Chapter 5
Predator–Prey Models

A predator is an organism that eats another organism. A prey is an organism that a predator eats. In ecology, a **predation** is a biological interaction where a predator feeds on a prey. Predation occurs in a wide variety of scenarios, for instance in wild life interactions (lions hunting zebras, foxes hunting rabbits), in herbivore–plant interactions (cows grazing), and in parasite–host interactions.

If the predator is to survive over many generations, it must ensure that it consumes sufficient amount of prey, otherwise its population will decrease over time and will eventually disappear. On the other hand, if the predator over-consumes the prey, the prey population will decrease and disappear, and then the predator will also die out, from starvation.

Thus the question arises: what is the best strategy of the predator that will ensure its survival. This question is very important to ecologists who are concerned with biodiversity. But it is also an important question in the food industry; for example, in the context of fishing, with humans as predator and fish as prey, what is the sustainable amount of fish harvesting?

In this chapter we use mathematics to provide answers to these questions. We begin with a simple example of predator–prey interaction.

We denote by x the density of a prey, that is, the number of prey animals per unit area on land (or volume, in sea) and by y the density of predators. We denote by a the net growth rate in x (birth minus natural death), and by c the net death rate of predators. The growth of predators is assumed to depend only on its consumption of the prey as food. Predation occurs when predator comes into close contact with prey, and we take this encounter to occur at an average rate b. Hence

$$\frac{dx}{dt} = ax - bxy. \tag{5.1}$$

The growth of predators is proportional to bxy (say, by a factor of d/b), so that

Electronic supplementary material The online version of this chapter (doi: 10.1007/978-3-319-29638-8_5) contains supplementary material, which is available to authorized users.

C.-S. Chou, A. Friedman, *Introduction to Mathematical Biology*,
Springer Undergraduate Texts in Mathematics and Technology,
DOI 10.1007/978-3-319-29638-8_5

$$\frac{dy}{dt} = dxy - cy. \tag{5.2}$$

In terms of dimensions,

$$[a] = \frac{1}{\text{time}}, \quad [b] = \frac{1}{\text{density of predator}} \frac{1}{\text{time}},$$

and

$$[c] = \frac{1}{\text{time}}, \quad [d] = \frac{1}{\text{density of prey}} \frac{1}{\text{time}}.$$

The system (5.1)–(5.2) has two equilibrium points. The first one is $(0,0)$; this corresponds to a situation where both species die. This equilibrium point is unstable. Indeed the Jacobian matrix at $(0,0)$ is

$$\begin{pmatrix} a & 0 \\ 0 & -c \end{pmatrix}$$

and one of the eigenvalues, namely a, is positive.

The second equilibrium point is $(\frac{c}{d}, \frac{a}{b})$ and the Jacobian matrix at this point is

$$\begin{pmatrix} 0 & \frac{-bc}{d} \\ \frac{ad}{b} & 0 \end{pmatrix}.$$

The corresponding eigenvalues are $\lambda = \pm i\sqrt{ac}$. According to Fig. 3.1(F) the portrait of all trajectories are circles. We conclude: The predator and prey will both survive forever, and their population will undergo periodic (seasonal) oscillations.

The system (5.1)–(5.2) is an example of what is known as **Lotka-Volterra** equations. One can introduce various variants into these equations. For example, if the prey population is quite congested, we may want to use the logistic growth for the prey (recall that logistic growth is introduced in Eq. (2.17)).

More general models of predator–prey are written in the form

$$\frac{dx}{dt} = xf(x,y), \quad \frac{dy}{dt} = yg(x,y),$$

where x is the prey and y is the predator, $\partial f/\partial y < 0, \partial g/\partial x > 0$, and $\partial f/\partial x < 0$ for large x, $\partial g/\partial y < 0$ for large y. The first two inequalities mean that the prey population is depleted by the predator and the predator population is increased by feeding on the prey. The last two inequalities represent natural death due to the logistic growth model.

Example 5.1. Consider the predator–prey system

$$\frac{dx}{dt} = ax(1-\frac{x}{A}) - bxy, \tag{5.3}$$

$$\frac{dy}{dt} = dxy(1-\frac{y}{B}) - cy, \tag{5.4}$$

where A and B are the carrying capacities for the prey x and the predator y, respectively. In order to compute the steady points and determine their stability we conveniently factor out x in (5.3) and y in (5.4), rewriting these equations in the form

$$\frac{dx}{dt} = x[a(1-\frac{x}{A}) - by], \tag{5.5}$$

$$\frac{dy}{dt} = y[dx(1-\frac{y}{B}) - c]. \tag{5.6}$$

Clearly, $(x,y) = (0,0)$ is a steady point with the Jacobian

$$\begin{pmatrix} a & 0 \\ 0 & -c \end{pmatrix},$$

so $(0,0)$ is unstable. The point $(\bar{x},\bar{y}) = (A,0)$ is another steady point with Jacobian

$$\begin{pmatrix} -a & -Ab \\ 0 & dA-c \end{pmatrix}.$$

Hence the steady state $(A,0)$, where only the prey survives, is stable if $dA - c < 0$.

The nonzero steady point $(\bar{x},\bar{y})$ (where $\bar{x} > 0$ and $\bar{y} > 0$) are determined by solving the equations

$$a(1-\frac{\bar{x}}{A}) - b\bar{y} = 0, \tag{5.7}$$

$$d\bar{x}(1-\frac{\bar{y}}{B}) - c = 0. \tag{5.8}$$

But before computing these points let us compute the Jacobian at $(\bar{x},\bar{y})$. In view of (5.7) we have

$$\frac{\partial}{\partial x}\{x[a(1-\frac{x}{A}) - by]\}|_{(\bar{x},\bar{y})} = \{x\frac{\partial}{\partial x}[a(1-\frac{x}{A}) - by]\}|_{(\bar{x},\bar{y})} = -\bar{x}\frac{a}{A};$$

similarly

$$\frac{\partial}{\partial y}\{y[dx(1-\frac{y}{B}) - c]\}|_{(\bar{x},\bar{y})} = \{y\frac{\partial}{\partial y}[dx(1-\frac{y}{B}) - c]\}|_{(\bar{x},\bar{y})} = -\bar{y}\frac{d\bar{x}}{B}.$$

where we have used (5.8). Hence

$$J(\bar{x},\bar{y}) = \begin{pmatrix} -\frac{\bar{x}a}{A} & -b\bar{x} \\ \bar{y}d(1-\frac{\bar{y}}{B}) & -\frac{\bar{y}d\bar{x}}{B} \end{pmatrix}. \tag{5.9}$$

We immediately see that $\text{trace}(J(\bar{x},\bar{y})) < 0$. Hence $(\bar{x},\bar{y})$ is stable if and only if $\det J(\bar{x},y) > 0$, where

$$\det J(\bar{x},\bar{y}) = d\bar{x}\bar{y}[\frac{a\bar{x}}{AB} + b(1-\frac{\bar{y}}{B})]. \tag{5.10}$$

To search for other steady points we substitute

$$\bar{y} = \frac{a}{b}(1-\frac{\bar{x}}{A}) \tag{5.11}$$

from (5.7) into (5.8) and obtain a quadratic equation for $\bar{x}$:

$$\alpha\bar{x}^2 + \beta\bar{x} - c = 0,$$

where

$$\alpha = \frac{ad}{bAB}, \quad \beta = d(1-\frac{a}{bB}).$$

The only positive solution is

$$\bar{x} = \frac{1}{2\alpha}[-\beta + \sqrt{\beta^2+4ac}]. \tag{5.12}$$

Since A is the carrying capacity of x, it is biologically natural to assume that $\bar{x} < A$. Actually, if $x(0) < A$ and $y(t)$ is assumed to be positive for all $t > 0$, then $x(t)$ will remain less than A for all $t > 0$. We can show this by contradiction: otherwise there is a first time, t_0, when $x(t)$ becomes equal to A, so that $x(t_0) = A$ and $\frac{dx}{dt}(t_0) \geq 0$. But, by (5.5)

$$\frac{dx}{dt}(t_0) = -bx(t_0)y(t_0) < 0,$$

which is a contradiction.

Similarly, it is natural to assume that $\bar{y} < B$. Hence $(\bar{x},\bar{y})$ is a biologically relevant steady state with $\bar{y} > 0$ if and only if $\bar{y}$ is given by (5.11), and the inequalities

$$\bar{x} < A, \quad \frac{a}{b}(1-\frac{\bar{x}}{A}) < B \tag{5.13}$$

hold, where $\bar{x}$ is defined by (5.12). These inequalities hold, for instance, if $\frac{a}{b} < B$ and c is small.

The above example is instructive in two ways. First, it shows that sometimes it is better to compute the Jacobian at $(\bar{x},\bar{y})$ before actually computing the steady point $(\bar{x},\bar{y})$ whose expression could be complicated. Second, it shows that by factoring out

x or y in the differential equations we are able to compute the Jacobian more easily. This last remark will be very useful in future computations, so we shall refer it as the 'factorization rule' and formulate it for general systems of equations.

Factorization Rule

Consider a system (4.1) where the f_i can be factored as follows:

$$f_1(x_1,x_2) = x_1 g_1(x_1,x_2), \quad f_2(x_1,x_2) = x_2 g_2(x_1,x_2),$$

so that

$$\frac{dx_1}{dt} = x_1 g_1(x_1,x_2), \quad \frac{dx_2}{dt} = x_2 g_2(x_1,x_2)$$

In this case there are equilibrium points $P_1 = (0,0), P_2 = (0,\bar{x}_2)$ if $g_2(0,\bar{x}_2) = 0$, $P_3 = (\bar{x}_1,0)$ if $g_1(\bar{x}_1,0) = 0$, and $P_4(\tilde{x}_1,\tilde{x}_2)$ if $g_1(\tilde{x}_1,\tilde{x}_2) = 0$, $g_2(\tilde{x}_1,\tilde{x}_2) = 0$. We can then quickly compute the Jacobian matrix $J(P_i)$ at each point P_i. For example, to compute $J(P_4)$ when $\tilde{x}_1 > 0, \tilde{x}_2 > 0$, we notice that since $g_1 = g_2 = 0$ at P_4,

$$J(P_4) = \begin{pmatrix} x_1 \frac{\partial g_1}{\partial x_1} & x_1 \frac{\partial g_1}{\partial x_2} \\ x_2 \frac{\partial g_2}{\partial x_1} & x_2 \frac{\partial g_1}{\partial x_2} \end{pmatrix}_{(\tilde{x}_1,\tilde{x}_2)}.$$

Similarly,

$$J(P_1) = \begin{pmatrix} g_1(0,0) & 0 \\ 0 & g_2(0,0) \end{pmatrix},$$

$$J(P_2) = \begin{pmatrix} g_1 & 0 \\ x_2 \frac{\partial g_2}{\partial x_1} & x_2 \frac{\partial g_1}{\partial x_2} \end{pmatrix}_{(0,\bar{x}_2)} \quad \text{when } \bar{x}_2 > 0,$$

and

$$J(P_3) = \begin{pmatrix} x_1 \frac{\partial g_1}{\partial x_1} & x_1 \frac{\partial g_1}{\partial x_2} \\ 0 & g_2 \end{pmatrix}_{(\bar{x}_1,0)} \quad \text{when } \bar{x}_1 > 0.$$

Example 5.2. Plant–herbivore model. The herbivore H feeds on plant P, which grows at rate r. We take the consumption rate of the plant to be

$$\frac{\sigma P}{1+P} H;$$

this means that, at small amount of P, H consumes P at a linear rate σP, but the rate of consumption by H is limited and, for simplicity, we assume that it cannot exceed σ. Thus,

$$\frac{dP}{dt} = rP - \sigma \frac{P}{1+P} H. \tag{5.14}$$

The equation for the herbivore is

$$\frac{dH}{dt} = \lambda\sigma\frac{P}{1+P}H - dH. \tag{5.15}$$

Here d is the death rate of H, and λ is the **yield constant**, that is,

$$\lambda = \frac{\text{mass of herbivore formed}}{\text{mass of plant used}};$$

naturally $\lambda < 1$. Note that if $\lambda\sigma < d$, that is, if the growth rate by consumption is less than the death rate, then $\frac{dH}{dt} < 0$ and the herbivore will die out. We rewrite the system (5.14)–(5.15) in the more convenient from

$$\frac{dP}{dt} = P(r - \frac{\sigma}{1+P}H), \tag{5.16}$$

$$\frac{dH}{dt} = H(\lambda\frac{\sigma P}{1+P} - d), \tag{5.17}$$

and assume that $\lambda\sigma > d$. Then the steady points are $(0,0)$ and $(\bar{P},\bar{H})$, where

$$\bar{P} = \frac{d}{\lambda\sigma - d}, \quad \bar{H} = \frac{\lambda r}{\lambda\sigma - d}.$$

Since

$$J(0,0) = \begin{pmatrix} r & 0 \\ 0 & -d \end{pmatrix},$$

the steady point $(0,0)$ is unstable. Using the factorization rule we find that

$$J(\bar{P},\bar{H}) = \begin{pmatrix} \frac{\bar{P}\sigma\bar{H}}{(1+\bar{P})^2} & -\frac{\sigma\bar{P}}{1+\bar{P}} \\ \frac{\lambda\sigma}{(1+\bar{P})^2} & 0 \end{pmatrix}.$$

Since $\operatorname{trace} J(\bar{P},\bar{H}) > 0$ and $\det J(\bar{P},\bar{H}) > 0$, and both eigenvalues of the characteristic equation have negative real parts, so $(\bar{P},\bar{H})$ is an unstable node. We conclude that the plant–herbivore model (5.14)–(5.15) has no stable steady states. In order to understand this situation better we look at the dynamics of system (5.16)–(5.17) on the P-H phase plane. The solutions of the system form trajectories on the phase plane, as depicted in Fig. 3.1; here we will analyze the direction of the trajectories, that is, $(\frac{dP}{dt}, \frac{dH}{dt})$ on the phase plane in order to get an idea of how the trajectories themselves look like. We introduce the nullclines (see definition in Chapter 4)

$$\Gamma_1 : \; r - \frac{\sigma}{1+P}H = 0, \quad \text{where } \frac{dP}{dt} = 0 \text{ on } \Gamma_1$$

$$\Gamma_2 : \; \frac{\lambda\sigma P}{1+P} - d = 0, \quad \text{where } \frac{dH}{dt} = 0 \text{ on } \Gamma_2.$$

The arrows in Figure 5.1 show the direction of the trajectories. Notice that

$$\frac{dH}{dt} > 0 \quad \text{if} \quad \lambda\sigma\frac{P}{1+P} - d > 0,$$

i.e., if $(\lambda\sigma - d)P - d > 0$, or

$$P > \frac{d}{\lambda\sigma - d}.$$

So on Γ_1

$$\frac{dP}{dt} = 0, \quad \text{and} \quad \frac{dH}{dt} > 0 \text{ if } P > \frac{d}{\lambda\sigma - d};$$

consequently the vector

$$\left(\frac{dP}{dt}, \frac{dH}{dt}\right)$$

points vertically upward at points of Γ_1 where $P > \frac{d}{\lambda\sigma - d}$. Similarly, on Γ_1,

$$\frac{dP}{dt} = 0, \quad \text{and} \quad \frac{dH}{dt} < 0 \text{ if } P < \frac{d}{\lambda\sigma - d},$$

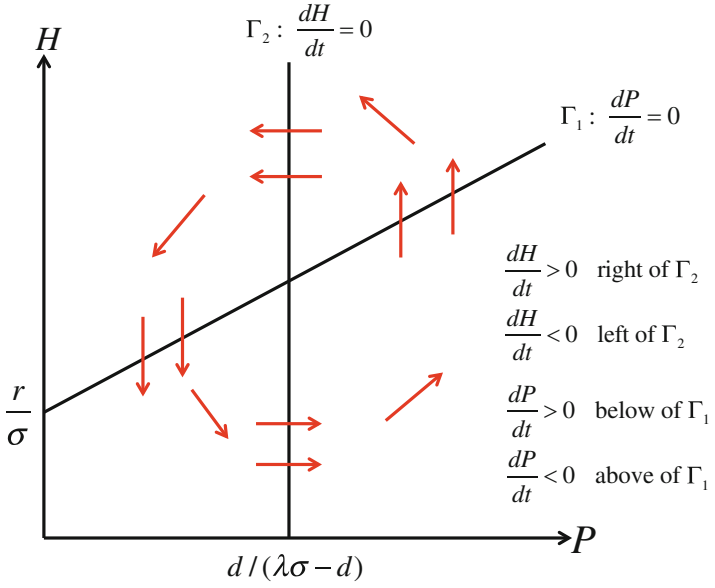

Fig. 5.1: Phase portrait of trajectories of system (5.16)–(5.17).

so that the vector $(dP/dt, dH/dt)$ points vertically downward. In the same way we can see that on Γ_2, where $P = \frac{d}{\lambda\sigma - d}$ and $\frac{dH}{dt} = 0$, the vector $(dP/dt, dH/dt)$ points

horizontally, to the right below Γ_1 (where $\frac{dP}{dt} > 0$) and to the left above Γ_1 (where $\frac{dP}{dt} < 0$). Next, in the region below Γ_1 and to the right of Γ_2,

$$\frac{dP}{dt} > 0, \quad \frac{dH}{dt} > 0$$

so that the vector $(dP/dt, dH/dt)$ points upward and to the right, as shown in Fig. 5.1. Similarly, the direction of the vector is upward to the left in the region above Γ_1 and to the right of Γ_2. On the left of Γ_2, the vector $(dP/dt, dH/dt)$ points downward: either to the left (above Γ_1) or to the right (below Γ_1). The arrows in Fig. 5.1 schematically summarize the above considerations.

We see that the phase portrait of the nonlinear system (5.16)–(5.17) is similar to the phase portrait of an unstable spiral, as in Fig. 3.1(D).

Problem 5.1. Consider a predator–prey model where the carrying capacity of the predator y depends linearly on the density of the prey:

$$\frac{dx}{dt} = ax(1 - \frac{x}{A}) - bxy,$$
$$\frac{dy}{dt} = dy(1 - \frac{y}{1+x}).$$

Find the steady points and determine their stability.

The **Allee** effect refers to the biological fact that increased fitness correlates positively with higher (but not overcrowded) population, or that 'undercrowding' decreases fitness. More specifically, if the size of a population is below a threshold then it is destined for extinction. Endangered species are often subject to the Allee effect.

Consider a predator–prey model where the prey is subject to the Allee effect,

$$\frac{dx}{dt} = rx(x - \alpha)(1 - x) - \sigma xy, \quad (0 < \alpha < 1), \tag{5.18}$$

that is, if the population $x(t)$ decreases below the threshold $x = \alpha$, then $x(t)$ will decrease to zero as $t \to \infty$. The predator y satisfies the equation

$$\frac{dy}{dt} = \lambda\sigma xy - \delta y, \tag{5.19}$$

where λ is a constant.

Problem 5.2. The point $(0,0)$ is an equilibrium point of the system (5.18)–(5.19). Determine whether it is asymptotically stable.

Problem 5.3. Show that if $\alpha < \frac{\delta}{\lambda\sigma} < 1$, then the system (5.18)–(5.19) has a second equilibrium point $(\bar{x}, \bar{y}) = (\frac{\delta}{\lambda\sigma}, \frac{r}{\sigma}(\frac{\delta}{\lambda\sigma} - \alpha)(1 - \frac{\delta}{\lambda\sigma}))$, and it is stable if

$$\frac{\delta}{\lambda\sigma} > \frac{1+\alpha}{2}.$$

This result shows that for the predator to survive, the prey must be allowed to survive, and the predator must adjust its maximum eating rate, σ, so that

$$\frac{\delta}{\lambda} < \sigma < \frac{\delta}{\lambda}\frac{2}{1+\alpha}.$$

If the Allee threshold, α, deteriorates and approaches 1, the predator must then decrease its rate of consumption of the prey and bring it closer to δ/λ, otherwise it will become extinct.

5.1 Numerical Simulations

The following algorithms 5.1 and 5.2 simulate (5.1)–(5.2). These codes demonstrate how to implement nonlinear systems (also see Chapter 4). In these codes, there are several parameters (a,b,c,d) which may be changed from simulation to simulation. We here define them as **global** variables, which can be recognized in files declaring them as global variables. It is convenient to use the global variables to define parameters that we would like to tune in models: we only have to assign values in the main file, without changing their numbers in the function files. But we also need to be careful with the names of these global parameters to prevent changing them accidentally in the code or using the same names to define other variables.

Algorithm 5.1. Main file for model (5.1)–(5.2) (main_predator_prey.m)

```
%%% This code simulates model (5.1)-(5.2).
close all, % close all the figure windows
clear all, % clear all the variables

%% define global variables
global a b c d
%% starting and final time
t0 = 0; tfinal = 5;
%% paramters
a = 5; b = 2; c = 9; d = 1;
%% initial conditions
v0 = [10,5];
[t,v] = ode45('fun_predator_prey',[t0,tfinal],v0);
subplot(2,1,1)
plot(t,v(:,1)) % plot the evolution of x
xlabel t, ylabel x
subplot(2,1,2)
plot(t,v(:,2)) % plot the evolution of y
xlabel t, ylabel y
```

Problem 5.4. Plot the time evolution of model (5.1)–(5.2) with $a = 5, b = 2$, $c = 9, d = 1$ starting from $(10,5)$, for time from 0 to 5.

Algorithm 5.2. fun_predator_prey.m

```
% This is the function file called by main_predator_prey.m
function dy = fun_predator_prey(t,v)
%% define global variables
global a b c d
dy = zeros(2,1);
dy(1) = a*v(1) - b*v(1)*v(2);
dy(2) = -c*v(2) + d*v(1)*v(2);
```

Problem 5.5. Hand draw the phase portrait for (5.1)–(5.2) with $a = 5, b = 2$, $c = 9, d = 1$ starting from several points near the nonzero steady point.

Problem 5.6. Change the codes (adding global variables A and B in both files, define the values in the main file, and change dy(1) and dy(2) in fun_predator_prey.m) to implement (5.3)–(5.4). Plot the time evolution with $a = 5, b = 2, c = 1, d = 1$, $A = 2, B = 3$ starting from $(10,5)$, for time from 0 to 10. What is the steady state you see from the simulation (you can print out the last row of the solution vector to get x and y). Verify the stability condition using this set of parameters.

5.1.1 Revisiting Euler Method for Solving ODE – Consistency and Convergence

We introduce some basic concepts in numerical analysis. These concepts will be important in general for choosing an appropriate scheme to use and assess the error of the selected algorithm. We will practice to write our own time integrator to solve ODE instead of using ode45 in MATLAB.

Consider a differential equation

$$\frac{dx}{dt} = f(x,t), \quad t \geq t_0, \quad x(t_0) = x_0, \tag{5.20}$$

where f is a continuously differentiable function in x and t and x_0 is an initial condition. Note that although here we consider a single equation where x is a real-valued function, the following discussion can be easily generalized to systems in which x and f represent vector-valued functions. There are various ways to derive Euler method; here we give one derivation based on interpolation.

Recall that forward Euler method for solving (5.20) has the formula (see Eq. 2.24)

$$X_{n+1} = X_n + hf(X_n, t_n), \tag{5.21}$$

where X_n denotes the approximate solution at time t_n, and $t_0 < t_1 < \cdots < t_N = T$ are equi-distanced grid points with $h = t_{n+1} - t_n$. These types of schemes are called **explicit schemes** because the solution X_{n+1} is explicitly defined as a function of X_n. In other words, knowing X_n, one can explicitly compute X_{n+1}. Furthermore, it

is called a **single step** method because it requires only solution at one time step in order to compute the solution at the following time step.

In order to understand how good the numerical solution is, we define **local truncation error**, d_n, to measure how closely the difference operator approximates the differential operator, for forward Euler method:

$$d_n \equiv \frac{x(t_{n+1}) - x(t_n)}{h} - f(x(t_n), t_n) = \frac{h}{2} x''(\bar{t}_n) + O(h^2),$$

where $\bar{t}_n$ is some point in the interval $[t_n, t_{n+1}]$. In other words, the truncation error is the measure of error by plugging in the exact solution into our numerical scheme. The truncation error analysis can be easily obtained by using Taylor expansion around $t = t_n$. If a method has the local truncation error $O(h^p)$, we say that the method is p-th order accurate. The forward Euler method is first order accurate because the leading term of d_n is of order h.

However, the real goal is not consistency but **convergence**. Assume Nh is bounded independently of N. The method is said to be **convergent of order** p if the **global** error e_n, where $e_n = X_n - x(t_n)$, $e_0 = 0$, satisfies

$$e_n = O(h^p), \quad n = 1, 2, \cdots, N.$$

That is, we hope that, after the accumulation of the local errors through all the steps, the errors can still be controlled and bounded by $O(h^p)$.

Example 5.3. Consider the problem

$$\frac{dy}{dt} = \lambda y, \quad y(0) = y_0.$$

We know that the exact solution is $y(t) = y_0 e^{\lambda t}$. If $\lambda < 0$, we expect that $|y(t)|$ exponentially decreases to 0. Let's apply forward Euler method to this problem, which we call the 'test problem.' We get

$$X_{n+1} = X_n + h\lambda X_n, \quad n = 0, 1, \ldots \tag{5.22}$$

with $X_0 = y_0$ and h being the time step in our discretization. Simplifying (5.22), we obtain

$$X_{n+1} = X_n(1 + h\lambda), \quad n = 0, 1, \ldots$$

and therefore

$$X_n = (1 + h\lambda)^n X_0 = (1 + h\lambda)^n y_0, \quad n = 0, 1, \ldots$$

Recall that we expect $|y(t)|$ to decrease exponentially, so we require the approximation to satisfy $|X_{n+1}| < |X_n|$, that is, $|1 + h\lambda| < 1$ $(-1 < 1 + h\lambda < 1)$. So in order to obtain the desired behavior of the solution, we need to require that

$$h < \frac{-2}{\lambda} \quad (\lambda < 0). \tag{5.23}$$

The condition (5.23) is a condition imposed on the time step, which we call the **stability condition**. If this condition is violated, then our numerical solution blows up, as can be seen in the numerical experiment in Problem 5.7.

Problem 5.7. Consider the test problem with $\lambda = 20$ and $y_0 = 1$. The sample MATLAB codes can be found in Algorithm 5.3. (a) Derive the stability condition. (b) Test $h = 0.01, 0.05, 0.1, 0.2$, with final time $T = 1$. Describe what you see and explain it theoretically.

Algorithm 5.3. Forward_Euler method for the test problem (forward_Euler.m)

```
% This is a code to solve dy/dt = lambda*(y), 0<t<1, y(t=0)=1
% (lambda=20)  using forward Euler

clear all;
lambda = -20;
h = .0005;
t = 0:h:1;
Nt = length(t);
y = zeros(Nt,2); % preset y as a zero matrix with the same length
                 % as t
y(1) = 1;          % initial condition; index starts from 1

for i = 1:(Nt-1)
   y(i+1) = y(i)+h*(lambda*(y(i)));
end

error = abs(y(end)-exp(lambda*t(end)))
plot(t,y), hold on
plot(t,exp(lambda*t),'r')
```

Problem 5.8. Consider the scalar problem

$$y' = -5ty^2 + \frac{5}{t} - \frac{1}{t^2}, \quad y(1) = 1.$$

(a) Verify that $y(t) = \frac{1}{t}$ is a solution to the problem. (b) Use forward Euler method until $t = 10$ (modify forward_Euler.m). Define the error to be the absolute value of the difference between the exact solution (in this problem, the exact solution is $\frac{1}{10}$, for $t = 10$) and the numerical solution (in the MATLAB code it will be 'y(end)'). Compute the error at $t = 10$ using $h = 0.0025, 0.005, 0.01, 0.02$. Verify that this method is first order accurate based on the errors (the error should decrease by half when you decrease h by half).

As mentioned above, the method (5.21) can be applied to systems, where x and f are vectors. Algorithm 5.4 is a sample code using forward Euler method to simulate (5.1)–(5.2). Note that in the code, we used the function 'fun_predator_pray.m'

that was defined in Algorithm 5.2. Since forward Euler is first order, we can see that when we increase h, the solution may be less accurate, and the shape is not as expected. But this inaccuracy will be gone once the time step is small enough.

Algorithm 5.4. Forward Euler method for predator–pray model (5.1)–(5.2) (forward_Euler_predator_prey.m)

```
% This code simulates model (5.1)-(5.2) using Forward Euler method.
close all, % close all the figure windows
clear all, % clear all the variables
%% define global parameters
global a b c d

%% starting and final time
t0 = 0; tfinal = 5;

%% paramters
a = 5; b = 2; c = 9; d = 1;

%% set up time step and vectors
h = .001;             % time step
t = t0 : h : tfinal; % discrete time steps
Nt = length(t);     % total number of time steps
v = zeros(Nt,2); % preset v as a zero matrix
% Note that v has two columns, each representing one variable

%% initial conditions
v(1,:) = [10,5];

for i = 1:(Nt-1)
    time = t(i); % current time
       z = v(i,:); % approximate solution at the current time step
     rhs = fun_predator_prey(time,z)';
   v(i+1,:) = z + h*rhs;
end

%% plot
subplot(2,1,1)
plot(t,v(:,1)) % plot the evolution of x
xlabel t, ylabel x
subplot(2,1,2)
plot(t,v(:,2)) % plot the evolution of y
xlabel t, ylabel y
```