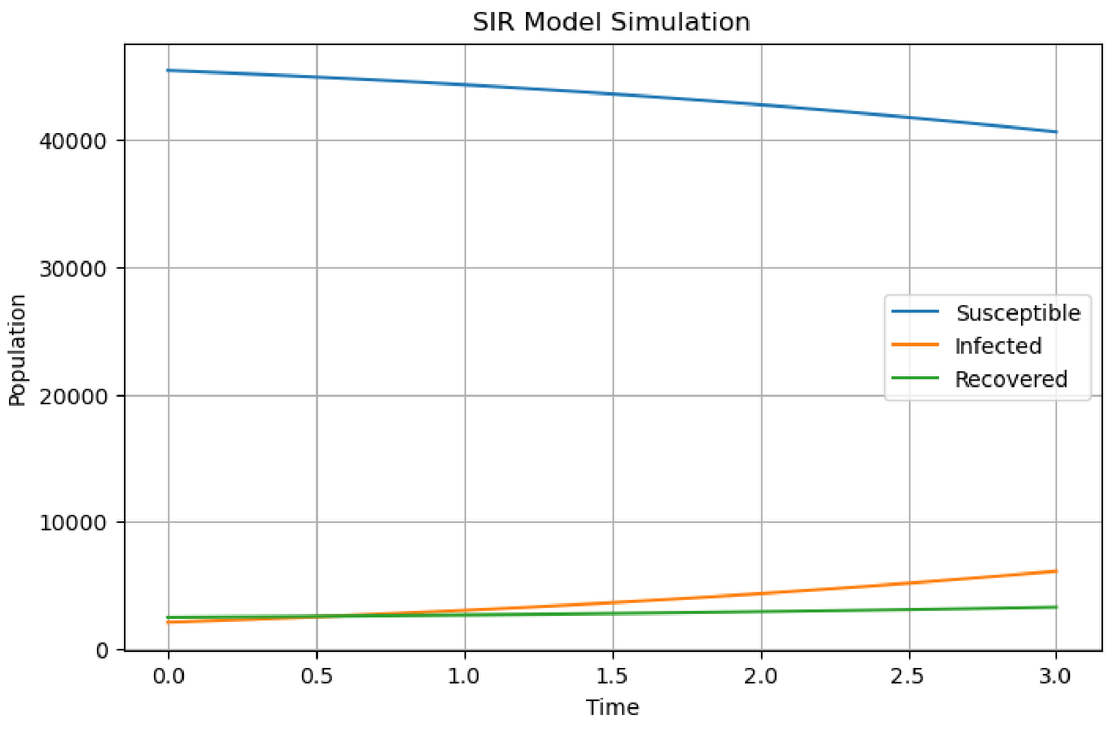
HW02 SIRplot

September 8, 2024

[3]:
```python
import matplotlib.pyplot as plt

def SIRplot(Tinit, Tfinal, S, I, R, A, L, steps):
    dT = (Tfinal - Tinit) / steps
    Tvals = [Tinit]
    Svals = [S]
    Ivals = [I]
    Rvals = [R]

    for k in range(1, steps+1):
        Sp = -A*S*I
        Ip = A*S*I - I/L
        Rp = I/L
        dS = Sp*dT
        dI = Ip*dT
        dR = Rp*dT

        Tinit = Tinit + dT
        S += dS
        I += dI
        R += dR

        Tvals.append(Tinit)
        Svals.append(S)
        Ivals.append(I)
        Rvals.append(R)

    plt.figure(figsize=(8, 5))
    plt.plot(Tvals, Svals, label="Susceptible")
    plt.plot(Tvals, Ivals, label="Infected")
    plt.plot(Tvals, Rvals, label="Recovered")

    plt.xlabel("Time")
    plt.ylabel("Population")
    plt.title("SIR Model Simulation")
    plt.legend()
    plt.grid(True)
    plt.show()
```

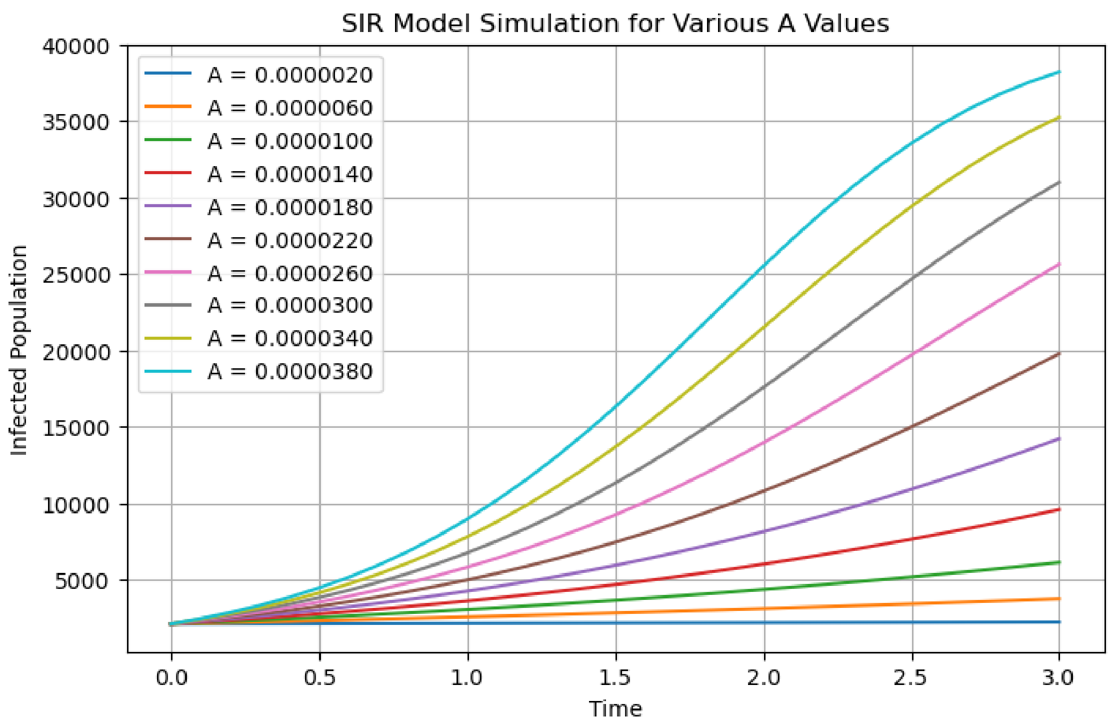
[4]:
```
SIRplot(0, 3, 45400, 2100, 2500, 30)
```



There should be at least 10 times as many timesteps as there are days we are looking at; ideally going to 100x to get the graph really smooth. Graphs with fewer timesteps will look less smooth or more "blocky" with "corners" at each timestep.

[15]:
```
# How Bad is Your Epidemic

import numpy as np

def Aplot(Tinit, Tfinal, S, I, R, steps):
    dT = (Tfinal - Tinit) / steps
    Tvals = np.linspace(Tinit, Tfinal, steps+1)
    Avals = np.arange(0.000002, 0.00004, 0.000004)

    plt.figure(figsize=(8, 5))

    for A in Avals:
        Scurr, Icurr, Rcurr = S, I, R
        Svals, Ivals, Rvals = [Scurr], [Icurr], [Rcurr]

        for k in range(steps):
            Sp = -A * Scurr * Icurr
            Ip = A * Scurr * Icurr - Icurr / 14
            Rp = Icurr / 14

            Scurr += Sp*dT
            Icurr += Ip*dT
            Rcurr += Rp*dT

            Svals.append(Scurr)
            Ivals.append(Icurr)
            Rvals.append(Rcurr)

        plt.plot(Tvals, Ivals, label=f"A = {A:.7f}")

    plt.xlabel("Time")
    plt.ylabel("Infected Population")
    plt.title("SIR Model Simulation for Various A Values")
    plt.legend()
    plt.grid(True)
    plt.show()
```

[16]:
```
Aplot(0, 3, 45400, 2100, 2500, 30)
```

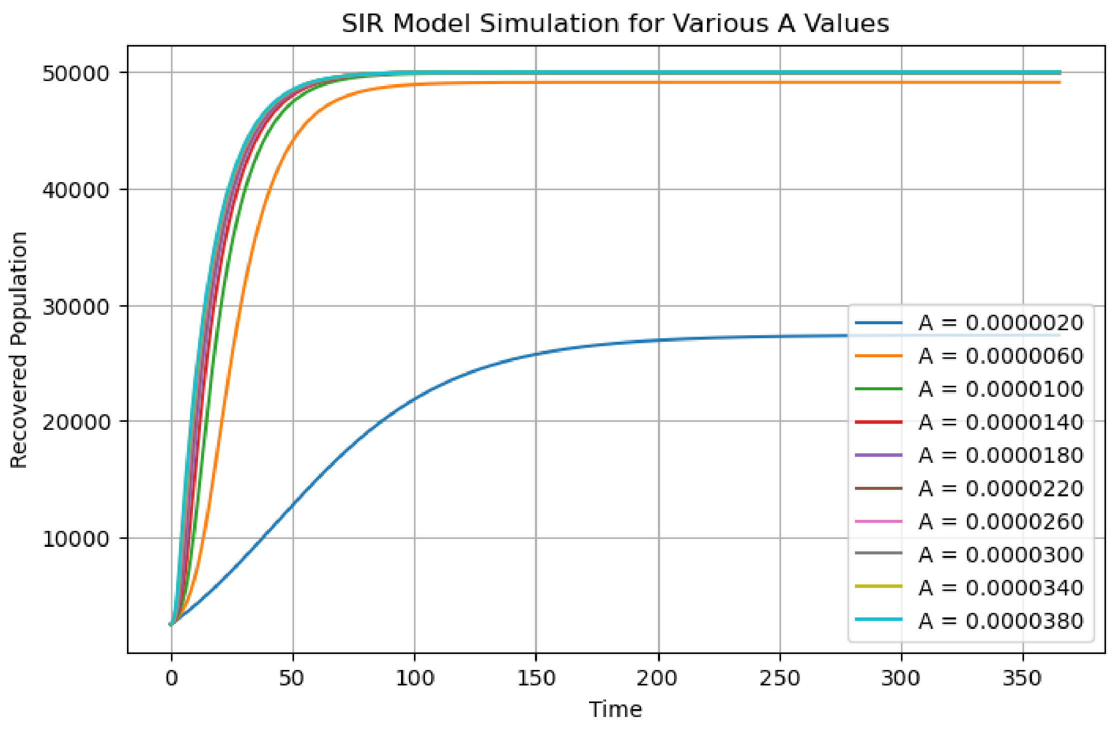


3. When A is larger, the Infected curve peaks much sooner, but when A is smaller, the Peak

doesn't happen for a while, and the peak is probably much smaller in total, since B isn't altered, so folks start recovering before the huge numbers of Infections can occur.

[18]:
```
def Rplot(Tinit, Tfinal, S, I, R, steps):
    dT = (Tfinal - Tinit) / steps
    Tvals = np.linspace(Tinit, Tfinal, steps+1)
    Avals = np.arange(0.000002, 0.00004, 0.000004)

    plt.figure(figsize=(8, 5))

    for A in Avals:
        Scurr, Icurr, Rcurr = S, I, R
        Svals, Ivals, Rvals = [Scurr], [Icurr], [Rcurr]

        for k in range(steps):
            Sp = -A * Scurr * Icurr
            Ip = A * Scurr * Icurr - Icurr / 14
            Rp = Icurr / 14

            Scurr += Sp*dT
            Icurr += Ip*dT
            Rcurr += Rp*dT

            Svals.append(Scurr)
            Ivals.append(Icurr)
            Rvals.append(Rcurr)

        plt.plot(Tvals, Rvals, label=f"A = {A:.7f}")

    plt.xlabel("Time")
    plt.ylabel("Recovered Population")
    plt.title("SIR Model Simulation for Various A Values")
    plt.legend()
    plt.grid(True)
    plt.show()
```

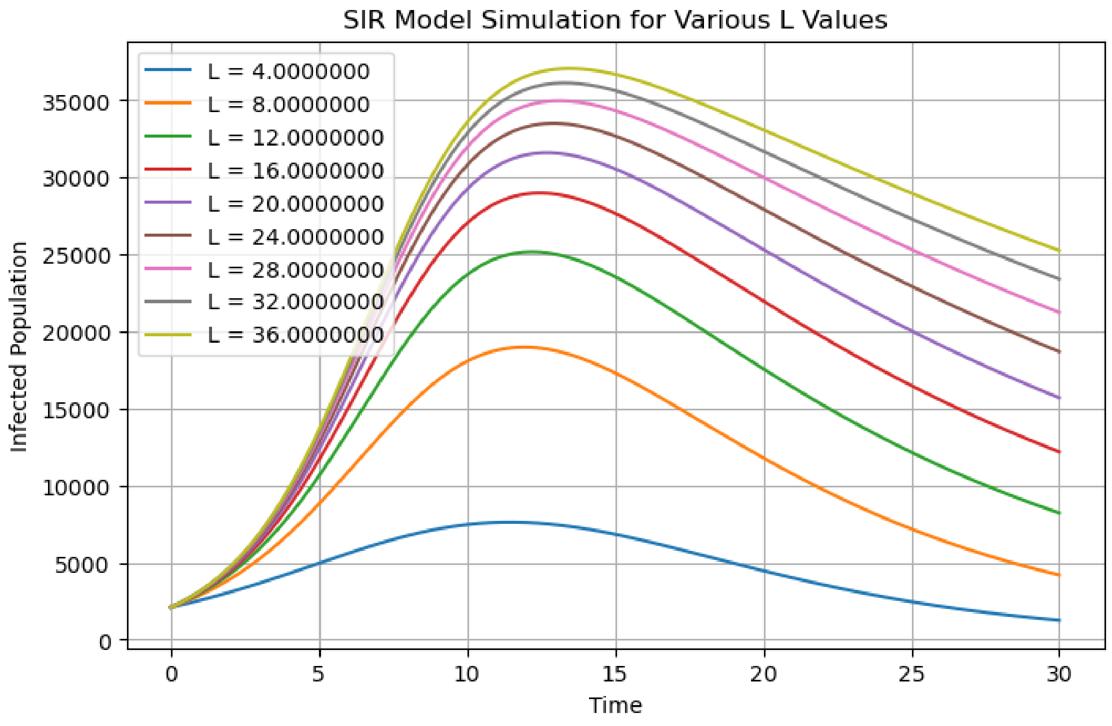
[23]:
```
Rplot(0, 365, 45400, 2100, 2500, 3650)
```



4. As soon as A hits 0.00001, pretty much everyone in the Susceptible group will contract the disease at some point or another. If A is less than 0.000006, then people in the susceptible group have a chance of never being infected. The only A that doesn't end with everyone in the R group being infected sometime during the epidemic is A = 0.000002.

[30]:
```
def Lplot(Tinit, Tfinal, S, I, R, steps):
    dT = (Tfinal - Tinit) / steps
    Tvals = np.linspace(Tinit, Tfinal, steps+1)
    Lvals = range(4, 40, 4)
    A = 0.00001
    plt.figure(figsize=(8, 5))

    for L in Lvals:
        Scurr, Icurr, Rcurr = S, I, R
        Svals, Ivals, Rvals = [Scurr], [Icurr], [Rcurr]

        for k in range(steps):
            Sp = -A * Scurr * Icurr
            Ip = A * Scurr * Icurr - Icurr / L
            Rp = Icurr / L

            Scurr += Sp*dT
            Icurr += Ip*dT
            Rcurr += Rp*dT
            Svals.append(Scurr)
            Ivals.append(Icurr)
            Rvals.append(Rcurr)

        plt.plot(Tvals, Ivals, label=f"L = {L:.7f}")

    plt.xlabel("Time")
    plt.ylabel("Infected Population")
    plt.title("SIR Model Simulation for Various L Values")
    plt.legend()
    plt.grid(True)
    plt.show()
```

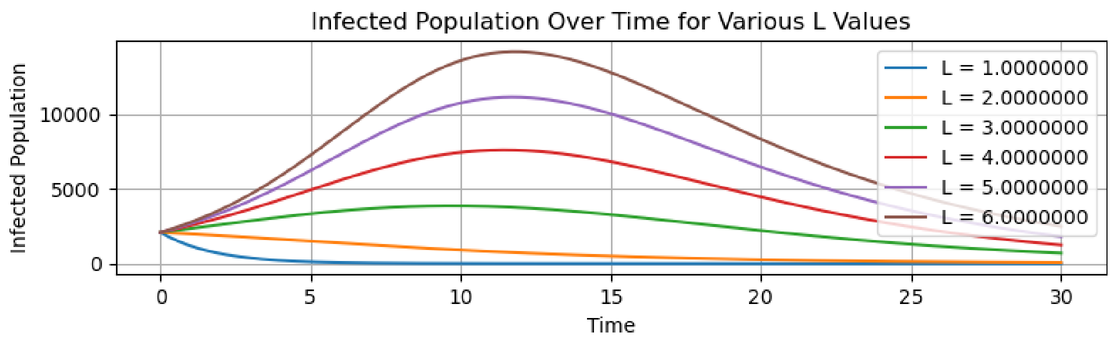
[31]:
```
Lplot(0, 30, 45400, 2100, 2500, 300)
```

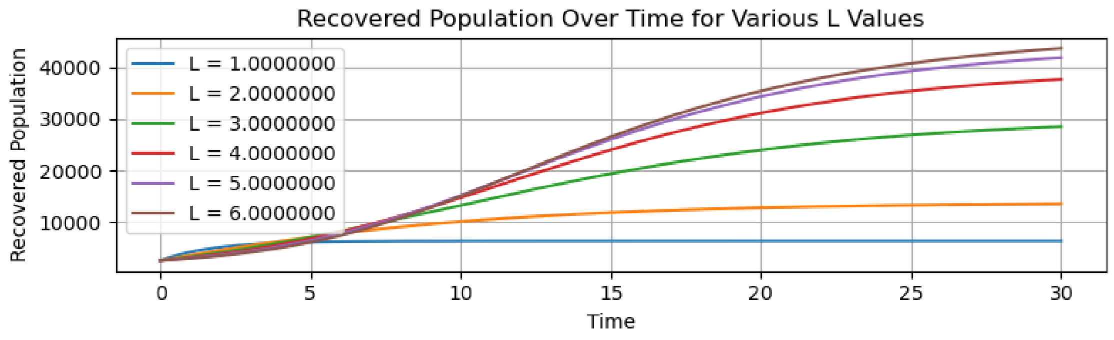


6. As L gets larger (and B gets smaller), the peak of the Infected curve also gets larger. Surprisingly enough, the Peaks don't happen at different times, they are all around 12-14 days.

[40]:
```
def Lplot2(Tinit, Tfinal, S, I, R, steps):
    dT = (Tfinal - Tinit) / steps
    Tvals = np.linspace(Tinit, Tfinal, steps+1)
    Lvals = range(1, 7)
    A = 0.00001
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 5))

    for L in Lvals:
        Scurr, Icurr, Rcurr = S, I, R
        Svals, Ivals, Rvals = [Scurr], [Icurr], [Rcurr]

        for k in range(steps):
            Sp = -A * Scurr * Icurr
            Ip = A * Scurr * Icurr - Icurr / L
            Rp = Icurr / L

            Scurr += Sp*dT
            Icurr += Ip*dT
            Rcurr += Rp*dT

            Svals.append(Scurr)
            Ivals.append(Icurr)
            Rvals.append(Rcurr)

        ax1.plot(Tvals, Ivals, label=f"L = {L:.7f}")
        ax2.plot(Tvals, Rvals, label=f"L = {L:.7f}")

    ax1.set_xlabel("Time")
    ax1.set_ylabel("Infected Population")
    ax1.set_title("Infected Population Over Time for Various L Values")
    ax1.legend()
    ax1.grid(True)

    ax2.set_xlabel("Time")
    ax2.set_ylabel("Recovered Population")
    ax2.set_title("Recovered Population Over Time for Various L Values")
    ax2.legend()
    ax2.grid(True)

    plt.tight_layout()
    plt.show()
```

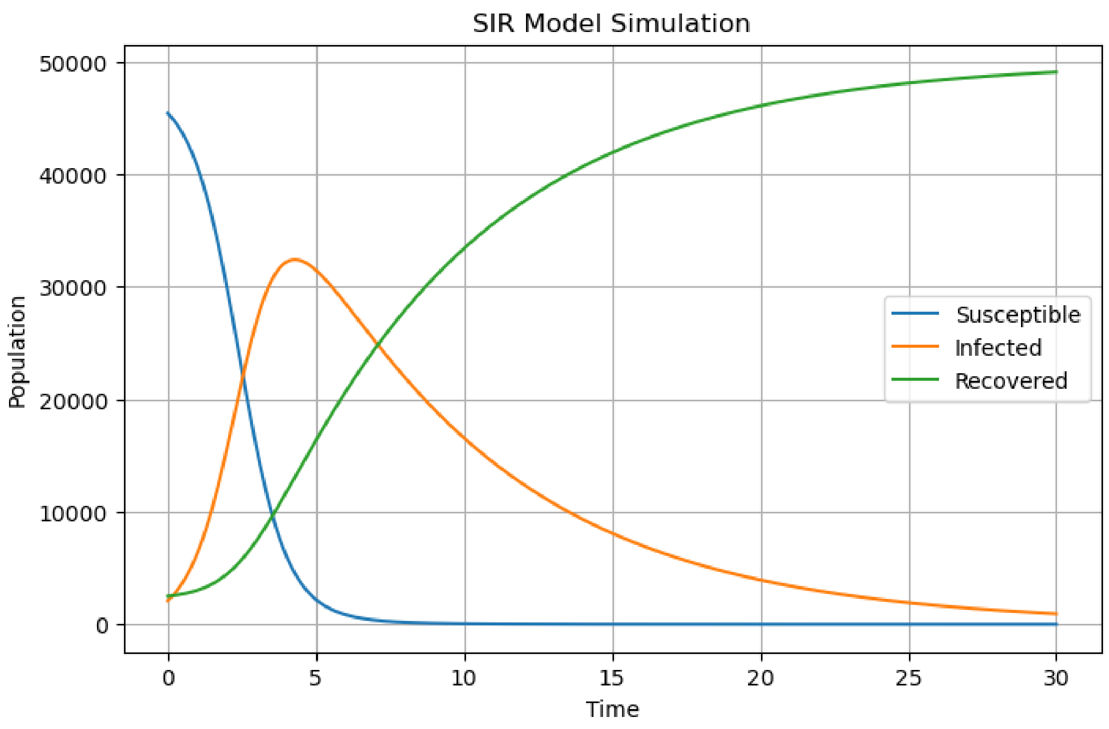
[41]:
```
Lplot2(0, 30, 45400, 2100, 2500, 300)
```



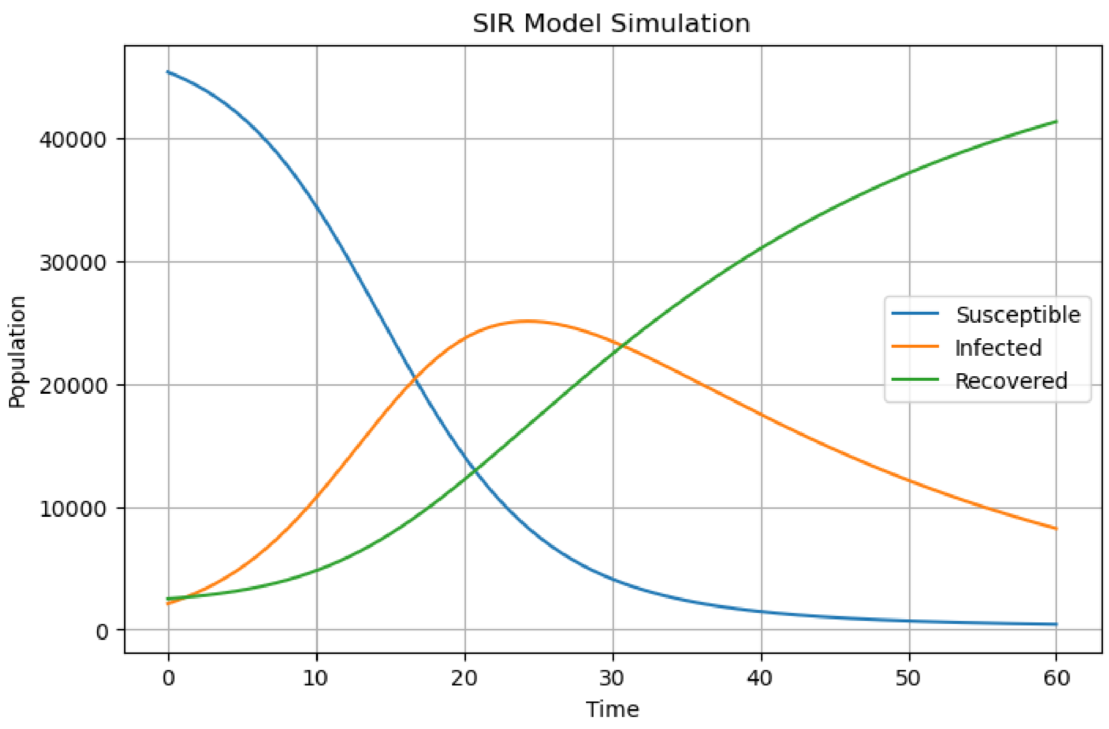


7. The big difference comes from the time needed to recover and how that makes it harder for more infections to happen, since folks recover before coming in contact with many people in the Susceptible population. Between L = 2 and L = 3 is when we see the biggest change, since the curves go from curving down (L=2) to curving up (L=3). We also see a big difference in total number of Recovered (total number of previously infected) between these two values for L as well. L=2 tends to flatten quickly, while L=3 or more shows the curve continuing to rise. At this point, our S would be equal to Sinitial - Rtotal.
8. If L is low enough, the total infected by end of epidemic is drastically reduced. For all of the Susceptible population to be Infected and Recover, L has to hit a certain threshold.

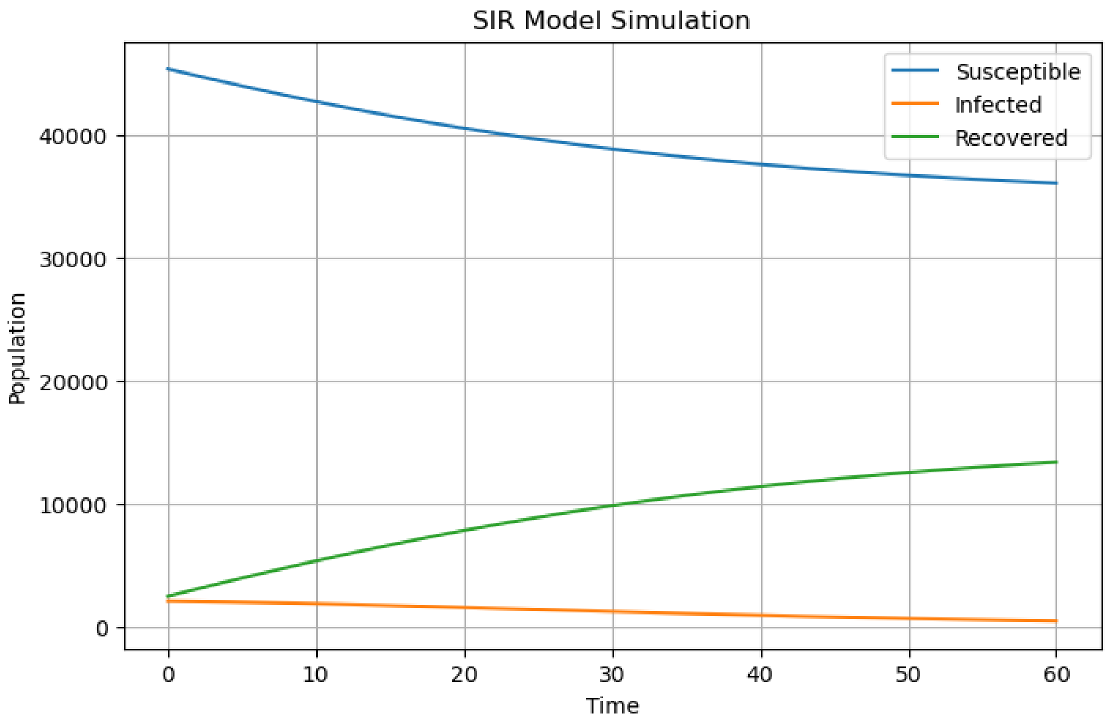
```
[52]: # A = 0.00003, L = 7
      SIRplot(0, 30, 45400, 2100, 2500, 0.00003, 7, 300)
```



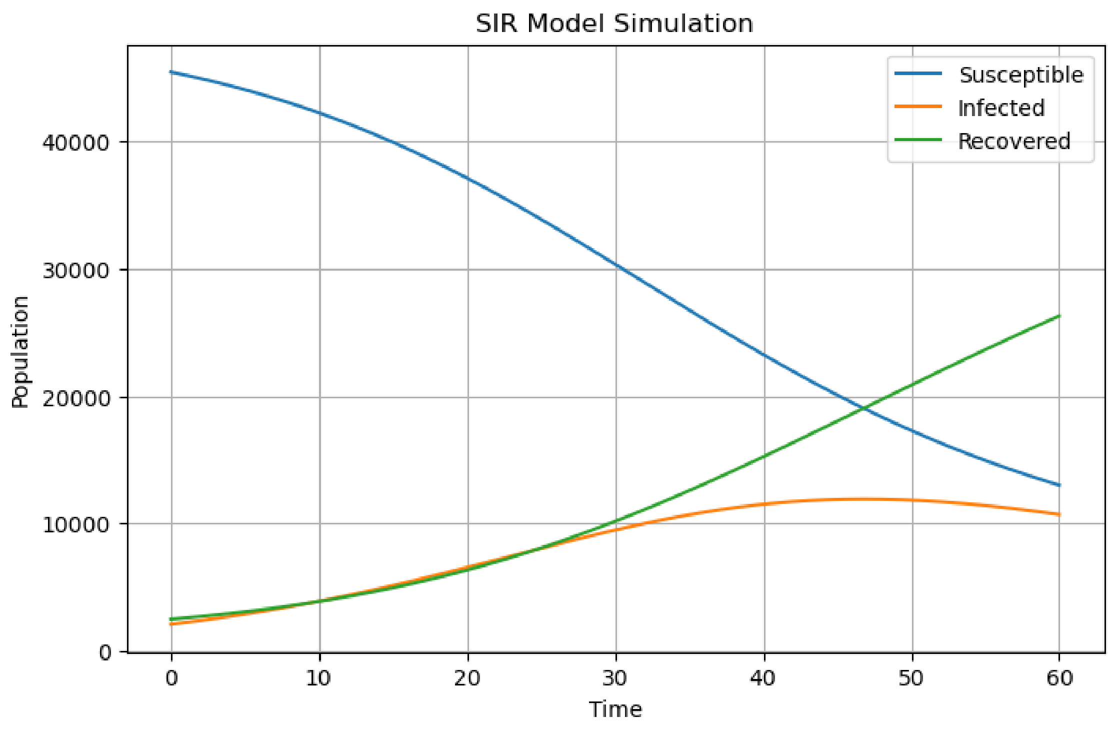
[59]:
```
# A = 0.000005, L = 24
SIRplot(0, 60, 45400, 2100, 2500, 0.000005, 24, 600)
```



[71]:
```
# A = 0.000003, L = 7
SIRplot(0, 60, 45400, 2100, 2500, 0.000003, 7, 600)
```



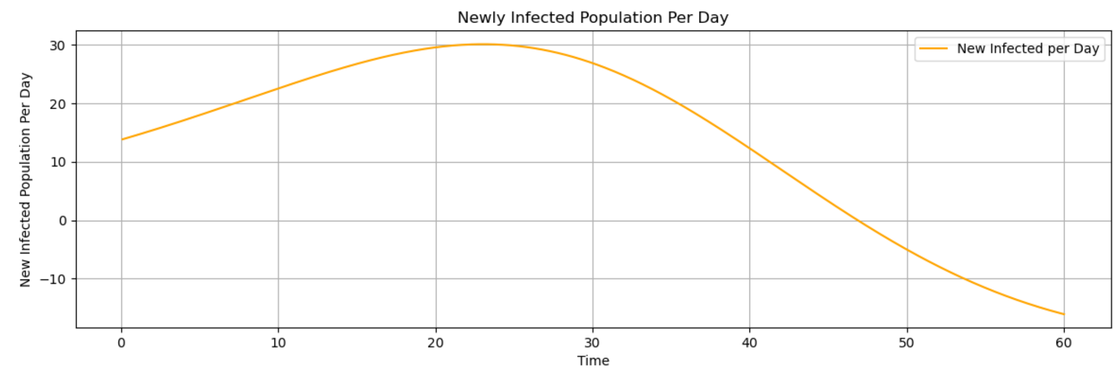
[78]:
```
# A = 0.0000025, L = 21
SIRplot(0, 60, 45400, 2100, 2500, 0.0000025, 21, 600)
```

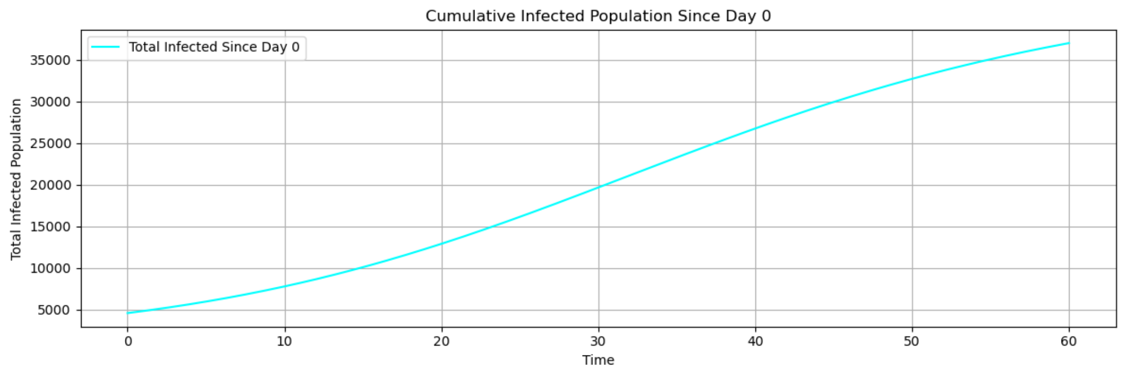


11. A higher infection rate (A) increases spread of disease, increasing the threshold. Larger L means an infected individual can spread disease for a longer time. Higher L also increases the threshold. B shows the currect infected population impact on the spread, but doesn't directly affect the threshold itself.

[79]:
```python
def SIRplot2(Tinit, Tfinal, S, I, R, A, L, steps):
    dT = (Tfinal - Tinit) / steps
    Tvals = [Tinit]
    Svals = [S]
    Ivals = [I]
    Rvals = [R]
    new_Ivals = []
    total_Ivals = [I+R]

    for k in range(1, steps+1):
        Sp = -A*S*I
        Ip = A*S*I - I/L
        Rp = I/L
        dS = Sp*dT
        dI = Ip*dT
        dR = Rp*dT

        Tinit = Tinit + dT
        S += dS
        I += dI
        R += dR

        Tvals.append(Tinit)
        Svals.append(S)
        Ivals.append(I)
        Rvals.append(R)

        new_Inf = Ivals[-1] - Ivals[-2]
        total_Inf = I + R

        new_Ivals.append(new_Inf)
        total_Ivals.append(total_Inf)

    plt.figure(figsize=(12, 8))

    plt.subplot(2, 1, 1)
    plt.plot(Tvals[1:], new_Ivals, label="New Infected per Day", color="orange")
    plt.xlabel("Time")
    plt.ylabel("New Infected Population Per Day")
    plt.title("Newly Infected Population Per Day")
    plt.legend()
    plt.grid(True)

    plt.subplot(2, 1, 2)
    plt.plot(Tvals, total_Ivals, label="Total Infected Since Day 0",␣
  ↪color="cyan")
    plt.xlabel("Time")
    plt.ylabel("Total Infected Population")
    plt.title("Cumulative Infected Population Since Day 0")
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

[80]:
```
SIRplot2(0, 60, 45400, 2100, 2500, 0.0000025, 21, 600)
```





CASE STUDY - Measles Panic

After plugging our given numbers into SIR model and coming up with some estimates for A, we found that A is ~0.0000065. After looking at the rest of the graph, the mayor is super wrong about this not being a big deal. Almost every member of the city's population will have the infection at some point within the first 2 months with the Peak of infection around day 33.

1.2 - 24, 25, 28

24.

k is the yearly rate of change in population per person (per capita) in said population. The net growth rate (P') is in units of persons per year. the population (P) is in units of persons, so yearly rate of change (k) is P'/P, so units of persons per year / persons.

25.

a) P' = 0.009(P); A' = 0.0216(A)

b) 0.009(37500000) = P', so P' = 337500; 0.0216(15000000) = 324000; The only guarantee that the net growth rate will be larger is if the 2 populations are the same to start As we see, Afghanistan has a larger per capita growth rate (0.0216) compared to Poland's (0.009) but Afghanistan's net growth rate (324000) is smaller than Poland's (337500) due to Poland starting with a much larger overall Population.

c)

years = amount of growth / growth rate; years = 1 person / P' (or A') years = 1 person / 337500 persons per year = 2.96e-6 years = 1.55 minutes

years = 1 person / 324000 person per year = 3.09e-6 years = 1.62 minutes

28.

a) C’ = -k(C-70)

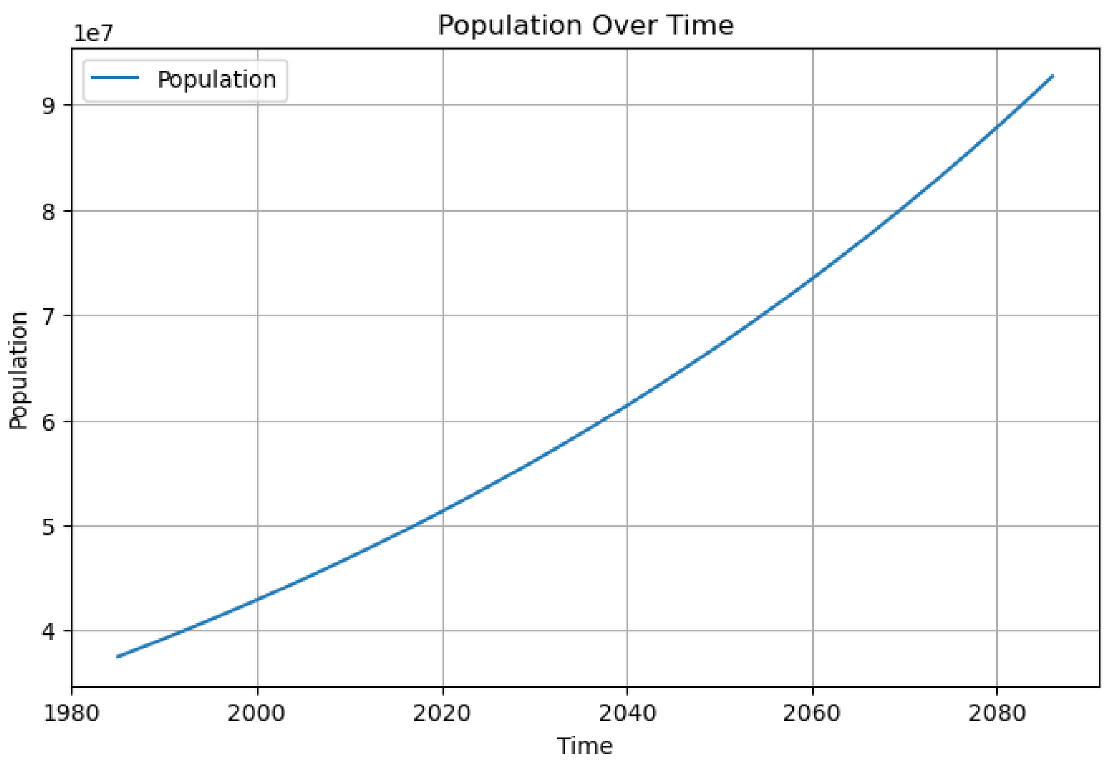
b) -9 = -k(180-70); k = 9/110 = 0.08182

c) -0.08182(120-70) = C’; -4.091 = C’

d) at 180, C’ = -9. at 120, C’ = -4. We can average the 2 rates and get 6.5. 180-120 = 60. 60/6.5 = 9.23, so around 9.23 minutes. It’s probably closer to 9 minutes, since C’ at 120 is slightly faster than -4.

[88]:
```
#2.2 number 1
def Poland(P, k, t):
    Tvals = [1985]
    Pvals = [P]
    year = 1985
    for year in range(1985, 1985+t+1):
        year += 1
        Pprime = k*P
        P += Pprime

        Pvals.append(P)
        Tvals.append(year)

    plt.figure(figsize=(8, 5))
    plt.plot(Tvals, Pvals, label="Population")

    plt.xlabel("Time")
    plt.ylabel("Population")
    plt.title("Population Over Time")
    plt.legend()
    plt.grid(True)
    plt.show()
```

[89]:
```
Poland(37500000, 0.009, 100)
```

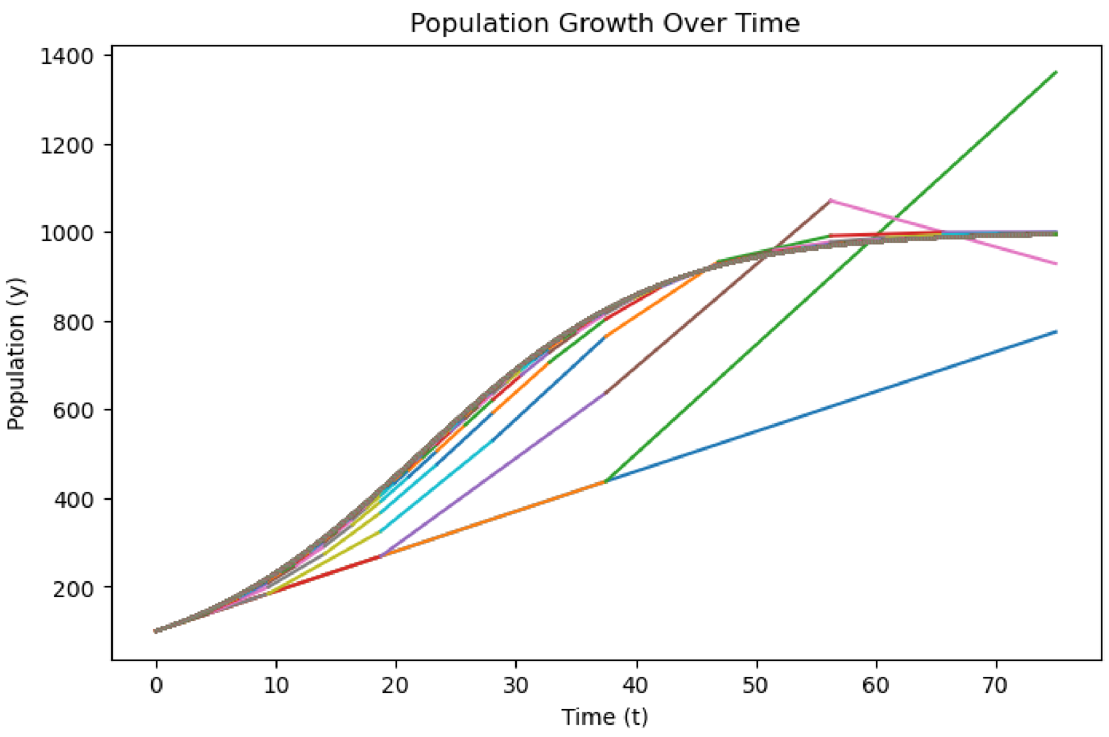


a) Population of Poland in 2085 is ~93000000 people

b) see graph above

[15]:
```python
#2.2 number 2

def sequence(Tinit, Tfinal, Yinit, r, cc, con):
    plt.figure(figsize=(8, 5))

    for j in range(r+1):
        t = Tinit
        y = Yinit
        steps = 2**(j-1)
        dT = (Tfinal - Tinit) / steps
        for k in range(int(steps)):
            Yprime = con * y * (1 - y / cc)
            dY = Yprime * dT

            plt.plot([t, t + dT], [y, y + dY])

            t += dT
            y += dY
    plt.xlabel("Time (t)")
    plt.ylabel("Population (y)")
    plt.title("Population Growth Over Time")

    plt.show()
```

[16]:
```
sequence(0, 75, 100, 14, 1000, 0.1)
```



2.2 Question 3:

a) as t becomes larger, the graph flattens out and creates a sort of plateau

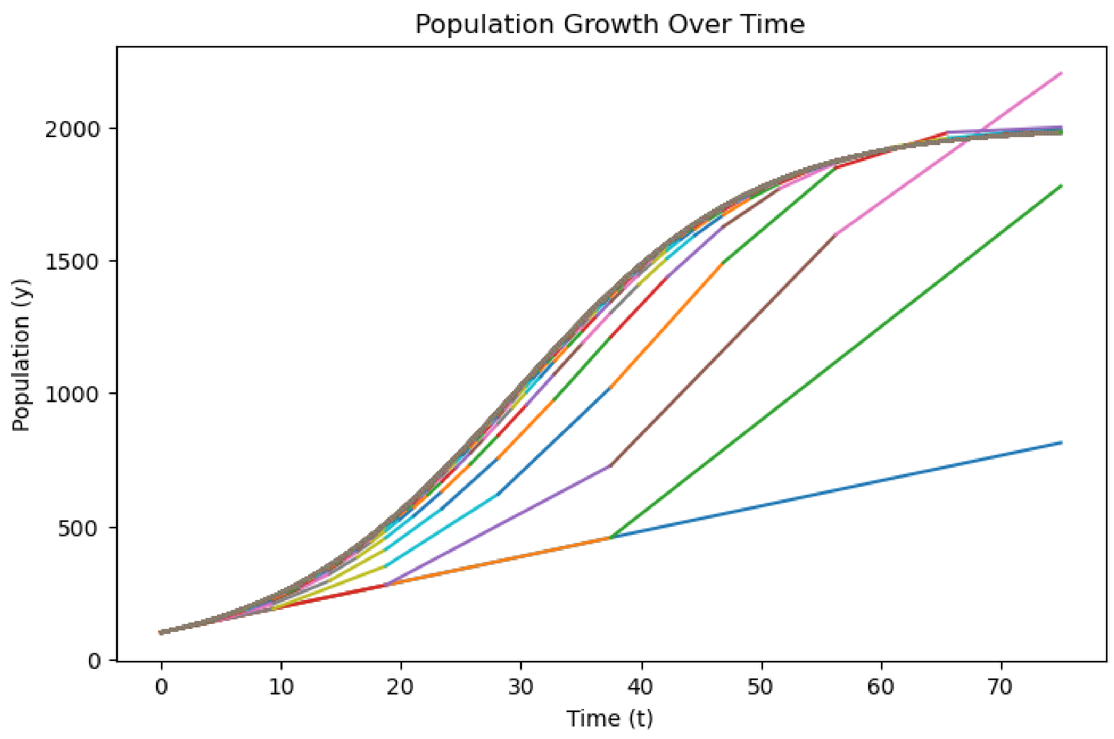
b) With y(0) = 1000, the population stays the same the whole time, making a flat graph at 1000 since the equation has (1 - 1000/1000), which makes 0, then .1y * 0 is still 0.

c) With y(0) = 1500, the population dips down until it hits 1000, then flattens like on b).

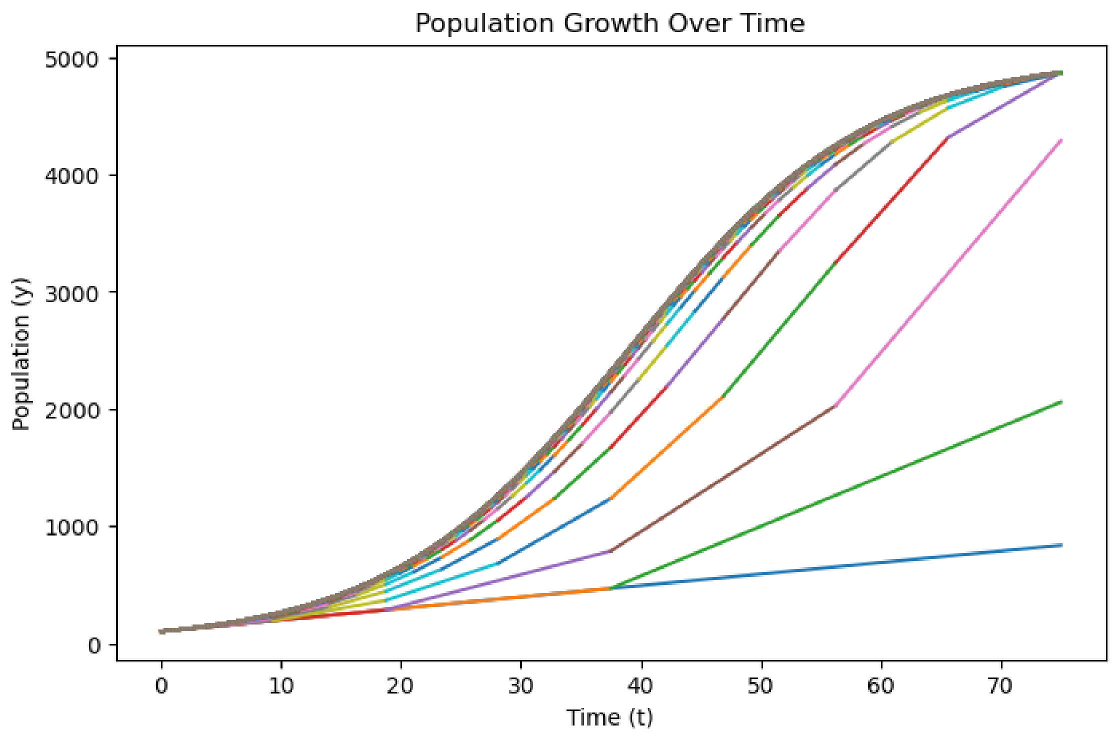
d) Again, the graph stays flat, but this time at the 0 mark on the y axis. This is because the .1y part of the equation would be 0, so y' would continually be zero.

e) In nature, species populations reach a carrying capacity/equilibrium depending on their food source and how much of it is around, hence why colder climates usually have less life and tropical climates have more life. Solely due to the abundance of food in the food chain.

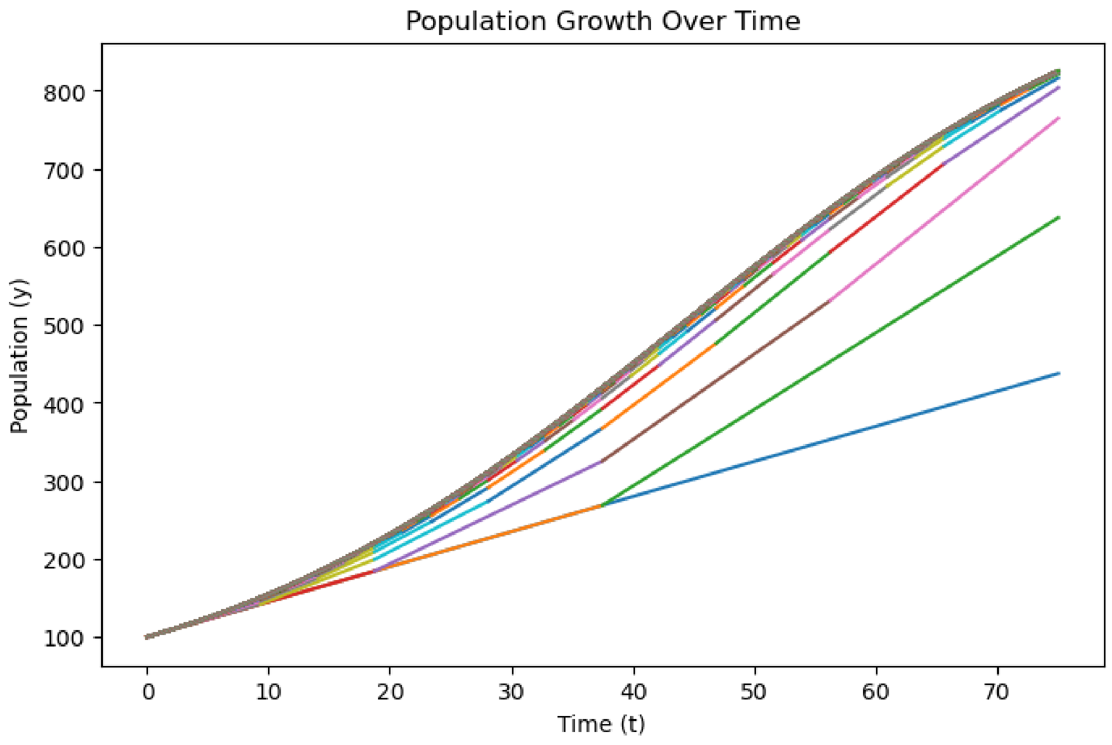
[17]:
```
#2.2 Question 4:
sequence(0, 75, 100, 14, 2000, 0.1)
```



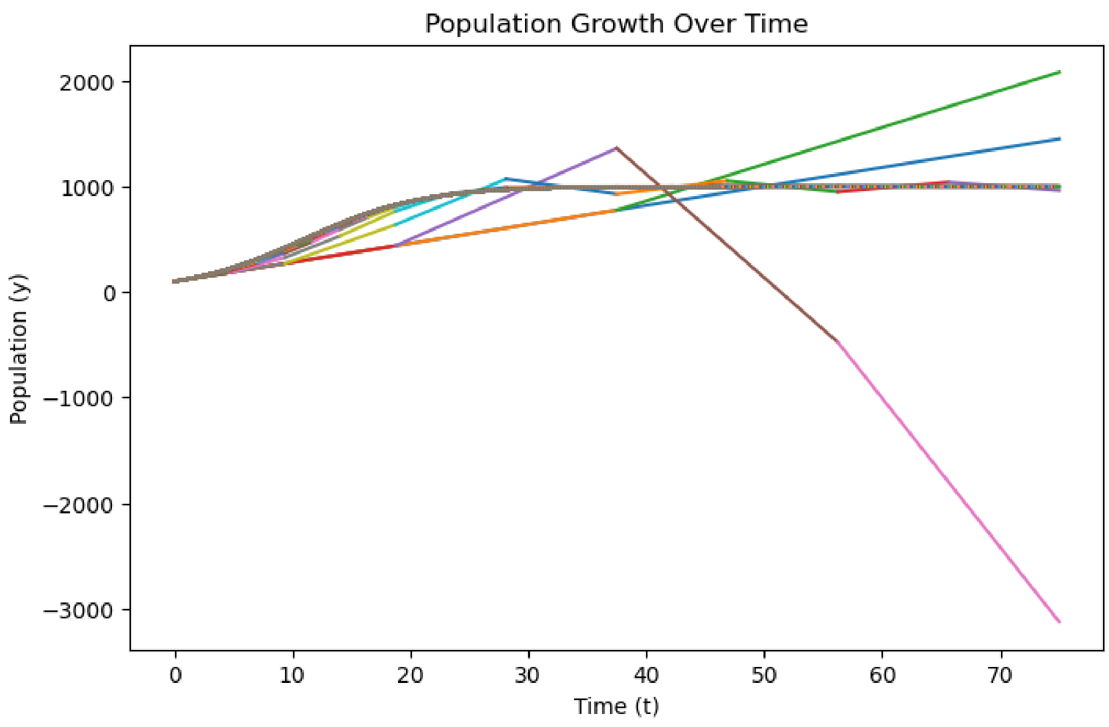
```
[18]: #2.2 Question 4 continued
      sequence(0, 75, 100, 14, 5000, 0.1)
```



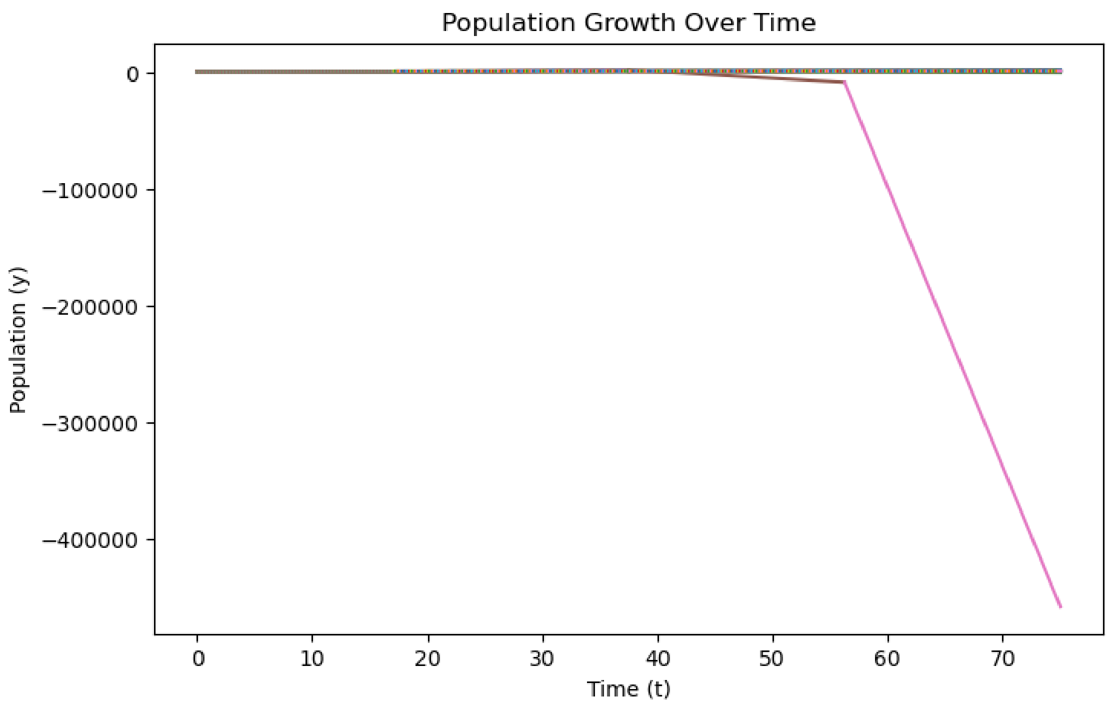
[19]:
```
#2.2 Question 5
sequence(0, 75, 100, 14, 1000, .05)
```



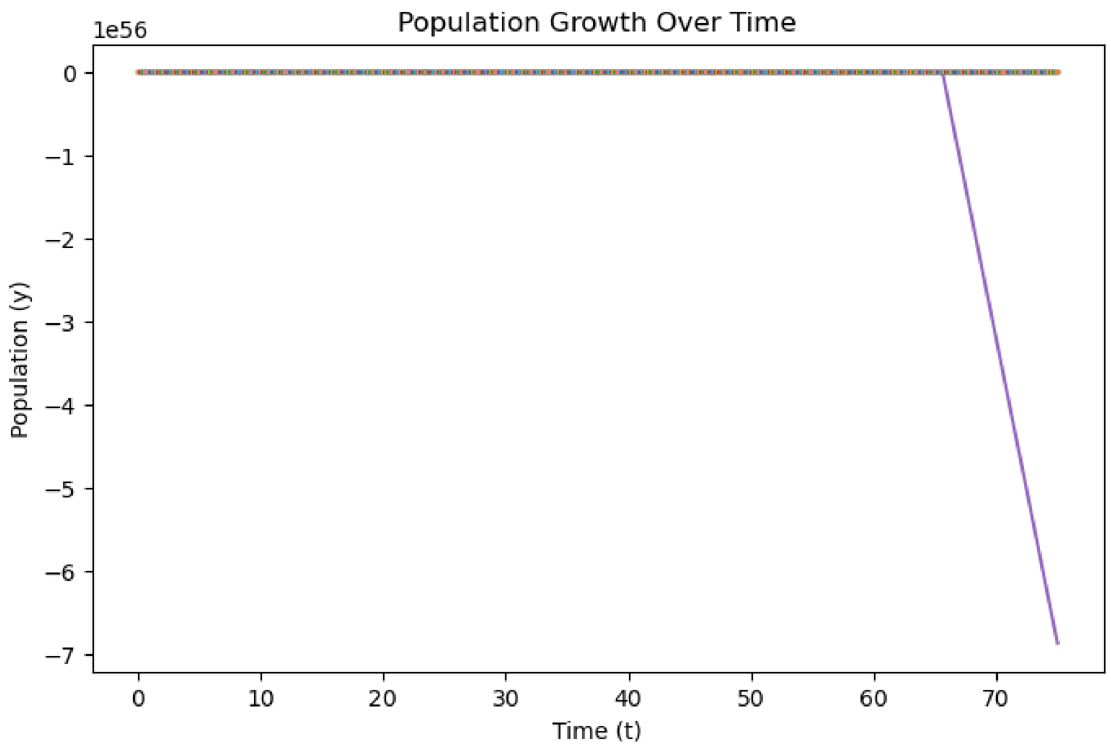
[20]:
```
#2.2 Question 5 continued
sequence(0, 75, 100, 14, 1000, .2)
```



[21]:
```
#2.2 Question 5 cont.
sequence(0, 75, 100, 14, 1000, .3)
```



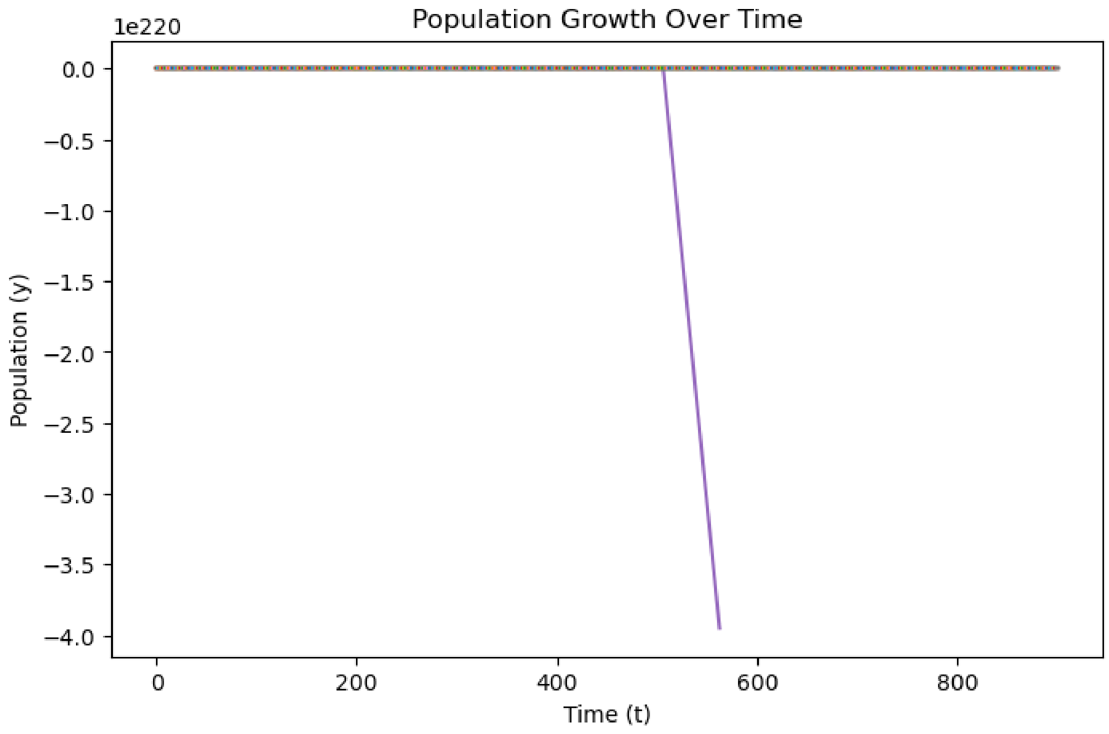
[22]:
```
#2.2 Question 5 cont.
sequence(0, 75, 100, 14, 1000, .6)
```



2.2 Question 5 explanation

The behavior of the solution hits its carrying capacity much quicker as the constant increases, but slower when he constant decreases.

[23]:
```
#2.2 Question 6
sequence(0, 900, 100, 14, 1000, .1)
```



2.2 Question 7

I looked up the general form of the logistic growth equation:

$$y(t) = \frac{K}{1 + (\frac{K - y_0}{y_0})e^{-rt}}$$

when we substitute in the information we are given, we get this:

$$900 = \frac{1000}{1 + (\frac{1000 - 100}{100})e^{-20r}}$$

which simplifies to:

$$900 = \frac{1000}{1 + 9e^{-20r}}$$

we do some simple algebra:

$$1 + 9e^{-20r} = \frac{1000}{900} = \frac{10}{9}$$

$$9e^{-20r} = \frac{10}{9} - 1 = \frac{1}{9}$$

$$e^{-20r} = \frac{1}{81}$$

Solving for r:

$$-20r = ln\frac{1}{81}$$

$$-20r = -ln(81)$$

$$r = \frac{ln(81)}{20}$$

$$r = \frac{4.3944}{20} \approx 0.2197$$

[ ]:

```
#2.3 Question 1
def func(x):
    Xvals = []
    Yvals = []

    for k in range(x):
        y = k**2 - 2**k
        Xvals.append(k)
        Yvals.append(y)
    print(Xvals)
    print(Yvals)

    plt.figure(figsize=(8, 5))
    plt.plot(Xvals, Yvals)

    plt.xlabel("X values")
    plt.ylabel("Y values")
    plt.title("X and Y values")
    plt.grid(True)
    plt.show()
```

[10]:

```
#2.3 Question 2
import math

def Length(xinit, xfinal, steps):
    dx = (xfinal - xinit) / steps
    total = 0
    for k in range(1, steps+1):
        xl = xinit + (k-1) * dx
        xr = xinit + k * dx
        yl = xl**2
        yr = xr**2
        seg = math.sqrt((xr - xl)**2 + (yr - yl)**2)
        total += seg
    print(steps, total)
```

[12]:
```
Length(0, 1, 2)
```

```
1 0.5590169943749475
2 0.9013878188659973
2 1.4604048132409448
```

[17]:
```
Length(0, 1, 4)
```

```
1 0.2576941016011038
2 0.3125
3 0.40019526483955303
4 0.5038911092686593
4 1.4742804757093162
```

[23]:
```
Length(0, 1, 8)
```

```
1 0.12597277731716483
2 0.1335000585205864
3 0.14740595518838442
4 0.1660960283239789
5 0.18814991529362962
6 0.2125229766989913
7 0.2385052737886523
8 0.265625
8 1.477777985131388
```

2.3 Question 3 3) yl = xl**2 yr = xr**2 dx = (xfinal - xinit) / steps

4) xl = xinit + (k-1) * dx xr = xinit + k * dx yl = xl**2 yr = xr**2

5) seg = math.sqrt((xr - xl)**2 + (yr - yl)**2)

6) xr - xl yr - yl

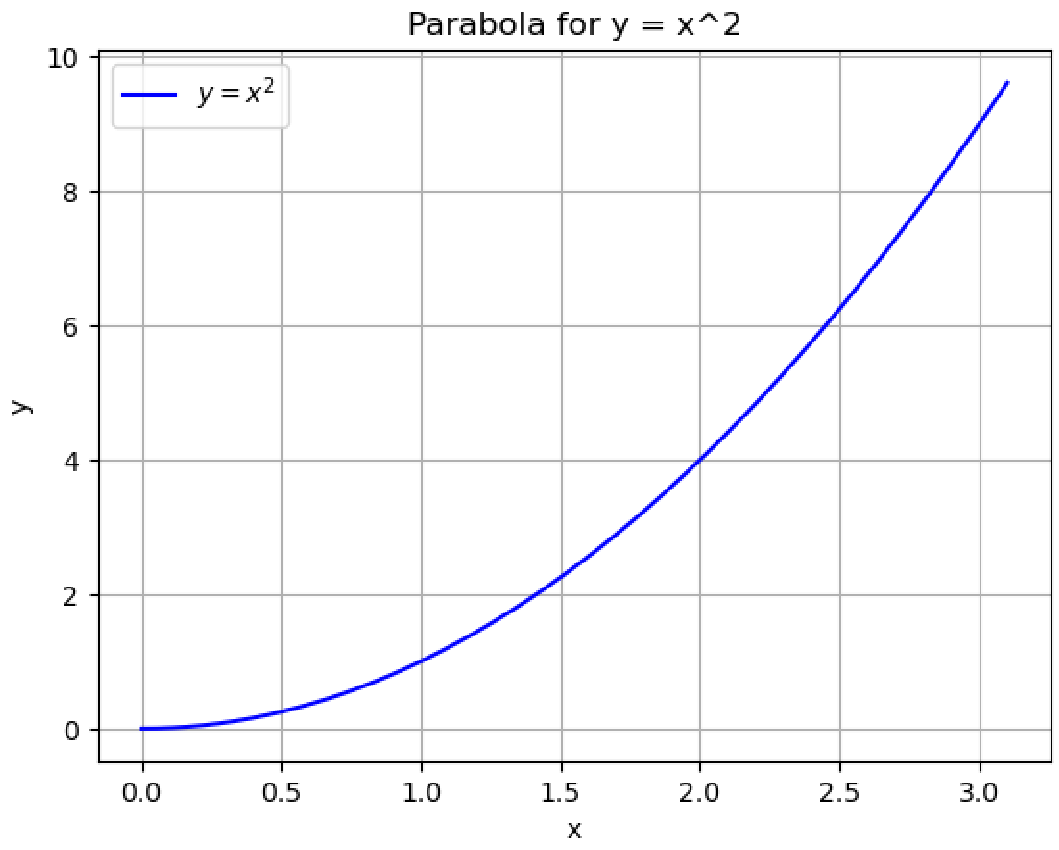
[12]:
```
#2.2 Question 7
Length(0, 1, 2)
Length(0, 1, 20)
Length(0, 1, 200)
Length(0, 1, 2000)
Length(0, 1, 20000)
Length(0, 1, 200000)
Length(0, 1, 2000000)
```

```
2 1.46040481324O9448
20 1.4787565120290738
200 1.4789409941539637
2000 1.4789428389106984
20000 1.4789428573582584
200000 1.4789428575427506
2000000 1.4789428575446044
```

2.2 Question 8 8) 8 decimal places = 1.47894285 12 decimal places = 1.478942857544

[21]:
```python
import numpy as np

def Length(xinit, xfinal, steps):
    dx = (xfinal - xinit) / steps
    total = 0

    for k in range(1, steps+1):
        xl = xinit + (k-1) * dx
        xr = xinit + k * dx
        yl = xl**2
        yr = xr**2

        seg = math.sqrt((xr - xl)**2 + (yr - yl)**2)
        total += seg

    print(steps, total)

    # Generate points to plot the parabola y = x^2
    x_curve = np.linspace(xinit, xfinal, 500)
    y_curve = x_curve**2

    # Plot the parabola and the line segments
    plt.plot(x_curve, y_curve, label=r'$y = x^2$', color='blue')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Parabola for y = x^2')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage
Length(0, 3.1, 100)
```

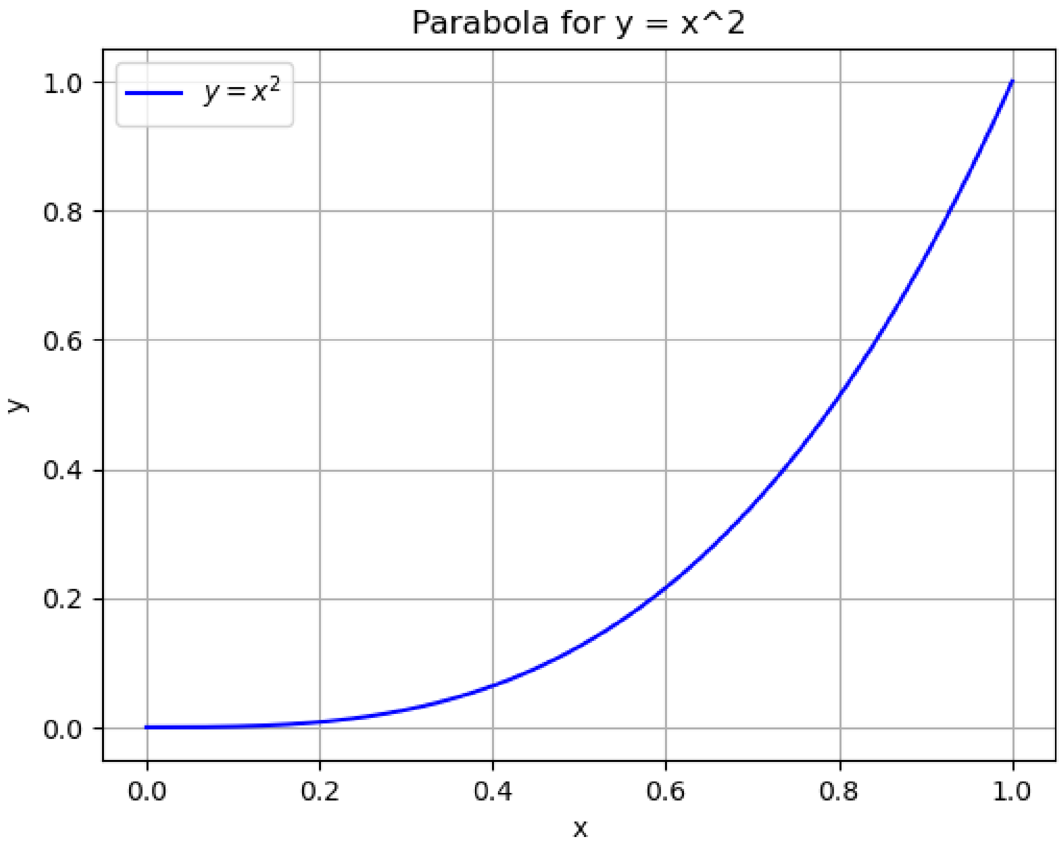
```
100 10.365152781199933
```



2.3 Question 9 9) by the time we've moved a distance of ~10, the parabola has only reached y ~9.4

[22]:
```
#2.3 Question 10
def Length(xinit, xfinal, steps):
    dx = (xfinal - xinit) / steps
    total = 0

    for k in range(1, steps+1):
        xl = xinit + (k-1) * dx
        xr = xinit + k * dx
        yl = xl**2
        yr = xr**2

        seg = math.sqrt((xr - xl)**2 + (yr - yl)**2)
        total += seg

    print(steps, total)

    x_curve = np.linspace(xinit, xfinal, 500)
    y_curve = x_curve**3
    plt.plot(x_curve, y_curve, label=r'$y = x^2$', color='blue')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Parabola for y = x^3')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage
Length(0, 1, 20000)
```

```
20000 1.4789428573582584
```



[24]:
```
#I thought the Babylon algorithm was interesting, so I coded it up too. It␣
↪seems to work alright.
def babylon(a, x, n):
    for k in range(n):
        x = (x + a / x) / 2
        print(x)
```

```
[25]: babylon(5, 2, 6)
```

```
2.25
2.2361111111111111
2.2360679779158037
2.23606797749979
2.23606797749979
2.23606797749979
```

```
[ ]:
```