

**MODELING THE COURSE OF A VIRAL ILLNESS AND ITS TREATMENT
(COMPUTER SIMULATION)**

May 2023

Rohan Saketh Pilli
Department of Mathematics, BASIS Independent Fremont

ABSTRACT

This study explores the computational modeling of viral illness progression and treatment through computational simulation, integrating calculus, computer science, and biology. The project employs Python programming to simulate the replication of viral particles and the body's immune response using differential equations and Euler's method for accurate time-stepped calculations.

The model investigates several scenarios, including unmitigated viral growth, immune response activation, and medication-assisted recovery at certain initial, but alterable, values, such as 10000 initial virus particles and an upper cap of 10^{12} virus particles. Simulations reveal that immune responses trigger when viral particles exceed 1 million, reducing replication rates from doubling every four hours to increasing by 60% in the same timeframe.

Using the Intermediate Value Theorem, the study determines that administering medication between 10.2 million and 10.3 million viral particles is crucial for patient survival, demonstrating the critical importance of medication timing. It is demonstrated that there are three phases depending on vaccination time. The patient's viral count goes to 0, the viral count exponentially increases, or there is a stagnant state in which the viral count stays relatively constant over a reasonable period of time. The model also optimizes minimal medication dosage, considering kidney filtration rates and immune response. An ideal administration rate of approximately 24.25 $\mu\text{g}/\text{hour}$ is identified to maintain effective drug levels above 90 micrograms in the patient's system.

The study also explores the concept of an equilibrium state where viral count remains stagnant under certain conditions, highlighting the complex dynamics between the different parameters. The simulation allows for parameter adjustments, enabling personalized modeling for different patient profiles, such as children or individuals with varying immune responses.

This study demonstrates the efficacy of computational tools in replicating complex biological interactions and informing medical intervention strategies. By providing a flexible framework for simulating viral progression and treatment, the study contributes to our understanding of disease dynamics and potential treatment optimizations through the integration of computational modeling with biological processes.

INTRODUCTION

The study art of approximation for continuous problems is commonly called the study of “Numerical Analysis”, according to the New Jersey Institute of Technology,

“Numerical analysis is the area of mathematics and computer science that creates, analyzes, and implements algorithms for solving numerically the problems of continuous mathematics. Such problems originate generally from real-world applications of algebra, geometry and calculus, and they involve variables which vary continuously; these problems occur throughout the natural sciences, social sciences, engineering, medicine, and business.”

At the AP Calculus or general High School Math level, besides the use of a graphing calculator, technology is not usually used in the classroom to build an intuition and strong foundation for students. If it is, it's mostly in the form of already-created tools.

This project, however, utilized my own computer science skills and mathematics skills to accurately, efficiently, and thoroughly analyze and simulate data for the project. In this project, python was used for the primary reason of personal choice. A language like Java would have been a possible alternative and computationally better choice. However, for the purposes of this project, the differences in time between the languages would not be majorly impactful for a relatively minor level of computation.

Additionally, I chose this project to experience a crossroads between biology, math, and computer science. By using computer science, the simulation could be tweaked however it is needed, allowing for scalability, alterability, and most importantly, ease of use.

PROBLEM FRAMING

“When viral particles of a certain virus enter the human body, they replicate rapidly. In about four hours, the number of viral particles has doubled. The immune system does not respond until there are about 1 million viral particles in the body.

The first response of the immune system is fever. The rise in temperature lowers the rate at which the viral particles replicate to 160 % every four hours, but the immune system can only eliminate these particular viral particles at the rate of about 50 000 viral particles per hour. Often people do not seek medical attention immediately as they think they have a common cold. If the number of viral particles, however, reaches 10^{12} , the person dies.” (Credit to the International Baccalaureate Organization)

SOLUTION PHASES

Modeling Infection:

Modelling infection

1. Model the initial phase of the illness for a person infected with 10 000 viral particles to determine how long it will take for the body's immune response to begin.
2. Using a spreadsheet, or otherwise, develop a model for the next phase of the illness, when the immune response has begun but no medications have yet been administered. Use the model to determine how long it will be before the patient dies if the infection is left untreated.

To model the viral growth, first it has to be understood that the explicit form for the equation with respect to time (hours) would follow the usual elementary calculus “population growth model”

$$P(t) = P_0 r^t$$

where P_0 is the initial population, r is the rate, and t is the time units.

For the purposes of the current project however, this equation has to be modified. It is known that the initial population of the virus is 10 000, and that the rate of growth is 2. However, the time interval is 4 hours, so to make the time unit per hour and still inline with the equation, the t is divided by 4 to give the following:

$$V(t) = V_0 r^{t/4}$$

$$V(t) = 10000 * 2^{t/4}$$

To model this simulation, using the explicit equation would have been more accurate. However, for purposes discussed later, using the differential equation and Euler’s Method proves to be more effective, giving the simulation more parameters to change. Additionally, the error is negligible, proven to be ~0.1% error through testing, a negligible effect for the purposes of the project, and it is very simple to increase the accuracy by reducing the step size for Euler’s method, which using Leibinez notation is represented by

$$f(x = x_0 + dt) = f(x_0) + \frac{df}{dx} * dx$$

In the context of this problem:

$$V(t = t_0 + dt) = V(t_0) + \frac{dV}{dt} * dt$$

where

$$\frac{dV}{dt} = \frac{10000 * 2^{t/4} * \ln 2}{4} = \frac{V * \ln 2}{4}$$

and dt is sufficiently small. In this simulation, dt was always 2^{-n} ($n \in \mathbb{N}$) to eliminate floating-point error solely for timekeeping, because that would guarantee proper addition for the internal clock of the simulation (being binary).

Through all this, the simulation [1] had now a solid calculation method as shown in the following code

```
t = 0
viralIncreaseRate = 2 # doubles every 4 hours
viralIncreaseTime = 4 # 4 hours
```

as some of the initial values, and

```
increment = (virusCount * (math.log(viralIncreaseRate)) / viralIncreaseTime) * dt
virusCount = virusCount + increment
```

as the code for the virus count progression, with of course $t = t + dt$ as well to progress through time.

There are multiple “scenarios” in the simulation. For this part, only scenario’s 1 and 2 are needed.

```
Scenario 1: Body does not fight back.  
Scenario 2: Body has an immune response
```

To check when the immune response begins, the program goes through a conditional each time tick, and outputs the phrase “Immune response occurring at t hours with V amount of virus particles”, and changes the rate of viral increase from 2 to 1.6, which was stated in the initial problem statement. The code for this is simple:

```
if (scenario >= 2 and not immuneResponseOccurring and virusCount > (10**6)):  
    print("Immune response occurring at ", t, " hours with ", virusCount, "  
amount of virus particles")  
    immuneResponseOccurring = True  
    viralIncreaseRate = 1.6
```

The output of the simulation and answers to the previous questions are as follows:

```
scenario 1  
Patient dies at t = 106.3115234375 and 1000143857709.6289 amount of virus particles  
  
scenario 2  
Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles  
Patient dies at t = 148.8798828125 and 1000064169553.3191 amount of virus particles
```

The error is acceptable, about $\sim 0.15\%$ for $dt = 2^{-10}$.

Modelling Recovery I:

Modelling recovery

An antiviral medication can be administered as soon as a person seeks medical attention. The medication does not affect the growth rate of the viruses but together with the immune response can eliminate 1.2 million viral particles per hour.

3. If the person is to make a full recovery, explain why effective medication must be administered before the number of viral particles reaches 9 to 10 million.

The question specifically asks why the effective medication must be administered in between 9 and 10 million, not the exact point. Therefore, the Intermediate Value Theorem (IVT) can be used in the following logical statement.

“ a is a survival value, b is a death value, so there must be some maximum value t ($a < x < b$) where x is the most amount of viral particles before the viral medication must be administered, or the patient will pass.”

The simulation takes into account the 1.2 million viral particles per hour medication+immune viral elimination as follows:

```
viralIncreaseTime = 4 # 4 hours
virusMedicationKillRate = 1.2*(10**6)
...
virusMedicationKillRateDecrement = virusMedicationKillRate * dt
...
if ((scenario == 3 or scenario == 4) and immuneResponseOccurring and
virusCount > scenariothreshold) or medicationBeingInserted:
    virusCount = virusCount - virusMedicationKillRateDecrement

    medicationBeingInserted = True # regardless of whether above or below
threshold (when the medication is inserted), medication will continue being
inserted
```

Following the scenario structure laid out:

```
scenario 3/4: Body gets the help of medicine
- 3: At 10.2 million, body is saved
- 4: At 10.3 million, it isn't
- Using IVT, the effective medication must be administered between the two
```

The following results are given:

```
scenario 3
Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles
Patient is clean at t = 107.1064453125

scenario 4
Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles
Patient dies at t = 189.1708984375 and 1000051545639.2037 amount of virus particles
```

Therefore, the latest viral count for the medicine injection to save the patient is between 10.2 million and 10.3 million virus particles, slightly off from the question's claim, but close enough to say around 10 million, with .3 million possibly as a margin of error.

Modelling Recovery II:

The antiviral medication is difficult for the body to adapt to, so it must *initially* be carefully introduced to the body over a four-hour time period of **continuous** intravenous dosing. This means the same amount of medication is entering the body at any given time during the first 4 hours. At the same time, however, the kidneys eliminate about 2.5 % of this medication per hour. The doctor has calculated that the patient needs at least 90 micrograms of medication to begin and maintain the rate of elimination of 1.2 million viral particles.

4. Create a mathematical model for this four-hour period so that by the end of the four hour period the patient has 90 micrograms of medication in their body. Find the solution to your model analytically, or estimate its solution with the help of technology.

To solve this question, another program was written in python [2]. The program simulates the conditions outline in the current question, and uses an “oscillating” approach.

Essentially, the program will start with a rate of 10 micrograms per hour. If the simulated medicine volume exceeds 90 micrograms before 4 hours, the rate is reduced by 2%, as there is still leeway that can be addressed to find a more exact answer. However, if the volume does not reach 90 micrograms within 4 hours, then the rate increases by 2% to reach such a goal.

Consequently, the output looks like the following:

```
too much time: amount = 85.7509804042117 where r = 22.522004568871505
too much time: amount = 87.4660001229133 where r = 22.972444660248936
too much time: amount = 89.21532001253854 where r = 23.431893553453914
90.00646606065608 at t = 3.9541015625 where r = 23.900531424522992
too much time: amount = 89.1589370609253 where r = 23.422520796032533
90.01273192235864 at t = 3.9560546875 where r = 23.890971211953183
too much time: amount = 89.12327348610678 where r = 23.413151787714117
90.01897630730039 at t = 3.9580078125 where r = 23.8814148234684
too much time: amount = 89.08762417670728 where r = 23.40378652699903
90.00408423244834 at t = 3.958984375 where r = 23.87186225753901
too much time: amount = 89.05198912704527 where r = 23.39442501238823
90.0102951917528 at t = 3.9609375 where r = 23.862313512635993
too much time: amount = 89.01636833139962 where r = 23.385067242383272
```

The value that is closest to 90 is the most accurate, which is shown to be the 4th entry at approximately 90.0 medicine volume, at a rate of approximately 23.9 micrograms per hour. Therefore, the most optimal rate to have the patient at approximately 90 micrograms of medication even with kidney filtration is about 23.9 micrograms per hour.

Modelling Recovery III:

5. What dosage, D , administered every four hours from the end of the first continual intravenous phase, would allow for the patient to maintain at least 90 micrograms of the medication in his system? Make sure you take into account the kidneys’ rate of elimination. Explain carefully how you came to this number.

Interestingly enough, it seems that the question, while not totally invalid, isn’t realistic or even that accurate.

Only one hour after the first intravenous injection, the kidneys actually reduce the amount of dosage in the body from 90 to ~87. The question is asking how much dosage D is needed *after 4 hours of the injection* to maintain the 90+ range. But then wouldn't that make the 3 hours before the dosage be < 90 , so there's no medication response anymore and this all breaks down?

Therefore, I chose to alter the question in the following way:

What dosage administered, D , would allow for the patient to maintain at least 90 micrograms of the medication in his system after N hours to be injected into again? Make sure you take into account the kidneys' rate of elimination. Explain carefully how you came to this number.

[Another program created in python [3] takes $N = 1$, but it can be altered to be any $N \in \mathbb{R}$. However, to reduce complication, the program was left with $N = 1$]

Essentially, after an N amount of hours, there needs to still be above 90 micrograms so that a little more can be injected into the patient, allowing for a permanent spot above 90 micrograms.

Using similar logic to the previous question, the program has an added condition where it simulates whether the patient can still have above 90 micrograms for the first hour. The following is a part of the output:

```
too much time: amount = 89.21532001253854 where r = 23.431893553453914
not possible to maintain, bumping it up... amount = 90.99962641279593 and r = 23.900531424522992
93.74781513045109 at t = 4.0009765625 where r = 24.622327473543585
93.71031600439936 at t = 4.0009765625 where r = 24.612478542554168
93.67283187800433 at t = 4.0009765625 where r = 24.602633551137146
93.63536274524508 at t = 4.0009765625 where r = 24.59279249771669
93.59790860014813 at t = 4.0009765625 where r = 24.582955380717603
93.56046943670583 at t = 4.0009765625 where r = 24.573122198565315
93.52304524892882 at t = 4.0009765625 where r = 24.563292949685888
93.48563603083451 at t = 4.0009765625 where r = 24.553467632506013
93.44824177642461 at t = 4.0009765625 where r = 24.543646245453008
93.41086247971018 at t = 4.0009765625 where r = 24.533828786954828
93.37349813473165 at t = 4.0009765625 where r = 24.524015255440045
93.33614873546287 at t = 4.0009765625 where r = 24.51420564933787
93.2988142759749 at t = 4.0009765625 where r = 24.504399967078136
93.26149475027071 at t = 4.0009765625 where r = 24.494598207091308
93.22419015235717 at t = 4.0009765625 where r = 24.48480036780847
93.18690047631242 at t = 4.0009765625 where r = 24.47500644766135
93.14962571611163 at t = 4.0009765625 where r = 24.465216445082284
93.11236586582248 at t = 4.0009765625 where r = 24.455430358504252
93.07512091947801 at t = 4.0009765625 where r = 24.445648186360852
93.03789087110924 at t = 4.0009765625 where r = 24.43586992708631
93.00067571476946 at t = 4.0009765625 where r = 24.426095579115472
92.96347544448425 at t = 4.0009765625 where r = 24.416325140883824
92.92629005430648 at t = 4.0009765625 where r = 24.40655861082747
92.88911953828249 at t = 4.0009765625 where r = 24.39679598738314
92.85196389045667 at t = 4.0009765625 where r = 24.387037268988188
92.81482310490325 at t = 4.0009765625 where r = 24.377282454080593
92.77769717566352 at t = 4.0009765625 where r = 24.367531541098963
92.7405860968013 at t = 4.0009765625 where r = 24.357784528482522
92.70348986236051 at t = 4.0009765625 where r = 24.34804141467113
92.66640846640561 at t = 4.0009765625 where r = 24.33830219810526
92.62934190303444 at t = 4.0009765625 where r = 24.328566877226017
92.59229016627148 at t = 4.0009765625 where r = 24.318835450475124
92.55525325019754 at t = 4.0009765625 where r = 24.309107916294934
92.51823114889172 at t = 4.0009765625 where r = 24.299384273128418
92.48122385643013 at t = 4.0009765625 where r = 24.289664519419166
92.44423136688906 at t = 4.0009765625 where r = 24.2799486536114
92.40725367434095 at t = 4.0009765625 where r = 24.270236674149952
92.37029077288261 at t = 4.0009765625 where r = 24.260528579480294
92.33334265657248 at t = 4.0009765625 where r = 24.2508243680485
not possible to maintain, bumping it up... amount = 92.29640931951384 and r = 24.241124038301283
95.08376088096287 at t = 4.0009765625 where r = 24.973205984257984
```

The rate ~24.25 micrograms per hour can *just* maintain this new condition. It is found by noticing the value just above the second “bump”.

Modelling Recovery IV:

6. Determine the last possible time from the onset of infection to start the regimen of medication. How long it will take to clear the viral particles from the patient's system? Show on a graph the entire treatment regimen from the time treatment begins until the viral particles are eliminated.

The following is a full simulation of the program:

(Rate for scenario 5 is the answer found in the above Question 5, approximately 24.2508243680485.

Additionally, when running the main simulation program, it seems the idea of “continuous dosing” and kidney distillation is implemented into the picture, the program becomes very different than expected.

For example, if the medication is started to be inserted at around 0.5 million, the patient is definitely going to survive.

At 10 million the patient dies.

However, between these values, things are unexpected. There almost seems to be an equilibrium, and no matter how long the simulation runs, even for 10000 simulation hours, the viral count just stays stagnant. Therefore, the program actually prompts for many values before doing this as the simulation time might take a long while if allowed to run forever)

```
23 Scenario 1: Body does not fight back.
24 Scenario 2: Body has an immune response
25 Scenario 3/4: Body gets the help of medicine
26 - 3: At 10.2 million, body is saved
27 - 4: At 10.3 million, it isn't
28 - Using IVI, the effective medication must be administered between the two
29
30 Scenario 5: Body gets help of slow-acting medicine.
31 - Solution on slideshow and implemented in program 2
32
33 """
34
35 if input("\n\nYou sure you wan...
36
37 main x
38
39 "C:\Users\Rohan Saketh Pilli\PycharmProjects\LearningPython\venv\Scripts\python.exe" "C:\Users\Rohan Saketh Pilli\PycharmProjects\MathProject2023\code\main.py"
40
41 Scenario 1
42 Patient dies at t = 106.3115234375 and 1000143857709.6289 amount of virus particles
43
44
45 Scenario 2
46 Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles
47 Patient dies at t = 148.8798828125 and 1000064169553.3191 amount of virus particles
48
49
50 Scenario 3
51 Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles
52 Patient is clean at t = 107.1064453125
53
54
55 Scenario 4
56 Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles
57 Patient dies at t = 189.1708984375 and 1000051545639.2037 amount of virus particles
58
59
60 You sure you want to see scenario 5? It is a very sensitive one in the rest of the data... y for yes: y
61 Scenario 5
62 ok, how many hours do you want this thing to simulate for: 10000
63 also, how at how many millions do you want to start seeing medication take effect: 0.5
64 Immune response occurring at 26.578125 hours with 1000078.2676707519 amount of virus particles
65 simulation exceeded max given time, here is the current amount of viral particles:
66 999320.1313122704
```

Additionally, the program displays the following graph:

Red: Scenario 1

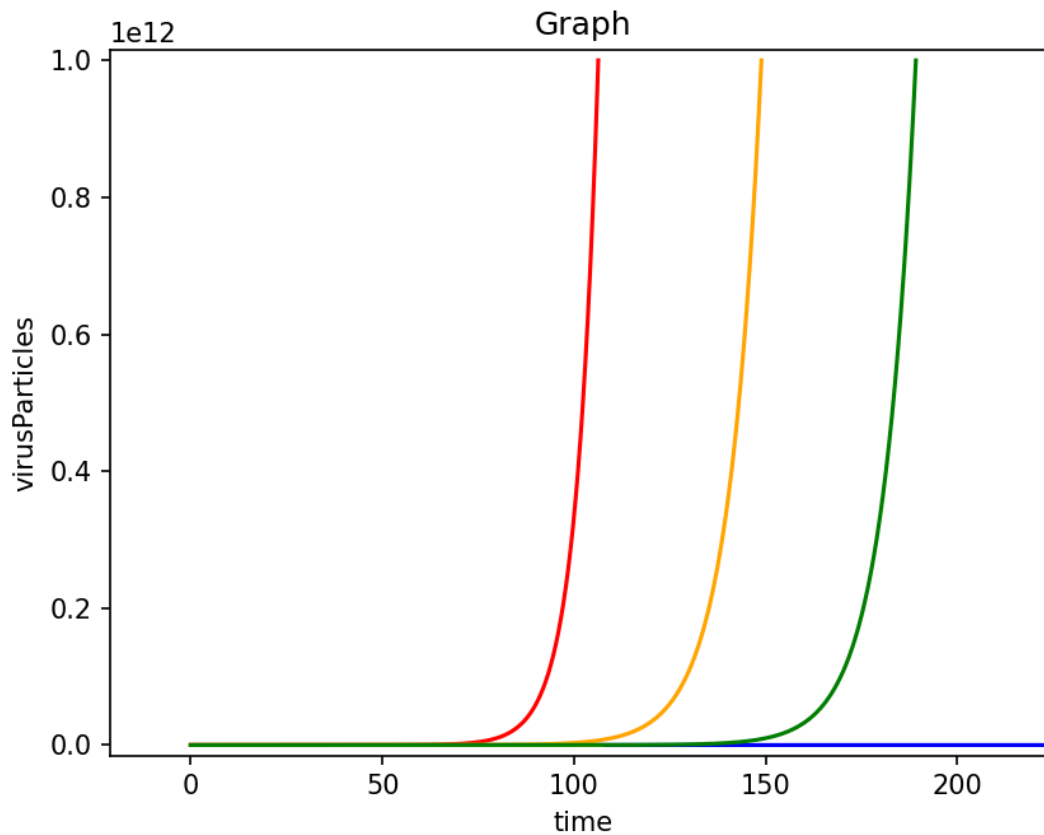
Orange/Gold: Scenario 2

Yellow: Scenario 3

- Patient is cured, so graph cannot be seen at this scale

Green: Scenario 4

Blue: Scenario 5



Modified Modeling:

Analyzing your models

Analyze all your models discussing any assumptions you have made, the strengths and weaknesses of the models, and the reliability of your results.

Applying your model

7. Explain how your models could be modified for use if the patient were not an adult, but a young child.

All the different parameters that can be tweaked by the program are the following:

```
def simulate(tMax, scenario=1, scenarioThreshold=0, minimumDose=0, rateofDosage=0):
    t = 0
    viralIncreaseRate = 2 # doubles every 4 hours
    viralIncreaseTime = 4 # 4 hours
    virusMedicationKillRate = 1.2*(10**6)
    virusImmuneKillRate = 50000
    currentDosageAmount = 0
    kidneyDistillationRate = 2.5/100

    startingVirusCount = 10000

    dt = (2 ** -10) # sim updates for every 2^-10 hours [base 2 eliminates floating-point nonsense]

    virusCount = startingVirusCount
```

Changing this for a child is quite simple. The virus replication rate can be increased, or immune response can be decreased, for example. There are many dials and knobs that can be turned to make this simulation more controllable.

ACKNOWLEDGEMENTS

My heartfelt thanks to Dr. Yu (Molecular Biology, Imperial College London) for his guidance and sincerity in mentoring me throughout this process.

PROGRAM CODE

Program 1 Code:

```
import matplotlib.pyplot as plt
import math
import decimal

'''
Exponential Equation:

Given
viralIncreaseRate = 2 #doubles every 4 hours
viralIncreaseTime = 4 #4 hours
virusMedicationKillRate = 1200000

startingVirusCount = 10000

Equation is  $10000 \cdot (2)^{(t/4)} = 10000 \cdot (2^{(1/4)})^t$ 
'''

x_axis = []
y_axis = []

'''
scenario 1: Body does not fight back.
scenario 2: Body has an immune response
scenario 3/4: Body gets the help of medicine
    - 3: At 10.2 million, body is saved
    - 4: At 10.3 million, it isn't
    - Using IVT, the effective medication must be administered between the two

scenario 5: Body gets help of slow-acting medicine.
    - Solution on slideshow and implemented in program 2
'''

def simulate(tMax, scenario=1, scenariothreshold=0, minimumDose=0,
rateofDosage=0):
    t = 0
    viralIncreaseRate = 2 # doubles every 4 hours
    viralIncreaseTime = 4 # 4 hours
    virusMedicationKillRate = 1.2*(10**6)
    virusImmuneKillRate = 50000
    currentDosageAmount = 0
    kidneyDistillationRate = 2.5/100
```

```

startingVirusCount = 10000

dt = (2 ** -10) # sim updates for every 2^-10 hours [base 2 eliminates
floating-point nonsense]

virusCount = startingVirusCount


# statuses
immuneResponseOccurring = False
medicationBeingInserted = False

global x_axis
global y_axis

x_axis.append([])
y_axis.append([])

virusMedicationKillRateDecrement = virusMedicationKillRate * dt

while True:
    if (virusCount > 10 ** 12):
        print("Patient dies at t = ", t, " and ", virusCount, " amount of
virus particles")
        break
    if(virusCount <= 0):
        print("Patient is clean at t = ", t)
        break

    if (scenario >= 2 and not immuneResponseOccurring and virusCount >
(10**6)):
        print("Immune response occurring at ", t, " hours with ",
virusCount, " amount of virus particles")
        immuneResponseOccurring = True
        viralIncreaseRate = 1.6

        increment = (virusCount * (math.log(viralIncreaseRate)) /
viralIncreaseTime)*dt
        virusCount = virusCount + increment

    if ((scenario == 3 or scenario == 4) and immuneResponseOccurring and
virusCount > scenariothreshold) or medicationBeingInserted:

```

```

        virusCount = virusCount - virusMedicationKillRateDecrement

        medicationBeingInserted = True # regardless of whether above or
below threshold (when the medication is inserted), medication will continue
being inserted

        if((scenario == 5 or scenario == 6) and immuneResponseOccurring and
virusCount > scenariothreshold) or medicationBeingInserted:
            currentDosageAmount = currentDosageAmount + rateofDosage*dt -
kidneyDistillationRate*currentDosageAmount*dt
            medicationBeingInserted = True

            if(currentDosageAmount > minimumDose):
                virusCount = virusCount - virusMedicationKillRateDecrement

        if(immuneResponseOccurring and not medicationBeingInserted):
            virusCount = virusCount - virusImmuneKillRate*dt

        y_axis[scenario - 1].append(virusCount)
        x_axis[scenario - 1].append(t)

        t = t + dt
        if (t >= (tMax)):
            print("simulation exceeded max given time, here is the current
amount of viral particles: ")
            print(virusCount)
            break

print("scenario 1")
simulate(500, 1)
print("\n\n\n")
print("scenario 2")
simulate(500, 2)
print("\n\n\n")
print("scenario 3")
simulate(500, 3, 10.2 * (10 ** 6))
print("\n\n\n")
print("scenario 4")
simulate(500, 4, 10.3 * (10 ** 6))
if(input("\n\nYou sure you want to see scenario 5? It is a very sensitive one
in the rest of the data... y for yes: ") == 'y'):
    print("scenario 5")
    simulate(int(input("ok, how many hours do you want this thing to simulate
for: ")), 5, float(input("also, how at how many millions do you want to start
seeing medication take effect: ")) * (10**6), 90, 24.2508243680485)
    plt.plot(x_axis[4], y_axis[4], color="blue") # scenario 5

```

```
print("\n\n\n")

# print("scenario 6")
# simulate(500, 6, 10.2 * (10**6), 90, 24.2508243680485)

plt.plot(x_axis[0], y_axis[0], color="red") # scenario 1
plt.plot(x_axis[1], y_axis[1], color="orange") # scenario 2
plt.plot(x_axis[2], y_axis[2], color="yellow") # scenario 3
plt.plot(x_axis[3], y_axis[3], color="green") # scenario 4
# plt.plot(x_axis[5], y_axis[5], color="black") # scenario 6

plt.title('Graph')
plt.xlabel('time')
plt.ylabel('virusParticles')
plt.show()
```

Program 2 Code:

```
#find ideal rate

import time

target = 90
timeGiven = 4
kidneyRate = 2.5/100
rate = 10 # per hour
dt = 2 ** -10
t = 0
amount=0
while True:
    if(t > timeGiven):
        print("too much time: amount = ", amount, " where r = ", rate)
        rate = rate*1.02
        time.sleep(0.25)
        target = 90
        timeGiven = 4
        kidneyRate = 2.5 / 100
        dt = 2 ** -10
        t = 0
        amount = 0

    amount = amount + rate*dt - kidneyRate*dt*amount

    if(amount >= target):
        print(amount, " at t = ", t, " where r = ", rate)
        rate = rate*0.98
        time.sleep(0.25)
        target = 90
        timeGiven = 4
        kidneyRate = 2.5 / 100
        dt = 2 ** -10
        t = 0
        amount = 0

    t = t+dt
```

Program 3 Code:

```
#find ideal sustainable rate

import time

target = 90
timeGiven = 4
kidneyRate = 2.5/100
rate = 10 # per hour
dt = 2 ** -10
t = 0
amount=0

while True:
    if(t > timeGiven):
        if (amount >= target):
            if (amount * (1 - kidneyRate) < target):
                print("not possible to maintain, bumping it up... amount = ",
amount, " and r = ", rate)
                rate = rate * 1.01
            else:
                print(amount, " at t = ", t, " where r = ", rate)
                rate = rate * 0.98
        else:
            print("too much time: amount = ", amount, " where r = ", rate)
            rate = rate*1.02
            time.sleep(0.2)
            target = 90
            timeGiven = 4
            kidneyRate = 2.5 / 100
            dt = 2 ** -10
            t = 0
            amount = 0

    amount = amount + rate*dt - kidneyRate*dt*amount
    t = t+dt
```