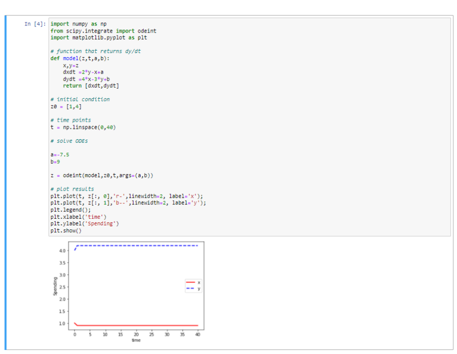
# ASSIGNMENT 2 - MATH 4090 (Winter 2022)

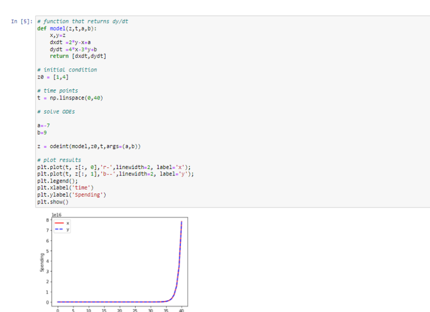
STUDENT: Francis Calingo (214451736)

**Question 1**:

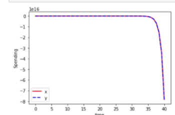
a. **Modelling Careers**
Some examples of careers that require mathematical modelling include machine learning scientists (a position Dr. Michael Akinwumi occupies), AI engineer (a position John Murdoch occupies), health economist (something Dr Angie Read specializes in), risk analyst, and methodologist.
According to a US News report, mathematicians can make 80,000 to 120,000 per year in 2020.
`https://money.usnews.com/careers/best-jobs/mathematician/salary`

b. **Arms Race**
Given that an increased distrust correlates to more militarization, a positive parameter means higher distrust, a negative parameter means lower distrust, and 0 means neutral.
In most cases, there will be a runaway arms race. However, for certain parameters, there will be a stabilized arms race. Take, for example, a=-7.5 and b=9. Country x's military spending stabilizes slightly around 4.25 and Country y's spending stabilizes around 1.75. If we were to slightly increase a or b, it immediately becomes a runaway race or shows an impossible scenario (i.e., spending approaches negative infinity).

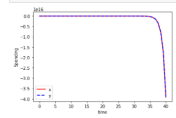




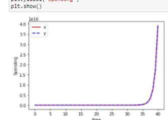
```
In [6]: # function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-8
b=9

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```




```
In [21]: # function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-7.5
b=8.5

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```

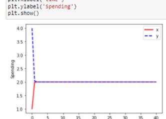



```
In [8]: # function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-7.5
b=9.5

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```

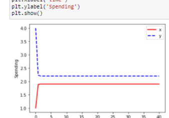



When a=b=-2, both countries' spending stabilizes around 2. Now observe that if we decrease a by 0.5 and increase b by 1, the arms race will still be stabilized, but Country Y's spending begins to outpace Country x's until x and y have constant spending (4 and 1, respectively) when a=-7 and b=9. This makes sense because since Country x has a low distrust of Country y, it's spending is much less. Conversely, Country y's strong distrust of x means its spending will vastly outpace x's.

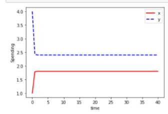
```
In [12]: # function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-2
b=-2

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```



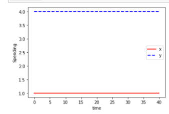

```
In [14]: # function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-2.5
b=-1

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```



In [15]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-3
b=0

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```

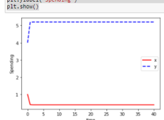



In [18]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-7
b=8

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```

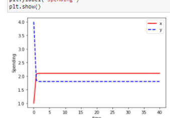



Let's start again at a=b=-2. Now observe that if we increase a by 0.5 and decrease b by 1, the arms race will still be stabilized, but Country x's spending begins to outpace Country y's. Again, this makes sense because as Country x's distrust increases, so does its spending, while as Country y's distrust decreases, so does its spending. Of course, we cannot keep decreasing/increasing that way, as the model will eventually reach negative spending, which is of course impossible.

In [19]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-10
b=14

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```

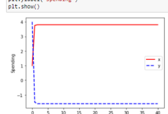



In [20]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=-1.5
b=-3

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```



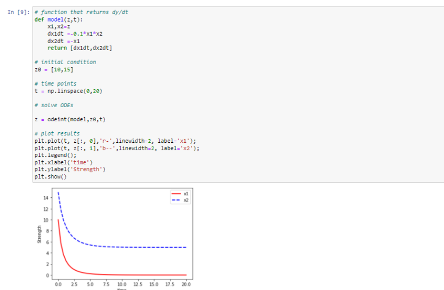

In [22]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=2
b=-10

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```


In [23]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    x,y=z
    dxdt =2*y-x+a
    dydt =4*x-3*y+b
    return [dxdt,dydt]

# initial condition
z0 = [1,4]

# time points
t = np.linspace(0,40)

# solve ODEs

a=7
b=-20

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Spending')
plt.show()
```



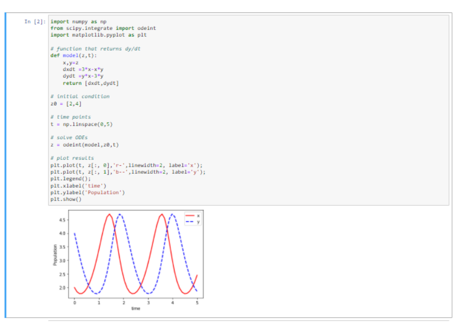

c. **Combat**

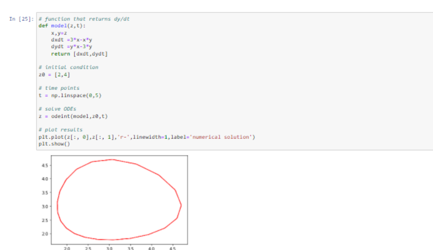
According to the results, conventional troops will win since over time, x1 will approach some constant slightly less than 6, but x2 will approach 0. This mode therefore supports the belief that because conventional troops are generally better trained, more well-equipped, and have better tactics, they will beat the guerrillas over time.




## Question 2:

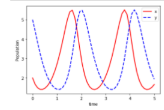
a. **Predator Prey**
For x0=2 and y0=4, the first graph is x and y plotted against t, and the second is x plotted against y.



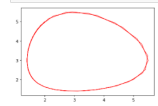




As expected, the first graph is periodic, given that prey population declines as predator population increases, but that trend reverses, when there is little prey for the predators to eat. The second graph is an ellipse, as expected when graphing x against y.
For x0=2 and y0=5, the first graph is x and y plotted against t, and the second is x plotted against y.

```
In [4]: # function that returns dy/dt
def model(z,t):
    x,y=z
    dxdt =3*x-x*y
    dydt =y*x-3*y
    return [dxdt,dydt]

# initial condition
z0 = [2,5]

# time points
t = np.linspace(0,5)

# solve ODEs
z = odeint(model,z0,t)

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Population')
plt.show()
```

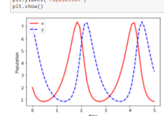



```
In [5]: # function that returns dy/dt
def model(z,t):
    x,y=z
    dxdt =3*x-x*y
    dydt =y*x-3*y
    return [dxdt,dydt]

# initial condition
z0 = [2,5]

# time points
t = np.linspace(0,5)

# solve ODEs
z = odeint(model,z0,t)

# plot results
plt.plot(z[:, 0],z[:, 1],'r-',linewidth=1,label='numerical solution')
plt.show()
```

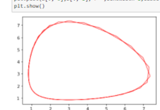


For x0=2 and y0=7, the first graph is x and y plotted against t, and the second is x plotted against y.

```
In [6]: # function that returns dy/dt
def model(z,t):
    x,y=z
    dxdt =3*x-x*y
    dydt =y*x-3*y
    return [dxdt,dydt]

# initial condition
z0 = [2,7]

# time points
t = np.linspace(0,5)

# solve ODEs
z = odeint(model,z0,t)

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='x');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='y');
plt.legend();
plt.xlabel('time')
plt.ylabel('Population')
plt.show()
```



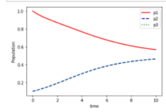

```
In [8]: # function that returns dy/dt
def model(z,t):
    x,y=z
    dxdt =3*x-x*y
    dydt =y*x-3*y
    return [dxdt,dydt]

# initial condition
z0 = [2,7]

# time points
t = np.linspace(0,5)

# solve ODEs
z = odeint(model,z0,t)

# plot results
plt.plot(z[:, 0],z[:, 1],'r-',linewidth=1,label='numerical solution')
plt.show()
```



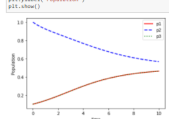
Observe that as y0 increases, there is a significant change in the dynamics of x and y. x's declines and y's growth are much steeper, with the reverse being true for their growth and declines, respectively. As a result, the x vs y plot is less smooth.

b. **Competing Species**

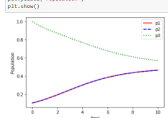
In [43]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    p1,p2,p3=z
    dp1dt =p1*(1-p1-a*p2-b*p3)
    dp2dt =p2*(1-b*p1-p2-a*p3)
    dp3dt =p3*(1-a*p1-b*p2-p3)
    return [dp1dt,dp2dt,dp3dt]

# initial condition
z0 = [1,0.1,0.1]

# time points
t = np.linspace(0,10)

# solve ODEs
a=0.5
b=0.5

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:,0],'r-',linewidth=2, label='p1');
plt.plot(t, z[:,1],'b--',linewidth=2, label='p2');
plt.plot(t, z[:,-1],'g:',linewidth=2, label='p3');
plt.legend();
plt.xlabel('time')
plt.ylabel('Population')
plt.show()
```




In [45]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    p1,p2,p3=z
    dp1dt =p1*(1-p1-a*p2-b*p3)
    dp2dt =p2*(1-b*p1-p2-a*p3)
    dp3dt =p3*(1-a*p1-b*p2-p3)
    return [dp1dt,dp2dt,dp3dt]

# initial condition
z0 = [0.1,1,0.1]

# time points
t = np.linspace(0,10)

# solve ODEs
a=0.5
b=0.5

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:,0],'r-',linewidth=2, label='p1');
plt.plot(t, z[:,1],'b--',linewidth=2, label='p2');
plt.plot(t, z[:,2],'g:',linewidth=2, label='p3');
plt.legend();
plt.xlabel('time')
plt.ylabel('Population')
plt.show()
```




In [46]:
```python
# function that returns dy/dt
def model(z,t,a,b):
    p1,p2,p3=z
    dp1dt =p1*(1-p1-a*p2-b*p3)
    dp2dt =p2*(1-b*p1-p2-a*p3)
    dp3dt =p3*(1-a*p1-b*p2-p3)
    return [dp1dt,dp2dt,dp3dt]

# initial condition
z0 = [0.1,0.1,1]

# time points
t = np.linspace(0,10)

# solve ODEs
a=0.5
b=0.5

z = odeint(model,z0,t,args=(a,b))

# plot results
plt.plot(t, z[:,0],'r-',linewidth=2, label='p1');
plt.plot(t, z[:,1],'b--',linewidth=2, label='p2');
plt.plot(t, z[:,2],'g:',linewidth=2, label='p3');
plt.legend();
plt.xlabel('time')
plt.ylabel('Population')
plt.show()
```




In each situation, two of the three Si's will have the same increasing behaviour, while the 3rd one is decreasing. In all cases, as t approaches infinity, all species will approach equilibrium. It appears that that they will approach a stable population of 0.5.

**Question 3**:

a. **SIR Model**
We can visualize the model as something like:
$S \rightarrow I \rightarrow R$
Since recoveries is dependent on the survival of infected individuals, then [change in R] $\alpha$ [change in I] so we have
$\frac{\delta R}{\delta t}$=qI, where q is a parameter.
Now recall that in the predator-prey model, where x is prey and y is predator, $\frac{\delta x}{\delta t}$=$ax - dxy$, where a is the growth rate of x and d is the death rate of x.

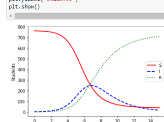
We can treat I as predator and S as prey. Of course, there will be no growth for S since it will either decrease or remain the same as people get infected. Also, there is N population. Therefore, we can say that $\frac{\delta S}{\delta t}$=$-\frac{pSI}{N}$, where p is a parameter.
Using the predator-prey model again, we have that $\frac{\delta y}{\delta t}$=$\beta xy - ky$, where $\beta$ is the growth rate for predator and k is the death rate. Treating I as "predator" and S as "prey", we have that $\frac{\delta I}{\delta t}$=$\frac{pSI}{N}$-$qI$. Think of q as the rate in which the infected decrease (i.e., recover), and p as the rate in which the infected increase (i.e., S becomes infected).

$\frac{\delta S}{\delta t}$=$-\frac{pSI}{N}$
$\frac{\delta I}{\delta t}$=$\frac{pSI}{N}$-$qI$
$\frac{\delta R}{\delta t}$=qI
N=763
S(0)=N-1=762
I(0)=1 (patient zero required to start an outbreak).
R(0)=0 (no one has recovered yet because 762 people have not been infected and there is 1 active case).

b. **Parameters Estimates**

Since N is a known parameter in this case, we'll have to estimate p and q. After testing several values, I chose p= and q= since the model looked like it made sense visually (the peak of I is around Day 7 and is around Infected Student=285). Also, I is near 0 at Day 15. The shape of I matches what an active cases model should look like, increasing until it reaches a certain point, then decreasing as recoveries outnumber infections and as epidemiological controls slows down the decrease of susceptible population.

```
In [41]: import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# function that returns dy/dt
def model(z,t,p,q,N):
    S,I,R=z
    dSdt =-p*I*S/N
    dIdt =p*I*S/N-q*I
    dRdt =q*I
    return [dSdt,dIdt,dRdt]

# initial condition
# I0 is 1 because patient 0 is required to simulate a disease outbreak, and since there are no recovered cases at the time, R0=0.
# Therefore, S0=N-1=762
z0 = [762,1,0]

# time points
t = np.linspace(0,15)

# solve ODEs

N=763
p=1.6
q=0.5

z = odeint(model,z0,t,args=(p,q,N))

# plot results
plt.plot(t, z[:, 0],'r-',linewidth=2, label='S');
plt.plot(t, z[:, 1],'b--',linewidth=2, label='I');
plt.plot(t, z[:,-1],'g:',linewidth=2, label='R');
plt.legend();
plt.xlabel('time')
plt.ylabel('Students')
plt.show()
```




c. **Goodness of Fit**

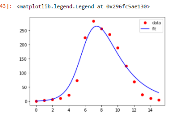
$R^2$ is a goodness of fit measure that can be used because it's a metric that measures how well the data fits the model. Mathematically it's denoted by $1 - \frac{SSResiduals}{SSTotal}$.

```
In [42]: import pandas as pd
         import numpy as np
         from scipy.optimize import curve_fit
         from scipy.integrate import odeint
         import matplotlib.pyplot as plt
         from sklearn.metrics import r2_score

         df = pd.read_csv("C:/Users/francali/Downloads/Britishfludata.csv")
         ddata =df['Day'].values
         sdata=df['Infected Students'].values
         df.head()
```

Out[42]:

| | Day | Infected Students |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 4 |
| 2 | 2 | 7 |
| 3 | 3 | 11 |
| 4 | 4 | 22 |

```
In [43]: def curvefit_model(t,p,q,N):
             'Function that returns estimated values of T computed from an ODE'
             def model(y,t):
                 S,I,R=y
                 dSdt =-p*I*S/N
                 dIdt =p*I*S/N-q*I
                 dRdt =q*I
                 return [dSdt,dIdt,dRdt]
             y0=[762,1,0]
             y= odeint(model,y0,t)
             return y[:, 1]
         # y[:,1] represents I, the function of interest, since we're fitting the infected students data
         p0=[1.6,  0.5, 763]
         # p0=[p,q,N]
         popt, pcov  = curve_fit(curvefit_model, ddata, sdata, p0)

         #popt=Optimal values for the parameters so that the sum of the squared residuals of f(xdata, *popt) - ydata is minimized.
         #pcov=The estimated covariance of popt. The diagonals provide the variance of the parameter estimate.
         #To compute one standard deviation errors on the parameters use perr = np.sqrt(np.diag(pcov)).
         p,q,N=popt
         r2 = print(r2_score(sdata, curvefit_model(ddata, p,q,N)))
         dfit = np.linspace(0,15)
         fit = curvefit_model(dfit, p,q,N)
         plt.plot(ddata, sdata, 'ro', label='data')
         plt.plot(dfit, fit, 'b-', label='fit')
         plt.legend(loc='best')

         #The R^2 is shown below. It's very close to 1, meaning that the model is a good fit.

         0.94901576535607

Out[43]: <matplotlib.legend.Legend at 0x296fc5ae130>
```




d. **Sensitivity**

```
In [38]: np.amax(z[:,1])
         #To find the peak of I

         247.23290234966433

In [24]: # function that returns dy/dt
         def model(z,t,p,q,N):
             S,I,R=z
             dSdt =-p*I*S/N
             dIdt =p*I*S/N-q*I
             dRdt =q*I
             return [dSdt,dIdt,dRdt]

         # initial condition
         z0 = [762,1,0]

         # time points
         t = np.linspace(0,15)

         # solve ODEs

         N=763
         p=1.601
         q=0.5


         z1 = odeint(model,z0,t,args=(p,q,N))

         np.amax(z1[:,1])
         # To find the peak of I if we add an extremely small number to parameter p (say, 0.001)

Out[24]: 247.43758877859327

In [25]: # function that returns dy/dt
         def model(z,t,p,q,N):
             S,I,R=z
             dSdt =-p*I*S/N
             dIdt =p*I*S/N-q*I
             dRdt =q*I
             return [dSdt,dIdt,dRdt]

         # initial condition
         z0 = [762,1,0]

         # time points
         t = np.linspace(0,15)

         # solve ODEs

         N=763
         p=1.6
         q=0.501

         z2 = odeint(model,z0,t,args=(p,q,N))

         np.amax(z2[:,1])
         # To find the peak of I if we add an extremely small number to parameter q (say, 0.001)

Out[25]: 246.66053775164949
```

Sensitivity of peak with respect to p is

$$\frac{Peak(p+\Delta p)-Peak(p)}{\Delta p} * \frac{p}{Peak(p)} = \frac{Peak(1.6+0.001)-Peak(1.6)}{0.001} * \frac{1.6}{Peak(1.6)} = \frac{247.438-247.233}{0.001} * \frac{1.6}{247.233}$$

$=1.327$

Sensitivity of peak with respect to q is

$$\frac{Peak(q+\Delta q)-Peak(q)}{\Delta q} * \frac{q}{Peak(q)} = \frac{Peak(0.5+0.001)-Peak(0.5)}{0.001} * \frac{0.5}{Peak(0.5)} = \frac{246.661-247.233}{0.001} * \frac{0.5}{247.233}$$

$=-1.157$

Both p and q have the same but opposite effect. This was confirmed when I was testing different parameters. Increasing one of the parameters moved the peak down, but increasing the other moved the peak right.