Part B: Problem 1

Bookmark this page

Problem Set due Dec 2, 2022 07:30 +08 Completed

Part B: Problem 1: Implementing a Simple Simulation (No Drug Treatment)

15.0/15.0 points (graded)

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.

SimpleVirus class

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__`, `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.

Hint: random seed

During debugging, you might want to use `random.seed(0)` 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)`. This method raises a `NoChildException` if the virus particle does not reproduce. For a reminder on raising execptions, review the Python docs.

`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.

Patient class

You will also need to implement the `Patient` class, 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. If you utilize the population density correctly, you shouldn't need to provide an explicit check that the virus population exceeds `maxPop` when you are calculating how many offspring are added to the population -- you just calculate the new population density and use that for the next call to `update`.

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.

Hint: mutating objects

Be very wary about mutating an object while iterating over its elements. It is best to avoid this entirely (consider introducing additional "helper" variables). See the 6.00.2x Style Guide for more information.

Note that the mapping between time steps and actual time will vary depending on the Calculator g considered, bu

About the grader: When defining a `Patient` class member variable to store the viruses list representing the virus population, please use the name `self.viruses` in order for your code to be compatible with the grader and to pass one of the tests.

Note: If you want to use numpy arrays, you should add the following lines at the beginning of your code for the grader:
`import os`
`os.environ["OPENBLAS_NUM_THREADS"] = "1"`
Then, do `import numpy as np` and use `np.METHOD_NAME` in your code.

```
# Part B: Problem 1
#  Bookmark this page
# Part B: Problem 1: Implementing a Simple Simulation (No Drug Treatment)
# 15.0/15.0 points (graded)
# We start with a trivial model of the virus population - the patient does not take any drugs and the vir
# drugs. We simply model the virus population inside a patient as if it were left untreated.

# SimpleVirus class
```