Assignment 2
FIT3139

Daniel Nguyen
32471033

Part 1 - Model Selection

Model used: Lotka-Volterra predator prey model

Continuous Time Equations:
The differential equations that represent the Lotka-Volterra model can be broken down as such:

Prey population change over time = Prey births - Prey deaths - Prey eaten by predators

$$\frac{dx}{dt} = bx - dx - axy$$

x = number of prey at the current time
b = prey birth rate
d = prey death rate
a = proportional constant of prey being eaten by predators

(b - d) can be simplified to r:

$$\frac{dx}{dt} = rx - axy$$

x = number of prey at the current time
r = prey growth rate
a = proportional constant of prey being eaten by predators

Pred. population change over time = Pred. Births - Pred. deaths + Existing pred. that ate enough prey

$$\frac{dy}{dt} = b'y - d'y + \beta xy$$

b’ = predator birth rate
d’ = predator death rate
β = proportional constant of predators that have eaten enough prey to survive

Simplifying (b’-d’) = s:

$$\frac{dy}{dt} = sy + \beta xy$$

y = number of predators at the current time
s = predator rate of growth
β = proportional constant of predators that have eaten enough prey to survive

Thus our models are:

$\frac{dx}{dt} = rx - axy$ for prey and

$\frac{dy}{dt} = sy + \beta xy$ for predator

Model Extension:

Lets assume some species for our base model, such as spiders (the predator), and flies (the prey). How would the introduction of a new species (frogs) that preys on both spiders and fish affect the ecosystem? In particular, the original model allows for coexistence of both prey and predator in a cyclical nature. Is this still possible for all species with the introduction of the new species?

Our new continuous model can be constructed as such:

$\frac{dx}{dt} = rx - \theta xy - axz$ for species x (flies)

Where:

r = natural growth rate of species x

θ = proportional constant of species x eaten by species y

a = proportional constant of species x eaten by species z

$\frac{dy}{dt} = sy - \beta xz + \Phi yx$ for species y (spiders)

Where:

s = natural growth rate of species y

β = proportional constant of species y eaten by species z

Φ = proportional constant of species y that have eaten species x to survive

$\frac{dz}{dt} = mz + \lambda zy + \rho xz$ for species z (frogs)

Where:

m = natural growth rate of species z

c = carrying capacity of species z

λ = proportional constant of species z that have eaten species y to survive

ρ = proportional constant of species z that have eaten species x to survive

And:

x = population of species x

y = population of species y

z = population of species z

The affects on population of all species can be explained by the new equations, as they factor in the growth rate of each species, in addition to growth in population caused by survival through the eating of the other species and the decline in population caused by being eaten by the other species.

Part 2 - Discrete time analysis

We can create difference equations from each of our continuous equations as such:

$X(t + 1) = X(t) + \Delta t(rX(t) - \theta X(t)Z(t) - aX(t)Y(t))$ for species x (flies)

Where:
r = natural growth rate of species x
θ = proportional constant of species x eaten by species z
a = proportional constant of species x eaten by species y

$Y(t + 1) = Y(t) + \Delta t(sY(t) - \beta Y(t)Z(t) + \Phi Y(t)X(t))$ for species y (spiders)

Where:
s = natural growth rate of species y
β = proportional constant of species y eaten by species z
Φ = proportional constant of species y that have eaten species x to survive

$Z(t + 1) = Z(t) + \Delta t(mZ(t) + \lambda Z(t)Y(t) + \rho Z(t)X(t)))$ for species z (frogs)

Where:
m = natural growth rate of species z
λ = proportional constant of species z that have eaten species y to survive
ρ = proportional constant of species z that have eaten species x to survive

And:
Δt = time difference between iterations
$X(t + 1)$ = Population of species x at next time period
$X(t)$ = Population of species x at current time period
$Y(t + 1)$ = Population of species y at next time period
$Y(t)$ = Population of species y at current time period
$Z(t + 1)$ = Population of species z at next time period
$Z(t)$ = Population of species z at current time period

Modelling:

I have chosen Δt values to clearly emphasise the discrete nature of these models and to highlight differences between our continuous ODE and difference equation, as modelling a continuous ODE using computer software is essentially constructing a discrete model with extremely low step size. I have included tables of all system parameters for each graph in the appendix, although I have mentioned the relevant parameters in each figure discussion.

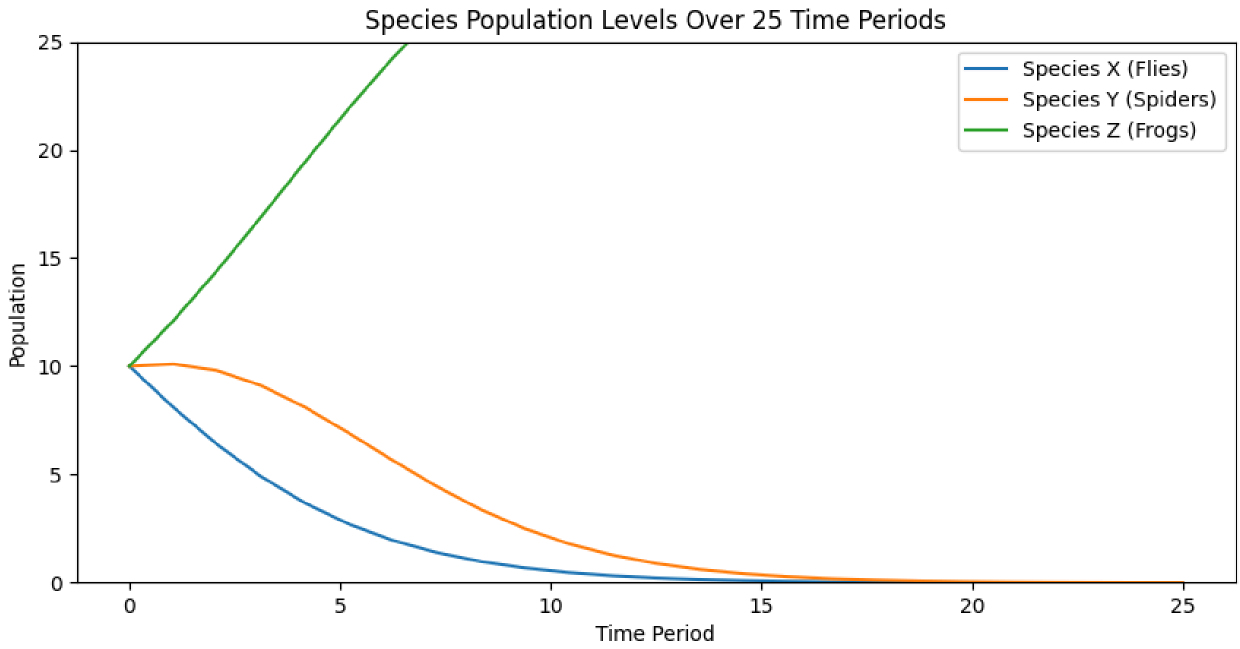


Fig. 1

When all starting conditions are the same for each species with positive growth rates, we see that both species with prey elements are eventually hunted to extinction.

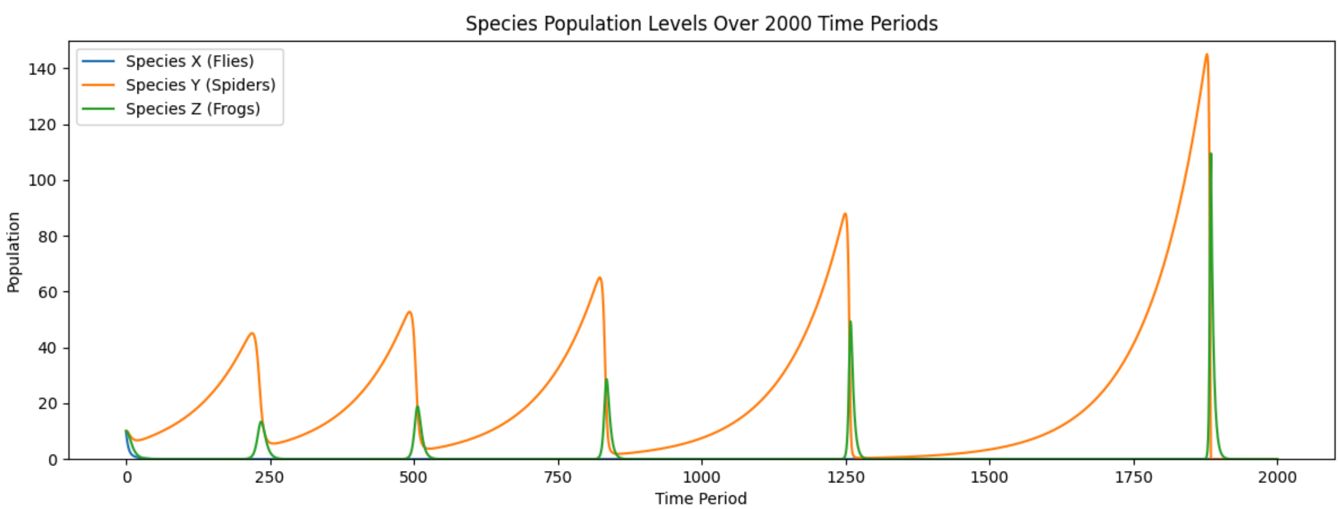


Fig 2.

When the intrinsic growth rate of the frog species is sufficiently negative (in this case -0.2), we see a quick extinction of flies, followed by increasingly amplified skewed oscillations of the remaining 2 species, until eventually spiders are hunted to extinction, followed quickly by an extinction of frogs resulting from a lack of prey.

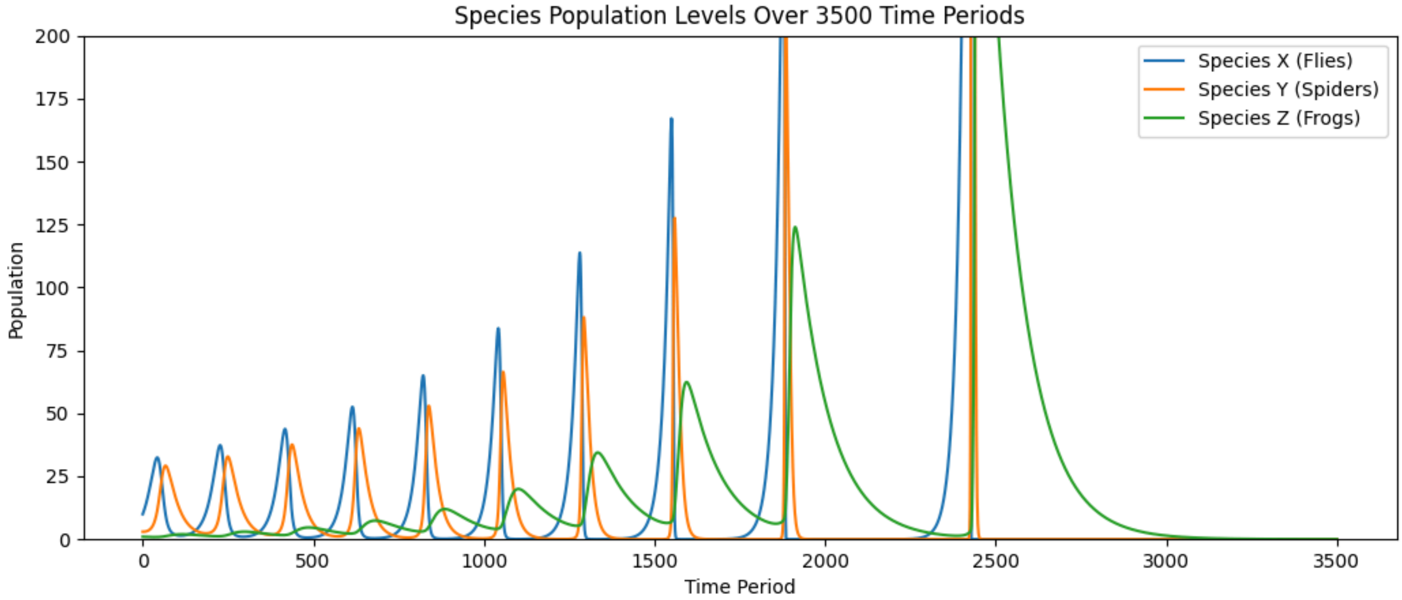


Fig 3.

When constructing the model with more realistic parameters, characterised by negative growth rates for spiders and frogs, differing starting populations, and with a lower step size (lower Δt), we can see that all species are able to survive until the growing amplitude of the oscillations causes the extinction of both flies and spiders, and then frogs via starvation. The specific survival and death parameters are also detailed in the appendix.

To truly show the difference that a change in step size makes, lets look at the same model parameters in figure 3, but with a Δt of 0.001:

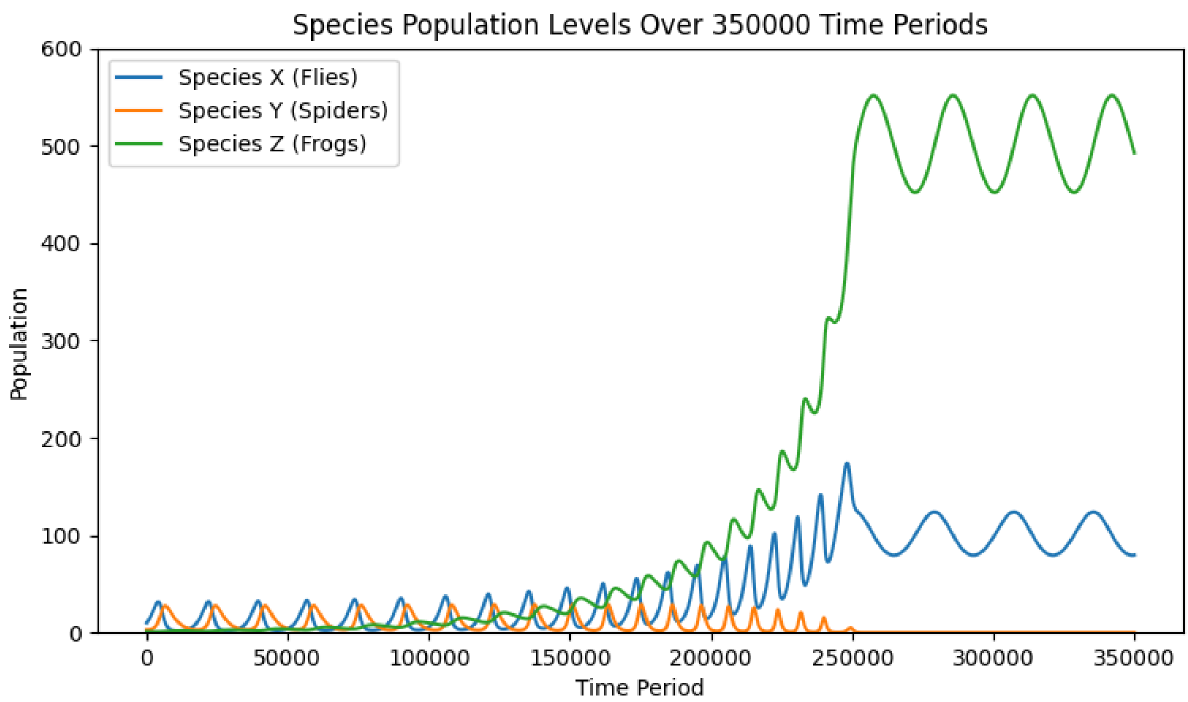


Figure 4.

We can see that as we increase the accuracy of the model, new behaviours emerge. In this case, the model parameters are the same as in figure 3, but we see that instead of resulting in the extinction of all species, only the extinction of spiders occurs, and the population of flies and frogs oscillates around an equilibrium.

General Conclusions:

Overall, the introduction of the new “apex predator” species into the ecosystem has serious effects on animal populations. Often, the extinction of one or multiple species will be caused.

Although there are initial conditions which foster the coexistence of pairs of species, I was not able to find model parameter values that resulted in a stable equilibrium in which all 3 species were able to survive indefinitely in a discrete time model. Although these are the conclusions I have made from a graphical analysis of the discrete difference equations, they may not actually be totally accurate, as discrete time analysis using eulers method creates error, and may not show the true picture of the system dynamics when compared to more accurate modelling, which will be explored in part 3.

Part 3 - Continuous time analysis

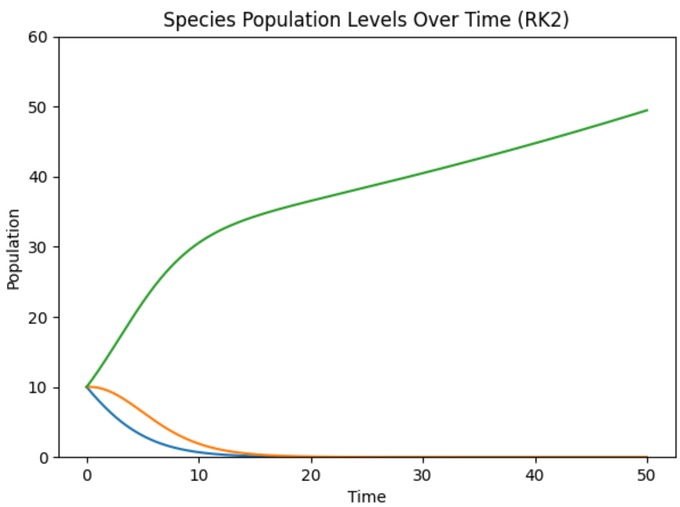
As detailed in Part 1, our continuous time ODEs are the following:

$$\frac{dx}{dt} = rx - \theta xy - axz \quad \text{for species x (flies)}$$

$$\frac{dy}{dt} = sy - \beta yz + \Phi yx \quad \text{for species y (spiders)}$$

$$\frac{dz}{dt} = mz + \lambda zy + \rho xz \quad \text{for species z (frogs)}$$

Now lets look at some models with parameters similar to our discrete analysis.



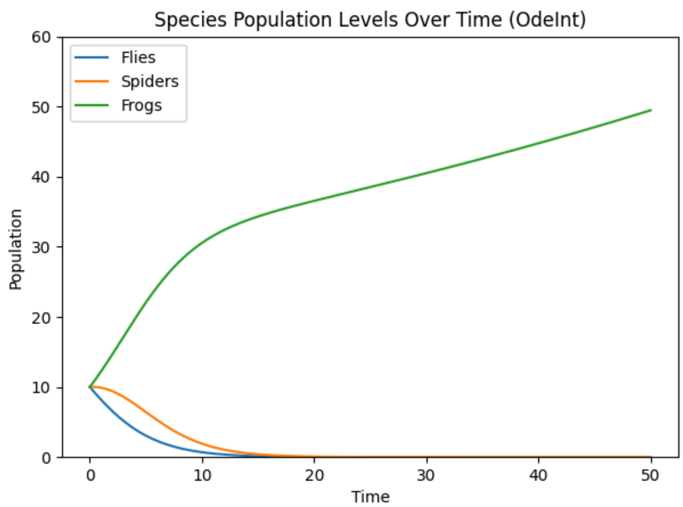


Fig 5.

With the same system parameters as Fig1, we see a similar graph to our discrete model, with unrestrained growth of Frogs after hunting the other species till extinction.

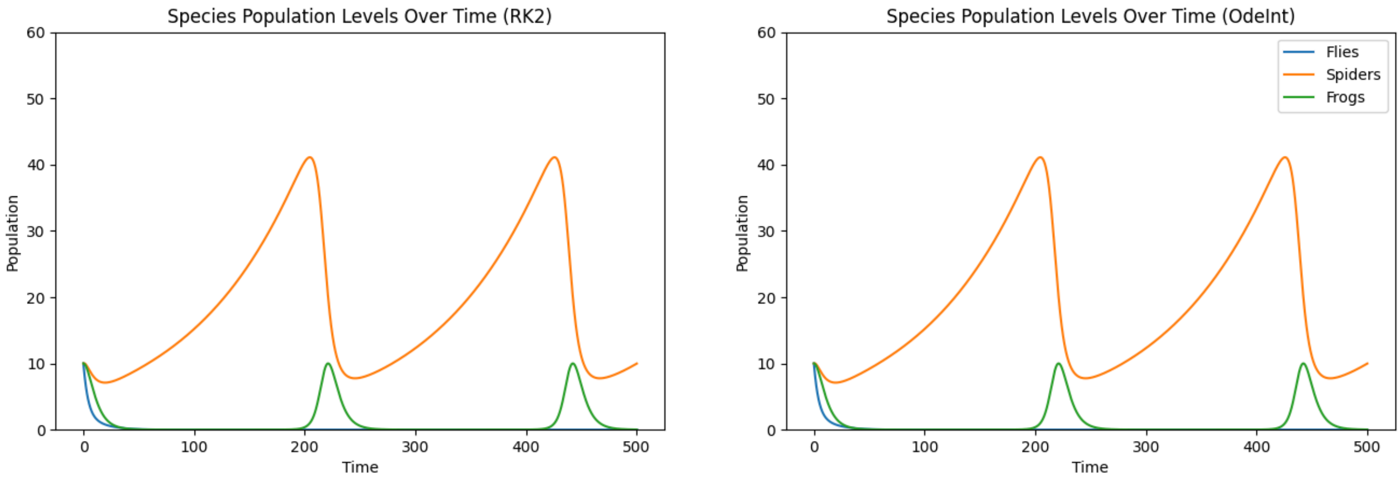


Fig 6.

Using the same system parameters as Fig2, we see a similar graph, but instead of increasing amplitude every cycle, we see that the amplitude is the same, and continues forever after the extinction of flies.

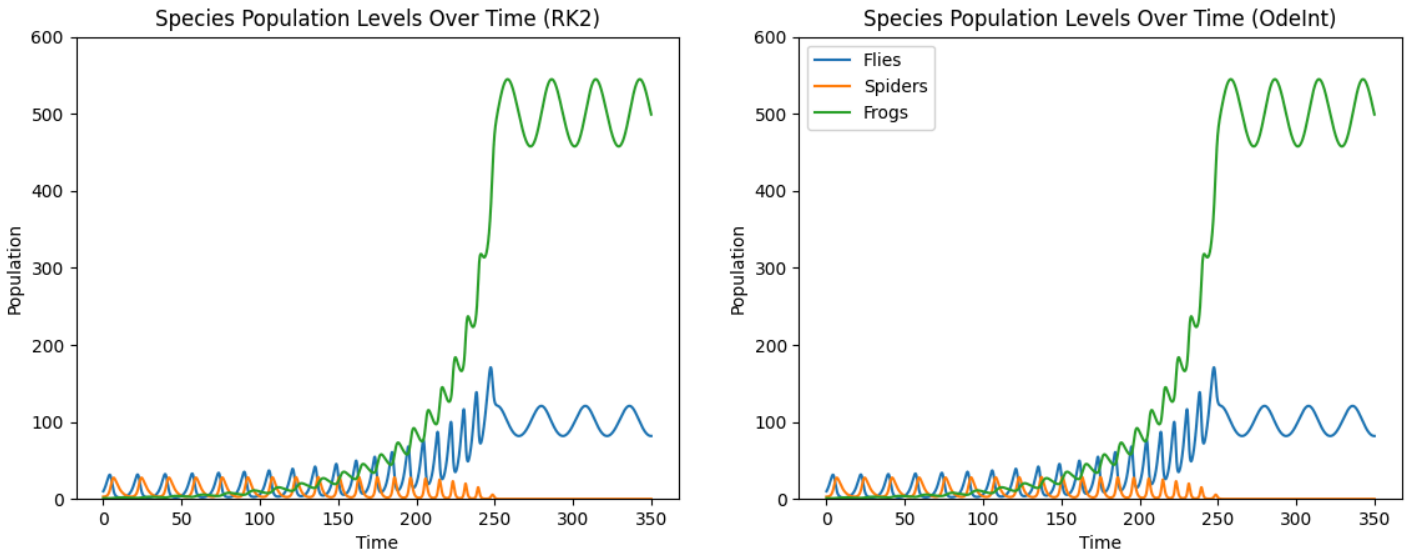


Fig 7.

We see a very similar graph to our discrete model with very small step size, seen in Fig5, where after the slow decline of spiders until extinction, flies and frogs live in equilibrium.

But What about Steady States?

When the equation for each differential equation is set to 0, and solved for x, y and z using pythons SymPy library, the set of values are the steady states for the system. Each steady state except 1 requires the population of either one or all species to be 0. The final steady state seems to allow for all 3 species to live in equilibrium. This steady state is:

X = $(-\ a\lambda s\ +\ \beta\lambda r\ +\ \beta m\theta)\ /\ (a\lambda\Phi\ -\ \beta\rho\theta)$
Y = $-\ (am\Phi\ -\ a\rho s\ +\ \beta r\rho)\ /\ (a\lambda\Phi\ -\ \beta\rho\theta)$
Z = $(\lambda\Phi r\ +\ m\Phi\theta\ -\ \rho s\theta)\ /\ (a\lambda\Phi\ -\ \beta\rho\theta)$

We can substitute in values to these equations, and look for positive x y and z values to find equilibrium population values for flies, spiders and frogs. One such example is

r = 0.5
θ = 0.045
a = 0.001

s = 0.5
β = 0.01
Φ = 0.4

m = -0.3
λ = 0.1
ρ = 0.001

Which results in equilibrium values of:

X = 7.964601769911503
Y = 2.920353982300884
Z = 368.58407079646025

The graph below in Figure 8 shows this equilibrium.

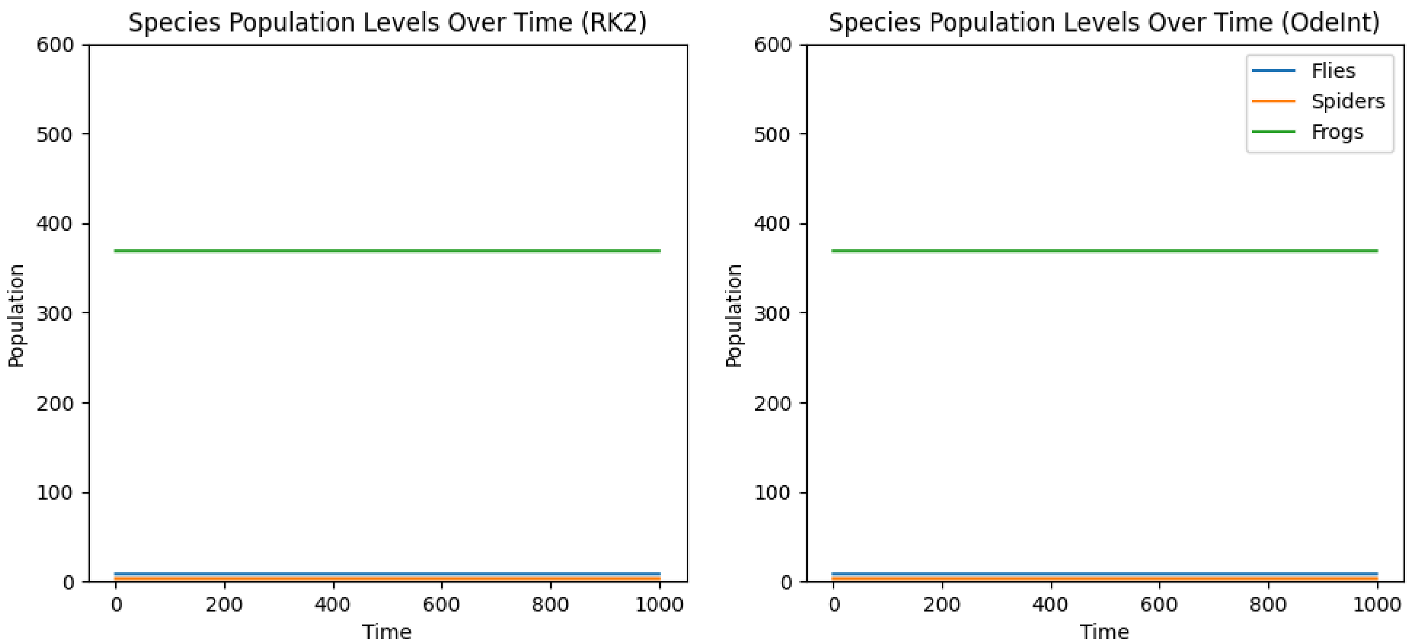
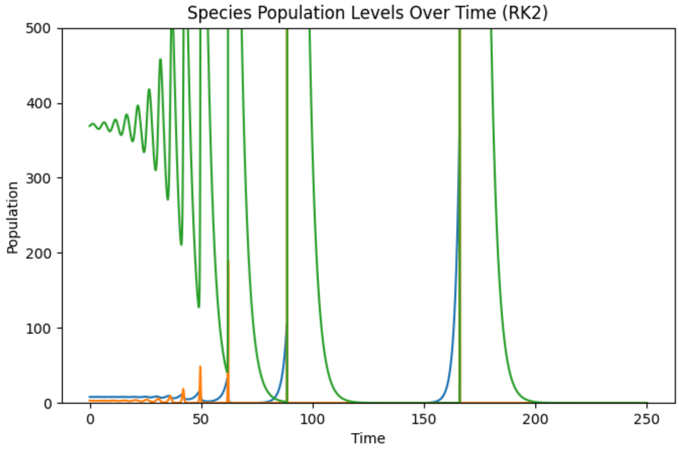


Fig 8.

We can see a constant level of population for each of the species if we set all model parameters to the steady state.

This however, unless engineered, is not something that would occur naturally, so what happens if we slightly perturb the equilibrium by even slightly changing our initial populations? Lets simply round them to the nearest whole number and see what happens.



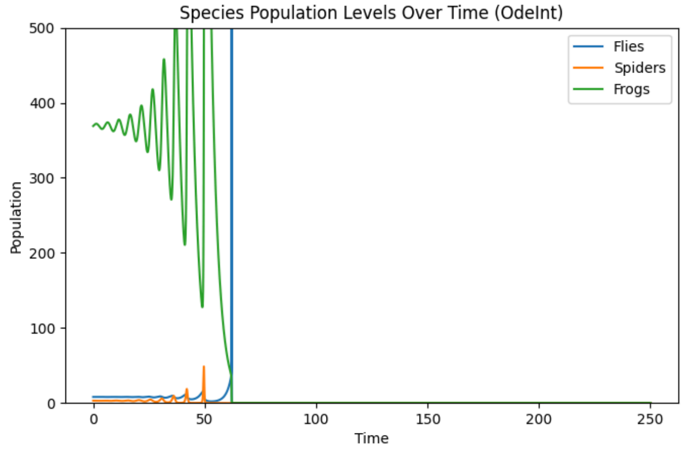


Fig 9.

With even small perturbations, we can see increasing large oscillations, to the point where even the OdeInt model fails. The oscillations are so large, and the populations are drawn so close to 0, that even with very large population numbers, eventually species extinction will result.

General Conclusions:
We can see that using either RK2 or OdeInt to solve the differential equations has made little difference, as both methods are very accurate. However, when values are too small or large, eventually the OdeInt Model fails, as seen in Fig9.

Using these methods is much better than using discrete modelling, as when using eulers method to construct difference equations, there is a small global error introduced, making our continuous methods slightly more accurate. Additionally, we see different behaviour shown in our models when using the more accurate methods for solving ODEs.

Looking at the graphs, I have made the conclusion that in my model, unless conditions and initial system parameters are exactly correct, the coexistence of all 3 species is not possible, as even slight changes in conditions results in a chain reaction that leads to the extinction of at least 1 species.

Part 4 - Steady State Analysis

A steady state occurs when the rate of change of the function is 0 for all variables, which in our model, means that at that point, there is no further change in species populations. We can find this for our continuous model by equating each variables rate of change to 0, and the values we get will also be the same for our discrete model as the populations are calculated as:

New population = old population + rate of change at old population * step size

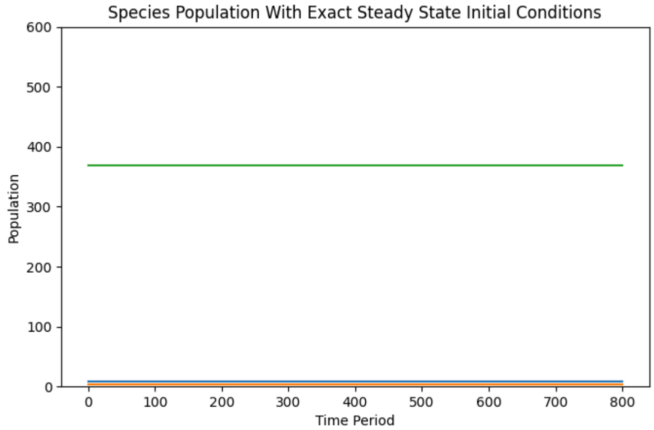
Where if the rate of change is 0, the new population will be the same as the old population, meaning that a steady state has been reached. The steady states for our model as the following:

State	X =	Y =	Z =
1	0	0	0
2	0	$-m/\lambda$	s/β
3	$-m/\rho$	0	r/a
4	$-s/\Phi$	r/θ	0
5	$(-a\lambda s + \beta\lambda r + \beta m\theta)/(a\lambda\Phi - \beta\rho\theta)$	$-(am\Phi - a\rho s + \beta r\rho)/(a\lambda\Phi - \beta\rho\theta)$	$(\lambda\Phi r + m\Phi\theta - \rho s\theta)/(a\lambda\Phi - \beta\rho\theta)$

We can see that either all species have a population of 0, or the other 2 species survive in a typical basic Lotka-Volterra fashion (as shown in Figure 7 after the extinction of spiders).

The final steady state is the one I am interested in, as it seems to suggest that perhaps all 3 species can survive in equilibrium. This was explored in Part 3 for our continuous model, and showed that this steady state was unstable, as a small change in initial population caused the population trajectories to diverge from the steady state, instead of converge (Figure 9).

What kind of system do the values create when using our discrete model?



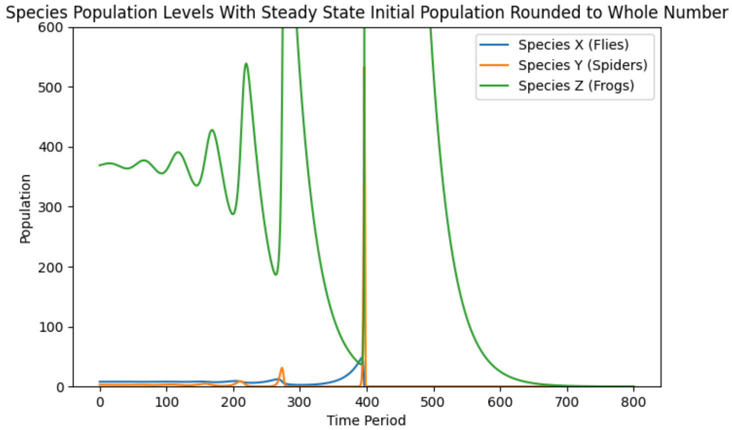


Fig 10.

We can see that there is indeed a steady state here, as with exact conditions, there is no change in population, however, as soon as we change the initial populations by rounding to the nearest whole number, we create increasing larger oscillations until all species have become extinct.

Phase Plane Analysis:
Steady state 1 is trivial, so I will not be discussing it. For steady states 2-4, the phase plane analysis are similar to the phase planes of the base lotka volterra.

Steady State 2:

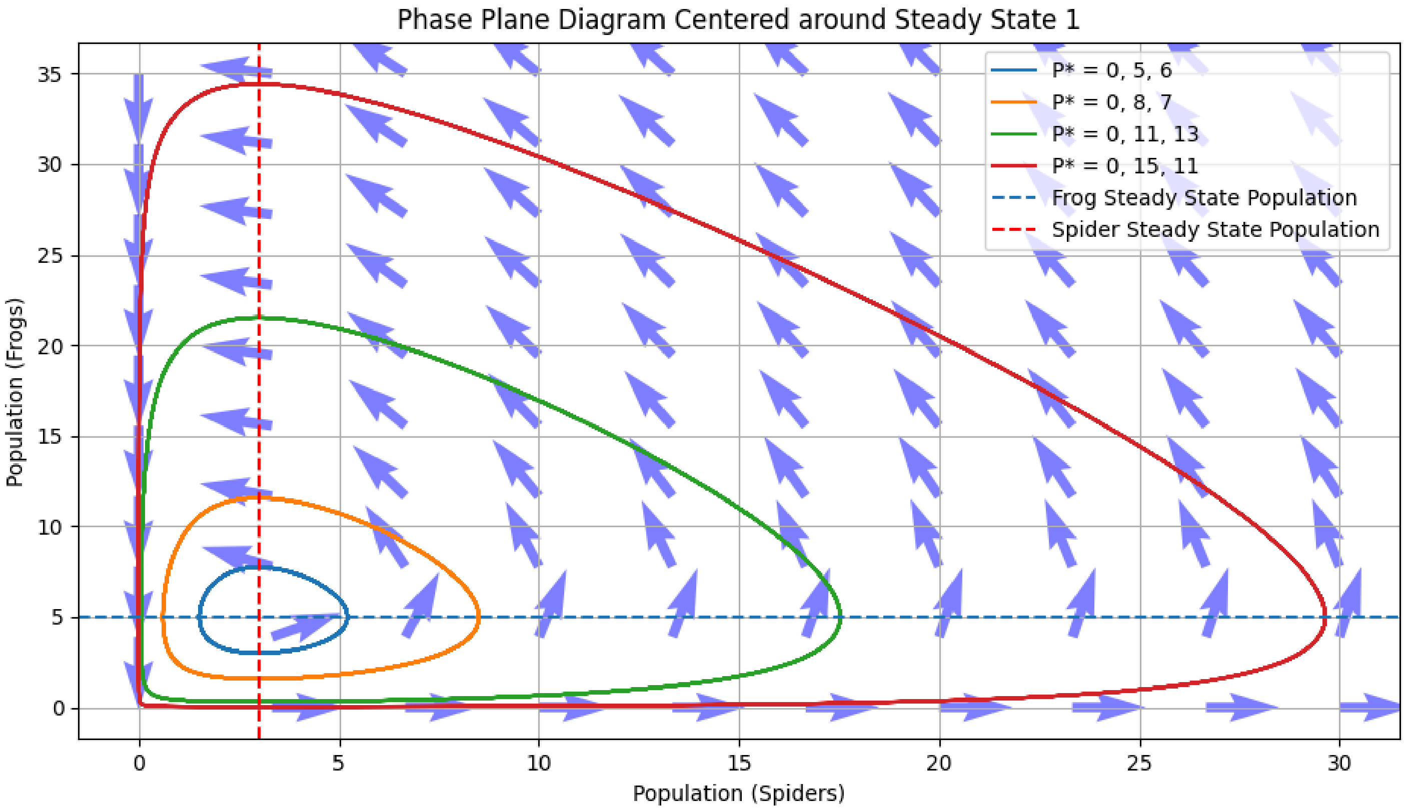


Fig11.

We can see that for steady state 2, limit cycles are shown, where the populations of spiders and frogs oscillate around the equilibrium.

Steady States 3 and 4:

Steady states 3 and 4 show similar phase plane plots to Figure 11, but for Flies and Frogs, and Flies and Spiders respectively. This relates back to our base Lotka-Volterra model, which also show the same patterns between the predator and prey.

Steady State 5:

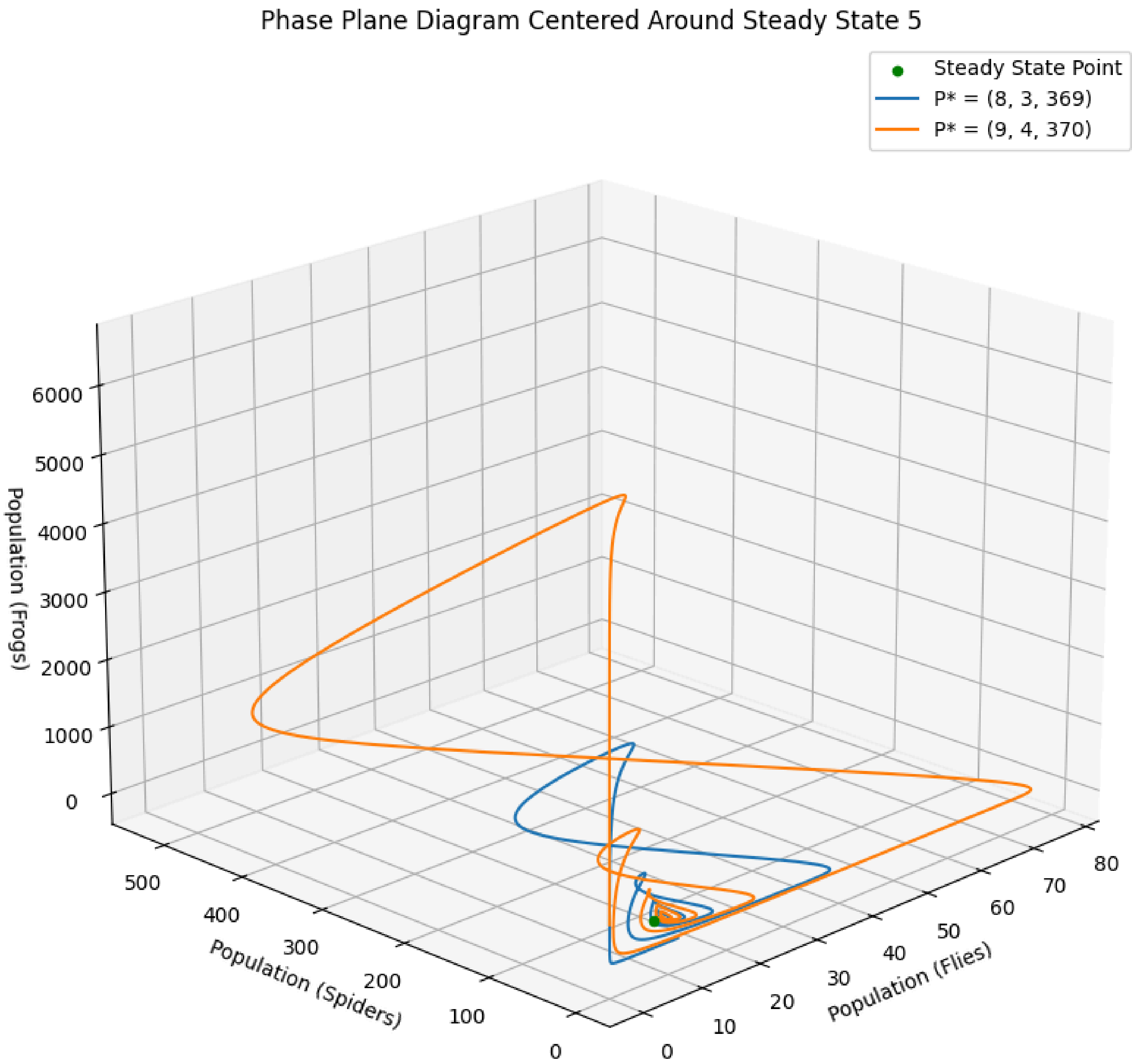


Fig 12.
#Note: When running the plotting code in my .py file, you can drag your mouse to rotate the plot around!

When constructing a 3 dimensional phase plane diagram and labelling the steady state point for steady state 5, we can see that even a small divergence from the steady state results in the line “spiralling out” from the steady state. This indicates that unless the initial starting conditions are exactly at the steady state, any small perturbation will cause the population curve to diverge away over time. This indicates that this steady state is unstable.

General Conclusions:
The question posed to be solved by the extended model concerned the introduction of a new "apex predator" species that preys on both original species in the system, and if this new system was sustainable.

From our phase plane analysis, we can see that the extinction of one species can cause the system to return to a basic Lotka-Volterra system with 2 species as predator and prey.

Additionally, it is technically possible for all 3 species to live in equilibrium if the initial conditions are perfect, however, this is very unlikely to occur, and with an unstable equilibrium, any change in initial conditions causes the population curve to spiral out from the equilibrium point.

Overall, I have made the conclusion through graphical analysis and phase space analysis that it is not possible to have a sustainable system with 3 species, all with non zero populations, unless the system parameters and initial conditions are exactly perfect.

Appendix

Figure System Parameters:

Figure 1	Flies	Spiders	Frogs

`deltaT = 1`	`xInitial = 10`
	`r = 0.01`
	`theta = 0.01`
	`a = 0.01`
		`yInitial = 10`
		`s = 0.01`
		`beta = 0.01`
		`phi = 0.01`
			`zInitial = 10`
			`m = 0.01`
			`lambdaSymbol = 0.01`
			`rho = 0.01`

Figure 2	Flies	Spiders	Frogs

`deltaT = 1`	`xInitial = 10`
	`r = 0.01`
	`theta = 0.01`
	`a = 0.01`
		`yInitial = 10`
		`s = 0.01`
		`beta = 0.01`
		`phi = 0.01`
			`zInitial = 10`
			`m = -0.2`
			`lambdaSymbol = 0.01`
			`rho = 0.01`

Figure 3	Flies	Spiders	Frogs

`deltaT = 0.1`	`xInitial = 10`
	`r = 0.5`
	`theta = 0.045`
	`a = 0.001`
		`yInitial = 3`
		`s = -0.3`
		`beta = 0.01`
		`phi = 0.03`
			`zInitial = 1`
			`m = -0.1`
			`lambdaSymbol = 0.01`
			`rho = 0.001`

Figure 4	Flies	Spiders	Frogs

`deltaT =`
`0.001`
	`xInitial = 10`
	`r = 0.5`
	`theta = 0.045`
		`yInitial = 3`
		`s = -0.3`
		`beta = 0.01`
			`zInitial = 1`
			`m = -0.1`
			`lambdaSymbol = 0.01`

	a = 0.001	phi = 0.03	rho = 0.001

Figure 5	Flies	Spiders	Frogs
RK2 and OdeInt used	xInitial = 10 r = 0.01 theta = 0.01 a = 0.01	yInitial = 10 s = 0.01 beta = 0.01 phi = 0.01	zInitial = 10 m = 0.01 lambdaSymbol = 0.01 rho = 0.01

Figure 6	Flies	Spiders	Frogs
RK2 and OdeInt used	xInitial = 10 r = 0.01 theta = 0.01 a = 0.01	yInitial = 10 s = 0.01 beta = 0.01 phi = 0.01	zInitial = 10 m = -0.2 lambdaSymbol = 0.01 rho = 0.01

Figure 7	Flies	Spiders	Frogs
RK2 and OdeInt used	xInitial = 10 r = 0.5 theta = 0.045 a = 0.001	yInitial = 3 s = -0.3 beta = 0.01 phi = 0.03	zInitial = 1 m = -0.1 lambdaSymbol = 0.01 rho = 0.001

Figure 8	Flies	Spiders	Frogs
RK2 and OdeInt used	xInitial = 7.964601769911503 r = 0.5 theta = 0.045 a = 0.001	yInitial = 2.920353982300884 s = 0.5 beta = 0.01 phi = 0.4	zInitial = 368.58407079646025 m = -0.3 lambdaSymbol = 0.1 rho = 0.001

Figure 9	Flies	Spiders	Frogs
RK2 and OdeInt used	xInitial = 8 r = 0.5 theta = 0.045 a = 0.001	yInitial = 3 s = 0.5 beta = 0.01 phi = 0.4	zInitial = 369 m = -0.3 lambdaSymbol = 0.1 rho = 0.001

Figure 10	Flies	Spiders	Frogs
deltaT = 0.1	xInitial = 8 r = 0.5 theta = 0.045 a = 0.001	yInitial = 3 s = 0.5 beta = 0.01 phi = 0.4	zInitial = 369 m = -0.3 lambdaSymbol = 0.1 rho = 0.001

Figure 11	Flies	Spiders	Frogs

`RK2 Used`	`r = 0`
	`theta = 0`
	`a = 0`
		`s = 0.5`
		`beta = 0.1`
		`phi = 0`
			`m = -0.3`
			`lambdaSymbol = 0.1`
			`rho = 0`

Figure 12	Flies	Spiders	Frogs

`RK2 Used`	`r = 0.5`
	`theta = 0.045`
	`a = 0.001`
		`s = -0.5`
		`beta = 0.01`
		`phi = 0.4`
			`m = -0.3`
			`lambdaSymbol = 0.1`
			`rho = 0.001`

Libraries Used:

Matplotlib:
    Python plotting functions
Numpy:
    Array creation and numerical methods
Sympy:
    Symbolic equation solving
Scipy.Integrate OdeInt:
    OdeInt Function

Python Code:

```
'''
FIT3139 Assignment 2

Name: Daniel Nguyen
Student Number: 32471033
Unit Code: FIT3139
Last Edit: 30/04 8:20PM
'''

import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
from scipy.integrate import odeint

#define our models
#the following functions are used to define our difference equations
def speciesXDiscrete(popX, popY, popZ, r, theta, a, deltaT):
    newPopX = popX + deltaT * (r * popX - theta * popX * popY - a * popX * popZ)
    return newPopX
```

```
def speciesYDiscrete(popX, popY, popZ, s, beta, phi, deltaT):
    newPopY = popY + deltaT * (s * popY - beta * popY * popZ + phi * popY * popX)
    return newPopY

def speciesZDiscrete(popX, popY, popZ, m, lambdaSymbol, rho, deltaT):
    newPopZ = popZ + deltaT * (m * popZ + lambdaSymbol * popZ * popY + rho * popZ *
popX)
    return newPopZ

#model our difference equations
def modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s, beta, phi,
m, lambdaSymbol, rho, deltaT):
    """_summary_
        Generate all model values for our difference equations based on initial
variables

    Args:
        timeVals = numpy array of time values for plotting
        the rest = inital values and coefficients

    Returns:
        tuple of arrays of species pop levels at each time value
    """
    xPop = [xInitial]
    yPop = [yInitial]
    zPop = [zInitial]

    for timeVal in timeVals:
        if timeVal == 0:
            pass
        else:
            newXPop = speciesXDiscrete(xPop[-1], yPop[-1], zPop[-1], r, theta, a,
deltaT)
            newYPop = speciesYDiscrete(xPop[-1], yPop[-1], zPop[-1], s, beta, phi,
deltaT)
            newZPop = speciesZDiscrete(xPop[-1], yPop[-1], zPop[-1], m, lambdaSymbol,
rho, deltaT)

            xPop.append(max(0, newXPop))
            yPop.append(max(0, newYPop))
            zPop.append(max(0, newZPop))

    return (xPop, yPop, zPop)

#define continuous model
def contModel(P, t, r, theta, a, s, beta, phi, m, lambdaSymbol, rho):
    """_summary_
```

```
        define ODEs

    Args:
        P0 = array of initial populations
        t = variable used for odeint, not used for RK2
        the rest = func coefficients

    Returns:
        numpy array of evaluated rate of change of func for each variable
    """
    dxdt = r * P[0] - theta * P[0] * P[1] - a * P[0] * P[2]
    dydt = s * P[1] - beta * P[1] * P[2] + phi * P[1] * P[0]
    dzdt = m * P[2] + lambdaSymbol * P[2] * P[1] + rho * P[2] * P[0]

    return np.array([dxdt, dydt, dzdt])

#scipy odeint for modelling
def odeintModel(xInitial, yInitial, zInitial, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, timeVals):
    """_summary_
        odeint method for solving ODEs

    Args:
        Initial pops
        func coefficients
        numpy array of time values

    Returns:
        numpy array of evaluated population levels for each variable
    """
    P0 = np.array([xInitial, yInitial, zInitial])
    P = odeint(contModel, P0, timeVals, args = (r, theta, a, s, beta, phi, m,
lambdaSymbol, rho,))
    x, y, z = P.T

    return x, y, z

#general rk2 implementation for modelling
#RK2a arg defines method used (e.g. use RK2a = 1/2 for Heuns method)
def RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m, lambdaSymbol, rho,
RK2a):
    """_summary_
        heuns method for solving odes

    Args:
        P0 = np array of initial pops
        maxTime = upper bound of time for model
        numSteps = number of points where pop is measured
```

```
        func coefficients
        RK2a = weighting of first slope

    Returns:
        numpy array of evaluated population levels for each variable
    """
    RK2b = 1 - RK2a
    RK2alpha = 1 / (2 * RK2b)
    RK2beta = 1 / (2 * RK2b)
    h = maxTime / numSteps

    timeVals = np.linspace(0, maxTime, num = numSteps + 1)

    P = np.zeros((numSteps + 1, 3))
    P[0] = P0

    for i in range(numSteps):
        k1 = contModel(P[i], None, r, theta, a, s, beta, phi, m, lambdaSymbol, rho)
        k2 = contModel(P[i] + RK2beta * h * k1, None, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho)
        newP = P[i] + h * (RK2a * k1 + RK2b * k2)
        P[i + 1] = newP

    return timeVals, P

#figure 1
def plotFig1():
    timeVals = np.linspace(0, 25, num = 25)

    xInitial = 10
    r = 0.01
    theta = 0.01
    a = 0.01

    yInitial = 10
    s = 0.01
    beta = 0.01
    phi = 0.01

    zInitial = 10
    m = 0.01
    lambdaSymbol = 0.01
    rho = 0.01

    deltaT = 1

    popVals = modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s,
beta, phi, m, lambdaSymbol, rho, deltaT)
```

```
    xPop = np.array(popVals[0])
    yPop = np.array(popVals[1])
    zPop = np.array(popVals[2])

    plt.plot(timeVals, xPop, label = "Species X (Flies)")
    plt.plot(timeVals, yPop, label = "Species Y (Spiders)")
    plt.plot(timeVals, zPop, label = "Species Z (Frogs)")
    plt.legend()
    plt.title("Species Population Levels Over 25 Time Periods")
    plt.ylabel("Population")
    plt.xlabel("Time Period")
    plt.ylim(0, 25)
    plt.show()

#figure 2
def plotFig2():
    timeVals = np.linspace(0, 2000, num = 2000)

    xInitial = 10
    r = 0.01
    theta = 0.01
    a = 0.01

    yInitial = 10
    s = 0.01
    beta = 0.01
    phi = 0.01

    zInitial = 10
    m = -0.2
    lambdaSymbol = 0.01
    rho = 0.01

    deltaT = 1

    popVals = modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s, 
beta, phi, m, lambdaSymbol, rho, deltaT)
    xPop = np.array(popVals[0])
    yPop = np.array(popVals[1])
    zPop = np.array(popVals[2])

    plt.plot(timeVals, xPop, label = "Species X (Flies)")
    plt.plot(timeVals, yPop, label = "Species Y (Spiders)")
    plt.plot(timeVals, zPop, label = "Species Z (Frogs)")
    plt.legend()
    plt.title("Species Population Levels Over 2000 Time Periods")
    plt.ylabel("Population")
    plt.xlabel("Time Period")
```

```
    plt.ylim(0, 150)
    plt.show()

#Figure 3
def plotFig3():
    timeVals = np.linspace(0, 3500, num = 3500)

    xInitial = 10
    r = 0.5
    theta = 0.045
    a = 0.001

    yInitial = 3
    s = -0.3
    beta = 0.01
    phi = 0.03

    zInitial = 1
    m = -0.1
    lambdaSymbol = 0.01
    rho = 0.001

    deltaT = 0.1

    popVals = modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s,
beta, phi, m, lambdaSymbol, rho, deltaT)
    xPop = np.array(popVals[0])
    yPop = np.array(popVals[1])
    zPop = np.array(popVals[2])

    plt.plot(timeVals, xPop, label = "Species X (Flies)")
    plt.plot(timeVals, yPop, label = "Species Y (Spiders)")
    plt.plot(timeVals, zPop, label = "Species Z (Frogs)")
    plt.legend()
    plt.title("Species Population Levels Over 3500 Time Periods")
    plt.ylabel("Population")
    plt.xlabel("Time Period")
    plt.ylim(0, 200)
    plt.show()

#Figure 4
def plotFig4():
    timeVals = np.linspace(0, 350000, num = 350000)

    xInitial = 10
    r = 0.5
    theta = 0.045
    a = 0.001
```

```
    yInitial = 3
    s = -0.3
    beta = 0.01
    phi = 0.03

    zInitial = 1
    m = -0.1
    lambdaSymbol = 0.01
    rho = 0.001

    deltaT = 0.001

    popVals = modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s, 
beta, phi, m, lambdaSymbol, rho, deltaT)
    xPop = np.array(popVals[0])
    yPop = np.array(popVals[1])
    zPop = np.array(popVals[2])

    plt.plot(timeVals, xPop, label = "Species X (Flies)")
    plt.plot(timeVals, yPop, label = "Species Y (Spiders)")
    plt.plot(timeVals, zPop, label = "Species Z (Frogs)")
    plt.legend()
    plt.title("Species Population Levels Over 350000 Time Periods")
    plt.ylabel("Population")
    plt.xlabel("Time Period")
    plt.ylim(0, 600)
    plt.show()

#Figure 5
def plotFig5():

    xInitial = 10
    r = 0.01
    theta = 0.01
    a = 0.01

    yInitial = 10
    s = 0.01
    beta = 0.01
    phi = 0.01

    zInitial = 10
    m = 0.01
    lambdaSymbol = 0.01
    rho = 0.01

    P0 = [xInitial, yInitial, zInitial]
```

```
    maxTime = 50
    numSteps = 1000
    RK2a = 1 / 2

    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)

    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.plot(timeVals, P)
    ax1.set_title("Species Population Levels Over Time (RK2)")
    ax1.set_ylabel("Population")
    ax1.set_xlabel("Time")
    ax1.set_ylim(0, 60)

    x, y, z = odeintModel(xInitial, yInitial, zInitial, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, timeVals)
    ax2.plot(timeVals, x, label = 'Flies')
    ax2.plot(timeVals, y, label = 'Spiders')
    ax2.plot(timeVals, z, label = 'Frogs')
    ax2.legend()
    ax2.set_title("Species Population Levels Over Time (OdeInt)")
    ax2.set_ylabel("Population")
    ax2.set_xlabel("Time")
    ax2.set_ylim(0, 60)

    plt.show()

#Figure 6
def plotFig6():

    xInitial = 10
    r = 0.01
    theta = 0.01
    a = 0.01

    yInitial = 10
    s = 0.01
    beta = 0.01
    phi = 0.01

    zInitial = 10
    m = -0.2
    lambdaSymbol = 0.01
    rho = 0.01

    P0 = [xInitial, yInitial, zInitial]
    maxTime = 500
    numSteps = 50000
```

```
    RK2a = 1 / 2

    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)

    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.plot(timeVals, P)
    ax1.set_title("Species Population Levels Over Time (RK2)")
    ax1.set_ylabel("Population")
    ax1.set_xlabel("Time")
    ax1.set_ylim(0, 60)

    x, y, z = odeintModel(xInitial, yInitial, zInitial, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, timeVals)
    ax2.plot(timeVals, x, label = 'Flies')
    ax2.plot(timeVals, y, label = 'Spiders')
    ax2.plot(timeVals, z, label = 'Frogs')
    ax2.legend()
    ax2.set_title("Species Population Levels Over Time (OdeInt)")
    ax2.set_ylabel("Population")
    ax2.set_xlabel("Time")
    ax2.set_ylim(0, 60)

    plt.show()

#Figure 7
def plotFig7():

    xInitial = 10
    r = 0.5
    theta = 0.045
    a = 0.001

    yInitial = 3
    s = -0.3
    beta = 0.01
    phi = 0.03

    zInitial = 1
    m = -0.1
    lambdaSymbol = 0.01
    rho = 0.001

    P0 = [xInitial, yInitial, zInitial]
    maxTime = 350
    numSteps = 350000
    RK2a = 1 / 2
```

```
    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)

    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.plot(timeVals, P)
    ax1.set_title("Species Population Levels Over Time (RK2)")
    ax1.set_ylabel("Population")
    ax1.set_xlabel("Time")
    ax1.set_ylim(0, 600)

    x, y, z = odeintModel(xInitial, yInitial, zInitial, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, timeVals)
    ax2.plot(timeVals, x, label = 'Flies')
    ax2.plot(timeVals, y, label = 'Spiders')
    ax2.plot(timeVals, z, label = 'Frogs')
    ax2.legend()
    ax2.set_title("Species Population Levels Over Time (OdeInt)")
    ax2.set_ylabel("Population")
    ax2.set_xlabel("Time")
    ax2.set_ylim(0, 600)

    plt.show()

#Figure 8
def plotFig8():

    xInitial = 7.964601769911503
    r = 0.5
    theta = 0.045
    a = 0.001

    yInitial = 2.920353982300884
    s = 0.5
    beta = 0.01
    phi = 0.4

    zInitial = 368.58407079646025
    m = -0.3
    lambdaSymbol = 0.1
    rho = 0.001

    P0 = [xInitial, yInitial, zInitial]
    maxTime = 1000
    numSteps = 100000
    RK2a = 1 / 2

    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
```

```
    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.plot(timeVals, P)
    ax1.set_title("Species Population Levels Over Time (RK2)")
    ax1.set_ylabel("Population")
    ax1.set_xlabel("Time")
    ax1.set_ylim(0, 600)

    x, y, z = odeintModel(xInitial, yInitial, zInitial, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, timeVals)
    ax2.plot(timeVals, x, label = 'Flies')
    ax2.plot(timeVals, y, label = 'Spiders')
    ax2.plot(timeVals, z, label = 'Frogs')
    ax2.legend()
    ax2.set_title("Species Population Levels Over Time (OdeInt)")
    ax2.set_ylabel("Population")
    ax2.set_xlabel("Time")
    ax2.set_ylim(0, 600)

    plt.show()

#Figure 9
def plotFig9():

    xInitial = 8
    r = 0.5
    theta = 0.045
    a = 0.001

    yInitial = 3
    s = 0.5
    beta = 0.01
    phi = 0.4

    zInitial = 369
    m = -0.3
    lambdaSymbol = 0.1
    rho = 0.001

    P0 = [xInitial, yInitial, zInitial]
    maxTime = 250
    numSteps = 250000
    RK2a = 1 / 2

    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)

    fig, (ax1, ax2) = plt.subplots(1, 2)
```

```
    ax1.plot(timeVals, P)
    ax1.set_title("Species Population Levels Over Time (RK2)")
    ax1.set_ylabel("Population")
    ax1.set_xlabel("Time")
    ax1.set_ylim(0, 500)

    x, y, z = odeintModel(xInitial, yInitial, zInitial, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, timeVals)
    ax2.plot(timeVals, x, label = 'Flies')
    ax2.plot(timeVals, y, label = 'Spiders')
    ax2.plot(timeVals, z, label = 'Frogs')
    ax2.legend()
    ax2.set_title("Species Population Levels Over Time (OdeInt)")
    ax2.set_ylabel("Population")
    ax2.set_xlabel("Time")
    ax2.set_ylim(0, 500)

    plt.show()

########################
# Uncomment to plot figs
########################

#plotFig1()
#plotFig2()
#plotFig3()
#plotFig4()
#plotFig5()
#plotFig6()
#plotFig7()
#plotFig8()
#plotFig9()

############################
# STEADY STATE ANALYSIS
############################

#getting steady states
def getSteadyStates():
    x, y, z = sp.symbols('x y z')
    r, theta, a, s, beta, phi, m, lambdaSymbol, rho = sp.symbols('r theta a s beta
phi m lambdaSymbol rho')

    dxdt = r * x - theta * x * y - a * x * z
    dydt = s * y - beta * y * z + phi * y * x
    dzdt = m * z + lambdaSymbol * z * y + rho * z * x

    steadyStates = sp.solve([dxdt, dydt, dzdt], (x, y, z))
```

```
    return steadyStates

#trial and error steady state value testing
def testSteadyStates():

    r = 0.5
    theta = 0.045
    a = 0.001

    s = 0.5
    beta = 0.01
    phi = 0.4

    m = -0.3
    lambdaSymbol = 0.1
    rho = 0.001

    x = (-a*lambdaSymbol*s + beta*lambdaSymbol*r + beta*m*theta)/(a*lambdaSymbol*phi
- beta*rho*theta)
    y = -(a*m*phi - a*rho*s + beta*r*rho)/(a*lambdaSymbol*phi - beta*rho*theta)
    z = (lambdaSymbol*phi*r + m*phi*theta - rho*s*theta)/(a*lambdaSymbol*phi -
beta*rho*theta)

    print((x, y, z))

##################
#uncomment to use
##################

#print(getSteadyStates())
#testSteadyStates()

#Figure 10 (Discrete Steady State)
def plotFig10():
    timeVals = np.linspace(0, 800, num = 800)

    xInitial = 7.964601769911503
    r = 0.5
    theta = 0.045
    a = 0.001

    yInitial = 2.920353982300884
    s = 0.5
    beta = 0.01
    phi = 0.4

    zInitial = 368.58407079646025
```

```
    m = -0.3
    lambdaSymbol = 0.1
    rho = 0.001

    deltaT = 0.1

    popVals = modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s,
beta, phi, m, lambdaSymbol, rho, deltaT)
    xPop = np.array(popVals[0])
    yPop = np.array(popVals[1])
    zPop = np.array(popVals[2])

    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.plot(timeVals, xPop, label = "Species X (Flies)")
    ax1.plot(timeVals, yPop, label = "Species Y (Spiders)")
    ax1.plot(timeVals, zPop, label = "Species Z (Frogs)")
    ax1.set_title("Species Population With Exact Steady State Initial Conditions")
    ax1.set_ylabel("Population")
    ax1.set_xlabel("Time Period")
    ax1.set_ylim(0, 600)

    xInitial = 8
    yInitial = 3
    zInitial = 369

    popVals = modelDiscrete(timeVals, xInitial, yInitial, zInitial, r, theta, a, s,
beta, phi, m, lambdaSymbol, rho, deltaT)
    xPop = np.array(popVals[0])
    yPop = np.array(popVals[1])
    zPop = np.array(popVals[2])

    ax2.plot(timeVals, xPop, label = "Species X (Flies)")
    ax2.plot(timeVals, yPop, label = "Species Y (Spiders)")
    ax2.plot(timeVals, zPop, label = "Species Z (Frogs)")
    ax2.legend()
    ax2.set_title("Species Population Levels With Steady State Initial Population
Rounded to Whole Number")
    ax2.set_ylabel("Population")
    ax2.set_xlabel("Time Period")
    ax2.set_ylim(0, 600)

    plt.show()

############################
#uncomment to plot figure 10
############################

#plotFig10()
```

```
############################
# Phase Plane Analysis
############################

#plot phase plane for Steady State 2
def plotFig11():
    r = 0
    theta = 0
    a = 0

    s = 0.5
    beta = 0.1
    phi = 0

    m = -0.3
    lambdaSymbol = 0.1
    rho = 0

    maxTime = 250
    numSteps = 250000
    RK2a = 1 / 2

    P0 = [0, 5, 6]
    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
    x = P[:, 0]
    y = P[:, 1]
    z = P[:, 2]
    plt.plot(y, z, label = "P* = 0, 5, 6")

    P0 = [0, 8, 7]
    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
    x = P[:, 0]
    y = P[:, 1]
    z = P[:, 2]
    plt.plot(y, z, label = "P* = 0, 8, 7")

    P0 = [0, 15, 11]
    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
    x = P[:, 0]
    y = P[:, 1]
    z = P[:, 2]
    plt.plot(y, z, label = "P* = 0, 11, 13")

    P0 = [0, 23, 17]
```

```
    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
    x = P[:, 0]
    y = P[:, 1]
    z = P[:, 2]
    plt.plot(y, z, label = "P* = 0, 15, 11")

    #create arrow grid
    xAxis = np.linspace(0, 30, 10)
    yAxis = np.linspace(0, 35, 10)

    for yVal in xAxis:
        for zVal in yAxis:
            P = [0, yVal, zVal]
            dxdt, dydt, dzdt = contModel(P, None, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho)
            plt.quiver(yVal, zVal, dydt, dzdt, color = 'blue', alpha = 0.5)

    #plot steady state lines
    ySteadyState = -m / lambdaSymbol
    zSteadyState = s / beta

    plt.axhline(y = zSteadyState, linestyle = '--', label = 'Frog Steady State
Population')
    plt.axvline(x = ySteadyState, color = 'red', linestyle = '--', label = 'Spider
Steady State Population')

    plt.title("Phase Plane Diagram Centered around Steady State 1")
    plt.ylabel("Population (Frogs)")
    plt.xlabel("Population (Spiders)")
    plt.legend()
    plt.grid(True)
    plt.show()

#plot phase plane for steady state 5
def plotFig12():

    xInitial = 8
    r = 0.5
    theta = 0.045
    a = 0.001

    yInitial = 3
    s = -0.5
    beta = 0.01
    phi = 0.4

    zInitial = 369
```

```
    m = -0.3
    lambdaSymbol = 0.1
    rho = 0.001

    P0 = [xInitial, yInitial, zInitial]
    maxTime = 50
    numSteps = 200000
    RK2a = 1 / 2

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(7.964601769911503, 2.920353982300884, 368.58407079646025, label =
"Steady State Point", color = 'green')

    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
    x = P[:, 0]
    y = P[:, 1]
    z = P[:, 2]
    ax.plot(x, y, z, label = 'P* = (8, 3, 369)')


    P0 = [9, 4, 370]
    timeVals, P = RK2(P0, maxTime, numSteps, r, theta, a, s, beta, phi, m,
lambdaSymbol, rho, RK2a)
    x = P[:, 0]
    y = P[:, 1]
    z = P[:, 2]
    ax.plot(x, y, z, label = 'P* = (9, 4, 370)')

    ax.set_title("Phase Plane Diagram Centered Around Steady State 5")
    ax.set_xlabel('Population (Flies)')
    ax.set_ylabel('Population (Spiders)')
    ax.set_zlabel('Population (Frogs)')
    ax.legend()
    plt.show()

#########################
# UNCOMMENT TO SHOW FIGS
#########################

#plotFig11()
#plotFig12()
```