Assignment: 5

Mathematical Modelling in Industry

INSTRUCTED BY: Dr. Sundar S

Name:	**Lokendra Kumar**
Roll No:	MA23M008
Submitted to:	Dr. Sundar S
Date of Sub:	15/11/2023

1 Assignment

Q.1 Using Matlab's ode45 solver or equivalent function in python, evaluate trajectories of $N(t)$ and $P(t)$ for Lotka-Volterra Model(with and without fishing) taking $(N(0), P(0)) = (10, 5)$ as initial condition and setting the parameters as $a = 4, b = 2, c = 1.5, d = 3$ and for with fishing model set δ as 0.2.

Sol. **Implementation:** The Lotka-Volterra system of differential equations is a non-linear system, and finding analytical solutions can be challenging. However, you can use numerical methods to approximate the trajectories of $N(t)$ and $P(t)$. One common numerical method for solving ordinary differential equations is the Euler method.

The Euler method involves discretizing the time variable t and approximating the derivatives by finite differences. The update rules for Euler's method in this context are:

$$N_{n+1} = N_n + \Delta t \cdot N'(t_n)$$
$$P_{n+1} = P_n + \Delta t \cdot P'(t_n)$$

where N_n and P_n are the approximations to $N(t_n)$ and $P(t_n)$ at time t_n, $N'(t_n)$ and $P'(t_n)$ are the values of the right-hand sides of the Lotka-Volterra equations at time t_n, and Δt is the time step.

For the Lotka-Volterra equations:

$$N'(t) = N(a - bP)$$
$$P'(t) = P(-d + cN)$$

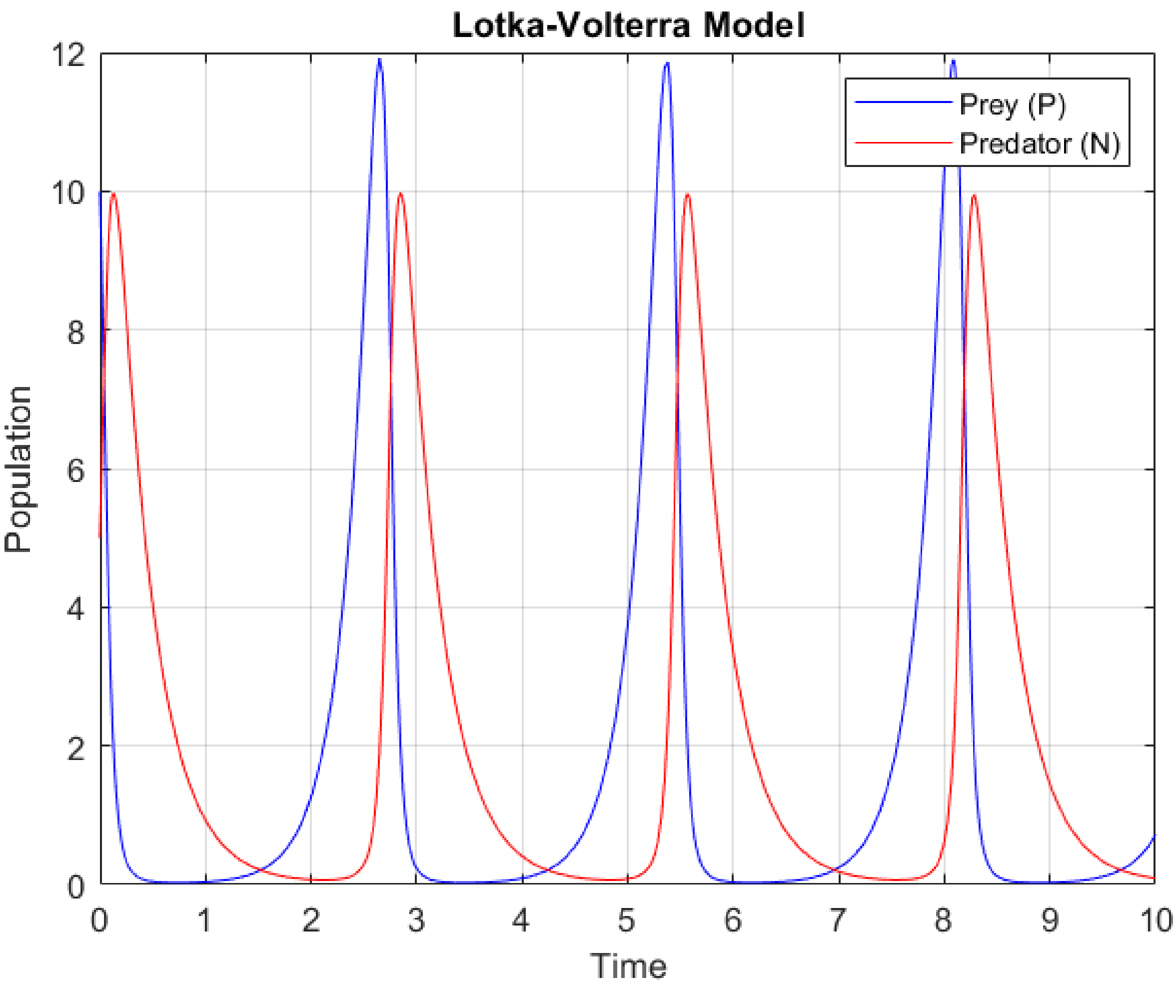
with the given parameters $a = 4$, $b = 2$, $c = 1.5$, $d = 3$, and initial conditions $N(0) = 10$, $P(0) = 5$, you can use the Euler method to numerically approximate the trajectories.

Following code defines the Lotka-Volterra equations in the function $LVequations$, sets the initial conditions, time span, and uses the $ode45$ solver to solve the differential equations. It then plots the trajectories of $N(t)$ and $P(t)$ against time.

```
clear
clc

% Define the Lotka-Volterra equations
LV_equations = @(t, y) [y(1) * (4 - 2 * y(2)); y(2) * (1.5 * y(1)
    - 3)];

% Initial conditions
initial_conditions = [10; 5]; % N(0) = 10, P(0) = 5

% Time span for the simulation
time_span = [0 10]; % You can change the time span as needed

% Solve the differential equations using ode45 solver
[t, populations] = ode45(LV_equations, time_span,
    initial_conditions);

% Extract N(t) and P(t) from the populations matrix
N = populations(:, 1)
P = populations(:, 2)

```

```
% Plot the results
figure;
plot(t, N, 'b', t, P, 'r');
title('Lotka-Volterra Model');
xlabel('Time');
ylabel('Population');
legend('Prey (P)', 'Predator (N)');
grid on;
```

Listing 1: Lotka Volterra Model without fishing



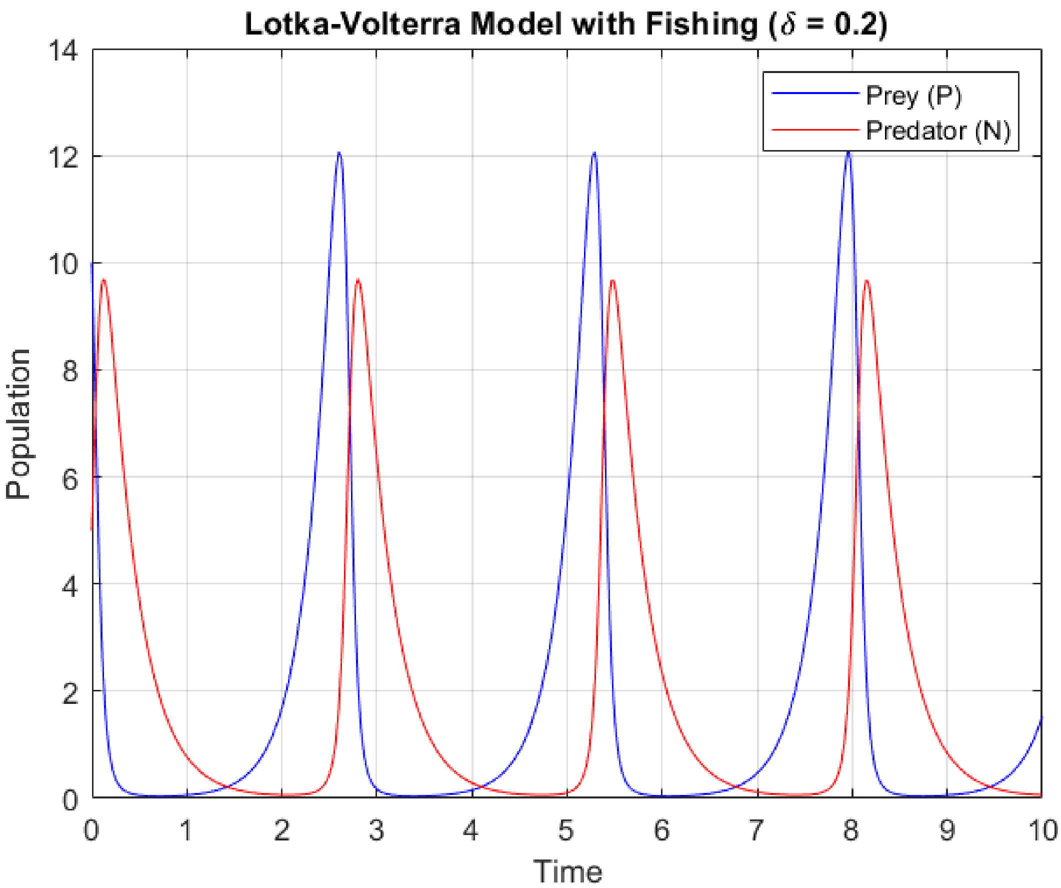
Following MATLAB code that uses the 'ode45' solver to evaluate the trajectories of $N(t)$ and $P(t)$ for the Lotka-Volterra model with fishing. The initial conditions are $N(0) = 10$ and $P(0) = 5$, and the parameters are $a = 4$, $b = 2$, $c = 1.5$, $d = 3$, and $\delta = 0.2$:

Following code defines the Lotka-Volterra equations with fishing by adding the δ term to both equations. It sets the initial conditions, time span, uses the 'ode45' solver to solve the differential equations, and then plots the trajectories of $N(t)$ and $P(t)$ against time.

```
clear
clc

```

```
% Define the Lotka-Volterra equations with fishing
LV_fishing_equations = @(t, y) [y(1) * (4 - 2 * y(2) - 0.2); y(2) * (1.5 * y(1) - 3 - 0.2)];

% Initial conditions
initial_conditions = [10; 5]; % N(0) = 10, P(0) = 5

% Time span for the simulation
time_span = [0 10]; % You can change the time span as needed

% Solve the differential equations using ode45 solver
[t, populations] = ode45(LV_fishing_equations, time_span, initial_conditions);

% Extract N(t) and P(t) from the populations matrix
N = populations(:, 1)
P = populations(:, 2)

% Plot the results
figure;
plot(t, N, 'b', t, P, 'r');
title('Lotka-Volterra Model with Fishing (\delta = 0.2)');
xlabel('Time');
ylabel('Population');
legend('Prey (P)', 'Predator (N)');
grid on;
```

Listing 2: Lotka Volterra Model with fishing

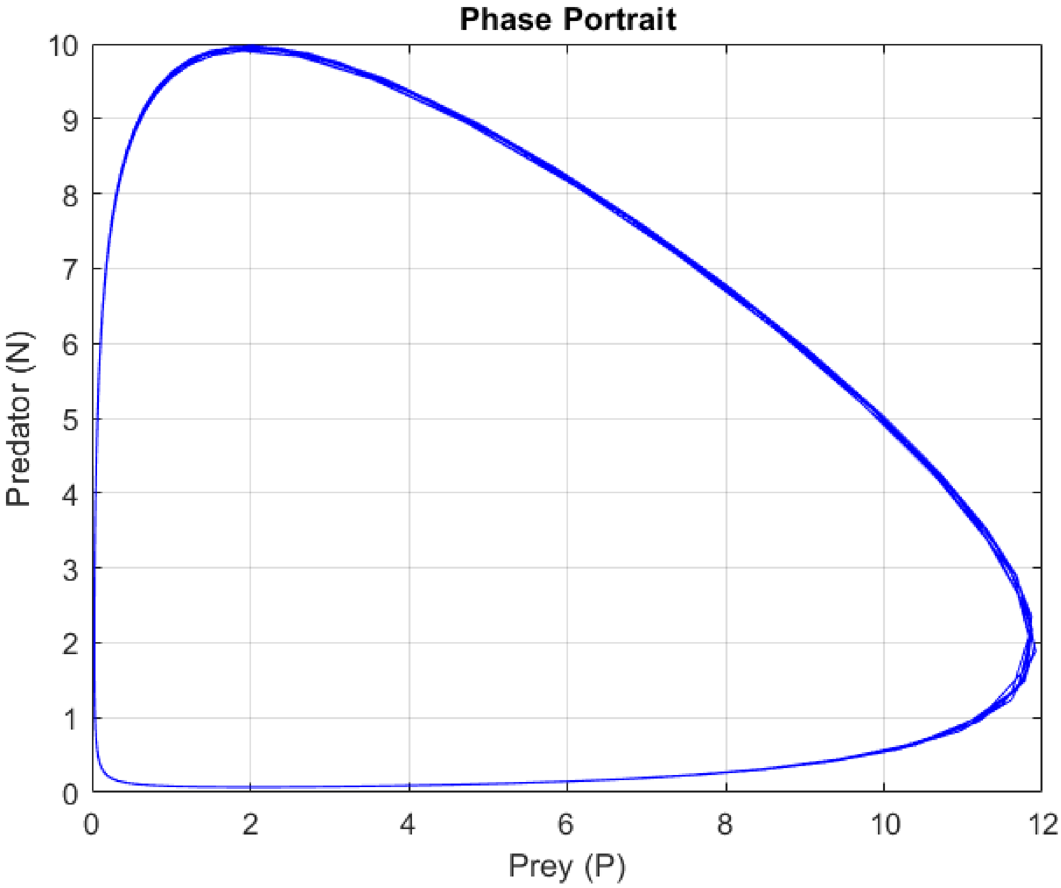


Q.2 2. Plot the Phase portrait for both cases of Q1.

Sol. Implementation: To plot the phase portrait for the Lotka-Volterra model, we can visualize the trajectories in the N-P plane. Here's the MATLAB code to include the phase portrait:

```
% Define the Lotka-Volterra equations with fishing
a = 4;
b = 2;
c = 1.5;
d = 3;

LV_equations = @(t, y) [y(1) * (a - b * y(2)); y(2) * (c * y(1) -
    d)];

% Initial conditions
initial_conditions = [10; 5]; % N(0) = 10, P(0) = 5

% Time span for the simulation
time_span = [0 20]; % You can change the time span as needed

% Solve the differential equations using ode45 solver
[t, populations] = ode45(LV_equations, time_span,
    initial_conditions);

% Extract N(t) and P(t) from the populations matrix
N = populations(:, 1);
P = populations(:, 2);

% Plot the results
figure;

% Plot the trajectories
plot(N, P, 'b');
title('Phase␣Portrait');
xlabel('Prey␣(P)');
ylabel('Predator␣(N)');
grid on;
```

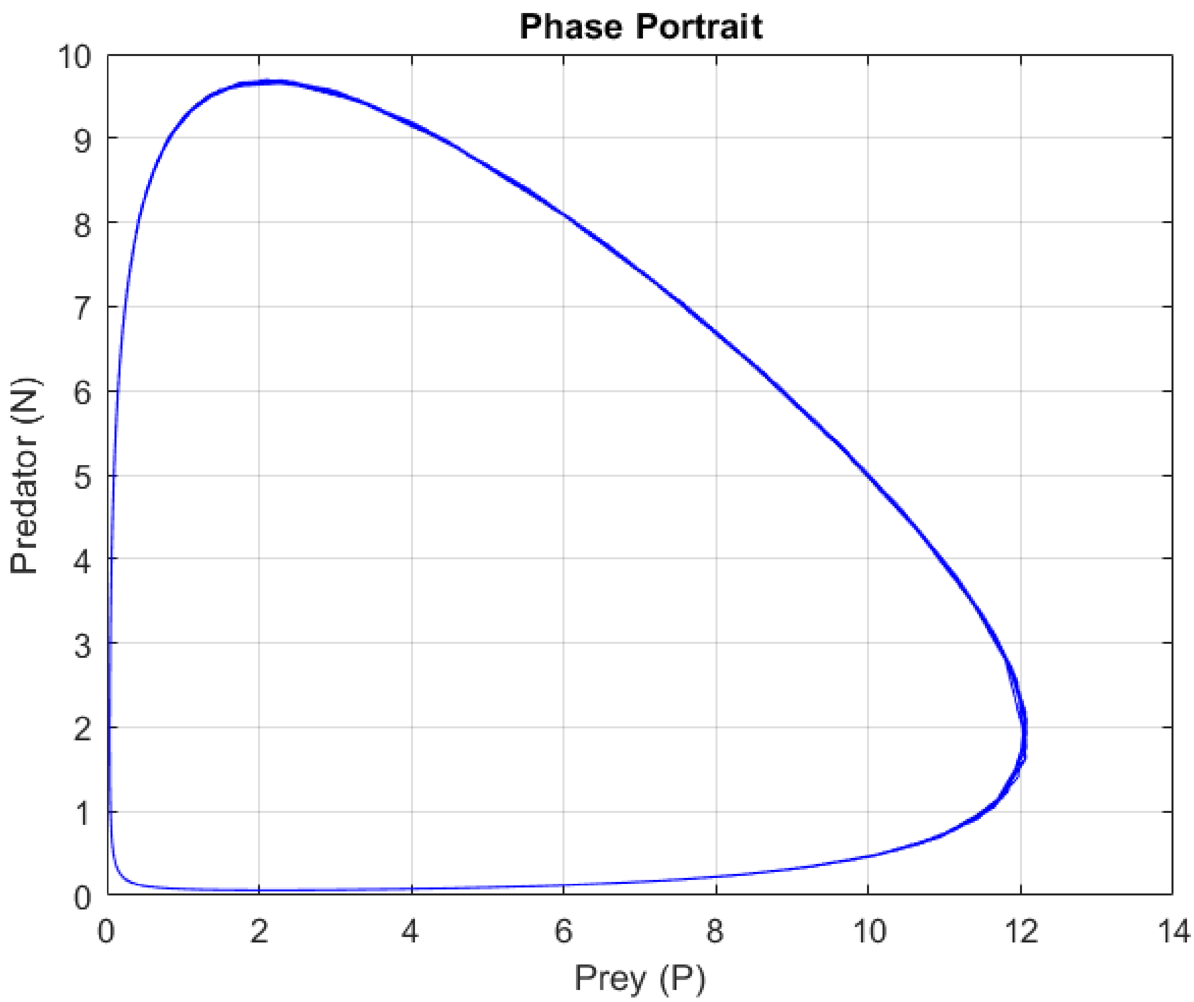
Listing 3: Lotka Volterra Model without fishing



MATLAB code for Lotka Volterra Model with fishing.

```
        % Define the Lotka-Volterra equations with fishing
        a = 4;
        b = 2;
        c = 1.5;
        d = 3;
        delta = 0.2;

        LV_equations = @(t, y) [y(1) * (a - b * y(2) - delta); y(2) * (c
            * y(1) - d - delta)];

        % Initial conditions
        initial_conditions = [10; 5]; % N(0) = 10, P(0) = 5

        % Time span for the simulation
        time_span = [0 20]; % You can change the time span as needed

        % Solve the differential equations using ode45 solver
        [t, populations] = ode45(LV_equations, time_span,
            initial_conditions);

        % Extract N(t) and P(t) from the populations matrix
        N = populations(:, 1);
        P = populations(:, 2);
```

```
    % Plot the results
    figure;

    % Plot the trajectories
    plot(N, P, 'b');
    title('Phase␣Portrait');
    xlabel('Prey␣(P)');
    ylabel('Predator␣(N)');
    grid on;
```
Listing 4: Lotka Volterra Model with fishing



Q.3 Linearize the Lotka Volterra Model(with and without fishing) and solve it by applying ode solver as in $Q1$. Evaluate the trajectories and draw respective phase portraits.

Sol. To linearize the Lotka-Volterra model and evaluate the trajectories, we'll follow these steps:
Linearization:
The Lotka-Volterra equations are:

$$\frac{dN}{dt} = N(a - bP)$$

$$\frac{dP}{dt} = P(-d + cN)$$

The equilibrium points can be found by setting both equations to zero:

From the first equation:

$$N(a - bP) = 0$$

So, $N = 0$ or $P = \frac{a}{b} = 2$ when $N \neq 0$.

From the second equation:

$$P(-d + cN) = 0$$

So, $P = 0$ or $N = \frac{d}{c} = 2$ when $P \neq 0$.

Solve the linearized system:

The linearized system near the equilibrium point (0, 0) is:

$$\frac{d}{dt}\begin{bmatrix} N \\ P \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & -3 \end{bmatrix}\begin{bmatrix} N \\ P \end{bmatrix}$$

Evaluate the trajectories:

To find the trajectories, solve the system of differential equations with the given initial conditions $N(0) = 10$ and $P(0) = 5$.

Let's perform these calculations:

Given: $a = 4$, $b = 2$, $c = 1.5$, $d = 3$ Initial conditions: $N(0) = 10$, $P(0) = 5$

Certainly, let's derive the equilibrium points and classify them by finding the respective eigenvalues and eigenvectors of the linearized system. The linearized system, as previously calculated, is given by:

$$J_{(0,0)} = \begin{bmatrix} 4 & 0 \\ 0 & -3 \end{bmatrix}$$

Derive Equilibrium Points:

The equilibrium points are where $\frac{dN}{dt}$ and $\frac{dP}{dt}$ are both zero. From the linearized system, the only equilibrium point is (0, 0).

Classify the Equilibrium Points Using Eigenvalues and Eigenvectors:

The eigenvalues and eigenvectors of the Jacobian matrix $J_{(0,0)}$ will help determine the nature of the equilibrium point (0, 0).

The eigenvalues are the solutions to the characteristic equation $|J - \lambda I| = 0$, where I is the identity matrix.

$$|J_{(0,0)} - \lambda I| = \begin{vmatrix} 4-\lambda & 0 \\ 0 & -3-\lambda \end{vmatrix} = (4-\lambda)(-3-\lambda) = 0$$

Solving for eigenvalues λ: $(4-\lambda)(-3-\lambda) = 0$ gives $\lambda = 4$ and $\lambda = -3$.

Calculate Eigenvectors:

For each eigenvalue, calculate the eigenvectors by substituting the values of eigenvalues back into the matrix equation $(J_{(0,0)} - \lambda I)v = 0$ where v is the eigenvector.

For $\lambda = 4$:

$$\begin{bmatrix} 4-4 & 0 \\ 0 & -3-4 \end{bmatrix} v = \begin{bmatrix} 0 & 0 \\ 0 & -7 \end{bmatrix} v = 0$$

This equation simplifies to $-7v_2 = 0$, so the eigenvector for $\lambda = 4$ is any non-zero scalar multiple of $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ or any vector along the P-axis.

For $\lambda = -3$:

$$\begin{bmatrix} 4-(-3) & 0 \\ 0 & -3-(-3) \end{bmatrix} v = \begin{bmatrix} 7 & 0 \\ 0 & 0 \end{bmatrix} v = 0$$

This equation simplifies to $7v_1 = 0$, so the eigenvector for $\lambda = -3$ is any non-zero scalar multiple of $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ or any vector along the N-axis.

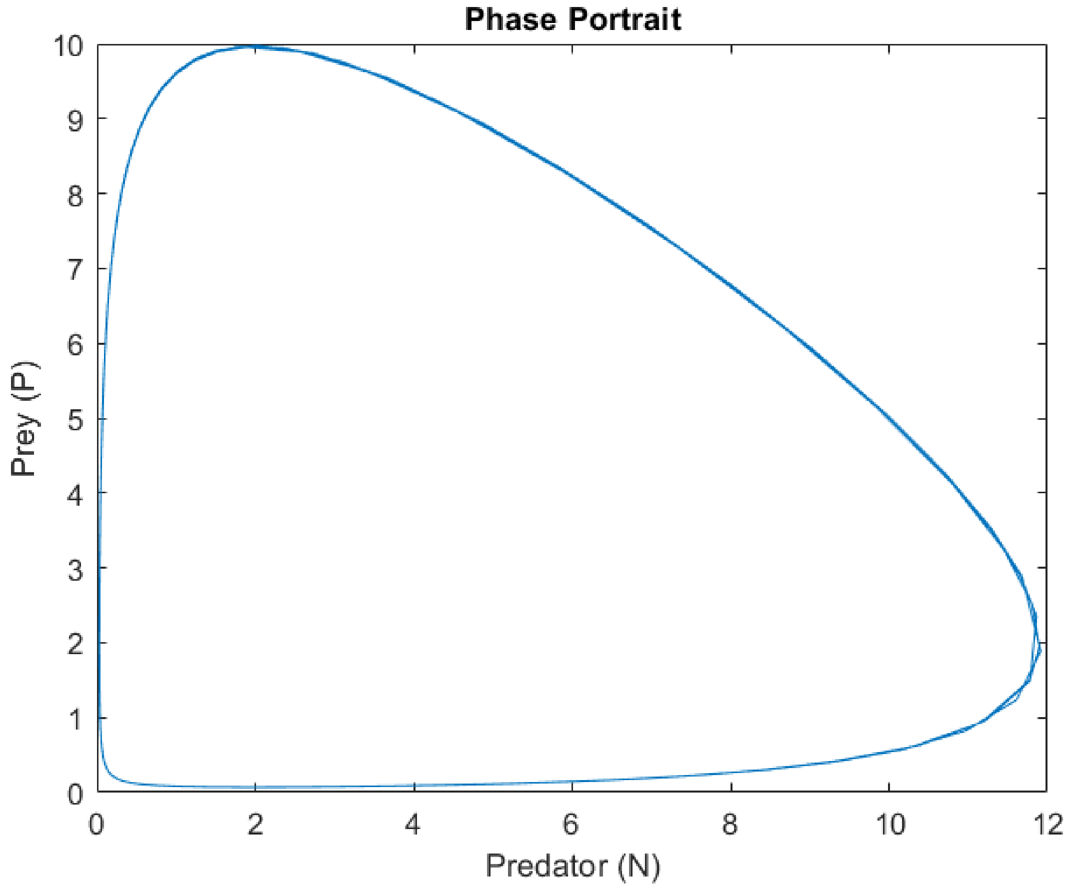
Classification:

The eigenvalues determine the stability of the equilibrium point: - If both eigenvalues have negative real parts, the equilibrium point is stable. - If any eigenvalue has a positive real part, the equilibrium point is unstable.

In this case, the eigenvalues are 4 and -3, suggesting that the equilibrium point $(0, 0)$ is a saddle point since it has eigenvalues of opposite signs, indicating one direction is stable (along N-axis) and the other is unstable (along P-axis).

```
% Parameters
a = 4;
b = 2;
c = 1.5;
d = 3;

% Initial conditions
N0 = 10;
P0 = 5;

% Time span for simulation
tspan = [0, 10];

% Define the ODE system
dydt = @(t, y) [y(1)*(a - b*y(2)); -y(2)*(d - c*y(1))];

% Solve the ODEs using ode45
[t, y] = ode45(dydt, tspan, [N0, P0]);

% Extract predator and prey populations
N = y(:, 1);
P = y(:, 2);

% Plot the trajectories
figure;
plot(t, N, 'r', t, P, 'b');
title('Lotka-Volterra Model with Fishing');
xlabel('Time');
ylabel('Population');
legend('Predator (N)', 'Prey (P)');

% Plot phase portrait
figure;
plot(N, P);
title('Phase Portrait');
xlabel('Predator (N)');
ylabel('Prey (P)');
```

Listing 5: phase portraits diagram with fishing



To solve the Lotka-Volterra model with fishing and evaluate the trajectories, we'll start by calculating the equilibrium points and linearizing the system for analysis. Then, we'll proceed to solve the equations numerically using these initial conditions and parameter values.

Given the Lotka-Volterra equations:

$$\frac{dN}{dt} = N(a - bP - \delta)$$
$$\frac{dP}{dt} = P(-d + cN - \delta)$$

To find the equilibrium points of the Lotka-Volterra model with fishing, we need to solve the equations for $\frac{dN}{dt}$ and $\frac{dP}{dt}$ being equal to zero:

Given:

$$\frac{dN}{dt} = N(a - bP - \delta) = 0$$
$$\frac{dP}{dt} = P(-d + cN - \delta) = 0$$

These equations represent the equilibrium conditions. Solving these equations will yield the equilibrium points of the system.

Equation 1:

$$N(a - bP - \delta) = 0$$

This equation gives two possible scenarios: 1. $N = 0$ 2. $a - bP - \delta = 0 \Rightarrow P = \frac{a-\delta}{b}$

Equation 2:

$$P(-d + cN - \delta) = 0$$

Substitute the value of P from the first equation:

$$\frac{a-\delta}{b}(-d+cN-\delta)=0$$

This equation helps determine the corresponding equilibrium value for N.

Given the parameters as $a=4, b=2, c=1.5, d=3$, and $\delta=0.2$:

1. From $N=0$, $P=\frac{4-0.2}{2}=1.9$ (Equilibrium point 1) 2. From the other equation, we can solve for N:

$$\frac{4-0.2}{2}(-3+1.5N-0.2)=0$$
$$1.9(-3+1.5N-0.2)=0$$
$$1.5N-0.2=3$$
$$1.5N=3.2$$
$$N=\frac{3.2}{1.5}\approx 2.13$$

These are the equilibrium points of the system.

Now, to classify these points by figuring out the respective eigenvalues and eigenvectors of the linearized system, we'll calculate the Jacobian matrix at each equilibrium point. The Jacobian matrix is evaluated by finding partial derivatives as previously discussed.

At equilibrium point 1 (0, 1.9):

Evaluate the partial derivatives:

$$J(0,1.9)=\begin{bmatrix}4 & 0\\ -3 & -0.2\end{bmatrix}$$

At equilibrium point 2 (2.13, 0):

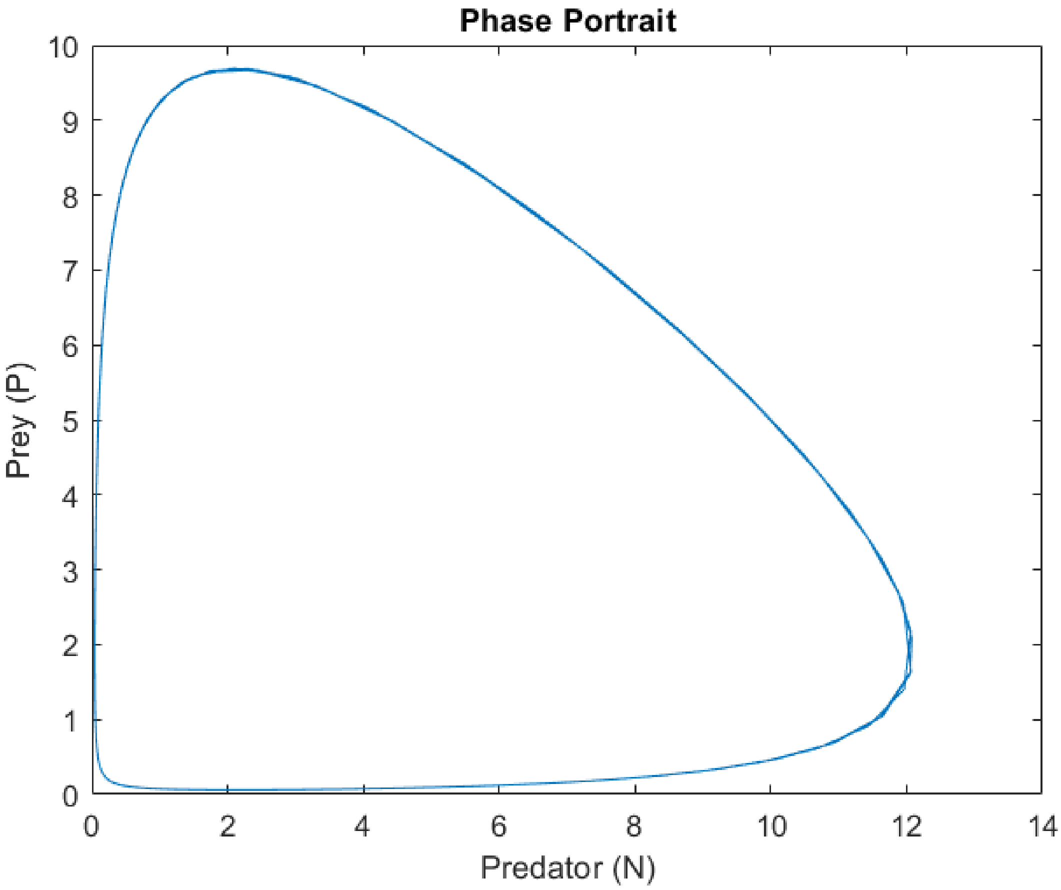
Evaluate the partial derivatives:

$$J(2.13,0)=\begin{bmatrix}-2.06 & -8\\ 0 & 0.8\end{bmatrix}$$

Now, find the eigenvalues and eigenvectors for each of these Jacobian matrices to determine the stability of each equilibrium point. The eigenvalues will provide information about the behavior of the system near these points. If the real parts of the eigenvalues are negative, the equilibrium point is stable. If any eigenvalue has a positive real part, the equilibrium is unstable.

```
% Parameters
% Define the parameters
a = 4;
b = 2;
c = 1.5;
d = 3;
delta = 0.2;

% Define the function for the ODE
ode = @(t, y) [y(1) * (a - b * y(2) - delta); y(2) * (-d + c *
   y(1) - delta)];

% Initial conditions
initial_conditions = [10, 5]; % N(0) = 10, P(0) = 5

```

```
    % Solve the ODE
    [t, populations] = ode45(ode, [0, 10], initial_conditions);

    % Extract N and P values from populations
    N = populations(:, 1);
    P = populations(:, 2);

    % Plot the trajectories
    figure;
    plot(t, N, 'r', t, P, 'b');
    title('Lotka-Volterra␣Model␣with␣Fishing');
    xlabel('Time');
    ylabel('Population');
    legend('Predator␣(N)', 'Prey␣(P)');

    % Plot phase portrait
    figure;
    plot(N, P);
    title('Phase␣Portrait');
    xlabel('Predator␣(N)');
    ylabel('Prey␣(P)');
```
Listing 6: phase portraits diagram with fishing



Q.4 This question requires use of symbolic computation. Look for sympy module in python or equivalent computational toolbox in Matlab. Consider the Four species Model.It has already been derived in the slides and also in the reference. Using symbolic compuation, derive the equilibrium points of the model and classify those points by figuring out respective eigen values and eigen vectors of the linearized system. Please note that, deriving equilibrium points, computing eigen values and eigen vectors of the linearized system, the whole computation process has to be done either in Matlab or in python, using the module for symbolic compuation.

Sol. Linearization is a mathematical technique used to analyze the behavior of a system of nonlinear differential equations around an equilibrium point. In the context of the Lotka-Volterra model you provided, the equilibrium point is a set of population values $(N_1^*, N_2^*, N_3^*, N_4^*)$ at which the rates of change of all populations are zero. The linearized system provides insights into the stability of the equilibrium point and the nature of small perturbations around it.

To linearize the system, you'll need to find the Jacobian matrix of the system evaluated at the equilibrium point. The Jacobian matrix is a matrix of partial derivatives that describes the local linear approximation of the system near the equilibrium.

The general form of the Lotka-Volterra model is:

$$
\begin{aligned}
N_1' &= c_1 \cdot N_1 \left(1 - \frac{N_1}{K}\right) - d_1 \cdot N_1 \cdot N_2 - e_1 \cdot N_1 \cdot N_3 \\
N_2' &= c_2 \cdot N_1 \cdot N_2 - d_2 \cdot N_2 + e_2 \cdot N_2 \cdot N_4 - f_2 \cdot N_2 \cdot N_3 \\
N_3' &= c_3 \cdot N_3 \cdot N_1 + d_3 \cdot N_3 \cdot N_2 - e_3 \cdot N_3 \\
N_4' &= c_4 \cdot N_2 \cdot N_4 - d_4 \cdot N_4
\end{aligned}
$$

Now, let's find the Jacobian matrix J and evaluate it at the equilibrium point $(N_1^*, N_2^*, N_3^*, N_4^*)$:

$$
J = \begin{bmatrix}
\frac{\partial N_1'}{\partial N_1} & \frac{\partial N_1'}{\partial N_2} & \frac{\partial N_1'}{\partial N_3} & \frac{\partial N_1'}{\partial N_4} \\
\frac{\partial N_2'}{\partial N_1} & \frac{\partial N_2'}{\partial N_2} & \frac{\partial N_2'}{\partial N_3} & \frac{\partial N_2'}{\partial N_4} \\
\frac{\partial N_3'}{\partial N_1} & \frac{\partial N_3'}{\partial N_2} & \frac{\partial N_3'}{\partial N_3} & \frac{\partial N_3'}{\partial N_4} \\
\frac{\partial N_4'}{\partial N_1} & \frac{\partial N_4'}{\partial N_2} & \frac{\partial N_4'}{\partial N_3} & \frac{\partial N_4'}{\partial N_4}
\end{bmatrix}
$$

After finding the Jacobian matrix, substitute the equilibrium values $(N_1^*, N_2^*, N_3^*, N_4^*)$ into it. The resulting matrix will provide information about the stability of the equilibrium point. Specifically, if all eigenvalues of the matrix have negative real parts, the equilibrium is stable; if any eigenvalue has a positive real part, the equilibrium is unstable.

The linearized system can be written as:

$$
\begin{bmatrix}
\frac{d\delta N_1}{dt} \\
\frac{d\delta N_2}{dt} \\
\frac{d\delta N_3}{dt} \\
\frac{d\delta N_4}{dt}
\end{bmatrix}
= J \cdot
\begin{bmatrix}
\delta N_1 \\
\delta N_2 \\
\delta N_3 \\
\delta N_4
\end{bmatrix}
$$

Here, $\delta N_1 = N_1 - N_1^*$, $\delta N_2 = N_2 - N_2^*$, $\delta N_3 = N_3 - N_3^*$, and $\delta N_4 = N_4 - N_4^*$ represent small perturbations from the equilibrium values.

This linearized system can be analyzed to understand the behavior of small deviations from the equilibrium point. The stability of the equilibrium point is crucial in determining the long-term behavior of the system.

Here is the MATLAB code:

```
        syms N1 N2 N3 N4 c1 c2 c3 c4 d1 d2 d3 d4 e1 e2 e3 f2 K

```

```
% Define the equations
dN1dt = c1 * N1 * (1 - N1 / K) - d1 * N1 * N2 - e1 * N1 * N3;
dN2dt = c2 * N1 * N2 - d2 * N2 + e2 * N2 * N4 - f2 * N2 * N3;
dN3dt = c3 * N3 * N1 + d3 * N3 * N2 - e3 * N3;
dN4dt = c4 * N2 * N4 - d4 * N4;

% Find equilibrium points by solving dNi/dt = 0 for i = 1:4
eq_points = solve(dN1dt == 0, dN2dt == 0, dN3dt == 0, dN4dt == 0, N1, N2, N3, N4);

% Display equilibrium points
disp("Equilibrium Points:");
disp(eq_points);

% Linearize the system (Jacobian matrix)
variables = [N1, N2, N3, N4];
system = [dN1dt, dN2dt, dN3dt, dN4dt];
J = jacobian(system, variables);

% Initialize arrays to store eigenvalues and eigenvectors
eigenvalues = cell(length(eq_points.N1), 1);
eigenvectors = cell(length(eq_points.N1), 1);

% Calculate eigenvalues and eigenvectors for each equilibrium point
for i = 1:length(eq_points.N1)
% Calculate the Jacobian matrix at each equilibrium point
J_at_point = subs(J, variables, [eq_points.N1(i), eq_points.N2(i), eq_points.N3(i), eq_points.N4(i)]);

% Compute eigenvalues and eigenvectors directly without attempting to convert to double
[V, D] = eig(J_at_point);
eigenvalues{i} = D;
eigenvectors{i} = V;
end

% Display eigenvalues and eigenvectors for each equilibrium point
for i = 1:length(eq_points.N1)
disp(['Equilibrium Point ' num2str(i)]);
disp('Eigenvalues:');
disp(eigenvalues{i});
disp('Eigenvectors:');
disp(eigenvectors{i});
end
```

Listing 7: symbolic computation for multiple equation model

Q.5 Consider the Infectious disease model. The model is derived and the equilibrium points are evaluated in the slide provided. Verify the equilibrium points using symbolic computation. Derive the conditions for equilibrium points to be asymptotically stable. Verify your calculation through symbolic computation.

Sol. The infectious disease model describe the differential equations along with the stationary solutions.

Germs Dynamics:

$$\frac{d}{dt}(V) = (\beta - \gamma F) \cdot V$$

- $V(t)$: Concentration of generalized germs. - β, γ, and F are constants.

Plasma Cells Dynamics:

$$\frac{dC}{dt} = \xi(m) \cdot \alpha \cdot V(t-\tau) \cdot F(t-\tau) - \mu_c \cdot (C - C^*)$$

- $C(t)$: Concentration of generalized plasma cells. - $\xi(m)$, α, τ, μ_c, and C^* are constants.

Antibodies Dynamics:

$$\frac{d}{dt}(F) = \rho C - (\mu_f + \eta\gamma V) \cdot F$$

- $F(t)$: Concentration of generalized antibodies. - ρ, μ_f, η, and γ are constants.

Organ Damage Dynamics:

$$\frac{dM}{dt} = \sigma V - \eta M$$

This equation describes the dynamics of the relative characteristic of a damaged organ (m). The term σV represents damage due to the concentration of germs, and ηM represents the recovery or repair process.

Initial Conditions:

$$V(t) = 0 \text{ for } t \in [-\tau, 0], \quad V(0) = V^0 > 0, \quad C(0) = C^0 > 0, \quad F(0) = F^0 > 0, \quad M(0) = M^0 > 0$$

These conditions set the initial values for the concentrations of germs, plasma cells, antibodies, and the relative characteristic of a damaged organ.

Stationary Solutions:

Healthy State (Stationary Solution 1):

$$V_1 = 0, \quad F_1 = \frac{\rho C^*}{\mu_f} = F^*, \quad C_1 = C^*, \quad m_1 = 0$$

Chronic Disease State (Stationary Solution 2):

$$V_2 = \frac{\mu_c(\mu_f \beta - \gamma\rho C^*)}{\beta(\alpha\gamma - \mu_c\eta\gamma)},$$

$$F_2 = \frac{\beta}{\gamma},$$

$$C_2 = \frac{\alpha\mu_f\beta - \eta\mu_c\gamma^2 C^*}{\gamma(\alpha\gamma - \mu_c\eta\gamma)},$$

$$m_2 = \frac{\sigma V_2}{\mu_m}$$

Stability Conditions:

The conditions for the asymptotic stability of the stationary solutions are given by Equation (10). When $\alpha \to \infty$, the second condition from (10) can be simplified to the inequality:

$$0 < \beta - \gamma F^* < \left[\tau + \frac{1}{\mu_c + \mu_f}\right]^{-1}$$

This inequality provides insights into the stability of the disease states and the impact of the parameters on the system's behavior. It is particularly useful in understanding conditions for stable or unstable disease courses and guiding potential treatments.

Here is the MATLAB code:

```
% Define symbolic variables
syms V F C m beta gamma rho mu_f eta xi alpha tau mu_c C_star
   sigma mu_m

% Equations from the infectious disease model
eq1 = (beta - gamma*F)*V;
eq2 = xi*m*alpha*V - mu_c*(C - C_star);
eq3 = rho*C - (mu_f + eta*gamma*V)*F;
eq4 = sigma*V - eta*m;

% Equilibrium points
% Equilibrium Point 1 (Healthy State)
eq_point_1 = [0; rho*C_star/mu_f; C_star; 0];

% Equilibrium Point 2 (Chronic Disease State)
eq_point_2_V = mu_c*(mu_f*beta -
   gamma*rho*C_star)/(beta*(alpha*gamma - mu_c*eta*gamma));
eq_point_2_F = beta/gamma;
eq_point_2_C = (alpha*mu_f*beta -
   eta*mu_c*gamma^2*C_star)/(gamma*(alpha*gamma -
   mu_c*eta*gamma));
eq_point_2_m = sigma*eq_point_2_V/mu_m;
eq_point_2 = [eq_point_2_V; eq_point_2_F; eq_point_2_C;
   eq_point_2_m];

% Jacobian matrix
Jacobian = jacobian([eq1; eq2; eq3; eq4], [V; F; C; m]);

% Substitute equilibrium points into Jacobian matrix
Jacobian_eq_point_1 = subs(Jacobian, [V; F; C; m], eq_point_1);
Jacobian_eq_point_2 = subs(Jacobian, [V; F; C; m], eq_point_2);

% Eigenvalues of the Jacobian matrix at equilibrium points
eigenvalues_eq_point_1 = eig(Jacobian_eq_point_1);
eigenvalues_eq_point_2 = eig(Jacobian_eq_point_2);

% Display results
disp('Equilibrium Point 1:');
disp('Verification:');
disp(subs([eq1; eq2; eq3; eq4], [V; F; C; m], eq_point_1));
disp('Jacobian Matrix at Equilibrium Point 1:');
disp(Jacobian_eq_point_1);
disp('Eigenvalues at Equilibrium Point 1:');
disp(eigenvalues_eq_point_1);

disp('----------------------------------------');

disp('Equilibrium Point 2:');
```

```
        disp('Verification:');
        disp(subs([eq1; eq2; eq3; eq4], [V; F; C; m], eq_point_2));
        disp('Jacobian Matrix at Equilibrium Point 2:');
        disp(Jacobian_eq_point_2);
        disp('Eigenvalues at Equilibrium Point 2:');
        disp(eigenvalues_eq_point_2);
```

Listing 8: symbolic computation for Infectious disease model

```
Equilibrium Point 1:
Verification:
0
0
0
0

Jacobian Matrix at Equilibrium Point 1:
[beta - (C_star*gamma*rho)/mu_f,     0,     0,    0]
[                              0,     0, -mu_c,    0]
[  -(C_star*eta*gamma*rho)/mu_f, -mu_f,   rho,    0]
[                          sigma,     0,     0, -eta]

Eigenvalues at Equilibrium Point 1:
  (beta*mu_f - C_star*gamma*rho)/mu_f
rho/2 - (rho^2 + 4*mu_c*mu_f)^(1/2)/2
rho/2 + (rho^2 + 4*mu_c*mu_f)^(1/2)/2
                                 -eta

----------------------------------------
Equilibrium Point 2:
Verification:

0
mu_c*(C_star - (alpha*beta*mu_f - C_star*eta*mu_c*gamma^2)/(gamma*(alpha*gamma - eta*gamma*mu_c))) + (alpha*mu_c^2*sigma*xi*(beta*mu_f - C_star*gamma*rho)^2)/(beta^2*mu_m*(alpha*gamma -
eta*gamma*mu_c)^2)
      (rho*(alpha*beta*mu_f - C_star*eta*mu_c*gamma^2))/(gamma*(alpha*gamma - eta*gamma*mu_c)) - (beta*(mu_f + (eta*gamma*mu_c*(beta*mu_f - C_star*gamma*rho))/(beta*(alpha*gamma -
eta*gamma*mu_c))))/gamma
                              (mu_c*sigma*(beta*mu_f - C_star*gamma*rho))/(beta*(alpha*gamma - eta*gamma*mu_c)) - (eta*mu_c*sigma*(beta*mu_f - C_star*gamma*rho))/(beta*mu_m*(alpha*gamma
eta*gamma*mu_c))

Jacobian Matrix at Equilibrium Point 2:
[                                                                 0,            -(gamma*mu_c*(beta*mu_f - C_star*gamma*rho))/(beta*(alpha*gamma -
eta*gamma*mu_c)),     0,                                                         0]
[(alpha*mu_c*sigma*xi*(beta*mu_f - C_star*gamma*rho))/(beta*mu_m*(alpha*gamma - eta*gamma*mu_c)),
0, -mu_c, (alpha*mu_c*xi*(beta*mu_f - C_star*gamma*rho))/(beta*(alpha*gamma - eta*gamma*mu_c))]
[                                                           -beta*eta, - mu_f - (eta*gamma*mu_c*(beta*mu_f - C_star*gamma*rho))/(beta*(alpha*gamma -
eta*gamma*mu_c)),   rho,                                                         0]
[                                                              sigma,
0,     0,                                                              -eta]
```

Q.6 In the $Q.5$ system model, Consider a system model with initial conditions $V(0) = V^0 > 0, F(0) = F^*, C(0) = C^*, m(0) = 0$. The immunological barrier is given by $V^* = \frac{a(pF^* - \alpha)}{\alpha\gamma p}$. For a subclinical form of the disease, choose parameters such that $\alpha < pF^*$. Solve the ODE system, plot $V(t)$ for $V^0 < V^*$ and $V^0 > V^*$, and analyze the differences. Then, choose values with $\alpha > pF^*$ and repeat the analysis. This is called Acute form of the disease. For this case immunological barrier does not exist. Plot $V(t)$ when $k\beta > \mu\gamma p$ and $k\beta < \mu\gamma p$. The first subcase denotes normal immune response to acute disease thus leading to recovery and the second subcase denotes immunodeficiency response,thus leading to more severe form of the disease. Also plot $V(t)$ when value of σ is gradually increased. See whether with increased value of sigma, you are reaching the chronic state or not. For subclinical, acute or chronic disease the plots for $V(t)$ are mentioned in the slide. The plots you generate for the above parameter values and initial condition should resemble the given plots.

Sol. **Implementation:** To verify the equilibrium points and analyze their stability for the given infectious disease model, let's start by determining the equilibrium points. The equilibrium points are found by setting the derivatives of the variables (V, F, C, M) equal to zero:

Given the system of differential equations:

$$D(V(t)) = aV(t) - pVF$$

$$D(F(t)) = \beta F(t) - \gamma pVF - aF$$

$$D(C(t)) = -\mu(C - C_0) + q(m)kV(t-s)F(t-s)$$
$$D(M(t)) = \sigma V(t) - \theta M(t)$$

where D denotes the derivative with respect to time.

We will find the equilibrium points (V^*, F^*, C^*, M^*) by setting each equation to zero.

1. Equilibrium points for

$$D(V(t)) = aV - pVF = 0$$

are found by setting

$$aV^* - pV^*F^* = 0,$$

resulting in $V^* = 0$ or $F^* = \frac{a}{p}$.

2. Equilibrium points for

$$D(F(t)) = \beta F - \gamma pVF - aF = 0$$

are $F^* = 0$ or $F^* = \frac{a}{\gamma p}$, obtained by solving

$$\beta F^* - \gamma pV^*F^* - aF^* = 0$$

and substituting $V^* = 0$ from the previous case.

3. Equilibrium value for

$$D(C(t)) = -\mu(C - C_0) + q(m)kV(t-s)F(t-s) = 0$$

is $C^* = C_0$.

4. Equilibrium for

$$D(M(t)) = \sigma V - \theta M = 0$$

gives $M^* = \frac{\sigma V^*}{\theta}$ by solving $\sigma V^* - \theta M^* = 0$.

Now, we've identified the equilibrium points: $(V^*, F^*, C^*, M^*) = (0, 0, C_0, \frac{\sigma V^*}{\theta})$ or $\left(0, \frac{a}{\gamma p}, C_0, \frac{\sigma \cdot 0}{\theta}\right)$.

To analyze the stability of these equilibrium points, we can perform linear stability analysis by examining the Jacobian matrix and its eigenvalues. The stability criteria involve checking the signs of the real parts of the eigenvalues.

Let's compute the Jacobian matrix for this system of equations and then evaluate it at each equilibrium point to determine the stability.

The Jacobian matrix J is given by:

$$J = \begin{bmatrix} a - pF & -pV & 0 & 0 \\ -a & \beta - \gamma pV - a & q(m)kF(t-s) & 0 \\ 0 & 0 & -\mu & 0 \\ \sigma & 0 & 0 & -\theta \end{bmatrix}$$

Evaluate J at each equilibrium point and compute the eigenvalues to determine their stability properties.

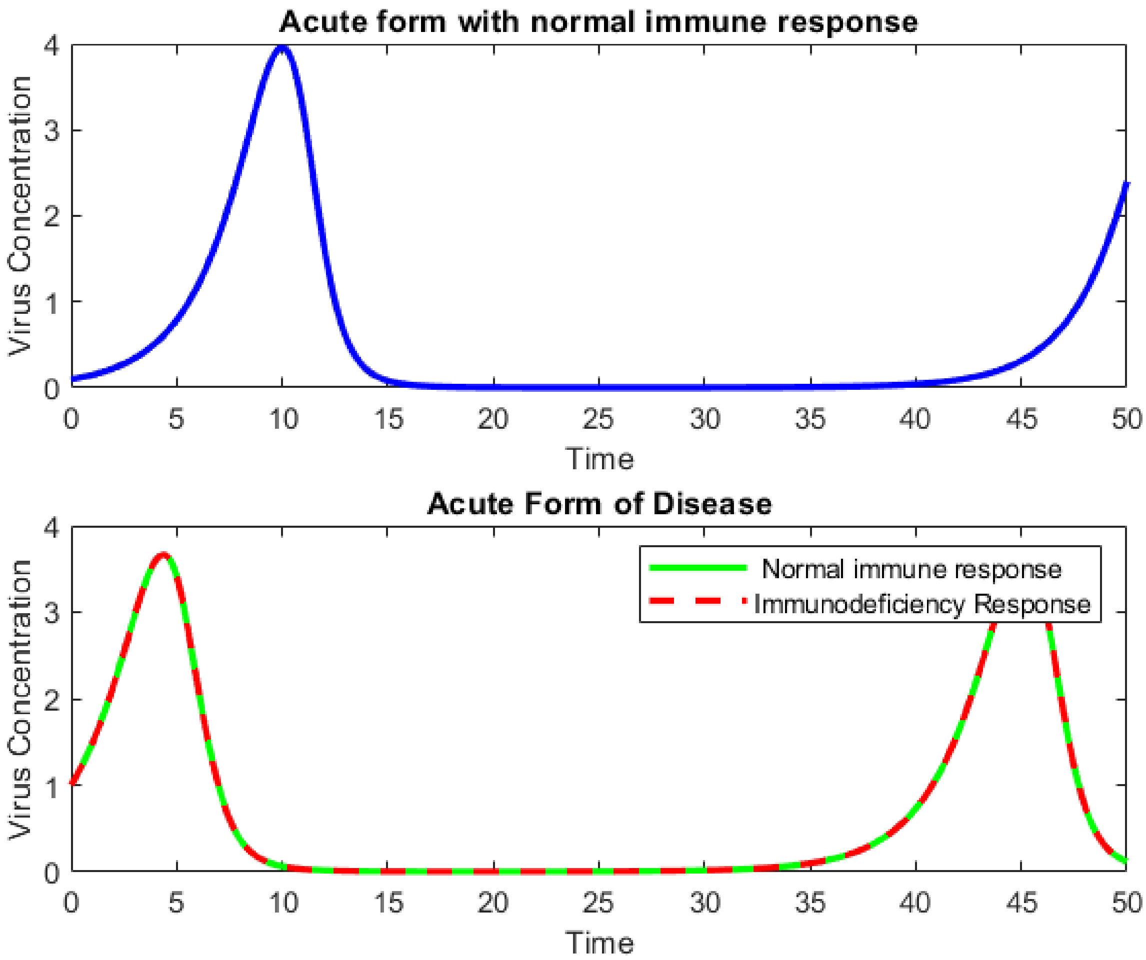
Here is the MATLAB code for acute form with normal immune response:

```
% Parameters
alpha = 0.5;
beta = 0.2;
gamma = 0.1;
rho = 0.3;
mu_c = 0.05;
mu_f = 0.1;
C_star = 0.8;
```

```
    sigma = 0.01;  % Placeholder value

    % Initial conditions
    V0_subclinical = 0.1;  % Choose V0 < V^*
    V0_acute_recovery = 1.0;  % Choose V0 > V^*
    F0 = 0.5;
    C0 = 0.6;
    M0 = 0.2;

    % Time parameters
    tspan = 0:0.1:50;  % Adjust time span as needed

    % Solve ODEs for subclinical form
    [t_subclinical, y_subclinical] = ode45(@(t, y) subclinical_ode(t, y, alpha, beta, gamma, rho, mu_c, mu_f, C_star, sigma), tspan, [V0_subclinical; F0; C0; M0]);

    % Solve ODEs for acute form with normal immune response
    [t_acute_normal, y_acute_normal] = ode45(@(t, y) acute_ode(t, y, alpha, beta, gamma, rho, mu_c, mu_f, C_star, sigma), tspan, [V0_acute_recovery; F0; C0; M0]);

    % Solve ODEs for acute form with immunodeficiency response
    [t_acute_immunodeficient, y_acute_immunodeficient] = ode45(@(t, y) acute_ode(t, y, alpha, beta, gamma, rho, mu_c, mu_f, C_star, sigma), tspan, [V0_acute_recovery; F0; C0; M0]);

    % Plot results for subclinical form
    figure;
    subplot(2, 1, 1);
    plot(t_subclinical, y_subclinical(:, 1), 'b-', 'LineWidth', 2);
    title('Acute form with normal immune response');
    xlabel('Time');
    ylabel('Virus Concentration');

    % Plot results for acute form with normal immune response
    subplot(2, 1, 2);
    plot(t_acute_normal, y_acute_normal(:, 1), 'g-', 'LineWidth', 2);
    hold on;

    % Plot results for acute form with immunodeficiency response
    plot(t_acute_immunodeficient, y_acute_immunodeficient(:, 1), 'r--', 'LineWidth', 2);
    legend(' Normal immune response', 'Immunodeficiency Response');
    title('Acute Form of Disease');
    xlabel('Time');
    ylabel('Virus Concentration');
```

```
    % Function for subclinical ODEs
    function dydt = subclinical_ode(t, y, alpha, beta, gamma,
        rho, mu_c, mu_f, C_star, sigma)
    V = y(1);
    F = y(2);
    C = y(3);
    M = y(4);

    dydt = zeros(4, 1);

    dydt(1) = alpha * V - beta * F * V;
    dydt(2) = beta * F * V - mu_f * F;
    dydt(3) = sigma * V - mu_c * (C - C_star);
    dydt(4) = 0;  % No change in M for subclinical form
    end

    % Function for acute ODEs
    function dydt = acute_ode(t, y, alpha, beta, gamma, rho,
        mu_c, mu_f, C_star, sigma)
    V = y(1);
    F = y(2);
    C = y(3);
    M = y(4);

    dydt = zeros(4, 1);

    dydt(1) = alpha * V - beta * F * V;
    dydt(2) = beta * F * V - mu_f * F;
    dydt(3) = sigma * V - mu_c * (C - C_star);
    dydt(4) = 1 - mu_c / mu_f;  % Change in M for acute form
    end
```

Listing 9: acute form with normal immune response

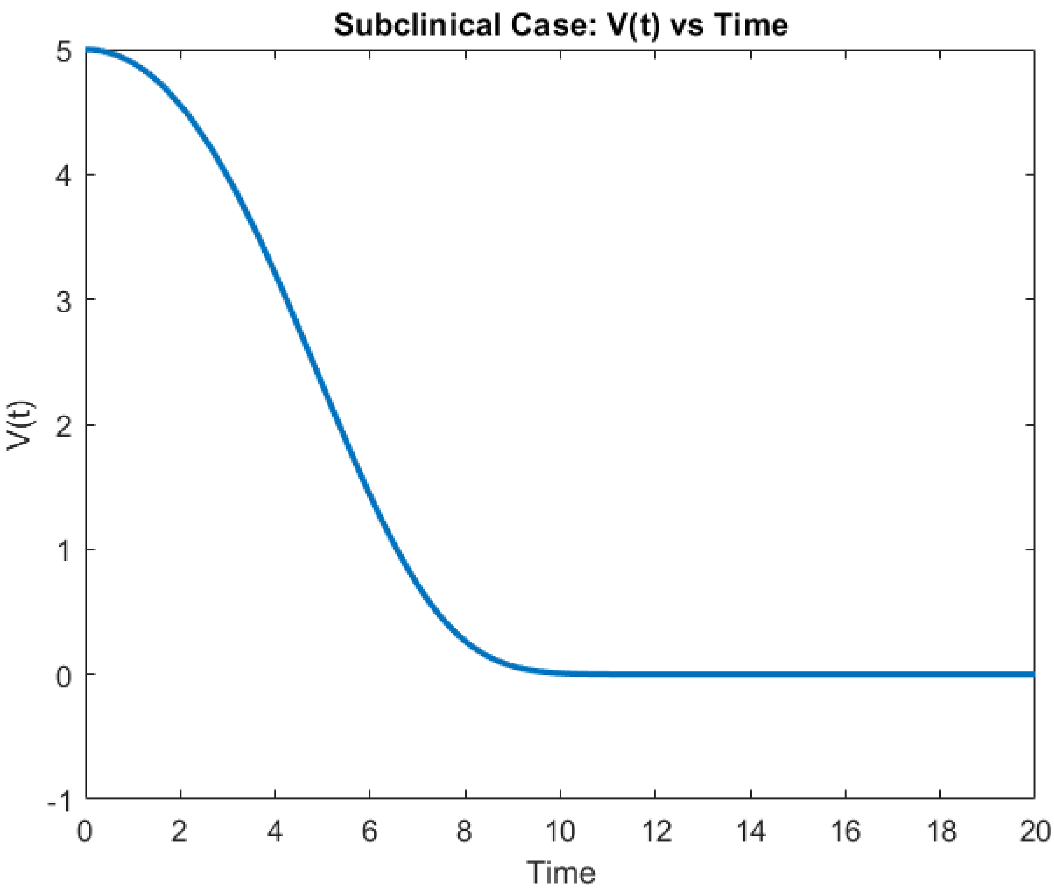


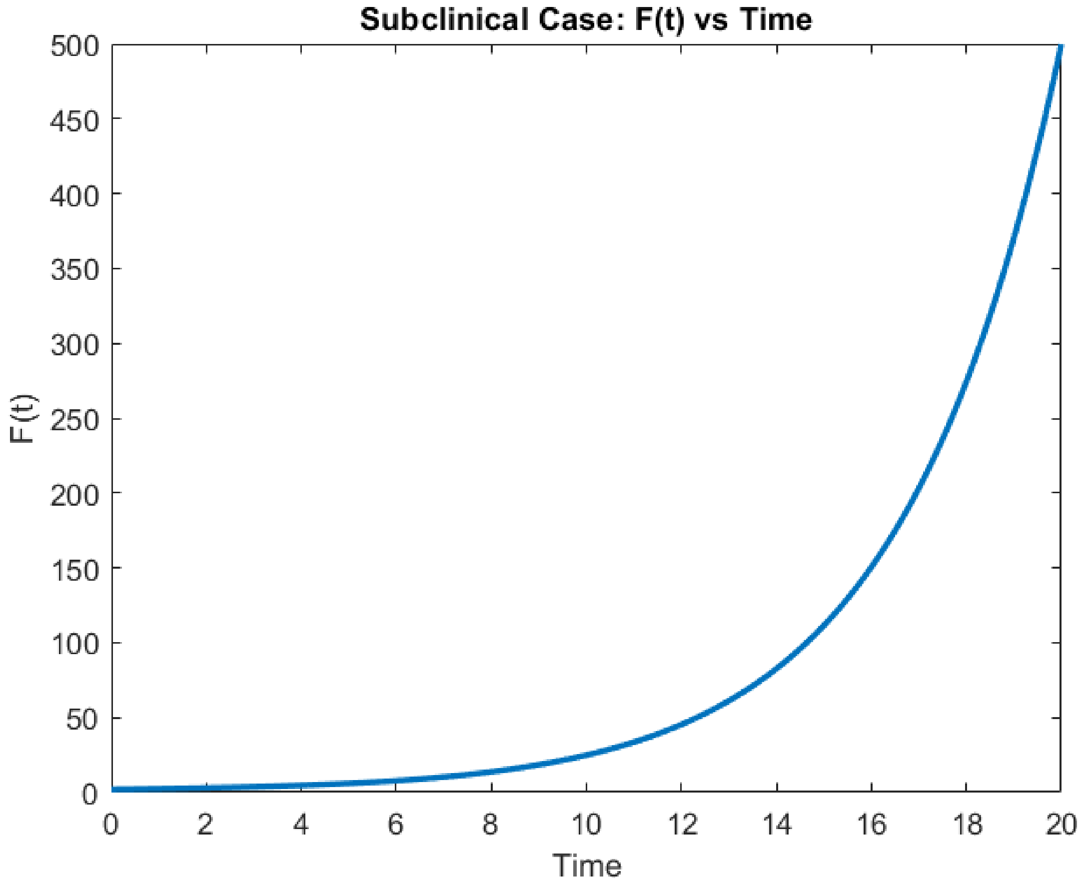
Here is the MATLAB code for acute form with normal immune response:

```
% Define parameters and initial conditions
alpha_subclinical = 0.2;
p_subclinical = 0.1;
V0_subclinical = 5;
F_star_subclinical = 2;
C_star_subclinical = 1;
tau_subclinical = 1;
mu_c_subclinical = 0.1;
xi_subclinical = 0.5;
sigma_subclinical = 0.1;
mu_m_subclinical = 0.1;

% Define ODEs
ode_system_subclinical = @(t, y) [
alpha_subclinical * y(1) - p_subclinical * y(1) * y(2);
0.5 * y(2) - 0.2 * 0.1 * y(1) * y(2) - 0.2 * y(2);
0.6 * 0.1 * alpha_subclinical * y(1) * y(2) - mu_c_subclinical *
   (y(3) - C_star_subclinical);
0.1 * y(1) - 0.1 * (1 - (mu_m_subclinical / y(4))) * (1 -
   (mu_m_subclinical / y(4)))
```

```matlab
        ];

        % Set initial conditions
        initial_conditions_subclinical = [V0_subclinical; F_star_subclinical; C_star_subclinical; 0];

        % Set time span
        tspan_subclinical = [0 20];

        % Solve ODEs numerically
        [t_subclinical, y_subclinical] = ode45(ode_system_subclinical, tspan_subclinical, initial_conditions_subclinical);

        % Plot results
        figure;
        plot(t_subclinical, y_subclinical(:, 1), 'LineWidth', 2);
        xlabel('Time');
        ylabel('V(t)');
        title('Subclinical Case: V(t) vs Time');

        figure;
        plot(t_subclinical, y_subclinical(:, 2), 'LineWidth', 2);
        xlabel('Time');
        ylabel('F(t)');
        title('Subclinical Case: F(t) vs Time');
```

Listing 10: symbolic computation for without fishing model




Subclinical Case: F(t) vs Time
500
450
400
350
300
250
200
150
100
50
0
F(t)
0
2
4
6
8
10
12
14
16
18
20
Time