Lotka-Volterra Model Investigation

Presented to: Computational Physics (PHY 365) Fall 2022

Jacob F. Hansman, Nick Argentieri

December 11, 2022

Contents

Abstract

Lotka- Volterra equations are a pair of first-order nonlinear differential equations that can be used to describe and model the population of predator and prey species as they interact with one another. Using Python to model solutions to the Lotka-Volterra equations, this report examines the results of the solutions for a number of different initial conditions and parameters. As well, it examines how the solutions evolve when the parameters of the equations are reduced and a third species is added to the equations.

Contributions

Efforts within this investigation were shared equally amongst the contributors. Nick Argentieri was responsible for the original RK4 code implementation of both the classic set of Lotka-Volterra equations and the two species reduced parameter model. This original implementation allowed us to view expected results for the simulations. Nick's contributions also included the creation of the equation set for the three species Lotka-Volterra model based off of the original set of equations for a two species model, the input of all the data for the beamer presentation, the writing of the abstract for report, and the creation of the bibliography. Jacob Hansman's contributions included a code implementation of ODEint and sliders for all of the population models investigated, the creation of the equation set for the three species Lotka-Volterra model based off of the reduced parameter model, the building of the framework of the presentation, and the writing of the rest of the report.

Mathematical Description

Classic Lotka-Volterra Model

The original, widely known Lotka-Volterra equations were created through the collaborative work by Vito Volterra and Alfred James Lotka and became well known in the late 1920s when Vito Volterra took Alfred Lotka's revolutionary ideas in theoretical population ecology and elaborated on them in a short published discussion. These equations were revolutionary in their approach to population ecology in that most ecologists at the time thought about population change in terms of entire food chains. Lotka and Volterra's approach skips the complications of food chain interactions and shows stability through a simpler model with just two species [5]. This model is constructed as the following equations:

Classic Lotka-Volterra Model Equations:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax - Bxy$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = Cxy - Dy$$

Classic Lotka-Volterra Model Parameter Descriptions:

A : Birth rate of rabbits
B : Death rate of rabbits due to consumption by foxes
C : Natural death rate of foxes
D : Parameter describing how many eaten rabbits produces a new fox

This model accounts for any basic 2 organisms' predator and prey relationship. The x terms account for the population change of the prey organism and the y terms account for the population change of the predator organism. In the population change equation for the prey organism, $\frac{\mathrm{d}x}{\mathrm{d}t}$; the Ax term represents the growth rate of the prey population, and the $-Bxy$ term represents the population decline rate due to consumption by the predator. In the population change equation for the predator organism, $\frac{\mathrm{d}y}{\mathrm{d}t}$; the Cxy term represents the population growth rate for the predator organism by consumption of prey, and the $-Dy$ term represents the population decline rate of the predator organism by natural death.

Reducing Parameters

What also makes Alfred Lotka and Vito Volterra's model of population change so revolutionary and fundamentally different from previous approaches is its level of utility and adaptability. The classic Lokta-Volterra Model can be adapted to fit many different types of situations, not limited to population change, through a reevaluation of the parameters needed to be used and the build of the equations. In this investigation the parameters were reduced from the classical model and the structure of the equations were slightly modified to emulate a food pressure placed on the prey organism in the model. The model for this is as follows:

Original Lotka-Volterra Model Equations with Reduced Parameters:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x - xy - \alpha x^2$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = xy - \beta y$$

Original Lotka-Volterra Model with Reduced Parameters Parameter Descriptions:

α : Parameter describing food pressure on the rabbits (availability of grass, etc.)
β : Natural death rate of foxes

This model again accounts for any basic 2 organism, predator and prey relationship, but is modified so that the growth rate of predator and prey populations are constant but food pressure for the prey organism and natural death rate for the predator are allowed for variable changes. In the population change equation for the prey organism, $\frac{\mathrm{d}x}{\mathrm{d}t}$; the x term represents the natural birth rate of the prey population, the $-xy$ term represents the decline in population due to consumption by predators, and the $-\alpha x^2$ term represents the food pressures placed on the prey population. In the population change equation for the predator organism, $\frac{\mathrm{d}y}{\mathrm{d}t}$; the xy term represents the population growth of the predators due to consumption of the prey, and the $-\beta y$ term represents decline in population due to the natural death rate of the predators.

Adding a Species

Once again, to apply and demonstrate the adaptability of the Lotka-Volterra Model, using the same framework as the classic model a new species can be added that is a predator above both organisms previously utilized, and the interactions between the species can be studied. In this situation, the prey is eaten by both predators, the first predator is eaten by the second, greater predator, and the great predator is eaten by nothing but consumes both the prey and the other predator. This model was built as follows:

Lotka-Volterra Model with added species (Dragon, Original model-based):

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax - Bxy - Exz$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = Cxy - Dy - Fyz$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} = Gxz + Hyz - Iz$$

Lotka-Volterra Model with added species (Dragon, Original model-based) Parameter Description:

A : Birth rate of rabbits
B : Death rate of rabbits due to foxes
C : Parameter describing how many eaten rabbits produces a new fox
D : Natural death rate of foxes
E : Death rate of rabbits due to dragons
F : Death rate of foxes due to dragons
G : Growth rate of the dragon population due to rabbit consumption
H : Growth rate of the dragon population due to fox consumption
I : Natural death rate of dragons

This set of 3 equations is very similar to the original Lotka-Volterra model both in the way that the equations are structured and in the way that the parameters are defined. But, this 3-species model differs greatly from the classic model in that it is no longer a 4-dimensional system, but rather a 9-dimensional system with the necessary added parameters. This added dimensionality creates for a much greater abundance of chaos presented in the species interactions as well as a noticeably diminished amount of realistic solutions. For the population change equation for the prey organism, $\frac{dx}{dt}$; the Ax term represents the population growth by the birth rate of the prey organisms, the $-Bxy$ term represents the population decline by consumption by the first predator, the $-Exz$ term represents decline in the population by consumption by the greater predator. In the population change equation for the first predator, $\frac{dy}{dt}$; the Cxy term represents the population growth for the first predator through consumption of the prey, the $-Dy$ term represents the natural death rate of the first predator, and the $-Fyz$ term represents the population decline rate of the first predator through consumption by the greater predator. In the population change equation for the great predator, $\frac{dz}{dt}$; the Gxz term represents the population growth of the great predator through consumption of the prey, the Hyz term represents the population growth of the great predator through consumption of the first predator, and the $-Iz$ term represents the natural death rate of the great predator.

Now, while it is interesting to view this 3-species model that is akin to the classic Lotka-Volterra model with 9 parameters to see what kind of results are yielded, it is more beneficial to take a look at a 3-species model that is based off the previously discussed reduced parameter model of the L-V equations. This is due to the lesser abundance of abrupt chaos and greater abundance of realistic solutions. This model's build is as follows:

Lotka-Volterra Model with added species (Dragon, reduced parameter model-based):

$$\frac{dx}{dt} = x - xy - \alpha x^2 - xz$$
$$\frac{dy}{dt} = xy - \beta y - yz$$
$$\frac{dz}{dt} = xz + xy - \gamma z$$

Lotka-Volterra Model with added species (Dragon, reduced parameter model-based) Parameter Description:

α : Parameter describing food pressure on the rabbits (availability of grass, etc.)
β : Natural death rate of foxes
γ : Natural death rate of dragons

Again, this system of 3 equations is similar structurally and parameter-wise to the previously discussed, reduced parameter L-V model constructed. However, the reduction of parameters allows for clearer analysis and thorough understanding of the equations themselves and their interactions with each other. In the population change equation for the prey organism, $\frac{dx}{dt}$; the x term represents the growth rate of the prey population due to natural birth, the $-xy$ term represents the decline of the population due to consumption by the first predator, the $-\alpha x^2$ term represents the population decline due to food pressures placed on the prey population, and the $-xz$ term represents the population decline due to consumption by the great predator. In the population change equation for the first predator, $\frac{dy}{dt}$; the xy term represents the population growth rate of the first predator population due to consumption of prey, the $-\beta y$ term represents the natural death rate of the first predator population, and the $-yz$ term represents the decline in population due to consumption by the great predator. In the population change equation for the great predator, $\frac{dz}{dt}$; the xz term represents the population growth by consumption of the prey, the $+xy$ term represents the population growth by consumption of the first predator, and the $-\gamma z$ term represents the population decline due to natural death of the great predator.

Numerical Methods Utilized

There were two major numerical methods utilized in the investigation of the Lotka-Volterra Equations, these being the 4th Order Runge-Kutta method for integration (RK4) and numpy's built in ODEint differential equation solver.

4th order Runge-Kutta Method for Integration

The 4th order Runge-Kutta Method is a method for integration where differential equations are solved by applying the slope of the midpoint to $x(t)$ to find $x(t + \Delta t)$. In programming terms, this is denoted as $x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t$. Where k_1 is the slope at the beginning of the interval being calculated, using x. $k2$ is the slope at the midpoint of the interval, using x and $k1$. $k3$ is again the calculated slope at the midpoint, this time using x and $k2$. $k4$ is the slope at the end of the interval, measured using x and $k3$. This method was useful in finding a static plot that gave an idea of a reasonable, realistic solution with the given initial conditions. However, when taking the investigation further to find out what differing the parameters does to the solutions, a more dynamic differential equation solver was needed.

ODEint

ODEint is a built in integrator within numpy that utilizes the complex LSODA algorithm for solving differential equations. This algorithm did not have to be explicitly defined within any of the programs due to it being included in the numpy package. All that was needed was a definition of the Lotka-Volterra equations utilized and python completed the rest of the computations automatically. The benefit in using ODEint over the 4th Order Runge-Kutta Method for integration was that then a dynamic, interactive program could be made with included sliders that can vary the parameters to the users content and simultaneously update the graph so the user can view the changes to the solutions as the parameters were changed.

Results

Classic Lotka Volterra Model Results

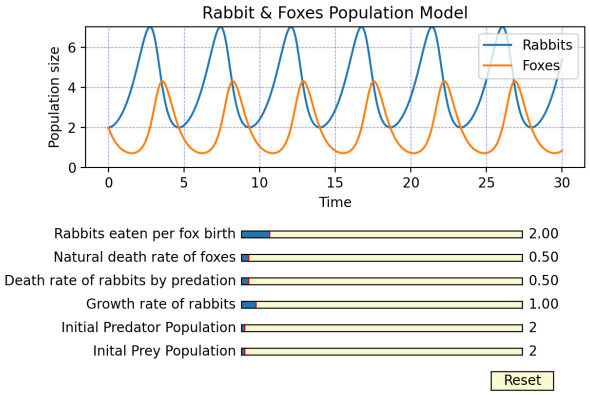

Figure 1: Typical Results (Classic L-V Model)

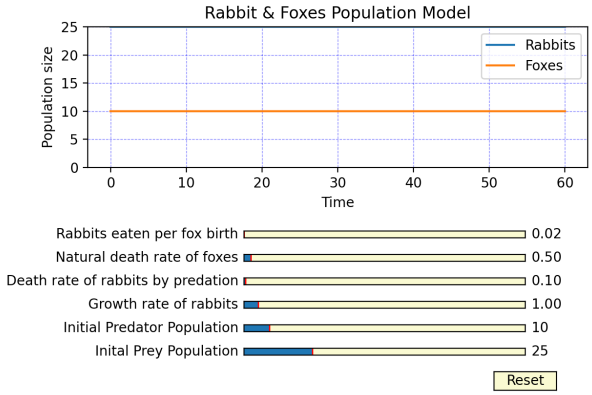

Figure 2: Steady States (Classic L-V Model)

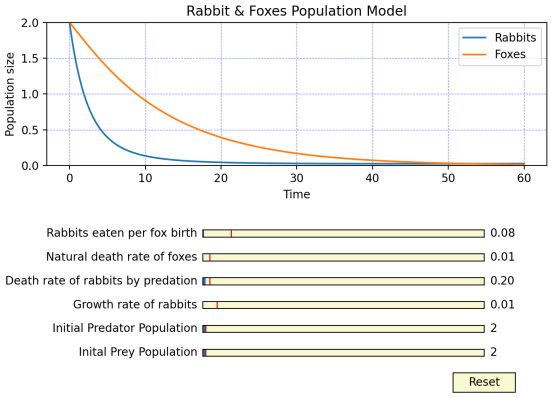

Figure 3: Extinction State (Classic L-V Model)

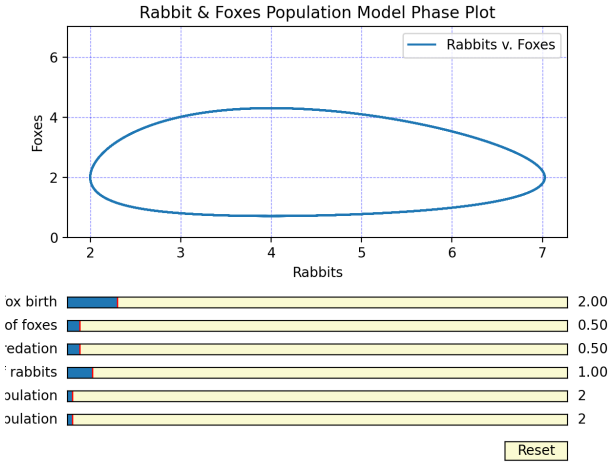

Figure 4: Phase Plot (Classic L-V Model)

Lotka Volterra Model with Reduced Parameters Results

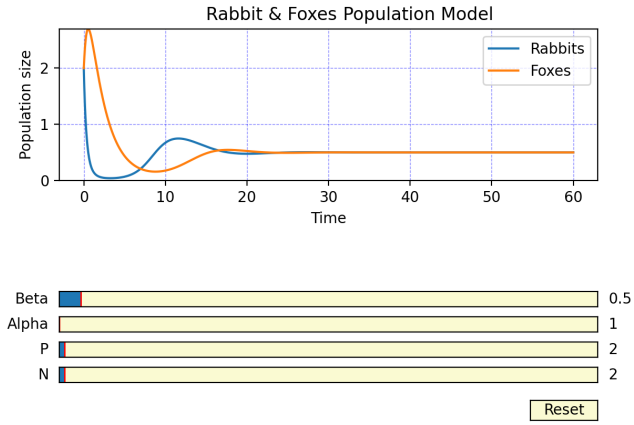

Figure 5: Typical Results (Reduced Parameter L-V Model)

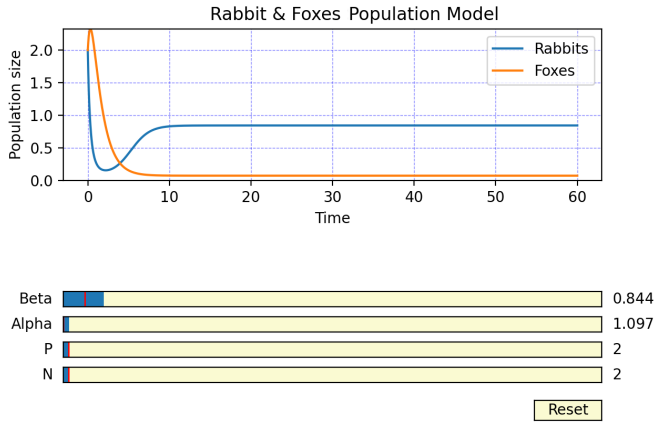

Figure 6: Steady States (Reduced Parameter L-V Model)

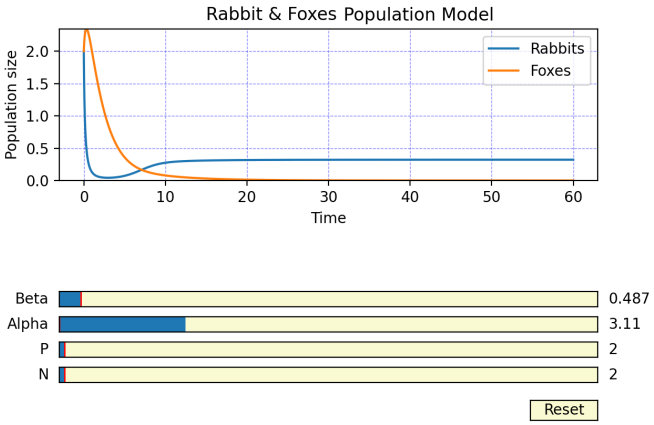

Figure 7: Extinction States (Reduced Parameter L-V Model)

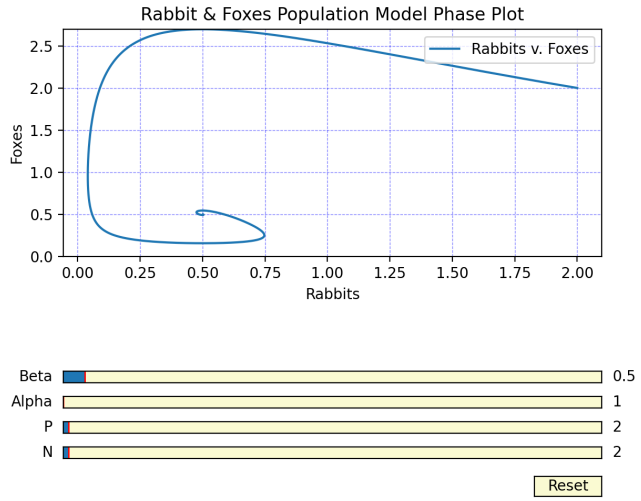

Figure 8: Phase Plot (Reduced Parameter L-V Model)

Classic Lotka Volterra Model with Added Species

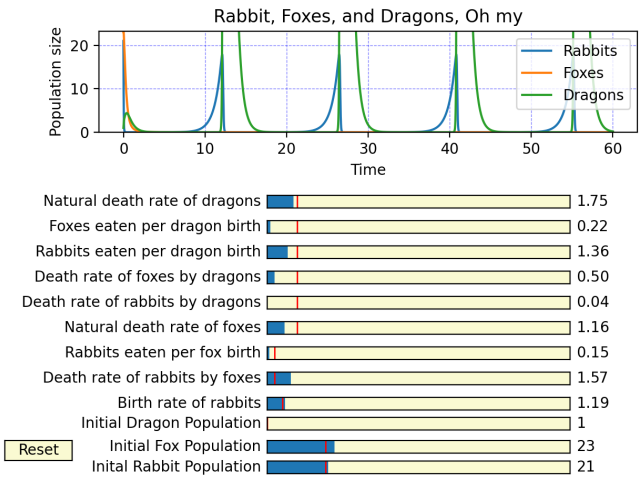

Figure 9: Typical Results (Classic L-V Model with Added Species)

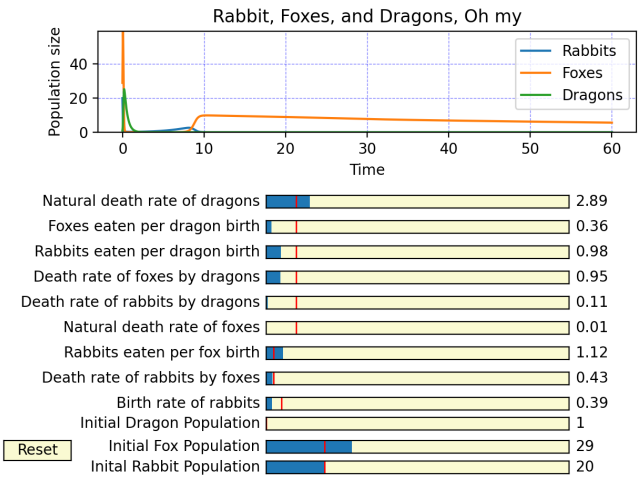

Figure 10: Steady States (Classic L-V Model with Added Species)

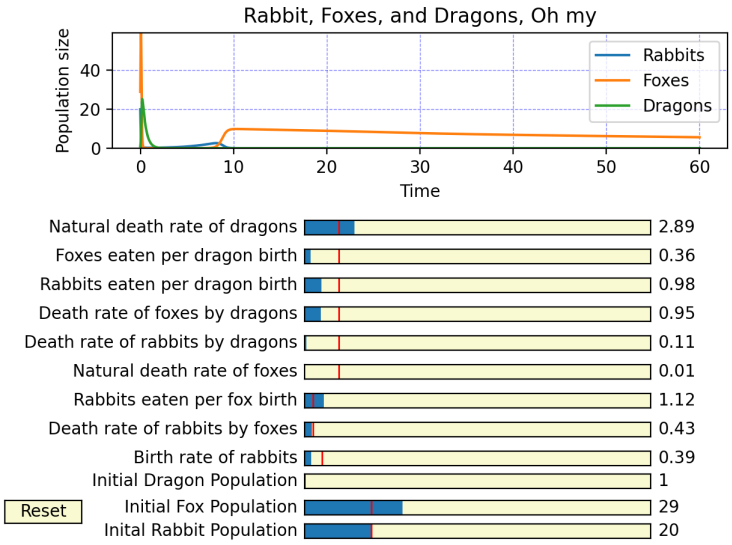

Figure 11: Extinction State Approach (Classic L-V Model with Added Species)

Lotka Volterra Model with Reduced Parameters and with Added Species Results

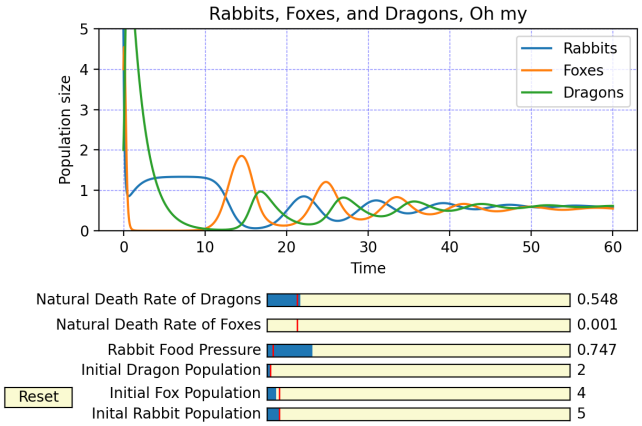

Figure 12: Typical Results (Reduced Parameter L-V Model with Added Species)

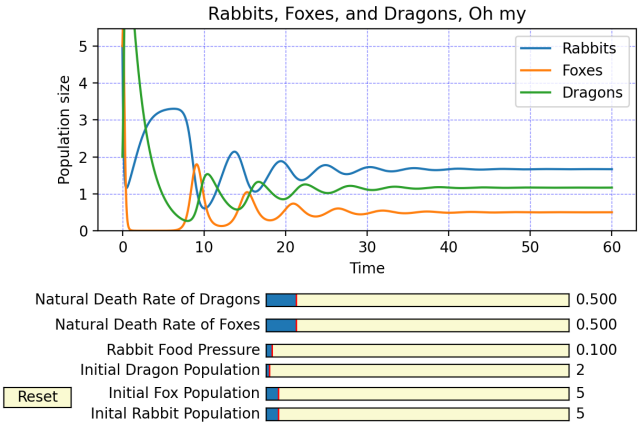

Figure 13: Steady States (Reduced Parameter L-V Model with Added Species)

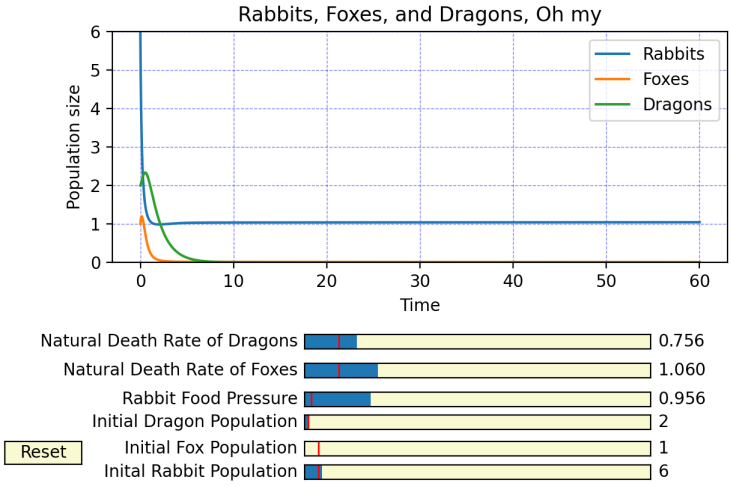

Figure 14: Extinction State (Reduced Parameter L-V Model with Added Species)

Other Interesting Results

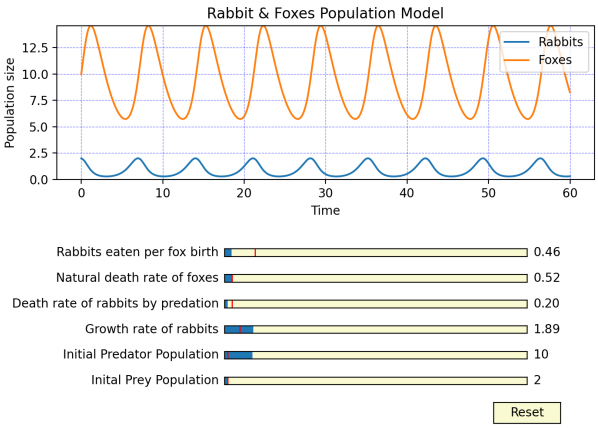

Figure 15: Interesting Result (Classic L-V Model)

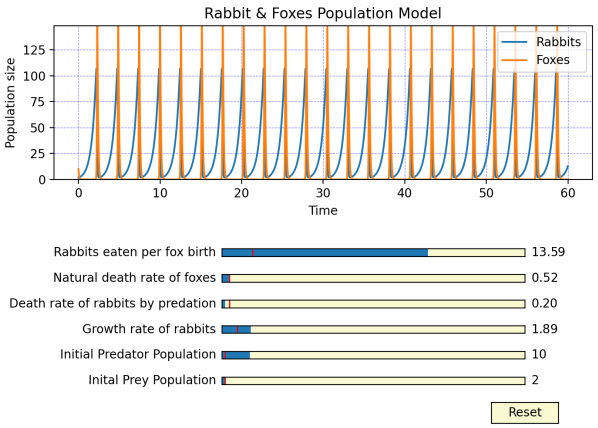

Figure 16: Another Interesting Result (Classic L-V Model)

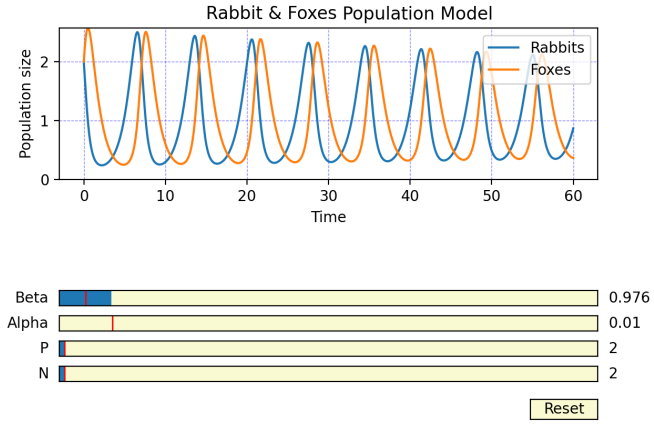

Figure 17: Interesting Result (Reduced Parameters L-V Model)

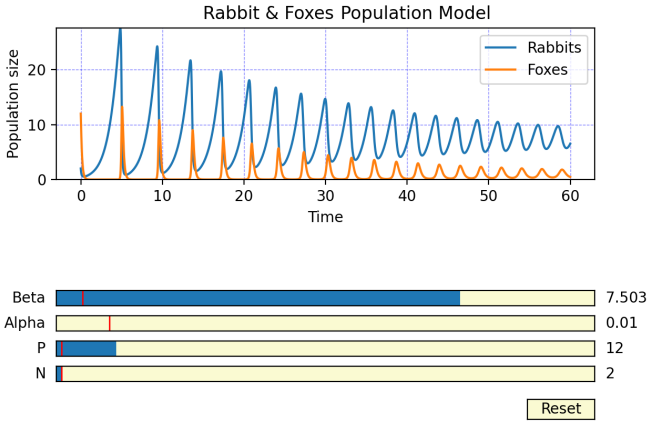

Figure 18: Another Interesting Result (Reduced Parameter L-V Model)

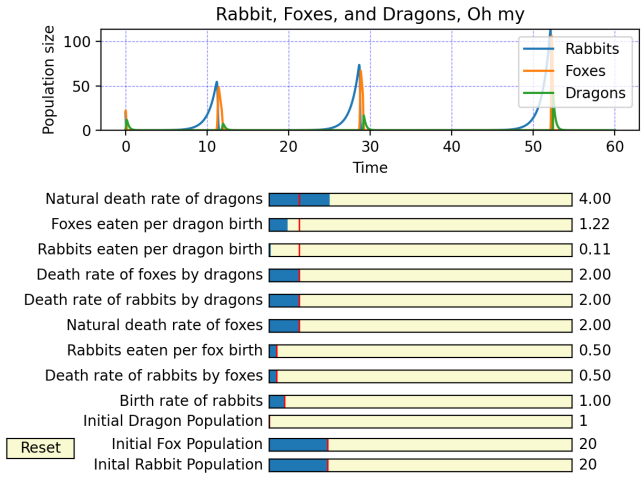

Figure 19: Interesting Result (Classic L-V Model with Added Species)

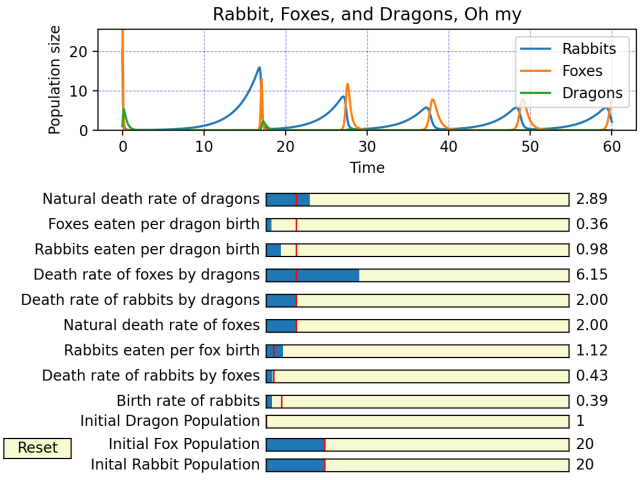

Figure 20: Another Interesting Result (Classic L-V Model with Added Species)

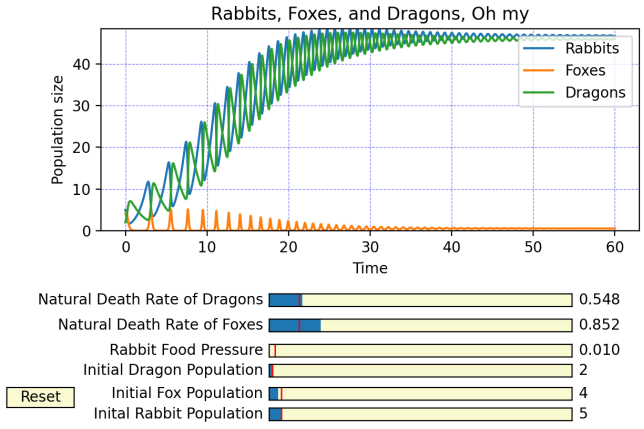

Figure 21: Interesting Result (Reduced Parameter L-V Model with Added Species)

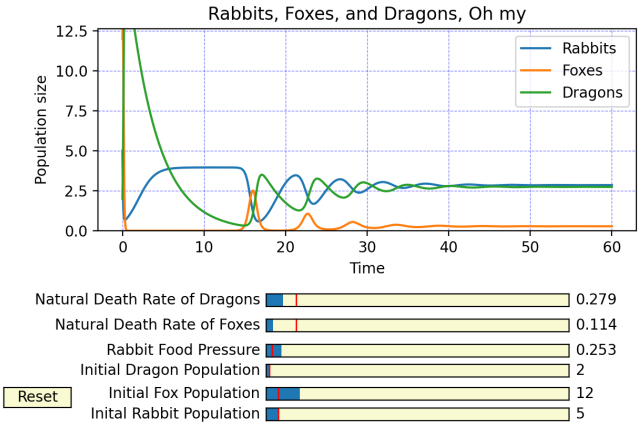

Figure 22: Another Interesting Result (Reduced Parameter L-V Model with Added Species)

Result Analysis

Classic Lotka Volterra Model Results

In general, the way that the code was written to investigation the Lotka-Volterra equations in full was extremely beneficial in analyzing a multitude of different situations other than just the classic L-V system. The sliders that were implemented along with ODEint to adjust the conditions of the parameters along with the initial populations allowed for an interactive search for different kinds of states of different kinds of systems. Rather than explicitly defining different kinds of situations within the written code functions with explicit parameter values, the program could be ran and then the parameters could be changed while simultaneously updating the graph. This allowed for a more consistent and coherent understanding of what changing different parameters resulted in. In general, the results produced were deemed successful, consistent with expectations put forth by investigations by other researchers.

As seen in the first section of the results given, what was resultant from the investigation of the classic set of Lotka-Volterra equations was akin to the results that are widely known and have been fully investigated previously by Vito Volterra and James Lotka along with countless other scientific population ecologists. In Figure 1, what is seen are the typical classical results from the Lotka-Volterra equations. Given initial conditions for the parameters of $A = 1$, $B = .5$, $C = .5$, and $D = 2$ (See Mathematical Description for parameter definitions), what is resultant is a stable, sinusoidal, cyclic interchange between the populations of prey and predators. If time were to have been extended, the pattern would continue forever stably. In Figure 2, the steady states of the two populations are shown. It is known that for a population to have a steady state, it must have a population change of 0, or $\frac{\mathrm{d}x}{\mathrm{d}t} = 0 = \frac{\mathrm{d}y}{\mathrm{d}t}$. This can be clearly seen in Figure 2, with completely linear results with a slope (change) of 0 at a population of 10 for the predators in the system and a population of 25 for the preys in the system. In Figure 3, the extinction state of the predator-prey system is shown. This was found, again, by adjustment of the parameters until an understanding was reached that to find a situation where one of or both of the species reached extinction, then the parameters that described the growth rates of the species had to be very low, while the parameters that described the death rates of the species had to be much higher than the parameters describing the growth, or birth rate. This is shown in Figure 3 with very low values for the parameters A and C, and with higher values for parameters B and D and the result is an exponential decline in both populations as they approach extinction. In Figure 4, the phase plot of the results of Figure 1 is shown using the initial conditions previously described. The resultant phase plot is shown as one solid, circular-esque shape due to the pattern of the typical results already described. The typical results with the given initial conditions for the parameters produced a sinusoidal, cyclic pattern between the oscillating populations with the waves representing the populations being consistently in phase with each other. So, it is expected that the phase plot would have the endpoints attached to each other and would show that this cycle repeats indefinitely.

Lotka Volterra Model with Reduced Parameters Results

The same slider and ODEint code implementation could be utilized to analyze a Lotka-Volterra population model with reduced parameters the same way that it analyzed the classic system. To accomplish this, the only thing that had to be changed in the code were the structure of the L-V equations in the defined function and the amount of slider implementations had to be reduced due to the reduction of the total amount of parameters in general. The results displayed were deemed successful and consistent with expectations put forth by previous investigations of similar models by other researchers.

As seen in the second section of the results provided, the implemented Lotka-Volterra model with reduced parameters presented solutions that were consistent with expectations. In Figure 5, the typical results for a system such as this one with reduced parameters is shown. With the given intial conditions for $\alpha = 1$ and

$\beta = .5$, the results shown show a quick oscillation between the two populations before the both settle down to a steady, equivalent population. This is due to the reduction of the complex parameters and instead an introduction of a food pressure parameter on the prey species. In Figure 6, a steady state solution of the system analyzed was found where the parameters describing the food pressure on the prey species and the natural death rate of the predator species were almost equivalent. This result shows again a small oscillation between the populations before a differing steady state between the two populations is reached, with the predator population remaining stable but near extinction and the prey population remaining stable at a greater population. In Figure 7, an extinction state of the predator species was shown. It is similar to the steady state solution shown in Figure 6, but the β parameter in Figure 7 was reduced so that the prey population could take over entirely, resulting in the extinction of the predator population. In Figure 8, the phase plot of the solution for Figure 5 is shown. This phase plot is consistent with what was expected due to the results shown in Figure 5. This phase plot was expected because in Figure 5, what is shown is one oscillation between the populations and then the populations settle in on one value. In the phase plot, this is exactly what is seen. There is one oscillation viewed from the initial start point that curls in and then settles in, or stops, on one value of .5.

Classic Lotka-Volterra Model with Added Species

Again, using the same slider and ODEint code implementation, the classic L-V model could be altered to include another species. Using the same parameter and equation build as the classic L-V model, when another species is added, it is necessary to add 5 more parameters to describe the system. Having 9 parameters, A, B, C, D, E, F, G, H , and I, introduces a lot more dimensionality into the system and thus a lot more odd solutions. With a 9-dimensional system compared to a 4-dimensional system, a lot more chaotic solutions are resultant as the 3 differential equations built for this system interact with each other in a way more complex manner than the classic L-V system. Despite this chaos, some expected results along with many interesting results were yielded.

Within this system, it was initially unknown the type of results that would be found with an added third species. This third species was built so that it would be a predator for both the initial prey species and for the initial predator species. With this model, the typical results found, as shown in Figure 9, with similar initial parameter values as the classic L-V model showed that the intermediary predator species died off quickly in most situations. Then, the prey species and the dominant species oscillated periodically constantly over time. Steady state solutions for this system were difficult to find due to just slight variations of parameters resulting in the system blowing up. So, a steady state approach was eventually found in Figure 10, with the intermediary predator approaching a constant population. This was found by making the parameter responsible for the natural death rate of the primary predator very high and by making the growth rate of the prey population very low. What resulted was a sort of psuedo-equilibrium where the intermediary predator population remained close to constant. In Figure 11, an extinction state for the prey and the primary predator populations was found. Similar to Figure 10, the death rate parameter for the primary predator was significantly higher than the other parameters and the birth rate parameter for the prey population was significantly lower than the other parameters. The intermediary predator population is again fairly constant but has a slight decline suggesting that over a longer time span it would die off as well.

Lotka-Volterra Model with Reduced Parameters and with Added Species Results

Yet again, the same slider and ODEint implementation could be applied to yet another type of population system. This time, the added species previously mentioned in the last system was kept in the model, but the amount of parameters was reduced to match that of the type of parameters utilized in the first reduced-parameter L-V model discussed in the second section of results. With the reduced amount of parameters for this 3 species L-V model, the solutions presented as the parameters were slightly changed were much more comprehensible and blew up into chaos far less often as the three differential equations utilized were not as complexly intertwined. Due to this simplification of the system, the results were much more consistent with expectations.

In Figure 12, the typical results for this 3 species, reduced parameter L-V model are shown. The results are similar to that of the original 2 species reduced parameter L-V model, in that all the species' populations oscillate for a while until they all stabilize in on one population value. However, these typical results were only able to be yielded if the parameter for the death rate of the intermediary predator was set very low. In Figure 13, the steady state solution of each of the populations is shown. This was found unexpectedly, as the initial conditions for the previous reduced parameter L-V model resulted in this differing steady state rather than all the populations centering in on one value. In Figure 14, the extinction state for the two predator populations is shown. For this to be achieved, an understanding was came to that the parameter for prey's food pressure must be increased so that the populations for the predators struggle. When the sliders were moved to match this parameter description, the extinction state presented itself. The prey population stabilizes as it is not being hunted anymore and the predator populations die out.

Other Interesting Results

Many other interesting results that did not fit into a specific solution category were also found through our ability to play around with the parameters in real-time. Some of these interesting results showed very strange behavior, while others showed expected behavior but yet still in odd forms.

In Figures 15 and 16, interesting results yielded from the classic Lotka-Volterra equation model are shown. In Figure 15, an oscillating semi-steady state solution is shown. We have concluded that this is a sort of steady state solution because the oscillations of the populations are constant and unchanging, yet at the same time the populations do not interact with each other. This is differing from the regular steady-state solution in that the population change, $\frac{dx}{dt}$ and $\frac{dy}{dt}$, is not zero, but at the same time the populations do not appear to ever change their oscillation nor do they ever approach extinction. In Figure 16, another interesting result is shown that is more akin to the expected behavior of the system. However, the peaks of the oscillations of the two populations are completely in phase with each other. This implies that both populations reach a maximum at the same time, and then both die off close to extinction over the same time frame. In Figures 17 and 18, interesting results for the reduced parameter L-V model are shown. In Figure 17, by setting the parameter for α at almost zero, an oscillatory motion of the two populations can be seen that is similar to that of the typical results of the classic L-V equation model. However, this solution has the two populations almost in phase and has the peaks of the population declining over time, suggesting a motion towards extinction over a long time frame. In Figure 18, another interesting result of the reduced parameter L-V model is shown that is similar to the results of Figure 16 in that the peaks of the population are in phase for a while, but then over time then distance themselves from each other and settle into a kind of oscillating semi-steady state like what is seen in Figure 15. Again, this was found by setting the value for α to be very low, but the parameter for β was also set very high as well. In Figures 19 and 20, interesting

results for the 3 species L-V model based off the classic L-V model are shown. In Figure 19, an interesting result is found where the peaks of the populations are again in phase with eachother, but this time around the peaks reach periods of extinction where then the populations all bounce back at the same time with growing max population peaks. In Figure 20, something slightly similar to Figure 19 is occurring, with the population peaks for the 3 species beginning in phase with one another. However, over time these waves distance themselves from each other and the populations begin to die off with each other, suggesting over a long time frame this would become an extinction state. In Figures 21 and 22, interesting results for the 3 species L-V equation model based off the previously used reduced parameter L-V equation model are shown. In Figure 21, a wildly interesting result is found through the reduction of the food pressure on the prey population parameter close to zero and by also raising the natural death rate of the intermediary and primary predators. What is resultant is a constant, exponentially increasing oscillation between the primary predator and the prey, while the intermediary predator population oscillates toward extinction. While this result is not realistic, it is still interesting to view this visually pleasing interaction between the populations. In Figure 22, another interesting result from this system is shown. In this solution, all 3 populations interact for a short time, the prey population presents a steady state for a short amount of time, the intermediary predator population heads for extinction while the prey population and the primary predator populations oscillate towards a singular population value and steady state as is seen in the typical results for the reduced parameter L-V model with only 2 species.

Result Analysis Conclusion

In conclusion, all of the results presented from each of the 4 systems implemented through the written code were consistent with expectations. Similarly, most of the solution results to these systems yielded were realistic solutions with the exception of some outlier unrealistic solutions that came about through extensive variation of the parameters in the equation system. With the results yielded, a further understanding of the complexities of population interactions and the parameters that guide these interactions was found. Through this further understanding, altering the parameters in real-time allowed us to, again in real-time, view and simultaneously understand the changes to populations through the parameter differences. Overall, this investigation on the Lotka-Volterra equation model was successful and led to a further understanding of ecological population modeling.

Bibliography

[1] A. Hills, Github: 'Modelling Predator-Prey Systems in Python,' (2017). https://github.com/INASIC/predator-prey-systems/blob/master/Modelling$%$20Predator-Prey$%$20Systems$%$20in$%$20Python.ipynb

[2] 'berna1111', Stack Overflow: 'How to make two sliders in matplotlib,' (2017). https://stackoverflow.com/questions/43381449/how-to-make-two-sliders-in-matplotlib

[3] J. Kingston, Stack Overflow: 'Interactive matplotlib plot with two sliders,' (2020). https://stackoverflow.com/questions/6697259/interactive-matplotlib-plot-with-two-sliders

[4] S. Blank, Python Programming in OpenGL: 'A Graphical Approach to Programming,' (2009). http://new.math.uiuc.edu/public198/ipython/stanblank/PyOpenGL.pdf

[5] S. Kingsland, Proceedings of the National Academy of Sciences of the United States of America 112, (2015). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4534218/

[6] Scientific Python: 'Lotka-Volterra equations' https://scientific-python.readthedocs.io/en/latest/notebooksrst/3OrdinaryDifferentialEquations/02Examples/\\LotkaVolterramodel.html

Code Listing

Classic Lotka Volterra Model:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons
from scipy.integrate import odeint
"""
Code adapted from: @berna1111's answer to https://stackoverflow.com/questions
    /43381449/how-to-make-two-sliders-in-matplotlib

"""


def lotka(x, t, params):
    N, P = x
    a, b, c, d = params
    """
    growth rate zero for steady states

    derivs = [0,0]
    """
    derivs = [a*N - b*N*P, c*N*P - d*P]
    return derivs



# Parameters
Preymin = 1
Preymax = 100
Predmin = 1
Predmax = 100
Amin = .01
Bmin = .01
Cmin = .01
Dmin = .01
Amax = 20
Bmax = 20
Cmax = 20
Dmax = 20
Prey_0 = 2
Pred_0 = 2
a0 = 1
b0 = .5
c0 = .5
d0 = 2
"""
steady state params:

a0 = 1
b0 = .1
c0 = .5
```

```
d0 = .02
"""
"""
steady state initial populations, growth rate 0

Prey_0 = c0/d0
Pred_0 = a0/b0
"""
a = 1
b = 0.5
c = 0.5
d = 2

params = [a, b, c, d]
x0=[Prey_0,Pred_0]
end_time = 60
tstep = 0.001

# Initial function values
t = np.arange(0, end_time, tstep)
prey, predator = odeint(lotka, x0, t, args=(params,)).T



fig = plt.figure()
ax = fig.add_axes([0.15, 0.5, 0.8, 0.3])
prey_plot = ax.plot(t, prey, label="Rabbits")[0]
predator_plot = ax.plot(t, predator, label="Foxes")[0]

ax.set_xlabel("Time")
ax.set_ylabel("Population size")
ax.legend(loc="upper right")
ax.set_title('Rabbit & Foxes Population Model')
ax.grid(color="b", alpha=0.5, linestyle="dashed", linewidth=0.5)
ax.set_ylim([0, np.max([prey, predator])])


axcolor = 'lightgoldenrodyellow'
axis_Prey = fig.add_axes([0.4, 0.1, 0.45, 0.015], facecolor=axcolor)
axis_Pred = fig.add_axes([0.4, 0.15, 0.45, 0.015], facecolor=axcolor)
axis_A = fig.add_axes([0.4, 0.20, 0.45, 0.015], facecolor=axcolor)
axis_B = fig.add_axes([0.4, 0.25, 0.45, 0.015], facecolor=axcolor)
axis_C = fig.add_axes([0.4, 0.30, 0.45, 0.015], facecolor=axcolor)
axis_D = fig.add_axes([0.4, 0.35, 0.45, 0.015], facecolor=axcolor)

# size: [left, bottom, width, height]

# create each slider on its corresponding place:
slider_Prey = Slider(axis_Prey, 'Inital Prey Population', Preymin, Preymax,
    valinit=Prey_0, valstep=1)
slider_Pred = Slider(axis_Pred, 'Initial Predator Population', Predmin, Predmax,
    valinit=Pred_0, valstep=1)
slider_A = Slider(axis_A, 'Growth rate of rabbits', Amin, Amax, valinit=a0)
```

```
slider_B = Slider(axis_B, 'Death rate of rabbits by predation', Bmin, Bmax,
    valinit=b0)
slider_C = Slider(axis_C, 'Natural death rate of foxes', Cmin, Cmax, valinit=c0)
slider_D = Slider(axis_D, 'Rabbits eaten per fox birth', Dmin, Dmax, valinit=d0)

def update(val):

    x = [slider_Prey.val, slider_Pred.val]
    newparams = [slider_A.val, slider_B.val, slider_C.val, slider_D.val]
    # recalculate the function values
    prey, predator = odeint(lotka, x, t, args=(newparams,)).T
    # update the value on the graph
    prey_plot.set_ydata(prey)
    predator_plot.set_ydata(predator)
    # redraw the graph
    fig.canvas.draw_idle()
    ax.set_ylim([0, np.max([prey, predator])])

# set both sliders to call update when their value is changed:
slider_Prey.on_changed(update)
slider_Pred.on_changed(update)
slider_A.on_changed(update)
slider_B.on_changed(update)
slider_C.on_changed(update)
slider_D.on_changed(update)

# create the reset button axis (where its drawn)
resetax = plt.axes([0.8, 0.025, 0.1, 0.04])
# and the button itself
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')

def reset(event):
    slider_Prey.reset()
    slider_Pred.reset()
    slider_A.reset()
    slider_B.reset()
    slider_C.reset()
    slider_D.reset()

button.on_clicked(reset)

plt.show()
```

Classic Lotka Volterra Model Phase Space Investigation:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons
from scipy.integrate import odeint
"""
Code adapted from: @berna1111's answer to https://stackoverflow.com/questions
    /43381449/how-to-make-two-sliders-in-matplotlib
```

```
"""
"""
Phase plot attempt:
"""

def lotka(x, t, params):
    N, P = x
    a, b, c, d = params
    derivs = [a*N - b*N*P, c*N*P - d*P]
    return derivs



# Parameters
Preymin = 1
Preymax = 100
Predmin = 1
Predmax = 100
Amin = .01
Bmin = .01
Cmin = .01
Dmin = .01
Amax = 20
Bmax = 20
Cmax = 20
Dmax = 20
Prey_0 = 2
Pred_0 = 2
a0 = 1
b0 = .5
c0 = .5
d0 = 2
a = 1
b = 0.5
c = 0.5
d = 2

params = [a, b, c, d]
x0=[Prey_0,Pred_0]
end_time = 60
tstep = 0.001

# Initial function values
t = np.arange(0, end_time, tstep)
prey, predator = odeint(lotka, x0, t, args=(params,)).T



fig = plt.figure()
ax = fig.add_axes([0.10, 0.5, 0.8, 0.45])
phase_plot = ax.plot(prey, predator, label="Rabbits v. Foxes")[0]

ax.set_xlabel("Rabbits")
```

```
ax.set_ylabel("Foxes")
ax.legend(loc="upper right")
ax.set_title('Rabbit & Foxes Population Model Phase Plot')
ax.grid(color="b", alpha=0.5, linestyle="dashed", linewidth=0.5)
ax.set_ylim([0, np.max([prey, predator])])


axcolor = 'lightgoldenrodyellow'
axis_Prey = fig.add_axes([0.10, 0.1, 0.8, 0.0225], facecolor=axcolor)
axis_Pred = fig.add_axes([0.10, 0.15, 0.8, 0.0225], facecolor=axcolor)
axis_A = fig.add_axes([0.10, 0.20, 0.8, 0.0225], facecolor=axcolor)
axis_B = fig.add_axes([0.10, 0.25, 0.8, 0.0225], facecolor=axcolor)
axis_C = fig.add_axes([0.10, 0.30, 0.8, 0.0225], facecolor=axcolor)
axis_D = fig.add_axes([0.10, 0.35, 0.8, 0.0225], facecolor=axcolor)

# size: [left, bottom, width, height]

# create each slider on its corresponding place:
slider_Prey = Slider(axis_Prey, 'Inital Prey Population', Preymin, Preymax,
    valinit=Prey_0, valstep=1)
slider_Pred = Slider(axis_Pred, 'Initial Predator Population', Predmin, Predmax,
    valinit=Pred_0, valstep=1)
slider_A = Slider(axis_A, 'Growth rate of rabbits', Amin, Amax, valinit=a0)
slider_B = Slider(axis_B, 'Death rate of rabbits by predation', Bmin, Bmax,
    valinit=b0)
slider_C = Slider(axis_C, 'Natural death rate of foxes', Cmin, Cmax, valinit=c0)
slider_D = Slider(axis_D, 'Rabbits eaten per fox birth', Dmin, Dmax, valinit=d0)

def update(val):

    x = [slider_Prey.val, slider_Pred.val]
    newparams = [slider_A.val, slider_B.val, slider_C.val, slider_D.val]
    # recalculate the function values
    prey, predator = odeint(lotka, x, t, args=(newparams,)).T
    # update the value on the graph
    phase_plot.set_xdata(prey)
    phase_plot.set_ydata(predator)
    # redraw the graph
    fig.canvas.draw_idle()
    ax.set_ylim([0, np.max([prey, predator])])
    ax.set_xlim([0, np.max([prey, predator])])

# set both sliders to call update when their value is changed:
slider_Prey.on_changed(update)
slider_Pred.on_changed(update)
slider_A.on_changed(update)
slider_B.on_changed(update)
slider_C.on_changed(update)
slider_D.on_changed(update)

# create the reset button axis (where its drawn)
resetax = plt.axes([0.8, 0.025, 0.1, 0.04])
# and the button itself
```

```
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')

def reset(event):
    slider_Prey.reset()
    slider_Pred.reset()
    slider_A.reset()
    slider_B.reset()
    slider_C.reset()
    slider_D.reset()

button.on_clicked(reset)

plt.show()
```

Lotka Volterra Model with Reduced Parameters:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons
from scipy.integrate import odeint

def lotka(x, t, params):
    N, P = x
    alpha,beta = params
    derivs = [N - N*P - (alpha * N**2), N*P - (beta*P)]
    return derivs



# Parameters
Nmin = 1
Nmax = 100

Pmin = 1
Pmax = 100

Alphamin = .01
Betamin = .01

Alphamax = 10
Betamax = 10

N0 = 2
P0 = 2
alpha0 = 1
beta0 = .5

alpha = 1
beta = 0.5


params = [alpha, beta]
x0=[N0,P0]
```

```
maxt = 60
tstep = 0.01

# Initial function values
t = np.arange(0, maxt, tstep)
prey, predator = odeint(lotka, x0, t, args=(params,)).T
# odeint returne a shape (2000, 2) array, with the value for
# each population in [[n_preys, n_predators], ...]
# The .T at the end transponses the array, so now we get each population
# over time in each line of the resultint (2, 2000) array.

# Create a figure and an axis to plot in:
fig = plt.figure()
ax = fig.add_axes([0.10, 0.5, 0.8, 0.3])
prey_plot = ax.plot(t, prey, label="Rabbits")[0]
predator_plot = ax.plot(t, predator, label="Foxes")[0]

ax.set_xlabel("Time")
ax.set_ylabel("Population size")
ax.legend(loc="upper right")
ax.set_title('Rabbit & Foxes Static Model')
ax.grid(color="b", alpha=0.5, linestyle="dashed", linewidth=0.5)
ax.set_ylim([0, np.max([prey, predator])])

# create a space in the figure to place the two sliders:
axcolor = 'lightgoldenrodyellow'
axis_N = fig.add_axes([0.10, 0.1, 0.8, 0.03], facecolor=axcolor)
axis_P = fig.add_axes([0.10, 0.15, 0.8, 0.03], facecolor=axcolor)
axis_Alpha = fig.add_axes([0.10, 0.20, 0.8, 0.03], facecolor=axcolor)
axis_Beta = fig.add_axes([0.10, 0.25, 0.8, 0.03], facecolor=axcolor)
# the first argument is the rectangle, with values in percentage of the figure
# size: [left, bottom, width, height]

# create each slider on its corresponding place:
slider_N = Slider(axis_N, 'N', Nmin, Nmax, valinit=N0, valstep=1)
slider_P = Slider(axis_P, 'P', Pmin, Pmax, valinit=P0, valstep=1)
slider_Alpha = Slider(axis_Alpha, 'Alpha', Alphamin, Alphamax, valinit=alpha0)
slider_Beta = Slider(axis_Beta, 'Beta', Betamin, Betamax, valinit=beta0)


def update(val):
    # retrieve the values from the sliders
    x = [slider_N.val, slider_P.val]
    newparams = [slider_Alpha.val, slider_Beta.val]
    # recalculate the function values
    prey, predator = odeint(lotka, x, t, args=(newparams,)).T
    # update the value on the graph
    prey_plot.set_ydata(prey)
    predator_plot.set_ydata(predator)
    # redraw the graph
    fig.canvas.draw_idle()
    ax.set_ylim([0, np.max([prey, predator])])

```

```
# set both sliders to call update when their value is changed:
slider_N.on_changed(update)
slider_P.on_changed(update)
slider_Alpha.on_changed(update)
slider_Beta.on_changed(update)


# create the reset button axis (where its drawn)
resetax = plt.axes([0.8, 0.025, 0.1, 0.04])
# and the button itself
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')

def reset(event):
    slider_N.reset()
    slider_P.reset()
    slider_Alpha.reset()
    slider_Beta.reset()

button.on_clicked(reset)

plt.show()
```

Lotka Volterra Model with Reduced Parameters Phase Space Investigation:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons
from scipy.integrate import odeint

# phase part 3:


def lotka(x, t, params):
    N, P = x
    alpha,beta = params
    derivs = [N - N*P - (alpha * N**2), N*P - (beta*P)]
    return derivs



# Parameters
Preymin = 1
Preymax = 100

Predmin = 1
Predmax = 100

Alphamin = 1
Betamin = .1

Alphamax = 10
Betamax = 10
```

```
Prey_0 = 2
Pred_0 = 2
alpha0 = 1
beta0 = .5

alpha = 1
beta = 0.5


params = [alpha, beta]
x0=[Prey_0,Pred_0]
maxt = 30
tstep = 0.01


t = np.arange(0, maxt, tstep)
prey, predator = odeint(lotka, x0, t, args=(params,)).T



fig = plt.figure()
ax = fig.add_axes([0.10, 0.5, 0.8, 0.45])
phase_plot = ax.plot(prey, predator, label="Rabbits v. Foxes")[0]


ax.set_xlabel("Rabbits")
ax.set_ylabel("Foxes")
ax.legend(loc="upper right")
ax.set_title('Rabbit & Foxes Population Model Phase Plot')
ax.grid(color="b", alpha=0.5, linestyle="dashed", linewidth=0.5)
ax.set_ylim([0, np.max([prey, predator])])


axcolor = 'lightgoldenrodyellow'
axis_Prey = fig.add_axes([0.10, 0.1, 0.8, 0.0225], facecolor=axcolor)
axis_Pred = fig.add_axes([0.10, 0.15, 0.8, 0.0225], facecolor=axcolor)
axis_Alpha = fig.add_axes([0.10, 0.20, 0.8, 0.0225], facecolor=axcolor)
axis_Beta = fig.add_axes([0.10, 0.25, 0.8, 0.0225], facecolor=axcolor)
# the first argument is the rectangle, with values in percentage of the figure
# size: [left, bottom, width, height]


slider_Prey = Slider(axis_Prey, 'N', Preymin, Preymax, valinit=Prey_0, valstep=1)
slider_Pred = Slider(axis_Pred, 'P', Predmin, Predmax, valinit=Pred_0, valstep=1)
slider_Alpha = Slider(axis_Alpha, 'Alpha', Alphamin, Alphamax, valinit=alpha0)
slider_Beta = Slider(axis_Beta, 'Beta', Betamin, Betamax, valinit=beta0)


def update(val):

    x = [slider_Prey.val, slider_Pred.val]
    newparams = [slider_Alpha.val, slider_Beta.val]
```

```
    prey, predator = odeint(lotka, x, t, args=(newparams,)).T

    phase_plot.set_xdata(prey)
    phase_plot.set_ydata(predator)

    fig.canvas.draw_idle()
    ax.set_ylim([0, np.max([prey, predator])])


slider_Prey.on_changed(update)
slider_Pred.on_changed(update)
slider_Alpha.on_changed(update)
slider_Beta.on_changed(update)



resetax = plt.axes([0.8, 0.025, 0.1, 0.04])

button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')

def reset(event):
    slider_Prey.reset()
    slider_Pred.reset()
    slider_Alpha.reset()
    slider_Beta.reset()

button.on_clicked(reset)

plt.show()
```

Classic Lotka Volterra Model with Added Species:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons
from scipy.integrate import odeint
"""
Code adapted from: @berna1111's answer to https://stackoverflow.com/questions
    /43381449/how-to-make-two-sliders-in-matplotlib

"""


def lotka(x, t, params):
    N, P, Dr = x
    a, b, c, d, e, f, g, h, i = params
    derivs = [a*N - b*N*P - e*N*Dr, c*N*P - f*Dr*P - d*P, g*N*Dr + h*P*Dr - i*Dr]
    return derivs



# Parameters
Preymin = 1
```

```
Preymax = 100
Predmin = 1
Predmax = 100
Dragmin = 1
Dragmax = 100
Amin = .01
Bmin = .01
Cmin = .01
Dmin = .01
Emin = .01
Fmin = .01
Gmin = .01
Hmin = .01
Imin = .01
Amax = 20
Bmax = 20
Cmax = 20
Dmax = 20
Emax = 20
Fmax = 20
Gmax = 20
Hmax = 20
Imax = 20
Prey_0 = 20
Pred_0 = 20
Drag_0 = 1
a0 = 1
b0 = .5
c0 = .5
d0 = 2
e0 = 2
f0 = 2
g0 = 2
h0 = 2
i0 = 2
a = .3
b = 0.5
c = 0.5
d = .5
e = .5
f = .5
g = .3
h = .8
i = .5
params = [a, b, c, d, e, f, g, h, i]
x0=[Prey_0,Pred_0,Drag_0]
end_time = 60
tstep = 0.001

# Initial function values
t = np.arange(0, end_time, tstep)
prey, predator, dragons = odeint(lotka, x0, t, args=(params,)).T
```

```
75
fig = plt.figure()
ax = fig.add_axes([0.15, 0.75, 0.8, .2])
prey_plot = ax.plot(t, prey, label="Rabbits")[0]
predator_plot = ax.plot(t, predator, label="Foxes")[0]
80 dragon_plot = ax.plot(t,dragons,label='Dragons')[0]

ax.set_xlabel("Time")
ax.set_ylabel("Population size")
ax.legend(loc="upper right")
85 ax.set_title('Rabbit, Foxes, and Dragons, Oh my')
ax.grid(color="b", alpha=0.5, linestyle="dashed", linewidth=0.5)
ax.set_ylim([0, np.max([prey, predator])])


90 axcolor = 'lightgoldenrodyellow'
axis_Prey = fig.add_axes([0.4, 0.074, 0.45, 0.025], facecolor=axcolor)
axis_Pred = fig.add_axes([0.4, 0.115, 0.45, 0.025], facecolor=axcolor)
axis_Drag = fig.add_axes([0.4, 0.16, 0.45, 0.025], facecolor=axcolor)
axis_A = fig.add_axes([0.4, 0.2, 0.45, 0.025], facecolor=axcolor)
95 axis_B = fig.add_axes([0.4, 0.25, 0.45, 0.025], facecolor=axcolor)
axis_C = fig.add_axes([0.4, 0.3, 0.45, 0.025], facecolor=axcolor)
axis_D = fig.add_axes([0.4, 0.35, 0.45, 0.025], facecolor=axcolor)
axis_E = fig.add_axes([0.4, 0.4, 0.45, 0.025], facecolor=axcolor)
axis_F = fig.add_axes([0.4, 0.45, 0.45, 0.025], facecolor=axcolor)
100 axis_G = fig.add_axes([0.4, 0.5, 0.45, 0.025], facecolor=axcolor)
axis_H = fig.add_axes([0.4, 0.55, 0.45, 0.025], facecolor=axcolor)
axis_I = fig.add_axes([0.4, 0.6, 0.45, 0.025], facecolor=axcolor)

# size: [left, bottom, width, height]
105

# create each slider on its corresponding place:
slider_Prey = Slider(axis_Prey, 'Inital Rabbit Population', Preymin, Preymax,
    valinit=Prey_0, valstep=1)
slider_Pred = Slider(axis_Pred, 'Initial Fox Population', Predmin, Predmax,
    valinit=Pred_0, valstep=1)
slider_Drag = Slider(axis_Drag, 'Initial Dragon Population',Dragmin,Dragmax,
    valinit=Drag_0,valstep =1)
110 slider_A = Slider(axis_A, 'Birth rate of rabbits', Amin, Amax, valinit=a0)
slider_B = Slider(axis_B, 'Death rate of rabbits by foxes', Bmin, Bmax, valinit=b0
    )
slider_C = Slider(axis_C, 'Rabbits eaten per fox birth', Cmin, Cmax, valinit=c0)
slider_D = Slider(axis_D, 'Natural death rate of foxes', Dmin, Dmax, valinit=d0)
slider_E = Slider(axis_E, 'Death rate of rabbits by dragons', Emin, Emax, valinit=
    e0)
115 slider_F = Slider(axis_F, 'Death rate of foxes by dragons', Fmin, Fmax, valinit=f0
    )
slider_G = Slider(axis_G, 'Rabbits eaten per dragon birth', Gmin, Gmax, valinit=g0
    )
slider_H = Slider(axis_H, 'Foxes eaten per dragon birth', Hmin, Hmax, valinit=h0)
slider_I = Slider(axis_I, 'Natural death rate of dragons', Imin, Imax, valinit=i0)
```

```
def update(val):

    x = [slider_Prey.val, slider_Pred.val, slider_Drag.val]
    newparams = [slider_A.val, slider_B.val, slider_C.val, slider_D.val,slider_E.
    val, slider_F.val, slider_G.val, slider_H.val, slider_I.val]
    # recalculate the function values
    prey, predator, dragons = odeint(lotka, x, t, args=(newparams,)).T
    # update the value on the graph
    prey_plot.set_ydata(prey)
    predator_plot.set_ydata(predator)
    dragon_plot.set_ydata(dragons)
    # redraw the graph
    fig.canvas.draw_idle()
    ax.set_ylim([0, np.max([prey, predator])])

# set both sliders to call update when their value is changed:
slider_Prey.on_changed(update)
slider_Pred.on_changed(update)
slider_A.on_changed(update)
slider_B.on_changed(update)
slider_C.on_changed(update)
slider_D.on_changed(update)
slider_E.on_changed(update)
slider_F.on_changed(update)
slider_G.on_changed(update)
slider_H.on_changed(update)
slider_I.on_changed(update)

# create the reset button axis (where its drawn)
resetax = plt.axes([0.01, 0.1, 0.1, 0.04])
# and the button itself
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')

def reset(event):
    slider_Prey.reset()
    slider_Pred.reset()
    slider_A.reset()
    slider_B.reset()
    slider_C.reset()
    slider_D.reset()
    slider_E.reset()
    slider_F.reset()
    slider_G.reset()
    slider_H.reset()
    slider_I.reset()

button.on_clicked(reset)

plt.show()
```

Lotka Volterra Model with Reduced Parameters and with Added Species:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons
from scipy.integrate import odeint
"""
Code adapted from: @berna1111's answer to https://stackoverflow.com/questions
    /43381449/how-to-make-two-sliders-in-matplotlib

"""


def lotka(x, t, params):
    N, P, Dr = x
    a, b, c = params
    derivs = [N - N*P - a*N**2, N*P - Dr*P - b*P, P*Dr - c*Dr]
    return derivs



# Parameters
Preymin = 1
Preymax = 100
Predmin = 1
Predmax = 100
Dragmin = 1
Dragmax = 100
Amin = .001
Bmin = .001
Cmin = .001
Amax = 5
Bmax = 5
Cmax = 5
Prey_0 = 5
Pred_0 = 5
Drag_0 = 2
a0 = .1
b0 = .5
c0 = .5
a = .3
b = 0.5
c = 0.5
params = [a, b, c]
x0=[Prey_0,Pred_0,Drag_0]
end_time = 60
tstep = 0.001

# Initial function values
t = np.arange(0, end_time, tstep)
prey, predator, dragons = odeint(lotka, x0, t, args=(params,)).T



fig = plt.figure()
ax = fig.add_axes([0.15, 0.45, 0.8, .4])
```

```
prey_plot = ax.plot(t, prey, label="Rabbits")[0]
predator_plot = ax.plot(t, predator, label="Foxes")[0]
dragon_plot = ax.plot(t,dragons,label='Dragons')[0]

ax.set_xlabel("Time")
ax.set_ylabel("Population size")
ax.legend(loc="upper right")
ax.set_title('Rabbits, Foxes, and Dragons, Oh my')
ax.grid(color="b", alpha=0.5, linestyle="dashed", linewidth=0.5)
ax.set_ylim([0, np.max([prey, predator])])


axcolor = 'lightgoldenrodyellow'
axis_Prey = fig.add_axes([0.4, 0.074, 0.45, 0.025], facecolor=axcolor)
axis_Pred = fig.add_axes([0.4, 0.115, 0.45, 0.025], facecolor=axcolor)
axis_Drag = fig.add_axes([0.4, 0.16, 0.45, 0.025], facecolor=axcolor)
axis_A = fig.add_axes([0.4, 0.2, 0.45, 0.025], facecolor=axcolor)
axis_B = fig.add_axes([0.4, 0.25, 0.45, 0.025], facecolor=axcolor)
axis_C = fig.add_axes([0.4, 0.3, 0.45, 0.025], facecolor=axcolor)


# size: [left, bottom, width, height]

# create each slider on its corresponding place:
slider_Prey = Slider(axis_Prey, 'Inital Rabbit Population', Preymin, Preymax,
    valinit=Prey_0, valstep=1)
slider_Pred = Slider(axis_Pred, 'Initial Fox Population', Predmin, Predmax,
    valinit=Pred_0, valstep=1)
slider_Drag = Slider(axis_Drag, 'Initial Dragon Population',Dragmin,Dragmax,
    valinit=Drag_0,valstep =1)
slider_A = Slider(axis_A, 'Rabbit Food Pressure', Amin, Amax, valinit=a0)
slider_B = Slider(axis_B, 'Natural Death Rate of Foxes', Bmin, Bmax, valinit=b0)
slider_C = Slider(axis_C, 'Natural Death Rate of Dragons', Cmin, Cmax, valinit=c0)


def update(val):

    x = [slider_Prey.val, slider_Pred.val, slider_Drag.val]
    newparams = [slider_A.val, slider_B.val, slider_C.val]
    # recalculate the function values
    prey, predator, dragons = odeint(lotka, x, t, args=(newparams,)).T
    # update the value on the graph
    prey_plot.set_ydata(prey)
    predator_plot.set_ydata(predator)
    dragon_plot.set_ydata(dragons)
    # redraw the graph
    fig.canvas.draw_idle()
    ax.set_ylim([0, np.max([prey, predator])])

# set both sliders to call update when their value is changed:
slider_Prey.on_changed(update)
slider_Pred.on_changed(update)
slider_A.on_changed(update)
```

```
slider_B.on_changed(update)
slider_C.on_changed(update)

# create the reset button axis (where its drawn)
resetax = plt.axes([0.01, 0.1, 0.1, 0.04])
# and the button itself
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')

def reset(event):
    slider_Prey.reset()
    slider_Pred.reset()
    slider_A.reset()
    slider_B.reset()
    slider_C.reset()


button.on_clicked(reset)

plt.show()
```