, we will use the implementation you filled-in for problem 1 to run a simulation. You will create a `Patient` instance with the following parameters, then run the simulation and answer several questions:

- `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. As with problem 2 (of P-set #7), perform multiple trials and make sure that your results are repeatable and representative.

Create 2 plots: the record of the average total population and the average population of guttagonol-resistant virus particles.

In your writeup, include these plots and answers to the questions: What trends do you observe? Are the trends consistent with your intuition?

```
def simulationWithDrug():
    """
    Runs simulations and plots graphs for problem 4.
```

```
    Instantiates a patient, runs a simulation for 150 timesteps, adds
    guttagonol, and runs the simulation for an additional 150 timesteps.

    total virus population vs. time  and guttagonol-resistant virus
population
    vs. time are plotted
    """
    # TODO
```

Problem 3: The Effect of Delaying Treatment on Patient Outcome

In this problem, we explore the effect ofdelaying treatment on the ability of the drug to eradicate the virus population. You will need to run multiple simulations to observe trends in the distributions of patient outcomes.

Run the simulation for 300, 150, 75, and 0 time steps before administering guttagonol to the patient. Then run the simulation for an additional 150 time steps. Use the same initialization parameters for ResistantVirus and Patient as you did for Problem 2.

For each of these 4 conditions, repeat the experiment enough times to get a reasonable insight into the expected result (i.e. 30 times). Use pylab's `hist()` function to plot a histogram of the final virus populations under each condition. You may also find pylab's `subplot` function to be helpful. The x-axis of the histogram should be the final total virus population values (choose x axis
increments or "histogram bins" according to the range of final virus population values you get by running the simulation multiple times). Then the y-axis of the histogram should be the number of patients belonging to each histogram bin. You should decide the number of times you need to repeat each condition in order to obtain a reasonable distribution. Justify your decision in your writeup.

HINT: It may take some time to run enough trials to arrive at a distribution for each condition. Debug your code using a small number of trials. Once your code is debugged, use a larger number of trials and expect the simulation to take a few minutes. Use print statements to monitor the simulation's progress. The simulation should take about 3-6 minutes to run a reasonable number of trials.

In your writeup, include the four histograms (one for each condition of 300, 150, 75, and 0 time step delays) and justify how many times you repeated each condition for a reasonable distribution. Then answer the following questions: If you consider final virus particle counts of 0-50 to be cured (or in remission), what percentage of patients were cured (or in remission) at the end of the simulation? What is the relationship between the number of patients cured (or in remission) and the delay in treatment? Explain how this relationship arises from the model.

```
def simulationDelayedTreatment():
    """
```

```
    Runs simulations and make histograms for problem 5.

    Runs multiple simulations to show the relationship between delayed
treatment
    and patient outcome.

    Histograms of final total virus populations are displayed for delays of
300,
    150, 75, 0 timesteps (followed by an additional 150 timesteps of
    simulation).
    """
    # TODO
```

Problem 4: Designing a Treatment Plan with Two Drugs

One approach to addressing the problem of acquired drug resistance is to use cocktails – administration of multiple drugs that act independently to attack the virus population.

In problems 4 and 5, we use two independently-acting drugs to treat the virus. We will use this model to decide the best way of administering the two drugs. Specifically, we examine the effect of a lag time between administering the first and second drugs on patient outcomes.

For problems 4-5, use the following parameters to initialize a Patient:

- `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, grimpex:False}
- `mutProb`, Probability of a mutation in a virus particle's offspring = 0.005

Run the simulation for 150 time steps before administering guttagonol to the patient. Then run the simulation for 300, 150, 75, and 0 time steps before administering a second drug, grimpex, to the patient. Finally, run the simulation for an additional 150 time steps.

For each of these 4 conditions, repeat the experiment enough times to get a reasonable condition (i.e. 30 times), while recording the final virus populations. Use pylab's `hist()` function to plot a histogram of the final total virus populations under each condition.

HINT: As with problem 3, the simulation will take a few minutes to run. Use print statements to monitor the simulation's progress.

In your writeup, include the 4 histograms (for 300, 150, 75, and 0 time steps) and answer the following: What percentage of patients were cured (or in remission) at the end of the simulation? What is the relationship between the number of patients cured (or in remission) and the time between administering the two drugs?

```
def simulationTwoDrugsDelayedTreatment():
    """
    Runs simulations and make histograms for problem 6.

    Runs multiple simulations to show the relationship between administration
    of multiple drugs and patient outcome.

    Histograms of final total virus populations are displayed for lag times
of
    150, 75, 0 timesteps between adding drugs (followed by an additional 150
    timesteps of simulation).
    """
    # TODO
```

Problem 5: Analysis of Virus Population Dynamics with Two Drugs

To better understand the relationship between patient outcome and the time between administering the drugs, we examine the virus population dynamics of two individual simulations from problem 4 in more detail.

Run a simulation for 150 time steps before administering guttagonol to the patient. Then run the simulation for an additional 300 time steps before administering a second drug, grimpex, to the patient. Then run the simulation for an additional 150 time steps. Use the same initialization parameters for `Patient` and `ResistantVirus` as you did for problem 4.

Run a second simulation for 150 time steps before simultaneously administering guttagonol and grimpex to the patient. Then run the simulation for an additional 150 time steps. Make sure you run the simulation multiple times to ensure that you are analyzing results that are representative of the most common outcome.

For both of these simulations, plot the total population, the population of guttagonol-resistant virus, the population of grimpex-resistant virus, and the population of viruses that are resistant to both drugs as a function of time.

In your writeup, include the plots from both simulations and explain the relationship between the patient outcome and the time between administering the two drugs arises.

```
def simulationTwoDrugsVirusPopulations():
    """
    Run simulations and plot graphs examining the relationship between
    administration of multiple drugs and patient outcome.

    Plots of total and drug-resistant viruses vs. time are made for a
```

```
    simulation with a 300 time step delay between administering the 2 drugs
and
    a simulations for which drugs are administered simultaneously.
    """
    # TODO
```

Problem 6: Patient Non-compliance

A very common problem is that a patient may not consistently take the drugs they are prescribed. They can sometimes forget or refuse to take their drugs. Describe in your writeup (do not write any code) how you would model such effects.

Hand-In Procedure

1. Save

Save your solutions as **`ps8.py`**. Your writeup should be called **`ps8_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 8
# 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.

6.00SC Introduction to Computer Science and Programming
Spring 2011