ISSN: 0974-5823 Vol. 7 No. 4 April, 2022

International Journal of Mechanical Engineering

NUMERICAL SOLUTION OF SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS USING RUNGE- KUTTA METHOD

V. Gomathi

Assistant Professor, PG and Research Department of Mathematics,

Theivanai Ammal College for Women (Autonomous), Villupuram-605602, Tamil Nadu, India

G. Rabiyabe

M.Sc Mathematics, PG and Research Department of Mathematics,

Theivanai Ammal College for Women (Autonomous), Villupuram-605602, Tamil Nadu, India

ABSTRACT:

This paper approaches to solve the system of ordinary differential equations specifically, Lotka Volterra equation also known as Predator- Prey equation using classical fourth order Runge-Kutta method (RK4) . Matlab and python are used as an important tool to obtain numerical solutions for different initial conditions (time t) to compare the result both numerically and also has shown the growth proportion graphically.

KEY WORDS:

Initial Value problem, Lotka - Volterra equation, Runge – Kutta Method, Matlab and Python.

INTRODUCTION:

Numerical analysis is a branch of Mathematics and computer science that aims to create, analyze, and implement systematic techniques for solving mathematical problems numerically [2]. Such problems may fall into categories: algebraic equations, transcendental equations, ode's and pde's [3,4]. Amongst, all present methods Runge –Kutta methods are standard method in the field of numerical ODEs [5]. In this paper, we have discussed the Lotka volterra equation also known as the predator-prey equations, are a pair of first order system of nonlinear differential equations frequently used to describe the dynamics of biological systems in which two species interact, one as a predator and the other as prey. The populations change through time according to the pair of equations. The physical meaning of the equation is it makes a number of assumptions, not necessarily realizable in nature, about the environment and evolution of the predator and prey populations [6]. During the process, the environment does not change in favor of one species, and genetic adaptation is in consequential. Predators have limitless appetite. The prey population finds ample food at all times. The food supply of the predator population depends entirely on the size of the prey population. The rate of change of population is proportional to its size. In this case the solution of the differential equations is deterministic and continuous. This, in turn, implies that the generations of both the predator and prey are continually overlapping [7].

The general objective of this paper is to develop an efficient and simple Matlab and python coding to solve and compare predator –prey model using RK4 method. It has been checked for two different initial conditions of (time t) and obtained solutions both numerically and graphically.

PRELIMINARIES:

Initial value problem:

An Initial value problem (IVP) is an ordinary differential equation together with an initial condition which specifies the value of the unknown function at a given point in the domain.

Definition: An initial value problem is a differential equation

$\mathbf{y'(t) = f(t, y(t))}$ **with** $\mathbf{f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n}$ where $\mathbf{\Omega}$ is an open set of $\mathbb{R} \times \mathbb{R}^n$, together with a point in t domain of $\mathbf{f\ (t_o, y_o) \in \Omega}$, called the Initial condition.

A solution to an initial value problem is a function y that is a solution to the differential equation and satisfies

$y(t_o) = y_o$

The initial value problem is given in the form of

$y' = \frac{dy}{dx} = f(x), y(x_o) = y_o , a \leq x \leq b$, for x is an idependent variable and $x_o = a$.

A general system of *m* linear equations with *n* unknowns can be written as

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m,$$

where $x_1, x_2, \ldots, x_n$ are the unknowns $a_{11}, a_{12}, \ldots, a_{mn}$ are the coefficients of the system and $b_1, b_2, \ldots, b_m$ are the constant terms.

Lotka volterra equation:

The Lotka-Volterra equations, also known as the predator prey equations, are a pair of first order nonlinear differential equations, frequently used to describe the dynamics of biological systems in which two species interact ,one as a predator and the other as prey. The populations change through time according to the pair of equations:

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

- X is the number of prey (for example: rabbits);
- Y is the number of some predator (for example: foxes);
- $\frac{dy}{dt}$ and $\frac{dx}{dt}$ represent the instantaneous growth rates of the two populations;
- t represents time;
- α, β, γ, δ are positive real parameters describing the interaction of the two species.

The model is simplified with the following assumptions:

(1) only two species exist: fox and rabbit;

(2) rabbits are born and then die through predation or inherent death;

(3) foxes are born and their birth rate is positively affected by the rate of predation, and they die naturally.

The characteristic of this model is that the population change of the predator and the prey are explained in terms of each other. The size of the fox population has a negative effect on the rabbit population, and the size of the rabbit population has a positive effect on the fox population. Let us first build a model for the system with a textual approach[9]. The Lotka–Volterra model can be described using a pair of first-order, nonlinear, differential equations as follows:

$\frac{dR}{dt}$ = rabbit birth rate× R− rabbit death rate × R× F

$\frac{dF}{dt}$ = fox birth rate × R × F − fox death rate × F

Where *R* and *F* are the population of rabbit and fox, respectively.

Runge kutta method:

In Numerical analysis ,the Runge Kutta methods are a family of Implicit and Explicit iterative methods, which include the Euler method, used in temporal discretization for the approximate solutions of Simultaneous non linear equations [2].These methods were developed around 1900 by the German mathematicians Carl Runge and Wilhelm Kutta. The most widely known member of the Runge kutta family is generally referred to as "RK4", the classical Runge-Kutta method" or simply as "the Runge Kutta method".

The family of explicit Runge –Kutta methods is a generalization of the RK4 method mentioned below:

$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$ **where**

$k_1 = f(t_n , y_n)$,

$k_2 = f(t_n + c_2 h, y_n + h (a_{12} k_1))$,

$k_3 = f(t_n + c_3 h, y_n + h (a_{31} k_1 + a_{32} k_2))$,

$$\vdots$$

$$k_s = f(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})).$$

To specify a particular method one needs to provide the integer s (the no. of. Stages), and the coefficients aij (for $1 \leq j < i \leq S$) , b_i (for i=1,2,3,….,s) and c_i (for i = 2,3,….,s).

MATLAB:

MATLAB (an abbreviation of "MATrix LABoratory") was invented by mathematician and computer programmer **Cleve Mole**r.[11]. It is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. It allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. The idea for MATLAB was based on his 1960s PhD thesis.

PYTHON:

PYTHON was invented by **Guido van Rossum** . He began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0 [21]. Python consistently ranks as one of the most popular programming languages. It is a high-level, interpreted, general-purpose programming language also constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects. It is meant to be an easily readable language and formatting is visually uncluttered.

RESULTS AND DISCUSSIONS:

We now obtain numerical and graphical solution of predator-prey model. The classical RK4 method is used to find solution in matlab and python as well. It is checked for two different values of time t in both programming language and obtained same results graphically.

Numerical Illustration: Consider there are two species of animals, a baboon (prey) and a cheetah (predator). If the initial conditions are 10 baboons and 10 cheetahs, one can plot the progression of the two species over time; given the parameters that the growth and death rates of baboon are 1.1 and 0.4 while that of cheetahs are 0.1 and 0.4 respectively. The choice of time interval is arbitrary.

One may also plot solutions parametrically as orbits in phase space, without representing time, but with one axis representing the number of prey and the other axis representing the number of predators for all times.

This corresponds to eliminating time from the two differential equations above to produce a single differential equation

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

relating the variables *x* and *y*. The solutions of this equation are closed curves. It is amenable to separation of variables: integrating

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

yields the implicit relationship

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

where *V* is a constant quantity depending on the initial conditions and conserved on each curve.

To find the solution , we need to calculate the k values in order to find the yn+1 value of Rk4 method.

Let the initial value problem is given in the form of

$y = \frac{dy}{dt} = f(t, y),\ y(t_o) = y_o$, here y is an unknown function of time t , which we would like to approximate; $\frac{dy}{dt}$, the rate at which y changes ,is a function of t and y itself. At the initial time t_o the corresponding y value is y_o. The function f and the initial conditions t_o, y_o are given.

Now pick a step-size h > 0 and define

$$y_{n+1} = y_n + \frac{1}{6} h (k_1 + 2k_2 + 2k_3 + k_4),$$

$t_{n+1} = t_n + h$

for n = 0,1,2,3,…., using

$k_1 = f(t_n, y_n)$,

$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right)$,

$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right)$,

$k_s = f(t_n + h, y_n + h k_3)$.

here y_{n+1} is the RK4 approximation of $y(t_{n+1})$, and the next value (y_{n+1}) is determined by the present value (y_n) plus the weighted average of four increments, where each increment is the product of the size of the interval, h, and an estimated slope specified by function f on the right-hand side of the differential equation. In averaging the four slopes, greater weight is given to the slopes at the midpoint.

- K_1 is the slope at the beginning of the interval, using y;
- k_2 is the slope at the midpoint of the interval, using y and k_1.
- k_3 is again the slope at the midpoint, but now using y and k_2.
- k_4 is the slope at the end of the interval, using y and k_3.

Proposed Algorithm using Matlab:

```
xdot = LotkaVolterraModel(x, params)
 alpha = params.alpha;
beta = params.beta;
gamma = params.gamma;
delta = params.delta;
 xdot = [alpha*x(1) - beta*x(1)*x(2)
       delta*x(1)*x(2) - gamma*x(2)];
     [x, t] = RungeKutta4(f, x0, t0, tf, dt)
 t = t0:dt:tf;
nt = numel(t);
nx = numel(x0);
x = nan(nx, nt);
x(:,1) = x0;
for k = 1:nt-1
      k1 = dt*f(t(k), x(:,k));
      k2 = dt*f(t(k) + dt/2, x(:,k) + k1/2);
      k3 = dt*f(t(k) + dt/2, x(:,k) + k2/2);
      k4 = dt*f(t(k) + dt, x(:,k) + k3);
    dx = (k1 + 2*k2 + 2*k3 + k4)/6;
      x(:,k+1) = x(:,k) + dx;


 %% Define problem
 params.alpha = 1.1;
params.beta = 0.4;
params.gamma = 0.4;
params.delta = 0.1;
 f = @(t,x) LotkaVolterraModel(x, params);
```

```
x0 = [20; 5];

 %% solve Diff.Eq.
 t0 = 0;
tf = 100;
dt = 0.01;
 [x, t] = RungeKutta4(f, x0, t0, tf, dt);

%% Plot Results
Figure;
subplot(1, 2, 1);
plot(t, x);
legend('preys', 'Predators');
xlabel('Time (t)');
grid on;
subplot(1, 2, 2);
plot( x(1,:), ( x(2,:));
xlabel('preys');
ylabel('Predators');
grid on;
```

Output:

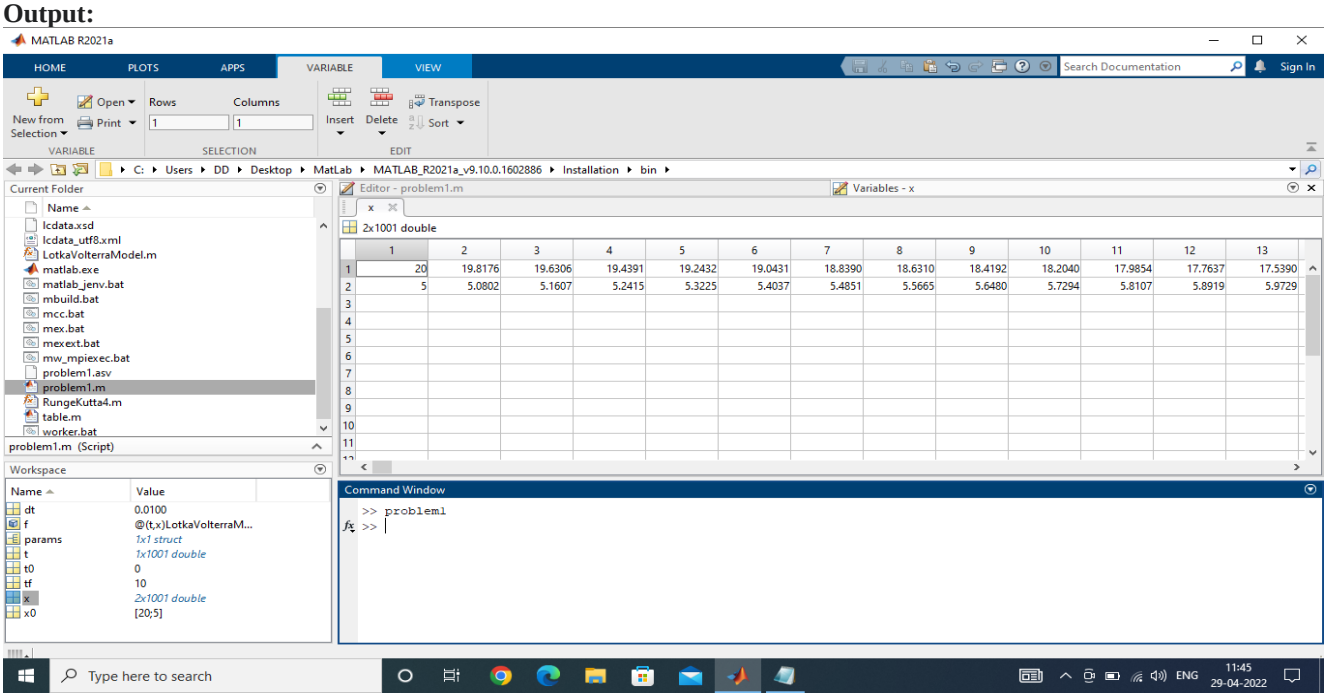

Fig 1.1 the values of t with step size h = 0.01

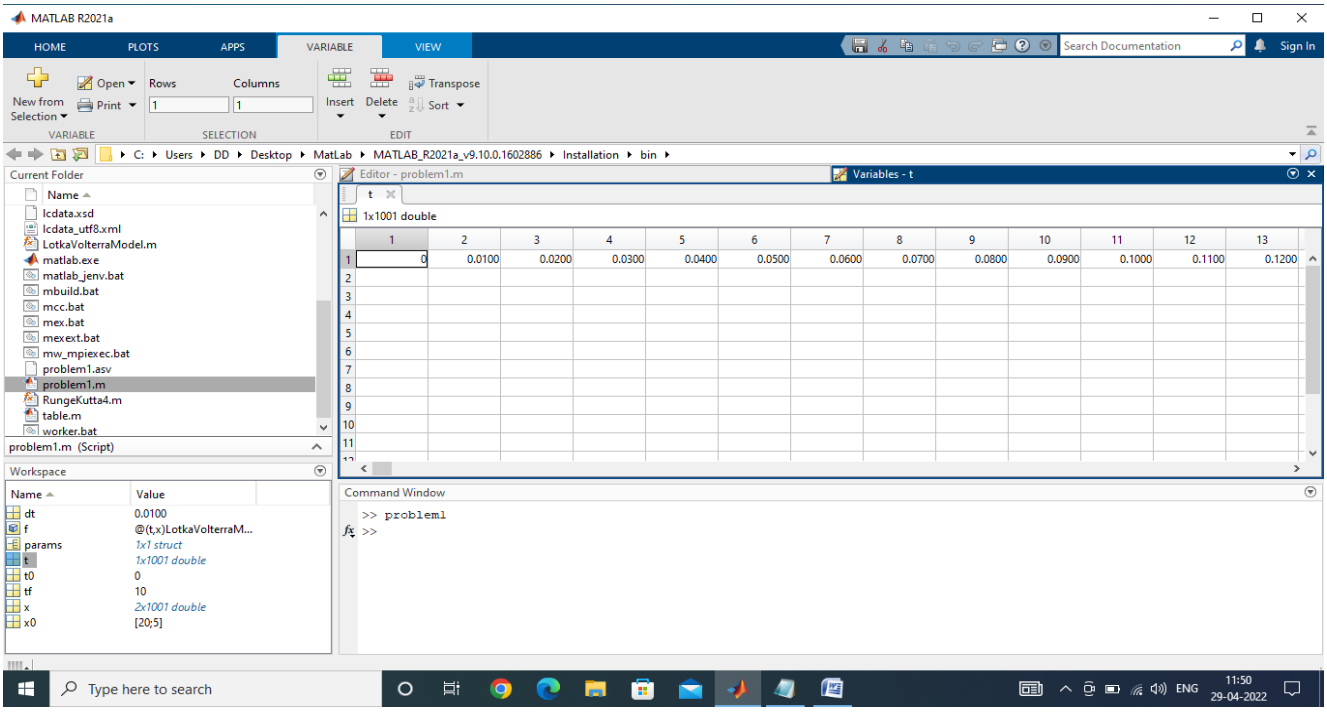

Fig 1.2 the values of x with respect to t values

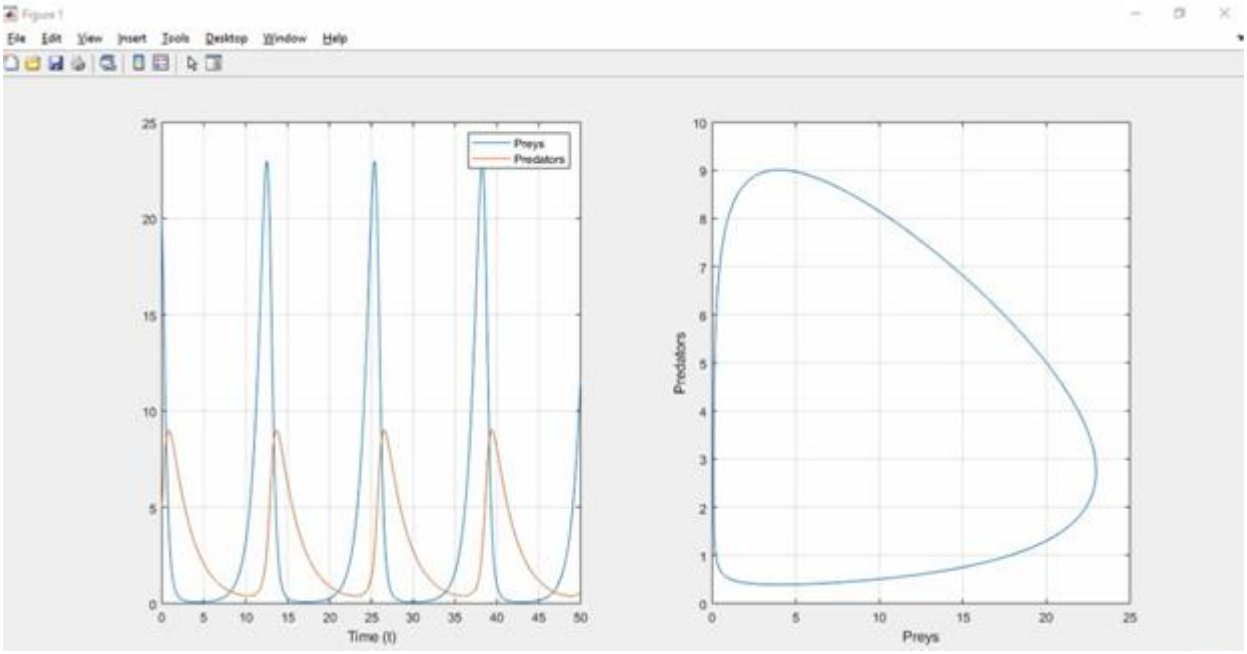

Fig 2.1 the graphical representation of predator-prey equation with time t = 50

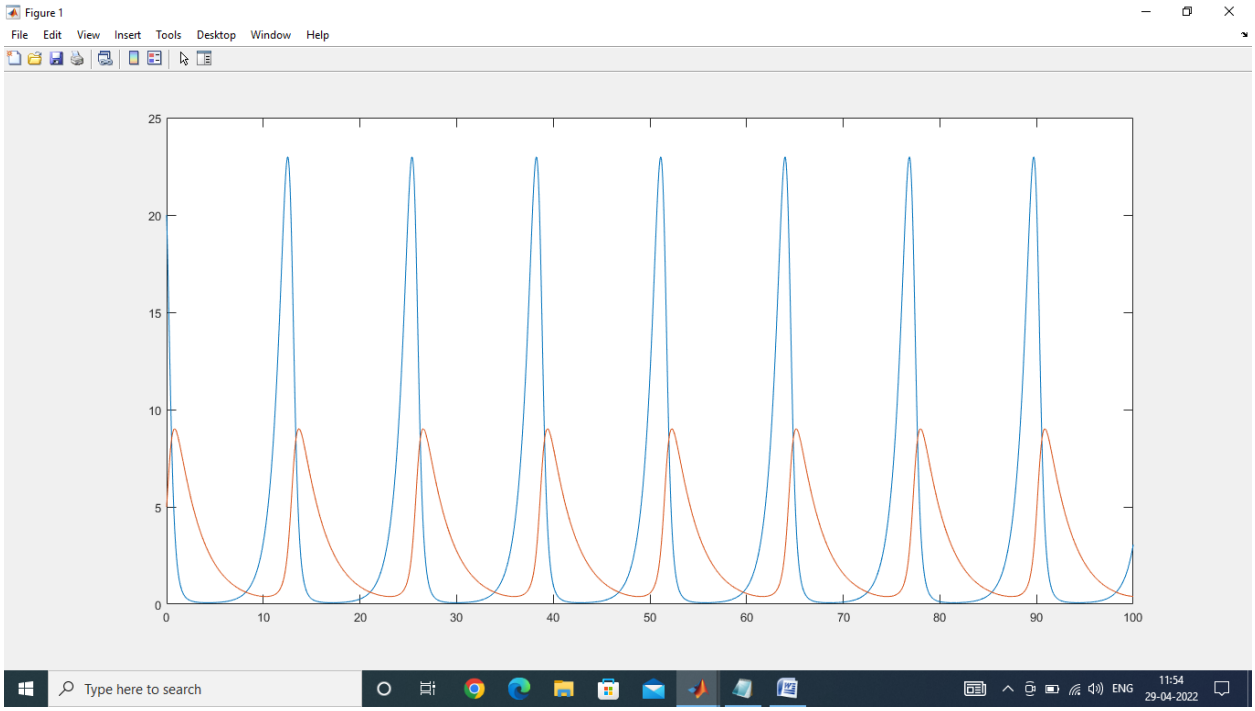

Fig 2.2 the graphical representation of predator-prey equation with time t = 100

.

Proposed Algorithm using Python:

```
import numpy as np
import matplotlib.pyplot as plt
# define the General Model
def LotkaVolterraModel(x, params):
  alpha = params["alpha"]
    beta = params["beta"]
gama = params["gama"]
    delta = params ["delta"]
xdot = np.array([alptha*x[0] - beta*x[0]*x[1], delta*x[0]*x[1] - gamma*x[1]])
  return xdot

# RK4 Method
    def RungeKutta4(f, x0, t0, tf, dt):
  t = np.arrange(t0, tf, dt)
nt = t.size
nx = x0.size
        x = np.zeros((nx,nt))
  x[:,0] = x0

for k in range(nt - 1):
            k1 = dt*f(t[k], x[:, k])
            k2 = dt*f(t[k] + dt/2, x[:,k] + k1/2)
            k3 = dt*f(t[k] + dt/2, x[:,k] + k2/2)
            k4 = dt*f(t[k] + dt, x[:,k] + k3)
```

```
        dx = (k1 + 2*k2 + 2*k3 + k4)/6
         x[:,k+1] = x[:,k] + dx
  return x,t

# Define Problem
  params = {"alpha": 1.1, "beta": 0.4, "gama": 0.4, "delta": 0.1}
    f = lambda t,x : LotkaVolterraModel(x, params)
    x0 = np.array([20,5])

# Solve the Diff. Eq.
    t0 = 0
    tf = 100
    dt = 0.01
  x, t = RungeKutta4(f, x0, t0, tf, dt)

  # Plot Results
plt.subplot(1, 2, 1)
plt.plot(t, x[0,:], "r", label="preys")
plt.plot(t, x[1,:], "b", label="predators")
plt.xlable("time (t)")
plt.grid()
plt.legend

plt.subplot(1, 2, 2)
plt.plot(x[0,:], x[1,:])
plt.xlabel("preys")
plt.ylabel("predators")
plt.grid()
plt.show()
```

Output:

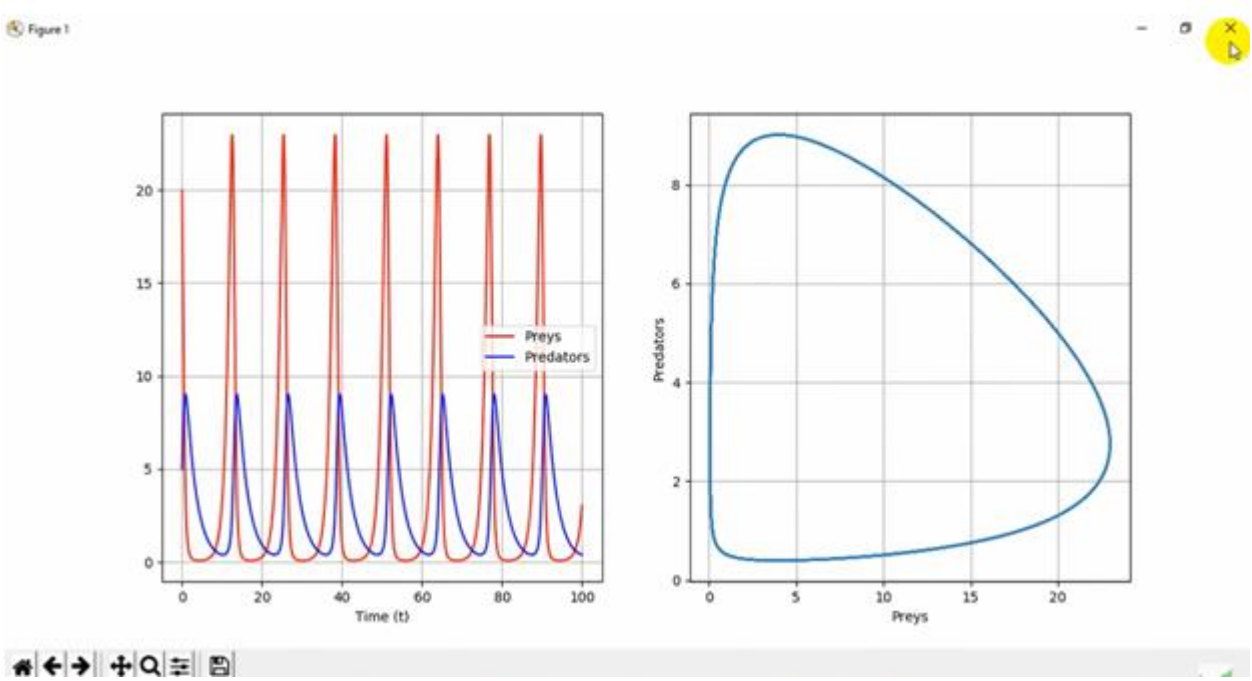

Fig 3.1 the graphical representation of predator-prey equation with time t = 100

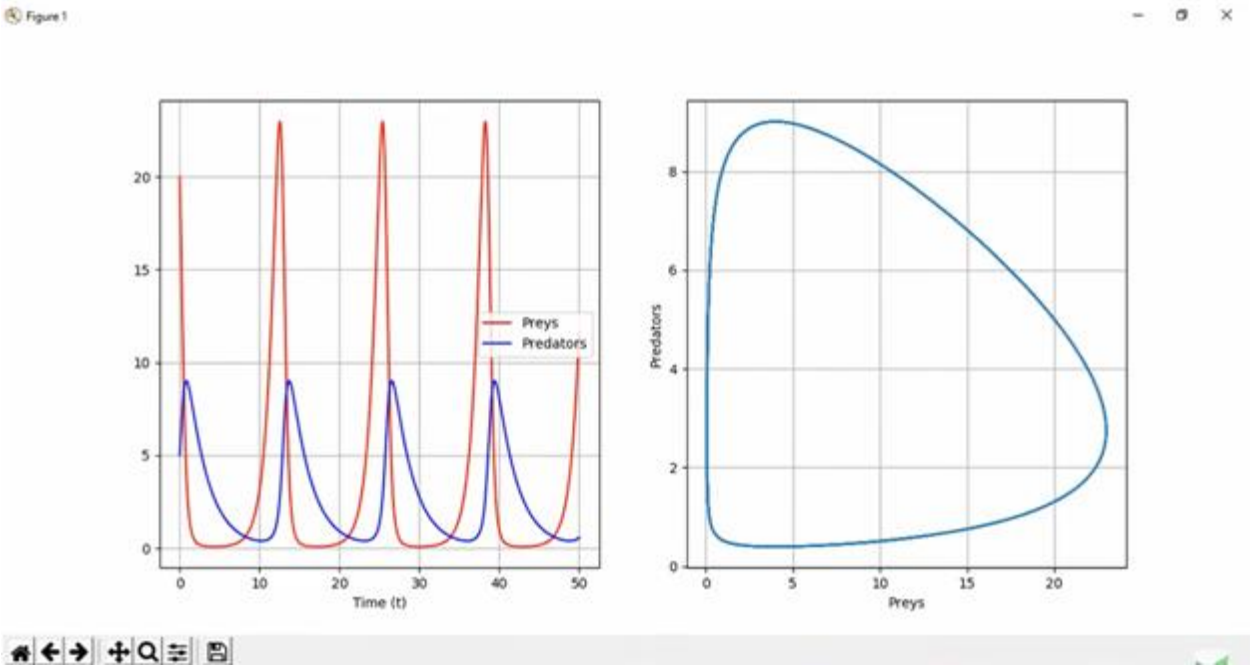

Fig 3.2 the graphical representation of predator-prey equation with time t = 50

CONCLUSION:

This paper concludes that solving numerical problems analytically using iterative methods such as Euler, Taylor's series, Implicit and Explicit Runge- Kutta methods, Adams-BashForth , Predictor-Corrector , Milne –Simpsons are not accurate globally and time consuming as well. Solving such problems computationally using Matlab, Python, psilab, Mathematicia ,etc., helps to obtain more accurate and quick results. We have tried to compare an ODEs" Lotka Volterra Predator-Prey model problem" using matlab and python both numerically and graphically, this study makes clear that output Fig 2.2in matlab and Fig 3.1in python indicates good agreement in graphical representation for time t = 100 and also Fig 2.1 in matlab ,Fig 3.2 in python indicates good agreement in graphraphical representation for time t = 50. Thus, both the programming language yields accurate and likely results, quickly while compared to solving analytically. Further, this predator –Prey model can be analysed with Pieciewise Analytic (PAM) Method.

REFERENCES:

[1] M K Jain, S R K Iyengar, R K Jain, (2019)" Numerical Methods for scientific and Engineering Computation(7th Ed),New Delhi, New age international publishers,ISBN:978-93-87477-25-4.

[2] Kedir Aliyi Koroche, (2021)" Numerical solution of first order ODE by using Runge kutta method", Ethiopia, ISSN:2575-5803.

[3] Sastry, S. S. (2006). "Introductory method of numerical analysis", Fourth-edition.

[4] Walter Gautschi (2012) "Numerical Analysis", Second Edition.

[5] Butcher, J. C., & Goodwin, N. (2008)." Numerical methods for ordinary differential equations (Vol. 2). New York: Wiley".

[6] "Predator-Prey Dynamics". www.tiem.utk.edu. Retrieved 2018-01-09.

[7] Cooke, D.; Hiorns, R. W.; et al. (1981). "The Mathematical Theory of the Dynamics of Biological Populations". Vol. II. Academic Press.

[8] Hoppensteadt, F. (2006). "Predator-prey model". Scholarpedia. **1** (10):1563. Bibcode:2006

SchpJ...1.1563H. doi:10.4249/scholarpedia.1563.

[9] Lotka, A. J. (1910). "Contribution to the Theory of Periodic Reaction". J. Phys. Chem. **14** (3): 271–274. doi:10.1021/j150111a004.

[10] Khaled Bathia, (2007) " Numerical solutions of the Multispecies predator-prey model by variational iteration method", Jordan, ISSN 1549-3636.

[11] Steven C. Chapra. "Applied Numerical Methods with MATLAB for Engineers and Scientists". HcGraw-Hill, New York, NY10020, third edition, 2012.

[12] N. Shawagfer., D. Kaya (2004) "Comparing Numerical Methods for the Solutions of Systems of Ordinary Differential Equations".Elsevier, Applied Mathematics Letters 17, pp. 323-328.

[13] C. Senthilnathan(2018). "A numerical Solutions of Initial Value Problems (IVP) for Ordinary Differential Equations (ODE) with Euler and Higher Order of Runge Kutta Methods Using Matlab". International Journal of Engineering Science Invention (IJESI) ISSN (online): 2319-6734, ISSN (print): 2319-6726 Volume 7, pp. 25- 31.

[14] Murad Hossen, Zain Ahmed, Rashadul Kabir, Zakir Hossan (2019). "Acomparative Investigation on Numerical Solution of Initial Value Problem by Using Modiffied Euler Method and Runge Kutta Method." ISOR Journal of Mathematics (IOSR-JM)e-ISSN: 2278-5728, P-ISSN: 2319-765X. Volume 15, pp. 40-45.

[15] J. C. Butcher. "The Numerical Analysis of Ordinary Differential Equations: Runge Kutta and General Linear Methods". John Wiley, New York, NY, 1987.

[16] John H. Mathews, Kurtis D. Fink." Numerical Methods Using Matlab", prentice Hall, Upper Saddle River, NJ 07458, third edition, 1999.

[17] Stephen J. Chapman. "MATLAB Programming for Engineers", Thomson Learning, 2004

[18] Eaqub Ali, S.M. (2006)." A Text Book of Numerical Methods with Computer Programming". Beauty Publication, Khulna.

[19] Volterra, V. (1931). "Variations and fluctuations of the number of individuals in animal species living together". In Chapman, R. N. (ed.). Animal Ecology. McGraw–Hill.

[20] Rosenzweig, M. L.; MacArthur, R.H. (1963). "Graphical representation and stability conditions of predator-prey interactions". American Naturalist. **97** (895): 209223. doi: 10.1086/282272. S2CID 84883526.

[21] Rossum, Guido Van (20 January 2009). "The History of Python: A Brief Timeline of Python". The History of Python. Archived from the original on 5 June 2020. Retrieved 5 March 2021.

[22] "Python 0.9.1 part 01/21". alt.sources archives. Archived from the original on 11 August 2021. Retrieved 11 August 2021.

[23] "Python 3.10.4 and 3.9.12 are now available out of schedule". 24 March 2022. Retrieved 24 March 2022.

[24] "Python 3.10.2, 3.9.10, and 3.11.0a4 are now available". 14 January 2022. Retrieved 15 January 2022.

[25] "Why is Python a dynamic language and also a strongly typed language - Python Wiki". wiki.python.org. Archived from the original on 14 March 2021. Retrieved 27 January 2021.