Lotka-Voltera Equations and More

November 19, 2024

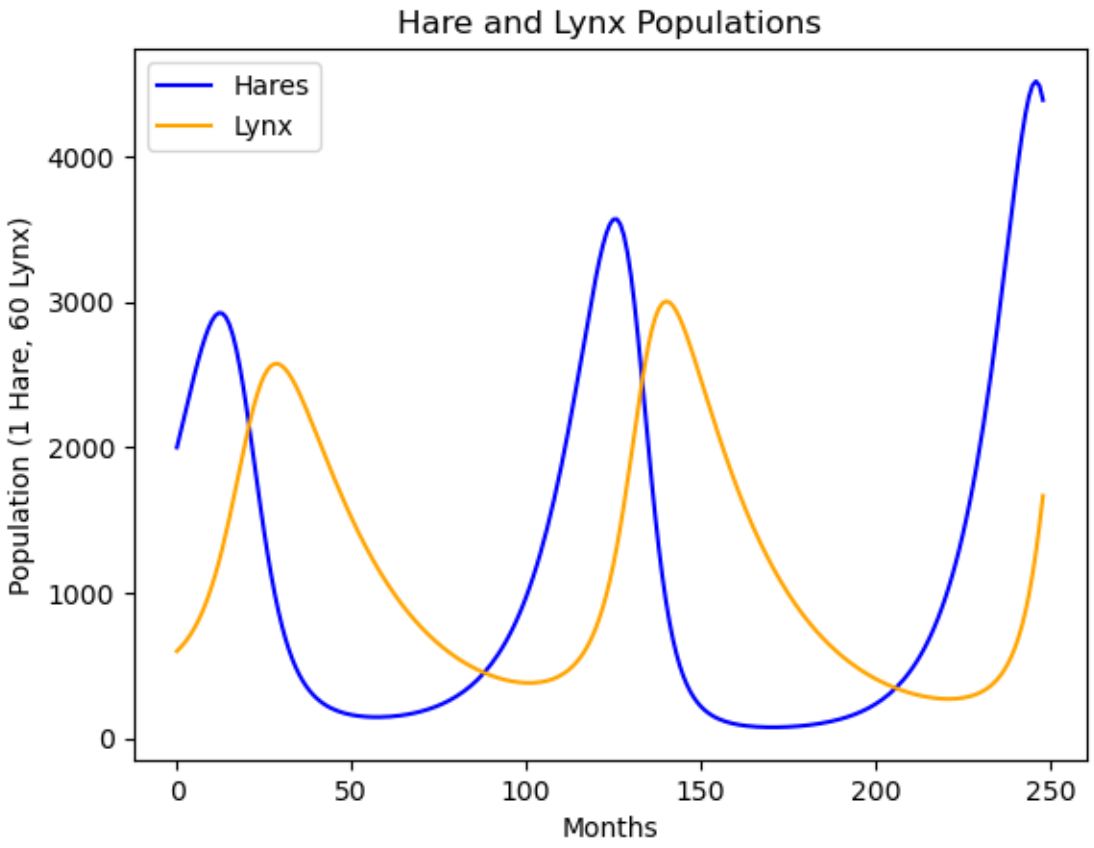
[1]:
```
import numpy as np
import numpy.linalg as npl
import math
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

Worked Example Read Calculus in Context, Chapter 4. This explores additional models that can be explored using this method. Copy and paste SIR plot here and modify to model the Lotka-Voerra Model from page 193 number 7. Explore some of the questions a - f on your own and add some short notes about this model. note: The scaling factor for lynx is 60 (page 194, part c).”plot H and 60L”

[2]:
```
def lynxHare(num_hare, num_lynx, t_final):
    # time variables
    t_initial = 0
    t = t_initial
    # Starting values at the beginning of model for hares and lynx (turns out␣
 ↪the plural of lynx is just lynx)
    H = num_hare
    L = num_lynx

    # Hares per month per hare
    a = .1
    # Hares per month per hare-lynx
    b = .005
    # Lynx per month per hare-lynx
    c = .00004
    # Lynx per month per lynx
    d = .04

    # Creating empty arrays to hold values we will plot
    H_graph = []
    L_graph = []

    # Change in time/number of steps
    delta_t = 1
    # Iterating through the number of line segments in our plot (the more␣
 ↪lines, the smoother the graph)
```

```
    for k in range (1, t_final):
        # Add current values to graph vectors (include Lynx scaling factor of␣
↪60)
        H_graph.append(H)
        L_graph.append(60*L)
        # Rate of change in Hare population
        Hprime = a*H - b*H*L
        # Rate of change in Lynx population
        Lprime = c*H*L - d*L

        # Actual change in Hare and Lynx populations
        delta_H = Hprime*delta_t
        delta_L = Lprime*delta_t

        #Updating the values of t, H, and L with our delta values
        t = t + delta_t
        H = H + delta_H
        L = L + delta_L

    # return the arrays with values over each step
    return H_graph, L_graph


# Plot the populations
H_line, L_line= lynxHare(2000, 10, 250)
plt.plot(H_line, color='blue', label='Hares')
plt.plot(L_line, color='orange', label='Lynx')
plt.legend(['Hares', 'Lynx'])
plt.xlabel('Months')
plt.ylabel('Population (1 Hare, 60 Lynx)')
plt.title('Hare and Lynx Populations')
```

[2]: Text(0.5, 1.0, 'Hare and Lynx Populations')



The graph above shows part C of number 7 on p. 193-194 of Calculus in Context, beginning with 2,000 hares and 10 lynx. It includes the basic assumptions of the two species model:

- In the absence of lynx, the hare population grows logistically - we see this in the equation for Hprime, where if the lynx population is 0 then the hare population is aH
- The population of hares declines in proportion to to H*L - see this in the second half of the equation for Hprime, -bHL. Both of these assumptions can be seen in the rise and fall of the blue line, which represents the hare population
- In the absence of hares, the lynx die of proportional to the number of lynx - the equation for Lprime shows that if H is 0, then the lynx decline at a rate of dL (and is demonstrated by the declines in the orange line, which occur following the decline of the blue line representing hares)
- The lynx population increases proportionately to the number of encounters between lynx and hares - this can be seen in the first half of Lprime, cHL where c represents lynx per month per hare-lynx.

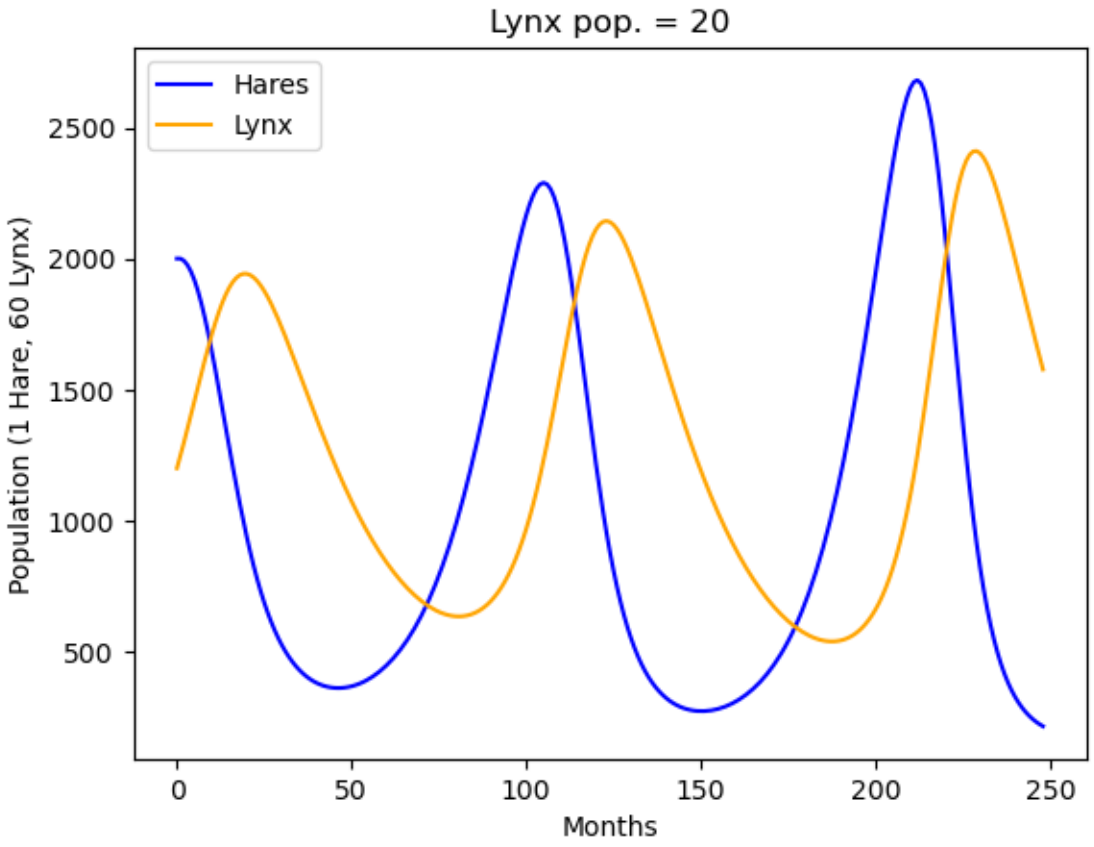
The hare-lynx units are especially interesting, as they complicate the model to include not only the populations of predators and prey, but also the encounters between the two (R.I.P. hares).

Below are the graphs for parts D and F of number 7, showing both the hare and lynx populations when L=20 and L=50. Doubling the lynx population from our initial graph condensed it, but changing it 50 did the opposite, spreading out the population peaks and valleys further over the

250 months.

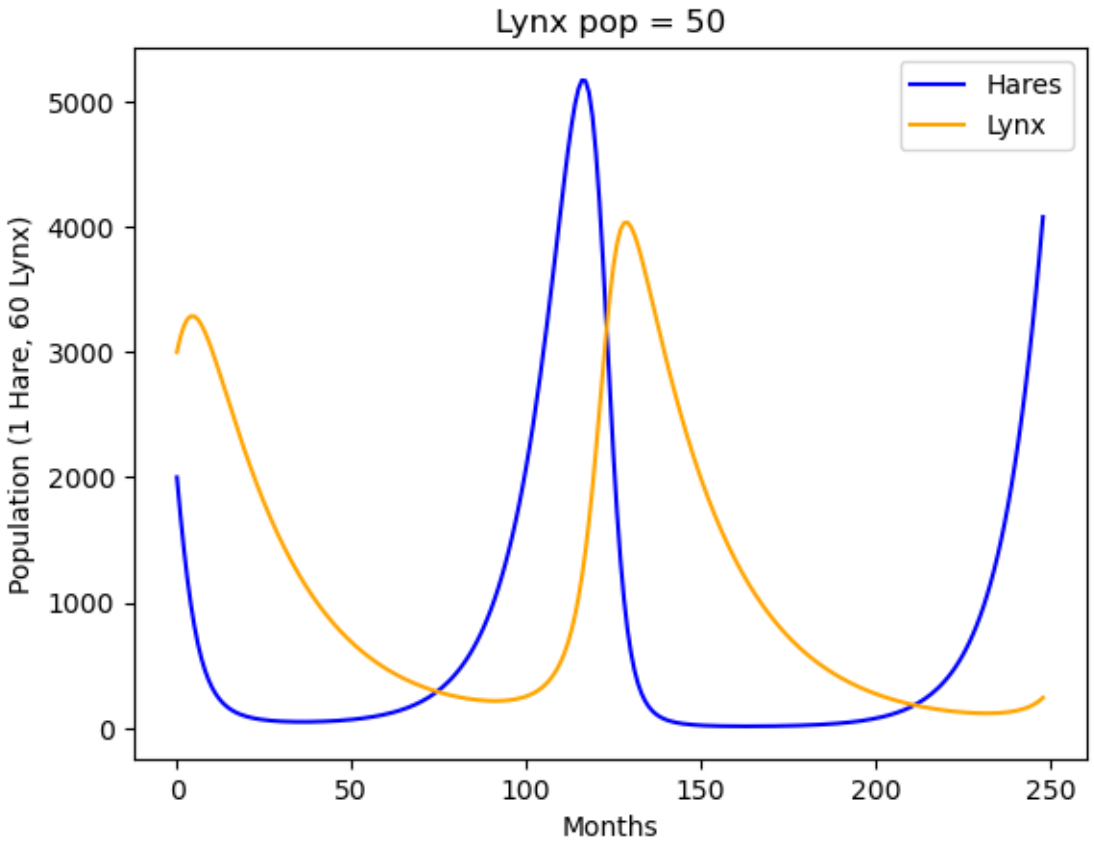
[3]:
```
H_line, L_line= lynxHare(2000, 20, 250)
plt.plot(H_line, color='blue', label='Hares')
plt.plot(L_line, color='orange', label='Lynx')
plt.legend(['Hares', 'Lynx'])
plt.xlabel('Months')
plt.ylabel('Population (1 Hare, 60 Lynx)')
plt.title('Lynx pop. = 20')
```

[3]:
```
Text(0.5, 1.0, 'Lynx pop. = 20')
```



[4]:
```
H_line, L_line= lynxHare(2000, 50, 250)
plt.plot(H_line, color='blue', label='Hares')
plt.plot(L_line, color='orange', label='Lynx')
plt.legend(['Hares', 'Lynx'])
plt.xlabel('Months')
plt.ylabel('Population (1 Hare, 60 Lynx)')
plt.title('Lynx pop = 50')
```

[4]: Text(0.5, 1.0, 'Lynx pop = 50')



Activity One: Do all parts of the May Model, 4.1 question 6.

0.1 May Model

his model has been proposed by the contemporary ecologist, R.M. May, to incorporate more realistic assumptions about the encounters between predators (foxes) and their prey (rabbits). So that you can work with quantities that are about the same size (and therefore plot them on the same graph), let y be the number of foxes and let x be the number of rabbits divided by 100—we are thus measuring rabbits in units of "hectorabbits".

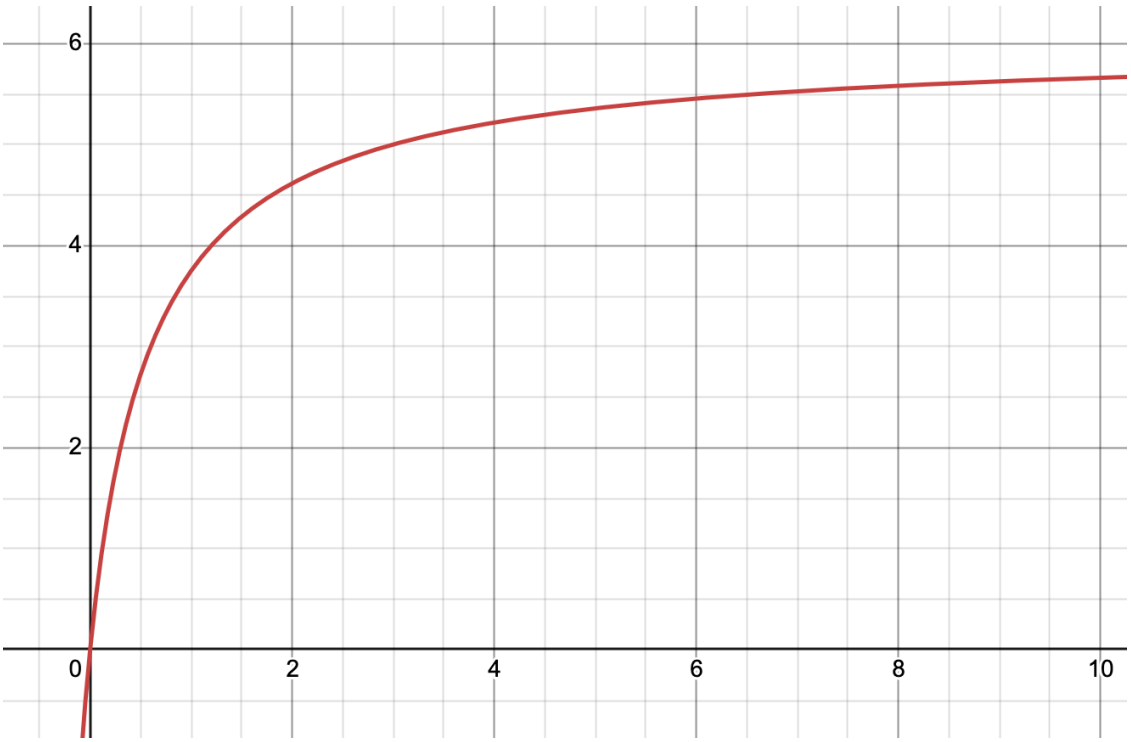
In his model, May makes the following assumptions:

- In the absence of foxes, the rabbits grow logistically.
- The number of rabbits a single fox eats in a given time period is a function D(x) of the number of rabbits available. D(x) varies from 0 if there are no rabbits available to some value c (the saturation value) if there is an unlimited supply of rabbits. The total number of rabbits consumed in the given time period will thus be D(x) $\cdot$ y
- The fox population is governed by the logistic equation, and the carrying capacity is proportional to the number of rabbits

0.1.1 A: Explain

why $D(x) = \frac{cx}{x+d}$ (d some constant) might be a reasonable model for the function D(x). Include a sketch of the graph of D in your discussion. What is the role of the parameter d? That is, what feature of rabbit – fox interactions is reflected by making d smaller or larger?

As shown above, $D(x)$ is the number of rabbits eaten by foxes in a given unit of time, expressed in hectorabbits. $D(x) = \frac{cx}{x+d}$ could be a reasonable model for the function $D(x)$ because as stated above, the number of rabbits a fox can eat is either limited by the fox itself (in the case of infinite rabbits), or by the population of rabbits x. In this equation, c represents an upper limit to how many rabbits one fox can eat, and this limit can not be surpassed as $\frac{x}{x+d}$ will have an upper limit of one when d is a positive number. Changing d changes the rate at which the graph reaches the limit c. When d is large, the graph becomes flatter in shape and c is approached more slowly. The graph of this function with $c = 6$ and $d = 1$ is shown below:



0.1.2 B:

Explain how the following system of equations incorporates May's assumptions. The parameters a, b, c, d, e and f are all positive.

$$x' = ax(1 - \frac{x}{b}) - \frac{cxy}{x+d}$$

$$y' = ey(1 - \frac{y}{fx}$$

These two equations represent the rate of change in the rabbit and fox populations respectively, in terms of the current rabbit and fox populations x and y. For both equations, the first term is the rate of a logistic growth rate function, where the population slowly increases, reaches a maximum

growth rate, and then slows as it approaches the limit represented by b for rabbits and fx for foxes. The denominator in the foxes growth equation being dependent on the rabbit population reflects the assumption that the carrying capacity is proportional to the number of rabbits. a and e in these equations set the initial growth rate. In the rabbit population growth equation, the second term in the equation is the amount of rabbits lost in the unit of time due to rabbit-fox interactions, and is the product $D(x) * y$ discussed in the second assumption. As described in part a, c represents the limit to how many rabbits a single fox can eat, and d represents how quickly that limit is reached.

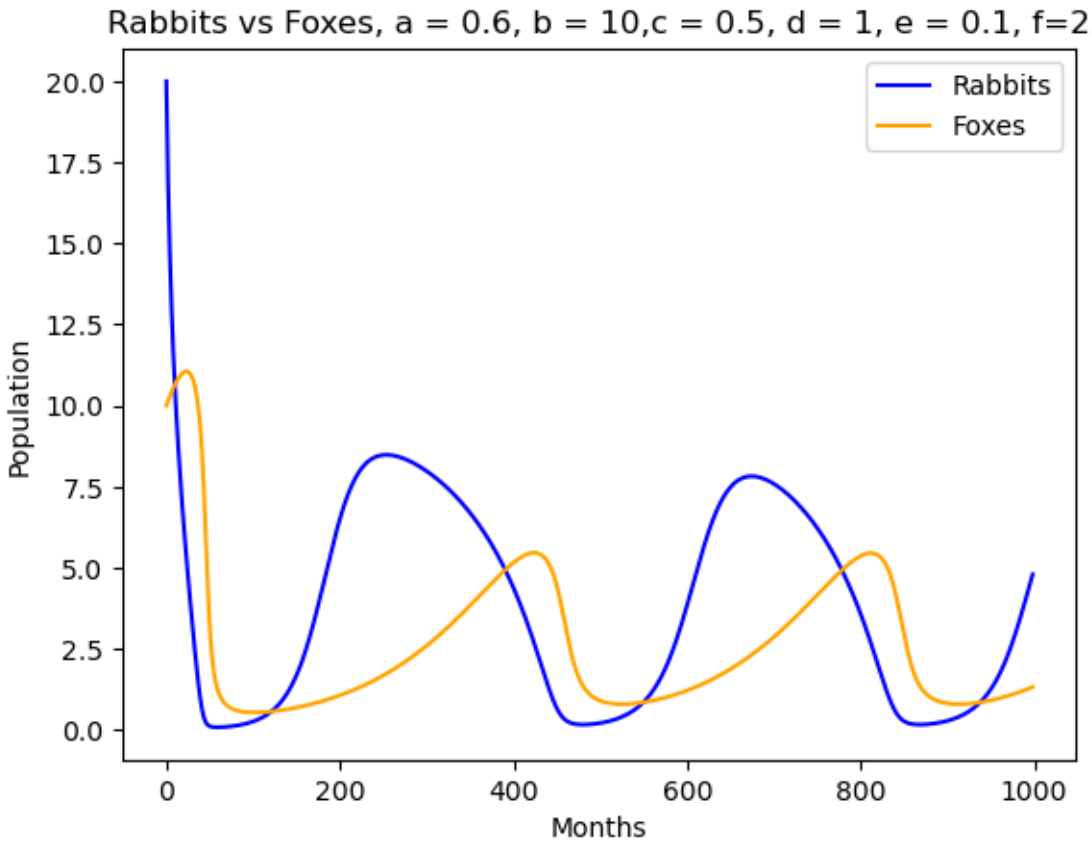
0.1.3 C:

Assume you begin with 2000 rabbits and 10 foxes. (Be careful: x(0) 6 =2000.) What does May's model predict will happen to the rabbits and foxes over time if the values of the parameters are a = .6, b = 10, c = .5, d = 1,e = .1 and f = 2? Use a suitable modification of the program SIRPLOT

On the plot below, the fox and rabbit populations flucctuate over time together - the population of foxes grows when the rabbit population is high, and then drops as the rabbit population declines. Likewise, the rabbit population grows when the fox population is low.

[5]:
```
def MayModel(R0, F0, tf, a0 = .6, b0 = 10, c0 = 0.5, d0 = 1, e0 = 0.1, f0 = 2):
    t_initial = 0
    t = t_initial

    R = R0
    F = F0

    a = a0
    b = b0
    c = c0
    d = d0
    e = e0
    f = f0

    # Creating empty arrays to hold values we will plot
    R_graph = []
    F_graph = []

    # Change in time/number of steps
    delta_t = 0.1

    # Iterating through the number of line segments in our plot (the more␣
 ↪lines, the smoother the graph)
    for k in range (1, tf*10):
        R_graph.append(R)
        F_graph.append(F)

        Rprime = (a*R) * (1-(R/b)) - ((c*R*F)/(R+d))
        Fprime = (e*F) * (1-(F/(f*R)))
```

```
        # Change in Rabbit and Fox populations
        delta_R = Rprime*delta_t
        delta_F = Fprime*delta_t

        #Updating the values of t, R, and F
        t = t + delta_t
        R = R + delta_R
        F = F + delta_F

    # return the arrays with values over each step
    fig, ax = plt.subplots()
    ax.plot(R_graph, color='blue', label='Rabbits')
    ax.plot(F_graph, color='orange', label='Foxes')
    ax.legend(['Rabbits', 'Foxes'])
    ax.set_title(f'Rabbits vs Foxes, a = {a}, b = {b},c = {c}, d = {d}, e =␣
 ↪{e}, f={f}')
    ax.set_xlabel('Months')
    ax.set_ylabel('Population')

    return R_graph, F_graph


# Plot the populations
#initial value of R is 2000/100, units of hectorabbits for better comparison
R_line, F_line = MayModel(20, 10, 100, c0 = 0.5)
```

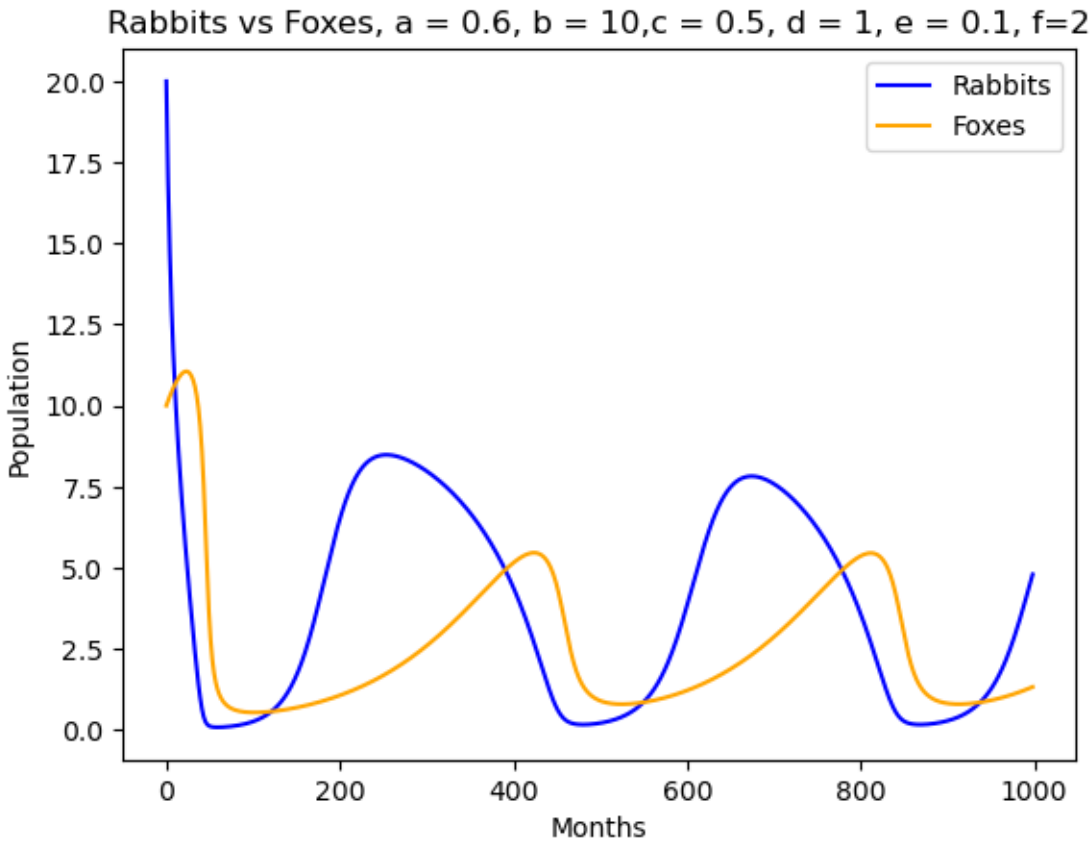


0.1.4 D.

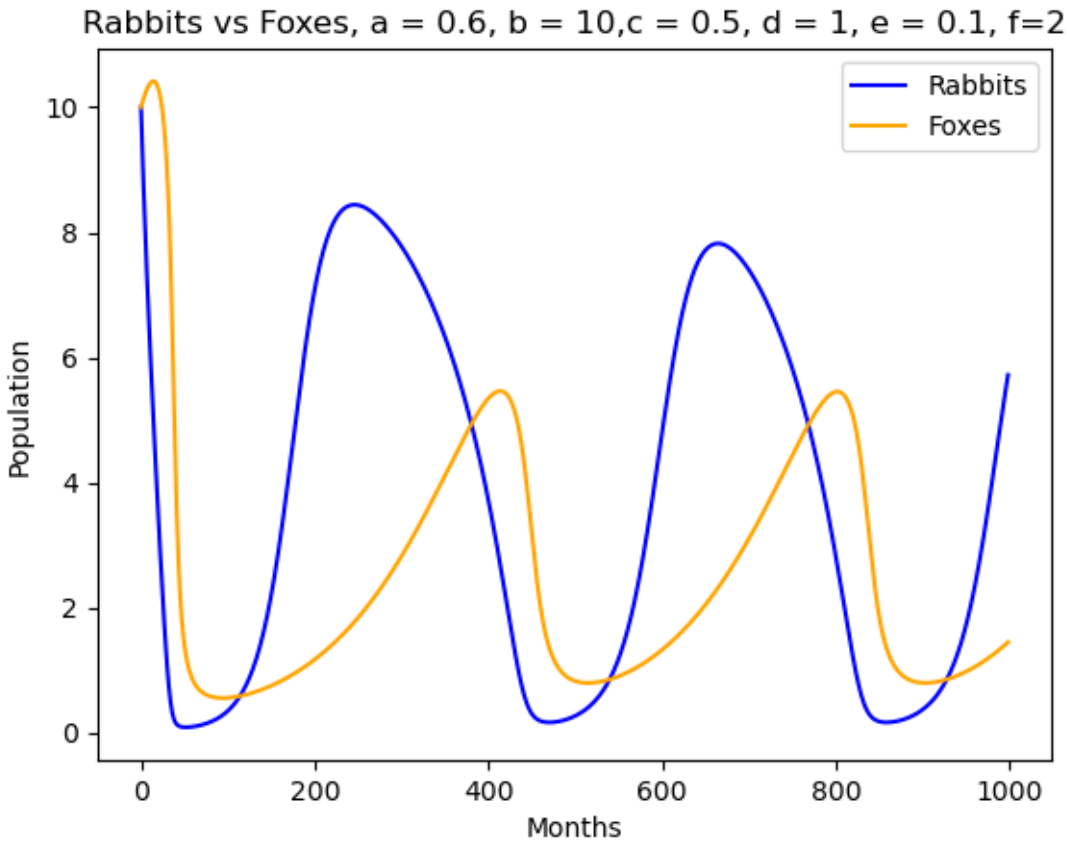
Using the same parameters, describe what happens if you begin with 2000 rabbits and 20 foxes; with 1000 rabbits and 10 foxes; with 1000 rabbits and 20 foxes. Does the eventual long-term behavior depend on the initial condition? How does the long-term behavior here compare with the long-term behavior of the two populations in the Lotka–Volterra model of the text?

Based on the three plots below, it looks like the long term behavior depends little on the initial condition. In all cases, the two populations grow and decline with each other, the maximum fox population is about 5 foxes to 100 rabbits. This is different than the Lotka-Volterra model in the text because the LV model as it appears that the populations have an almost triangular or "shark fin" waveform as opposed to the periodic one presented by the LV model. Both models do fluctuate and return to the same values at different intervals.

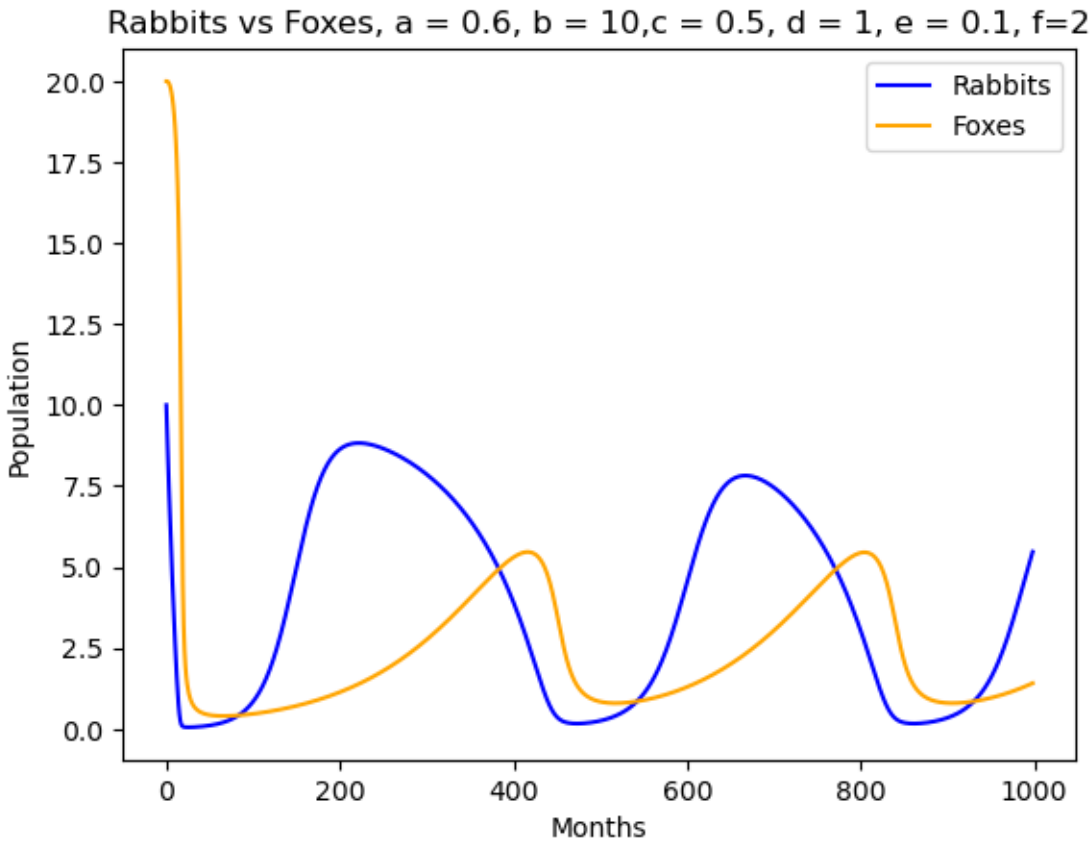
[6]:
```
Rd1, Fd1 = MayModel(20, 10, 100)
```



[7]:
```
Rd2, Fd2 = MayModel(10, 10, 100)
```



[8]:
```
Rd3, Fd3 = MayModel(10, 20, 100)
```



0.1.5 E

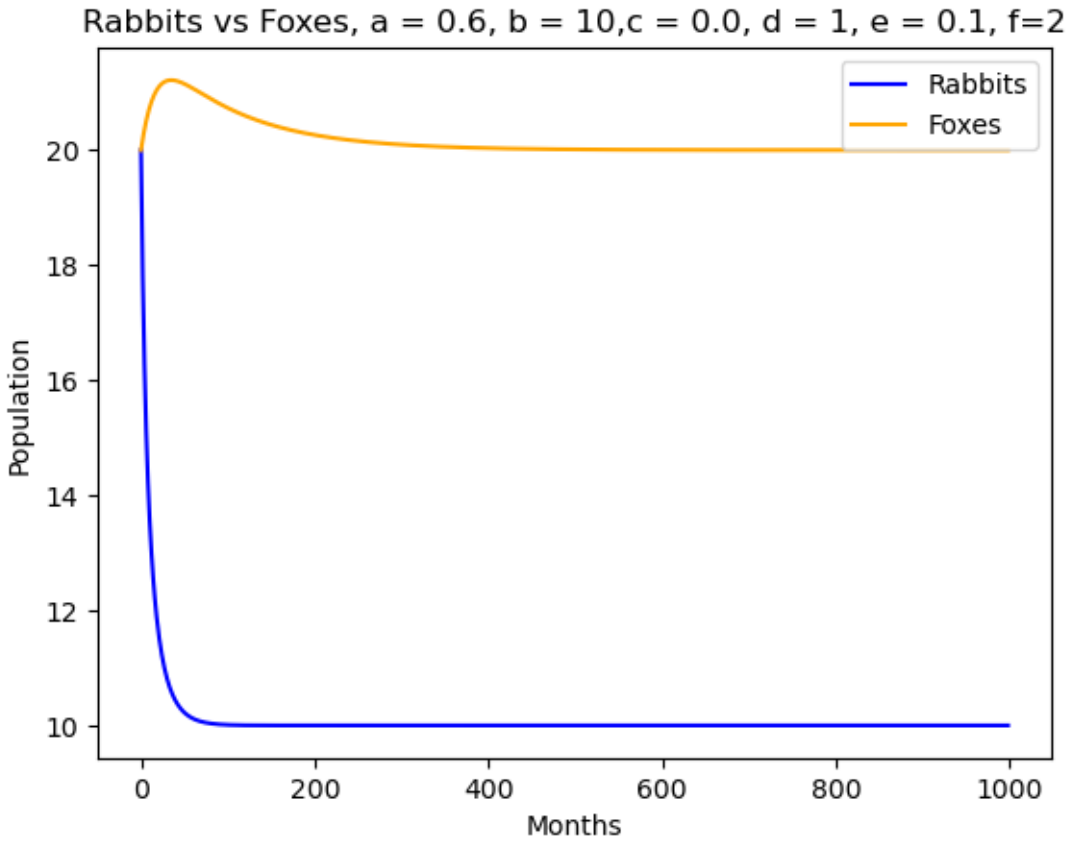
Using 2000 rabbits and 20 foxes as the initial values, let's see how the behavior of the solutions is affected by changing the values of the parameter c, the saturation value for the number of rabbits (measured in centirabbits, remember) a single fox can eat in a month. Keeping all the other parameters (a, b, d, . . .) fixed at the values given above, get solution curves for c = .5, c = .45, c = .4, . . . , c = .15, and c = .1 . The solutions undergo a qualitative change somewhere between c = .3 and c = .25. Describe this change. Can you pinpoint the crucial value of c more closely? This phenomenon is an example of Hopf bifurcation, which we will look at more closely in chapter 8. The May model undergoes a Hopf bifurcation as you vary each of the other parameters as well. Choose a couple of them and locate approximately the associated bifurcation values.

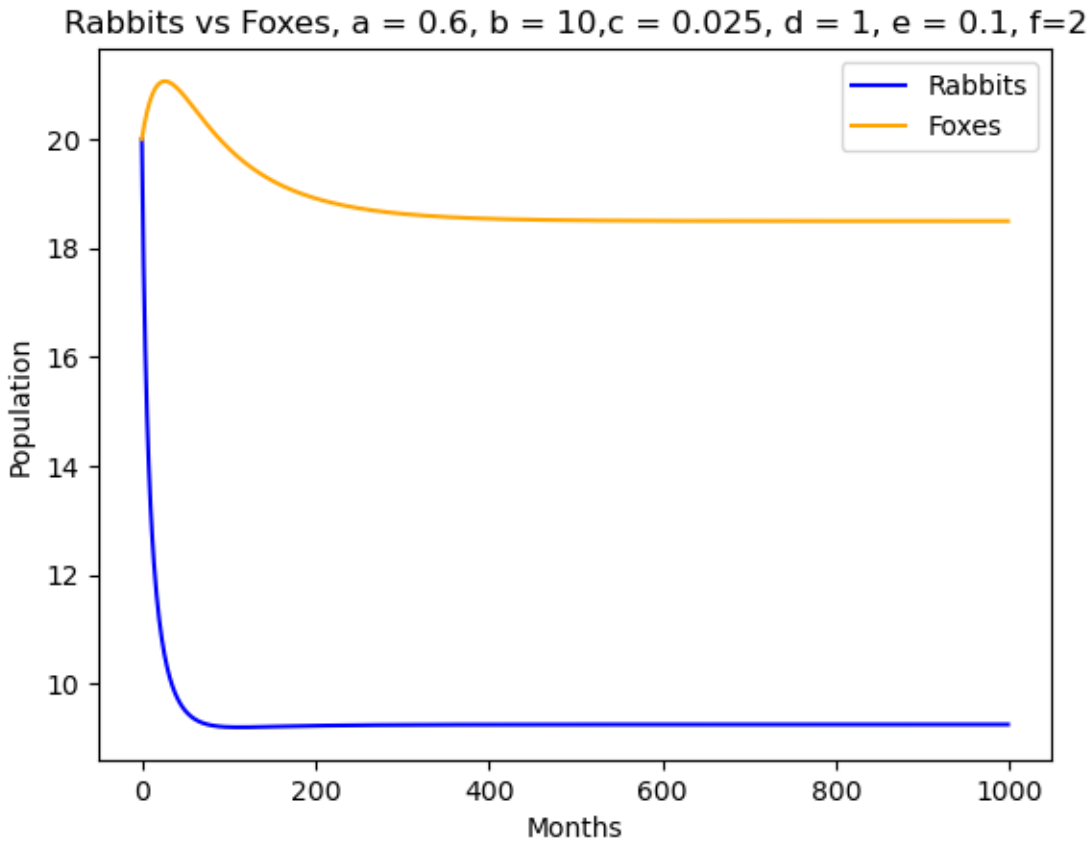
In the plots below, it appears that the plots become periodic when the c value is about 0.275 or higher. For the other parameters (when the other 5 are kept the same as above: - a: Bifurcation happens when a = 0.5 or higher - d: Bifurcation happens when d = 2 or lower.

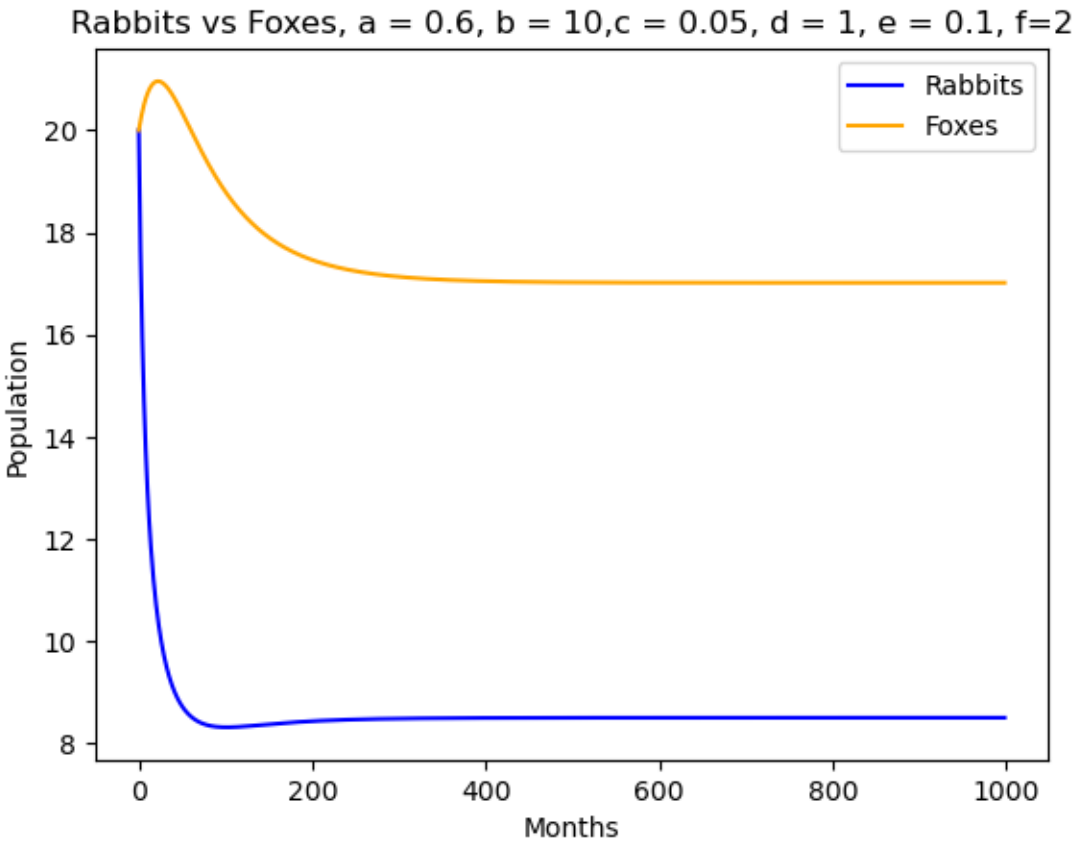
[9]:
```
for k in range(20):
    c = k/40
    MayModel(20, 20, 100, c0 = c)
for l in range(10):
    a = l/10
```

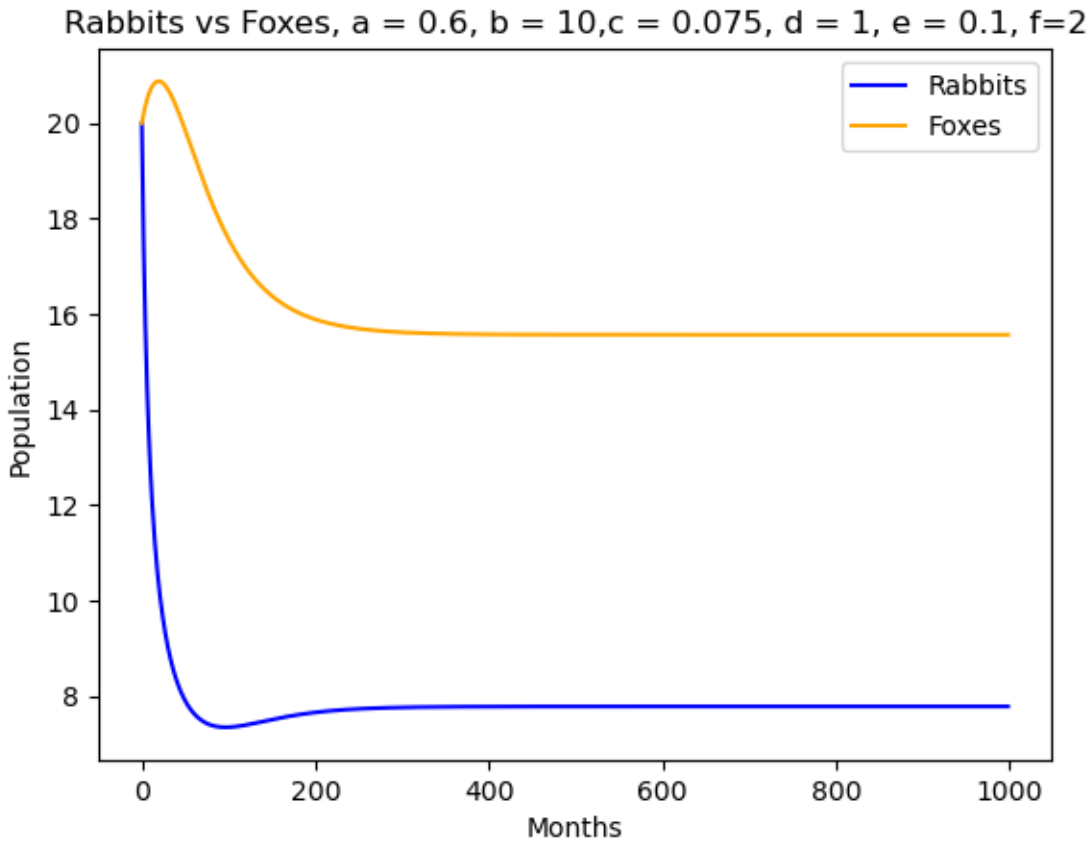
```
    MayModel(20, 20, 100, a0 = a)
for m in range(5):
    d = m
    MayModel(20, 20, 100, d0 = d)
```

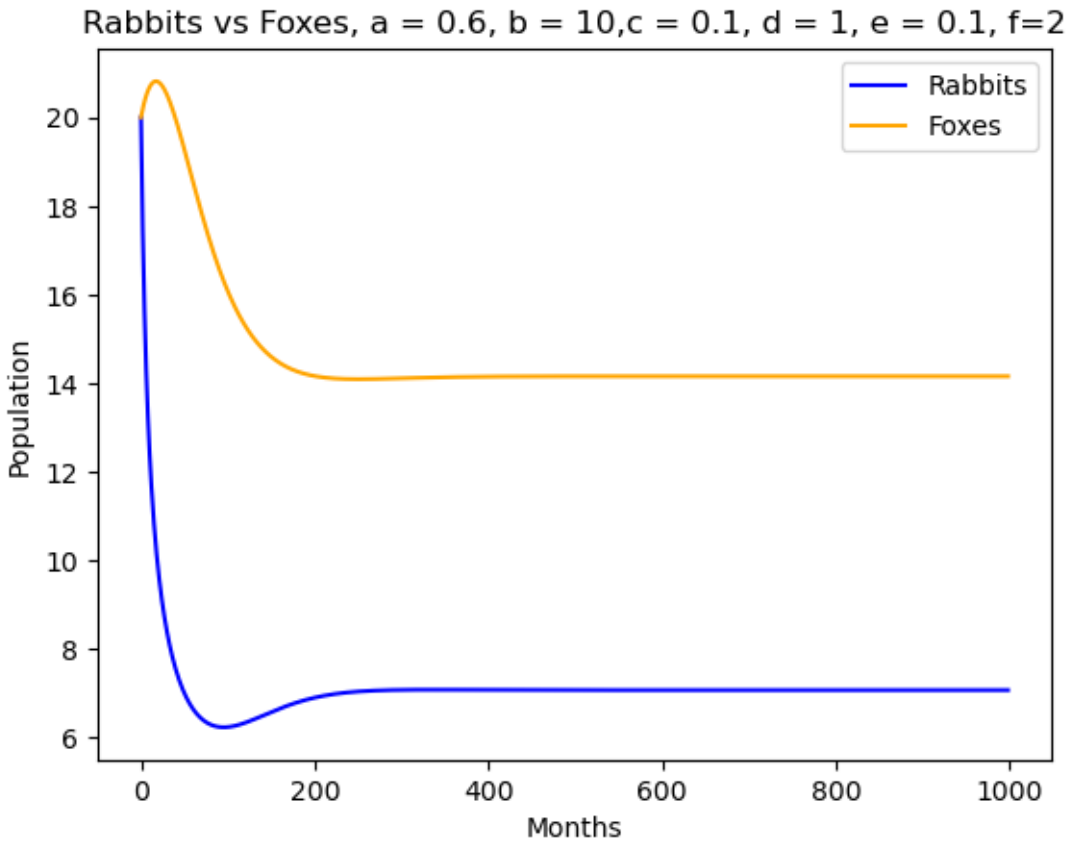
```
/tmp/ipykernel_159/1081000395.py:40: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  fig, ax = plt.subplots()
```

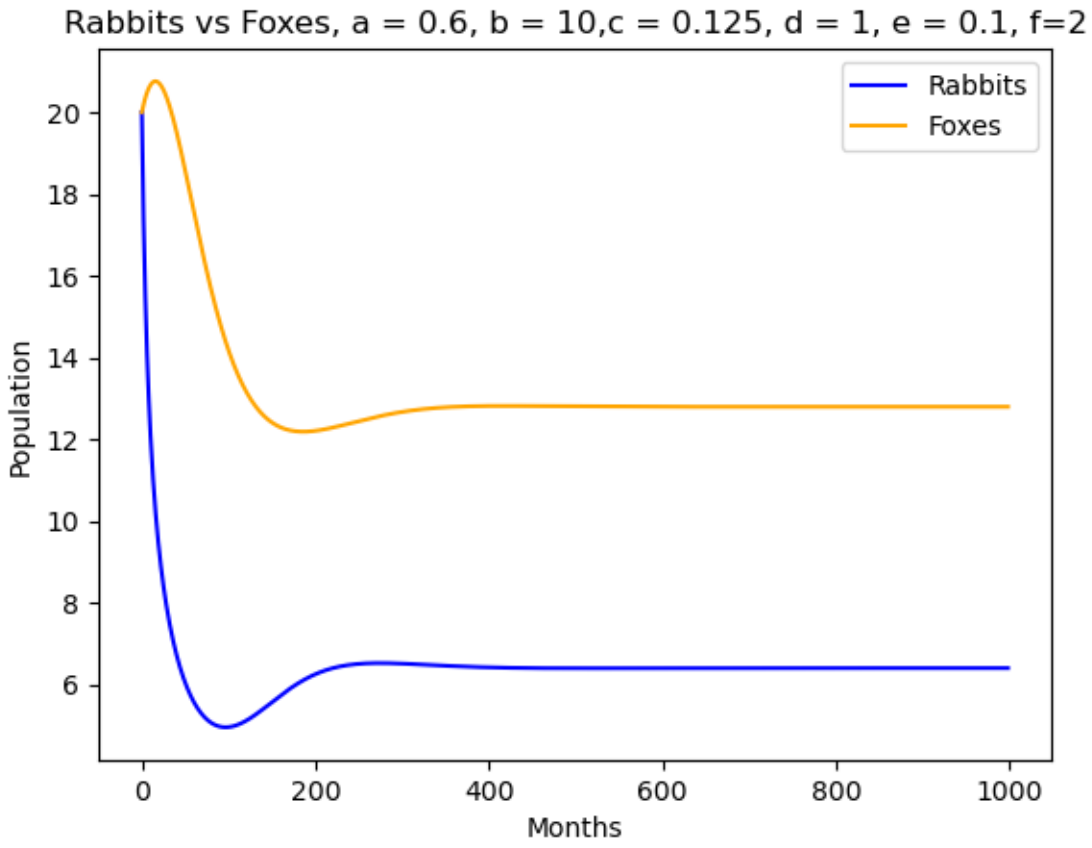


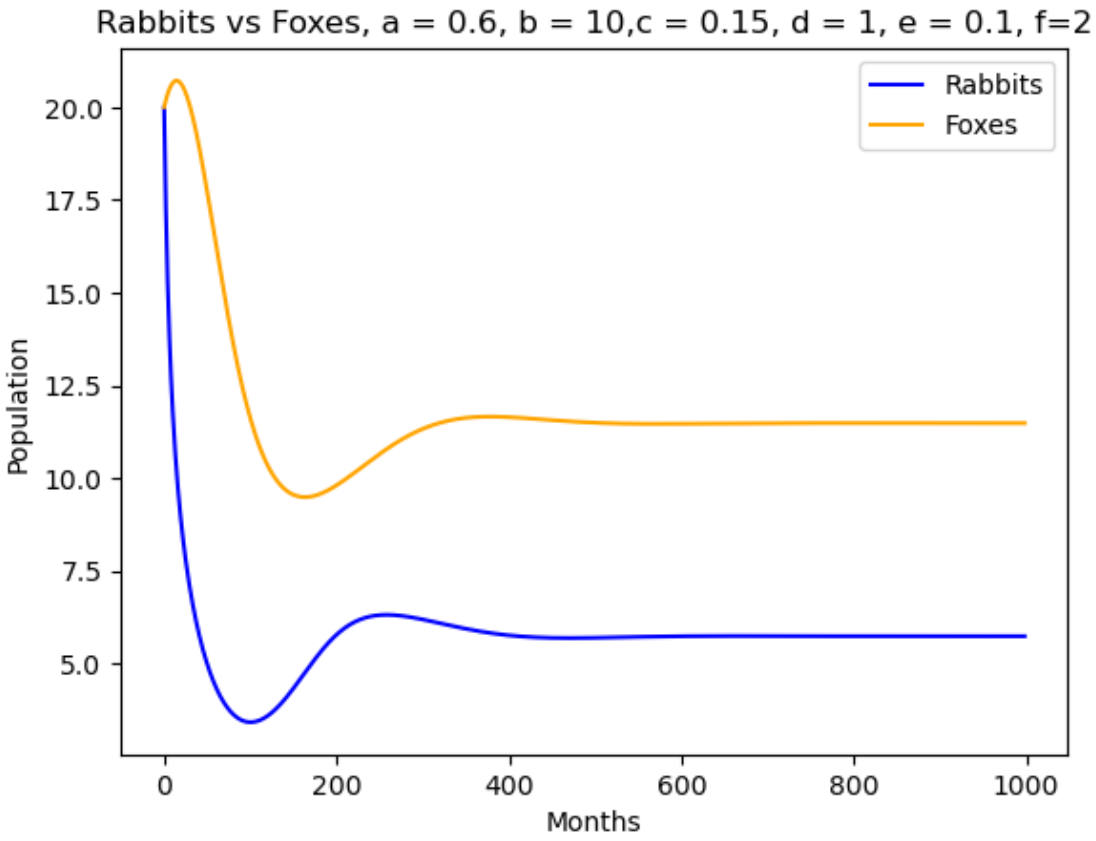

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.025, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20
18
16
14
12
10
0
200
400
600
800
1000

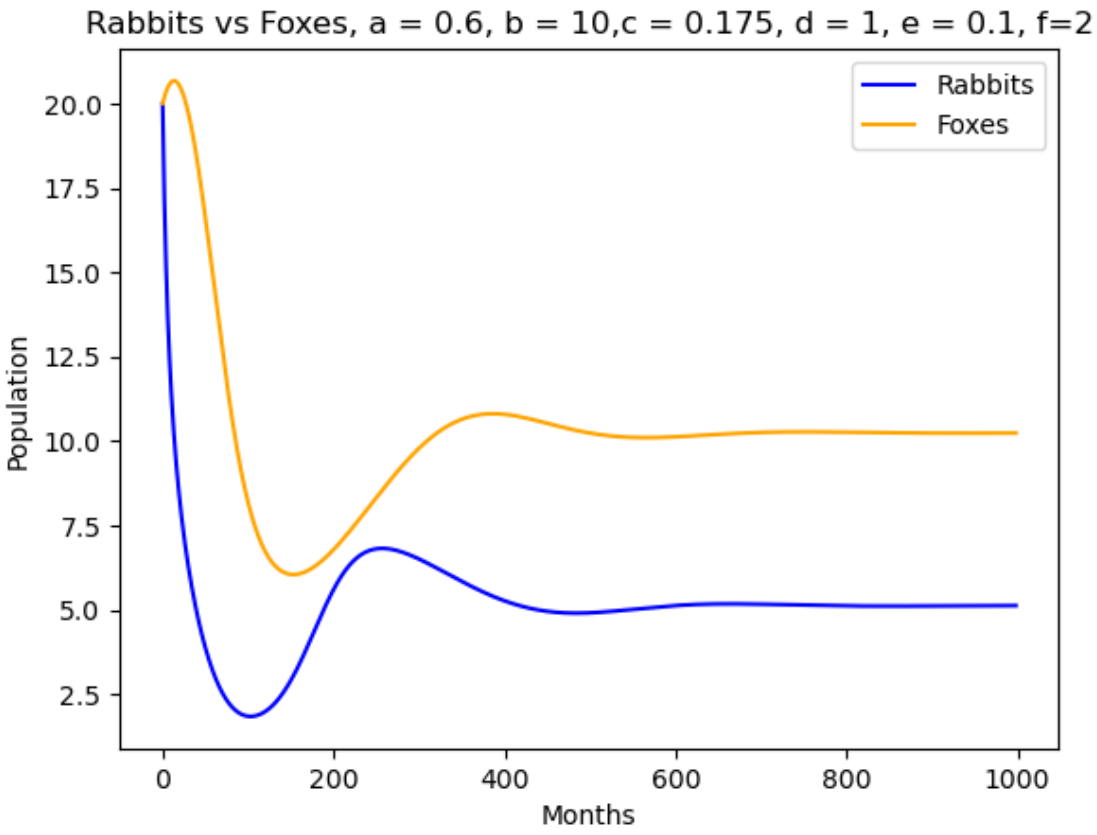

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.05, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20
18
16
14
12
10
8
0
200
400
600
800
1000

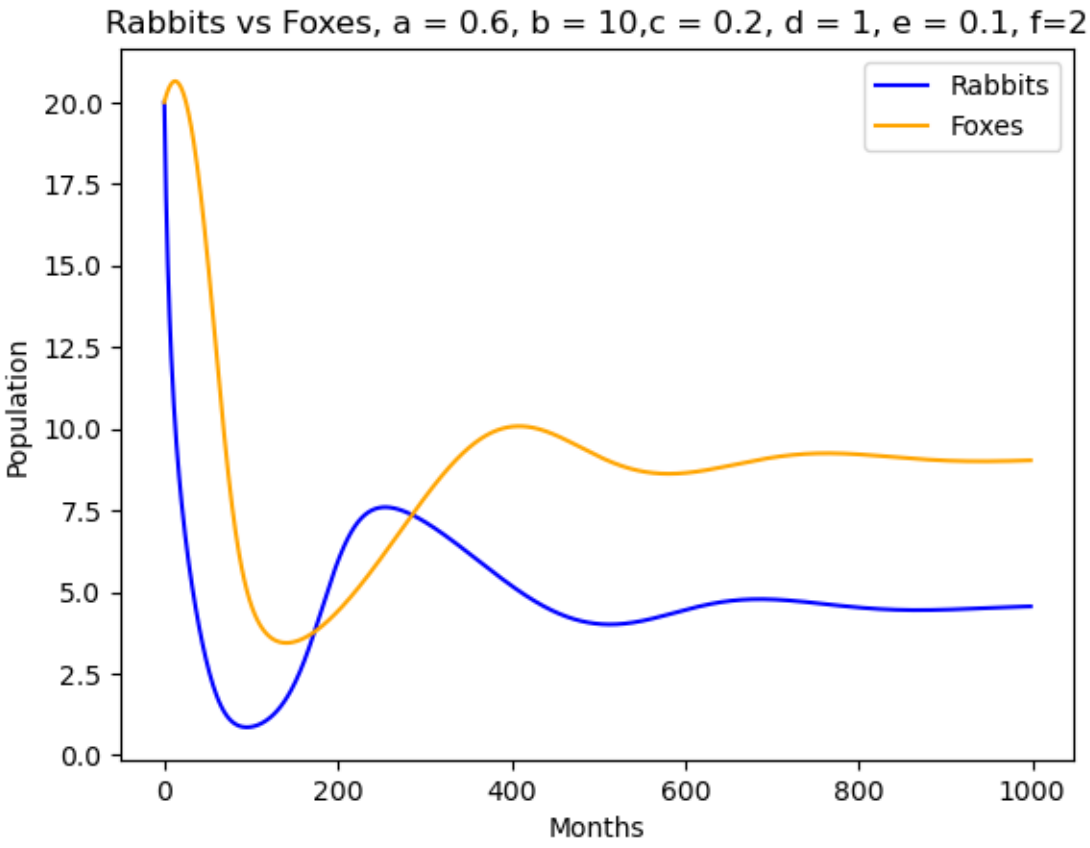

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.075, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20
18
16
14
12
10
8
0
200
400
600
800
1000

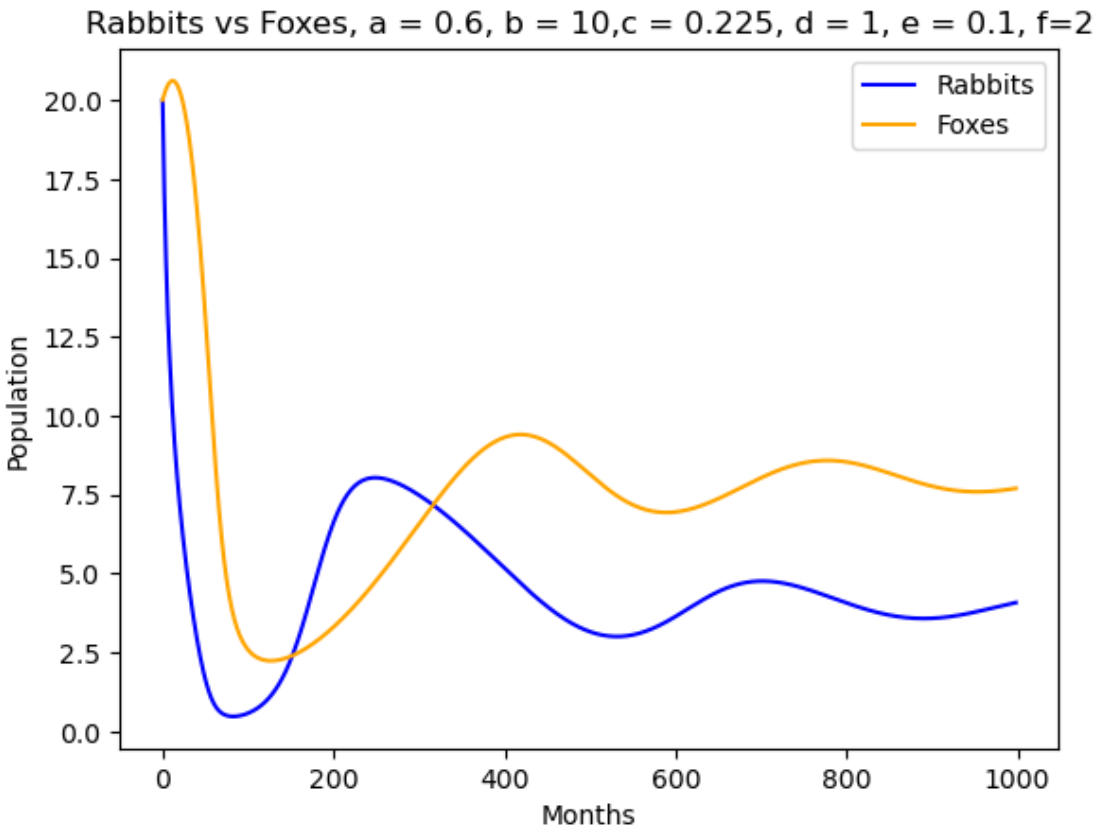

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.1, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20
18
16
14
12
10
8
6
0
200
400
600
800
1000

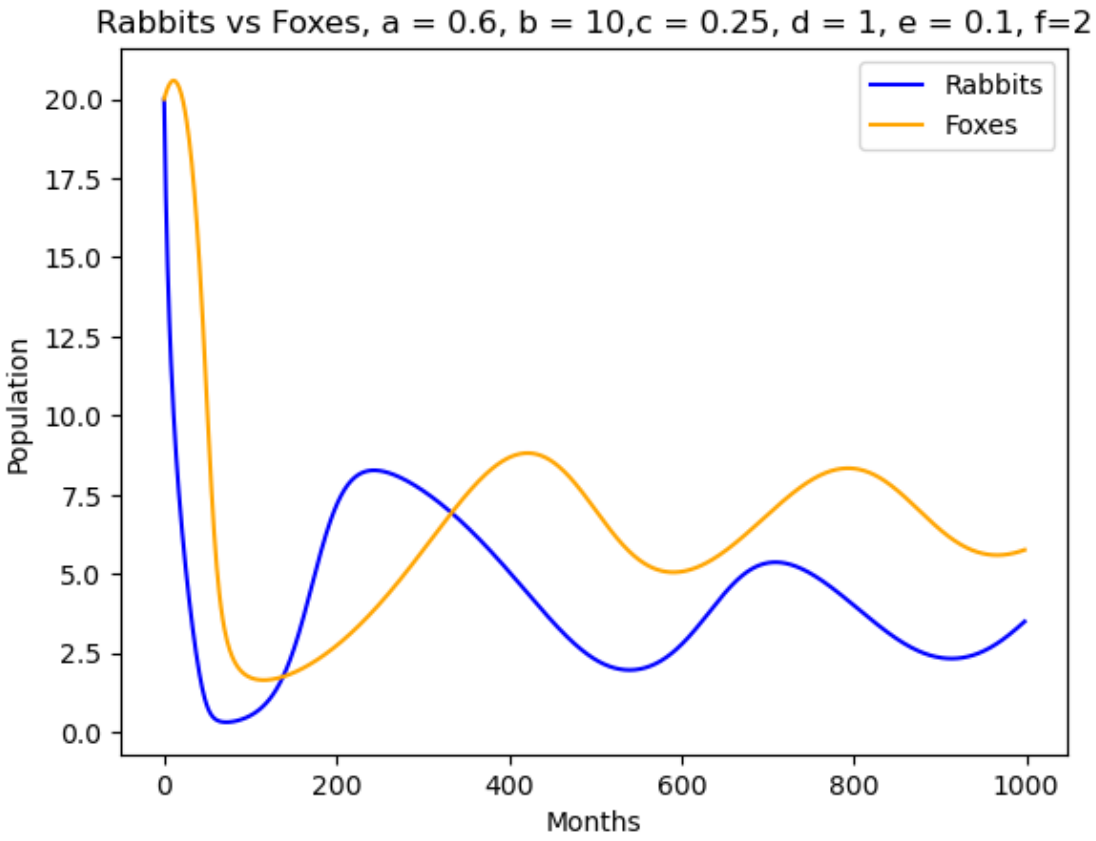

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.125, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20
18
16
14
12
10
8
6
0
200
400
600
800
1000

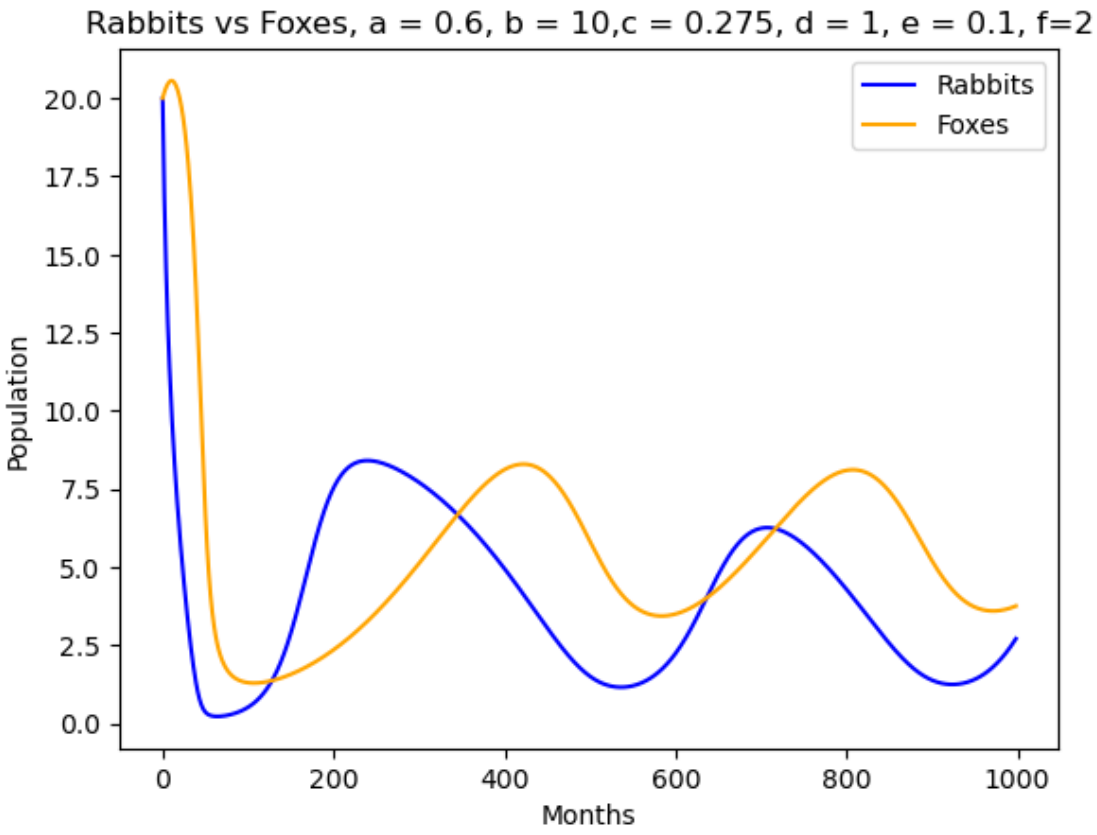

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.15, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
20.0
17.5
15.0
12.5
10.0
7.5
5.0
0
200
400
600
800
1000
Months

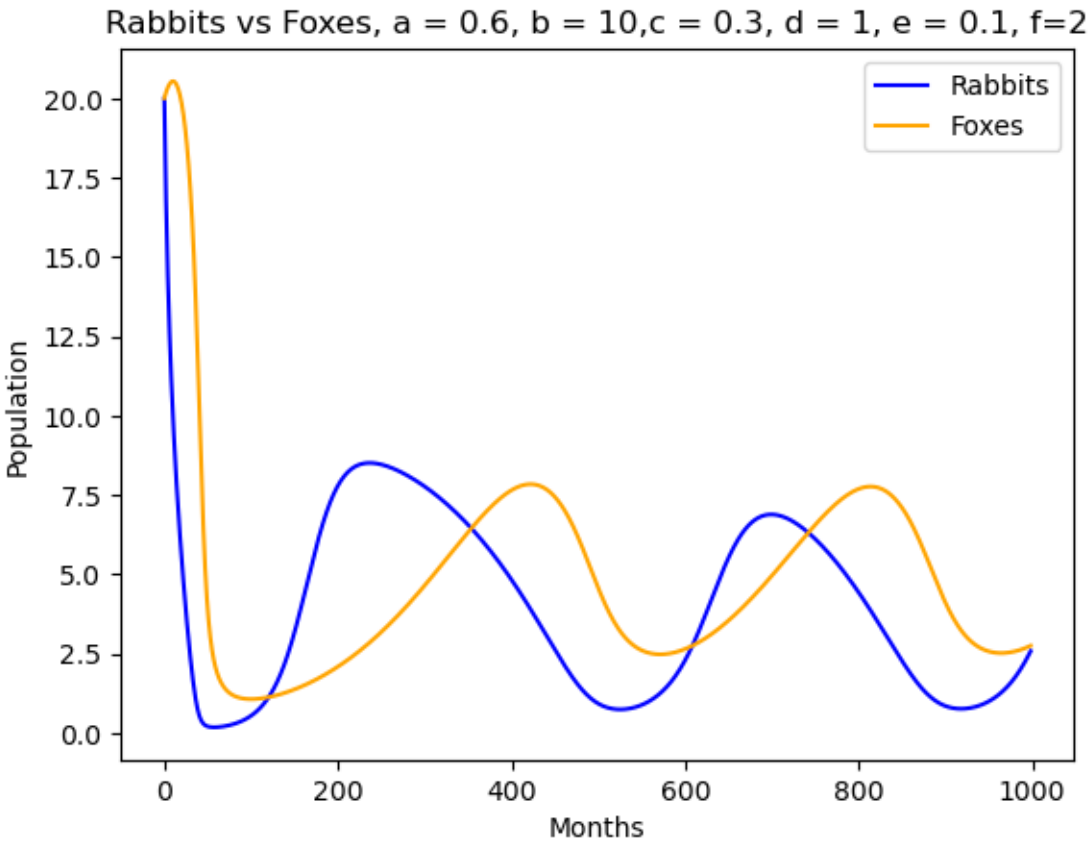

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.175, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
200
400
600
800
1000

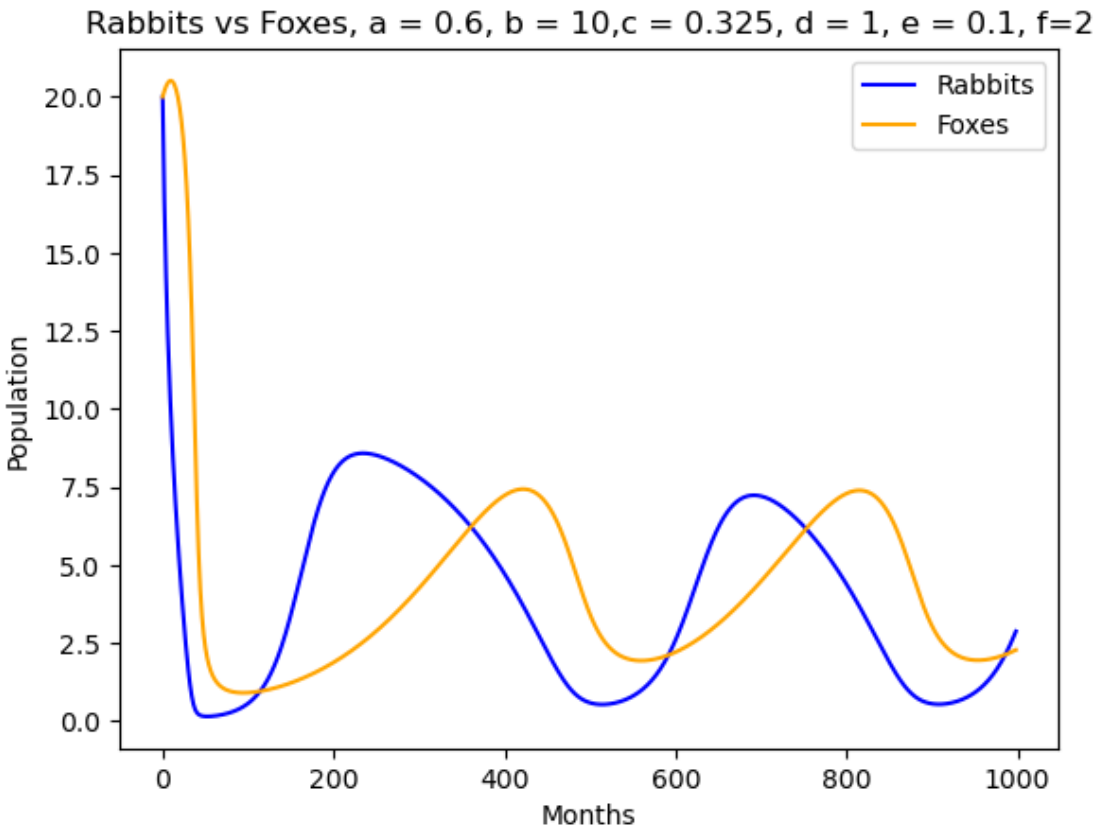

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.2, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

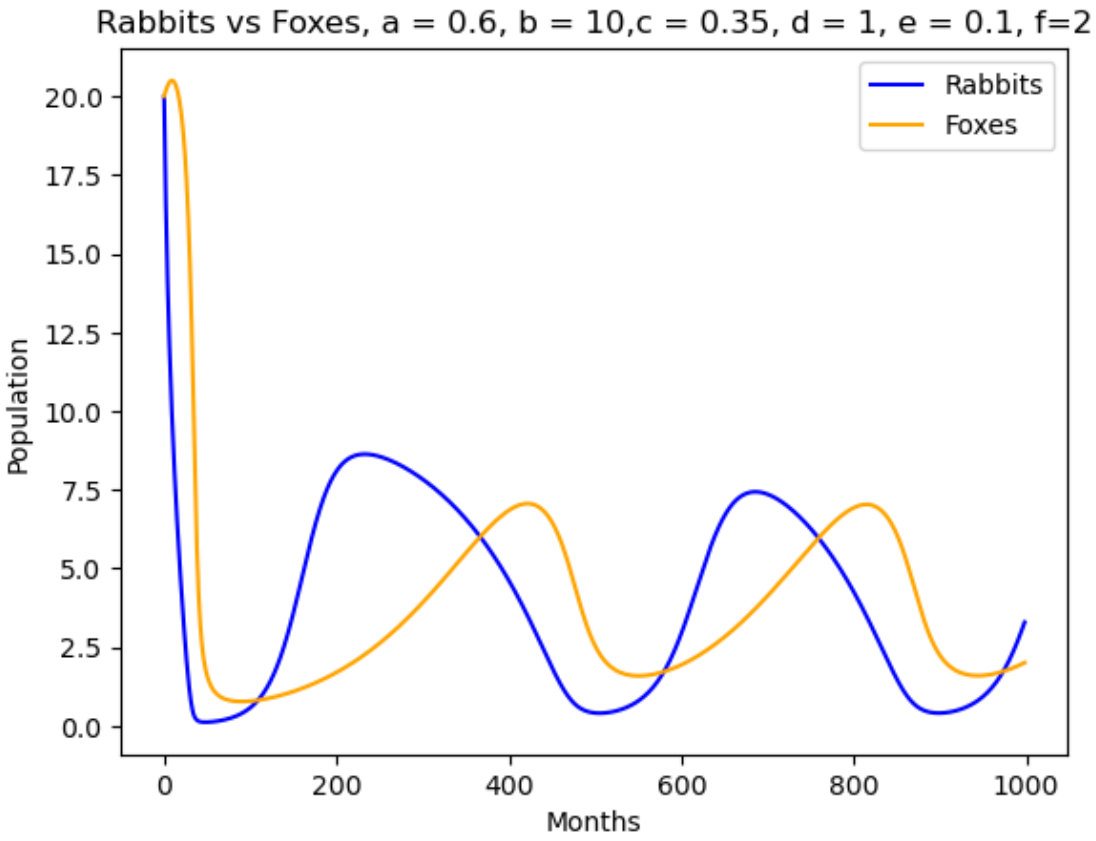

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.225, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

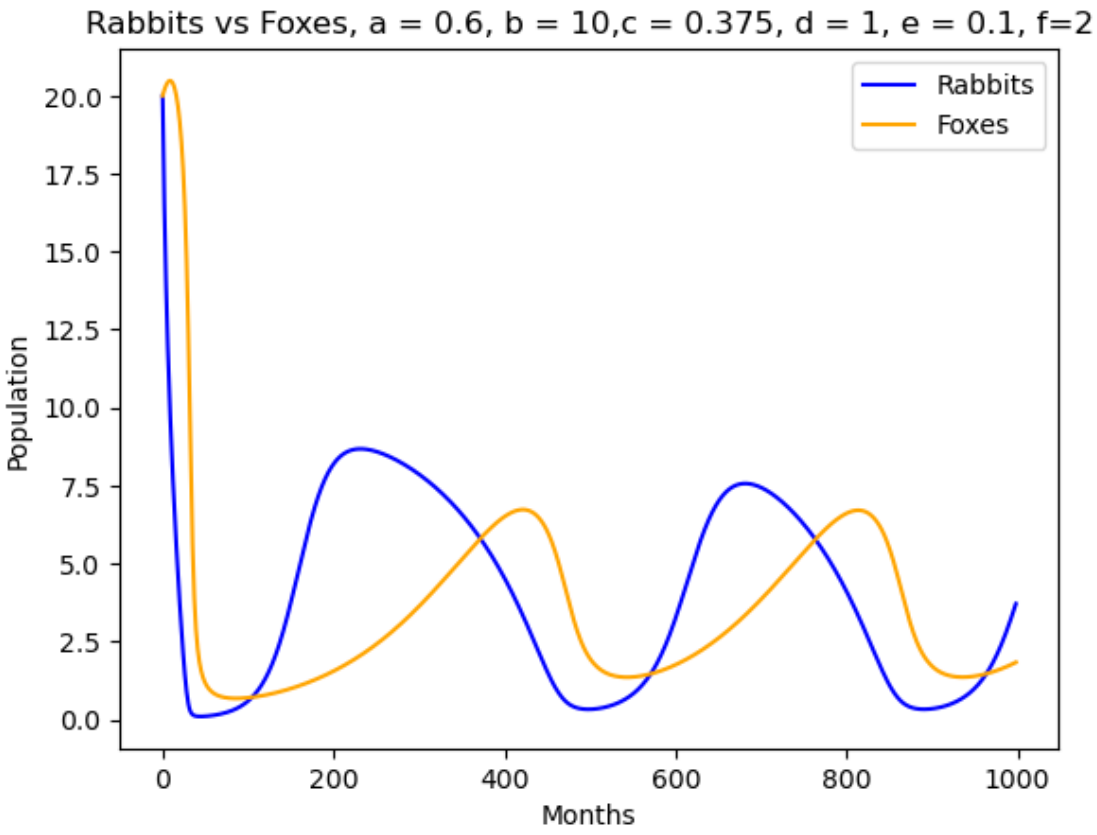

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.25, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

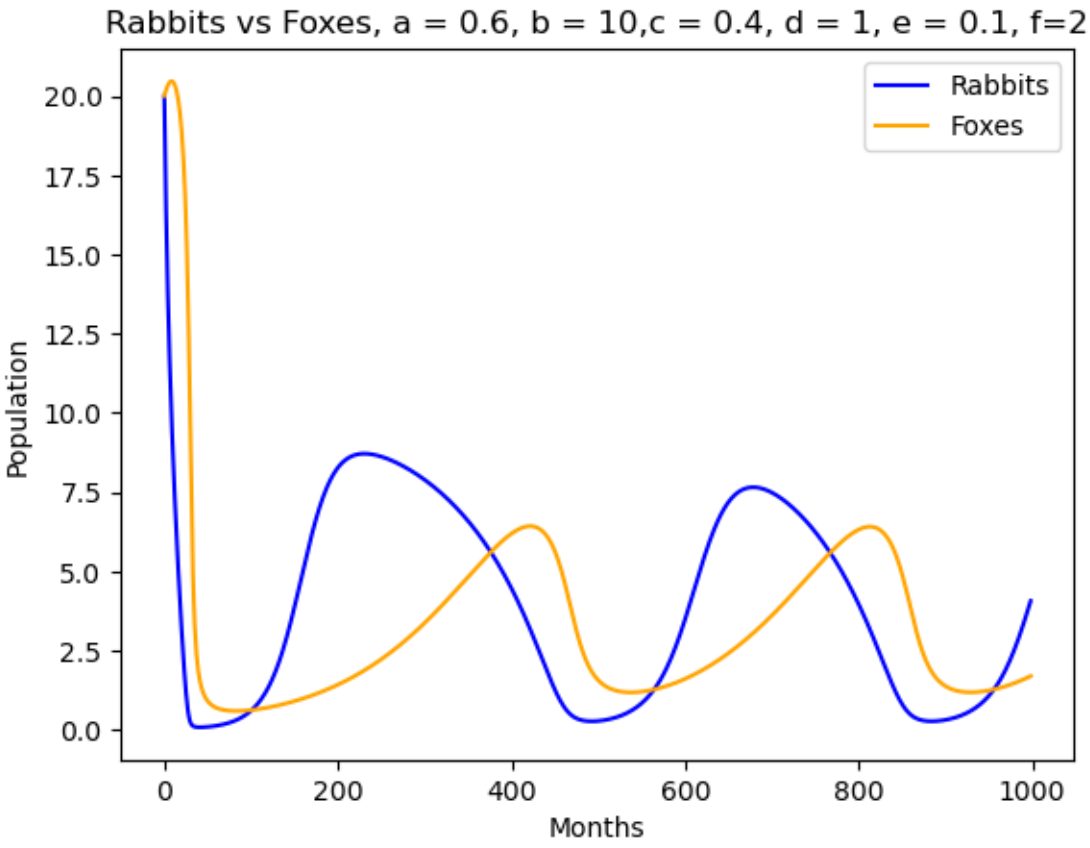

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.275, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

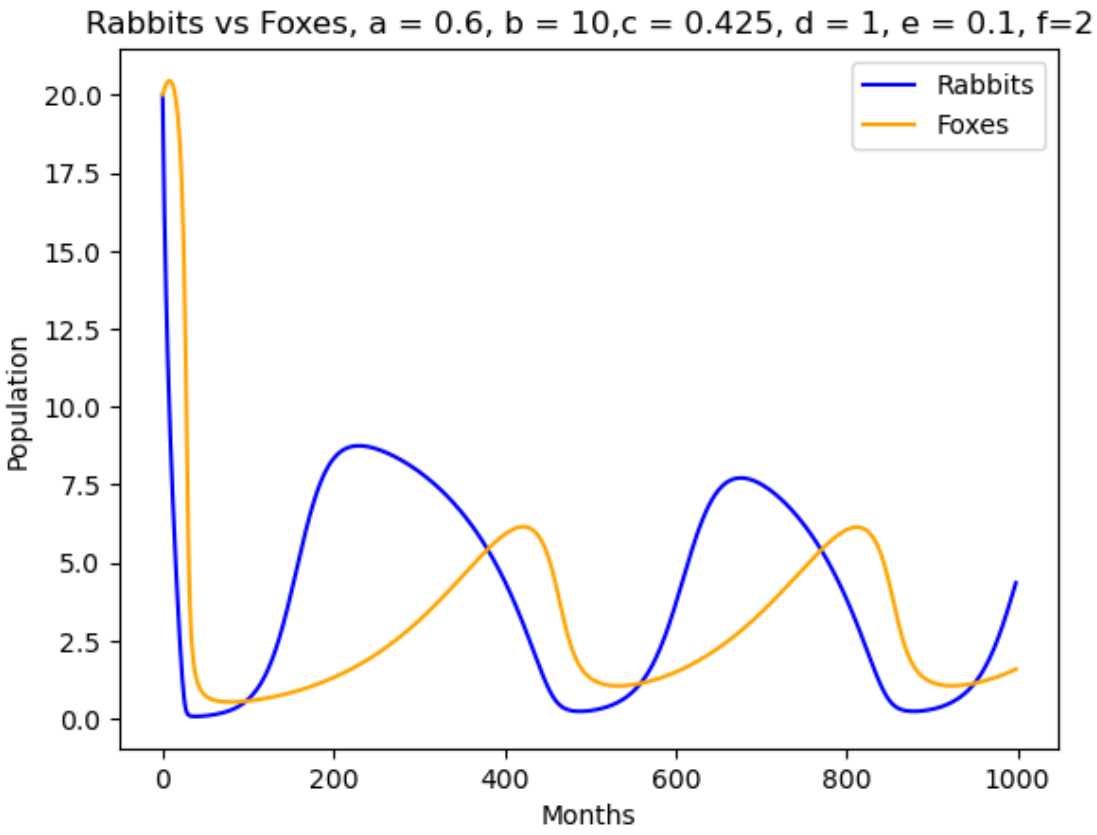

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.3, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

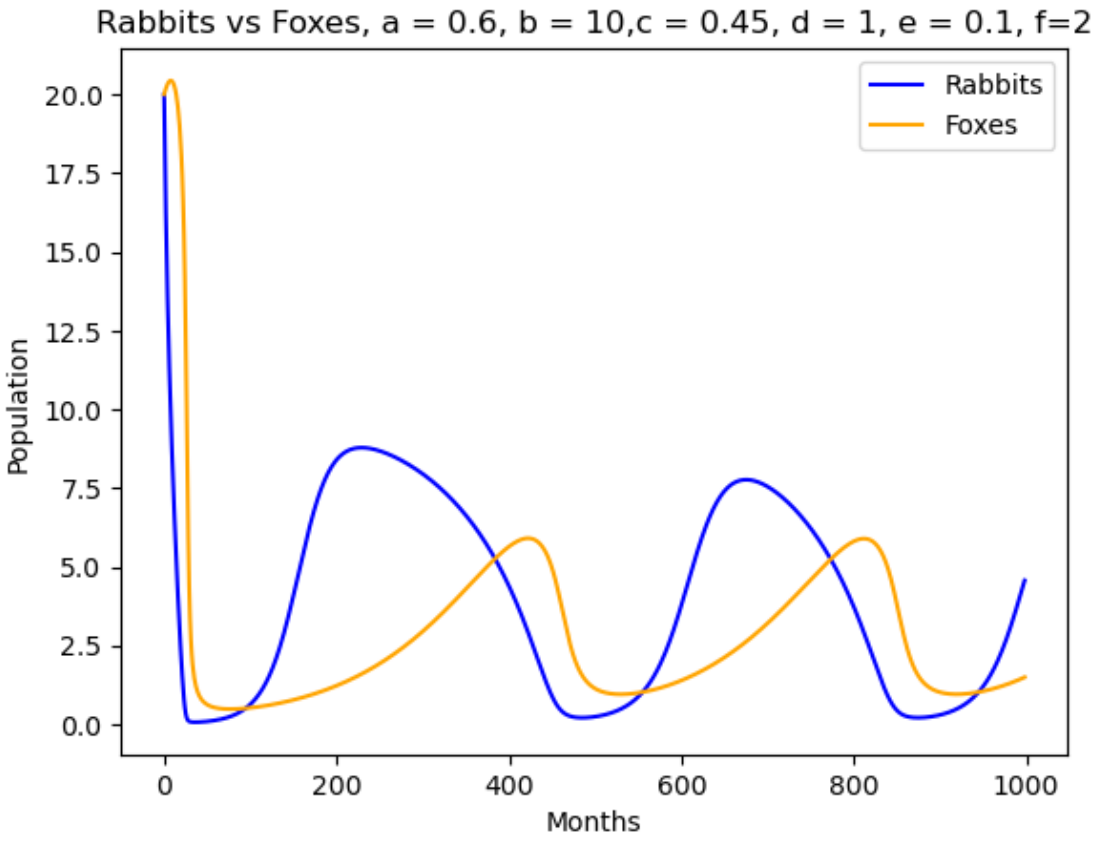

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.325, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

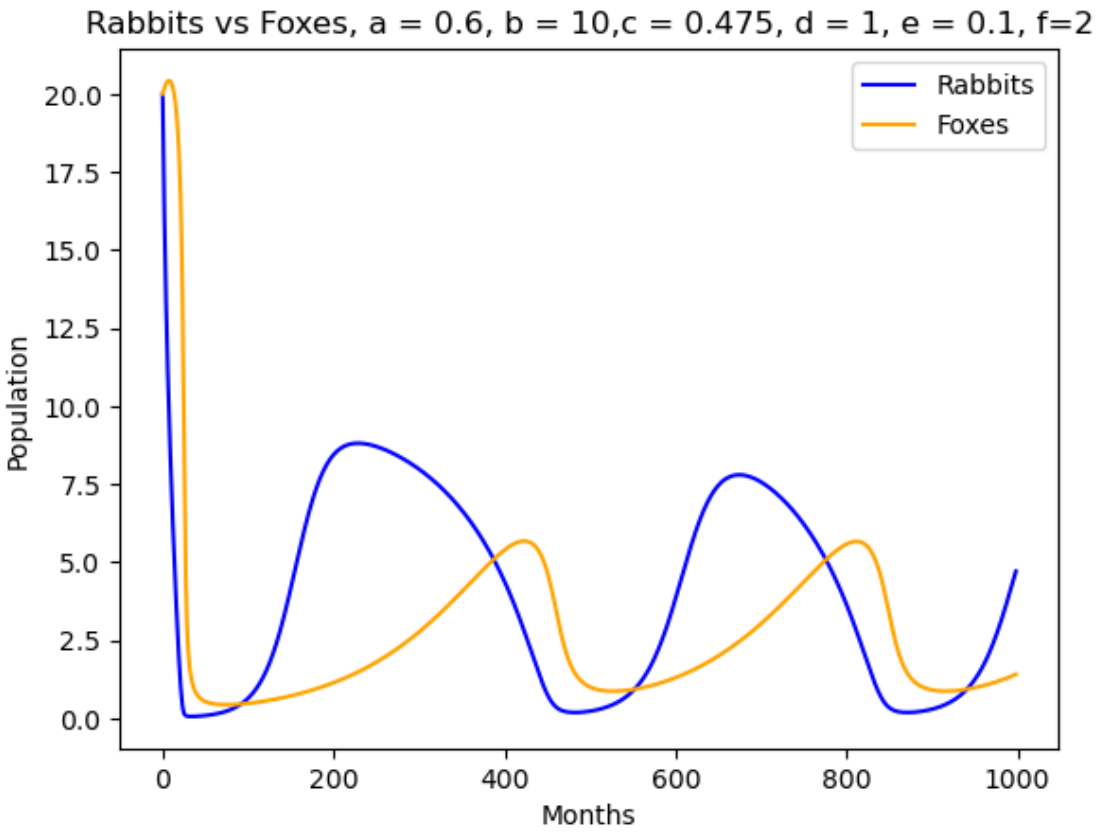

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.35, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

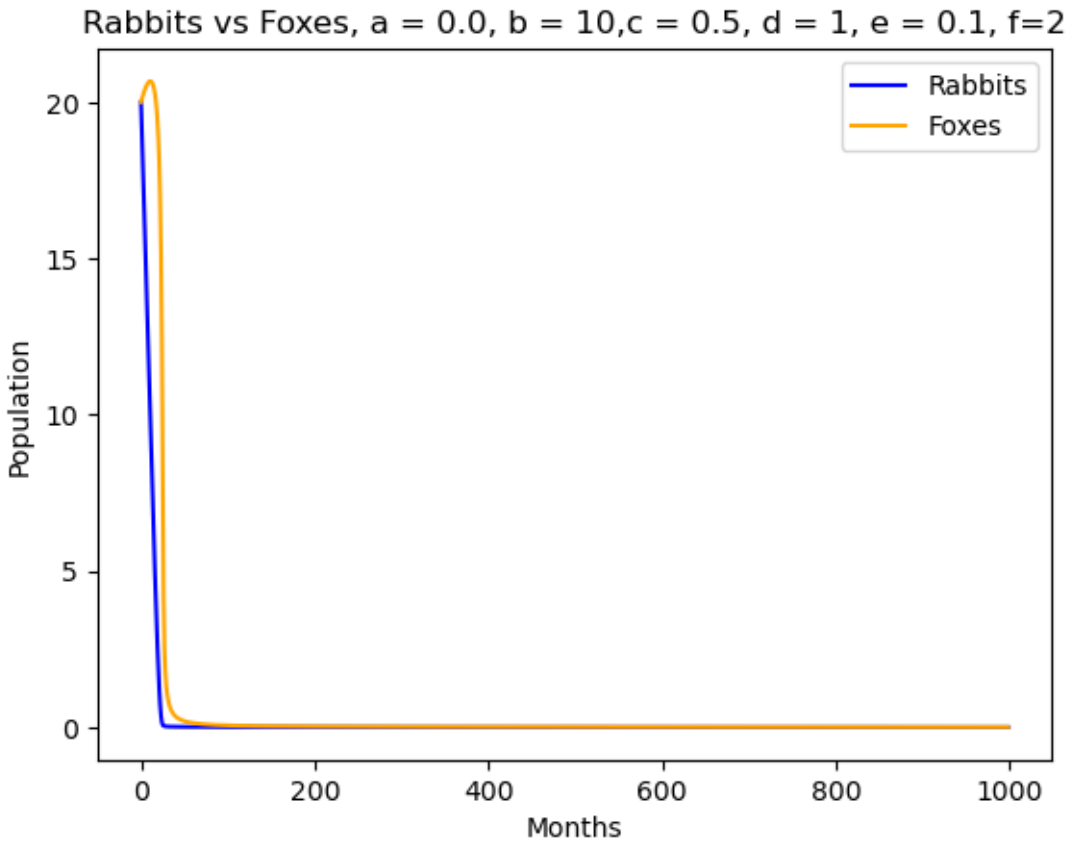

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.375, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

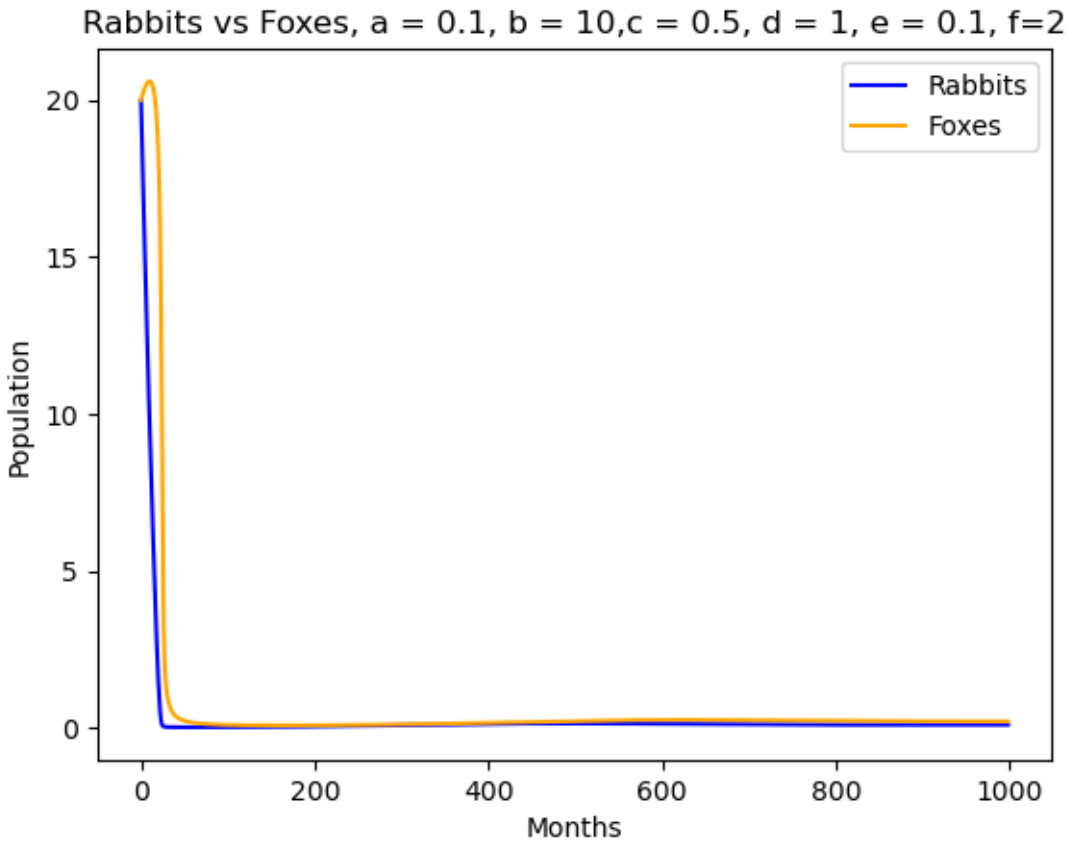

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.4, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

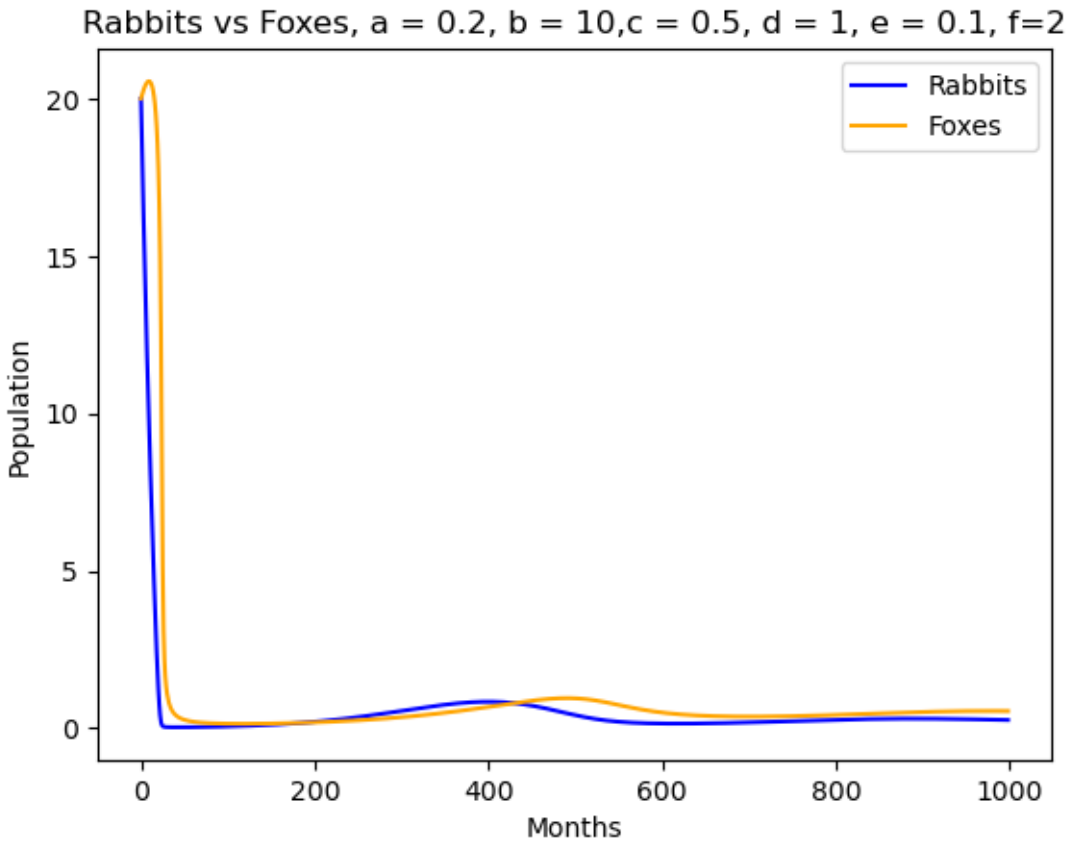

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.425, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

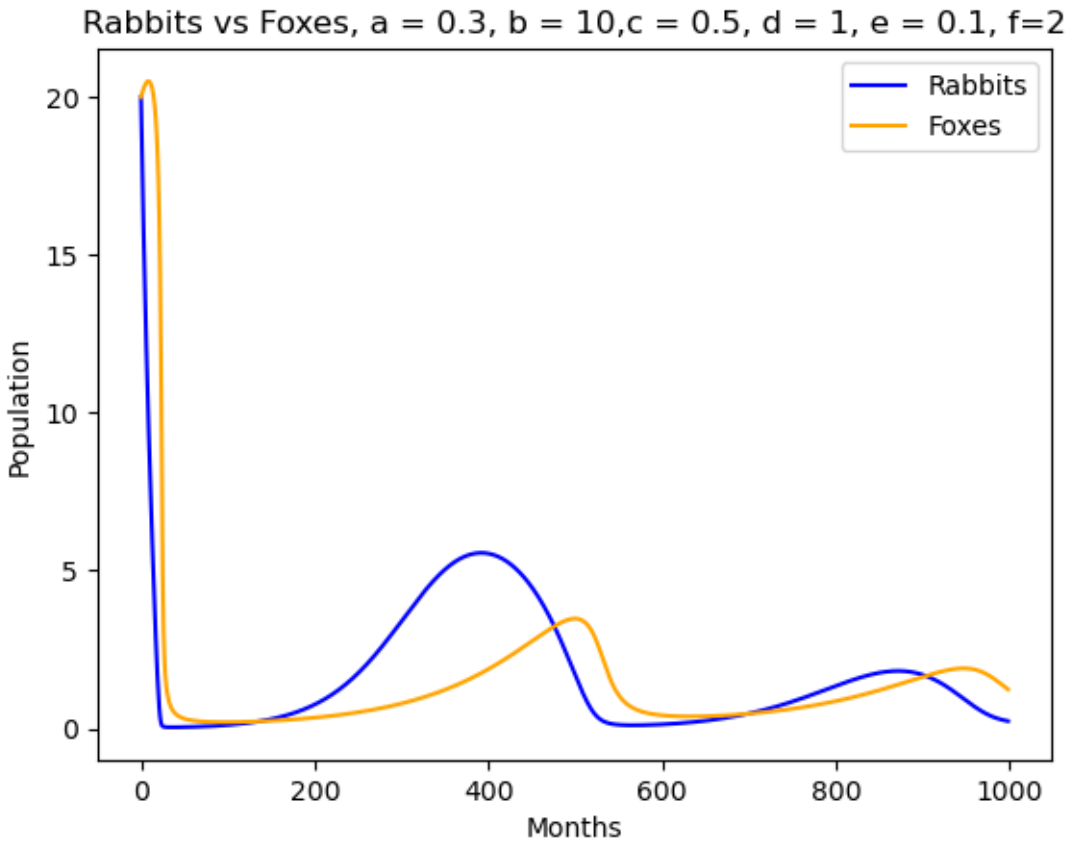

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.45, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

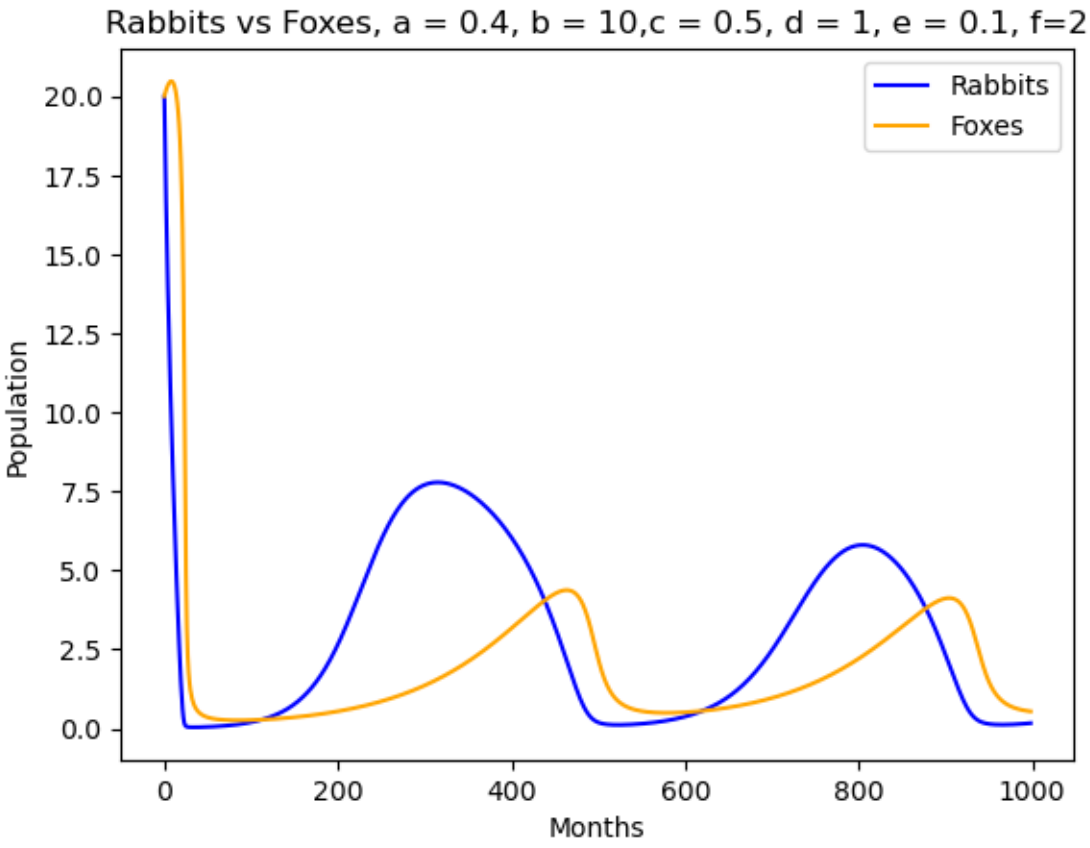

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.475, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

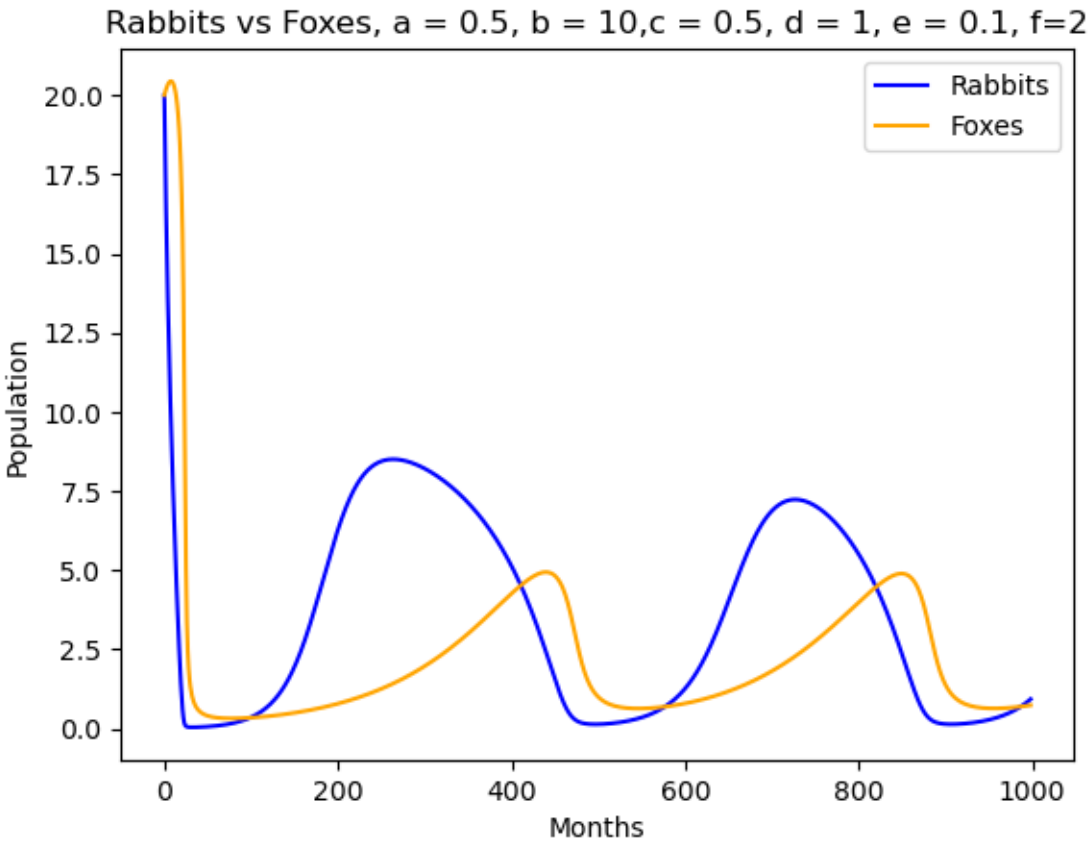

Rabbits vs Foxes, a = 0.0, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0
5
10
15
20
0
200
400
600
800
1000

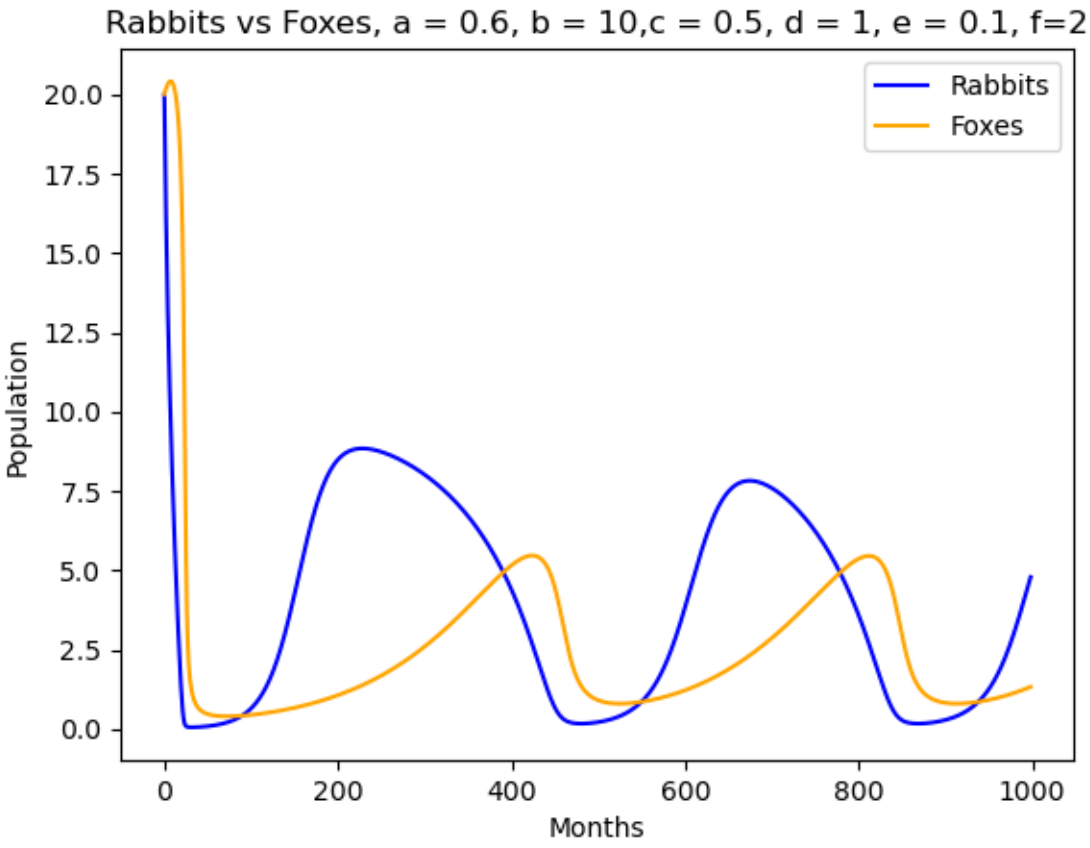

Rabbits vs Foxes, a = 0.1, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0
5
10
15
20
0
200
400
600
800
1000

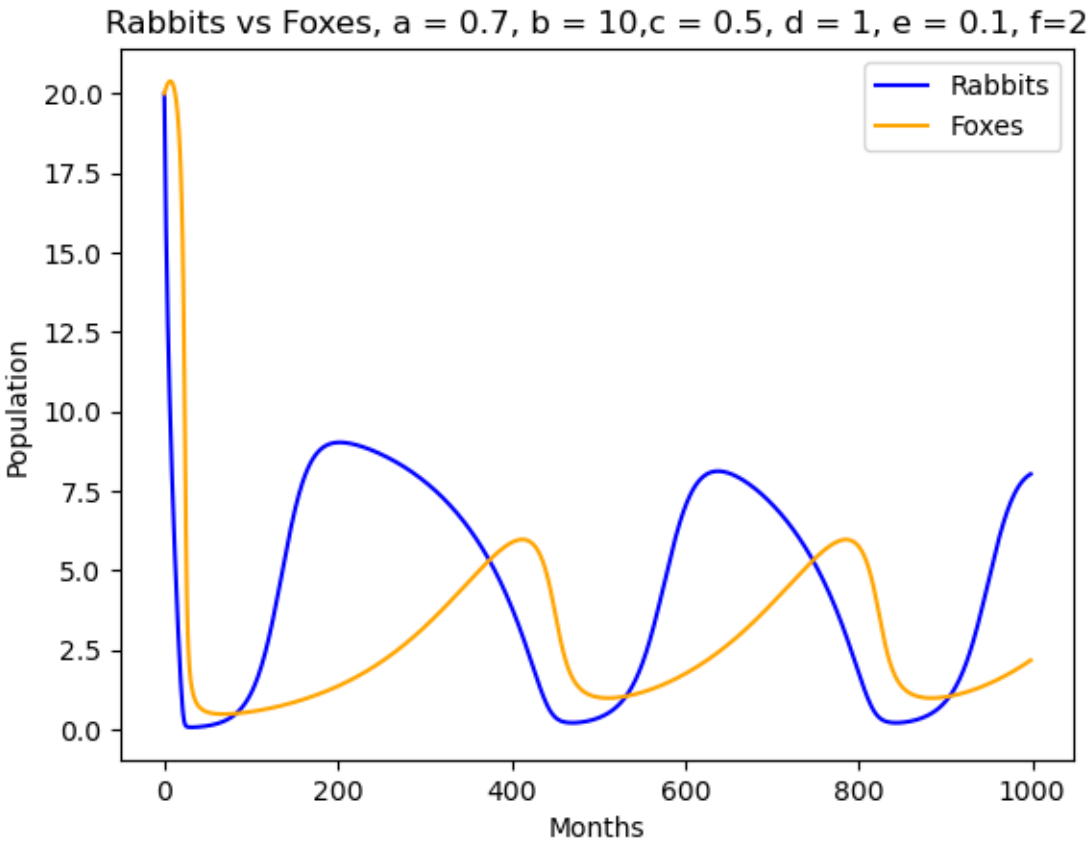

Rabbits vs Foxes, a = 0.2, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0
5
10
15
20
0
200
400
600
800
1000

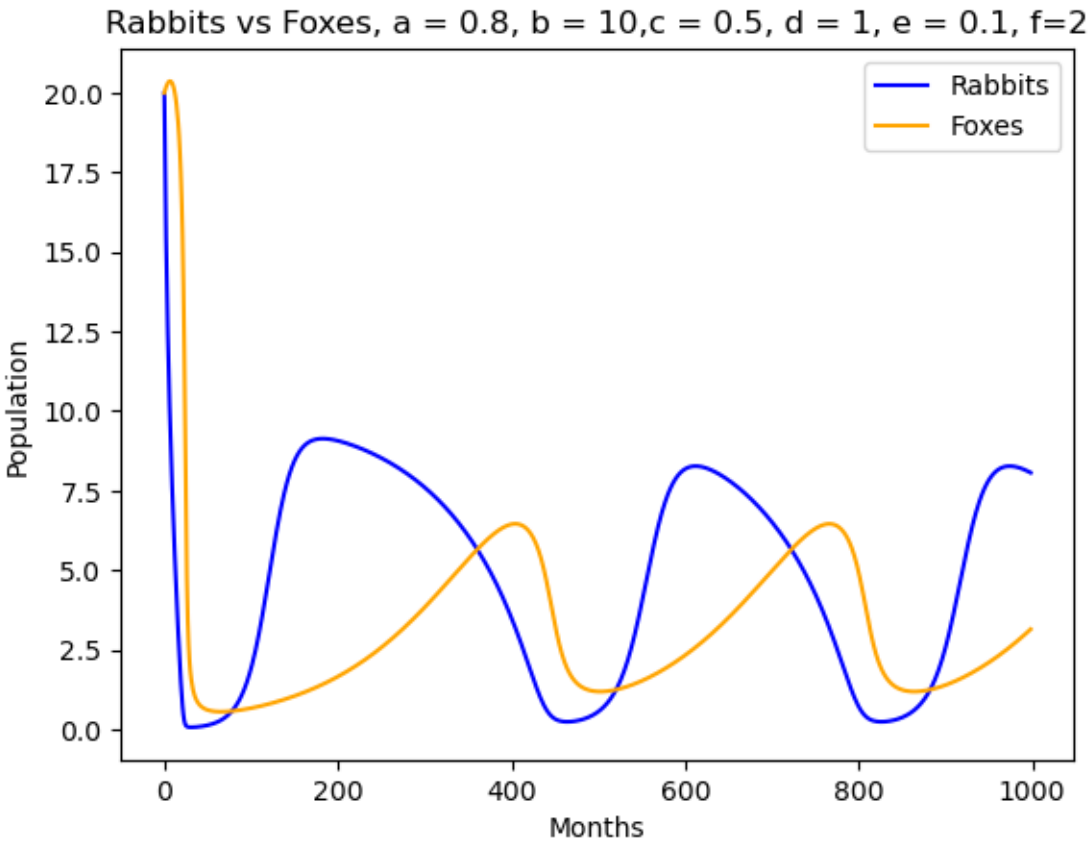

Rabbits vs Foxes, a = 0.3, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0
5
10
15
20
0
200
400
600
800
1000

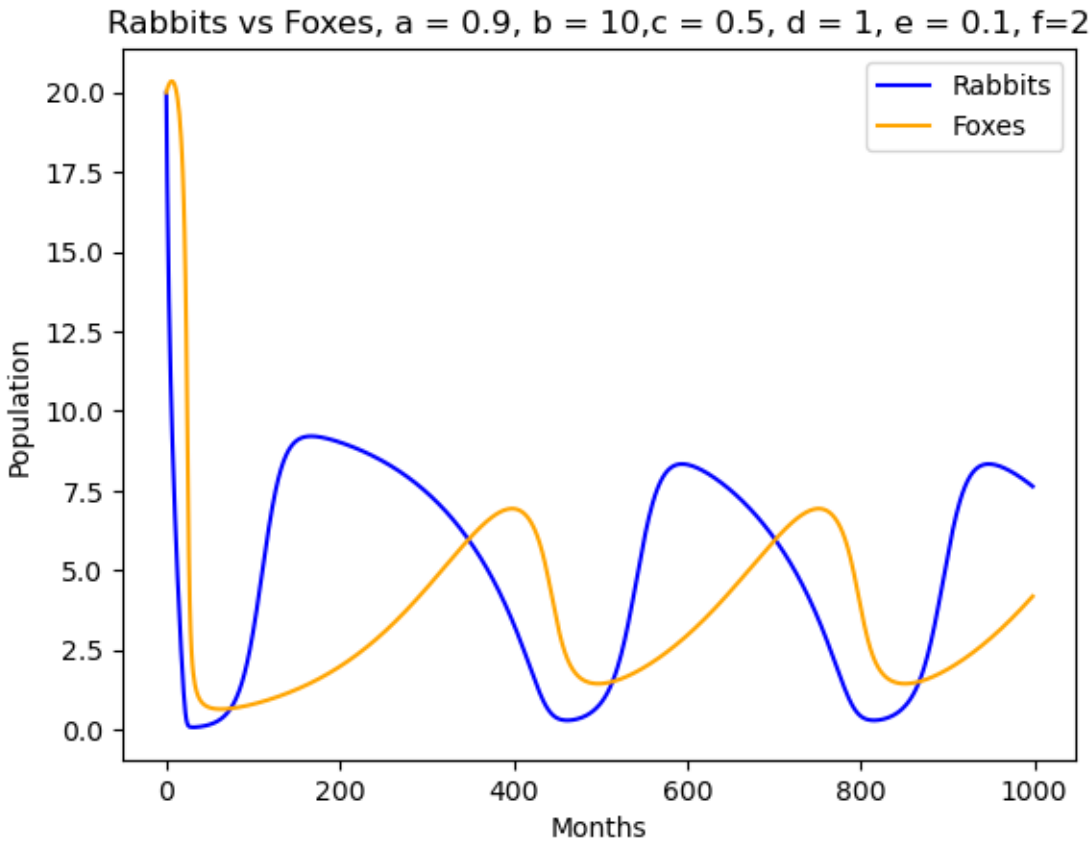

Rabbits vs Foxes, a = 0.4, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

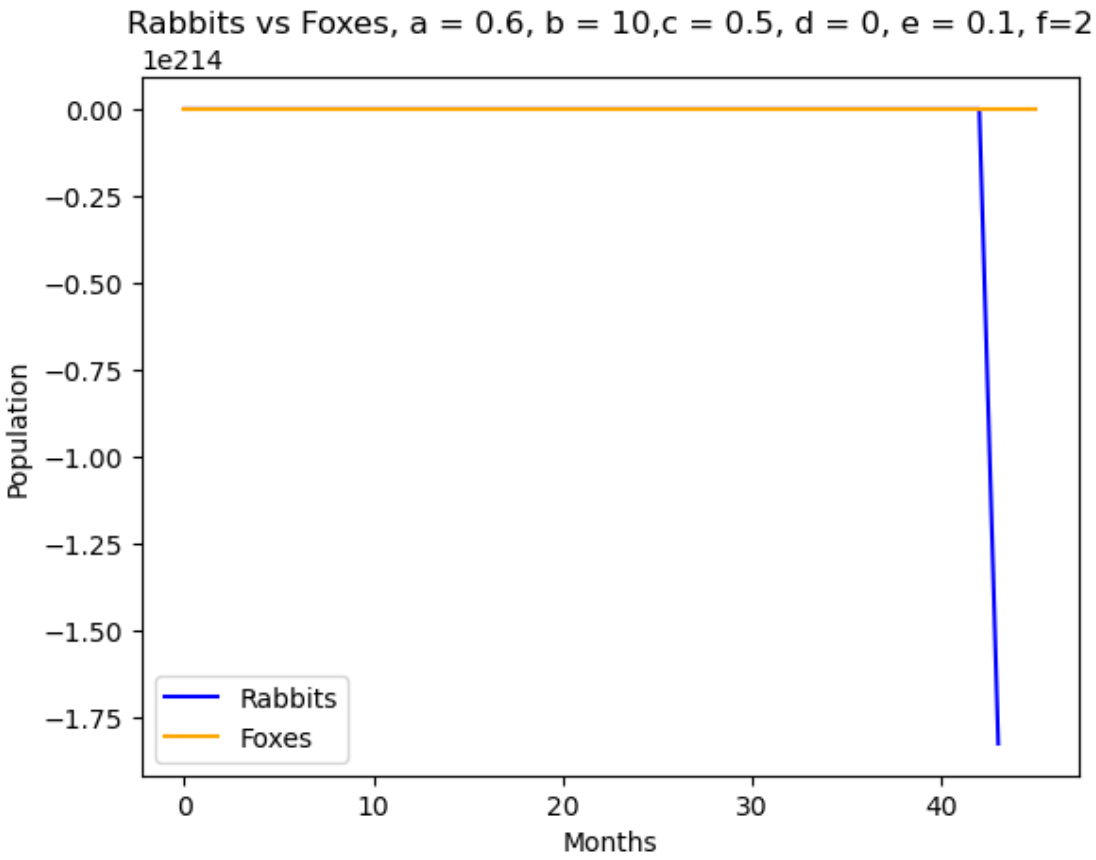

Rabbits vs Foxes, a = 0.5, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

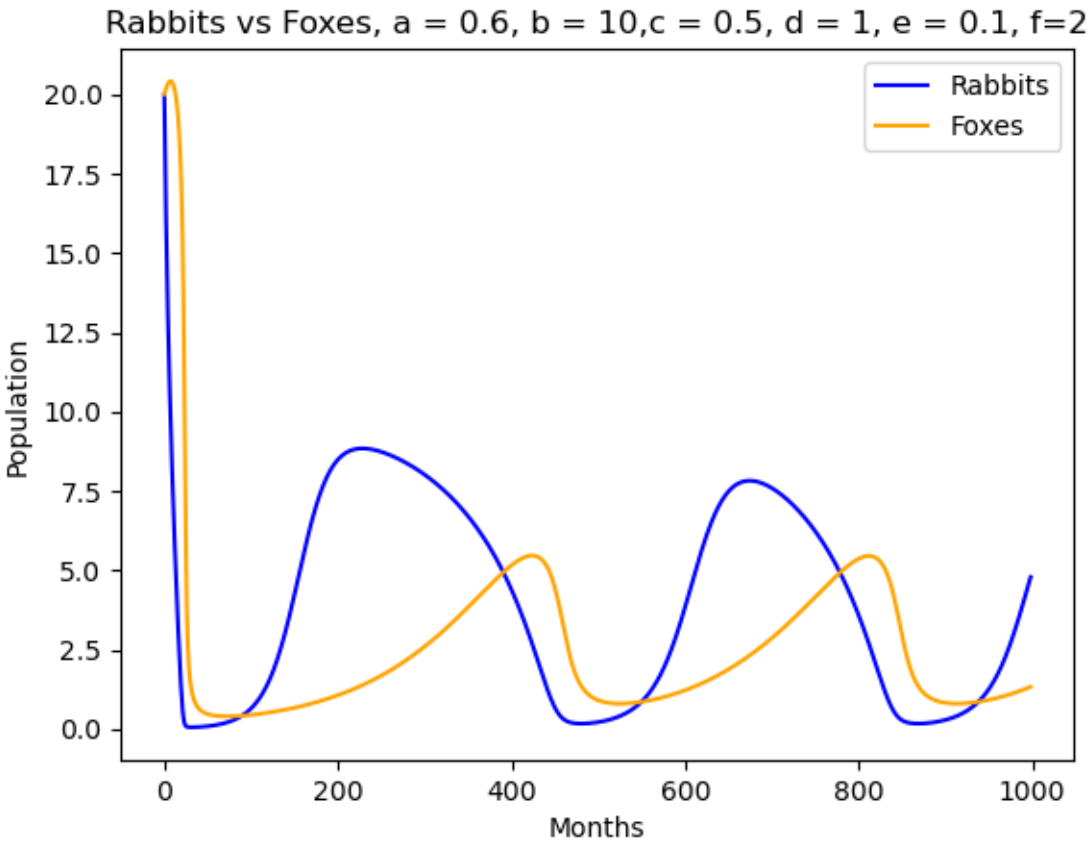

Rabbits vs Foxes, a = 0.6, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

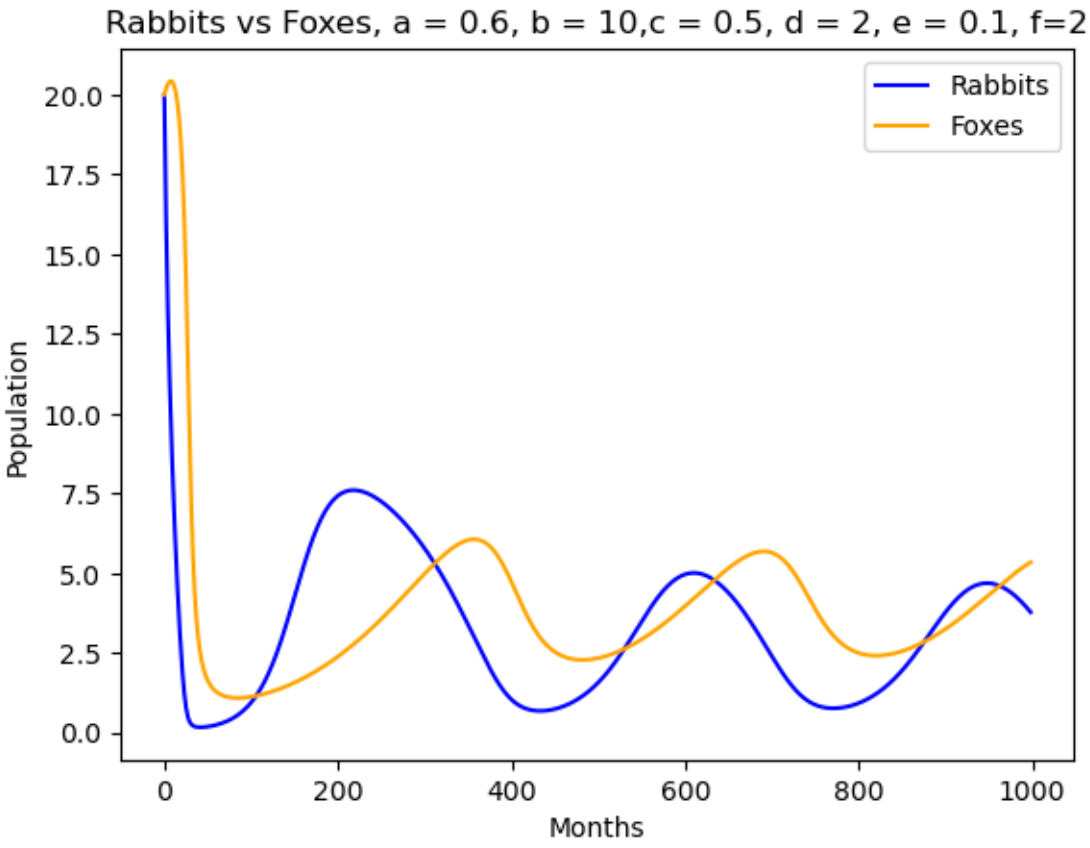

Rabbits vs Foxes, a = 0.7, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

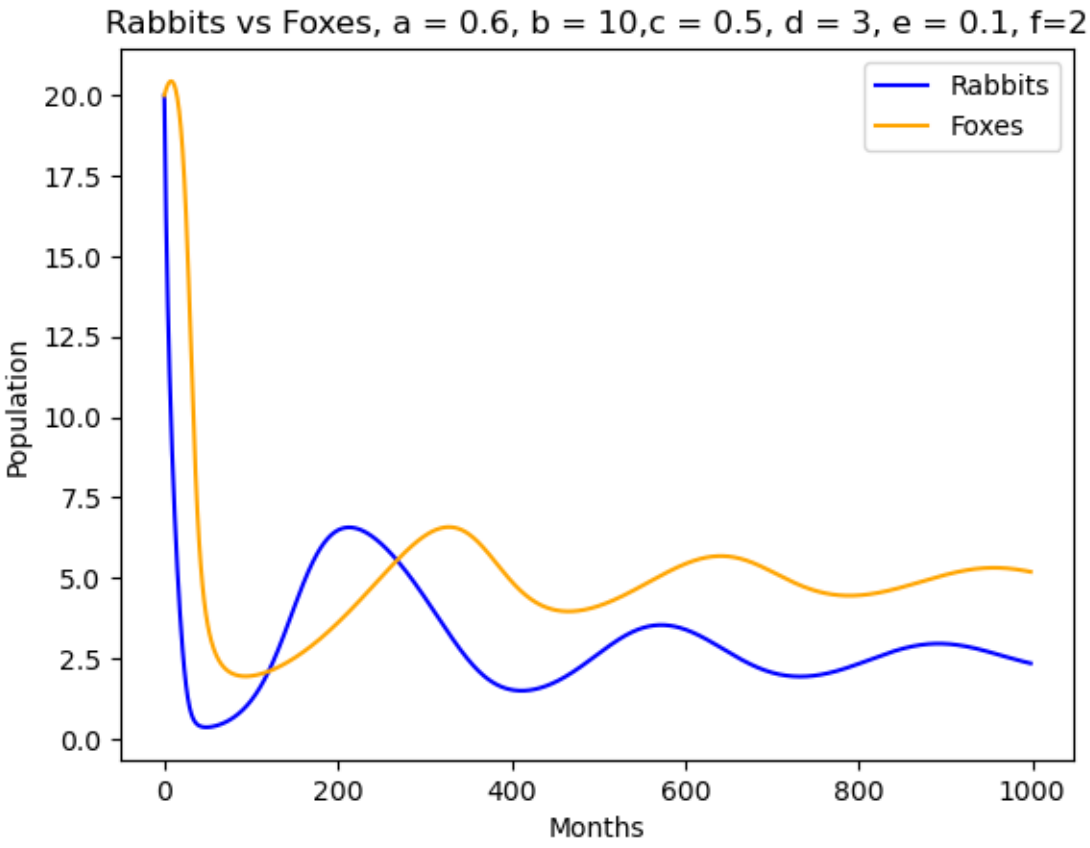

Rabbits vs Foxes, a = 0.8, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

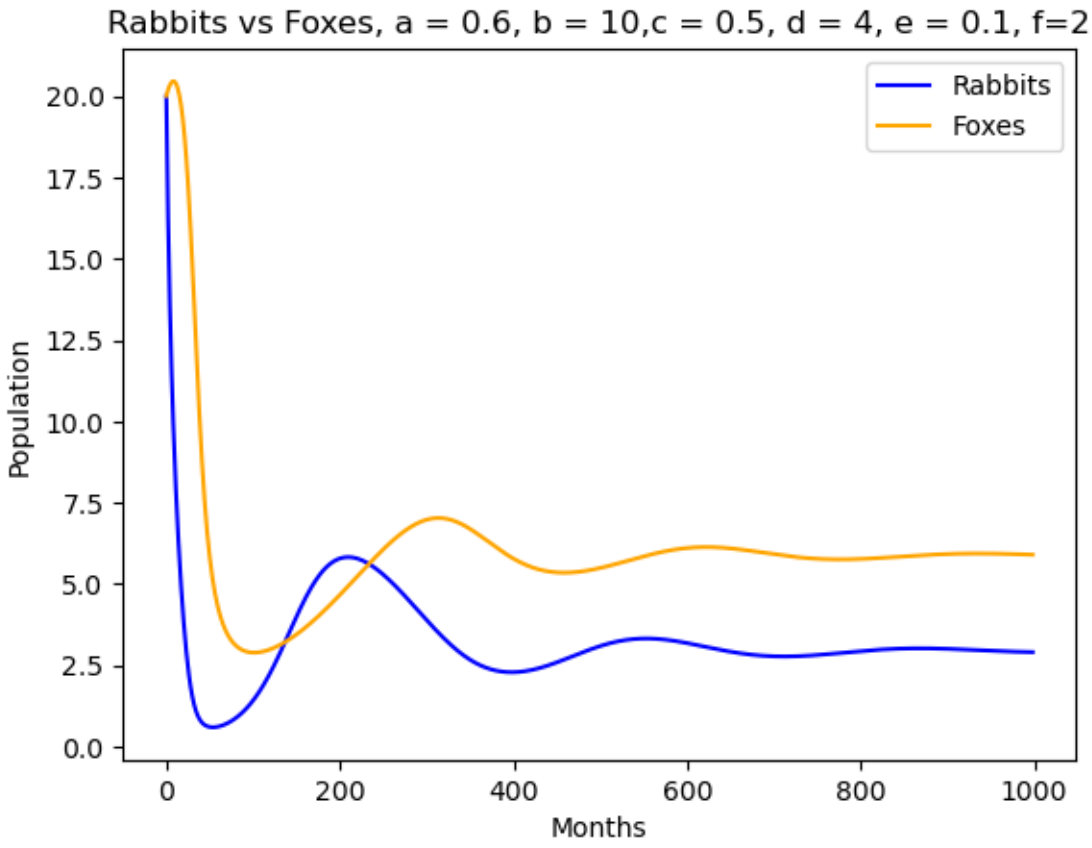

Rabbits vs Foxes, a = 0.9, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000


Rabbits vs Foxes, a = 0.6, b = 10,c = 0.5, d = 0, e = 0.1, f=2
1e214
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
Population
Rabbits
Foxes
0
10
20
30
40
Months


Rabbits vs Foxes, a = 0.6, b = 10,c = 0.5, d = 1, e = 0.1, f=2
Rabbits
Foxes
Population
Months
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000


Rabbits vs Foxes, a = 0.6, b = 10,c = 0.5, d = 2, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000


Rabbits vs Foxes, a = 0.6, b = 10,c = 0.5, d = 3, e = 0.1, f=2
Rabbits
Foxes
Population
Months
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000



Activity Two: Do all parts of 4.1 #8, #9, #10, #11 Fermentation. It's great to discover these equations on your own,however, please check with me and Piazza to make sure you've got them right.

1 8.

1.1 A.

In the first model assume that the sugar supply is not depleted, that no alcohol appears, and that the yeast simply grows logistically. Begin by adding 0.5 lb of yeast to a large vat of grape juice whose carrying capacity is 10 lbs of yeast. Assume that the natural growth rate of the yeast is 0.2 lbs of yeast per hour, per pound of yeast. Let Y (t) be the number of pounds of live yeast present after t hours; what differential equation describes the growth of Y?

We can model this scenario using the logistic function. The function describing the growth of Y will involve the growth rate of the yeast and the carrying capacity of the vat. So, the logistic equation is:

$$Y' = 0.2Y(1 - \frac{Y}{10})$$

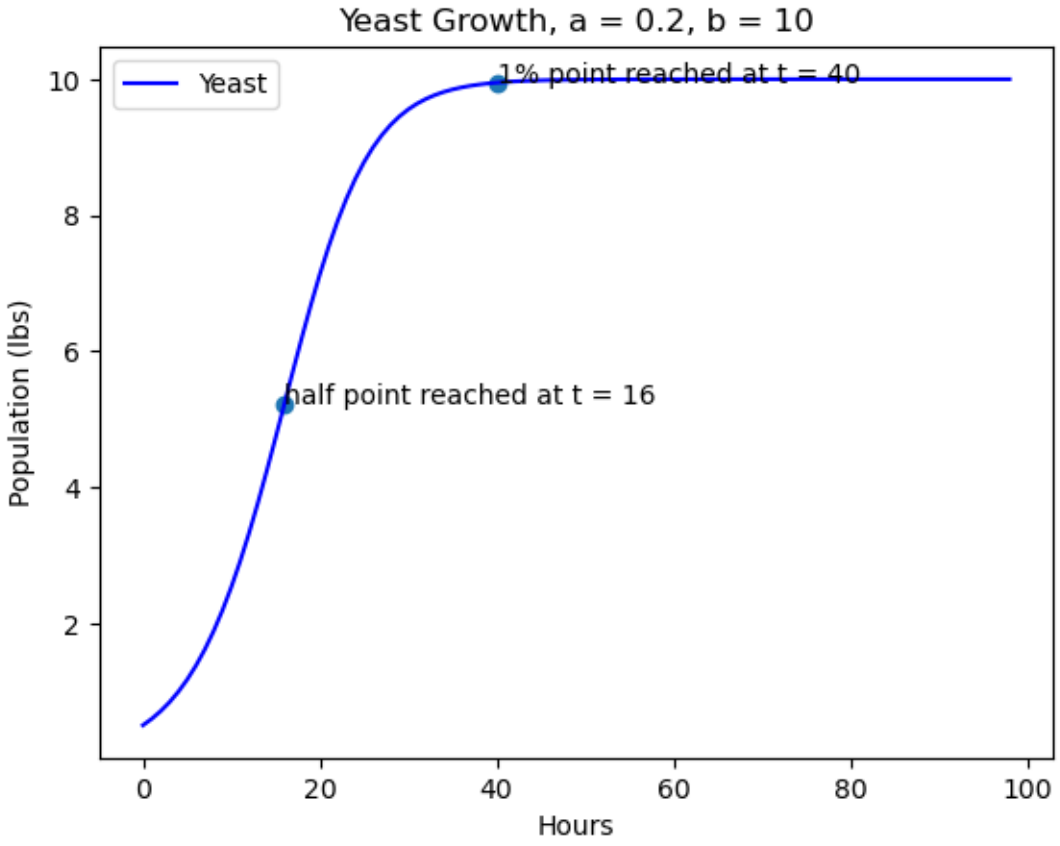
1.2 B.

Graph the solution Y (t), for example by using a suitable modification of the program SIRPLOT. Indicate on your graph approximately when the yeast reaches one-half the carrying capacity of the vat, and when it gets to within 1% of the carrying capacity.

[31]:
```
def YeastGrowth(Y0, tf, a0 = 0.2, b0 = 10, show_plot = True):
    t_initial = 0
    t = t_initial
    delta_t = 1
    Y = Y0

    a = a0
    b = b0

    Y_graph = []

    half_pt = (0, 0)
    onep_pt = (0, 0)

    # Iterating through the number of line segments in our plot (the more␣
↪lines, the smoother the graph)
    for k in range (1, tf*10):
        Y_graph.append(Y)
        if (Y >=(b0/2)-0.25) and (Y <= (b0/2)+0.25):
            print('half point found')
            half_pt = (t, Y)
        elif (Y >= (b0-(b0*0.01)) and Y < (b0-(b0*0.005))):
            onep_pt = (t, Y)


        Yprime = (a*Y) * (1-(Y/b))

        delta_Y = Yprime*delta_t

        t = t + delta_t
        Y = Y + delta_Y

    if show_plot==True:
        fig, ax = plt.subplots()
        ax.plot(Y_graph, color='blue', label='Yeast')
        ax.scatter([half_pt[0], onep_pt[0]], [half_pt[1], onep_pt[1]])
        ax.legend()
        ax.annotate(f'half point reached at t = {half_pt[0]}', half_pt)
        ax.annotate(f'1% point reached at t = {onep_pt[0]}', onep_pt)
        ax.set_title(f'Yeast Growth, a = {a}, b = {b}')
        ax.set_xlabel('Hours')
        ax.set_ylabel('Population (lbs)')
```

```
    return Y_graph

a8 = YeastGrowth(0.5, 10)
```

```
half point found
```



1.3 C.

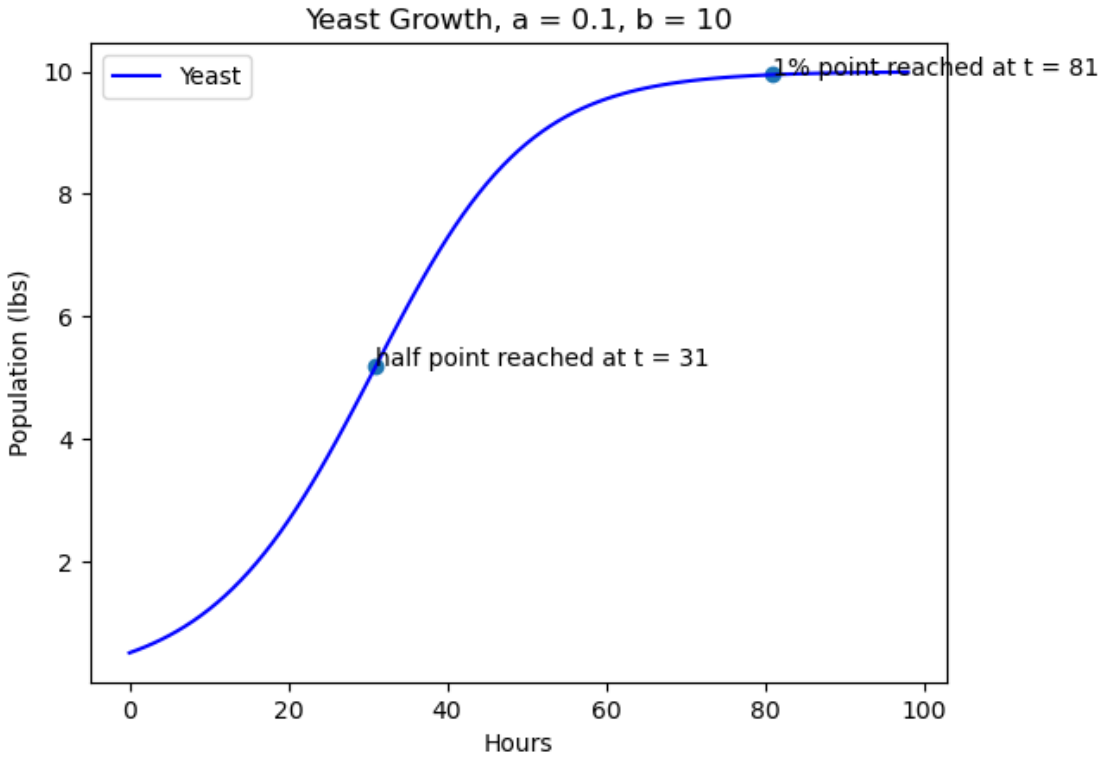
Suppose you use a second strain of yeast whose natural growth rate is only half that of the first strain of yeast. If you put 0.5 lb of this yeast into the vat of grape juice, when will it reach one-half the carrying capacity of the vat, and when will it get to within 1% of the carrying capacity? Compare these values to the values produced by the first strain of yeast: are they larger, or smaller? Sketch, on the same graph as in part (b), the way this yeast grows over time.

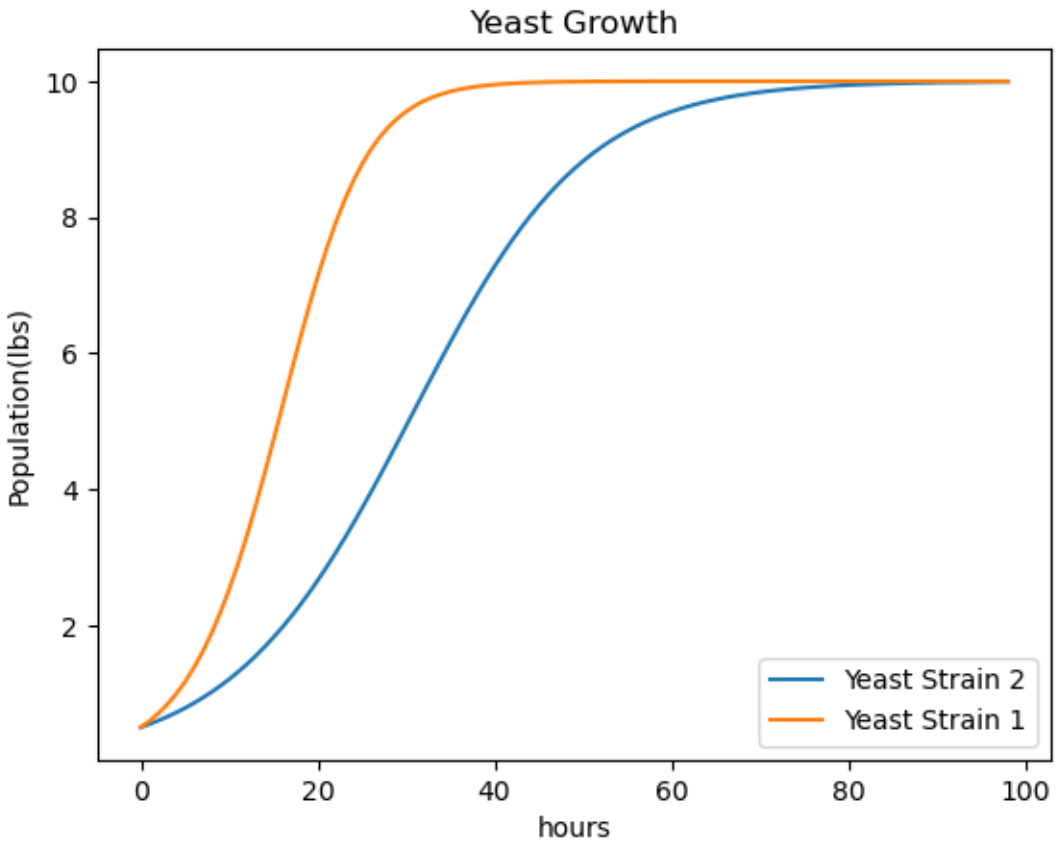
The second strain of yeast takes about twice as long as the first strain to reach the same proportions of the carrying capacity

[33]:
```
b8 = YeastGrowth(0.5, 10, a0 = 0.1);

fig, ax = plt.subplots()
```

```
ax.plot(b8, label = "Yeast Strain 2")
ax.plot(a8, label = "Yeast Strain 1")
ax.legend()
ax.set_title("Yeast Growth")
ax.set_ylabel("Population(lbs)")
ax.set_xlabel("hours")
plt.show()
```

```
half point found
half point found
```





2 9

2.1 a

Now consider how the yeast produces alcohol. Suppose that waste products are generated at a rate proportional to the amount of yeast present; specifically, suppose each pound of yeast produces 0.05 lbs of alcohol per hour. Let A(t) denote the amount of alcohol generated after t hours. Write a differential equation that describes the growth of A.

$$A' = 0.05Y$$

2.2 b

Consider the toxic effect of the alcohol on the yeast. Assume that yeast cells die at a rate proportional to the amount of alcohol present, and also to the amount of yeast present. Specifically, assume that, in each pound of yeast, a pound of alcohol will kill 0.1 lb of yeast per hour. Then, if there are Y lbs of yeast and A lbs of alcohol, how many pounds of yeast will die in one hour? Modify the original logistic equation for Y (strain 1) to take this effect into account. The modification involves subtracting off a new term that describes the rate at which alcohol kills yeast. What is the new differential equation?

$$Y' = 0.2Y(1 - \frac{Y}{10}) - 0.1AY$$

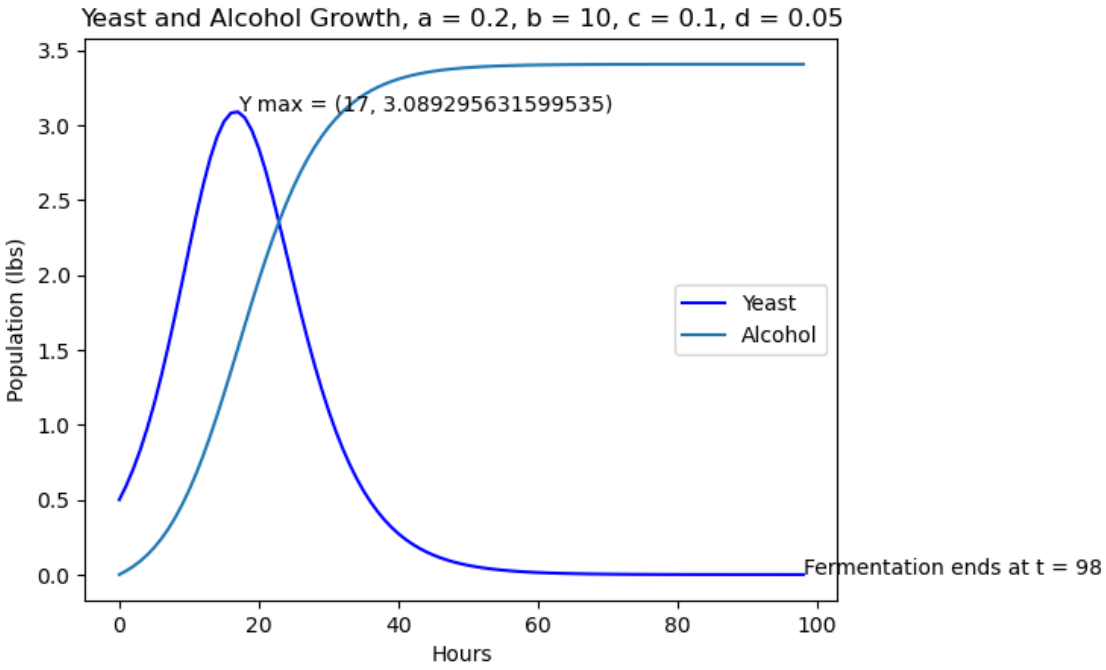
2.3 c

You should now have two differential equations describing the rates of growth of yeast and alcohol. The equations are coupled, in the sense that the yeast equation involves alcohol, and the alcohol equation involves yeast. Assuming that the vat contains, initially, 0.5 lb of yeast and no alcohol, describe by means of a graph what happens to the yeast. How close does the yeast get to carrying capacity, and when does this happen? Does the fermentation end? If so, when; and how much alcohol has been produced by that time? (Note that since Y will never get all the way to 0, you will need to adopt some convention like Y .01 to specify the end of fermentation.)

The yeast does not get close to the carrying capacity at all, only about 30% of the way there after 17 hours. The fermentation does end after 98 hours and 3.5lbs of alcohol have been produced at that point.

[60]:
```
def YeastAlcoholGrowth(Y0, A0, tf, a0 = 0.2, b0 = 10, c0 = 0.1, d0 = 0.05,␣
↪show_plot = True):
    t_initial = 0
    t = t_initial
    delta_t = 1

    Y = Y0
    A = A0

    a = a0
    b = b0
    c = c0
    d = d0

    Y_graph = []
    A_graph = []

    half_pt = (0, 0)
    onep_pt = (0, 0)
    Ymax = 0

    # Iterating through the number of line segments in our plot (the more␣
↪lines, the smoother the graph)
    for k in range (1, tf):
        Y_graph.append(Y)
        A_graph.append(A)

        if Y > Ymax:
            Ymax = Y
            Ymaxpt = (t, Y)
```

```
        if (Y >=(b0/2)-0.25) and (Y <= (b0/2)+0.25):
            print('half point found')
            half_pt = (t, Y)
        elif (Y < 0.1) :
            fend_pt = (t, Y)


        Yprime = ((a*Y) * (1-(Y/b))) - (c * A * Y)
        Aprime = (d * Y)

        delta_Y = Yprime*delta_t
        delta_A = Aprime*delta_t

        t = t + delta_t
        Y = Y + delta_Y
        A = A + delta_A

    if show_plot==True:
        fig, ax = plt.subplots()
        ax.plot(Y_graph, color='blue', label='Yeast')
        ax.plot(A_graph, label='Alcohol')
        ax.legend()
        #ax.annotate(f'half point reached at t = {half_pt[0]}', half_pt)
        ax.annotate(f'Fermentation ends at t = {fend_pt[0]}', fend_pt)
        ax.annotate(f'Y max = {Ymaxpt}', Ymaxpt)
        ax.set_title(f'Yeast and Alcohol Growth, a = {a}, b = {b}, c = {c}, d =␣
↪{d}')
        ax.set_xlabel('Hours')
        ax.set_ylabel('Population (lbs)')

    return Y_graph, A_graph

c9 = YeastAlcoholGrowth(0.5, 0, 100)
```



3 10

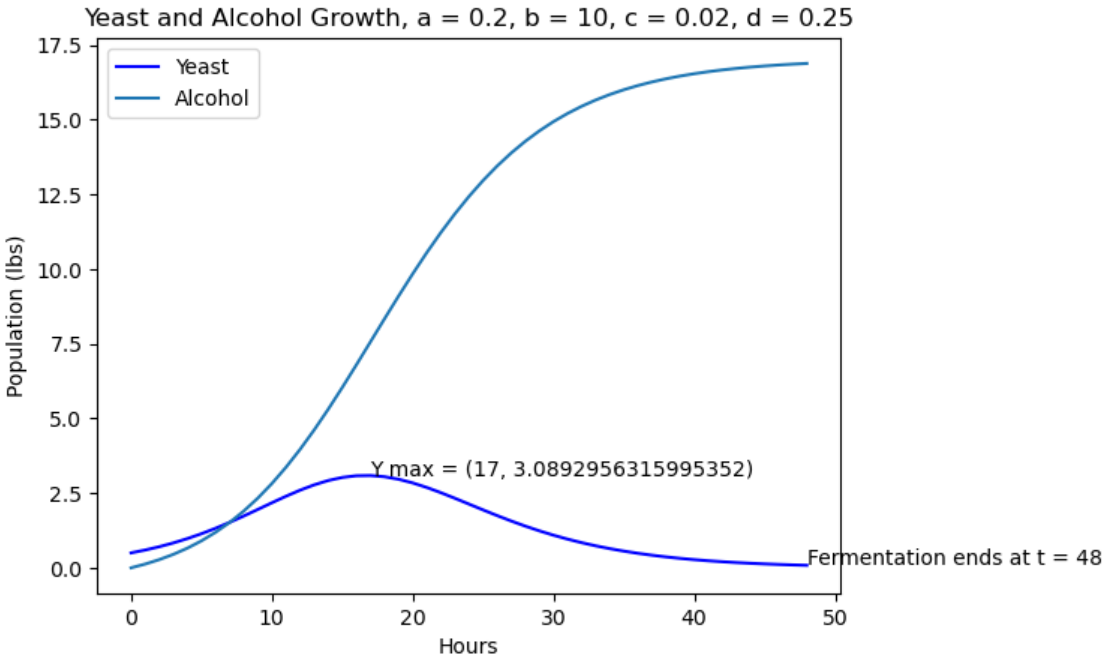
What happens if the rates of toxicity and alcohol production are different? Specifically, increase the rate of alcohol production by a factor of five—from 0.05 to 0.25 lbs of alcohol per hour, per pound of yeast—and at the same time reduce the toxicity rate by the same factor—from 0.10 to 0.02 lb of yeast per hour, per pound of alcohol and pound of yeast. How do these changes affect the time it takes for fermentation to end? How do they affect the amount of alcohol produced? What happens if only the rate of alcohol production is changed? What happens if only the toxicity rate is reduced?

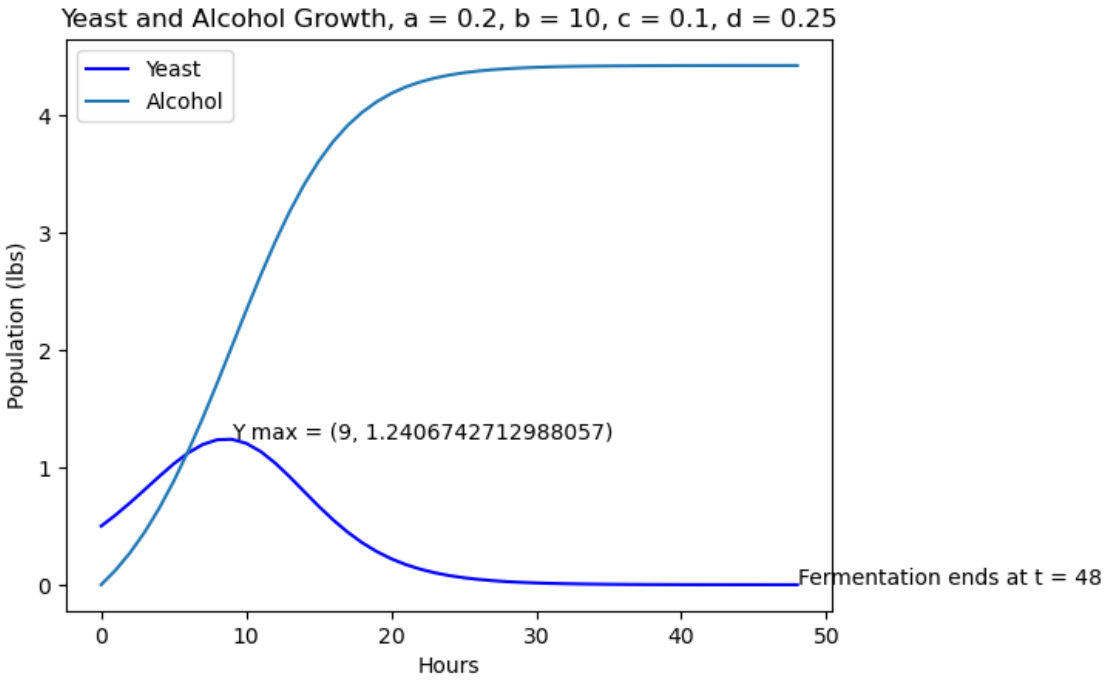
Fermentation ends at half the time the fermentation with the original constants took. Much more alcohol is produced, despite the maximum amount of yeast staying the same. When only the rate of alcohol production is increased to 0.25, the fermentation ends at the same time, but the yeast reaches it's max population faster, and less alcohol is produced overall. When only the toxicity rate is reduced, the fermentation takes much longer, the amount of total alcohol is reduced, and the maximum amount of yeast is increased

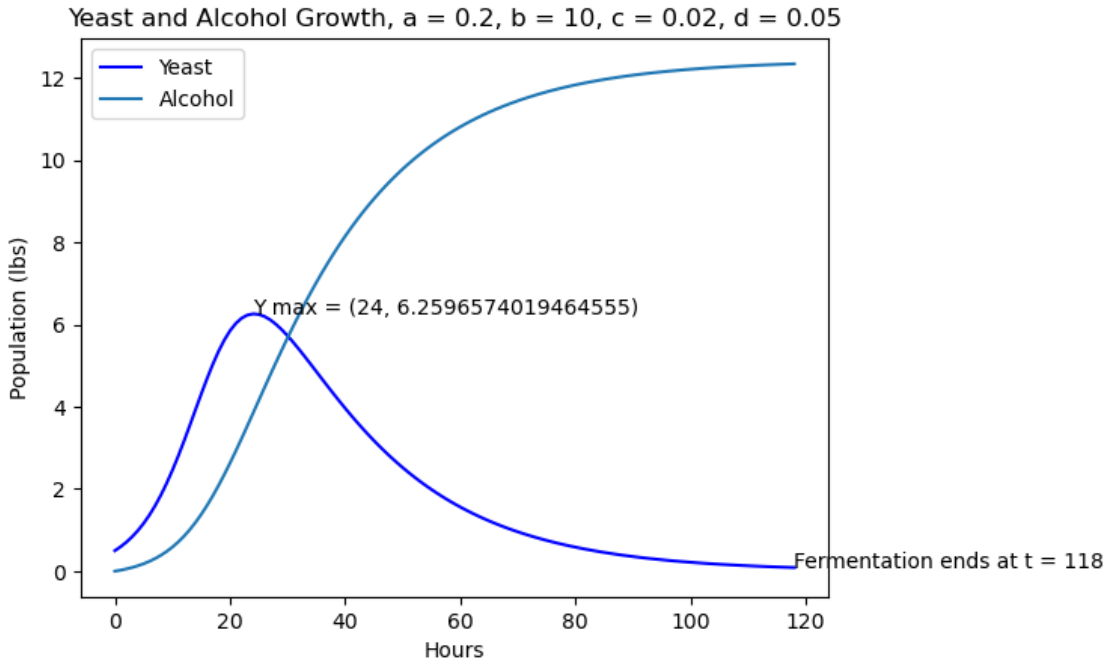
[65]:
```
o10 = YeastAlcoholGrowth(0.5, 0,50, c0 = 0.02, d0 = 0.25)
a10 = YeastAlcoholGrowth(0.5, 0, 50, c0 = 0.1, d0 = 0.25) #alcohol production␣
↪changed
b10 = YeastAlcoholGrowth(0.5, 0, 120, c0 = 0.02, d0 = 0.05) #toxicity changed
```

```
half point found
half point found
half point found
```

```
half point found
```







4 11

4.1 a

The third model will take into account that the sugar in the grape juice is consumed. Suppose the yeast consumes .15 lb of sugar per hour, per lb of yeast. Let S(t) be the amount of sugar in the vat after t hours. Write a differential equation that describes what happens to S over time.

$$S' = -0.15 * Y * S$$

4.2 b

Since the carrying capacity of the vat depends on the amount of sugar in it, the carrying capacity must now vary. Assume that the carrying capacity of S lbs of sugar is .4 S lbs of yeast. How much sugar is needed to maintain a carrying capacity of 10 lbs of yeast? How much is needed to maintain a carrying capacity of 1 lb of yeast? Rewrite the logistic equation for yeast so that the carrying capacity is .4 S lbs, instead of 10 lbs, of yeast. Retain the term you developed in 9.b to reflect the toxic impact of alcohol on the yeast

if the carrying capacity is 10, then the amount of sugar needed to maintain the carrying capacity is

$$10 = 0.4S$$

$$S = 25$$

For a carrying capacity of 1lb of yeast then, the amount of sugar is $S = 1/0.4 = 2.5$ *The new logistic equation for* Y *is written below:*

$$Y' = 0.2Y(1 - \frac{Y}{0.4 * S}) - 0.1AY$$

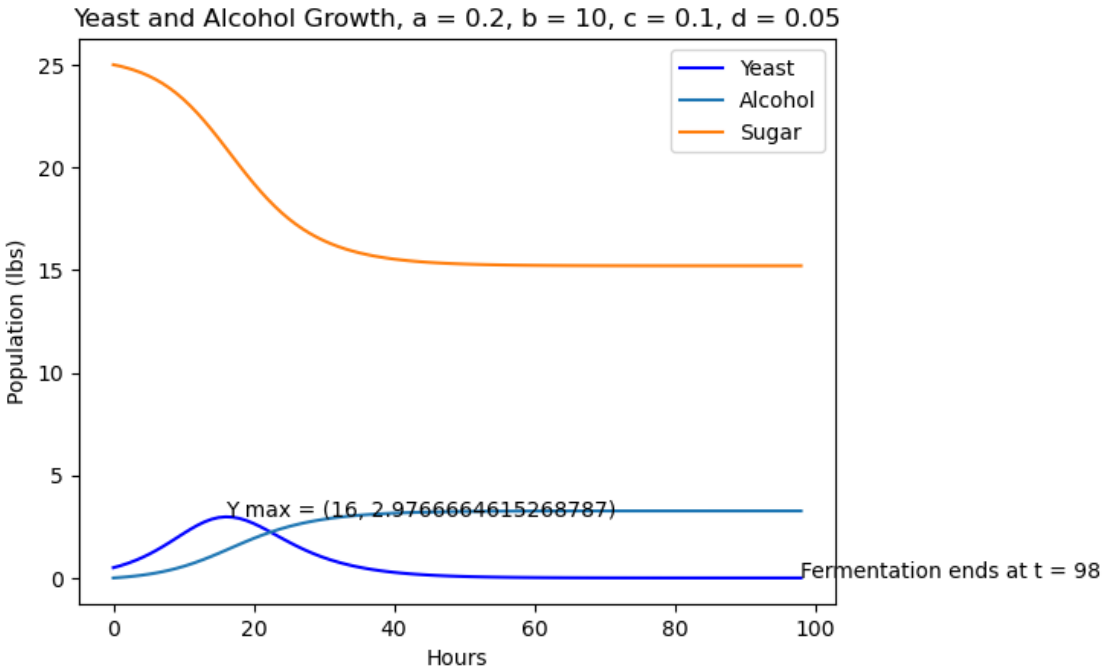
4.3 c

There are now three differential equations. Using them, describe what happens to .5 lbs of yeast that is put into a vat of grape juice that contains 25 lbs of sugar at the start. Does all the sugar disappear? Does all the yeast disappear? How long does it take before there is only .01 lb of yeast? How much sugar is left then? How much alcohol has been produced by that time?

Not all of the sugar disappears in this scenario, because there is not enough yeast to eat it! It takes about 98 hours for the yeast to decrease to 0.01 lbs, and for fermentation to end. At this point, the amount of sugar has stabilized at around 15 lbs, and about 3 lbs of alcohol have been produced.

[77]:
```
def YASGrowth(Y0, A0, S0, tf, a0 = 0.2, b0 = 10, c0 = 0.1, d0 = 0.05, e0 = 0.4,␣
↪f0 = 0.15, show_plot = True):
    t_initial = 0
    t = t_initial
    delta_t = 1

    Y = Y0
    A = A0
    S = S0

    a = a0
    b = b0
    c = c0
    d = d0
    e = e0
    f = f0

    Y_graph = []
    A_graph = []
    S_graph = []

    half_pt = (0, 0)
    onep_pt = (0, 0)
    Ymax = 0

    # Iterating through the number of line segments in our plot (the more␣
↪lines, the smoother the graph)
    for k in range (1, tf):
        Y_graph.append(Y)
        A_graph.append(A)
        S_graph.append(S)

        if Y > Ymax:
            Ymax = Y
            Ymaxpt = (t, Y)

        if (Y >=(b0/2)-0.25) and (Y <= (b0/2)+0.25):
```

```
            print('half point found')
            half_pt = (t, Y)
        elif (Y < 0.01) :
            fend_pt = (t, Y)


        Yprime = ((a*Y) * (1-(Y/(e * S)))) - (c * A * Y)
        Aprime = (d * Y)
        Sprime = (-f * Y)

        delta_Y = Yprime*delta_t
        delta_A = Aprime*delta_t
        delta_S = Sprime*delta_t


        t = t + delta_t
        Y = Y + delta_Y
        A = A + delta_A
        S = S + delta_S

    if show_plot==True:
        fig, ax = plt.subplots()
        ax.plot(Y_graph, color='blue', label='Yeast')
        ax.plot(A_graph, label='Alcohol')
        ax.plot(S_graph, label='Sugar')
        ax.legend()
        #ax.annotate(f'half point reached at t = {half_pt[0]}', half_pt)
        ax.annotate(f'Fermentation ends at t = {fend_pt[0]}', fend_pt)
        ax.annotate(f'Y max = {Ymaxpt}', Ymaxpt)
        ax.set_title(f'Yeast and Alcohol Growth, a = {a}, b = {b}, c = {c}, d =␣
↪{d}')
        ax.set_xlabel('Hours')
        ax.set_ylabel('Population (lbs)')

    return Y_graph, A_graph, S_graph

c11 = YASGrowth(0.5, 0, 25, 100)
```



Activity Three: Choose any of the models from the book to expand upon and develop. This does not have to be super complex, please scale it to the time you have available. + The simple Lotka-Volterra model from #7 is easy to expand to a complex ecosystem with MANY equations (mice, foxes, bears, aliens, and can accomodate a Zombie Apocolypse) + Expand SIR, add complexities with immunity, other diseases (try Chicken Pox and Shingles together),add deaths, more variables, or compare various epidemic responses. + Check out the examples in 4.1 #12-#14(Newton's Cooling, can you see applications to crime scene investigations?) + Check out 4.1 #19-#24 from more ideas + If you did this activity in Linear Algebra and/or want a challenge, look up research papers about modeling with differential equations and try to reproduce the model - this could be a start for a final project.

4.4 Newtons Law of Cooling

I decided to expand upon the model presented in the book by showing what happens to the temperature of the liquid when the environment it is placed in cools progressively, as though left outside while the sun was setting. I also added a term to see the rate at which a solid (like sugar) dissolves in the liquid and changes the cooling constant k of the liquid.

The original equation for the temperature Q is

$$-0.1(Q - 20)$$

, where $k = 0.1$ and $T = 20$ in degrees Celcius. Although the book does account for temperature changes in a warming container, it does not account for an environment that is also cooling and not being warmed by the mug. In this case, the set of equations is :

$$Q' = -a(Q - A)$$

$$A' = -b(A - c)$$

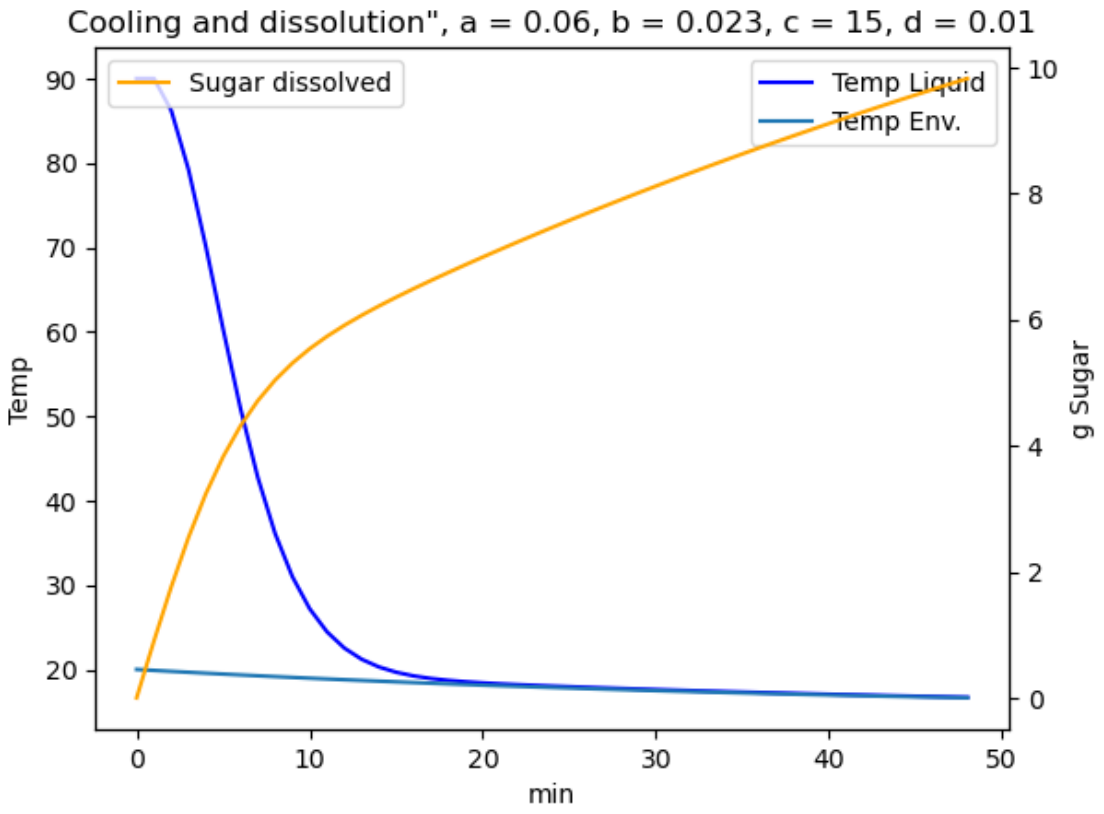
where c is the minimum temperature. I also wanted to add a third equation that changes the cooling constant k based on how much sugar is dissolved in it. In this set of equations, the rate of sugar dissolving is proportional to the temperature of the liquid in the mug and how much sugar is left in the mug, and a constant d that represents how many grams of sugar per minute proportional to the temperature can be dissolved. The equation for how much sugar dissolves should be logistic as well, approaching the limit of the initial amount of sugar added e So, our final set of equations is:

$$Q' = (-aS)(Q - A)$$
$$A' = -b(A - c)$$
$$S' = dQ * (1 - \frac{S}{e})$$

The code modeling these equations is shown below:

[164]:
```
def Cooling(Q0, A0, S0, tf, a0 = 0.06, b0 = 0.023, c0 = 15, d0 = 0.01, e0 = 20,␣
↪show_plot = True):
    t_initial = 0
    t = t_initial
    delta_t = 1

    Q = Q0
    A = A0
    S = S0

    a = a0 #how much sugar changes the cooling constant
    b = b0 #cooling constant of air
    c = c0 #minimum temperature to be reached by environment
    d = d0 #Rate at which a liquid of temperature Q can dissolve x grams of␣
↪sugar in one minute
    e = e0 #e initial amount of undissolved sugar added to cup

    Q_graph = []
    A_graph = []
    S_graph = []


    # Iterating through the number of line segments in our plot (the more␣
↪lines, the smoother the graph)
    for k in range (1, tf):
        Q_graph.append(Q)
        A_graph.append(A)
        S_graph.append(S)


        Qprime = (-a*S) * (Q-A)
```

```
        Aprime = -b*(A-c)
        Sprime = (d * Q) * (1-(S/e))

        delta_Q = Qprime*delta_t
        delta_A = Aprime*delta_t
        delta_S = Sprime*delta_t


        t = t + delta_t
        Q = Q + delta_Q
        A = A + delta_A
        S = S + delta_S

    if show_plot==True:
        fig, ax = plt.subplots()
        ax2 = ax.twinx()
        ax.plot(Q_graph, color='blue', label='Temp Liquid')
        ax.plot(A_graph, label='Temp Env.')
        ax2.plot(S_graph, color = "orange", label='Sugar dissolved')
        ax.legend()
        ax2.legend()

        ax.set_title(f'Cooling and dissolution", a = {a}, b = {b}, c = {c}, d =␣
↪{d}')
        ax.set_xlabel('min')
        ax.set_ylabel('Temp')
        ax2.set_ylabel('g Sugar')
        plt.show()

    return Q_graph, A_graph, S_graph

Cooling(90, 20, 0, 50);
```


Cooling and dissolution", a = 0.06, b = 0.023, c = 15, d = 0.01
Sugar dissolved
Temp Liquid
Temp Env.
Temp
g Sugar
min
90
80
70
60
50
40
30
20
10
8
6
4
2
0
0
10
20
30
40
50

[ ]: