MA4080 - Computational Task 2

by
Tim Meiwald

November 11, 2019

Contents

1 Classic Endemic Model

1.1 Theory and Analysis

Classic Endemic Model Equations

$$\frac{ds}{dt} = -\beta is + \mu - \mu s,\ s(0) = s_0 \geq 0 \tag{1}$$

$$\frac{di}{dt} = \beta is - (\gamma + \mu)i,\ i(0) = i_0 \geq 0 \tag{2}$$

Where β is the contact rate of each person in a population.
μ is a factor that affects both births and deaths depending on what it's multiplied by. Where the mean lifetime is $\frac{1}{\mu}$
γ is the mean infections decay rate. Where $\frac{1}{\gamma}$ is the average infectious period.
s is the susceptible fraction of the population to an infection.
i is the infective fraction of the population.

As we can see in Figures 1,2,3,4,5,6 and 7, the Classic Endemic model is insensitive to the initial conditions since each set of graphs all arrive at the same fraction of the susceptible population, and infective population, i. Though they obviously start at different fractions of s and i. The change in β values merely affects the final value of the endemic equilibrium, $s(\infty) = \frac{1}{\sigma}$ where $\sigma = \frac{\beta}{\gamma+\mu}$. The graphs spiral into the endemic equilibrium because at first the infective fraction increases while the susceptible fraction decreases due to infection then as the infected fraction becomes too large, the speed of which depends on the contact rate β. The infected die off or recover and then die from old age while the susceptible fraction increases due to new births/immigration(if a local model) etc. At which point the infected fraction increases again due to the relatively larger amount of susceptible population. This pattern continues with ever smaller swings between the susceptible and infective fraction until the endemic equilibrium is reached. This happens because unlike Figure 5 in Hethcote 2000, the value for σ remains above 1 at all times with our values of $\gamma = 0.3,\ \mu = 0.016,\ \beta \in [0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]$. In Figure 6 where Hethcote 2000 the σ value is specifically chosen to be larger than 3 in order to show the spiral into the endemic equilibrium it looks similar to Figures 1,2,3,4,5,6 and 7. Though it does not look identical since our graphs were required to be in the (i,s) format while Figure 6 in Hethcote 2000 is in the (s,i) format.

1.2 Graphs

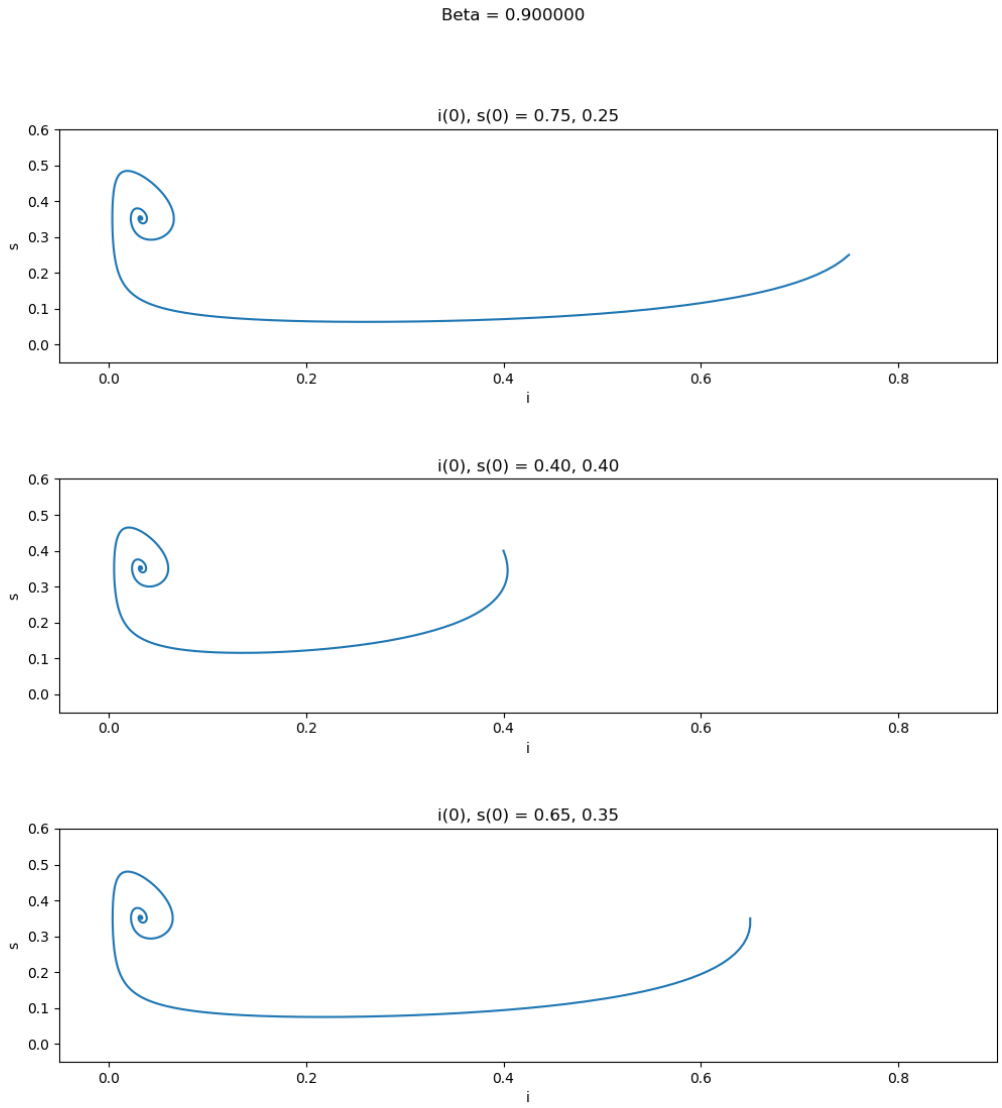

Figure 1

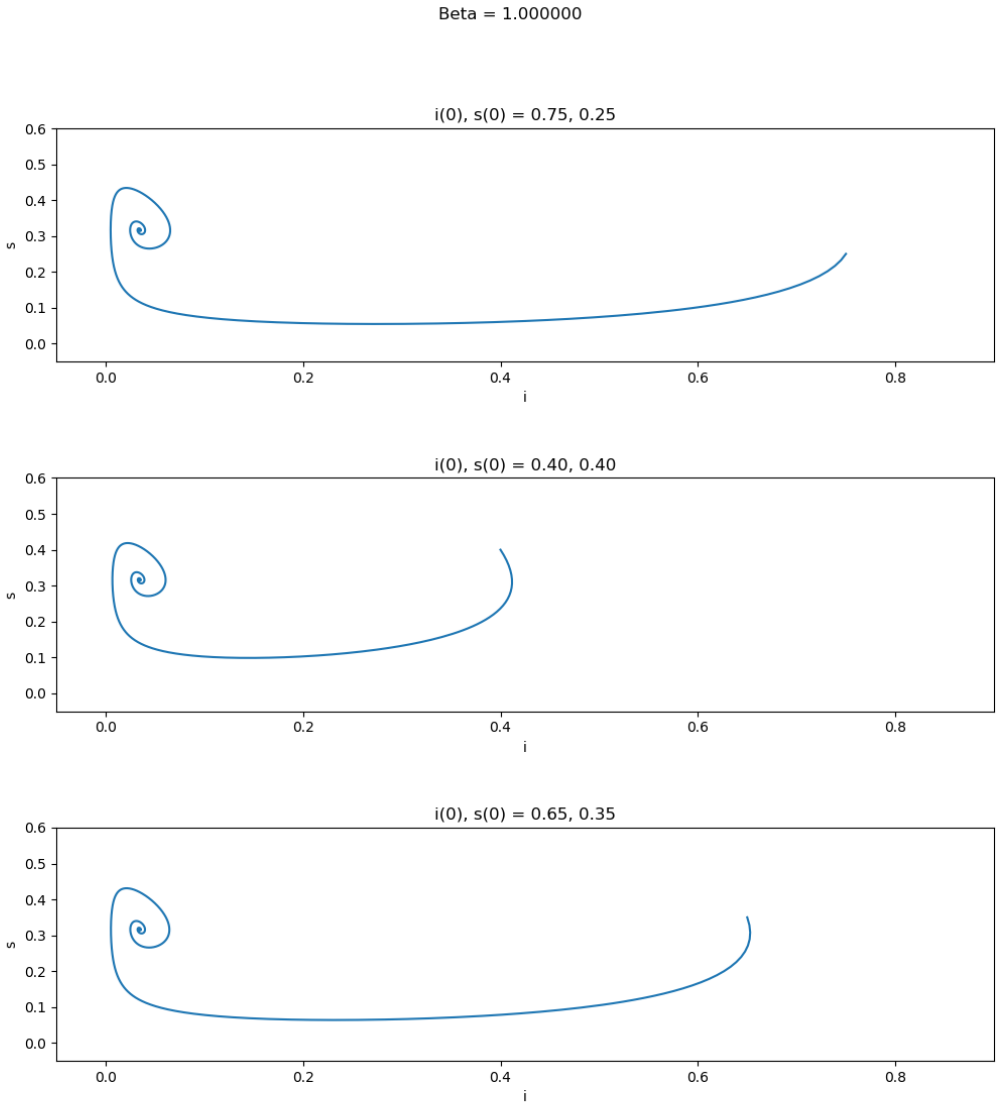
Beta = 1.000000
i(0), s(0) = 0.75, 0.25
i(0), s(0) = 0.40, 0.40
i(0), s(0) = 0.65, 0.35
s
i
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.8

Figure 2

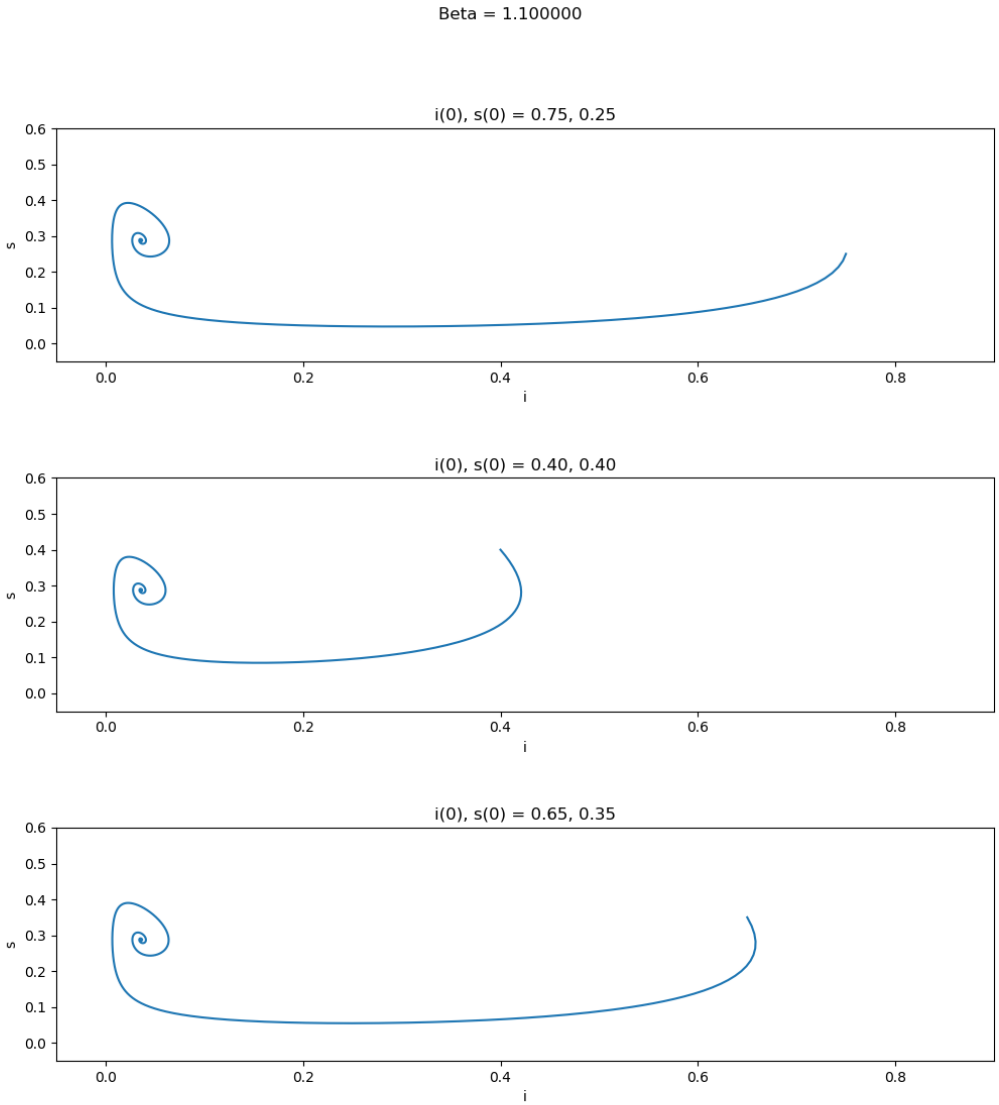
Beta = 1.100000
i(0), s(0) = 0.75, 0.25
i(0), s(0) = 0.40, 0.40
i(0), s(0) = 0.65, 0.35
s
i
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.8

Figure 3

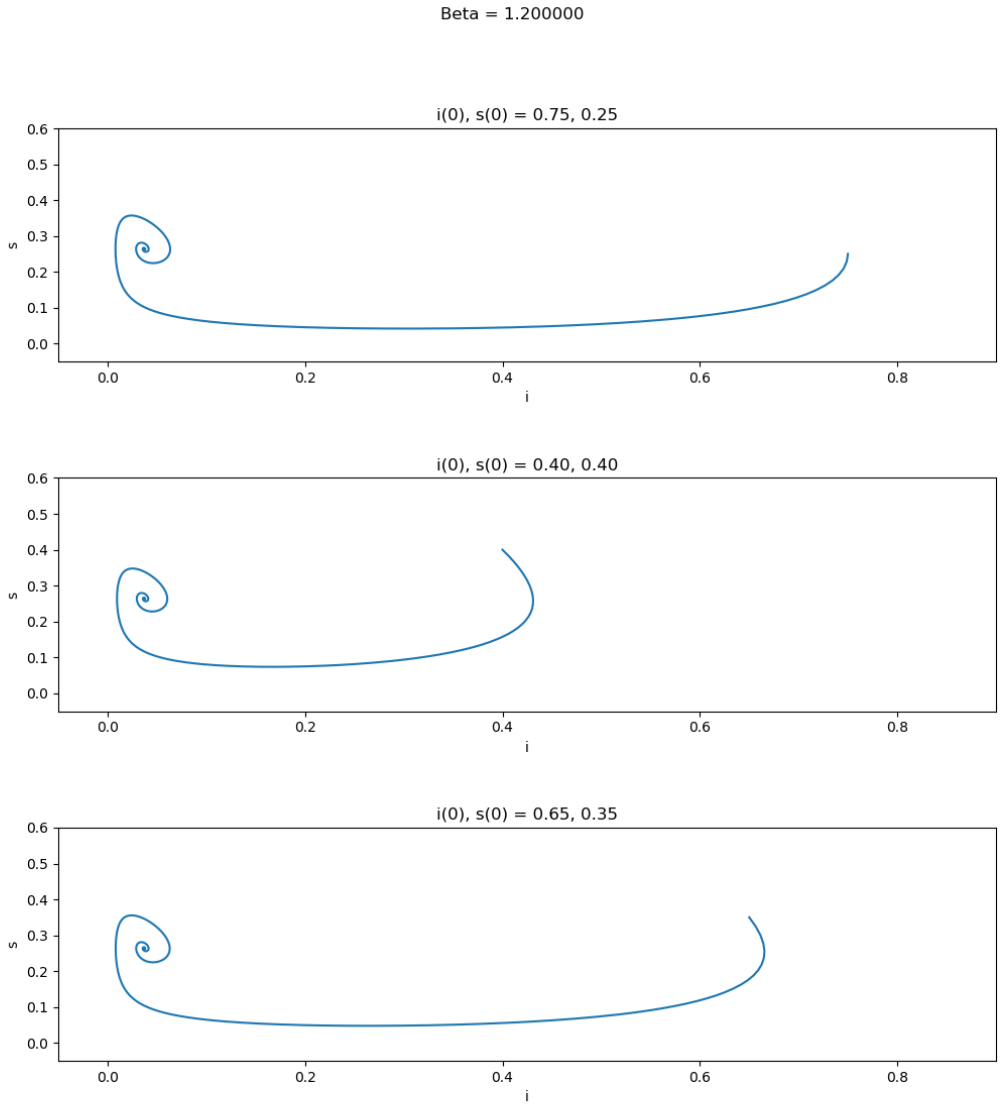
Beta = 1.200000
i(0), s(0) = 0.75, 0.25
i(0), s(0) = 0.40, 0.40
i(0), s(0) = 0.65, 0.35
s
i
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.8

Figure 4

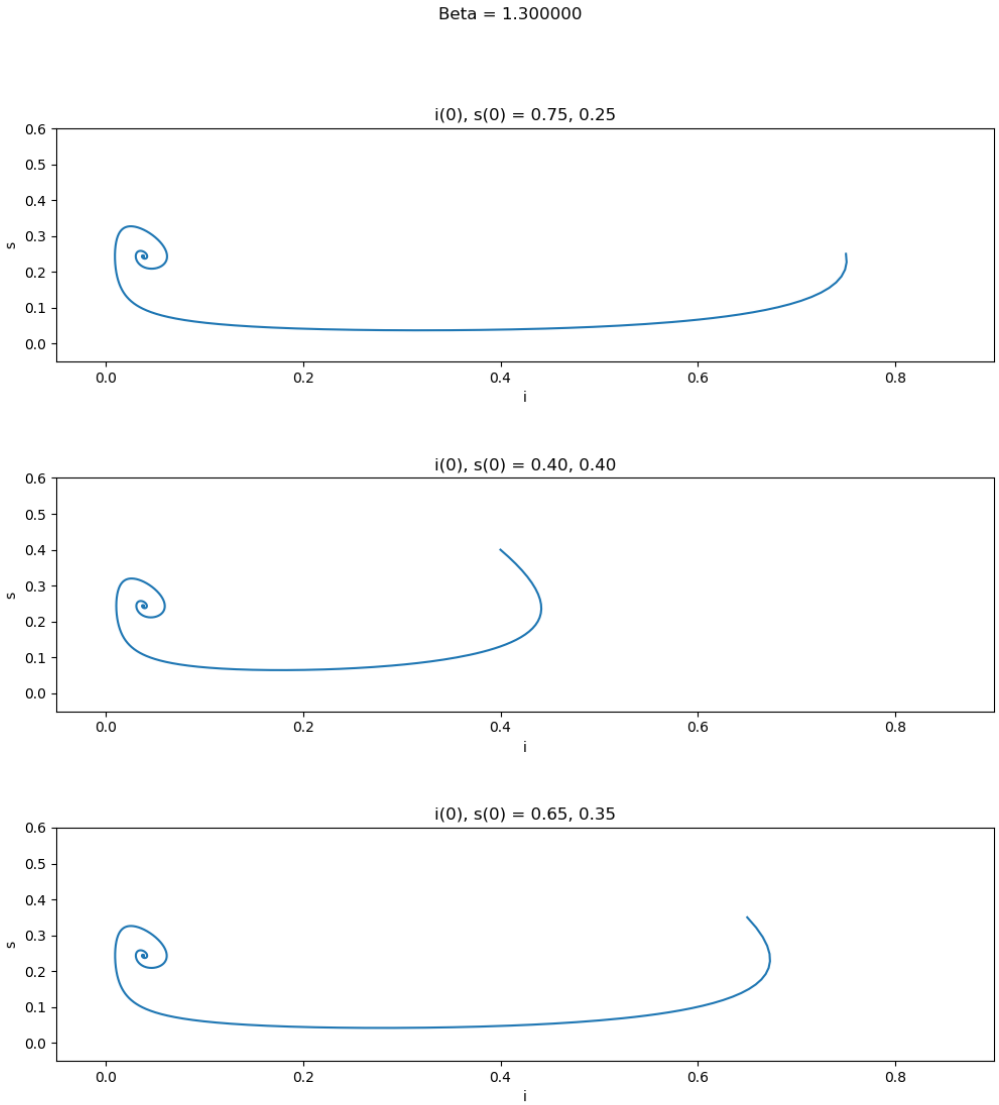
Beta = 1.300000
i(0), s(0) = 0.75, 0.25
i(0), s(0) = 0.40, 0.40
i(0), s(0) = 0.65, 0.35
s
i
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.8

Figure 5

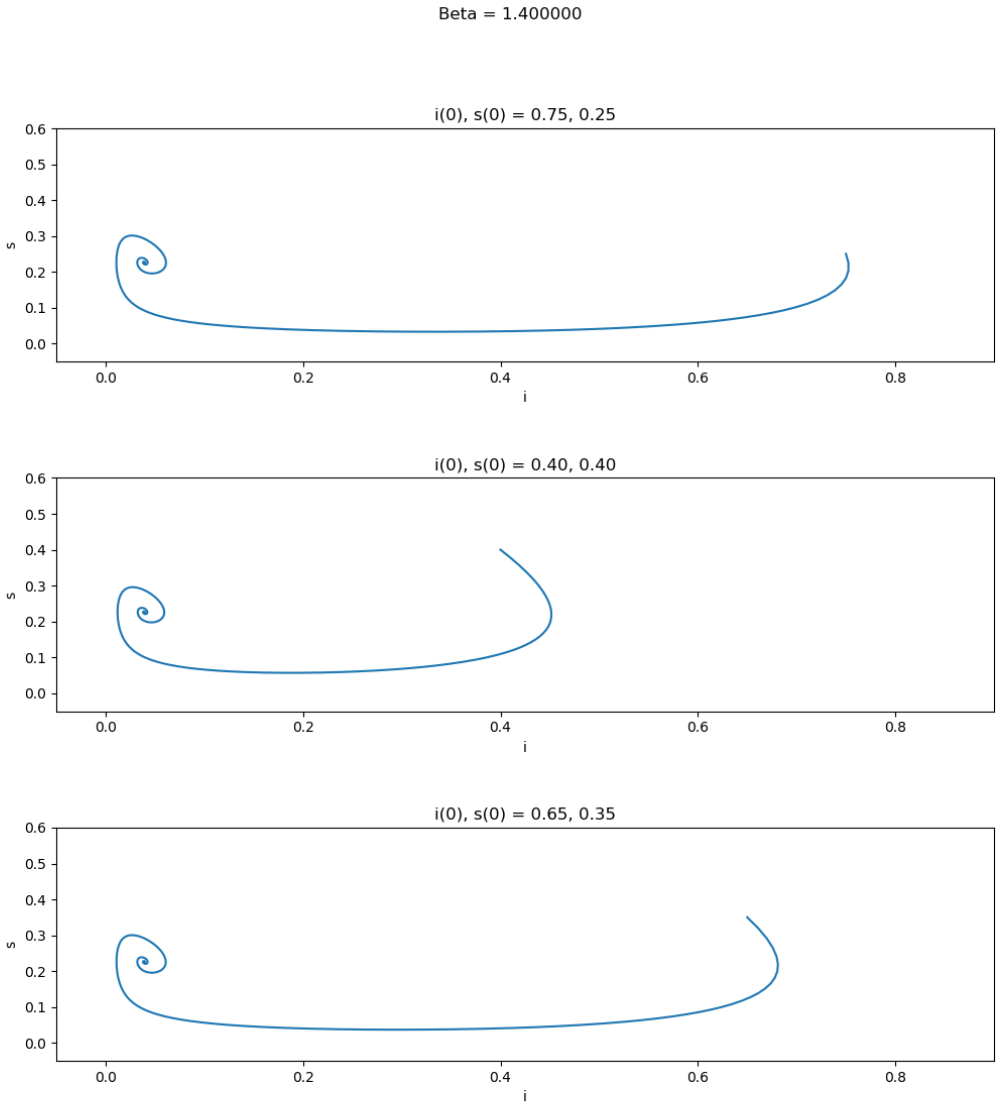
Beta = 1.400000
i(0), s(0) = 0.75, 0.25
i(0), s(0) = 0.40, 0.40
i(0), s(0) = 0.65, 0.35
s
i
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.8

Figure 6

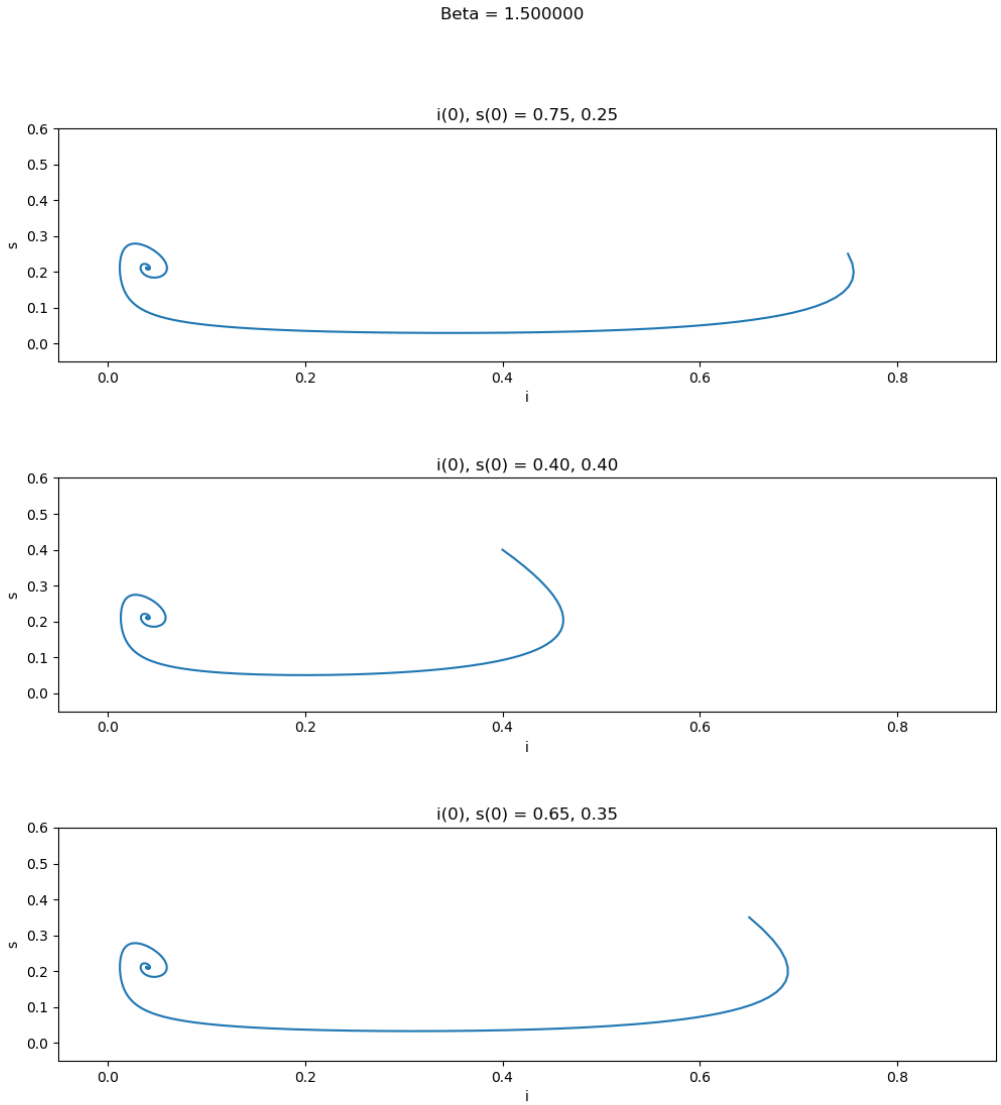
Beta = 1.500000
i(0), s(0) = 0.75, 0.25
i(0), s(0) = 0.40, 0.40
i(0), s(0) = 0.65, 0.35
s
i
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.8

Figure 7

1.3 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  9 17:35:42 2019

@author: Tim Meiwald
"""

import numpy as np
import matplotlib.pyplot as plt
#Three initial Conditions


s0 = np.array([0.35,0.25,0.4],dtype = "float64")
i0 = np.array([0.65,0.75,0.4],dtype = "float64")

#Constants
gamma = 0.3
mu = 0.016
Beta = np.arange(0.9,1.7,0.1)
#time step Size
t_step = 0.1
#Number of steps
N_step = 2500


#Classic Endemic Model
#Function 1
#ds/dt = -Beta*i*s + mu - mu*s
#Function 2
#di/dt = Beta*i*s - (gamma + mu)*i

def CEM(s0,i0,gamma,Beta,mu,t_step,N_step):
    #Classic Endemic Model
    ds_dt_Results = np.ndarray(N_step,dtype = "float64")
    di_dt_Results = np.ndarray(N_step,dtype = "float64")
    ds_dt_Results[0] = s0
    di_dt_Results[0] = i0
    for k in np.arange(0,N_step-1,1):
        #Assigning to s and i for readability
        s = ds_dt_Results[k]
        i = di_dt_Results[k]

        #Classic Endemic Model Functions
        ds_dt = -Beta*i*s + mu - mu*s
        di_dt = Beta*i*s - (gamma + mu)*i
        #Update the s and I values by adding the derivative multipled by the time
        #step, dt = 1 so not shown in code
        #Euler method! Runge_Kutta would give better, more stable results
        s = s + ds_dt*t_step
        i = i + di_dt*t_step

        #Add to the Results
        ds_dt_Results[k+1] = s
        di_dt_Results[k+1] = i

    return ds_dt_Results, di_dt_Results

```

```
58
59 for i in np.arange(0,8,1):
60     plt.figure(figsize = (12.0,12.0))
61     plt.suptitle("Beta = %f" % (Beta[i]))
62     plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
63     for j in np.arange(0,3,1):
64         x = "31%d" % j
65         plt.subplot(x)
66         T = CEM(s0[j],i0[j],gamma,Beta[i],mu,t_step,N_step)
67         plt.plot(T[1],T[0])
68         plt.title("i(0), s(0) = %.2f, %.2f" % (i0[j],s0[j]))
69         plt.xlabel("i")
70         plt.ylabel("s")
71         plt.ylim((-0.05,0.6))
72         plt.xlim((-0.05,0.9))
73     plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical
    Modelling/MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task1_%d.png" %
     i)
```

2 Mass action Law

2.1 Theory and Analysis

Mass action law equations

$$\gamma_{\rho i} = \beta_{\rho i} - \alpha_{\rho i} \tag{3}$$

$$r_\rho = k_\rho \sum_{i=1}^{n} c^{\alpha_{\rho i}} \tag{4}$$

$$\frac{d\vec{c}}{dt} = \sum_{\rho} \gamma_\rho r_\rho \tag{5}$$

Reactions

$$2\,\mathrm{Z} \rightleftharpoons 2\,\mathrm{X} \tag{6}$$

$$\mathrm{Z} \rightleftharpoons \mathrm{Y} \tag{7}$$

$$\mathrm{X} + \mathrm{Y} \longrightarrow 2\,\mathrm{Z} \tag{8}$$

$$\mathrm{Z} \rightleftharpoons \mathrm{Q} \tag{9}$$

Using Equations (3),(4) and (5) and the knowledge that $x + y + q + z = 1$ we can find that the MAL equations for x,y and q are

MAL Equations for above reactions

$$\frac{dX}{dt} = 2k_1 Z^2 - 2k_{-1} X^2 - k_3 XY \tag{10}$$

$$\frac{dY}{dt} = k_2 Z - k_{-2} Y - k_3 XY \tag{11}$$

$$\frac{dQ}{dt} = k_4 Z - k_{-4} Q \tag{12}$$

$$\frac{dZ}{dt} = -2k_1 Z^2 + 2k_{-1} X^2 - k_2 Z + k_{-2} Y + 2k_3 XY - k_4 Z + k_{-4} Q \tag{13}$$

We can then use the above equations with the naive numerical method of the Euler method to numerically plot out the behaviour of $x(t), y(t), q(t)$

2.2 Graphs

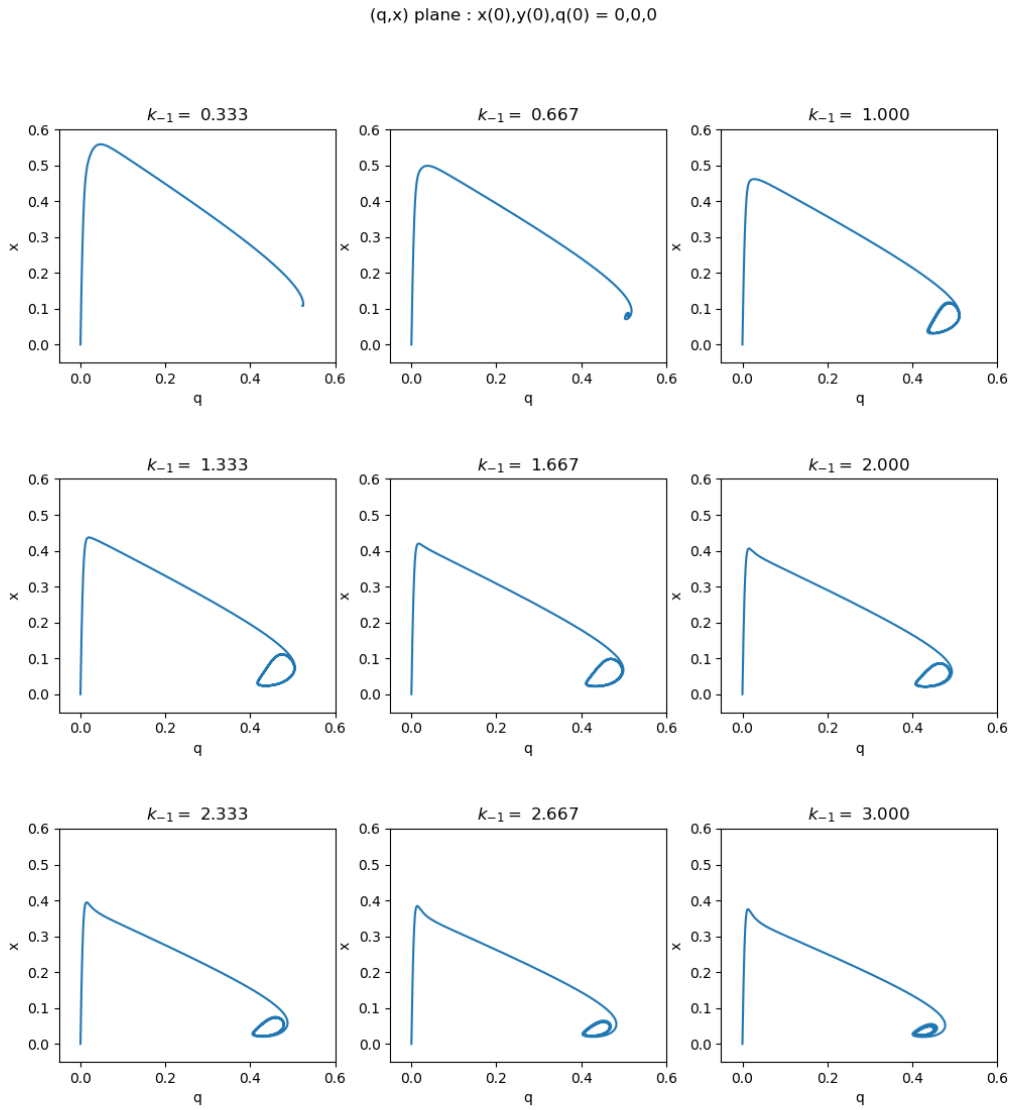

Figure 8: For the initial conditions where $X(0), Y(0), Z(0) = 0, 0, 0$ plotting the X values against Q. As we can see in the plots for small k_{-1} X rapidly increases then declines as Q increases while for larger k_{-1} we can see that a periodic function is set up where the values for X and Q periodically repeat because of the oval we can see in this phase diagram. Also at larger k_{-1}, X initially in the transient phase does not increase as much as for smaller k_{-1}, which makes sense since larger k_{-1} implies a faster reaction from 2X to 2Z, which then induces a faster increase in Q since it's concentration only depends on Z.

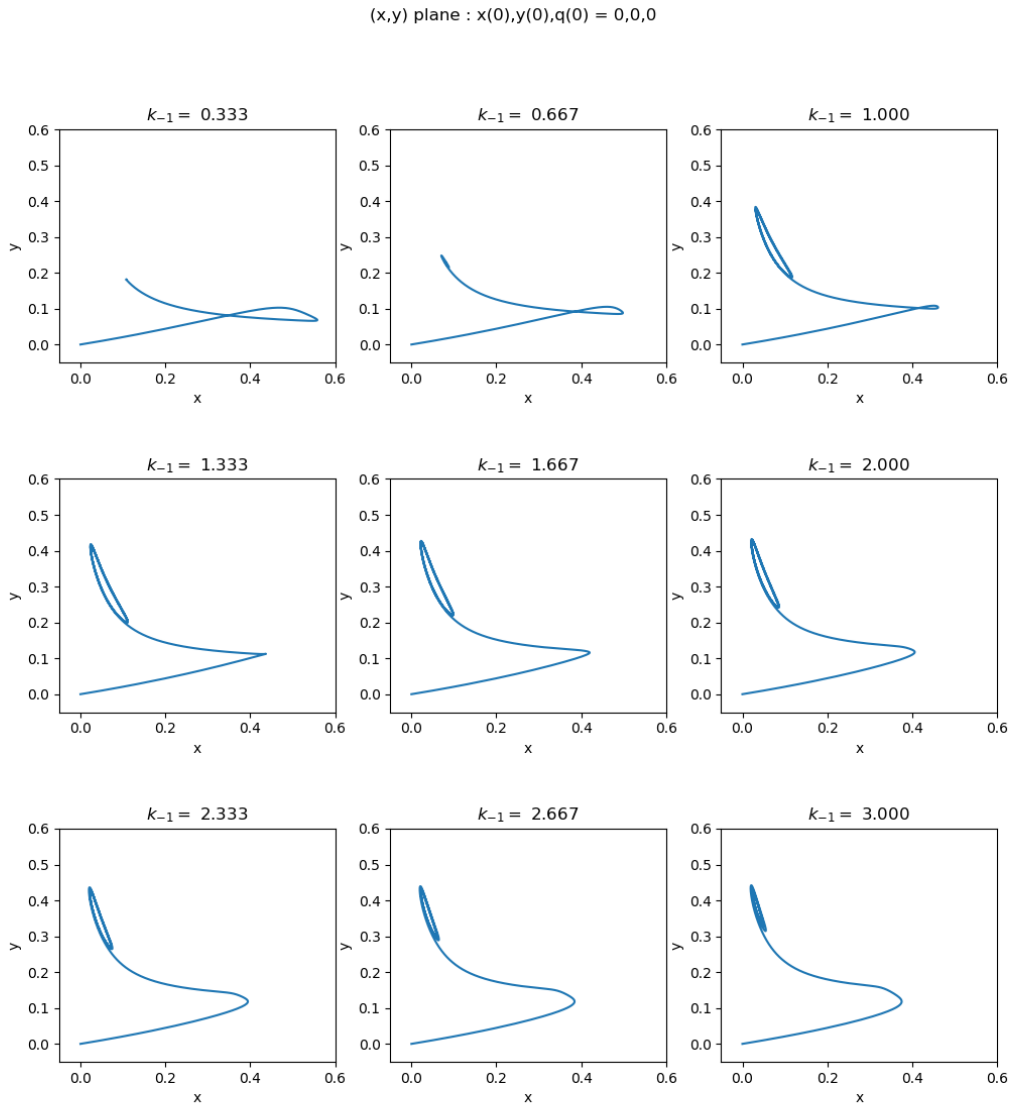

Figure 9: For the initial conditions where $X(0), Y(0), Z(0) = 0, 0, 0$ plotting the Y values against X. The behaviour is more complicated in this graph. Since first X increases a lot with Y increasing only a little and then Y increases rapidly while X decreases rapidly. Again for smaller k_{-1} the graph simply terminates at some point while for larger k_{-1} we can see that the X and Y values become periodic as they move around the oval section. From this we can see that for larger k_{-1} and Figure 8 that the values for X, Y, Q would start at the initial state and then move from a transient phase with less predictable behaviour into a steady state phase with very predictable concentration levels.

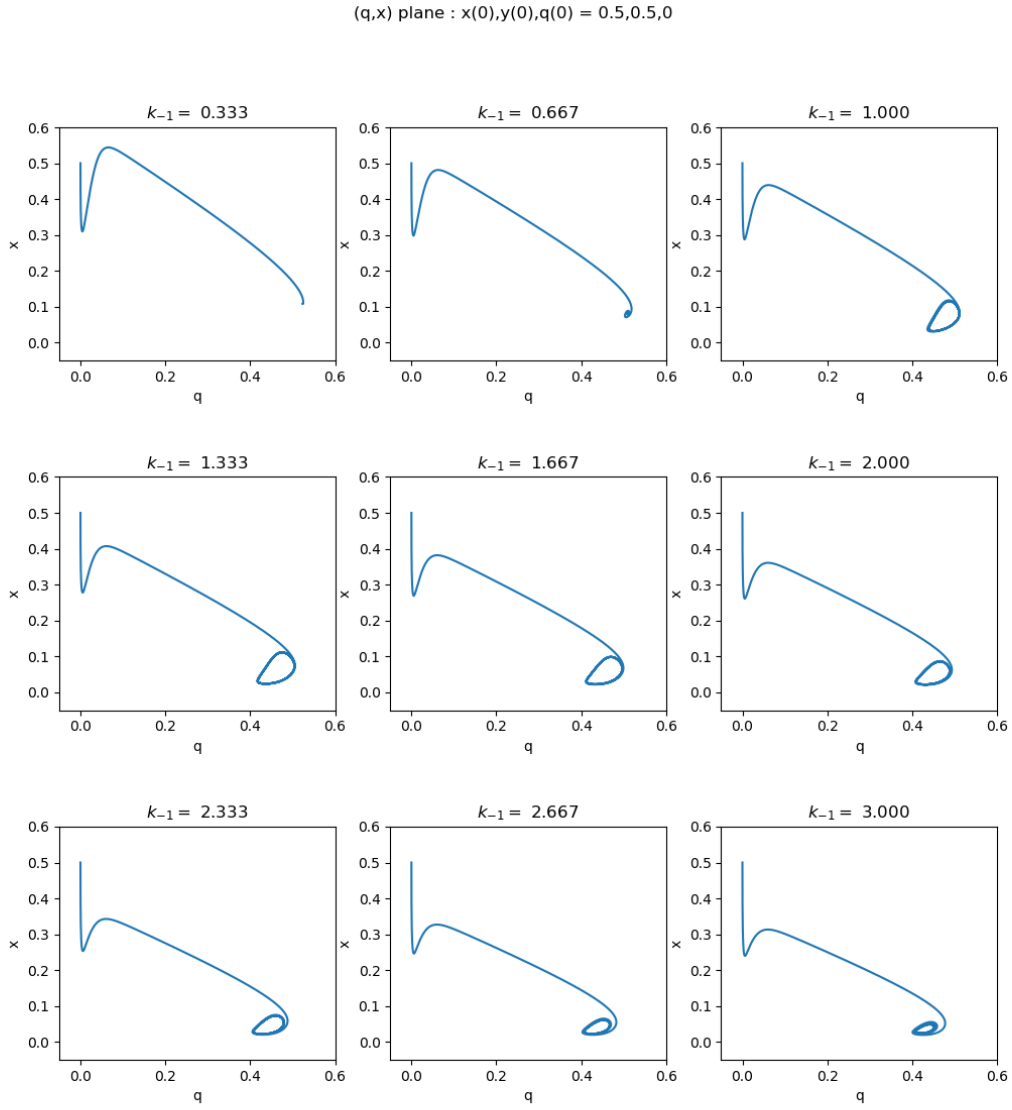

Figure 10: For the initial conditions where $X(0), Y(0), Z(0) = 0.5, 0.5, 0$ plotting the X values against Q. As we can see when we compare these graphs to Figure 8, the initial conditions affect the behaviour of the concentration levels in the initial transient phase drastically however both end up in the same region of the phase diagram in the same oval shape. Implying that their steady state behaviour is largely from what can be easily seen insensitive to the initial conditions and only depends on the functions themselves.

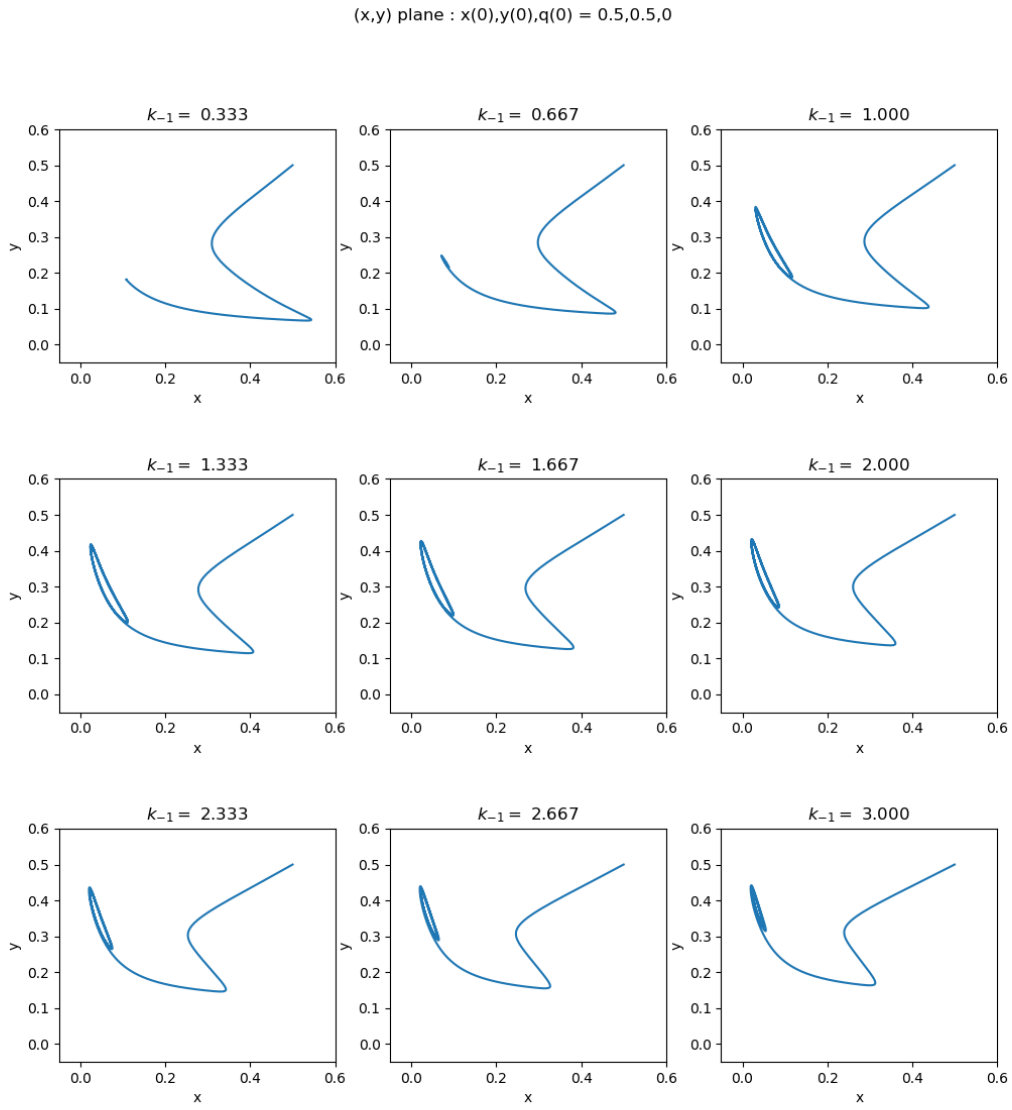

Figure 11: For the initial conditions where $X(0), Y(0), Z(0) = 0.5, 0.5, 0$ plotting the Y values against X. As we can see when we compare these graphs to Figure 9, the initial conditions affect the behaviour of the concentration levels in the initial transient phase drastically however both end up in the same region of the phase diagram in the same oval shape. Implying that their steady state behaviour is largely from what can be easily seen insensitive to the initial conditions and only depends on the functions themselves.

2.3 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  9 18:43:30 2019

@author: Tim Meiwald
"""
import numpy as np
import matplotlib.pyplot as plt
#z + x + y + q = 1 = const




k = np.array([2.5,1,8.5,0.07],dtype = "float64")
k_neg = np.array([2,0.111,0,0.025],dtype = "float64")
# x,y,q,z, x+y+q+z = 1 = const
InitCond1 = np.array([0,0,0,1],dtype = "float64")
InitCond2 = np.array([0.5,0.5,0,0],dtype = "float64")

gamma1 = np.array([2,0,0,-2],dtype = "float64")
gamma2 = np.array([0,1,0,-1],dtype = "float64")
gamma3 = np.array([1,1,0,2],dtype = "float64")
gamma4 = np.array([0,0,1,-1],dtype = "float64")


t_step = 0.01
N_step = 60000
def MAL(t_step,N_step,InitCond,kNegInt):
    k1Neg = np.arange(0,10,1)/3
    k1Neg = k1Neg[kNegInt]
    k_neg[0] = k1Neg
    x = InitCond[0]
    y = InitCond[1]
    q = InitCond[2]
    z = InitCond[3]
    ResultArr = np.ndarray((N_step,5),dtype = "float64")
    ResultArr[0,:4] = InitCond
    ResultArr[0,4] = 0
    for i in np.arange(1,N_step,1):
        dx_dt = 2*k[0]*z**2 - 2*k_neg[0]*x**2 - k[2]*x*y
        dy_dt = k[1]*z - k_neg[1]*y - k[2]*x*y
        dq_dt = k[3]*z - k_neg[3]*q
        dz_dt = -2*k[0]*z**2 + 2*k_neg[0]*x**2 - k[1]*z + k_neg[1]*y + 2*k[2]*x*y - k
    [3]*z + k_neg[3]*q
        #print(dx_dt,dy_dt,dq_dt,dz_dt)
        x = x + t_step*dx_dt
        y = y + t_step*dy_dt
        q = q + t_step*dq_dt
        z = z + t_step*dz_dt

        ResultArr[i,0] = x
        ResultArr[i,1] = y
        ResultArr[i,2] = q
        ResultArr[i,3] = z
        ResultArr[i,4] = i*t_step
        #print(x+y+q+z)
        #print(x,y,q,z)
```

```
    return ResultArr

plt.figure(figsize = (12.0,12.0))
plt.suptitle("(q,x) plane : x(0),y(0),q(0) = 0,0,0")
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
for i in np.arange(0,10,1):
    Res = MAL(t_step,N_step,InitCond1,i)
    x = "33%d" % i
    plt.subplot(x)
    k1NegVal = np.arange(0,10,1)/3
    plt.title(r"$k_{-1} =$ %.3f" % k1NegVal[i])
    plt.xlabel("q")
    plt.ylabel("x")
    plt.ylim((-0.05,0.6))
    plt.xlim((-0.05,0.6))
    plt.plot(Res[:,2],Res[:,0])
plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical Modelling/
    MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task2_1.png")

plt.figure(figsize = (12.0,12.0))
plt.suptitle("(x,y) plane : x(0),y(0),q(0) = 0,0,0")
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
for i in np.arange(0,10,1):
    Res = MAL(t_step,N_step,InitCond1,i)
    x = "33%d" % i
    plt.subplot(x)
    k1NegVal = np.arange(0,10,1)/3
    plt.title(r"$k_{-1} =$ %.3f" % k1NegVal[i])
    plt.xlabel("x")
    plt.ylabel("y")
    plt.ylim((-0.05,0.6))
    plt.xlim((-0.05,0.6))
    plt.plot(Res[:,0],Res[:,1])
plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical Modelling/
    MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task2_2.png")

plt.figure(figsize = (12.0,12.0))
plt.suptitle("(q,x) plane : x(0),y(0),q(0) = 0.5,0.5,0")
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
for i in np.arange(0,10,1):
    Res = MAL(t_step,N_step,InitCond2,i)
    x = "33%d" % i
    plt.subplot(x)
    k1NegVal = np.arange(0,10,1)/3
    plt.title(r"$k_{-1} =$ %.3f" % k1NegVal[i])
    plt.xlabel("q")
    plt.ylabel("x")
    plt.ylim((-0.05,0.6))
    plt.xlim((-0.05,0.6))
    plt.plot(Res[:,2],Res[:,0])
plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical Modelling/
    MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task2_3.png")

plt.figure(figsize = (12.0,12.0))
plt.suptitle("(x,y) plane : x(0),y(0),q(0) = 0.5,0.5,0")
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
for i in np.arange(0,10,1):
    Res = MAL(t_step,N_step,InitCond2,i)
    x = "33%d" % i
```

```
113     plt.subplot(x)
114     k1NegVal = np.arange(0,10,1)/3
115     plt.title(r"$k_{-1} =$ %.3f" % k1NegVal[i])
116     plt.xlabel("x")
117     plt.ylabel("y")
118     plt.ylim((-0.05,0.6))
119     plt.xlim((-0.05,0.6))
120     plt.plot(Res[:,0],Res[:,1])
121 plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical Modelling/
    MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task2_4.png")
```

3 Lotka-Volterra

3.1 Theory and Analysis

Lotka-Volterra Equations

$$\frac{dx}{dt} = \alpha x - \beta xy \tag{14}$$

$$\frac{dy}{dt} = \delta xy - \gamma y \tag{15}$$

Lotka-Volterra Equations with Logistic growth

$$\frac{dx}{dt} = \alpha x(1 - Kx) - \beta xy \tag{16}$$

$$\frac{dy}{dt} = \delta xy - \gamma y \tag{17}$$

3.2 Graphs

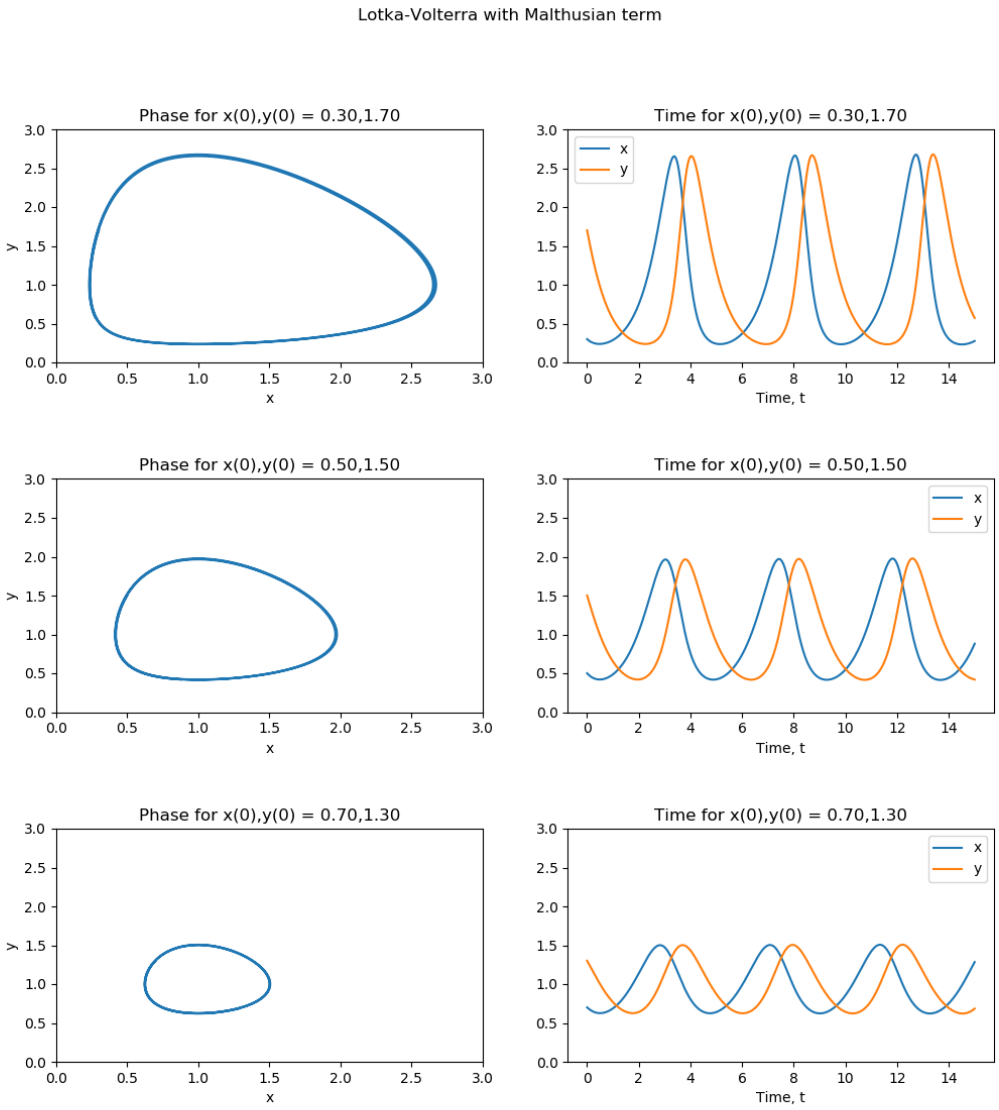

Figure 12: In this figure we can clearly see that the results with a Malthusian term are periodic but is sensitive to the initial conditions. Effectively, the initial conditions regulate the amplitude of the sinusoidal function. The values for x trail the values for y because originally the Lotka-Volterra equations represented the relationship between predator and prey populations. In the case of predator and prey relations as the number of prey increases there is a time lag until the number of predators increase despite sufficient food due to pregnancy taking time etc. As the predator numbers then increase the number of prey falls as they get eaten by the predator, at which point after some delay due to fat reserves on the predator etc, the predators start to starve to death and their population collapses. At which point prey increases due to the lack of predators. And thus it continues in a cycle. The Lotka-Volterra equations do however assume that the prey always has enough food etc. Effectively, the predator and prey populations function like an undamped SHM.

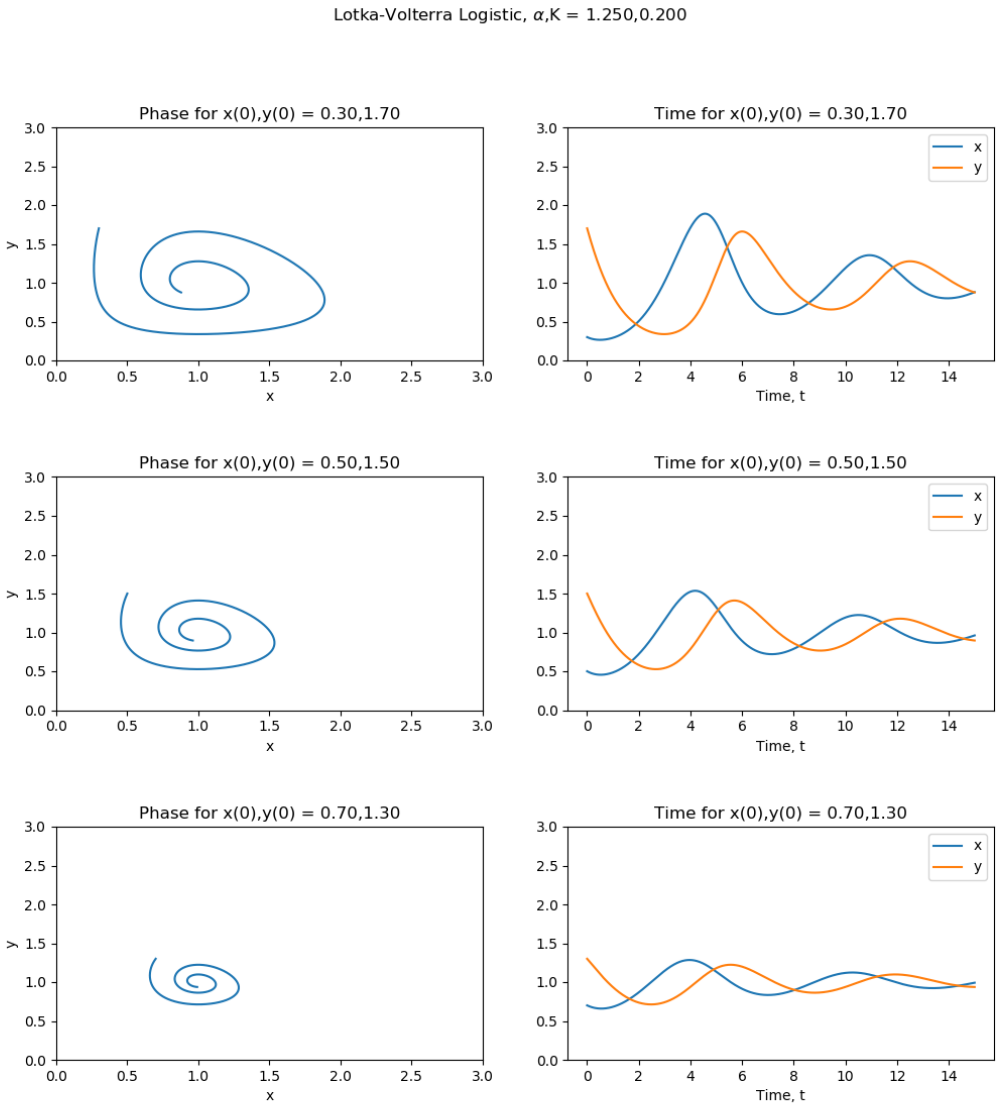

Figure 13: In this figure we added logistic growth with $\alpha, K = 1.25, 0.2$ where α is the growth rate and K is the carrying capacity. As we can see for these values of α, K the behaviour is still sinusoidal as in Figure 12 however the carrying capacity acts as a damping force meaning that as time goes on the difference between population highs and population lows reduces. This makes sense because every time the population of the prey species approaches the carrying capacity their growth slows and as their growth slows so does the prey species time lagged growth slow which in turn reduces the speed of population collapse of the prey due to not as high numbers of predators. However in this figure the periodicity does not stop it simply reduces in amplitude. i.e it functions as an underdamped SHM

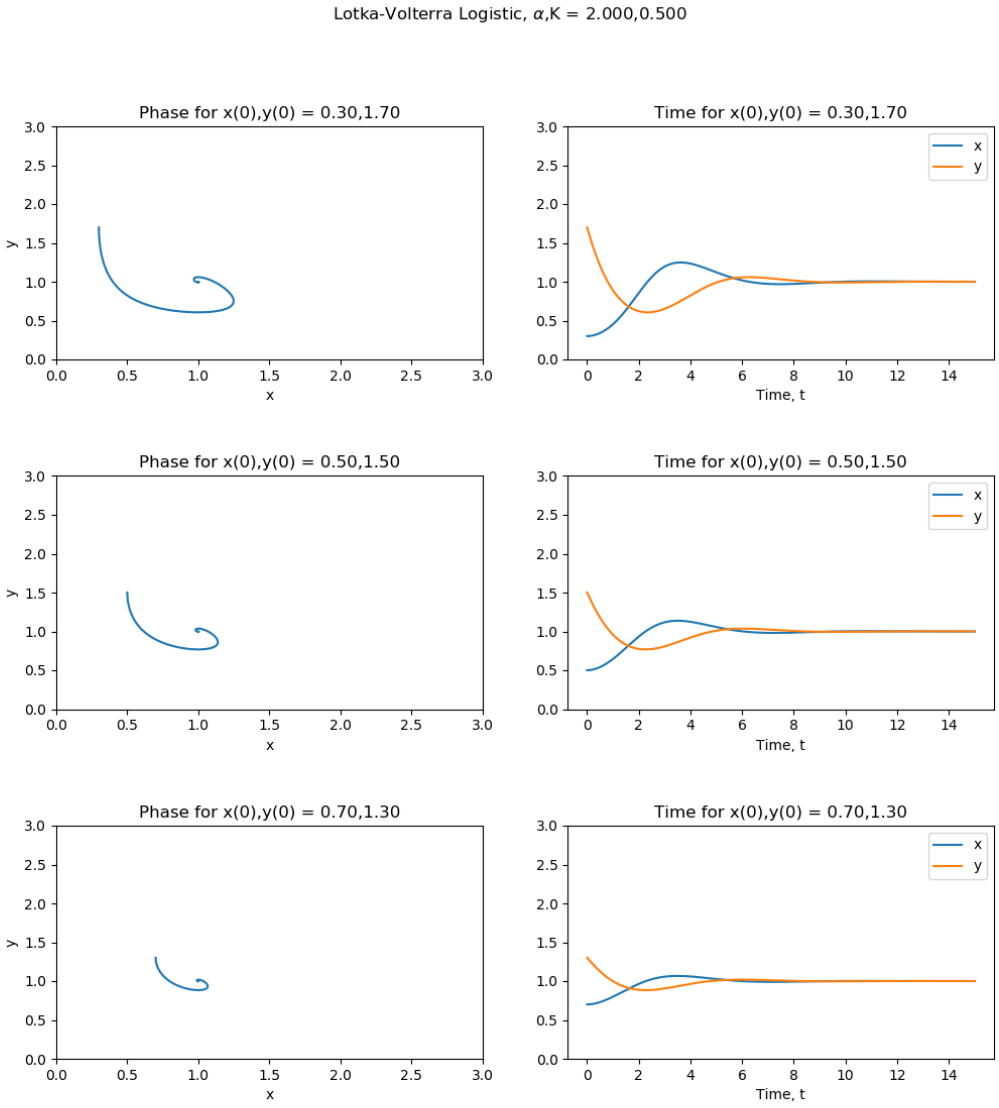

Figure 14: In this figure like in Figure 13 the prey and population levels are underdamped however the damping factor is larger than in Figure 13 so it approaches the equilibrium level faster.

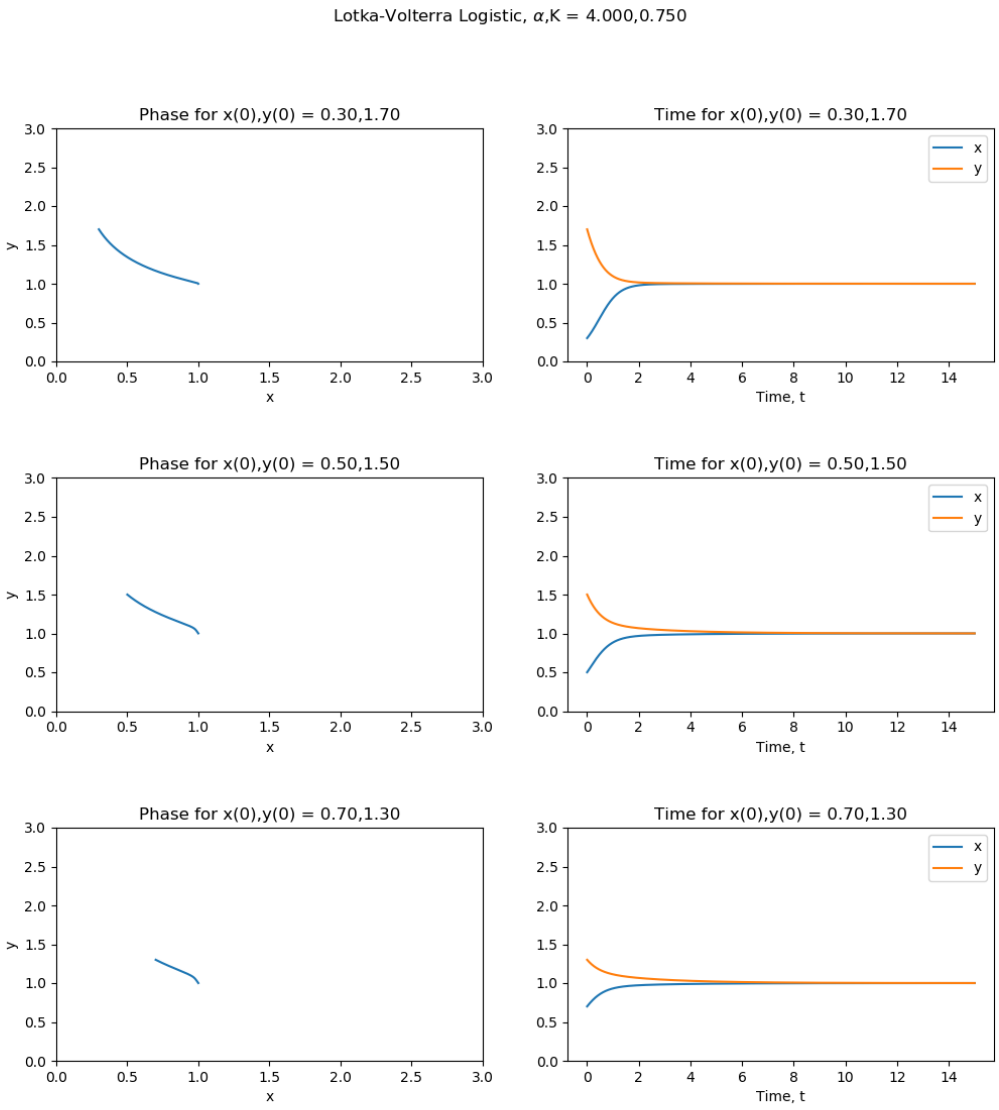

Figure 15: In this figure the prey and population levels are critically damped so each population relaxes to the equilbrium level quickly. So this figure functions as a critically damped SHM.

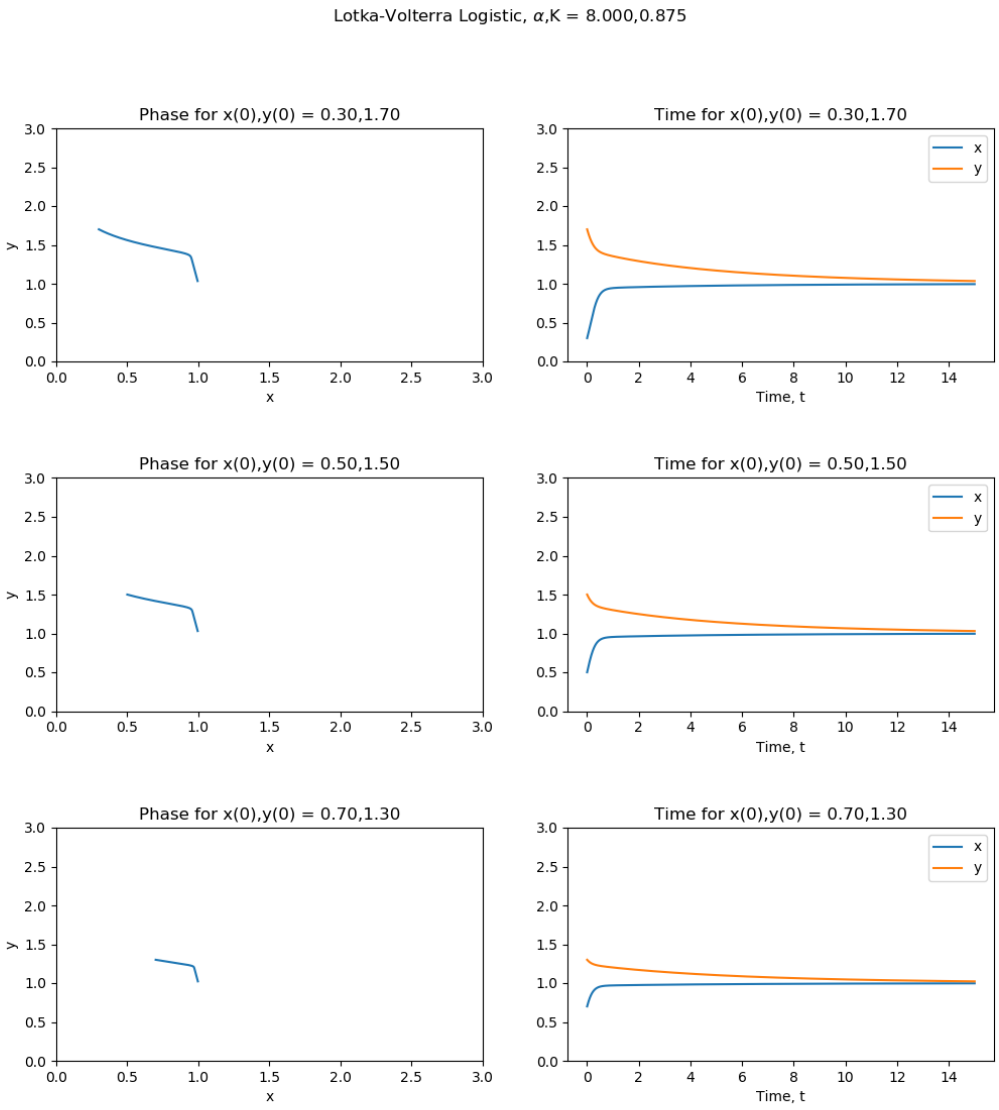

Figure 16: In this figure the prey and population levels are over damped so each population relaxes to the equilbrium level slowly. So this figure functions as an overdamped SHM.

In all of the above Logistic Growth Lotka-Volterra Graphs(Figures 14,15,16) we can see that the equilibrium value of the predator and prey species is insensitive to the initial conditions since it is always at x,y = 1. Overall the key takeaway from these plots is that firstly, ironically the predator population constantly chases the population level of the prey and that the carrying capacity i.e the environmental capability of supporting a certain amount of the prey acts as an important stabilizer of predator and prey population levels. And therefore the effects of changing the carrying capacity of the environment cause predictable population collapses or growths in both prey and predator species. Obviously, we could

extend the Lotka-Volterra equations to be more realistic by chaining the predator and prey equations together such that we have multiple levels of species with each predating on the lower level while being the prey of the upper level. For example, Krill both eat Plankton and are eaten by Shrimps which are eaten by larger animals and so forth.

3.3 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  9 23:25:11 2019

@author: Tim Meiwald
"""

import numpy as np
import matplotlib.pyplot as plt



alpha = 1.5
beta = 1.5
gamma = 1.5
delta = 1.5

x = np.array([0.3,0.5,0.7],dtype = "float64")
y = np.array([1.7,1.5,1.3],dtype = "float64")

t_step = 0.001
N_step = 15000

def LotkaVolterra(t_step,N_step,x,y):
    ResultArr = np.ndarray((N_step,3),dtype = "float64")
    ResultArr[0,0] = x
    ResultArr[0,1] = y
    for i in np.arange(0,N_step,1):
        ResultArr[i,2] = t_step*i
    for i in np.arange(0,N_step-1,1):
        x = ResultArr[i,0]
        y = ResultArr[i,1]
        dx_dt = alpha*x - beta*x*y
        dy_dt = delta*x*y - gamma*y

        ResultArr[i+1,0] = x + dx_dt*t_step
        ResultArr[i+1,1] = y + dy_dt*t_step
    return ResultArr


```

```
plt.figure(figsize = (12.0,12.0))
plt.suptitle("Lotka-Volterra with Malthusian term")
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
for i in np.arange(0,3,1):
    Res = LotkaVolterra(t_step,N_step,x[i],y[i])
    x1 = "32%d" % (1+(2*i))
    x2 = "32%d" % (1+(2*i + 1))
    plt.subplot(x1)
    plt.plot(Res[:,0],Res[:,1])
    plt.xlim((0,3))
    plt.ylim((0,3))
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(r"Phase for x(0),y(0) = %.2f,%.2f" % (x[i],y[i]) )
    plt.subplot(x2)
    plt.plot(Res[:,2],Res[:,0],label = "x")
    plt.plot(Res[:,2],Res[:,1],label = "y")
    plt.xlabel("Time, t")
    plt.ylim((0,3))
    plt.title(r"Time for x(0),y(0) = %.2f,%.2f" % (x[i],y[i]))
    plt.legend()
plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical Modelling/
    MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task3LV.png")

alpha = np.array([1.25,2,4,8],dtype = "float64")
K = np.array([0.2,0.5,0.75,0.875],dtype = "float64")


beta = 1.
gamma = 1.
delta = 1.


def LotkaVolterraLogistic(t_step,N_step,x,y,K,alpha):
    ResultArr = np.ndarray((N_step,3),dtype = "float64")
    ResultArr[0,0] = x
    ResultArr[0,1] = y
    for i in np.arange(0,N_step,1):
        ResultArr[i,2] = t_step*i
    for i in np.arange(0,N_step-1,1):
        x = ResultArr[i,0]
        y = ResultArr[i,1]
        dx_dt = alpha*x*(1-K*x) - beta*x*y
        dy_dt = delta*x*y - gamma*y

        ResultArr[i+1,0] = x + dx_dt*t_step
        ResultArr[i+1,1] = y + dy_dt*t_step
    return ResultArr

for j in np.arange(0,4,1):
    plt.figure(figsize = (12.0,12.0))
    plt.suptitle(r"Lotka-Volterra Logistic, $\alpha$,K = %.3f,%.3f" % (alpha[j],K[j]))
    plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
    for i in np.arange(0,3,1):
        Res = LotkaVolterraLogistic(t_step,N_step,x[i],y[i],K[j],alpha[j])
        x1 = "32%d" % (1+(2*i))
        x2 = "32%d" % (1+(2*i + 1))
        plt.subplot(x1)
        plt.plot(Res[:,0],Res[:,1])
```

```
99          plt.xlabel("x")
100         plt.ylabel("y")
101         plt.xlim((0,3))
102         plt.ylim((0,3))
103         plt.title(r"Phase for x(0),y(0) = %.2f,%.2f" % (x[i],y[i]) )
104         plt.subplot(x2)
105         plt.plot(Res[:,2],Res[:,0],label = "x")
106         plt.plot(Res[:,2],Res[:,1],label = "y")
107         plt.xlabel("Time, t")
108         plt.ylim((0,3))
109         plt.title(r"Time for x(0),y(0) = %.2f,%.2f" % (x[i],y[i]))
110         plt.legend()
111     plt.savefig("C:/Users/timme/Documents/M.Sc Applied Computation and Numerical
    Modelling/MA7080 - Mathematical Modelling/ComputationalTask2/Figures/Task3LLV%d.png"
     % j)
```

References

Hethcote, Herbert W. 2000. "The Mathematics of Infectious Diseases." *Society for Industrial and Applied Mathematics.*