Lotka Volterra Predator Prey Model

Pawanpreet Kaur (2020PHY1092)

Preetpal Singh (2020PHY1140)

Anjali (2020PHY1164)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

November 9, 2021

Project Report Submitted to

Dr. Mamta and Dr. H. C. Ramo

as part of internal assessment for the course

“32223902 - Computational Physics Skills”

Abstract

Predator-prey models are arguably the building blocks of the bio- and ecosystems as biomasses are grown out of their resource masses. Species compete, evolve and disperse simply for the purpose of seeking resources to sustain their struggle for their very existence.

One of the most ecological applications of differential equations systems is predator-prey problem. In fact, differential equations are very useful in many areas of applied sciences. However, most of the nature problems involve with some unknown function. In this paper, an environmental case containing two related populations of prey species and predator species is studied. It is expected that two population make influence on the size of each other. Since it involves some assumptions, so this model is quite unrealistic.

Contents

1 Introduction **1**

2 Theory **3**

3 Methodology **9**

3.1 Algorihtms 9

4 Analysis of Numerical Results **12**

5 Summary **19**

A Programs **20**

B Contribution of team mates **28**

1 Introduction

Motivation

As differential equations are one of the most important concepts related to analyse real life phenomenas. We are doing Lotka Volterra Model because it gives us glimpse predator and pre their interaction works and interestingly gives us mathematical formulation through which we can make predictions to take legitimite actions to make this wildlife balance.

History[1]

In the 1920s, the Italian mathematician Vito Volterra proposed a differential equations model to describe the population dynamics of two interacting species of a predator and its prey. He hoped to explain the increasing in predator fish (and so,decreasing in prey fish) in the Adriatic Sea during World War I. Independently, these equations studied by Volterra were derived by Alfred Lotka to describe a hypothetical chemical reaction in which the chemical concentrations oscillate , in the United States. There are many species of animals in nature where one species feeds on another species. The first species and the second one are called predator and prey respectively.

Lotka Volterra Model

The Lotka-Volterra equations, also known as predator-prey equations, were developed to describe the dynamics of biological systems. This system of non-linear differential equations can be described as a more general version of a Kolmogorov model[2] because it focuses only on the predator-prey interactions and ignores competition, disease, and mutualism which the Kolmogorov model includes.

The Lotka-Volterra equations can be written simply as a system of first-order non-linear ordinary differential equations (ODEs). Since the equations are differential in nature, the solutions are deterministic (no randomness is involved, and the same initial conditions will produce the same outcome), and the time is continuous (the generations of predators and prey are continually overlapping).

Predator Prey Equations[1]

The model satisfies the system of ODEs:

$$\frac{dx}{dt} = x(\alpha - \beta y)$$
$$\frac{dy}{dt} = -y(\gamma - \delta x)$$

where,

- x is the number of prey.
- y is the number of some predator.
- $\frac{dy}{dt}$ and $\frac{dx}{dt}$ represent the growth rates of the two populations over time
- t represents time.
- α, β, γ and δ are parameters describing the interaction of the two species.

Last paragraph should contain plan of the report - one sentence about each of the following sections.

2 Theory

Problem

In this project, we analyse Lotka-Volterra Predator-Prey Model that describe the interaction between predator and prey in given environment with some assumptions. With the help of this model we predict the number of predator and preys at particular time in future from the point where we consider our data. So, we analyse our results with different numerical methods and plotting techniques.

Assumptions made for the Model

To keep the model simple, some assumptions are made that would be unrealistic in most of the predator-prey situations in real world. Specifically, it is assumed that

1. Only two species exist : Predator and Prey.
2. The prey population finds ample food at all times and are born and then die through predation or inherent death.
3. Predators have limitless appetite and they are born and their birth rate is positively affected by the rate of predation, and they die naturally.
4. The food supply of the predator population depends entirely on the size of the prey population.
5. The rate of change of population is proportional to its size.
6. During the process, the environment does not change in favour of one species and the genetic adaptation is sufficiently slow.

Description of Parameters

- α is the growth rate of species x (the prey) in the absence of interaction with species y (the predators)
- β measures the impact of predation on $\dot{x}/x$ (the rate at which predators destroy prey)
- δ denotes the net rate of growth (or immigration) of the predator population in response to the size of the prey population
- γ is the death (or emigration) rate of species y in the absence of interaction with species x

Realistic Model

Very few such "pure" predator-prey interactions have been observed in nature. A simplified interaction is seen in Canadian northern forests where populations of the lynx and the snowshoe hare are intertwined in a life and death struggle. There are good records of pelts of these species trappers brought to the Hudson Bay Company.

Figure 1: **Lynx and Hare: Specialized tightly linked predator and prey relationship.[3]**

Hudson Bay Company

Detailed records on pelts were collected over almost 100 years. Below is data from 1900-1920.[4] [5]

Year	Hares *(1000)	Lynx *(1000)
1900	30	4
1901	47.2	6.1
1902	70.2	9.8
1903	77.4	35.2
1904	36.3	59.4
1905	20.6	41.7
1906	18.1	19
1907	21.4	13
1908	22	8.3
1909	25.4	9.1
1910	27.1	7.4

Year	Hares *(1000)	Lynx *(1000)
1911	40.3	8
1912	57	12.3
1913	76.6	19.5
1914	52.3	45.7
1915	19.5	51.1
1916	11.2	29.7
1917	7.6	15.8
1918	14	9.7
1919	16.2	10.1
1920	24.7	8.6

Figure 2: Data from 1900-1920

- Data from 1900-1920 show distinct rise of hares followed by a rise in lynx.
- Theory has predicted that following a rise of prey, then populations of predator increase.
- Develop Lotka-Volterra model exhibiting this behavior
- This simplified system creates a good opportunity to create a mathematical model.

Mathematical Modelling

Prey Equation

$$\frac{dx}{dt} = \alpha x - \beta xy = f(x, y)$$

The prey are assumed to have an unlimited food supply, and to reproduce exponentially unless subject to predation; this exponential growth is represented in the equation above by the term αx. The rate of predation upon the prey is assumed to be proportional to the rate at which the predators and the prey meet; this is represented above by βxy. If either x or y is zero then there can be no predation.

Predator Equation

$$\frac{dy}{dt} = \delta xy - \gamma y = g(x, y)$$

In this equation, δxy represents the growth of the predator population. (Note the similarity to the predation rate; however, a different constant is used as the rate at which the predator population grows is not necessarily equal to the rate at which it consumes the prey). γy represents the loss rate of the predators due to either natural death or emigration; it leads to an exponential decay in the absence of prey.

Population Equilibrium(x_{eq}, y_{eq})

Population equilibrium occurs in the model when neither of the population levels is changing, i.e. when both of the derivatives are equal to 0. These points are called critical points.

$$\frac{dy}{dt} = \delta xy - \gamma y = 0$$

$$\frac{dx}{dt} = \alpha x - \beta xy = 0$$

The possibilities are,

- $y = 0$, $x = 0$ so $y_{eq} = 0$, $x_{eq} = 0$
- $\alpha - \beta y = 0 \Longrightarrow \mathrm{y}_{eq} = \frac{\alpha}{\beta}$ similarily $x_{eq} = \frac{\gamma}{\delta}$

Implementation of Linearisation for Predator-Prey Problem [6]

$$f(x, y) \approx f(x_{eq}, y_{eq}) + \left(\frac{\partial f}{\partial x}\right)(x - x_{eq}) + \left(\frac{\partial f}{\partial y}\right)(y - y_{eq})$$

$$g(x, y) \approx g(x_{eq}, y_{eq}) + \left(\frac{\partial g}{\partial x}\right)(x - x_{eq}) + \left(\frac{\partial g}{\partial y}\right)(y - y_{eq})$$

A critical point has $f(x_{eq}, y_{eq}) = g(x_{eq}, y_{eq}) = 0$. So, the equation becomes linear combination of x and y. So, the general equation becomes,

$$\begin{bmatrix} (x-x_{eq})' \\ (y-y_{eq})' \end{bmatrix} \approx \begin{bmatrix} \partial f/\partial x & \partial f/\partial y \\ \partial g/\partial x & \partial g/\partial y \end{bmatrix} \begin{bmatrix} x-x_{eq} \\ y-y_{eq} \end{bmatrix} = J \begin{bmatrix} x-x_{eq} \\ y-y_{eq} \end{bmatrix}$$

Implementation of Linearisation for Predator-Prey Problem

The critical points are (0,0) and $(\frac{\gamma}{\delta},\frac{\alpha}{\beta})$

First critical point

At $x_{eq}, y_{eq} = (0,0)$

$$J = \begin{bmatrix} \partial f/\partial x & \partial f/\partial y \\ \partial g/\partial x & \partial g/\partial y \end{bmatrix} = \begin{bmatrix} \alpha - \beta y_{eq} & -\beta x_{eq} \\ \delta y_{eq} & \delta x_{eq} - \gamma \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & -\gamma \end{bmatrix}$$

The eigenvalues of this matrix are $\lambda_1 = \alpha, \quad \lambda_2 = -\gamma$. As the eigen values are oppsoite in sign and always greater than zero, so the fixed point near origin will be saddle point. Near critical point (0,0) the baboon's population $x(t)$ will grow but the population of cheetahs $y(t)$ will decay. It can only happen when there is very less interaction between predator and prey. So, extinction can only happen when prey are artifically eradicated due to which cheetahs will die due to natural reasons(starvation etc).

$$\begin{bmatrix} \frac{dx(t)}{dt} \\ \frac{dy(t)}{dt} \end{bmatrix} = c_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} e^{\alpha t} + c_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} e^{-\gamma t}$$

Second Critical Point

At (x$_{eq}, y_{eq}) = (\gamma_{\overline{\delta}}, \frac{\alpha}{\beta})$

$$J\left(\frac{\gamma}{\delta}, \frac{\alpha}{\beta}\right) = \begin{bmatrix} \partial f/\partial x & \partial f/\partial y \\ \partial g/\partial x & \partial g/\partial y \end{bmatrix} = \begin{bmatrix} \alpha - \beta y_{eq} & -\beta x_{eq} \\ \delta y_{eq} & \delta x_{eq} - \gamma \end{bmatrix} = \begin{bmatrix} 0 & -\frac{\beta\gamma}{\delta} \\ \frac{\alpha\delta}{\beta} & 0 \end{bmatrix}$$

The eigenvalues of this matrix are $\lambda_1 = i\sqrt{\alpha\gamma} = +i\omega, \quad \lambda_2 = -i\sqrt{\alpha\gamma} = -i\omega$. Because the real part is zero, so the stability is neutral and critical points are center. The solution will form closed trajectories surrounding the critical point(1,1). Consequently, the levels of the predator and prey populations cycle, and oscillate around this fixed point.

Extra baboons → Cheetahs increase → Baboons decrease → Cheetahs decrease → Extra Baboons

$$\begin{bmatrix} \frac{dx(t)}{dt} \\ \frac{dy(t)}{dt} \end{bmatrix} = c_1 \begin{bmatrix} \cos(\omega t) \\ A\sin(\omega t) \end{bmatrix} + c_2 \begin{bmatrix} \sin(\omega t) \\ -A\cos(\omega t) \end{bmatrix}$$

where,

$$A = \frac{\delta}{\beta}\sqrt{\frac{\alpha}{\gamma}}$$

To check whether the solution forms a perfect circle we will solve for $\frac{dx}{dy}$ with separation of variables.

$$\frac{dx}{dy} = \frac{dx/dt}{dy/dt} = \frac{f}{g} = \frac{x(\alpha - \beta y)}{y(\delta x - \gamma)}$$

gives,

$$\frac{\delta x - \gamma}{x}dx = \frac{\alpha - \beta y}{y}dy$$

$$\frac{\alpha - \beta y}{y}dy - \frac{\delta x - \gamma}{x}dx = 0$$

Integrating on both sides with respect to y and x gives

$$Q(x, y) = \alpha\ln(y) - \beta y + \gamma\ln(x) - \delta x = C$$

3 Methodology

Numerical Methods: Euler, Rk2, Rk4

Inbuilt Numerical Method: scipy.integrate.odeint [7]

3.1 Algorihtms

Algorithm 1 Slope Function(f)

▷ *input time, initial conditions and parameters α, β, δ, γ stored in tuple* ◁

procedure INPUT($t, x, parameters$)

x, y = X ▷ *An array to store x, y*

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

▷ *growth rate of prey*

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

▷ *growth rate of predator*

return array of $\frac{dx}{dt}$ and $\frac{dy}{dt}$

Algorithm 2 Euler Method

function INPUT(f, *initial conditions*, t_{max}, t_{min}, N, *parameters*)

▷ *Here N is number of steps, t_{max}, t_{min} are initial and final conditions* ◁

Define t ← array of time

▷ *Time array between t_{max} and t_{min} having N data points* ◁

Calculate dt ← step size

▷ *Difference of two consecutive elements of time array* ◁

Define X ← Empty array

▷ *Empty array having two columns and N rows to store values of predator and prey* $X \leftarrow$ ◁

X_0

▷ *Assigning Initial conditions to output array* ◁

for all $i \in \{1, \ldots, N\}$ **do**

$X_{i+1} = X_i + f\,(t_i, x_i, paramteres)\,dt$

return X, t

Algorithm 3 Rk2 Method

function INPUT(f, *initial conditions*, t_{max}, t_{min}, N, *parameters*)

▷ *Here N is number of steps,* t_{max}, t_{min} *are initial and final conditions* ◁

Define t ← array of time

▷ *Time array between* t_{max} *and* t_{min} *having* N *data points* ◁

Calculate dt ← step size

▷ *Difference of two consecutive elements of time array* ◁

Define X ← Empty array

▷ *Empty array having two columns and N rows to store values of predator and prey* $X \leftarrow$ ◁

X_0

▷ *Assigning Initial conditions to output array* ◁

for all $i \in \{1, \ldots, N\}$ **do**

$k_1 = dt(f\,(t_i, X_i, paramteres))$

$k_2 = dt(f\,(t_i + dt, x_i + k_1, paramteres))$

$X_{i+1} = X_i + \frac{(k_1+k_2)}{2}$

return X, t

Algorithm 4 Rk4 Method

function INPUT(f, *initial conditions*, t_{max}, t_{min}, N, *parameters*)

▷ *Here N is number of steps, t_{max}, t_{min} are initial and final conditions* ◁

Define t ← array of time

▷ *Time array between t_{max} and t_{min} having N data points* ◁

Calculate dt ← step size

▷ *Difference of two consecutive elements of time array* ◁

Define X ← Empty array

▷ *Empty array having two columns and N rows to store values of predator and prey* $X \leftarrow$ ◁

X_0

▷ *Assigning Initial conditions to output array* ◁

for all $i \in \{1, \ldots, N\}$ **do**

$$k_1 = f\left(t_i, X_i, paramteres\right)$$
$$k_2 = dt(f\left(t_i + dt/2, x_i + dt/2 \times k_1, paramteres\right))$$
$$k_3 = hf\left(t_i + \frac{dt}{2}, X_i + dt \times k_3, parameters\right)$$
$$k_4 = f\left(t_i + dt, X_i + dt \times k_3, parameters\right)$$
$$X_{i+1} = X_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

return X, t

4 Analysis of Numerical Results

Include all graphs, tables and analysis of the results. This should be a detailed section.

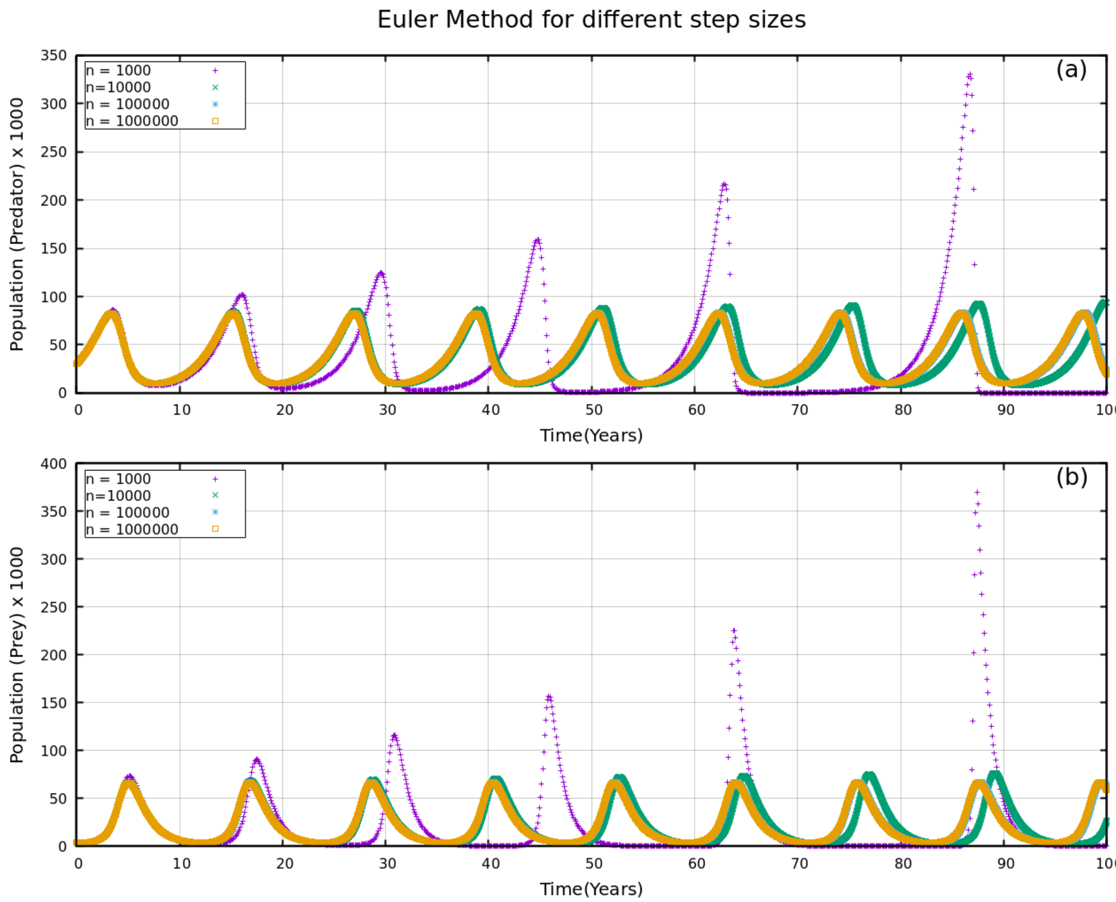

Figure 3: **Euler Method For different step sizes:** + symbol in purple color for 1000 data points, × symbol in green 10000, * symbol in sky blue 100000 and □ symbol in mustard for 1000000 data points respectively.The results for Euler's Method converge for N=100000.
SubFig(a): Population of Predator v/s time(years), **SubFig(b):** Population of Prey v/s time(years)

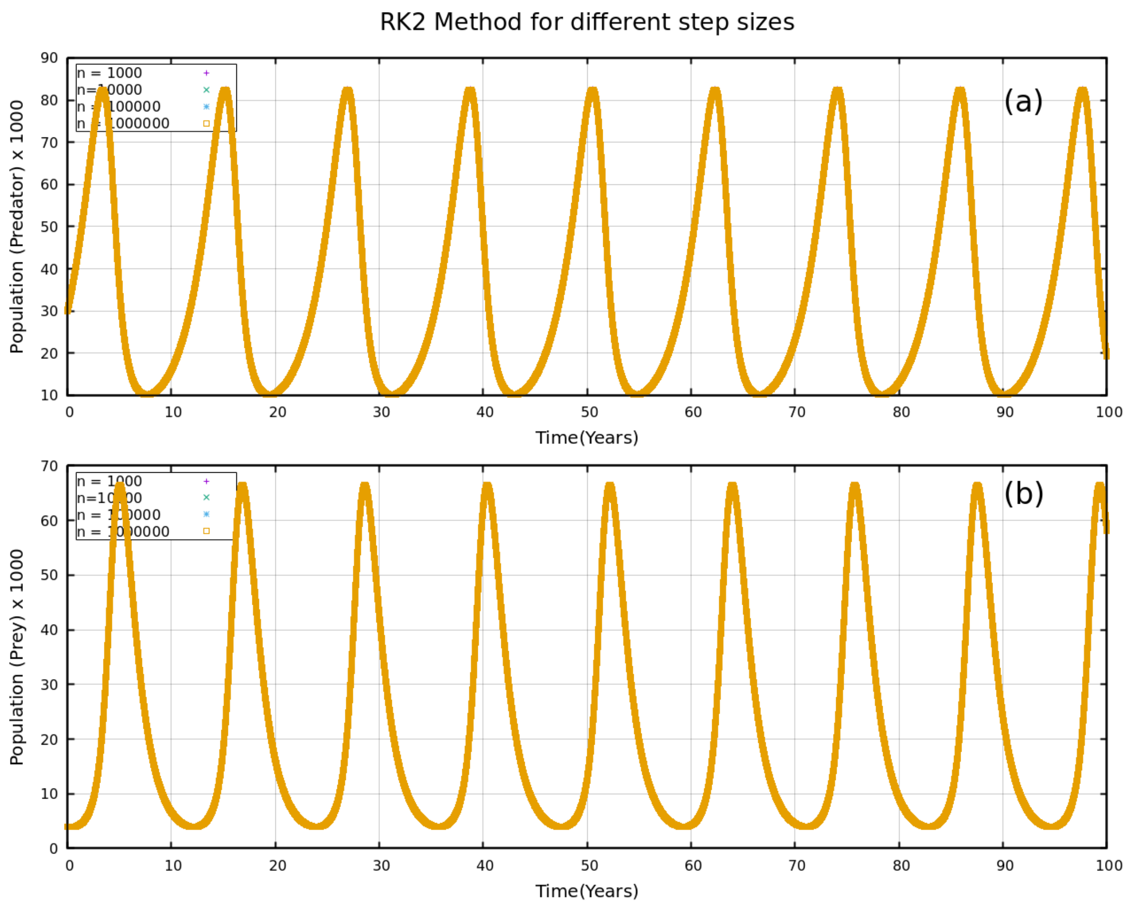

Figure 4: **RK2 Method For different step sizes:** + symbol in purple color for 1000 data points, $\times$ symbol in green 10000, * symbol in sky blue 100000 and $\square$ symbol in mustard for 1000000 data points respectively.The results for RK2 Method converge for N=1000.

SubFig(a): Population of Predator v/s time(years), **SubFig(b):** Population of Prey v/s time(years)

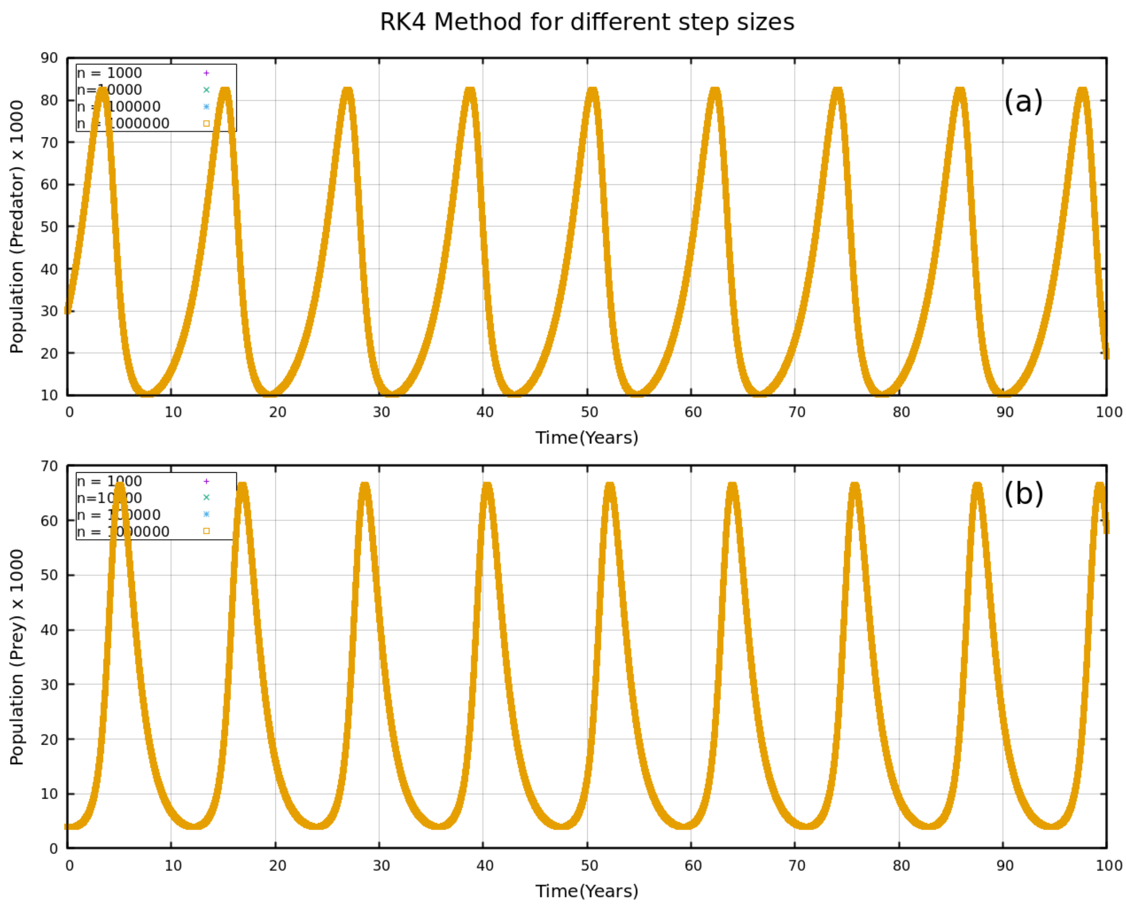

Figure 5: **RK4 Method For different step sizes:** + symbol in purple color for 1000 data points, × symbol in green 10000, * symbol in sky blue 100000 and □ symbol in mustard for 1000000 data points respectively. The results for RK4 Method converge for N=1000.

SubFig(a): Population of Predator v/s time(years), **SubFig(b):** Population of Prey v/s time(years)

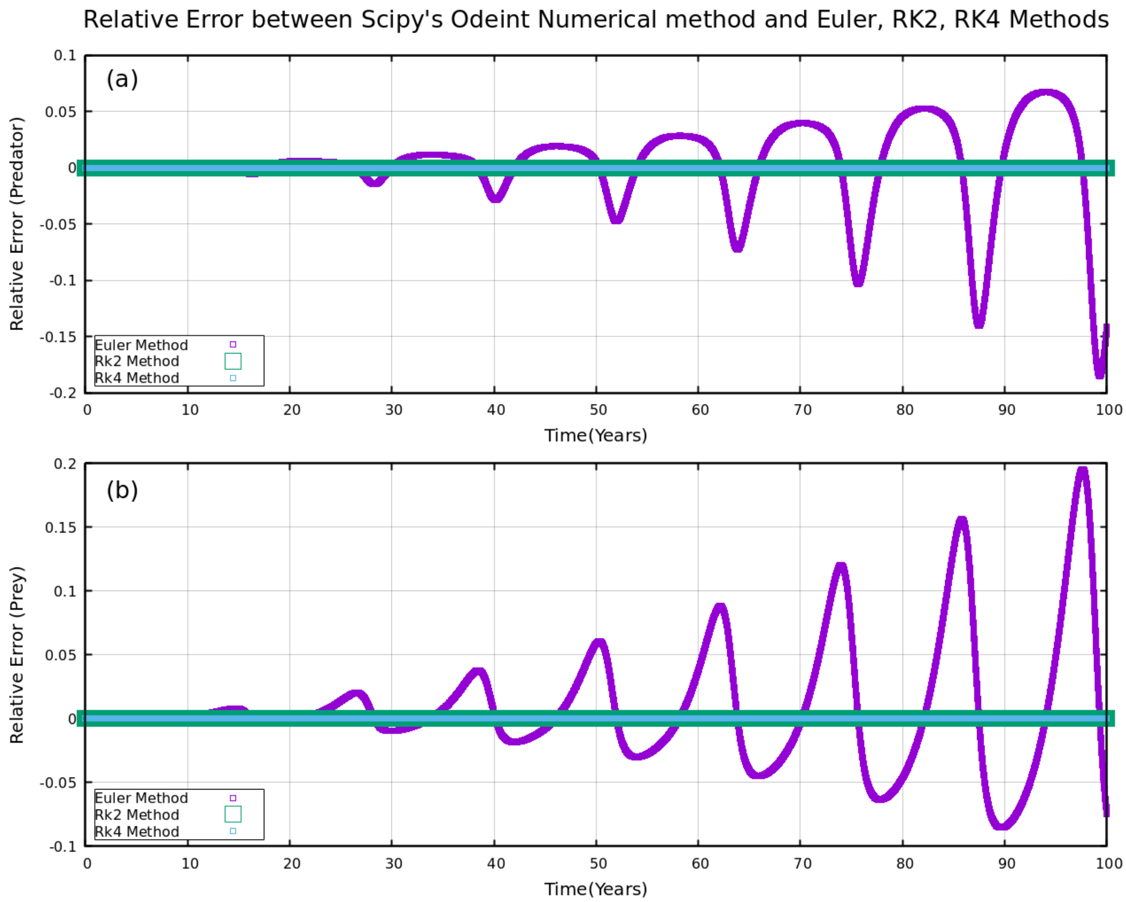

Figure 6: **Euler Method:** 100000 data points; **RK2 Method:** 10000 data points and **RK4 Method:** 1000 data points. The results of Euler's Method won't converge with odeint(inbuilt) method because of the limitations of Euler's Method and function's oscillatory behaviour but it converges for RK2 and RK4 Method

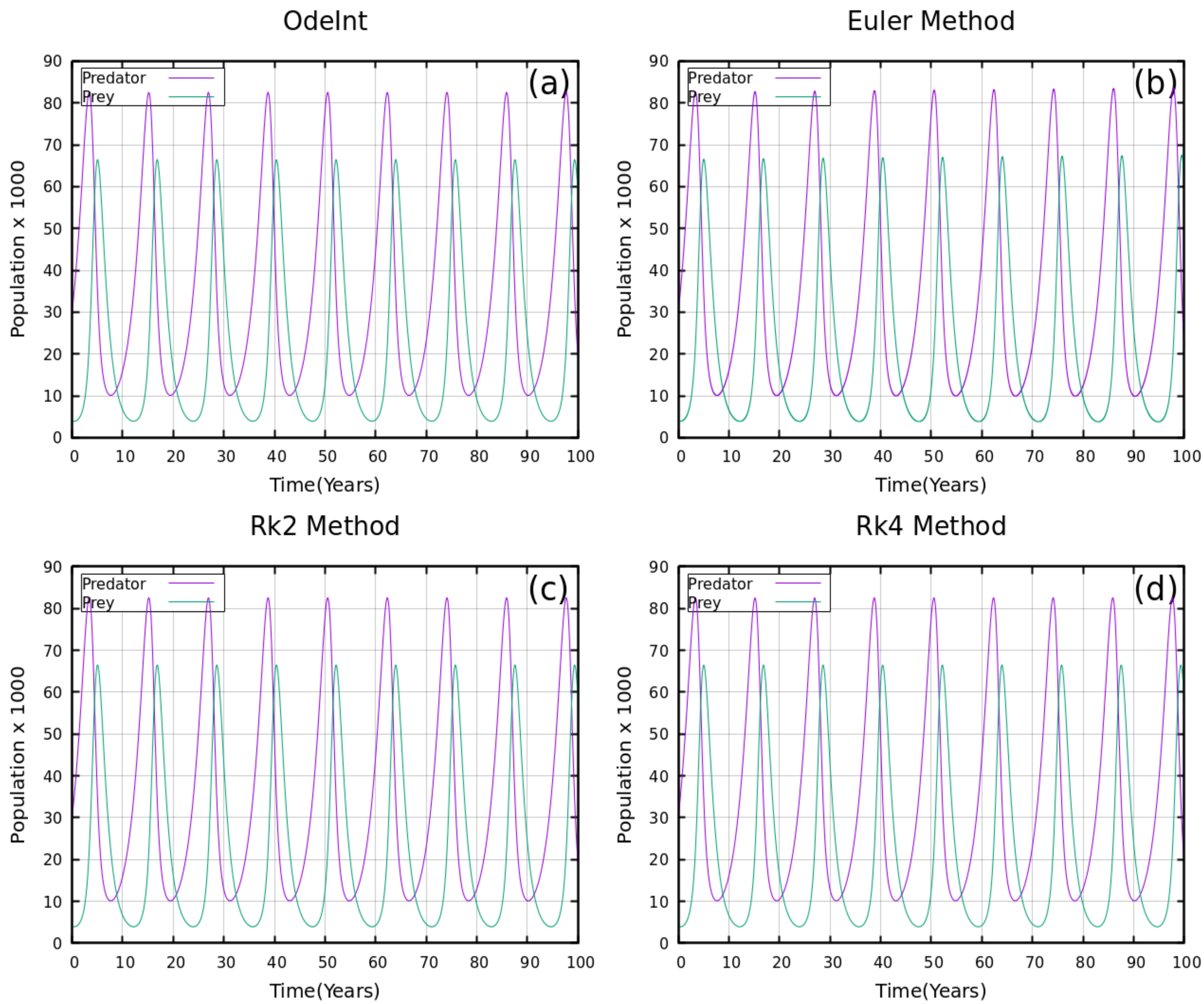

Figure 7: Time v/s Population **Euler Method:** 100000 data points; **RK2 Method:** 10000 data points and **RK4 Method:** 1000 data points

Looking at the results of the simulation we can see that as the population of the prey begins the rise, the number of predators also begins to rise till the point at which predators kill off the prey faster than they can reproduce. Then the numbers begin to fall for the prey which thus causes a lack of food for the predators which numbers also begin to decline. The solution to this simulation is periodic meaning that the cycle will continue ad infinitum with the rise and fall of both populations. This looks very similar to the solution of simple harmonic motion, such as an un-damped spring-mass system except for the addition of a secondary plot.

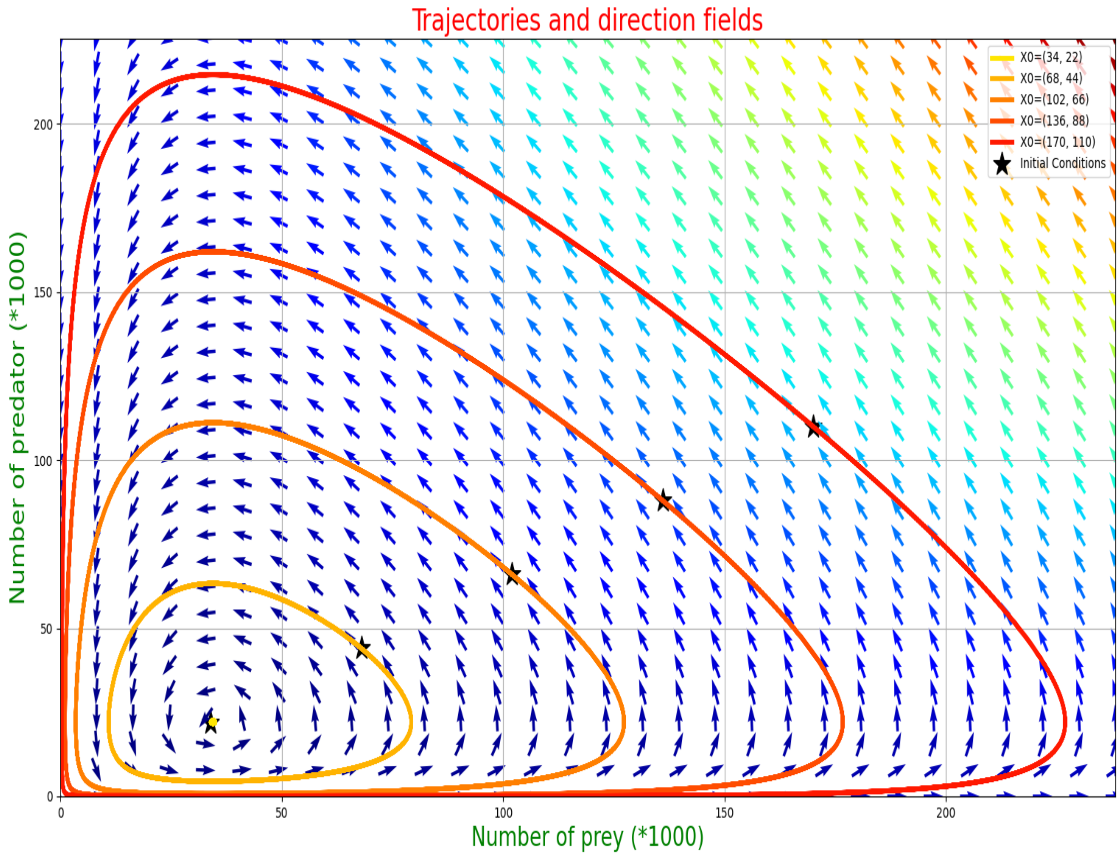

Figure 8: **Trajectories and Direction fields:** Phase-space plot for the predator prey problem for various initial conditions of the prey and predator population. The contours describe solutions of the system determined by their initial data, and since they are closed curves, the solutions are periodic oscillations.

FOR N = 10000

---------------------TABLE SHOWING VARIATION OF POPULATION OF PREY(x) WITH TIME(t)------------------------

	t	x(ODEINT)*1000	x(Euler)*1000	x(RK2)*1000	x(RK4)*1000	[x(odeint)-x(euler)]/x(odeint)	[x(odeint)-x(rk2)]/x(odeint)	[x(odeint)-x(rk4)]/x(odeint)
0	0.000000	30.000000	30.000000	30.000000	30.000000	0.000000	0.000000e+00	0.000000e+00
1	0.010001	30.111531	30.111311	30.111530	30.111530	0.000007	9.869714e-09	5.007144e-09
2	0.020002	30.223501	30.223061	30.223500	30.223501	0.000015	1.915543e-08	9.441297e-09
3	0.030003	30.335912	30.335250	30.335911	30.335911	0.000022	2.700742e-08	1.245321e-08
4	0.040004	30.448764	30.447880	30.448763	30.448764	0.000029	3.434617e-08	1.496383e-08
...	...	...	...	...	...	...	...	...
9995	99.959996	19.894606	93.784316	19.893532	19.894604	-3.714057	5.396033e-05	1.079629e-07
9996	99.969997	19.745512	93.712130	19.744451	19.745510	-3.745996	5.373737e-05	1.098885e-07
9997	99.979998	19.598332	93.633255	19.597284	19.598330	-3.777613	5.351077e-05	1.115662e-07
9998	99.989999	19.453046	93.547626	19.452009	19.453044	-3.808894	5.328068e-05	1.130449e-07
9999	100.000000	19.309631	93.455176	19.308607	19.309629	-3.839822	5.304741e-05	1.145574e-07

[10000 rows x 8 columns]

-------------------TABLE SHOWING VARIATION OF POPULATION OF PREDATOR(y) WITH TIME(t)----------------------

	t	y(ODEINT)*1000	y(Euler)*1000	y(RK2)*1000	y(RK4)*1000	[y(odeint)-y(euler)]/y(odeint)	[y(odeint)-y(rk2)]/y(odeint)	[y(odeint)-y(rk4)]/y(odeint)
0	0.000000	4.000000	4.000000	4.000000	4.000000	0.000000	0.000000e+00	0.000000e+00
1	0.010001	3.995933	3.995880	3.995933	3.995933	0.000013	1.947843e-08	2.829633e-09
2	0.020002	3.991972	3.991865	3.991972	3.991972	0.000027	3.902531e-08	5.636342e-09
3	0.030003	3.988117	3.987957	3.988117	3.988117	0.000040	5.798329e-08	7.763305e-09
4	0.040004	3.984369	3.984155	3.984369	3.984369	0.000054	7.701774e-08	9.876409e-09
...	...	...	...	...	...	...	...	...
9995	99.959996	58.887239	25.851829	58.886276	58.887246	0.560994	1.635062e-05	-1.222495e-07
9996	99.969997	58.689617	26.202845	58.688639	58.689624	0.553535	1.666247e-05	-1.218456e-07
9997	99.979998	58.490674	26.558193	58.489682	58.490681	0.545941	1.697053e-05	-1.213839e-07
9998	99.989999	58.290453	26.917880	58.289447	58.290461	0.538211	1.727483e-05	-1.208725e-07
9999	100.000000	58.088998	27.281911	58.087977	58.089005	0.530343	1.757533e-05	-1.203579e-07

[10000 rows x 8 columns]

GROWTH RATE AT EQUILIBRIUM POINTS FOUND ANALYTICALLY

At (0,0), Growth Rate = [0. -0.]

At (gamma/delta,alpha/beta), Growth Rate = [0. 0.]

Figure 9: Table

5 Summary

- This section contains summary of your results or conclusions
- It should also include our experience and what you have learnt in this project
- Length should be 300- 500 words

References

[1] "Lotka–Volterra equations - Wikipedia."

[2] "Kolmogorov equations - Wikipedia."

[3] "Photograph of Canadian Lynx and Snowshoe Hare Balance Problem Used in... | Download Scientific Diagram."

[4] "Untitled Document."

[5] "Some Mathematical Problems In Biology by Murray Gerstenhaber | Goodreads."

[6] "Differential Equations and Linear Algebra by Gilbert Strang | Goodreads."

[7] "scipy.integrate.odeint — SciPy v1.7.1 Manual."

A Programs

Python Program including numerical methods impementation and data generation for graph plotting

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import integrate
from matplotlib.animation import FuncAnimation
import pylab as p

def eqn(t,X, cons):   #Defining Equations
    x, y = X          #Assigning x,y values
    alpha,beta,delta,gamma=cons       #Assigning values to parameters
    dotx = x * (float(alpha) - float(beta) * y)       #dx/dt = alpha*x-beta*x*y
    doty = y * (-float(gamma)+ float(delta) * x)      #dy/dt = delta*x*y-gamma*y
    return np.array([dotx, doty])                     #Returns dx/dt and dy/dt in array

def eqn1(X, t, alpha, beta, delta, gamma):           #same equations , but taking alpha,beta,delta,gamma parameters as input instead of array
    x, y = X
    dotx = x * (float(alpha) - float(beta) * y)
    doty = y * (-float(gamma)+ float(delta) * x)
    return np.array([dotx, doty])

def Euler(func, X0, tmin,tmax,N, cons):    #Euler Method
    t = np.linspace(tmin,tmax, N)          #Creating array containg N-2 points between tmin and tmax (Basically calculation points)
    dt = t[1] - t[0]                       # calculating step size
    X  = np.zeros([N, len(X0)])            #creating dummy for output array containing x,y values
    X[0] = X0                              #assigning initial values to the output array
    for i in range(N-1):
        X[i+1] = X[i] + func(t[i],X[i],cons) * dt          #Updating values of the dummy array that we created
```

```
    return X,t                                                     #returns array of updated values of X and array t

def RK2(func, X0, tmin, tmax,N,cons):          #Runga kutta 2 method
    t = np.linspace(tmin,tmax, N)   #Creating array containg N-2 points between tmin and tmax (Basically calculation points)
    dt = t[1] - t[0]                      # calculating step size
    X  = np.zeros([N, len(X0)])       #creating dummy for output array containing x,y values
    X[0] = X0                             #assigning initial values to the output array
    for i in range(N-1):
        k1 =dt* func(t[i], X[i], cons)
        k2 = dt*func(t[i] + dt,X[i] +  k1 , cons)
        X[i+1] = X[i] + (k1 +  k2 )/2                                 #Updating values of the dummy array that we created
    return X,t                                              #returns array of updated values of X and array t

def RK4(func, X0, tmin, tmax,N,cons):          #Runga kutta 4 method
    t = np.linspace(tmin,tmax, N)   #Creating array containg N-2 points between tmin and tmax (Basically calculation points)
    dt = t[1] - t[0]                      # calculating step size
    X  = np.zeros([N, len(X0)])       #creating dummy for output array containing x,y values
    X[0] = X0                             #assigning initial values to the output array
    for i in range(N-1):
        k1 = func( t[i],X[i], cons)
        k2 = func(t[i] + dt/2.,X[i] + dt/2. * k1, cons)
        k3 = func( t[i] + dt/2., X[i] + dt/2. * k2,cons)
        k4 = func( t[i] + dt, X[i] + dt    * k3,cons)
        X[i+1] = X[i] + dt / 6. * (k1 + 2. * k2 + 2. * k3 + k4)       #Updating values of the dummy array that we created
    return X,t                                                         #returns array of updated values of X and array t

def V(X,cons):      #Analytic solution of the equation V = delta*x - gamma*ln(x)+beta*y-alpha*ln(y)
    x,y=X
    alpha,beta,delta,gamma=cons
    return delta*x -gamma*np.log(x)+beta*y-alpha*np.log(y)
```

```python

def main():
    t = np.linspace(0.,tmax, Nt)    #Creating array containg N-2 points between tmin and tmax (Basically calculation points)
    X0 = [x0, y0]     #Initial conditions
    cons=(alpha,beta,delta,gamma)
    res = integrate.odeint(eqn1, X0, t, args = cons)   #Calling Scipy's odeint function
    x, y = res.T
    Xe = Euler(eqn, X0,0.,tmax,Nt,cons)   #calling Euler
    x1,y1=Xe[0].T
    Xr2 = RK2(eqn, X0,0.,tmax,Nt,cons)    #calling RK2
    x2,y2=Xr2[0].T
    Xr4 = RK4(eqn, X0,0.,tmax,Nt,cons)     #calling RK4
    x3,y3=Xr4[0].T

    xee=[];yee=[];xer2=[];yer2=[];xer4=[];yer4=[]              #Creating empty lists for appending values of abs(odeint-numrical method(euler/rk2/rk4))/odeint
    for i in range(len(x)):                                   #comparing our results of Euler,RK2,RK4 with Scipy's ODEint for finding error
        xee.append((x[i]-x1[i])/x[i])
        yee.append((y[i]-y1[i])/y[i])
        xer2.append((x[i]-x2[i])/x[i])
        yer2.append((y[i]-y2[i])/y[i])
        xer4.append((x[i]-x3[i])/x[i])
        yer4.append((y[i]-y3[i])/y[i])             #appending values
    V_data = V(X=[x3,y3],cons=cons)
    return (t,x,x1,x2,x3,y,y1,y2,y3,xee,yee,xer2,yer2,xer4,yer4,V_data)

x0 = 30;y0 = 4       # (*1000) INITIAL CONDITIONS
alpha = 0.453;beta = 0.0205;gamma = 0.790;delta = 0.0229;tmax = 100      #Assigning values of parameters
N_arr=[1000,10000,100000,1000000]                #array for different number of steps

for Nt in N_arr:                                 #calculations for different N and storing values in csv files
    k=main()
    t,x,x1,x2,x3,y,y1,y2,y3,xee,yee,xer2,yer2,xer4,yer4,V_data=k
    DataOut111 = np.column_stack((t,x,x1,x2,x3,y,y1,y2,y3,xee,yee,xer2,yer2,xer4,yer4,V_data))
```

```
    if Nt==1000:
        np.savetxt('data_1000.csv', DataOut111,delimiter=',')
    elif Nt==10000:
        np.savetxt('data_10000.csv', DataOut111,delimiter=',')
    elif Nt==100000:
        np.savetxt('data_100000.csv', DataOut111,delimiter=',')
    elif Nt==1000000:
        np.savetxt('data_1000000.csv', DataOut111,delimiter=',')

Nt=10000
G=main()
t,x,x1,x2,x3,y,y1,y2,y3,xee,yee,xer2,yer2,xer4,yer4,V_data=G
print("FOR N = ",Nt)
print("----------------------TABLE SHOWING VARIATION OF POPULATION OF
    PREY(x) WITH TIME(t)-------------------------")
data={"t":t,"x(ODEINT)*1000":x ,"x(Euler)*1000":x1,"x(RK2)*1000":x2,"x(
    RK4)*1000":x3,"[x(odeint)-x(euler)]/x(odeint)":xee,"[x(odeint)-x(rk2
    )]/x(odeint)":xer2,"[x(odeint)-x(rk4)]/x(odeint)":xer4}
print(pd.DataFrame(data))
print("--------------------TABLE SHOWING VARIATION OF POPULATION OF
    PREDATOR(y) WITH TIME(t)-----------------------")
data={"t":t,"y(ODEINT)*1000":y ,"y(Euler)*1000":y1,"y(RK2)*1000":y2,"y(
    RK4)*1000":y3,"[y(odeint)-y(euler)]/y(odeint)":yee,"[y(odeint)-y(rk2
    )]/y(odeint)":yer2,"[y(odeint)-y(rk4)]/y(odeint)":yer4}
print(pd.DataFrame(data))

X_e1 = np.array([     0 ,   0])                   #Equilibrium condition
     1
X_e2 = np.array([ gamma/delta, alpha/beta])       #Equilibrium condition
     2
print("GROWTH RATE AT EQUILIBRIUM POINTS FOUND ANALYTICALLY")
print("At (0,0), Growth Rate = ",eqn1(X_e1,t,alpha,beta,delta,gamma))
print("At (gamma/delta,alpha/beta), Growth Rate = ",eqn1(X_e2,t,alpha,
    beta,delta,gamma))

#Plotting Trajectories and direction fields
values  = np.linspace(1,5, 5)
X_f1 = np.array([ int(gamma/delta), int(alpha/beta)])
vcolors = plt.cm.autumn_r(np.linspace(0.1, 0.9, len(values)))  # colors
     for each trajectory
h1=[];h2=[]
for v, col in zip(values, vcolors):
```

```
    X0 = v * X_f1                                  # starting point
    X = integrate.odeint(eqn1, X0, t, args = (alpha, beta, delta, gamma))
    plt.plot( X[:,0], X[:,1], lw=3.5, color=col, label='X0=(%.f, %.f)' % ( X0[0], X0[1]) )
    plt.legend()
    h1.append(X0[0])
    h2.append(X0[1])

ymax = plt.ylim(ymin=0)[1]                        # get axis limits
xmax = plt.xlim(xmin=0)[1]
nb_points   = 30

x = np.linspace(0, xmax, nb_points)
y = np.linspace(0, ymax, nb_points)

X1 , Y1  = p.meshgrid(x, y)    # create a grid
DX1, DY1 = eqn1([X1,Y1], t, alpha, beta, delta, gamma)              # compute growth rate on the grid
M = (np.hypot(DX1, DY1))                               # Norm of the growth rate
M[ M == 0] = 1.                                  # Avoid zero division errors
DX1 /= M                                               # Normalize each arrows
DY1 /= M

plt.title('Trajectories and direction fields',fontsize=22,c="r")
plt.scatter(h1,h2,color="black",marker="*",s=300,label="Initial Conditions")
Q = plt.quiver(X1, Y1, DX1, DY1, M, pivot='mid', cmap=p.cm.jet)
plt.xlabel('Number of prey (*1000)',fontsize=19,c="green")
plt.ylabel('Number of predator (*1000)',fontsize=19,c="green")
plt.legend()
plt.grid()
plt.xlim(0, xmax)
plt.ylim(0, ymax)
plt.show()

#ANIMATION
fig = plt.figure()
ax1 = plt.subplot(2, 1, 1)
ax2 = plt.subplot(2, 1, 2)
```

```
data_skip = 200

def init_func():
    ax1.clear()
    ax2.clear()
    ax1.set_xlabel('Time ')
    ax1.set_ylabel('[Prey(red) & Predator(green)]*1000')
    ax2.set_xlabel('Prey (*1000)')
    ax2.set_ylabel('Predator (*1000)')
    ax1.set_xlim((t[0], t[-1]))
    ax1.set_ylim((0, 85))
    ax1.grid()
    ax2.set_xlim((5,85))
    ax2.set_ylim((0,70))
    ax2.grid()

def update_plot(i):
    ax1.plot(t[i:i+data_skip], x3[i:i+data_skip], color='k')
    ax1.scatter(t[i], x3[i], marker='o', color='r',label="prey")
    ax1.plot(t[i:i+data_skip], y3[i:i+data_skip], color='k')
    ax1.scatter(t[i], y3[i], marker='o', color='green',label="predator")
    ax2.plot(x3[i:i+data_skip], y3[i:i+data_skip], color='k')
    ax2.scatter(x3[i], y3[i], marker='o', color='magenta')

anim = FuncAnimation(fig,
                     update_plot,
                     frames=np.arange(0, len(t), data_skip),
                     init_func=init_func,
                     interval=1)

anim.save('animation_LotVol.gif', dpi=150, fps=10, writer='ffmpeg')
```

Listing 1: Python example

Programs for Plotting

```
#Euler for diff h
set term pngcairo enhanced size 1280,1024
set datafile separator ","
set output 'euler_diff_h.png'
set border linewidth 2

set multiplot layout 2,1
set key top left Left box title
set xlabel 'Time(Years)'
set ylabel 'Population'
set title "Euler Method(Predator) for different step sizes"  font "enhanced [,20]"
plot "Euler_data.csv" u 1:2 title "n = 1000" , "Euler_data.csv" u 4:5 title "n=10000" , "Euler_data.csv" u 7:8 title "n = 100000", "Euler_data.csv" u 10:11 title "n = 1000000"
set title "Euler Method(Prey) for different step sizes"  font "enhanced [,20]"
plot "Euler_data.csv" u 1:3 title "n = 1000" , "Euler_data.csv" u 4:6 title "n=10000" , "Euler_data.csv" u 7:9 title "n = 100000", "Euler_data.csv" u 10:12 title "n = 1000000"
unset multiplot

#Plotting time v/s population
set term pngcairo enhanced size 1280,1024


set datafile separator ","
set output 'time_vs_population.png'
set border linewidth 2

set style line 1 linecolor rgb 'blue' linetype 1 linewidth 8
set style line 1 linecolor rgb 'green' linetype 1 linewidth 8
set multiplot layout 2,2 #title "Time v/s Population" font "enhanced [,30]"
set key top left Left box title
set xlabel 'Time(Years)'
set ylabel 'Population'
set title "OdeInt"  font "enhanced [,20]"
plot "test.csv" u 1:2 title "Predator" w l , "test.csv" u 1:6 title "Prey"  w l
```

```
set title "Euler Method"  font "enhanced [,20]"
plot "test.csv" u 1:3 title "Predator"  w l, "test.csv" u 1:7  title "Prey" w l
set title "Rk2 Method"  font "enhanced [,20]"
plot "test.csv" u 1:4  title "Predator" w l, "test.csv" u 1:8  title "Prey" w l
set title "Rk4 Method"  font "enhanced [,20]"
plot "test.csv" u 1:5  title "Predator" w l, "test.csv" u 1:9  title "Prey" w l
set title "rk4"

unset multiplot



#Error Plot for predator and prey
set term pngcairo enhanced size 1280,1024
set datafile separator ","
set output 'error_plot.png'
set border linewidth 2

set multiplot layout 2,1 #title "Time v/s Population" font "enhanced [,30]"
set key bottom left Left box title
set xlabel 'Time(Years)'
set ylabel 'Absolute Error'
set title "Error Plot(Predator)"  font "enhanced [,20]"
plot "test.csv" u 1:10 pt 4 title "Euler Method" , "test.csv" u 1:13 pt 4 ps 3 title "Rk2 Method" , "test.csv" u 1:15 pt 4 title "Rk4 Method"
set title "Error Plot(Prey)"  font "enhanced [,20]"
plot "test.csv" u 1:11 pt 4 title "Euler Method"  , "test.csv" u 1:13 pt 4 ps 3 title "Rk2 Method" , "test.csv" u 1:15 pt 4 title "Rk4 Method"
unset multiplot
```

Listing 2: Plotting example

B Contribution of team mates

Contribution of "*name of partner A*"

- In Formulation of the problem:
- In Programming:
- In Plotting Graphs:
- In Report Writing:

Contribution of "*name of partner B*"

- In Formulation of the problem:
- In Programming:
- In Plotting Graphs:
- In Report Writing:

Contribution of "*name of partner C*"

- In Formulation of the problem:
- In Programming:
- In Plotting Graphs:
- In Report Writing: