Predator Prey Model Implementation in Python

***Abstract*— Stochastic spatial models are turning into a famous tool for comprehend the environmental and development of biological system issues. Biological frameworks can be investigated through mathematics. Some new hypothesis can be generated by researchers or scientists better than with only experimental observations with the help of numerical models. Biological problems can be translated into an assortment of numerical equations from which solutions are visualized. Here we would like to discuss a model named Lotka-Volterra, which represents the populace of two categories i.e. Predator and another one is prey [1].**

Predator prey can be considered as interlink in term of stochastic portrayal of this Lotka-Volterra model and investigate the utilization of stochastic processes to annihilation conduct of the interlinking population. Simulations for stochastic processes of continuous time Lotka-Volterra are presented by us. Here we use Euler method to solve the predator prey system [2]

Introduction

First we would like to know what predator and prey really means. A living creature which has been eaten by another living creature is known as prey and on the other hand a living creature who eats another living creature is known as predator. For example in animals only, lion eats zebra, bear eats fish, fox eats rabbit etc. In plants, berries have been eaten by a bear, lettuce has been eaten by a rabbit, and leaves have been eaten by a grasshopper [2].

Predator and Prey grows together. Prey plays an essential role in the environment of predator because predator should eat the prey for its survival, so it evolves the necessary skills to do so, which includes its speed, stealth, camouflage (to hide while approaching the prey), good senses of smell, sight, and hearing (to locate the prey), immunity to the prey's poison, poison (to kill the prey), the appropriate mouth and digestive system, and more. On the other hand a predator plays an essential role in the environment of prey. The prey also evolved whatever was required to avoid being eaten because the predator is a part of the prey's environment and the prey dies if it is eaten by the predator. Examples include speed, camouflage, a keen sense of smell, sight, or hearing, thorns, poison, etc. [2]

Predation's numerical models are the most ancient in the ecology. The rise and fall of the Adriatic fishing fleets is thought to have inspired the thoughts of the Italian mathematician Volterra regarding predation. The success of others attracted more fishermen when the fishing was good. After a while, the number of fishermen also decreased as the fish population began to dwindle, possibly as a result of overfishing. The cycle eventually repeated. Understanding what causes these oscillations is a major goal of population interaction modelling), etc. In fact, the outcome of such an effort is the first Lotka-Volterra system. [2]

I. LODKA-VOLTERRA EQUATIONS

$$Dx/dt = \alpha x - \beta xy$$
$$Dy/dt = \sigma xy - \gamma y$$

A. *Parameters*

Dx/dt = Growth rate of rabbit population

Dy/dt = Growth rate of fox population

x = Rabbit (Prey) population

y = Fox (Predator) population

α = Growth rate of rabbits

β = Death rates of rabbits due to predation from foxes

γ = Natural death rate of foxes

σ = Factors describing how many consumed rabbits can create more foxes. [3]

B. *Meaning of the Equations*

According to the above equations, many several assumptions for the dynamics of predator-prey systems have been made by Vodka-Volterra model. There are some of the assumptions

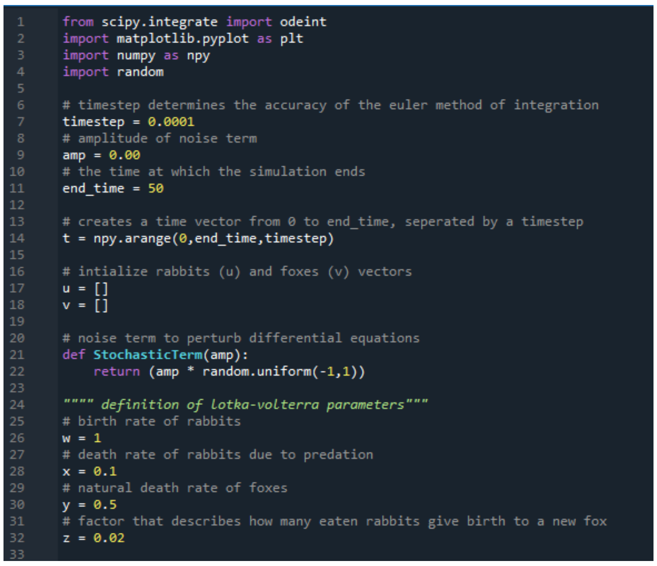
- Unlimited food has been provided to the rabbit population.
- Rate of change of size of the populations are proportional to their size.
- Foxes are dependent on rabbits, they recognized rabbits as the only food source in the world.
- The foxes have unlimited hunger.[3]

II. LODKA-VOLTERRA MODEL

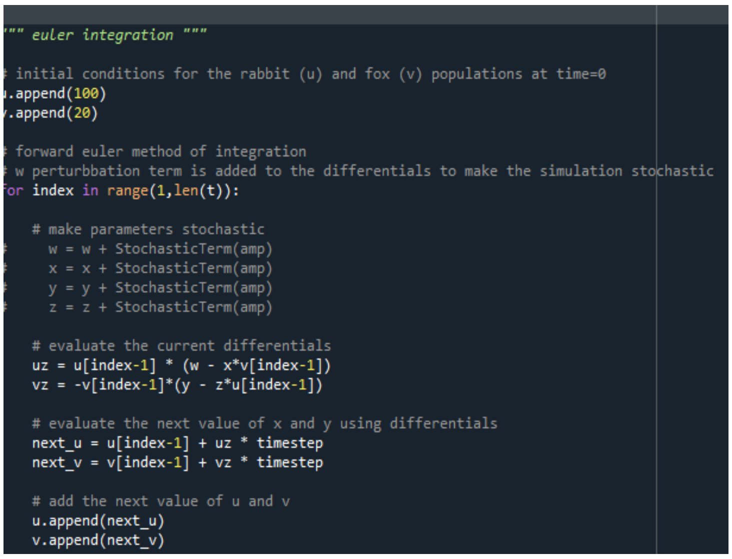
In the study of classification interlinkage in an ecological circumstances deterministic numerical models have been commonly used. According to the classical representation of predator prey interactions as put together by Lotka- Volterra model, the population of prey grows aggressively when the predators are not available. Firstly, a deterministic, idealized model has been built for predator-prey systems. In order to better represent the noise that occurs in real-world circumstances, stochastics will be included to the equations toward the end. [3]

A. *Simulation*

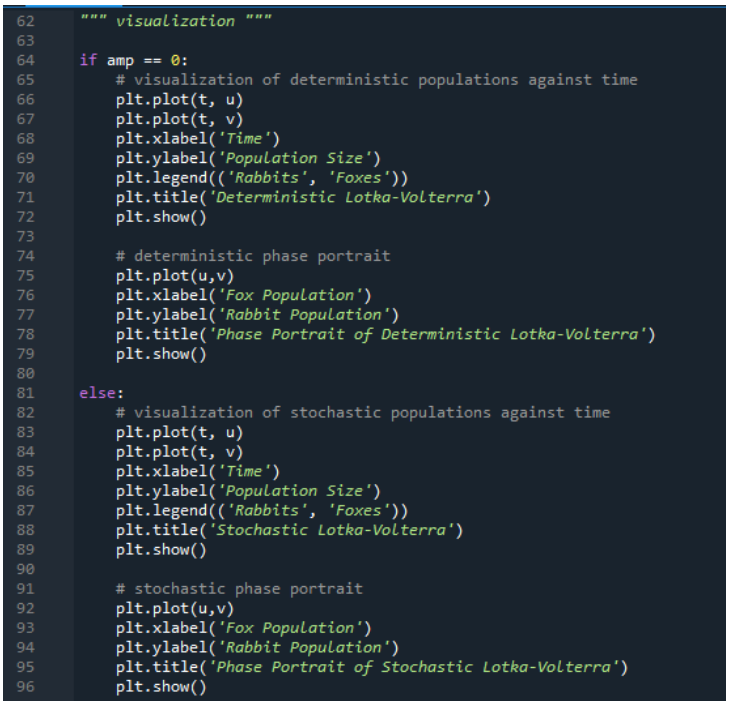
Now we have to use the deterministic Lodka-Volterra Model to replicate the behavior of rabbit and fox populations without including stochastic noise in the simulation.



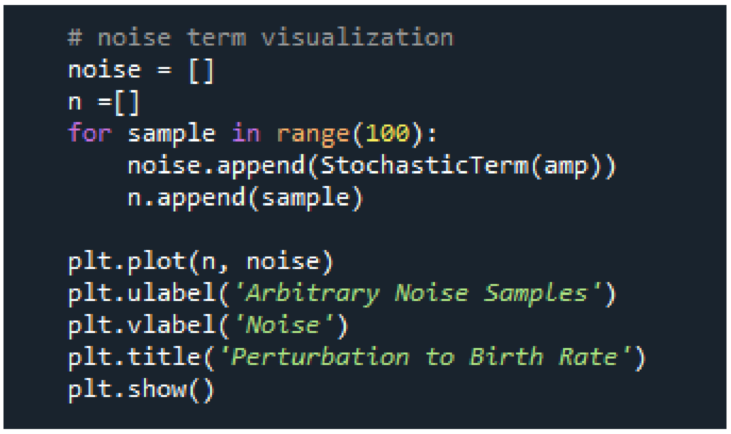
```python
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as npy
import random

# timestep determines the accuracy of the euler method of integration
timestep = 0.0001
# amplitude of noise term
amp = 0.00
# the time at which the simulation ends
end_time = 50

# creates a time vector from 0 to end_time, seperated by a timestep
t = npy.arange(0,end_time,timestep)

# intialize rabbits (u) and foxes (v) vectors
u = []
v = []

# noise term to perturb differential equations
def StochasticTerm(amp):
    return (amp * random.uniform(-1,1))

"""" definition of lotka-volterra parameters"""
# birth rate of rabbits
w = 1
# death rate of rabbits due to predation
x = 0.1
# natural death rate of foxes
y = 0.5
# factor that describes how many eaten rabbits give birth to a new fox
z = 0.02

```



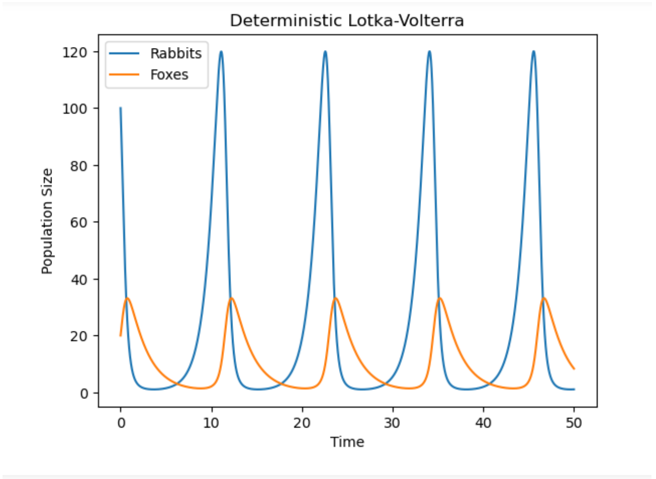
```python
"" euler integration """

# initial conditions for the rabbit (u) and fox (v) populations at time=0
.append(100)
.append(20)

# forward euler method of integration
# w perturbbation term is added to the differentials to make the simulation stochastic
or index in range(1,len(t)):

    # make parameters stochastic
#     w = w + StochasticTerm(amp)
#     x = x + StochasticTerm(amp)
#     y = y + StochasticTerm(amp)
#     z = z + StochasticTerm(amp)

    # evaluate the current differentials
    uz = u[index-1] * (w - x*v[index-1])
    vz = -v[index-1]*(y - z*u[index-1])

    # evaluate the next value of x and y using differentials
    next_u = u[index-1] + uz * timestep
    next_v = v[index-1] + vz * timestep

    # add the next value of u and v
    u.append(next_u)
    v.append(next_v)
```

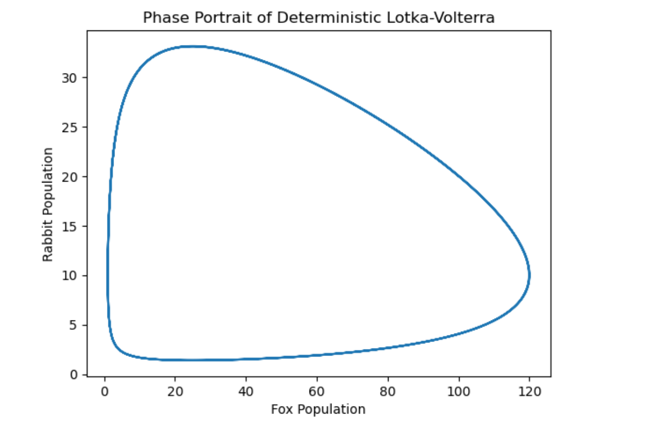


```python
""" visualization """

if amp == 0:
    # visualization of deterministic populations against time
    plt.plot(t, u)
    plt.plot(t, v)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Deterministic Lotka-Volterra')
    plt.show()

    # deterministic phase portrait
    plt.plot(u,v)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Deterministic Lotka-Volterra')
    plt.show()

else:
    # visualization of stochastic populations against time
    plt.plot(t, u)
    plt.plot(t, v)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Stochastic Lotka-Volterra')
    plt.show()

    # stochastic phase portrait
    plt.plot(u,v)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Stochastic Lotka-Volterra')
    plt.show()
```



```python
# noise term visualization
noise = []
n =[]
for sample in range(100):
    noise.append(StochasticTerm(amp))
    n.append(sample)

plt.plot(n, noise)
plt.ulabel('Arbitrary Noise Samples')
plt.vlabel('Noise')
plt.title('Perturbation to Birth Rate')
plt.show()
```





B. *Explaination*

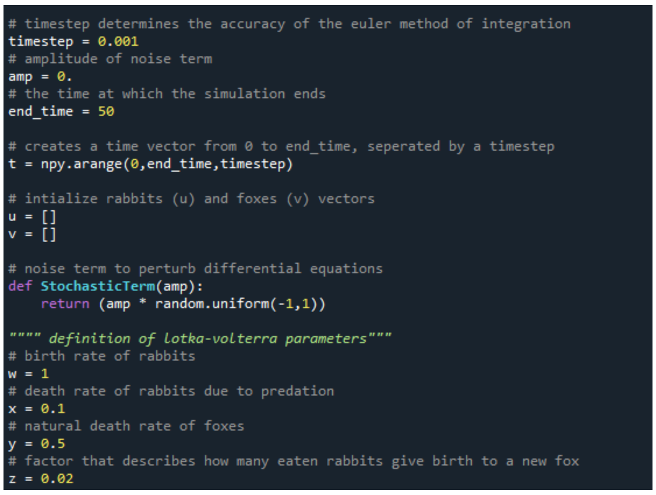
- We have analyzed that foxes and rabbits have an oscillating relationship in the first graph. As increase in the number of rabbits, giving the foxes more food. The production of more foxes have increased. Fox numbers have increased, and they are now consuming the rabbit population more quickly. The abundance of foxes causes the rabbit population to decline, which in turn causes the fox population to decline since they have less to eat
- We can see a closed loop for the phase portrait in the second graph. This proves that, when subjected to the Euler technique, these differential equations are stable.
- The integration technique selected for analyzing the rabbit and fox populations is to blame for the errors in our numerical simulation.
- The Euler technique was selected as the approach for numerical integration. The step-size used, which adds to a discretization error, and any rounding errors are inherent faults in this method.
- The discretization error, which makes up the majority of the errors in our program, can be reduced by reducing the step-size. It is important to note that when the step-size increases, the observed closed-loop spirals and the populations diverge. This demonstrates how the equations are "unstable" mathematically.

C. *Steady States of the Populations*

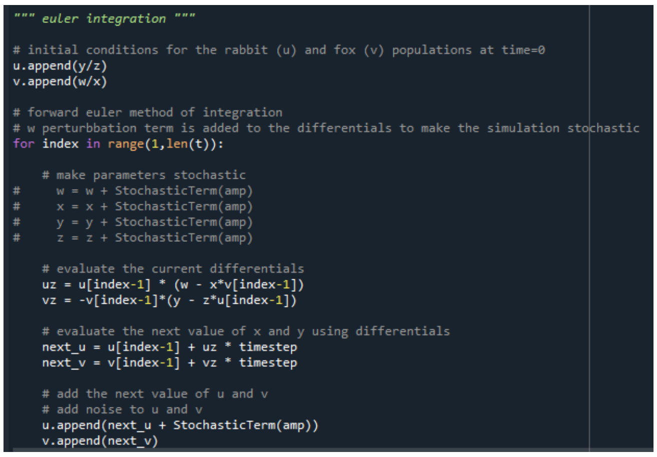
When growth rates are equal to zero, populations reach a stable state. We arrive at two fixed points for the rabbit and fox populations at this point in equilibrium.

x = c/d

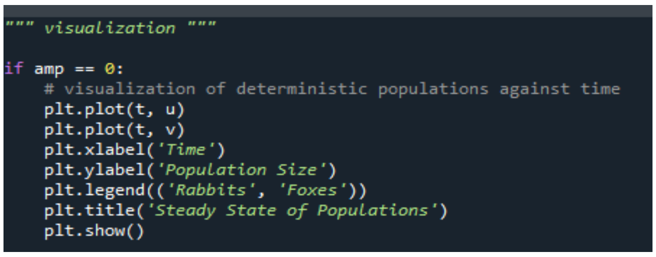
y = a/b



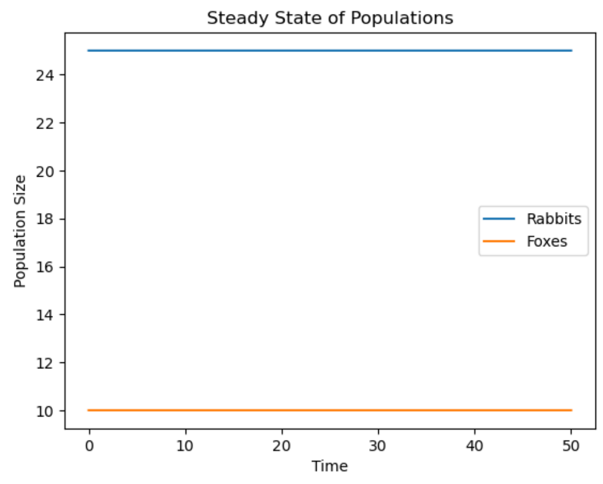
```
# timestep determines the accuracy of the euler method of integration
timestep = 0.001
# amplitude of noise term
amp = 0.
# the time at which the simulation ends
end_time = 50

# creates a time vector from 0 to end_time, seperated by a timestep
t = npy.arange(0,end_time,timestep)

# intialize rabbits (u) and foxes (v) vectors
u = []
v = []

# noise term to perturb differential equations
def StochasticTerm(amp):
    return (amp * random.uniform(-1,1))

"""" definition of lotka-volterra parameters"""
# birth rate of rabbits
w = 1
# death rate of rabbits due to predation
x = 0.1
# natural death rate of foxes
y = 0.5
# factor that describes how many eaten rabbits give birth to a new fox
z = 0.02
```



```
""" euler integration """

# initial conditions for the rabbit (u) and fox (v) populations at time=0
u.append(y/z)
v.append(w/x)

# forward euler method of integration
# w perturbbation term is added to the differentials to make the simulation stochastic
for index in range(1,len(t)):

    # make parameters stochastic
#     w = w + StochasticTerm(amp)
#     x = x + StochasticTerm(amp)
#     y = y + StochasticTerm(amp)
#     z = z + StochasticTerm(amp)

    # evaluate the current differentials
    uz = u[index-1] * (w - x*v[index-1])
    vz = -v[index-1]*(y - z*u[index-1])

    # evaluate the next value of x and y using differentials
    next_u = u[index-1] + uz * timestep
    next_v = v[index-1] + vz * timestep

    # add the next value of u and v
    # add noise to u and v
    u.append(next_u + StochasticTerm(amp))
    v.append(next_v)
```



```
""" visualization """

if amp == 0:
    # visualization of deterministic populations against time
    plt.plot(t, u)
    plt.plot(t, v)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Steady State of Populations')
    plt.show()
```



D. *Perturbing the Rabbit population out of Steady State*

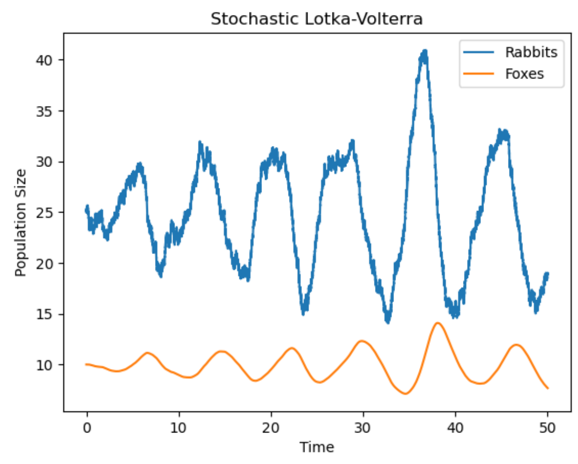
In this instance, we begin with initial populations that reflect the steady state. However, as we proceed via the Euler technique, we introduce a stochastic term that perturbs the freshly formed rabbit populations. We notice that the rabbit population is perturbed out of its steady state by the noise produced by the stochastic term, which then starts the fox population oscillating in time with the rabbit population. [4]

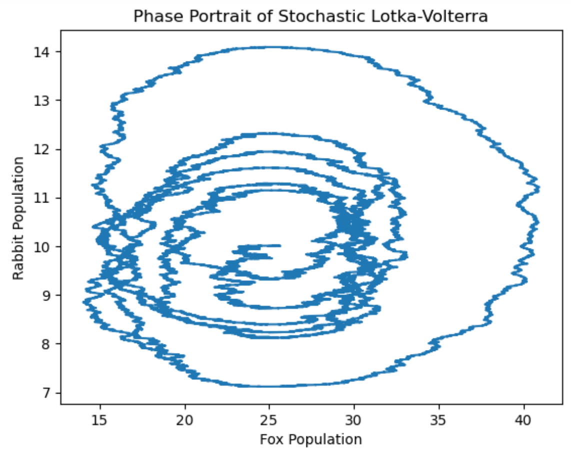
```python
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as npy
import random

# timestep determines the accuracy of the euler method of integration
timestep = 0.001
# amplitude of noise term
amp = 0.1
# the time at which the simulation ends
end_time = 50

# creates a time vector from 0 to end_time, seperated by a timestep
t = npy.arange(0,end_time,timestep)

# intialize rabbits (x) and foxes (y) vectors
x = []
y = []

# noise term to perturb differential equations
def StochasticTerm(amp):
    return (amp * random.uniform(-1,1))

"""" definition of lotka-volterra parameters"""
# birth rate of rabbits
a = 1
# death rate of rabbits due to predation
b = 0.1
```

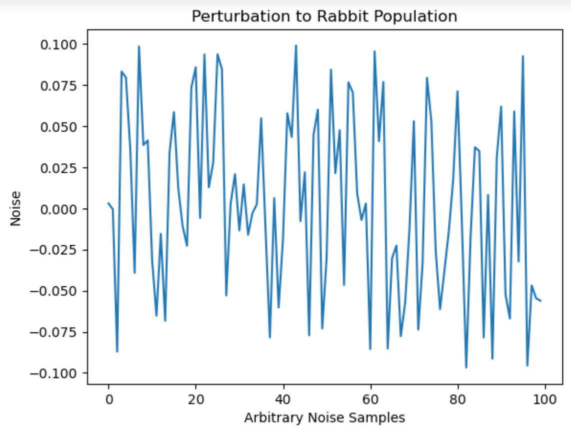
```python
# natural death rate of foxes
c = 0.5
# factor that describes how many eaten rabbits give birth to a new fox
d = 0.02

""" euler integration """

# initial conditions for the rabbit (x) and fox (y) populations at time=0
x.append(c/d)
y.append(a/b)

# forward euler method of integration
# a perturbbation term is added to the differentials to make the simulation stochastic
for index in range(1,len(t)):

    # make parameters stochastic
#     a = a + StochasticTerm(amp)
#     b = b + StochasticTerm(amp)
#     c = c + StochasticTerm(amp)
#     d = d + StochasticTerm(amp)

    # evaluate the current differentials
    xd = x[index-1] * (a - b*y[index-1])
    yd = -y[index-1]*(c - d*x[index-1])

    # evaluate the next value of x and y using differentials
    next_x = x[index-1] + xd * timestep
    next_y = y[index-1] + yd * timestep
```

```python
    # add the next value of x and y
    # add noise to x and y
    x.append(next_x + StochasticTerm(amp))
    y.append(next_y)

""" visualization """

if amp == 0:
    # visualization of deterministic populations against time
    plt.plot(t, x)
    plt.plot(t, y)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Deterministic Lotka-Volterra')
    plt.show()

    # deterministic phase portrait
    plt.plot(x,y)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Deterministic Lotka-Volterra')
    plt.show()
```

```python
else:
    # visualization of stochastic populations against time
    plt.plot(t, x)
    plt.plot(t, y)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Stochastic Lotka-Volterra')
    plt.show()

    # stochastic phase portrait
    plt.plot(x,y)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Stochastic Lotka-Volterra')
    plt.show()

    # noise term visualization
    noise = []
    n =[]
    for sample in range(100):
        noise.append(StochasticTerm(amp))
        n.append(sample)

    plt.plot(n, noise)
    plt.xlabel('Arbitrary Noise Samples')
    plt.ylabel('Noise')
    plt.title('Perturbation to Rabbit Population')
    plt.show()
```







III. Applying Noise to the Lodka-Volterra Parameters

Here, we implement noise to the rabbit birth rate. In the Lotka-Volterra equations, noise is thus specifically applied to the parameter "a."

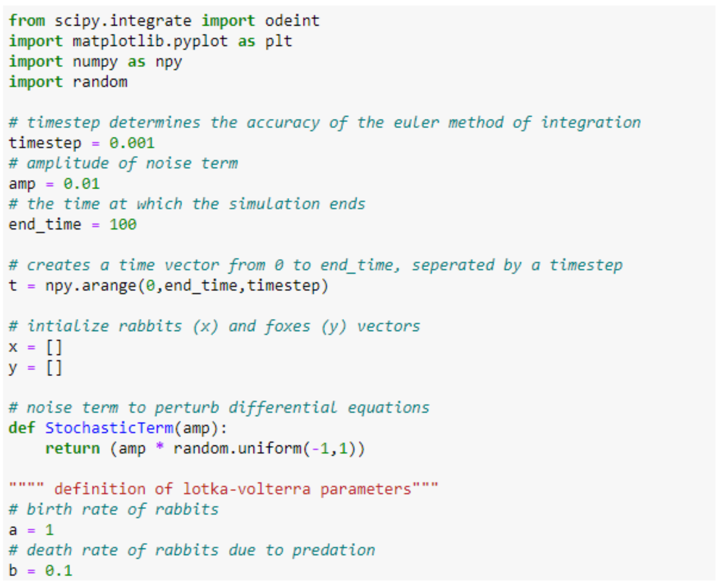
A. *Stochastic Birth Rate*

In this case, the integration defines the birth rate's value, a, as stochastic.

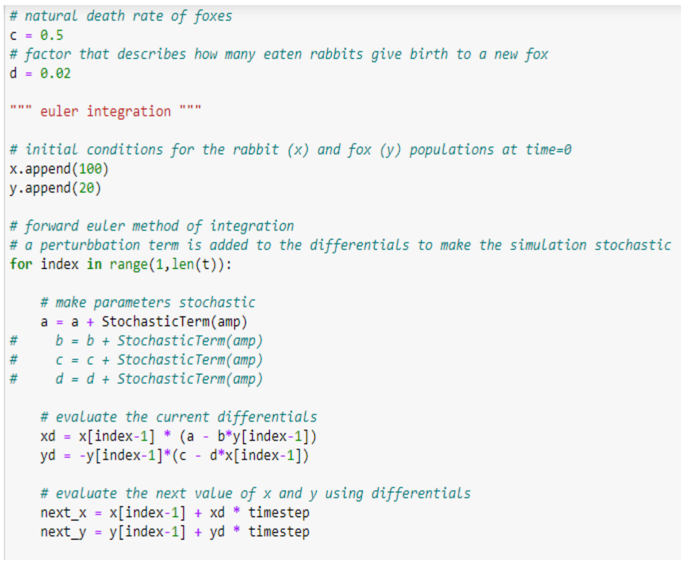
Here is how we define a:

a = a + StochasticTerm

As a result, the value of a has a mean that is similar to its deterministic value, but it also includes noise that varies in direct proportion to the birth rate. [4]

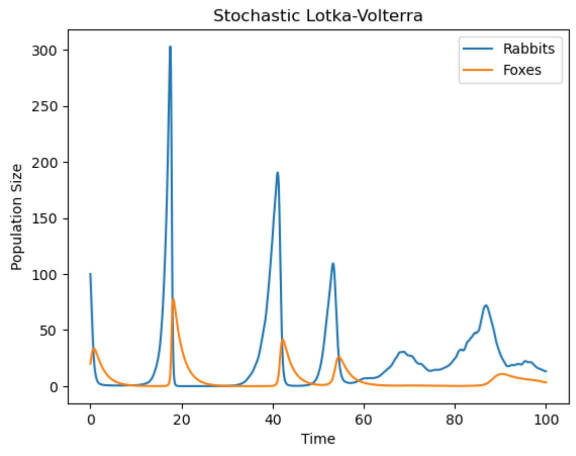


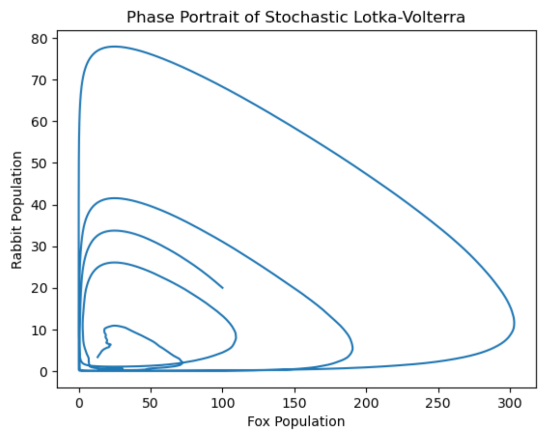
```python
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as npy
import random

# timestep determines the accuracy of the euler method of integration
timestep = 0.001
# amplitude of noise term
amp = 0.01
# the time at which the simulation ends
end_time = 100

# creates a time vector from 0 to end_time, seperated by a timestep
t = npy.arange(0,end_time,timestep)

# intialize rabbits (x) and foxes (y) vectors
x = []
y = []

# noise term to perturb differential equations
def StochasticTerm(amp):
    return (amp * random.uniform(-1,1))

"""" definition of lotka-volterra parameters"""
# birth rate of rabbits
a = 1
# death rate of rabbits due to predation
b = 0.1
```

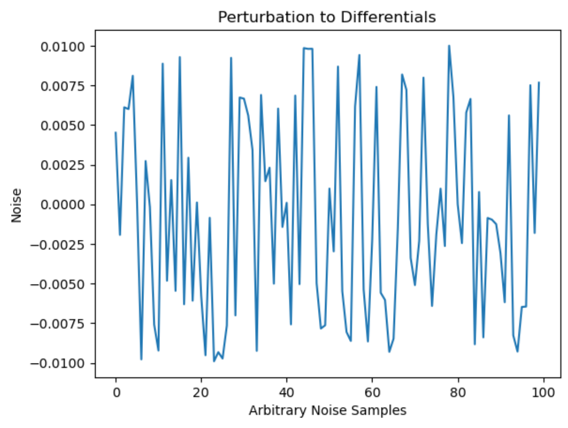


```python
# natural death rate of foxes
c = 0.5
# factor that describes how many eaten rabbits give birth to a new fox
d = 0.02

""" euler integration """

# initial conditions for the rabbit (x) and fox (y) populations at time=0
x.append(100)
y.append(20)

# forward euler method of integration
# a perturbbation term is added to the differentials to make the simulation stochastic
for index in range(1,len(t)):

    # make parameters stochastic
    a = a + StochasticTerm(amp)
#     b = b + StochasticTerm(amp)
#     c = c + StochasticTerm(amp)
#     d = d + StochasticTerm(amp)

    # evaluate the current differentials
    xd = x[index-1] * (a - b*y[index-1])
    yd = -y[index-1]*(c - d*x[index-1])

    # evaluate the next value of x and y using differentials
    next_x = x[index-1] + xd * timestep
    next_y = y[index-1] + yd * timestep
    # add the next value of x and y
    # add noise to x and y
    x.append(next_x)
    y.append(next_y)

""" visualization """

if amp == 0:
    # visualization of deterministic populations against time
    plt.plot(t, x)
    plt.plot(t, y)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Deterministic Lotka-Volterra')
    plt.show()

    # deterministic phase portrait
    plt.plot(x,y)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Deterministic Lotka-Volterra')
    plt.show()
```

```python
else:
    # visualization of stochastic populations against time
    plt.plot(t, x)
    plt.plot(t, y)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Stochastic Lotka-Volterra')
    plt.show()

    # stochastic phase portrait
    plt.plot(x,y)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Stochastic Lotka-Volterra')
    plt.show()

    # noise term visualization
    noise = []
    n =[]
    for sample in range(100):
        noise.append(StochasticTerm(amp))
        n.append(sample)

    plt.plot(n, noise)
    plt.xlabel('Arbitrary Noise Samples')
    plt.ylabel('Noise')
    plt.title('Perturbation to Differentials')
    plt.show()
```



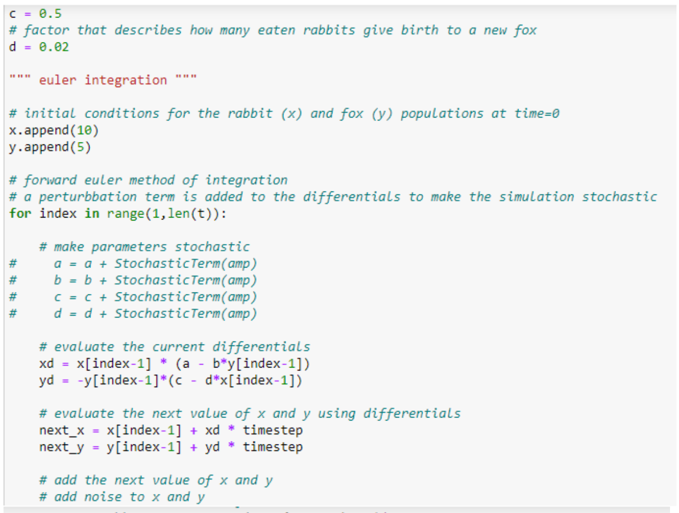




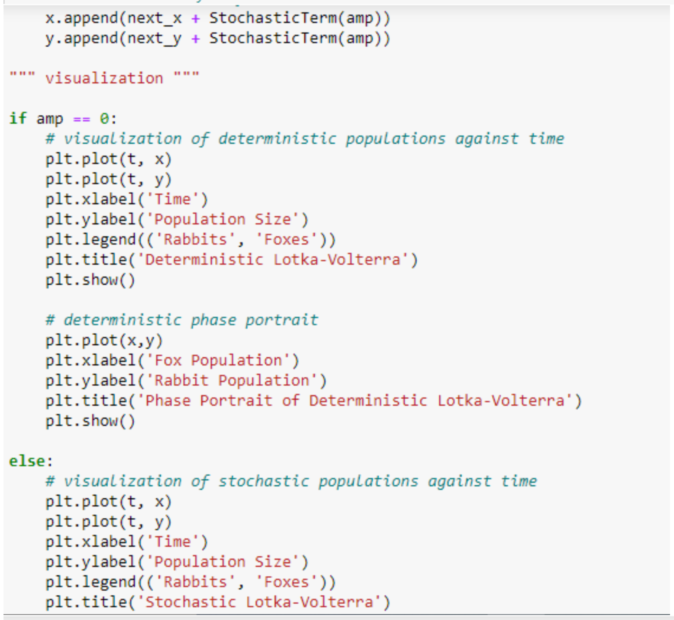
IV. Adding Noise to Populations after Integration

I first use the Euler method to calculate the values of x and y, after which I add noise to these values and save them. These stochastic values of x and y are then used to evaluate the differentials in the subsequent iteration of the Euler method. [4]

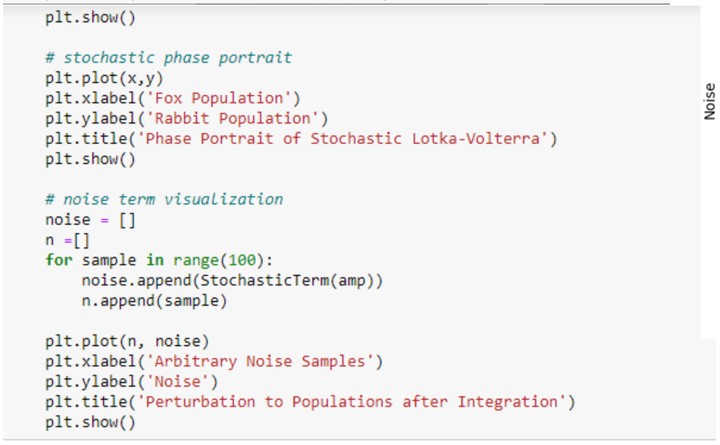
```python
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as npy
import random

# timestep determines the accuracy of the euler method of integration
timestep = 0.001
# amplitude of noise term
amp = 0.1
# the time at which the simulation ends
end_time = 50

# creates a time vector from 0 to end_time, seperated by a timestep
t = npy.arange(0,end_time,timestep)

# intialize rabbits (x) and foxes (y) vectors
x = []
y = []

# noise term to perturb differential equations
def StochasticTerm(amp):
    return (amp * random.uniform(-1,1))

"""" definition of lotka-volterra parameters"""
# birth rate of rabbits
a = 1
# death rate of rabbits due to predation
b = 0.1
# natural death rate of foxes
```



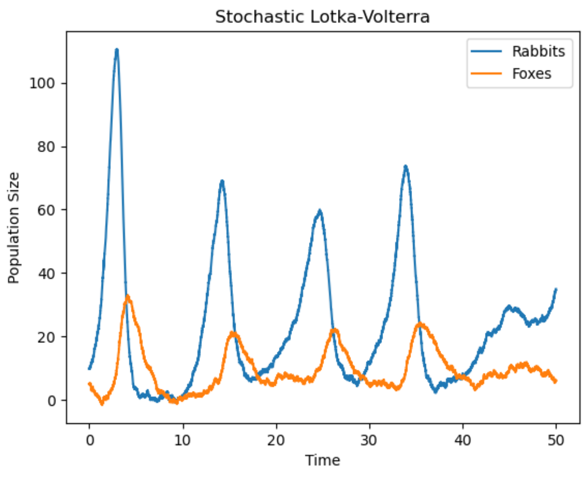
```
c = 0.5
# factor that describes how many eaten rabbits give birth to a new fox
d = 0.02

""" euler integration """

# initial conditions for the rabbit (x) and fox (y) populations at time=0
x.append(10)
y.append(5)

# forward euler method of integration
# a perturbbation term is added to the differentials to make the simulation stochastic
for index in range(1,len(t)):

    # make parameters stochastic
#     a = a + StochasticTerm(amp)
#     b = b + StochasticTerm(amp)
#     c = c + StochasticTerm(amp)
#     d = d + StochasticTerm(amp)

    # evaluate the current differentials
    xd = x[index-1] * (a - b*y[index-1])
    yd = -y[index-1]*(c - d*x[index-1])

    # evaluate the next value of x and y using differentials
    next_x = x[index-1] + xd * timestep
    next_y = y[index-1] + yd * timestep

    # add the next value of x and y
    # add noise to x and y
```

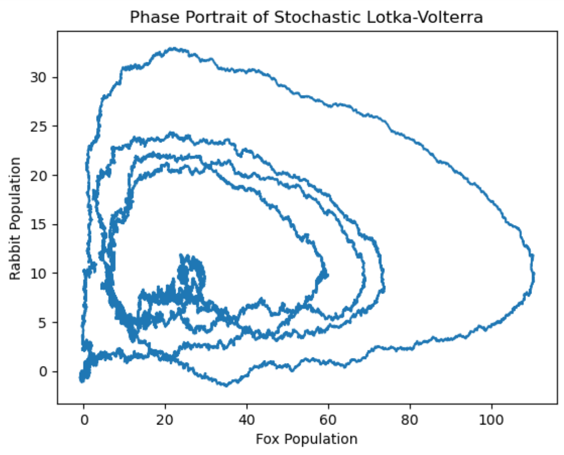


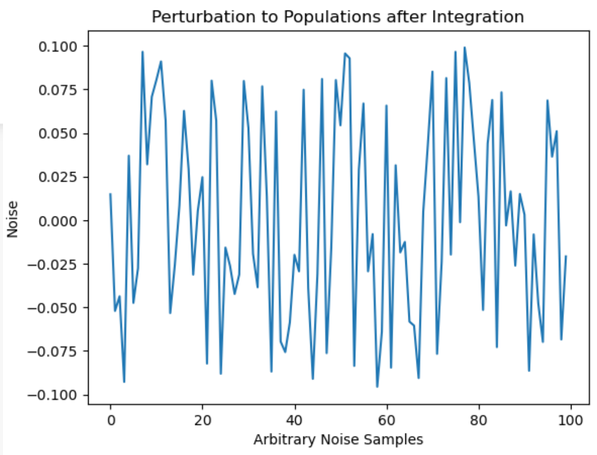
```
    x.append(next_x + StochasticTerm(amp))
    y.append(next_y + StochasticTerm(amp))

""" visualization """

if amp == 0:
    # visualization of deterministic populations against time
    plt.plot(t, x)
    plt.plot(t, y)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Deterministic Lotka-Volterra')
    plt.show()

    # deterministic phase portrait
    plt.plot(x,y)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Deterministic Lotka-Volterra')
    plt.show()

else:
    # visualization of stochastic populations against time
    plt.plot(t, x)
    plt.plot(t, y)
    plt.xlabel('Time')
    plt.ylabel('Population Size')
    plt.legend(('Rabbits', 'Foxes'))
    plt.title('Stochastic Lotka-Volterra')
```



```
    plt.show()

    # stochastic phase portrait
    plt.plot(x,y)
    plt.xlabel('Fox Population')
    plt.ylabel('Rabbit Population')
    plt.title('Phase Portrait of Stochastic Lotka-Volterra')
    plt.show()

    # noise term visualization
    noise = []
    n =[]
    for sample in range(100):
        noise.append(StochasticTerm(amp))
        n.append(sample)

    plt.plot(n, noise)
    plt.xlabel('Arbitrary Noise Samples')
    plt.ylabel('Noise')
    plt.title('Perturbation to Populations after Integration')
    plt.show()
```







V. Discussion

Here, we see that the populations begin to fluctuate with an increasing amplitude at the beginning of the simulation as the rabbit population is continuously exposed to the uniformly distributed noise.

The fox population undergoes a non-zero shift in its differential when the rabbit population is shaken out of its steady state, which perturbs it as well. As a result, if only one species experiences a slight shift, the other species will also experience

a disruption from its steady state. Both species then exhibit the oscillatory behavior. [4]

VI. Conclusion

Notably, the ratio of the organisms over time (particularly in the simulations) has striking similarities to the Lotka-Volterra equations that are used to model such biological systems. Although it is not often stable, this variance results in fascinating population dynamics. As a result, an increase in prey numbers indicates an increase in predator numbers; an increase in predator numbers means a decrease in prey numbers; a decrease in prey numbers implies a decrease in predator numbers; and finally, an increase in prey numbers implies a decrease in predator numbers.

We developed a model to show the oscillatory link between the populations in the first section. The outcomes are consistent with the oscillatory relationship observed in nature, as shown by the empirical data presented above. The populations' stable states, when neither population altered over time, were then assessed. By adding stochastic noise to one of the steady populations, we demonstrated toward the end of the notebook that both species are upset out of their steady states and start to oscillate. [2]

References

[1] Lawrence Omotosho, Morufu Oyedunsi Olayiwola, Patrick Ozoh Phd, “Simulation of Mathematical Model for Ecological Surveillance of Prey-Predator” , Osun State University.

[2] Taleb A. S. Obaid, “The Predator-Prey Model Simulation”, Department of Computer Science,University of Basra,Basra, IRAQ.

[3] Mohammad Faris Laham, Isthrinayagy Krishnarajah,Abdulqadir Jummat, “A NUMERICAL STUDY ON PREDATOR PREY MODEL”, Institute for Mathematical Research, Universiti Putra Malaysia.

[4] Modelling Predator-Prey Systems in Python

https://github.com/INASIC/predator-prey_systems/blob/master/Modelling%20Predator-Prey%20Systems%20in%20Python.ipynb