Propagation Malware in Wireless Systems: A variation of the SIR Model (SEIRV)

Name: Christ-Brian Amedjonekou

Date: 05/15/2019

MAT 4880-D692 (Math Modeling II) SEIRV Model Final Project

Spring 2019, Section: D692, Code: 36561

Instructor: Johann Thiel

Abstract

In this study, we set out to create an SEIRV Model that models propagation of malware in wireless systems. The goal is to run simulations of the dynamical system for arbitrary lengths of time. Malware Propagation Models have previously been used to characterize the behavior of the network compartments with passage of time; This is what we'll do here through simulation. We will model this problem using a discrete time dynamical system as it will serve as an approximation to the continuous dynamical system. Python and Euler's Method are used to create and run the simulations for this problem. Ultimately, we seek to demonstrate the effect that the timestep, Δt, has on the simulation; Particularly, the effect of a 10%, 50%, and 100% increase in the timestep. We expect that a large increase in the timestep will produce a delay that will cause the simulation to fail to mimic its original characteristics (chaos).

Introduction

Malware, or malicious software, is commonly used to cause damage to computers, servers, and computer networks; You can think of it as a disease that afflicts computers. This resemblance is why researchers, scientists, and the like create malware analysis models using network compartments (w/ the passage of time) and dynamical systems. This particular model, the SEIRV Model, is a 5-compartment model. We will run simulations given a set of default parameters and initial values, then determine how different time-steps (Δt) affect the simulations overall. We will determine how much larger the time-step must become before our system stops mimicking the original system characteristics (and we introduce chaos).

Assumptions and Definitions

For the model of the continuous dynamical system, we have the following state variables:

- S(t) = Susceptibles
- E(t) = Exposed (latent)
- I(t) = Infectious
- R(t) = Recovered (temporarily immune)
- V(t) = Vaccinated (immunized)

For the state variables, we assume that $S, E, I, R, V \geq 0$. This is verified for our simulation as we have initial values for S, E, I, R, V respectively that are all equal to or larger than 0. The number of Infected, I, is given to be 1 node with respect to the total population. Number of Susceptibles, S, is equal to 100 nodes. As opposed to previous SIR Models we now have an in between case, E, the number of exposed which is given to be 3 nodes. The recovered nodes, R, and the vaccinated nodes, V, are initialized to 0. This makes sense because you cannot be recovered or vaccinated prior to malware spreading. We also have the parameters that influences the nature of the dynamical system, shown here:

- $\sigma = Distribution\ Density$
- $r = transmission\ range$
    - $\sigma\ \pi\ r_0^2 = effective\ contact\ with\ an\ infected\ node\ for\ the\ transfer\ of\ infection$
- $\lambda = inclusion\ rate\ of\ new\ nodes\ into\ the\ network$
- $\beta = infection\ contact\ rate$

- $\tau = Death\ Rate\ of\ the\ nodes\ due\ to\ hardware\ or\ software\ failure$
- $\omega = Crashing\ rate\ due\ to\ attack\ of\ malicious\ objects\ (a\ worm\ in\ this\ case)$
- $\theta = Rate\ at\ which\ exposed\ nodes\ become\ infectious$
- $\nu = Rate\ of\ recovery$
- $\phi = Rate\ at\ which\ recovered\ nodes\ become\ susceptible\ to\ infection$
- $\rho = Rate\ of\ vaccination\ for\ susceptible\ sensor\ nodes$
- $\xi = Rate\ of\ transmission\ from\ the\ "V"\ compartment\ to\ the\ "S"\ Compartment$

These parameters are also initialized at the beginning of the system. Sigma, the distribution density (σ), has a value of 0.5. The Initial Transmission Range, r_0, has a range of 2 meters. Lambda, λ, the inclusion rate of new nodes has a value of 0.33. Beta, β, infection contact rate, has a value of 0.1. Tau, τ, the death rate of the nodes has a value of 0.003. Omega, ω, the crashing rate is set to 0.07. Theta, θ, the rate at which the exposed become infected is set to 0.25. The rate of recovery (ν) is set to 0.4, Rate of infection for recovered node (ϕ) is set to 0.3, Rate of vaccination (ρ) is set to is set to 0.3, Rate of transmission from Vaccinated to the Susceptibles (ξ) is set to 0.06. Finally, we have the following assumption for the dynamical system:

$$\frac{\partial \mathrm{S}}{\partial \mathrm{t}} = \lambda - \beta\, SI\, \sigma\, \pi\, r_0^2 - \tau\, S - \rho\, S + \phi\, R + \xi\, V$$
$$\frac{\partial \mathrm{E}}{\partial \mathrm{t}} = \beta\, SI\, \sigma\, \pi\, r_0^2 - (\tau + \theta)E$$
$$\frac{\partial I}{\partial t} = \theta\, E - (\tau + \omega + \nu)I$$
$$\frac{\partial R}{\partial t} = \nu\, I - (\tau + \phi)R$$
$$\frac{\partial V}{\partial t} = \rho\, S - (\tau + \xi)V$$

Recall that a differential equation $\left(\mathrm{e.g.}: \frac{\partial S}{\partial t}, \frac{\partial \mathrm{E}}{\partial \mathrm{t}}, \frac{\partial I}{\partial t}, \frac{\partial R}{\partial t}, \frac{\partial V}{\partial t}\right)$ simply relates the rate of change (derivate) to the state variables. The dynamical system in question is continuous but will be modeled with a discrete time dynamical system. We will use Euler's Method to model the continuous system by way of a discrete one and Python to simulate this system.

Analysis

To model this continuous system (system of differential equations) as a discrete dynamical system (system of difference equations) we had to assume that the time-step, Δt, was relatively small. The reason for this is due to the nature of continuous systems: there is barely any time delay in continuous systems, change happens so quickly. With a large enough time-step, approximations via discrete methods fail to mimic the original continuous system. We avoid this by assuming a small time-step, $\Delta t = 0.1$. To simulate the system, the code segment in Figure 1 was used:

```
# Initial Conditions
L = [[0, S, E, I, R, V]]

# Euler's Implementation
for i in range(N):
    t_last, S_last, E_last = L[-1][0], L[-1][1], L[-1][2]
    I_last, R_last, V_last = L[-1][3], L[-1][4], L[-1][5]
    t_next = t_last + delT
    S_next = S_last + (delT * deltaS(S_last, E_last, I_last, R_last, V_last))
    E_next = E_last + (delT * deltaE(S_last, E_last, I_last, R_last, V_last))
    I_next = I_last + (delT * deltaI(S_last, E_last, I_last, R_last, V_last))
    R_next = R_last + (delT * deltaR(S_last, E_last, I_last, R_last, V_last))
    V_next = V_last + (delT * deltaV(S_last, E_last, I_last, R_last, V_last))
    L.append([t_next, S_next, E_next, I_next, R_next, V_next])
```

Figure 1: Euler's Method Implementation for SEIRV in Python

The code segment takes the initial values of the five state variables, and places them into a list. We implement Euler's Method [$x_{n+1} = x_n + \Delta t * f(x_n)$] for each of the state variables, and place them in a loop that iterates for as long as we decide the loop to run. Each time the loop runs, it multiplies the change by the time-step [$\Delta t * f(x_n)$] and adds the result to the current element in the sequence to get the next element in the sequence, then appends it to the list. Once this is complete we are able to plot the results via Python's package, Matplotlib.

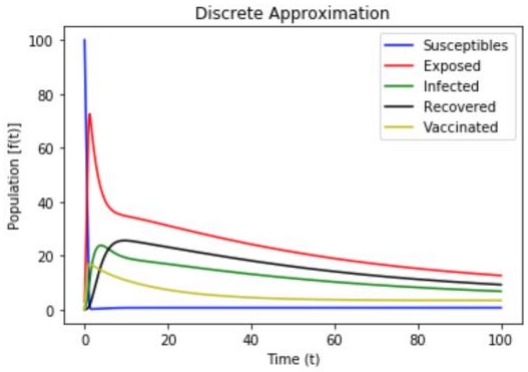


Figure 2: Simulation $\Delta t = 0.1$

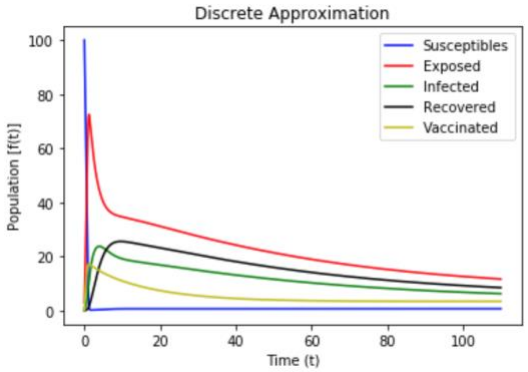


Figure 3: Simulation $\Delta t = 0.11$

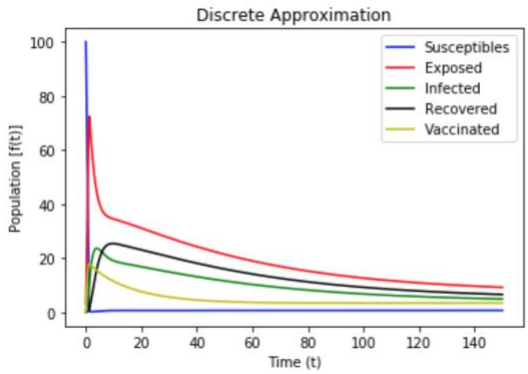


Figure 4: Simulation $\Delta t = 0.15$

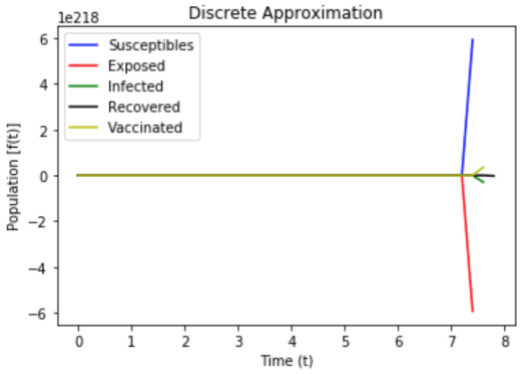


Figure 5: Simulation $\Delta t = 0.2$

The simulations, in the context of the problem, show that there will be an immediate spike in Exposed, Infected, Recovered, and Vaccinated following by a decrease working nodes in each compartment. What this seems to indicate is that virus still persists and will continue to until all nodes in the system are destroyed. Regarding the effect of the time-step on the simulations, the simulations show no significant change between the original [Figure 2], the 10% increase [Figure 3], and the 50% percent increase [Figure 4] in time. However, as we do a 100% increase (doubling our time step [Figure 5]) we start to see our model fail to resemble the original model.

Interpretations and Conclusions

Based on the result from Figures 2 - 5 of the SEIRV Compartment Simulations, we can verify that we can model continuous dynamical systems w/ discrete ones given that our time step is small enough; $\frac{\partial x}{\partial t} \approx \frac{\Delta x}{\Delta y}$ given that $\Delta\, t$ is small enough. This is verified w/ the simulations, as it isn't until $\Delta\, t$ is twice as large that we begin to see the model failing to mimic the original system.

Appendix

MAT 4880-D692 (Math Modeling II) Final Project Sim

May 16, 2019

1 Propagation Malware in Wireless Systems: A variation of the SIR Model (SEIRV)

1.0.1 Abstract:

In this study, we set out to create an SEIRV Model that models propagation of malware in wireless systems. The goal is to run simulations of the dynamical system for arbitrary lengths of time. Malware Propagation Models have previously been used to characterize the behavior of the network compartments with passage of time; This is what we'll do here through simulation. We will model this problem using a discrete time dynamical system as it will serve as an approximation to the continuous dynamical system. Python and Euler's Method are used to create and run the simulations for this problem. Ultimately, we seek to demonstrate the effect that the timestep, , has on the simulation; Particularly, the effect of a 10%, 50%, and 100% increase in the timestep. We expect that a large increase in the timestep will produce a delay that will cause the simulation to fail to mimic its original characteristics (chaos).

1.0.2 Introduction

Malware, or malicious software, is commonly used to cause damage to computers, servers, and computer networks; You can think of it as a disease that afflicts computers. This resemblance is why researchers, scientists, and the like create malware analysis models using network compartments (w/ the passage of time) and dynamical systems. This particular model, the SEIRV Model, is a 5-compartment model. We will run simulations given a set of default parameters and initial values, then determine how different time-steps () affect the simulations overall. We will determine how much larger the time-step must become before our system stops mimicking the original system characteristics (and we introduce chaos).

1.0.3 Assumptions/Definitions:

State Variables:

- S(t) = Susceptibles
- E(t) = Exposed (latent)
- I(t) = Infectious
- R(t) = Recovered (temporarily immune)
- V(t) = Vaccinated (immunized)
- N(t) = Total Population (sum of all the above)

Parameters:

- $\sigma =$ Distribution Density
- $r =$ transmission range
  - $\sigma \pi r_0^2 =$ effective contact with an infected node for the transfer of infection
- $\lambda =$ inclusion rate of new nodes into the network
- $\beta =$ infection contact rate
- $\tau =$ Death Rate of the nodes due to hardware or software failure
- $\omega =$ Crashing rate due to attack of malicious objects (a worm in this case)
- $\theta =$ Rate at which exposed nodes become infectious
- $\nu =$ Rate of recovery
- $\phi =$ Rate at which recovered nodes become susceptible to infection
- $\rho =$ Rate of vaccination for susceptible sensor nodes
- $\xi =$ Rate of transmission from the Vaccinated compartment to the Susceptible Compartment

Assumptions:
For our model, we'll assume the following:

- $S = 100; E = 3; I = 1; R = 0; V = 0$
- $\lambda = 0.33; \beta = 0.1; \tau = 0.003; \omega = 0.07; \theta = 0.25$
- $\nu = 0.4; \phi = 0.3; \rho = 0.3; \xi = 0.06; \sigma = 0.5; r_0 = 2$ meters
- $\frac{dS}{dt} = \lambda - \beta SI\sigma\pi r_0^2 - \tau S - \rho S + \phi R + \xi V$
- $\frac{dE}{dt} = \beta SI\sigma\pi r_0^2 - (\tau + \theta)E$
- $\frac{dI}{dt} = \theta E - (\tau + \omega + \nu)I$
- $\frac{dR}{dt} = \nu I - (\tau + \phi)R$
- $\frac{dV}{dt} = \rho S - (\tau + \xi)V$
- $S, E, I, R, V \geq 0$

Assumptions: Compartment Model Image

```
In [1]: from IPython.display import Image as i
        i(filename='/Users/Chris/Desktop/Images/Compartment.png', height= 1000, width= 400)
```

Out[1]:

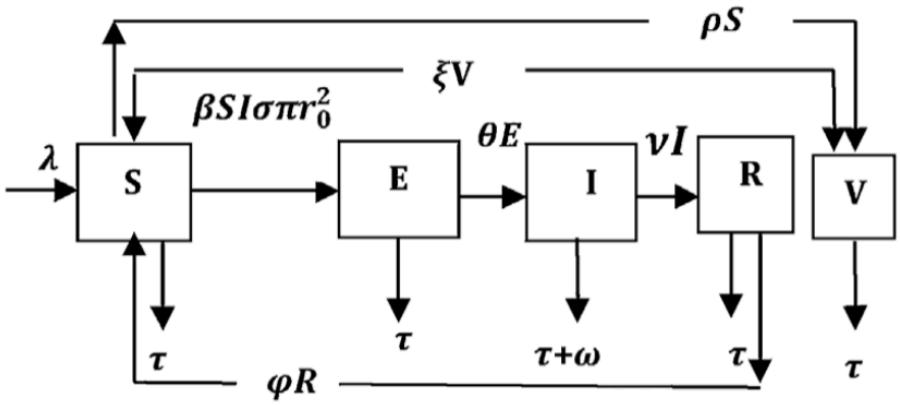


Fig. 2. Schematic diagram for the flow of worms in sensor networks [13].

1.0.4 Analysis:

Modeling Approach:

- The network system was modeled as a discrete dynamical system, a system of difference equations. This dynamical system consisted of five state variables: S, E, I, R, V.
- The state space is shown below:

$$S = \{(S, E, I, R, V) : S \geq 0, E \geq 0, I \geq 0, R \geq 0, V \geq 0\}$$

- Given that Δt is relatively small, we can simulate a continuous dynamical system of differential equations w/ a discrete dynamical system of difference equations $\left(\frac{dx}{dt} \approx \frac{\Delta x}{\Delta t}\right)$. The system of five difference equations is also shown below:

$$\frac{\Delta S}{\Delta t} = \lambda - \beta SI\sigma\pi r_0^2 - \tau S - \rho S + \phi R + \xi V$$

$$\frac{\Delta E}{\Delta t} = \beta SI\sigma\pi r_0^2 - (\tau + \theta)E$$

$$\frac{\Delta I}{\Delta t} = \theta E - (\tau + \omega + \nu)I$$

$$\frac{\Delta R}{\Delta t} = \nu I - (\tau + \phi)R$$

$$\frac{\Delta V}{\Delta t} = \rho S - (\tau + \xi)V$$

- Euler's Method was used to simulate this Discrete Dynamical System: $x_{n+1} = x_n + \Delta t * 0.5x_n(1 - x_n)$.
- Python was used to create/run the simulations.
- The simulations, in the context of the problem, show that there will be an immediate spike in Exposed, Infected, Recovered, and Vaccinated following by a decrease working nodes in each compartment. What this seems to indicate is that virus still persists and will continue to until all nodes in the system are destroyed.
- The simulations show no significant change between the original, the 10% increase, and the 50% percent increase in time. However, as we do a 100% increase (doubling our time step) we start to see our model fail to resemble the original model.

1.0.5 Code

```
# Initial Conditions
L = [[0, S, E, I, R, V]]

# Euler's Implementation
for i in range(N):
    t_last, S_last, E_last = L[-1][0], L[-1][1], L[-1][2]
    I_last, R_last, V_last = L[-1][3], L[-1][4], L[-1][5]
    t_next = t_last + delT
    S_next = S_last + (delT * deltaS(S_last, E_last, I_last, R_last, V_last))
    E_next = E_last + (delT * deltaE(S_last, E_last, I_last, R_last, V_last))
    I_next = I_last + (delT * deltaI(S_last, E_last, I_last, R_last, V_last))
    R_next = R_last + (delT * deltaR(S_last, E_last, I_last, R_last, V_last))
    V_next = V_last + (delT * deltaV(S_last, E_last, I_last, R_last, V_last))
    L.append([t_next, S_next, E_next, I_next, R_next, V_next])
```

```
In [2]: import matplotlib.pyplot as plt
        import math as m
        import numpy as np
```

```
In [3]: def plot_simulation(S, E, I, R, V, lmbda= 0.33, beta= 0.1, sigma= 0.5,
                           tau= 0.003, omega= 0.07, theta= 0.25, nu= 0.4,
                           phi= 0.3, rho= 0.3, xi= 0.06, r0= 2, delT= 0.1, N = 5):

            # Dynamical System
            deltaS = lambda S, E, I, R, V: lmbda - beta*S*I*sigma*m.pi*r0**2 - tau*S - rho*S +
            deltaE = lambda S, E, I, R, V: beta*S*I*sigma*m.pi*r0**2 - (tau + theta)*E
            deltaI = lambda S, E, I, R, V: theta*E - (tau + omega + nu)*I
            deltaR = lambda S, E, I, R, V: nu*I - (tau + phi)*R
            deltaV = lambda S, E, I, R, V: rho*S - (tau + xi)*V

            # Sets up the Figure
            fig, ax = plt.subplots()
```

```
# Initial Conditions
L = [[0, S, E, I, R, V]]

# Euler's Implementation
for i in range(N):
    t_last, S_last, E_last = L[-1][0], L[-1][1], L[-1][2]
    I_last, R_last, V_last = L[-1][3], L[-1][4], L[-1][5]
    t_next = t_last + delT
    S_next = S_last + (delT * deltaS(S_last, E_last, I_last, R_last, V_last))
    E_next = E_last + (delT * deltaE(S_last, E_last, I_last, R_last, V_last))
    I_next = I_last + (delT * deltaI(S_last, E_last, I_last, R_last, V_last))
    R_next = R_last + (delT * deltaR(S_last, E_last, I_last, R_last, V_last))
    V_next = V_last + (delT * deltaV(S_last, E_last, I_last, R_last, V_last))
    L.append([t_next, S_next, E_next, I_next, R_next, V_next])


L = np.array(L)

# Plots the Dynamical System against time.
ax.plot(L[:,0], L[:,1], c= 'b', label= 'Susceptibles')
ax.plot(L[:,0], L[:,2], c= 'r', label= 'Exposed')
ax.plot(L[:,0], L[:,3], c= 'g', label= 'Infected')
ax.plot(L[:,0], L[:,4], c= 'k', label= 'Recovered')
ax.plot(L[:,0], L[:,5], c= 'y', label= 'Vaccinated')
ax.set(xlabel= 'Time (t)', ylabel='Population [f(t)]', title= 'Discrete Approximati
ax.legend(loc= 'best')
```

Figure 1: $\Delta t = 0.1$

```
In [4]: plot_simulation(100, 3, 1, 0, 0, N= 1000)
```

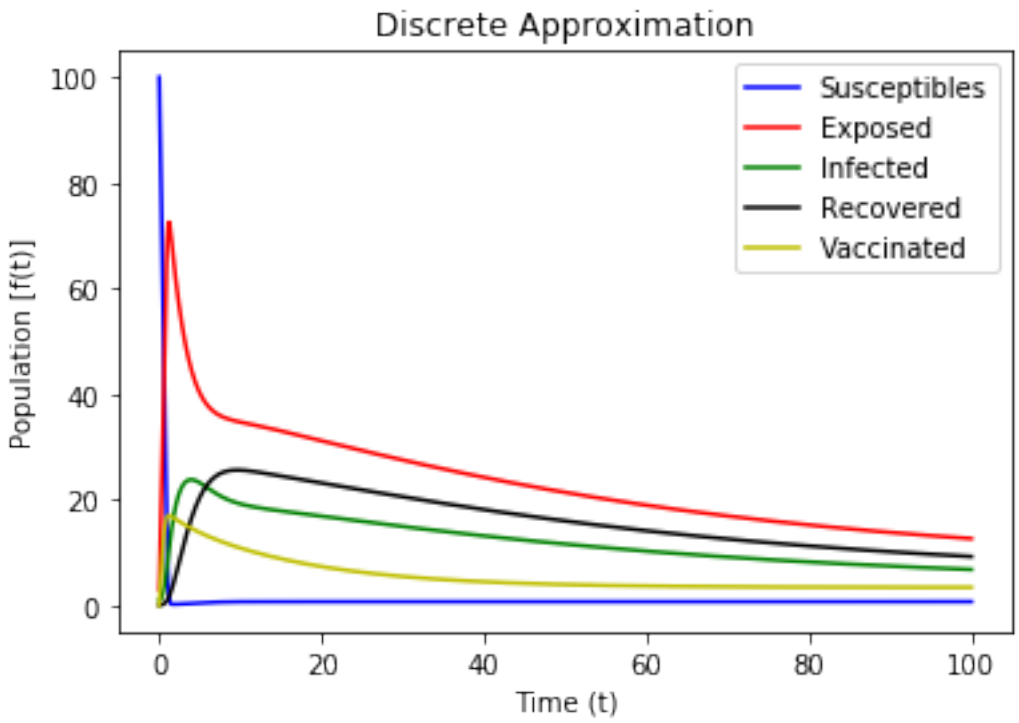


Figure 2: $\Delta t = 0.11$

```
In [5]: plot_simulation(100, 3, 1, 0, 0, delT= 0.11, N= 1000)
```

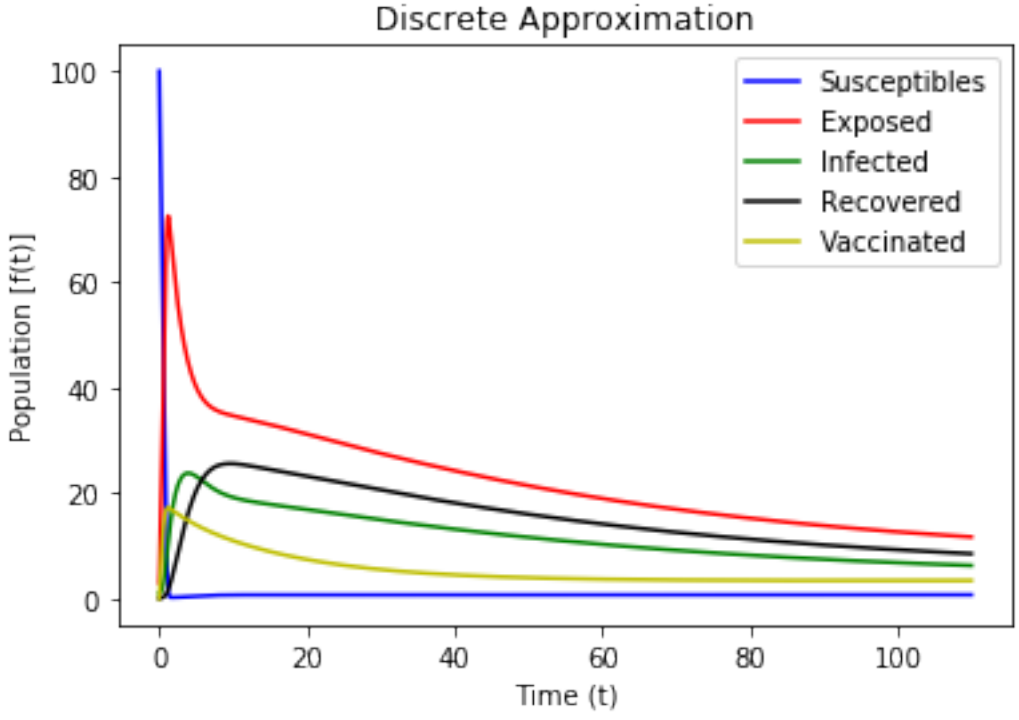


Figure 3: $\Delta t = 0.15$

```
In [6]: plot_simulation(100, 3, 1, 0, 0, delT= 0.15, N= 1000)
```

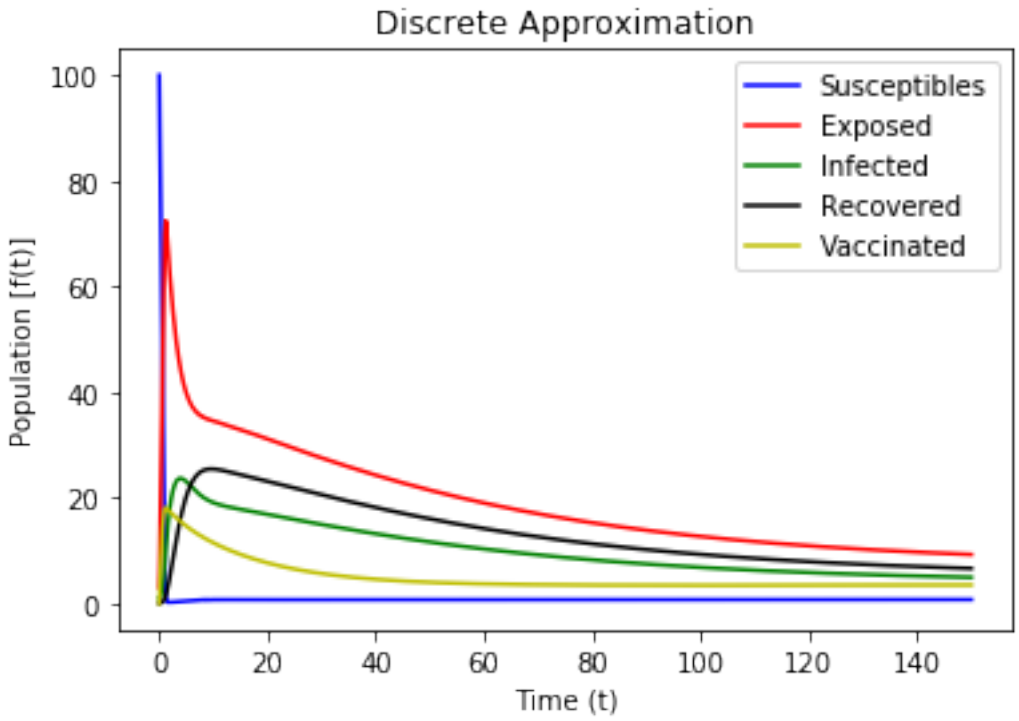
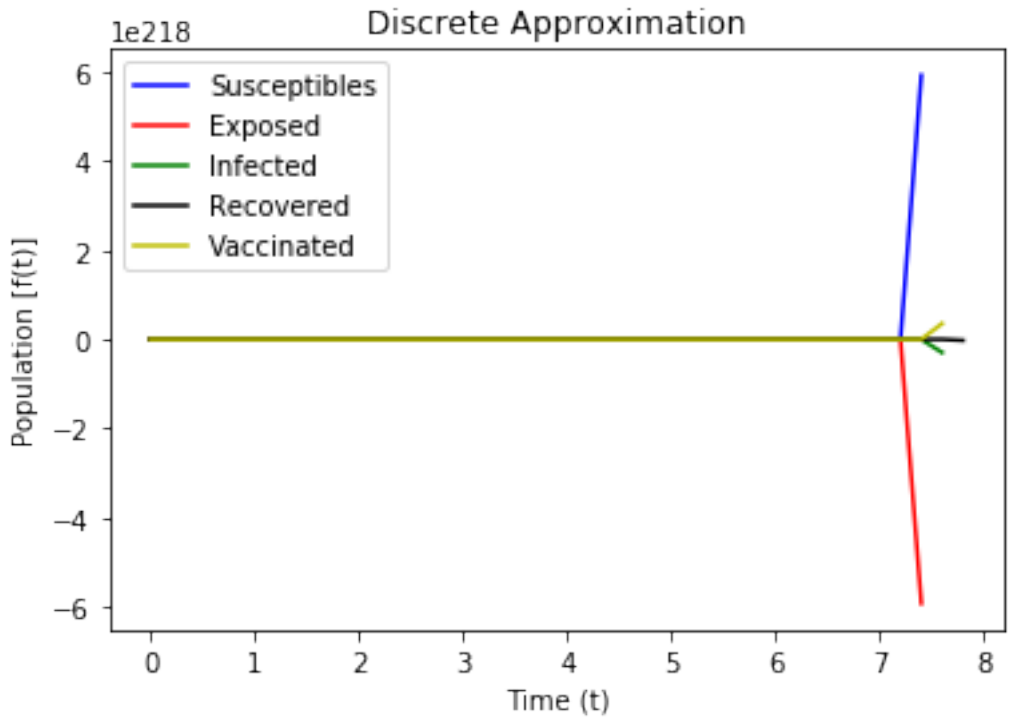


Figure 4: $\Delta t = 0.2$

```
In [7]: plot_simulation(100, 3, 1, 0, 0, delT= 0.2, N= 1000)
```



1.0.6 Interpretations and Conclusions:

- Based on the result from Figures 1 - 4 of the SEIRV Compartment Simulations, we can verify that we can model continuous dynamical systems w/ discrete ones given that our time step is small enough; $\frac{dx}{dt} \approx \frac{\Delta x}{\Delta y}$ given that Δt is small enough. This is verified w/ the simulations as it isn't until Δt is twice as large that we begin to see the model failing to look like the original system.