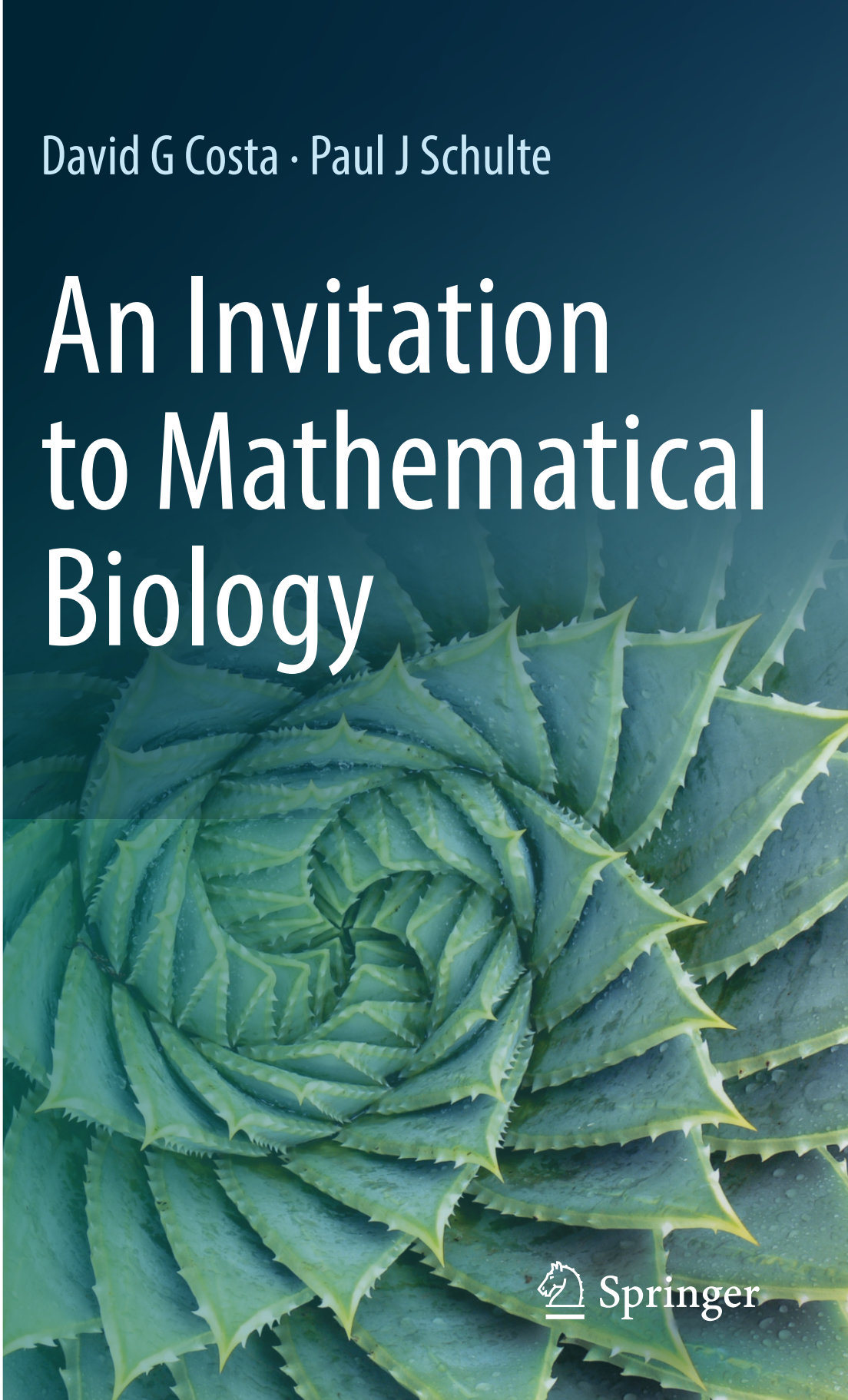
David G Costa · Paul J Schulte

An Invitation to Mathematical Biology



Springer

An Invitation to Mathematical Biology

David G Costa • Paul J Schulte

An Invitation to Mathematical Biology

Springer

David G Costa
Department of Mathematical Sciences
University of Nevada, Las Vegas
Las Vegas, NV, USA

Paul J Schulte
School of Life Sciences
University of Nevada, Las Vegas
Las Vegas, NV, USA

ISBN 978-3-031-40257-9 ISBN 978-3-031-40258-6 (eBook)
https://doi.org/10.1007/978-3-031-40258-6

Leaf arrangements in plants like this *Aloe polyphylla* (commonly called spiral aloe) form beautiful spiral patterns. Many such spiral patterns are found in biology and some have been described mathematically based on the Fibonacci Sequence of numbers. We thought of this image for the cover as an artistic expression of the links between biology and mathematics.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

What is the purpose of this text? After many years of teaching at our university, we have noticed that mathematics students tend to not have a formal background in biological systems. On the other hand, biology students often have a fear of mathematics. Many biology students may have always been interested in science and, in part, chose biology because of a perception that it was one scientific field that could be pursued without needing any math skills or background. We think that this misconception has gradually increased in significance.

Although statistics might be present in most biology degree programs, mathematics up to and beyond calculus is often not. We believe that a common educational scenario in biology has at least partially neglected mathematics. Therefore, the students are minimally trained in mathematics through graduate school and on into academic or other research work. Not being comfortable with applications of mathematics, they tend not to use it as a primary feature in their research. Later in their careers if they stay in academia, they are arguably not likely to encourage their graduate students in this regard. Thus the cycle continues.

Our goal with this text, therefore, is to provide an introduction to this interdisciplinary field in a way that will not overly terrify the undergraduate biology major, but will highlight what can be learned about biological systems by applying an approach based in mathematics. In terms of expectations for student backgrounds, we would suggest, as we do for our courses, that students should have mathematics up to at least introductory calculus, although we provide a review in an appendix of this text. For biology, students should have an introductory series in biology to provide a basic background in biology. Although additional prior course work would be useful, our goal is to draw from students aiming for majors in biology or mathematics.

In summary, this little book is intended to be an "invitation" (as the title says) to mathematical biology which: (1) can serve as a stepping stone to more comprehensive books, while (2) also offering a selected variety of topics that could be studied in deeper detail later on. It is intended for students with a minimal knowledge (freshman level) in mathematics and biology. As explained in the Introduction, the task can be accomplished with the use of computer software like Mathematica, MATLAB, or Python that we use in this text.

David Costa is a mathematician specializing in ordinary and partial differential equations, particularly through the use of the calculus of variations. Paul Schulte is a biologist specializing in plant physiology, particularly transport processes within plants. The authors have long been interested in biological applications of mathematics. Support for an undergraduate program in biomathematics was developed as part of a funded National Science Foundation grant in undergraduate biology and mathematics. We developed a two-course series in biomathematics and have continued to teach this coursework. Historically we have used the terms biomathematics and mathematical biology almost inter-changeably. But one could argue that mathematical biology is more meaningful for our purposes because it suggests that the goal is to consider biological processes and what we can learn by applying mathematical approaches. This consideration forms the basis and organization for this book.

A brief word about the field of statistics. While we would certainly agree to the importance of this field in the analysis of biological data, we feel that statistics is well represented in biology. Most undergraduate biology degree programs likely include at least one course in statistics. Therefore, this text will focus on other areas of mathematics that are not so prevalent in the usual biology degree programs.

Las Vegas, NV, USA

David G Costa
Paul J Schulte

Contents

Chapter 1
Introduction

What is mathematical biology? The field of mathematical biology may be defined as the application of mathematical approaches for understanding biological processes. This may involve traditional mathematics such as developing analytical solutions but also numerical, computational approaches utilizing computational methods. Therefore, we will describe some of these numerical methods, and we will also provide tutorials in the use of a few computer software systems that have become very useful in applying the mathematical approaches. To that end, it is worth considering that biological systems tend to be extraordinarily complex and mathematical descriptions of these systems are often not solvable by analytical methods. A quick perusal of several journals in mathematical biology shows that research papers are typically dominated by computational approaches. Thus the field of computational biology has become increasingly important, as noted in the Math 2025 report (National Research Council 2013). Computational biology has been defined by some as equivalent to bioinformatics, but we choose a broader definition encompassing any use of computational approaches to studying biological systems. Given the growing role of computational approaches in biology, an increasing focus on courses that teach computational methods will be important, as noted by Pevzner and Shamir (2009). This could include numerical methods and computer programming along with an introduction to existing software tools such as Mathematica, MATLAB, or Python.

Why is mathematical biology important? For the next generation of biologists to contribute to new mathematical directions at this interface, much greater interaction between biologists and mathematicians will be necessary. The importance of an enhanced link between mathematics and biology is clear from the NSF-sponsored workshop entitled Mathematics and Biology: The Interface, Challenges and Opportunity, where the authors describe the potential for mathematics to revolutionize biology as Newton's calculus did for physics (Levin 1992). Benefits will flow in both directions: such interactions will also enrich mathematics, as has occurred in the past where dynamical systems and chaos theory were largely developed in response to biological problems.

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_1

We are inclined to believe that many biologists traditionally do not have a strong (or even adequate) formal background in mathematics and tend to avoid using mathematics in their research. Therefore, they do not see a need to require mathematical training for their students – and the cycle continues! A modern physicist would be lost without mathematics and perhaps the same will be true of future biologists. The problem is not just with biologists and their lack of mathematical training: Michael Reed in the Department of Mathematics at Duke University has written, "most mathematicians know less biology than their 10th grade daughters" (Reed 2004). In the traditional academic environment among the sciences, most faculty are members of a specific scientific discipline and largely conduct their research and scholarly activities within that discipline. These traditional units may inhibit the degree of collaboration that will be so important for fields such as biology, where progress will depend on "the phase transition that comes with the total integration of mathematical and empirical approaches" to biology (Levin 1992). Therefore, we believe that the lack of training beginning at early career stages and the lack of facilities to encourage interaction serve as major barriers for this emerging mathematical biology field (Bialek and Botstein 2004).

As part of addressing the needs just described, this text is organized to focus on the presentation of a broad array of biology topics, many of which would be covered in introductory biology courses but typically without the use of mathematical approaches. These topics cover a range from ecological systems such as population modeling, models of epidemics and infectious diseases, and evolutionary processes such as natural selection. Mathematical approaches in this text include discrete equations and ordinary differential equations, where we will present a variety of tools ranging from analytical or graphical solutions to computational methods using software such as Mathematica, MATLAB, and Python. It is our hope that these examples and the application of readily amenable graphical approaches and computational software will serve to inspire students toward a greater appreciation for the role of mathematics in biology.

References

Bialek W, Botstein D (2004) Introductory science and mathematics education for 21st-century biologists. Science 303:788–790

Levin SA (1992) Mathematics & biology: the interface. Lawrence Berkeley Laboratory, University of California, Berkeley. Pub-701

National Research Council (2013) The mathematical sciences in 2025. The National Academies Press, Washington, DC

Pevzner P, Shamir R (2009) Computing has changed biology – biology education must catch up. Science 325:541–542

Reed M (2004) Why is mathematical biology so hard? Notices AMS 51:338–342

Chapter 2
Exponential Growth and Decay

Many physical phenomena are governed by an (approximate) exponential growth or exponential decay, and clearly biological phenomena are not an exception as we shall see next. Typical examples of such growth and decay situations, which are rather exact and appear in most calculus books, are bacteria growth, infectious disease recovery, and radioactive decay (an interested reader may want to check "compound interest" as related to exponential growth).

2.1 Exponential Growth

Example 2.1 A culture of bacteria in a food fertile medium increases in size during a certain time interval Δt (as each cell divides into two) and continues doing that for as long as there is food available in the medium. For example, let us consider the case of the bacterial species *Escherichia coli* (which is well known to live in human intestines).

Say one places 60 cells of *Escherichia coli* bacteria in a broth-nutrient medium, and each cell divides into two every 20 min. Then, after the first 20 min there are 120 cells, after 20 more minutes there are 240 cells, then 480 cells after 60 min, and so on, the number of cells keeps doubling for as long as there is food available in the medium in question, let us say 180 min.

Therefore, assuming that there is plenty of nutrients to last for 3 h ($n = 9$, 9 times 20 min is 3 h), there will be a total of

$$N = (60)2^9 \qquad \text{where } 2^9 = 512$$

In other words, there will be 30,720 cells of *Escherichia coli* in the medium after 180 min ($n = 9$). Now, if we go far beyond $n = 9$, it is natural to guess that such subdividing process will slow down, tending to level off after the food or nutrient is

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_2

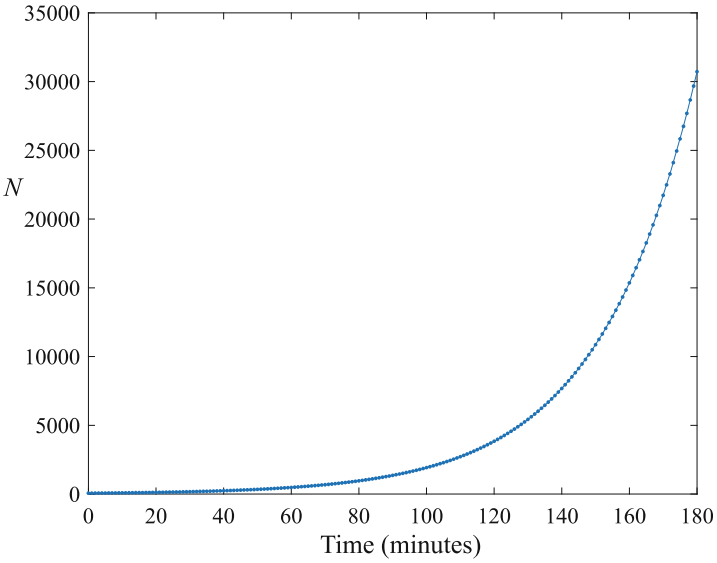


Fig. 2.1 Number of cells following exponential growth as illustrated in Example 2.1, starting with 60 cells

gone. In fact, we shall see this leveling off in another example below. Note that considering the above situation for all further integers n would eventually lead to the unrealistic exponential growth situation

$$N = (60)2^n \qquad \text{for all } n \geq 1$$

One can also express this exponential growth model in the form of a discrete time equation.

$$N_{t+1} = N_t + kN_t \tag{2.1}$$

where k is calculated from the doubling of cells in 20 min converted to our time unit of minutes:

$$k = 2^{\frac{1}{20}} - 1$$

From Eq. 2.1 and starting with 60 cells at $t = 0$, we would calculate the number of cells at $t = 60$, 120, and 180 min as 480, 3840, and 30,720 respectively (Fig. 2.1). Note that starting with 60 cells at time zero would give the same value of 30,720 $t = 3$ h as noted above.

Example 2.2 The previous example of exponential growth appears to suggest that the number of cells would increase without limit. However one can imagine that for many actual populations of organisms, limitations would be present that prevent the unbounded growth. Here, we will consider a population of bacteria growing under

Table 2.1 Bacterial cell growth over the course of several hours

Time (hours)	Population size
0	0.0031
1	0.0046
2	0.0094
3	0.0218
4	0.0469
5	0.0759
6	0.0989
7	0.1172
8	0.1310
9	0.1407
10	0.1481
11	0.1532
12	0.1551
13	0.1575
14	0.1580
15	0.1590

Population size was estimated by the optical density of the medium

lab conditions. Data were provided courtesy of Ernesto Abel-Santos in the Department of Chemistry and Biochemistry at the University of Nevada, Las Vegas. An experiment was performed measuring the growth of *Bacillus anthracis* at various hours in a certain medium. One can notice from the table that there is an initial rapid growth (like in Example 2.1) but after about 10 h, the growth slows, eventually leveling off approaching the so-called "carrying capacity" of the medium. This pattern is sometimes referred to as an *S curve* or a logistic curve (Table 2.1).

We can fit a curve to this set of data that follows a logistic function. In this case, we used a MATLAB function that minimizes the errors between the data points and the fitted curve, an approach that has been called nonlinear least squares. The fitted curve appears to reasonably approximate the actual observed data (Fig. 2.2). The curve fitting process also gives us estimates of two important parameters: a growth rate and a carrying capacity.

Looking back at the previous table and graph, we can note a value for the carrying capacity, estimated by the curve fitting as 0.1562. Indeed, it is natural to infer that the change $\Delta p_n = p_{n+1} - p_n$ is proportional to the product $p_n\,(0.1562 - p_n)$, since Δp_n is approximately zero when $(0.1562 - p_n) = 0$. In other words, we are inferring that

$$\Delta p_n = p_{n+1} - p_n = r p_n (0.1562 - p_n) \tag{2.2}$$

where $r > 0$ is some constant. In fact, the logistic curve fit described earlier gives a value of $r = 4.417$. Therefore, we get the formula

$$p_{n+1} = p_n + 4.417 p_n (0.1562 - p_n)$$

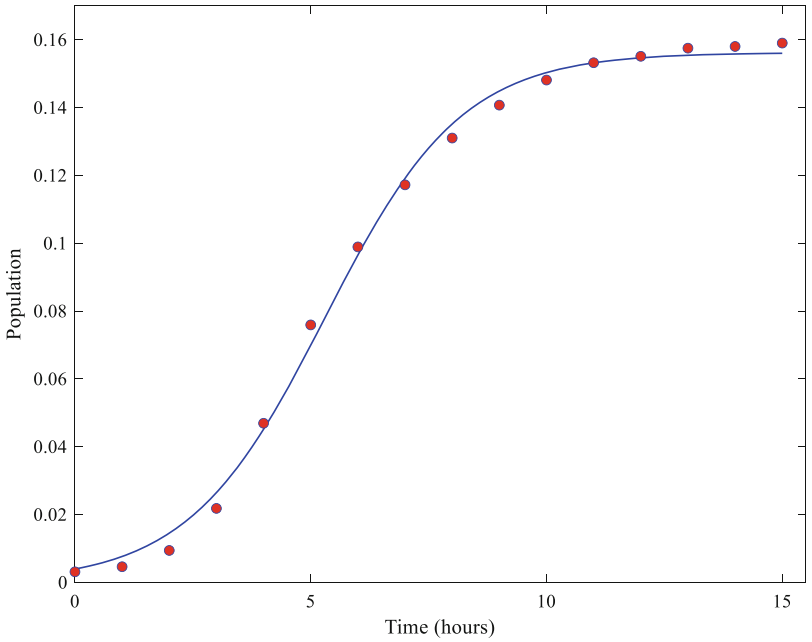


Fig. 2.2 Bacterial cell population size (estimated as optical depth) as a function of time. The circles show the observed data and the line shows the logistic curve fit to those data points

which can be solved numerically to get $p_1 = 0.0052$ from $p_0 = 0.0031$, then p_2 from p_1, and so on. If you are up to the task of doing the calculations and plotting these predicted values, you can compare those in the previous table and the plot in Fig. 2.2. Of course we can automate this process with a simple Python program as shown here:

```
# Discrete Logistic
import numpy as np
import matplotlib.pyplot as plt

r = 4.41708
K = 0.15618
steps = 15
p = np.zeros(steps+1)
n = np.zeros(steps+1)
p[0] = 0.0031
n[0] = 0.0
for i in range(0, steps):
 p[i+1] = p[i] + r*p[i]*(K - p[i])
 n[i+1] = n[i] + 1.0
```

```
# Display data
print (np.transpose([n,p]))

# plot results
plt.plot(n,p, 'b.', fillstyle='full')
plt.xlabel('Time')
plt.ylabel('p')
plt.show()

# Save results to a file
np.savetxt('DiscreteLog.txt', np.transpose([n,p]), fmt='%4.1f %
8.4f', delimiter=',', newline='\n')
```

We will examine further this logistic population model in the next chapter.

2.2 Exponential Decay

Example 2.3 Now, a realistic "exponential decay" governs the spontaneous decay of the mass m of a radioactive substance (Nagy 2011; Sharon and Sharon 2021). In this case, starting with a certain mass m_0, it gets halved after each length of time H. Here, H is called the half-life of the substance [a characteristic of each radioactive substance indicating the time in years when its mass gets halved].

Thus, an initial mass m_0 is halved to $m_0/2$ at time $t = H$, then halved to $m_0/4$ at time $t = 2H$, and so on "ad infinitum." Indeed, this exponential decay is rather exact and the half-life of radioactive substances can vary from some thousands to millions of years. For instance, carbon-14 (an isotope of the common carbon-12), which is used in estimating the age of an artifact, has a half-life of about 5730 years, while uranium-235 has a half-life of about 710 million years!

As you can infer from Example 2.1, the rather accurate process of exponential decay for radioactive substances is governed by the equation

$$m_n = m_0(1/2)^n, n = 0, 1, 2, 3, \ldots (\text{n}H \text{ years})$$

For time expressed in years, we can calculate:

$$m_t = m_0(1/2)^{t/H}$$

As applied to the exponential growth in Example 2.1, we can also express a discrete time equation for exponential decay as:

$$N_{t+1} = N_t - kN_t \tag{2.3}$$

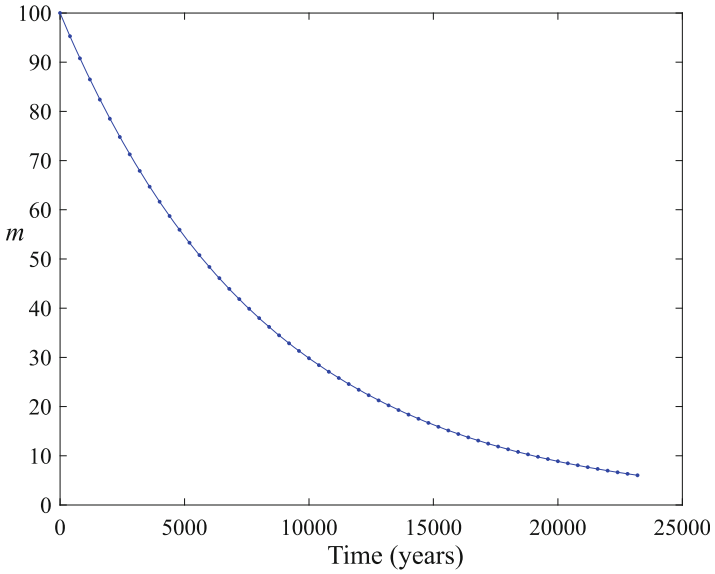


Fig. 2.3 Exponential decay of radioactive carbon-14, starting with $m_0 = 100$ over a time period of four half-lives. Symbols are plotted every 400 years

where the decay constant k is:

$$k = \ln(2)/H$$

We can solve the discrete exponential decay equation (Eq. 2.3) for the case of ^{14}C with a half-life of 5730 years over a time period of four half-lives or 22,920 years (Fig. 2.3).

Example 2.4 Now we consider another example of exponential decay from a biological perspective concerning the recovery of infected individuals in a population. I represents the number of infected individuals. Within a population of infected, 1/4 of them recover per day and let us call this value $\lambda = 0.25$. Therefore, our time unit is one day and our infected recovery model becomes:

$$I_{n+1} = I_n - \lambda I_n$$

We can solve this numerically starting with an initial size of the infected population $I_0 = 100$ (see Fig. 2.4).

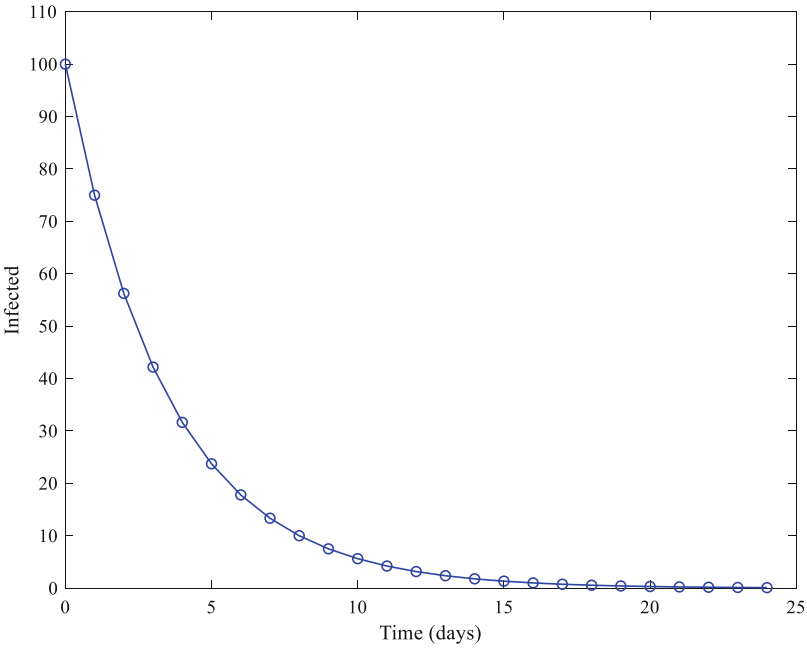


Fig. 2.4 Exponential decay in the size of the infected population

2.3 Summary

We considered pure exponential growth for a population of bacteria in a growth medium assumed to be unlimited for their growth. Rapid growth occurs but without limit and this would be unreasonable for an actual population which would have limited food available. A second example considered growth in a population of bacteria. Although the initial growth was rapid as in the first example, it eventually slowed to approach a constant carrying capacity of the growth medium. A third example considered exponential decay of a radioactive substance whereby the remaining radioactive matter decreases by half after each half-life interval, characteristic of the particular radioactive substance. Lastly, we considered the exponential decay of a population of infected individuals, which also declines rapidly at first to approach zero whereby all of the initially infected individuals have recovered.

2.4 Exercises

2.4.1 For Example 2.1, we considered exponential growth for a population of cells starting with 60 cells. Calculate and plot the population growth starting with only 10 cells and the same value of the growth constant.

2.4.2 Consider the model for growth with a carrying capacity as described by Eq. 2.2. In the earlier text, we choose values for r and K from the bacteria experiment. Solve this model and plot the population size for a doubled value of r to show how an increased growth rate constant would change the timing of the population's approach to its carrying capacity.

2.4.3 Solve and draw a decay plot using Eq. 2.2 for uranium-235 with a half-life of 710 million years. A related calculation was done earlier for carbon-14 (Fig. 2.3).

References

Nagy S (2011) Kinetics of radioactive decay. In: Vértes A, Nagy A, Klencsár Z, Lovas R, Rösch F (eds) Handbook of nuclear chemistry, 2nd edn. Springer Science+Business Media, Dordrecht

Sharon M, Sharon M (2021) Nuclear chemistry, 2nd edn. Springer Nature, Cham

Chapter 3
Discrete Time Models

The previous chapter presented a model of exponential growth. However, we also saw that in a realistic situation of growth of a bacterial culture where measurements were taken at discrete times (hours, in that case), the population did not grow without limit but tended to level off in an S-shape form, called a "logistic curve." What happened here was that, due to the limited amount of food in the medium, the growth of bacteria cells slowed down due to competition for food and, as a consequence, the population also started to grow at a slower pace tending more and more to stabilize in growth. In this chapter, we will present a general model for such growth along with two alternative models.

Mathematically, this process can be described by a so-called "Discrete Logistic Model." That is an equation of the form similar to that expressed in Chap. 2, Eq. 2.2.

$$N_{t+1} = N_t + rN_t\left(1 - \frac{N_t}{K}\right) = N_t + rN_t - \frac{r}{K}N_t^2 \tag{3.1}$$

where r is a reproductive rate constant and K is a carrying capacity of the model. As shown in Chap. 2, an exponential growth model is biologically unrealistic because it shows unlimited growth. As we will see, the discrete logistic model incorporates a carrying capacity which prevents unlimited growth. The term with N_t^2 represents intraspecific competition within the population.

3.1 Solutions of the Discrete Logistic

Therefore Eq. 3.1 is an example of the *Discrete Logistic Model*. A solution, using $r = 1.5$ and $K = 100$, would look like the typical S-shaped logistic curve (Fig. 3.1).

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_3

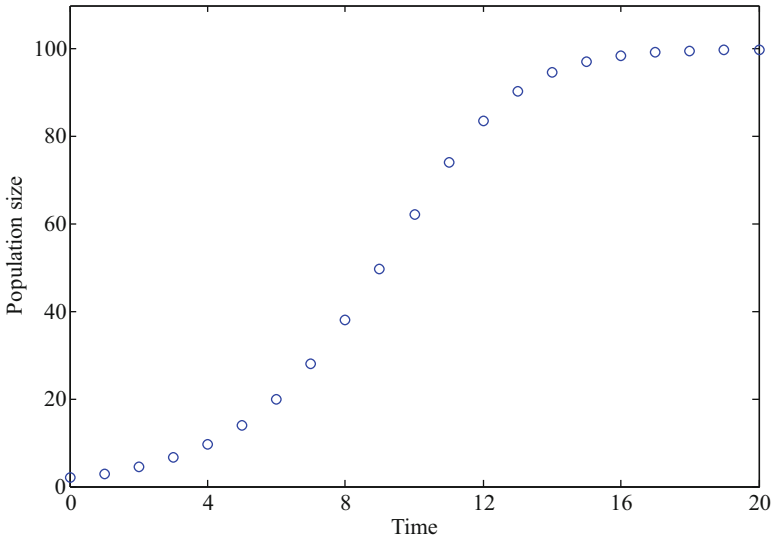


Fig. 3.1 A plot of the solution for the discrete logistic population model with $r = 1.5$ and $K = 100$

3.2 Enhancements to the Discrete Logistic Function

We should note that implementation of the Discrete Logistic Equation runs into a problem, giving a negative value for large values of N. Of course, there are also alternative forms to the Discrete Logistic Equation as suggested by de Vries et al. (2006) that resolve this problem.

Option I One of the alternative models is the so-called *Beverton-Holt model* (Beverton and Holt 1993; Geritz and Kisdi 2004), derived for fisheries, with the Discrete Logistic Eq. 3.1 being replaced by

$$N_{t+1} = \left(\frac{r}{1 + \frac{(r-1)N_t}{K}} \right) N_t \tag{3.2}$$

with $r > 1$ and (of course) $K > 0$. Note $N_{t+1} > 0$ for all t.

Option II Another model is the so-called *Ricker model* (Ricker 1954; Geritz and Kisdi 2004), also derived for fisheries, with the Discrete Logistic Eq. (3.1) being replaced by

$$N_{t+1} = e^{r\left(1 - \frac{N_t}{K}\right)} N_t \tag{3.3}$$

with $r > 1$ and $K > 0$. Again, note that $N_{t+1} > 0$ for all t.

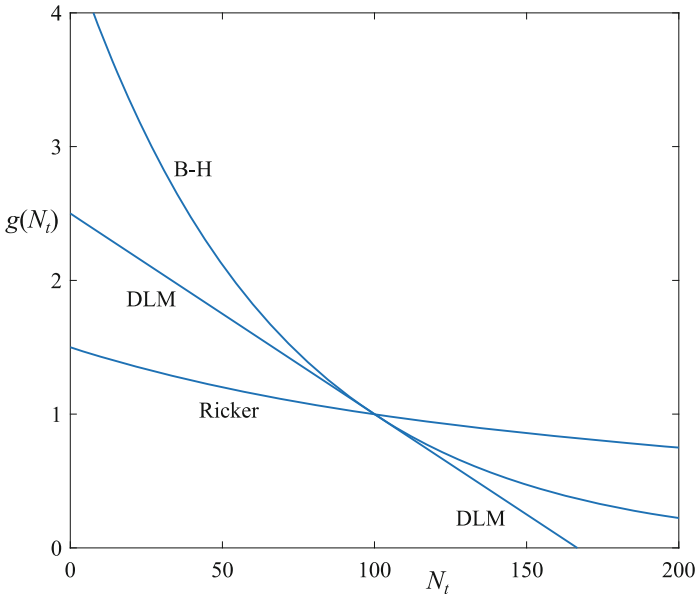


Fig. 3.2 Growth functions ($g(N)$) for the three discrete model options. DLM, discrete logistic model; B-H, Beverton-Holt model; Ricker, the Ricker model. For the results shown here, $r = 1.5$ and $K = 100$

All three of these models can also be expressed in terms of a growth function such that:

$$N_{t+1} = g(N_t)N_t$$

If we look at the manner in which the growth functions $g(N)$ depend on N (Fig. 3.2) we can see how the issue leading to a negative N_{t+1} arises for the discrete logistic but not for the Beverton-Holt and Ricker models. Note in the figure that all three functions meet at $N = K$ (100 in this graph), where the growth function equals 1.0.

Finally, as pointed out in the Preface and Introduction, our upcoming approach will make extensive use of Computer Algebra Software (CAS) like Mathematica and MATLAB to compensate for the minimal requirement of mathematical knowledge at the Freshman level that we are assuming in this course. (Of course, the reader must understand the meaning of the mathematical parameters and terminology being used, as well as be able to properly interpret the results!)

3.3 Summary

The discrete logistic model resolved a problem with the simple exponential growth model presented in Chap. 2 by incorporating a carrying capacity that keeps the population from growing without limit. On the other hand, the discrete logistic model has a weakness such that with sufficiently large values of N, the next calculated population size will be negative. The alternative models such as the Beverton-Holt and Ricker models resolve this weakness by the use of a growth function that does not become negative.

3.4 Exercises

3.4.1 Solve the Beverton-Holt model and present a graph showing the population size versus time when $r = 1.5$ and $K = 50$.

3.4.2 Solve the Ricker model and present a graph showing the population size versus time when $r = 1.5$ and $K = 50$.

3.4.3 Using either the Beverton-Holt or Ricker models, show that for an initial value of the population size (N) greater than K, the population declines to the steady-state K. Solve the model and present your results as a graph of population size versus time.

References

Beverton RJH, Holt SJ (1993) On the dynamics of exploited fish populations. Springer Science + Business Media, Dordrecht

de Vries G, Hillen T, Lewis M, Müller J, Schönfisch B (2006) A course in mathematical biology. Quantitative modeling with mathematical and computational methods. Society for Industrial and Applied Mathematics, Philadelphia

Geritz SAH, Kisdi E (2004) On the mechanistic underpinning of discrete-time population models with complex dynamics. J Theor Biol 228:261–269

Ricker WE (1954) Stock and recruitment. J Fish Res Board Can 11:559–623

Chapter 4
Fixed Points, Stability, and Cobwebbing

In Chap. 3, we studied a few examples of Discrete Time Models. In particular, we derived the important example of the Discrete Logistic Equation shown in Eq. 3.1. In this chapter, we shall consider a general form of a Discrete Time Model (also called a "Discrete Dynamical System" in mathematics). We will introduce methods for determining the fixed points (values of the population size (x) which do not change for future time steps), and the stability of these fixed points.

4.1 Fixed Points and Cobwebbing

A General Discrete Time Model is a discrete time equation of the form:

$$x_{n+1} = f(x_n), \quad n = 0, 1, 2, \ldots \tag{4.1}$$

where $f(x)$ is a given real function together with an initial state x_0. The orbit of Eq. 4.1 is the sequence of points of the form

$$x_1 = f(x_0), \qquad x_2 = f(x_1), \ldots$$

And so on, for all n, starting with $n = 0$. Here are some more definitions.

Definitions

1. An Equilibrium State or a fixed point of $f(x)$ is any $\hat{x}$ satisfying

$$f(\hat{x}) = \hat{x} \tag{4.2}$$

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_4

2. An equilibrium state is said to be stable if any orbit starting sufficiently close to $\hat{x}$ stays close to $\hat{x}$. Otherwise, $\hat{x}$ is said to be unstable. In typical situations if $\hat{x}$ is stable, such orbits will approach $\hat{x}$, while if $\hat{x}$ is unstable, orbits go away from $\hat{x}$ from at least one of its sides.
3. Cobwebbing is a graphical technique used to visualize orbits on the graph of $f(x)$ in order to determine whether a given fixed point is stable or unstable (see Elaydi 1999).

We give a couple of examples below. For the first one, dealing with the biologically meaningful discrete logistic time model considered in Chap. 3, we illustrate Definition 1 by finding the fixed points. In the second example, still dealing with the discrete logistic time model, we illustrate Definitions 2 and 3.

Example 4.1 Consider a simple discrete logistic model of the form:

$$x_{n+1} = a x_n (1 - x_n)$$

Then from

$$f(x) = a x (1 - x)$$

and setting $f(x) = x$, we obtain

$$a x (1 - x) = x$$

which yields the equilibrium states, or fixed points

$$\hat{x}^1 = 0 \text{ and } \hat{x}^2 = 1 - 1/a$$

We can illustrate all three definitions using Mathematica (Fig. 4.1).

Adding the cobwebbing process to Fig. 4.1 shows the stability of the fixed point at $x = 0.6$ (Fig. 4.2). The process is carried out by starting a vertical line at some point along the x-axis and drawing vertically to the $f(x)$ curve. Then continue the line horizontally to the diagonal ($f(x) = x$) line. Repeat that process until it becomes clear if the lines are approaching the fixed point (where the two curves meet). Here (Fig. 4.2) it can be seen that the cobwebbing lines move away from the unstable fixed point at $x = 0$ and toward the stable fixed point at $x = 0.6$. In practice, one should also draw cobwebbing lines from a point on the x-axis greater than 0.6.

Discretelogistic

Create model function

```
In[1]:= f[x_] := a x (1 - x)
```

Set parameters

```
In[2]:= a = 2.5;
```

Find fixed points

```
In[3]:= Solve[f[x] - x == 0, x]
Out[3]= {{x → 0.}, {x → 0.6}}
```

Plot results

```
In[4]:= Plot[{f[x], x}, {x, 0, 0.8}]
```

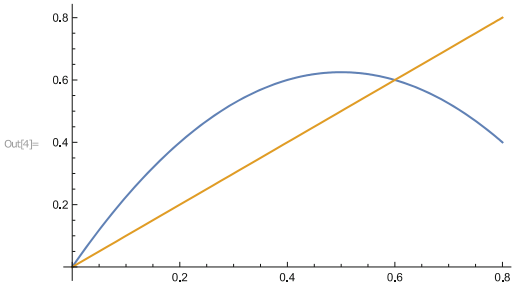


Fig. 4.1 Mathematica notebook showing the finding of fixed points

4.2 Linear Stability Analysis

We can also test stability through Calculus, via the analytical version of the so-called "Linear Stability Analysis" (see Elaydi 1999). Namely, suppose $\hat{x}$ is a fixed point of a given discrete time equation

$$x_{n+1} = f(x_n)$$

where $f(x)$ is a "nice" differentiable function. Then

- If $|f'(\hat{x})|$ is less than 1 (i.e., $-1 < f'(\hat{x}) < 1$) then $\hat{x}$ is stable.
- If $|f'(\hat{x})|$ is greater than 1 (i.e., $f'(\hat{x}) < -1$ or $f'(\hat{x}) > 1$), then $\hat{x}$ is unstable.
- If $|f'(\hat{x})|$ is equal to 1 (i.e., $f'(\hat{x}) = 1$ or $f'(\hat{x}) = -1$), the linear stability analysis is inconclusive and one should use cobwebbing to decide on stability.

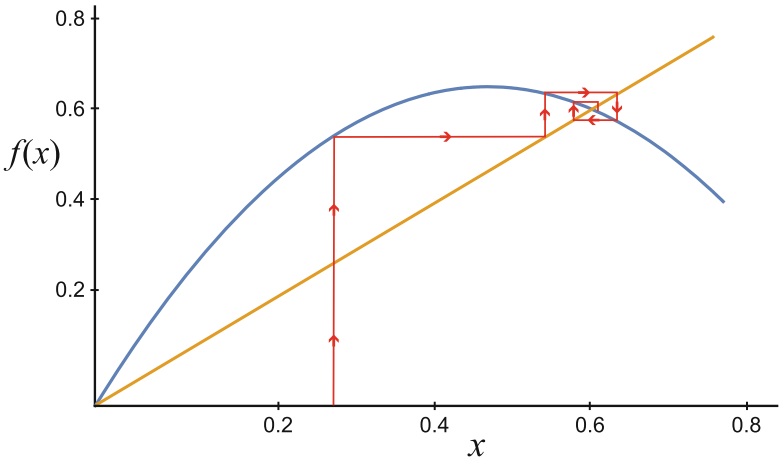


Fig. 4.2 Cobwebbing to show the stability of the fixed point at $x = 0.6$

Example Consider a discrete logistic equation:

$$x_{n+1} = 2\left(x_n - \frac{x_n^2}{2}\right)$$

where $f(x) = 2x - x^2$. Then, solving $f(x) = x$ for x and calculating $f'(x) = 2 - 2x$ and noting that the fixed points are $\hat{x}^1 = 0$ and $\hat{x}^2 = 1$, we calculate that

$$f'(\hat{x}^1) = 2$$
$$f'(\hat{x}^2) = 0$$

And we conclude that $\hat{x}^1$ is unstable and $\hat{x}^2$ is stable.

This can also be shown using Mathematica (Fig. 4.3).

4.3 Summary

In this chapter, we have described the calculation of fixed points for a general discrete time model. Next, we considered a method for determining if those fixed points are stable or unstable using the graphical approach of cobwebbing. A more analytical approach for assessing the stability of fixed points was derived as linear stability analysis based on methods from calculus for finding the derivative of the model function. Two examples were presented using the software Mathematica for finding fixed points and applying the linear stability analysis. Later, we will see examples where the linear stability analysis is inconclusive and the complementary cobwebbing method should be used.

Discretelogistic

Create model function and derivative

```
In[ ]:= f[x_] := 2 x - x^2
In[ ]:= fprime[x_] = D[f[x], x]
Out[ ]=
        2 - 2 x
```

Find fixed points

```
In[ ]:= Solve[f[x] - x == 0, x]
Out[ ]=
        {{x → 0}, {x → 1}}
```

Assess stability of fixed points

```
In[ ]:= fprime[0]
Out[ ]=
        2
```

This fixed point is unstable.

```
In[ ]:= fprime[1]
Out[ ]=
        0
```

This fixed point is stable

Fig. 4.3 Mathematica notebook showing the application of linear stability analysis

4.4 Exercises

4.4.1 Consider a discrete equation:

$$x_{n+1} = 4x_n - x_2^2$$

Find the fixed points and determine their stability using linear stability analysis.

4.4.2 Consider a discrete equation:

$$x_{n+1} = 3x_n - 3x_n^2 + x_n^3$$

Find the fixed points and determine their stability using linear stability analysis.

4.4.3 Consider a discrete equation:

$$x_{n+1} = 1 - x_n$$

Find the fixed points and determine their stability using linear stability analysis or cobwebbing.

4.4.4 Consider this difference equation:

$$x_{n+1} = x_n^2 - x_n + 1$$

You can find a fixed point at $x = 1$. Using linear stability analysis, show that assessing the stability of this fixed point is inconclusive. Next, draw a cobwebbing plot of $f(x)$ versus x. Notice that the fixed point appears to be stable starting at $x < 1$, but unstable starting at $x > 1$. For such a case, the fixed point would be referred to as unstable.

Reference

Elaydi SN (1999) An introduction to difference equations. Springer Science+Business Media, New York

Chapter 5
Population Genetics Models

In Chap. 4, we defined and studied the notions of equilibrium state or fixed point, stability, and cobwebbing, and illustrated them with a few examples. In this chapter, we shall consider a biology application to Population Genetics. The first application will consider wing color in moths in the case where two phenotypes (the outward expression of a trait) are present: moths with white or black wings. Next, we will expand this to a situation where three phenotypes are present, with white, black, or gray wing colors.

First, we need to review some basic terminology in genetics (see e.g. King 2003). Here are some definitions.

"Genes" are the fundamental units of heredity carrying information from one generation to the next. Due to mutations, a gene can exist in a number of different forms, or alleles; in our introductory approach we will only consider genes with 2 alleles A and a. Alleles (one for each parent) do interact to produce a trait, e.g., eye color in humans.

"Chromosomes" are structures in the cell nucleus that carries the genes.

"Diploid Organisms" are those consisting of two sets of chromosomes.

"Genotype" refers to the actual genetic makeup of an individual. The specific alleles present, e.g., AA, Aa, and so forth.

"Phenotype" refers to an outwardly expressed trait of an individual.

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_5

5.1 Two Phenotypes Case

Let us consider the trait wing color in moths. The following modeling approaches were inspired by Tatum (2003). In general, many traits are determined by two alleles. Since we are only considering two alleles, A and a, a particular moth can have one of the following three allele compositions, AA, Aa, or aa. So, if we assume that the moths have one of two phenotypes, "white-winged" or "black-winged," let us say that moths with the allele composition AA or Aa develop white wings and moths with the allele composition aa develop black wings. In other words, we are assuming that the allele A determines the color white-wing regardless of the allele a, and we say that the allele A is dominant and a is recessive. Then, the following questions naturally arise in a given population of moths:

(i) How do the allele frequencies change over the various generations of the population in question?
(ii) Do recessive alleles disappear over time?
(iii) Does a selection mechanism exist?

For this case of two phenotypes, we have two scenarios:

I. No selection occurs. In this scenario, one makes the following assumptions:

- Random mating
- No mutations can occur (so new alleles do not enter the population)
- No gene flow or migration
- No selection occurs
- A large population is present so that genetic drift does not occur

Letting p_n denote the frequency of one allele (say A) at the nth generation ($n = 0$, 1, 2, ...), then one has (as demonstrated below)[1]

$$p_{n+1} = p_n$$

for all n. This is the situation of the so-called **Hardy-Weinberg Law** (see Hardy 1908; Weinberg 1908; Crow 1988). Clearly, in this situation, the answers to (i)–(iii) above are:

(i) Allele frequencies do not change over generations.
(ii) and (iii) No, recessive alleles do not disappear and selection is not considered for this case.

[1] Note the frequency q_n of the allele a satisfies $p_n + q_n = 1$ for all $n = 0, 1, 2, \ldots$ so it would not matter which allele we picked to study its frequency because we can always calculate the frequency for the allele by that formula.

Table 5.1 Punnett Square based on the probabilities for each possible offspring

			Parent 2	A	a
				p_n	$1-p_n$
Parent 1	A	p_n		p_n^2	$p_n(1-p_n)$
	a	$1-p_n$		$p_n(1-p_n)$	$(1-p_n)^2$

Proof of Hardy-Weinberg Law: We will use the so-called Punnett Square,[2] depicted below. The table shows how the frequencies of the alleles A and a of any given generation determine the three different genotypes AA, $Aa = aA$, and aa in the following generation (Table 5.1).

The probability p_{n+1} of getting the allele A in an individual with genotype AA, Aa, or aa is 1, 1/2, or 0, respectively. Thus, the weighted average of probabilities of getting the allele A in generation $n + 1$ is given by

$$p_{n+1} = \frac{1 \cdot p_n^2 + (1/2) \cdot 2p_n(1-p_n) + 0 \cdot (1-p_n)^2}{p_n^2 + 2p_n(1-p_n) + (1-p_n)^2} = \frac{p_n}{(p_n + (1-p_n))^2} \quad (5.1)$$
$$= p_n$$

II. Selection is occurring

In our previous case (I), recall that we consider the trait wing color in moths with the phenotype white-winged moths corresponding to the allele composition AA or Aa, and the phenotype black-winged moths corresponding to the allele composition aa. Now, with selection present, this means that parameters α, β with respective values $0 \le \alpha \le 1$, $0 \le \beta \le 1$ are assigned representing the fractions of white-winged moths and black-winged moths, respectively, that survive to produce each next generation. Say, a value of $\alpha = 0.4$ less than $\beta = 0.8$ would mean that white-winged moths would have a 40% chance of surviving as opposed to black-winged moths having an 80% chance of surviving from each generation to the next. Perhaps because white-winged moths are more visible to predators than black-winged moths. In this case, we write $\alpha < \beta$ and say that black-winged moths have a selective advantage over the white-winged moths. Then, similarly to the Hardy-Weinberg situation, the weighted average of probabilities of getting the allele A in generation $n + 1$, considering also the selective pressures α and β, will now be given by

$$\begin{aligned} p_{n+1} &= \frac{1 \cdot \alpha p_n^2 + (1/2) \cdot 2\alpha p_n(1-p_n) + 0 \cdot \beta(1-p_n)^2}{\alpha p_n^2 + 2\alpha p_n(1-p_n) + \beta(1-p_n)^2} \\ &= \frac{\alpha p_n}{\alpha p_n^2 + 2\alpha p_n - 2\alpha p_n^2 - \beta\left(1 - 2p_n + p_n^2\right)} \\ &= \frac{\alpha p_n}{(\beta - \alpha)p_n^2 - 2(\beta - \alpha)p_n + \beta} \end{aligned} \quad (5.2)$$

[2] See e.g. King (2003).

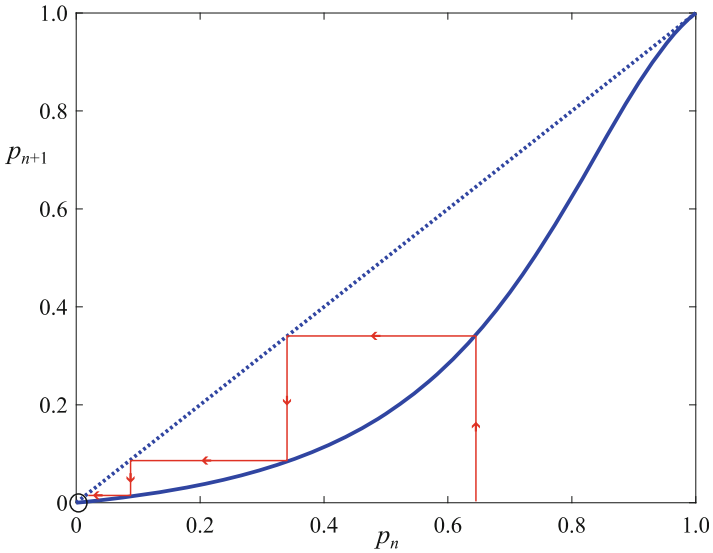


Fig. 5.1 Cobwebbing for the scenario with $\alpha = 0.1$ and $\beta = 0.8$, showing that $p_n = 0$ is a stable steady-state as indicated by the circle at that point

Using Eq. 5.2, we will consider two scenarios, giving one moth phenotype the advantage over the other.

Scenario 1 ($\alpha < \beta$) using $\alpha = 0.1$ and $\beta = 0.8$ (Fig. 5.1). Black-wing moths have the selective advantage.

Scenario 2 ($\alpha > \beta$) using $\alpha = 0.8$ and $\beta = 0.1$ (Fig. 5.2). White-wing moths have the selective advantage

5.2 Three Phenotypes Case

Let us continue to consider the trait wing color in moths, but now with the three genotypes *AA*, *Aa*, and *aa* giving rise here to three phenotypes. Let us suppose that genotype *AA* develops white-wing moths under selective pressure $0 \leq \alpha \leq 1$, genotype *aa* develops black-wing moths under selective pressure $0 \leq \beta \leq 1$, and genotype *Aa* develops gray-wing moths under selective pressure $0 \leq \gamma \leq 1$. In this situation, similarly to the case of 2 phenotypes, after a little bit of algebra, we arrive at the following expression for p_{n+1} in terms of p_n:

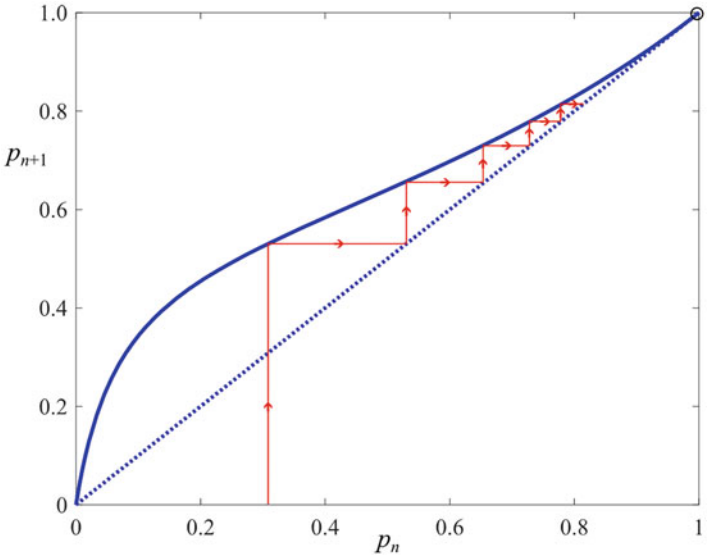


Fig. 5.2 Cobwebbing for the scenario with $\alpha = 0.8$ and $\beta = 0.1$, showing that $p_n = 1$ is a stable steady-state as indicated by the circle at that point

$$p_{n+1} = \frac{1 \cdot \alpha p_n^2 + (1/2) \cdot 2\gamma p_n(1-p_n) + 0 \cdot \beta(1-p_n)^2}{\alpha p_n^2 + 2\gamma p_n(1-p_n) + \beta(1-p_n)^2}$$
$$p_{n+1} = \frac{(\alpha-\gamma)p_n^2 + \gamma p_n}{(\alpha - 2\gamma + \beta)p_n^2 + 2(\gamma-\beta)p_n + \beta} \tag{5.3}$$

We are going to consider 4 different scenarios depending on the relative selective advantage for the three phenotypes, hence the strengths of α, β, and γ.

Scenario 1 ($\alpha > \gamma > \beta$), using the relative strength parameter values $\alpha = 0.8$, $\gamma = 0.5$, and $\beta = 0.1$ (Fig. 5.3). White-wing moths have the selective advantage. Regardless of the initial frequency of the allele A, the frequency of A tends to one and only white-wing moths eventually survive.

Scenario 2 ($\alpha < \gamma < \beta$) using the relative strength parameter values $\alpha = 0.1$, $\gamma = 0.5$, and $\beta = 0.9$ (Fig. 5.4). Black-wing moths have the selective advantage and regardless of the initial frequency of the allele A, the frequency of A tends to zero and only black-wing moths eventually survive.

Scenario 3 ($\gamma > \alpha, \beta$) using the relative strength parameter values $\alpha = 0.1$, $\gamma = 0.5$, and $\beta = 0.2$ (Fig. 5.5). Gray-wing moths have the selective advantage and regardless of the initial frequency of the allele A, the frequency of A tends to an intermediate value and all 3 phenotypes of moths eventually survive and coexist.

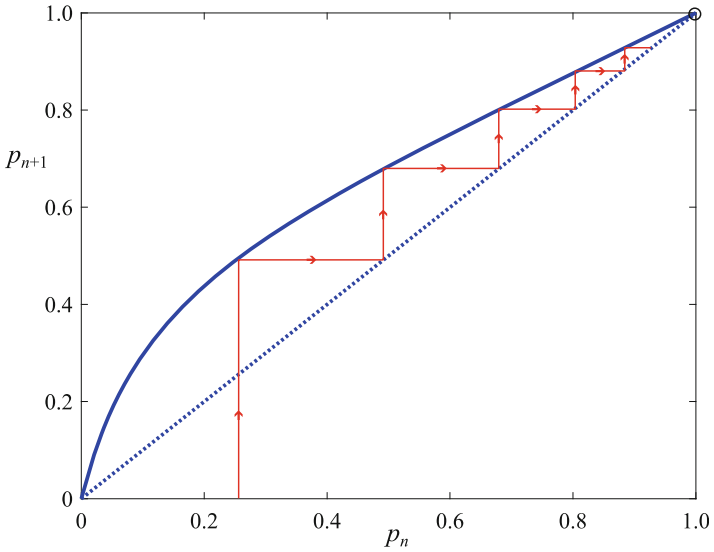


Fig. 5.3 Cobwebbing for the scenario with $\alpha = 0.8$, $\gamma = 0.5$, and $\beta = 0.1$, showing that $p_n = 1$ is a stable steady-state as indicated by the circle at that point

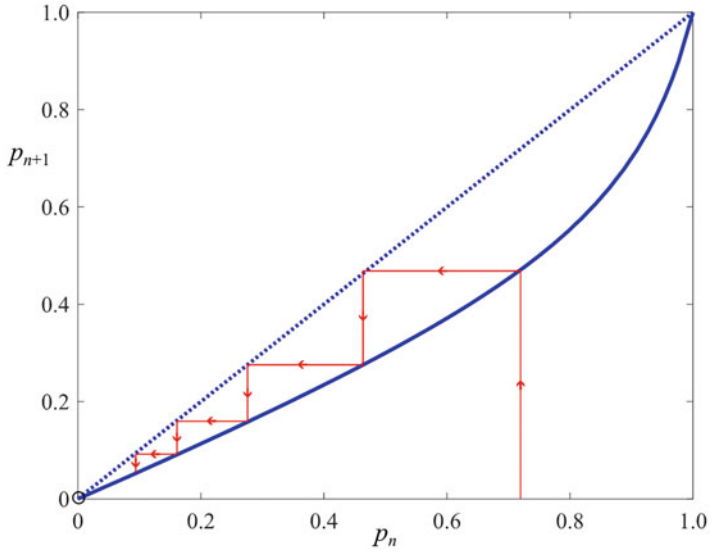


Fig. 5.4 Cobwebbing for the scenario with $\alpha = 0.1$, $\gamma = 0.5$, and $\beta = 0.9$, showing that $p_n = 0$ is a stable steady-state as indicated by the circle at that point

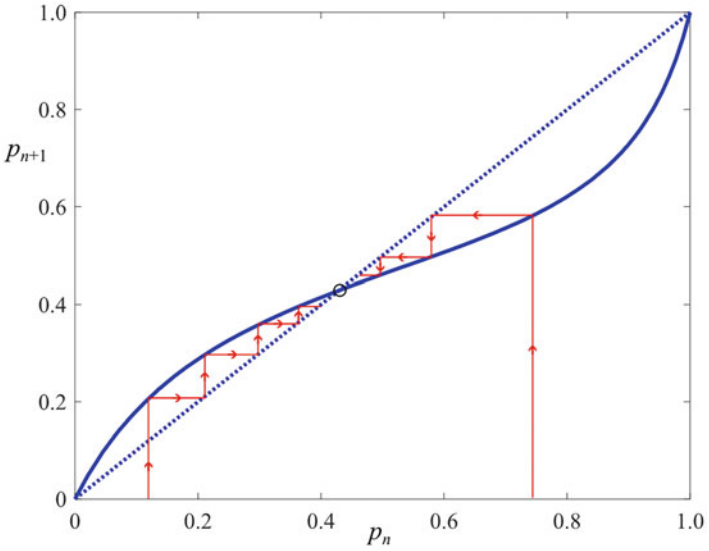


Fig. 5.5 Cobwebbing for the scenario with $\alpha = 0.1$, $\gamma = 0.5$, and $\beta = 0.2$, showing a stable steady-state at an intermediate value of p_n, as indicated by the circle at that point

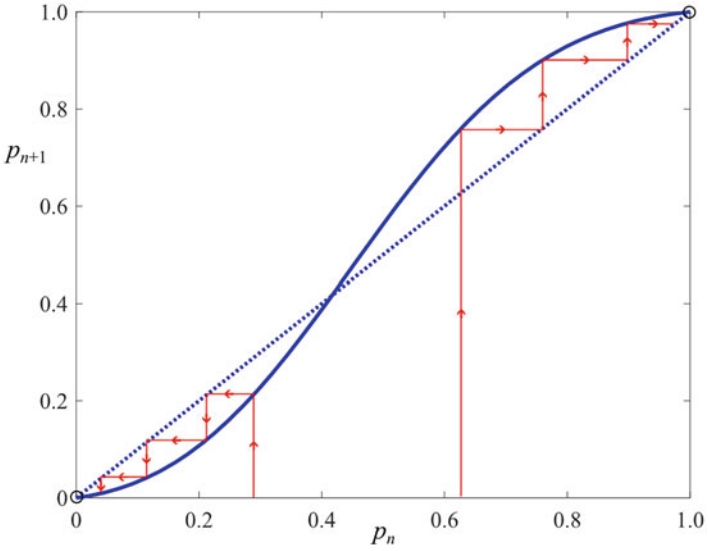


Fig. 5.6 Cobwebbing for the scenario with $\alpha = 0.8$, $\gamma = 0.1$, and $\beta = 0.6$, showing that $p_n = 0$ and $p_n = 1$ are both stable steady-states as indicated by the circles at those points

Scenario 4 ($\gamma < \alpha, \beta$) using the relative strength parameter values $\alpha = 0.8$, $\gamma = 0.1$, and $\beta = 0.6$ (Fig. 5.6). Gray-wing moths have the selective disadvantage and depending on the initial frequency of the allele A, either the frequency of A will approach zero, or the frequency of allele a will approach zero. All moths will either be white-wing (AA), or black-wing (aa).

5.3 Summary

When considering the case of two phenotypes (white or black wing color), the model included selection factors that determined the selective advantage for each phenotype. The phenotype having the greater selective advantage led to a situation where only that one respective moth wing color was present in the population after some period of time. For the three phenotypes case various selective advantage factors were tried. Two scenarios led to only one phenotype eventually being present. Another scenario led to the coexistence of all three phenotypes. Lastly, if the gray-winged moths had a selective disadvantage, the population either became all white-winged moths or all black-winged moths depending on the initial value of the allele frequencies.

5.4 Exercises

5.4.1 Refer to the Mathematica application in Fig. 4.1 of Chap. 4. For this exercise, you would modify the notebook, changing the function to the one expressed in Eq. 5.2. Solve the two phenotypes case based on the values of the selection factors used in Figs. 5.1 and 5.2. Your notebook would create plots like that shown in Fig. 4.1 that could be used for cobwebbing.

5.4.2 Refer to the Mathematica application in Fig. 4.1 of Chap. 4. For this exercise, you would modify the notebook, changing the function to the one expressed in Eq. 5.3. Solve the three phenotypes case based on the values of the selection factors used in Fig. 5.5, leading to the coexistence of all three phenotypes. Your notebook would create a plot like that shown in Fig. 5.5 that could be used for cobwebbing.

5.4.3 Using any of the software tools we have introduced, develop a solution for the two phenotypes case that will allow you to create a plot of p_n versus n.

References

Crow JF (1988) Eight years ago: the beginnings of population genetics. Genetics 119:473–476

Hardy GH (1908) Mendelian proportions in a mixed population. Science 28:49–50

King RM (2003) Biology made simple – a complete introduction to the science of life – from single cell to human anatomy. Broadway Books, New York

Tatum J (2003) Maths and moths. π in the Sky, September 5–9

Weinberg W (1908) Über den Nachweis der Vererbung beim Menschen. Jahreshefte des Vereins für Varterländische Naturkdunde in Württemberg 64:369–382

Chapter 6
Chaotic Systems

We have described the discrete logistic population model in Chap. 3. Here we will offer a slightly modified version that, under some conditions, will show very unusual behavior. You will see that chaotic systems can have an extreme sensitivity to initial values and to the values of the parameters.

6.1 Robert May's Model

This model was developed by Robert May, a theoretical ecologist, who provided an introduction with papers in the mid-1970s (May 1975, 1976). The model equation is deceptively simple:

$$N_{t+1} = aN_t(1 - N_t) \tag{6.1}$$

For this model, the population size (N) is expressed as a fraction of the population carrying capacity, and so it is typically a quantity between 0 and 1. The value of a must be between 0 and 4 but we leave it to Exercise 6.5.1 to show this is true.

6.2 Solving the Model

May's model could even be solved with just a calculator, but let's consider a computer application. A simple Python program (Fig. 6.1) could be used or a closely related MATLAB program could also be used (Fig. 6.2).

For small values of the parameter a, the solution looks similar to that described earlier for the discrete logistic model. But see what happens as the value of a is increased (Fig. 6.3). As you can see, for intermediate values of a, the solution oscillates between two values. For larger values of a, the solution oscillates in a

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_6

```
# May's Chaotic system population model
import numpy as np
import matplotlib.pyplot as plt

a = 1.5
steps = 15
N = np.zeros(steps+1)
t = np.zeros(steps+1)
N[0] = 0.05
t[0] = 0.0
for i in range(0, steps):
    N[i+1] = a*N[i]*(1 - N[i])
    t[i+1] = t[i] + 1.0
print (N)

# plot results
plt.plot(t,N)
plt.xlabel('Time')
plt.ylabel('N')
plt.savefig("MaysChaos.svg")
plt.show()
```

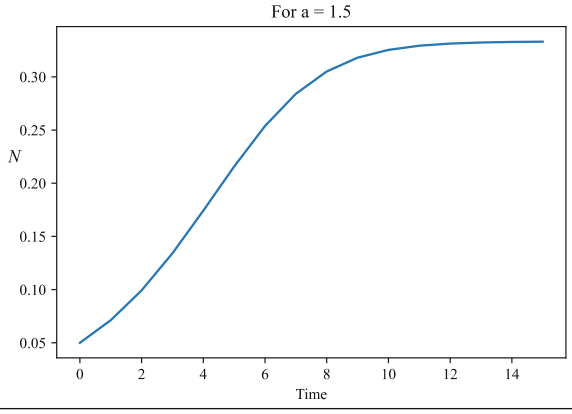


Fig. 6.1 A Python program for solving May's model and the graph of the solution as drawn by the program

manner that appears to show no particular pattern. Even more useful is to compare solutions using this larger value of a and with small changes in the initial starting value for the population size (Fig. 6.4). Although these small changes in the initial population size are not visible as separate curves initially, eventually the populations oscillate with completely different patterns. This extreme sensitivity to initial conditions is the classic hallmark of a chaotic system. Indeed this sensitivity is behind our inability to accurately predict weather beyond a time frame of several days, as noted by Edward Lorenz (see Lorenz 1963, 1995). As an aside, the term "butterfly effect" is often used in describing this high sensitivity to initial conditions and may have its source in a short story by the science fiction writer Ray Bradbury entitled A Sound of Thunder as published in 1952.

```
% Script to run the solution
steps = 20;
N = zeros(1,steps);
t = zeros(1,steps);
N(1) = 0.05;
for i = 1:1:steps
    N(i+1) = maysfunc(N(i));
    t(i+1) = t(i) + 1;
end
plot(t,N);
title('Mays model solution for a = 1.5')
xlabel('Time')
ylabel('N')


function [ nnew ] = maysfunc( n )
a = 1.5;
nnew = a*n*(1-n);
end
```

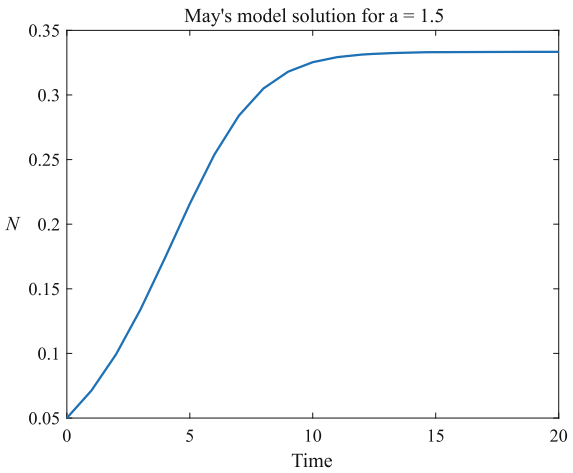


Fig. 6.2 MATLAB approach for solving May's population model

6.3 Model Fixed Points

We discussed the idea of fixed points in relation to discrete equations earlier. May's model equation has a fixed point that may be determined analytically as follows:

$$N = aN(1-N)$$
$$1 = a(1-N)$$
$$1/a = 1-N$$
$$N = 1 - 1/a$$

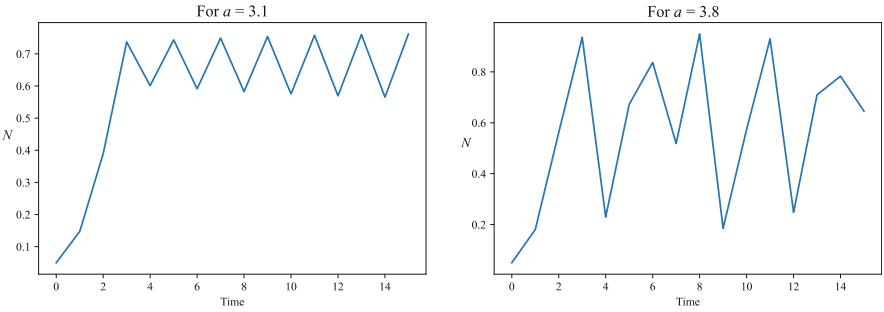


Fig. 6.3 Solutions of May's model for larger values of the parameter a, with $N_0 = 0.05$

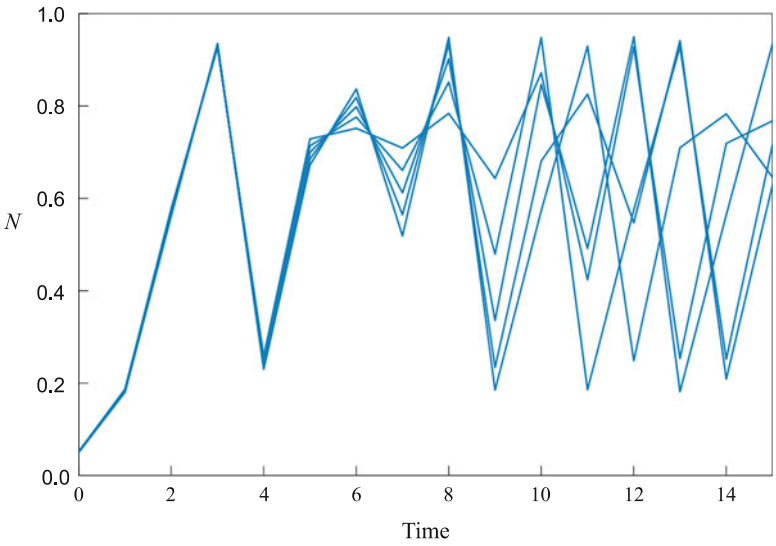


Fig. 6.4 Five solutions of May's model with each having a 1% change in the initial population size and using $a = 3.8$

We can examine the stability of this fixed point (note that $N = 0$ is also a fixed point) with Mathematica (Fig. 6.5). Note that in the first graph, even though the initial population size was close to the fixed point, the solution eventually deviates and begins oscillating again. Why is this? For $a = 3.8$, using the above expression, the fixed point is

$$N = 1 - 1/3.8$$
$$N = 1 - 10/38$$
$$N = 38/38 - 10/38 = 28/38$$

May's Chaos and fixed points

```
Clear[a];
```

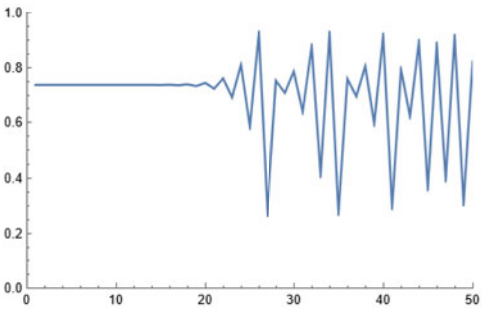
Find fixed points for May's model

```
Solve[a*n*(1 - n) - n == 0, n]
```

$$\left\{\{n \to 0\}, \left\{n \to \frac{-1+a}{a}\right\}\right\}$$

Consider the model behavior for a = 3.8

```
a = 38 / 10;
rtable = RecurrenceTable[{n[t + 1] == a*n[t]*(1 - n[t]), n[0] == 0.736842}, n, {t, 0, 50}];
ListLinePlot[rtable, PlotRange → {{0, 50}, {0, 1}}]
```



The solution becomes chaotic

What if the initial N is given as the fraction 28/38 instead of a close approximation as in the above case?

```
rtable = RecurrenceTable[{n[t + 1] == a*n[t]*(1 - n[t]), n[0] == 28/38}, n, {t, 0, 30}]
```

$$\left\{\frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19},\right.$$
$$\left.\frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}, \frac{14}{19}\right\}$$

Fig. 6.5 Mathematica notebook application of May's population model

This quantity is a rational number, but when expressed in a floating point variable, cannot be perfectly represented with a limited number of digits. A little work with a calculator that can consider 32 digits gives a value of 0.736842105263157894736... which repeats after the 18th digit to the right of the decimal. A typical Python (or C-language) program with variables of type float or double can only carry about 16 digits. Therefore, it should not be surprising that the solution in the first part of the Mathematica notebook (Fig. 6.5) begins to oscillate again because we cannot perfectly represent this value of the fixed point. However, notice the second part of the notebook where the initial value is left as a fraction 28/38. Mathematica has the interesting feature whereby if you enter a value as a fraction, the value is kept in that form and not converted to the floating point form, thus maintaining its accuracy for subsequent calculations. As you can see in the notebook for this case, the solution stays at this initial value. With reference to our earlier discussion of fixed points and stability, we would say that in this case 28/38 is a fixed point but an unstable fixed point because even a very slight deviation from that value of N leads the solution away from the fixed point. Hopefully, you see this as another outcome of the high sensitivity of this chaotic system to initial conditions.

6.4 Summary

Robert May's model shows very different behaviors depending on the value of the parameter a. Small values a such as between 1 and 2 appear similar to the discrete logistic described in Chap. 3. However for values such as 3.1, the population oscillates between two values. Larger values of a such as 3.8 lead to chaotic oscillations in N. For these large values of a, the solution is also very sensitive to small changes in the initial value for N. These behaviors are classic characteristics of chaotic systems.

6.5 Exercises

6.5.1 For Eq. 6.1 to make sense, the values of N should be between 0 and 1. First, find the value of N between 0 and 1 such that $N(1 - N)$ has a maximum value. You will find that maximum value to be ¼ corresponding to $N = ½$. Since $aN(1 - N)$ must be always less than 1, show that a must be less than 4.

6.5.2 Solve May's model for the value of $a = 4.1$, showing what eventually happens to N.

6.5.3 Notice the example in Fig. 6.4 showing the model's sensitivity to the initial value. Create a similar approach showing solutions for small changes in the value of a, starting at 3.8.

References

Lorenz EN (1963) Deterministic nonperiodic flow. J Atmos Sci 20:130–141
Lorenz EN (1995) The essence of chaos. University of Washington Press, Seattle
May RM (1975) Biological populations obeying difference equations: stable points, stable cycles, and chaos. J Theor Biol 51:511–524
May RM (1976) Simple mathematical models with very complicated dynamics. Nature 261:459–467

Chapter 7
Continuous Time Models

The exponential growth of a culture of bacteria in a limited growth medium would not continue indefinitely without bounds due to the limitation of food in the medium, which practically speaking would lead to an eventual leveling of the number of bacteria. In this chapter, we will derive a Continuous Time Model for the phenomenon of logistic growth. Next, we will develop an approach, called phase-line analysis, to find steady states and assess their stability in continuous time models. Lastly, we will extend this logistic model to consider the inclusion of a harvesting component (such as in fisheries) to the equation, with three possible forms.

7.1 The Continuous Logistic Equation

Recall that in Chap. 2 we introduced examples of discrete exponential growth situations through experiments on growth of bacteria in a food medium at various discrete time intervals (say, in hours). Such an experiment would lead over time to a discrete set of points in an S-shaped curve called a discrete logistic curve. In Chap. 3, we introduced the notion of a discrete time model of the form

$$N_{t+1} = N_t + rN_t\left(1 - \frac{N_t}{K}\right)$$

as a more realistic model than the purely discrete exponential growth model.

In this chapter, we will consider a continuous version of the discrete logistic equation called the Continuous Logistic Equation, given by a differential equation of the form

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_7

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right). \tag{7.1}$$

Here, N (population size or density, the dependent variable) is a function of t (time, the independent variable), with $r > 0$ and $K > 0$ being given parameters.

7.2 Equilibrium States and Their Stability

An autonomous differential equation is one of the form (see e.g. Boyce and DiPrima 1992)

$$\frac{dy}{dt} = f(y)$$

such that the independent variable t does not appear explicitly on the right-hand side of the equation. In this case, it is very easy to determine its equilibrium or steady-states and their stability from the graph of $f(y)$ representing dy/dt versus y, a so-called phase-line analysis.

For example, let us apply phase-line analysis to the continuous logistic Eq. 7.1 as shown in Fig. 7.1.

The points of intersection of the curve with the N-axis (in this case, $N = 0$ and $N = 1$) represent the steady states of the continuous logistic equation. For regions of the graph where the curve $f(N)$ representing dN/dt is positive, we draw arrows on the horizontal axis pointing to the right because N would be increasing. On the other hand, for regions where the curve $f(N)$ is negative, we draw arrows pointing to the

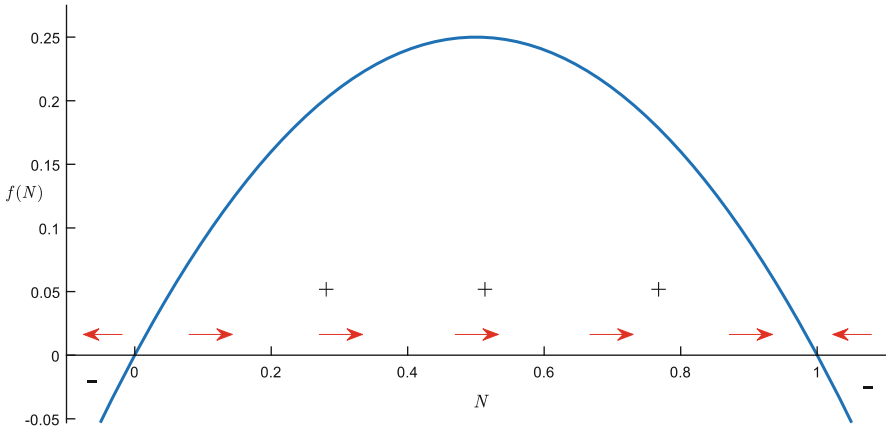


Fig. 7.1 Phase-line analysis applied to the continuous logistic equation. For this example, $r = K = 1$

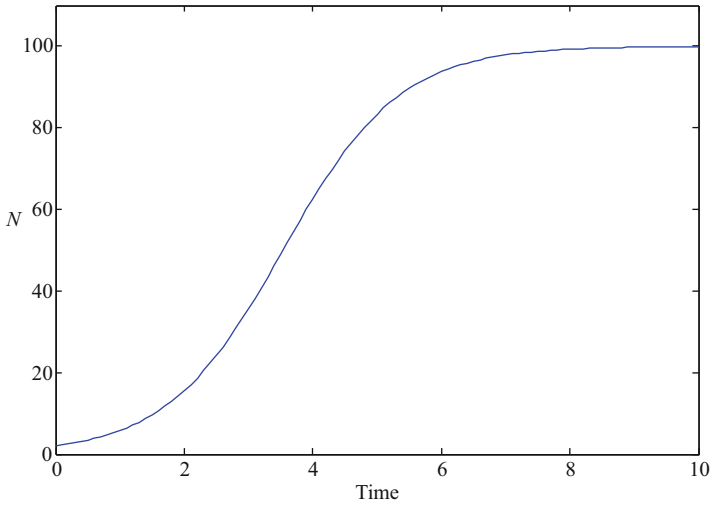


Fig. 7.2 The solution of the continuous logistic equation for $r = 1.1$, $K = 100$

left because N would be decreasing. Observing the region near the steady-states, if the arrows appear to converge on the steady-state, we conclude that the given steady-state is stable. If the arrows point away at least on one side of the steady-state, we conclude that the given steady-state is unstable.

Perhaps one would be interested in visualizing the trajectory of the population over time in a graph of $N(t)$ versus t. For simplicity, we illustrate that in the case of the logistic equation

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right)$$

with $r = 1.1$ and $K = 100$ and starting with a small N_0 much less than K. The solution would describe a logistic curve (Fig. 7.2).

One can solve this ODE for the above case using a Python program with the ODE solver "solve_ivp" as shown here:

```
# Logistic growth model
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# n - population size
def LogisticModel(t, n):
  r = 1.1
  k = 100
  return r*n*(1-n/k)
```

```
n0 = 2.0
t0 = 0
tf = 10
steps = 10*tf
sol = solve_ivp(LogisticModel, [t0, tf], [n0], method='RK45',
 dense_output=True)
t = np.linspace(t0,tf,steps+1)
n = sol.sol(t)

# Display data
#print (np.transpose([t,n[0]]))

# plot results
plt.rcParams['font.family']='Times New Roman'
plt.plot(t, n[0])
plt.xlabel('Time')
plt.ylabel('Population size')
plt.savefig('Logistic.svg')

# Save results to a file
f = open('Logistic.txt', "w")
for i in range(0, steps+1, 1):
  print("%5.1f, %6.2f" % (t[i], n[0][i]), file=f)
f.close()
```

The solution could also be obtained using the ODE solvers in MATLAB (see Appendix 4) or Mathematica (see Appendix 3)

7.3 Continuous Logistic Equation with Harvesting

In this section, we will introduce some models for the study of fisheries and hunting (see e.g. Giordano et al. 2003). This will be done by adding a negative term to our continuous logistic Eq. 7.1 representing the amount of harvesting (fishing or hunting) taken from the current population $N(t)$ per unit of time.

(i) The simplest situation corresponds to constant harvesting. Clearly, this is not very realistic, since it does not take into account the number of organisms in the population at each time, $N(t)$, and can lead to extinction when the existing population $N(t)$ gets smaller than H.

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right) - H \tag{7.2}$$

(ii) A better situation corresponds to harvesting at a fixed fraction of the current population $N(t)$ at each time (e.g., $0.5N$, or $0.3N$, etc.), which is governed by the equation

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right) - HN \tag{7.3}$$

This way, because the harvesting is set to be a fixed proportion of the population $N(t)$ at each time t, one can show that we avoid having extinction of any given population.

(iii) An even better situation happens when harvesting is a function that can vary with $N(t)$ but is limited depending on fishing ability

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right) - \frac{aN}{b+N} \tag{7.4}$$

where a and b are positive constants. If N is small relative to b, then harvesting will be proportional to N as in case (ii). However, if N is much greater than b, then harvesting approaches the constant a. This situation would apply if the harvesters have only a limited capability to harvest, a not unreasonable situation.

Now let's determine the stability of the new two steady states in harvesting case (i) (Fig. 7.3). The steady-state $N = 0.113$ is unstable because the population N is going away of this steady-state on its left as well as on its right. The steady-state $N = 0.887$ is stable because the population N is going toward this steady-state both from the left and from the right. Notice that this form of harvesting decreases the

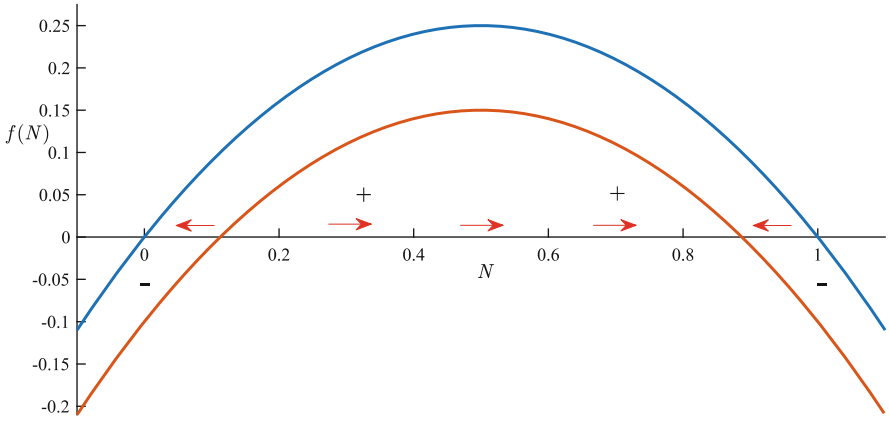


Fig. 7.3 Phase-line analysis applied to the continuous logistic equation with constant harvesting as in case (i) with $r = K = 1$ and $H = 0.1$. For comparison, the upper curve shows the case without harvesting as depicted in Fig. 7.1. The lower curve shows the case with harvesting and here the arrows apply to this case

stable steady-state from the value without harvesting while also increasing the value of the unstable steady-state. We leave the analysis of the other two cases to the exercises below.

7.4 Summary

In this chapter, we have presented a continuous time logistic model for population growth that includes a carrying capacity term to prevent the population from growing without limit. The phase-line analysis method was developed for finding steady-states of an ordinary differential equation model and to assess their stability. Three model additions were presented to consider various approaches to modeling harvesting. In the first case with harvesting expressed as a constant rate independent of population size, the stable steady-state population size is reduced in comparison to the steady-state population without harvesting.

7.5 Exercises

7.5.1 Consider the harvesting case (i) with H constant, independent of N (Fig. 7.3). Calculate the maximum harvesting rate to insure the population does not go extinct.

7.5.2 Case (ii) is described by Eq. 7.3, where the harvesting rate is proportional to the population size N. Create a plot corresponding to this case (similar to Fig. 7.3) for a value of $H = 0.5$

7.5.3 Case (iii) is described by Eq. 7.4. If N is small, then harvesting will be proportional to N as in case (ii). However, if N is much greater than b, then harvesting approaches the constant a. Create a plot corresponding to this case (similar to Fig. 7.3) for $a = 10$ and $b = 50$.

References

Boyce WE, DiPrima RC (1992) Elementary differential equations and boundary value problems, 5th edn. Wiley, New York

Giordano FR, Weir MD, Fox WP (2003) A first course in mathematical modeling, 3rd edn. Brooks/Cole Publishing, Pacific Grove

Chapter 8
Organism-Organism Interaction Models

8.1 Interaction Models Introduction

The models introduced in Chap. 7 considered a single group of organisms. Here, we will expand this into groups of two organisms that interact in some manner. The nature of the interactions can take on several forms (see Beltrami 1987; Murray 2002). Organisms can compete for resources, they can involve one group that is a predator and the other a prey, or they may have an interaction that is beneficial to both groups, termed a mutualism. First, let's consider how we will model the process of interactions. How many interactions occur in a given time?

We will take an approximation of the number of interactions from the Law of Mass Action, which is often used to describe chemical reactions. The rate of such a reaction can be taken as proportional to the product of the concentrations of the two chemicals. We will apply this concept to interactions between groups of organisms, where the rate of interaction will be proportional to the product of the number of organisms in each group. In a general form, a model of the two groups would be

$$\begin{aligned} \frac{dN_1}{dt} &= \alpha N_1 + \beta N_1 N_2 \\ \frac{dN_2}{dt} &= \gamma N_2 + \delta N_1 N_2 \end{aligned} \tag{8.1}$$

where N_1 and N_2 are the sizes of population 1 and population 2 (N_1 and $N_2 > 0$). The constants α, β, γ, and δ can be positive or negative in this general case. It is useful to think of the biological meanings of the sign on those constants. If α is positive, its term will cause population N_1 to grow, whereas if α is negative, its term will cause population N_1 to decline. In fact if we ignore the interaction term with the product $N_1 N_2$, the model equation describes exponential growth or decay. The previous state-

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_8

ment for α would also apply to γ in the equation for N_2. Now if β is positive in the interaction term for population N_1, we would be describing an interaction that tends to increase N_1. Likewise, if δ is positive in the interaction term for population N_2, we would be describing an interaction that tends to increase N_2. On the other hand, if β is negative in the interaction term for population N_1, we would be describing an interaction that tends to decrease N_1. Likewise, if δ is negative in the interaction term for population N_2, we would be describing an interaction that tends to decrease N_2.

In the remainder of this chapter, we will consider models will three combinations of the sign issues described above for the constants α, β, γ, and δ. Instead of using these Greek letters for the constants, we will use a, b, c, and d as positive constants so that we can explicitly show the sign of each term in the equation. This will become apparent in each case below.

8.2 Competition

For a competition model, the interaction terms for both organism groups will be negative to reflect the reduction in population size that may occur due to the interaction. Note that if we also used a negative term for the population growth (a negative α and γ in Eq. 8.1), the combination of negative growth and competitive interaction would lead to both populations declining to zero: so we will not consider that example here. Our competition model would be expressed as

$$\begin{aligned} \frac{dN_1}{dt} &= aN_1 - bN_1N_2 \\ \frac{dN_2}{dt} &= cN_2 - dN_1N_2 \end{aligned} \tag{8.2}$$

with $a, b, c, d > 0$.

We can analyze this model through phase-plane analysis, a process extending the phase-line analysis for a single ODE used in Chap. 7. Instead of identifying steady-state points as in phase-line analysis, we find lines upon which each population does not change: these are called nullclines or isoclines.

For population N_1, the two nullclines are found as follows:

$$\begin{aligned} &\frac{dN_1}{dt} = 0 = aN_1 - bN_1N_2 \\ &N_1 = 0 \quad \text{and} \quad N_2 = \frac{aN_1}{bN_1} = \frac{a}{b} \end{aligned} \tag{8.3}$$

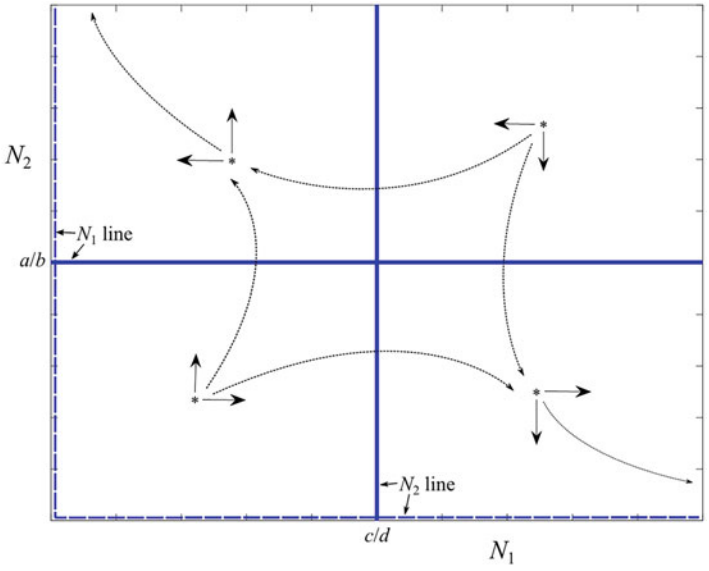


Fig. 8.1 Phase-plane diagram for the competition model

For population N_2, the two nullclines are found as follows:

$$\frac{dN_2}{dt} = 0 = cN_2 - dN_1N_2$$
$$N_2 = 0 \quad \text{and} \quad N_1 = \frac{cN_2}{dN_2} = \frac{c}{d} \tag{8.4}$$

The above nullclines are then plotted in a graph (Fig. 8.1). We then identify points in each of the four regions in the graph and determine the directions each population would move if the solution were located at each of the four points. For example in the right half region (Fig. 8.1), the vertical arrows should reflect how N_2 would change in this region, Here, N_1 is greater than c/d and from the nullclines equations (Eq. 8.4), dN_2/dt would be negative, hence the arrows point downward. From this approach, the vertical arrows on the left half would point upward because N_1 is less than c/d. Horizontal arrows indicate how N_1 would change in each region. For the upper half, N_2 is greater than a/b and so from the nullclines equations (Eq. 8.3), dN_1/dt would be negative, hence the arrows point to the left. Arrows in the lower half would thus point to the right.

For this competition model, it appears that either population N_1 or N_2 might win. However, the winning population grows without limit and this would be a weakness of this model that we will address in Chap. 10.

8.3 Predator-Prey

For a predator-prey model, the interaction terms for the predator (P) will be positive to indicate that the predation event positively affects the predator group and the interaction term for the prey group (N) will be negative to reflect the reduction in population size that would occur due to predation. Our predator-prey model would be expressed as

$$\begin{aligned} \frac{dP}{dt} &= -aP + bPN \\ \frac{dN}{dt} &= cN - dPN \end{aligned} \tag{8.5}$$

with $a, b, c, d > 0$. Note that if we used a positive term for the predator population growth, the combination of positive growth and positive interaction would lead to the predator population growing without limit. So we will not consider that example here. Similarly, if we used a negative term for the prey population growth, the combination of negative growth and negative interaction would lead to the prey population declining to zero. Such a scenario can be solved, but we just consider it not interesting from a biological perspective.

For the predator population, the two nullclines are found as follows:

$$\begin{aligned} &\frac{dP}{dt} = 0 = -aP + bPN \\ &P = 0 \quad \text{and} \quad N = \frac{aP}{bP} = \frac{a}{b} \end{aligned}$$

For the prey population, the two nullclines are found as follows:

$$\begin{aligned} &\frac{dN}{dt} = 0 = cN - dPN \\ &N = 0 \quad \text{and} \quad P = \frac{cN}{dN} = \frac{c}{d} \end{aligned}$$

The above nullclines are then plotted in a graph (Fig. 8.2). We then identify points in each of the four regions in the figure and determine the directions each population would move if located at each of the four points.

For this predator-prey model, the two populations oscillate in a steady manner. The predation interaction results in a decline in the prey population when the predators are at large numbers (the right half of Fig. 8.2). On the other hand, a small predator population allows the prey population to grow (left half of Fig. 8.2). If the prey population is small, the predator population declines for lack of food (bottom half of Fig. 8.2). Of course if the prey population is large, the predator population will grow because of a good food supply (upper half of Fig. 8.2). Note that a weakness of this model is apparent if the predator population is zero, the prey population (see Eq. 8.5, if $P = 0$, the prey equation becomes a simple exponential growth model) grows without limit.

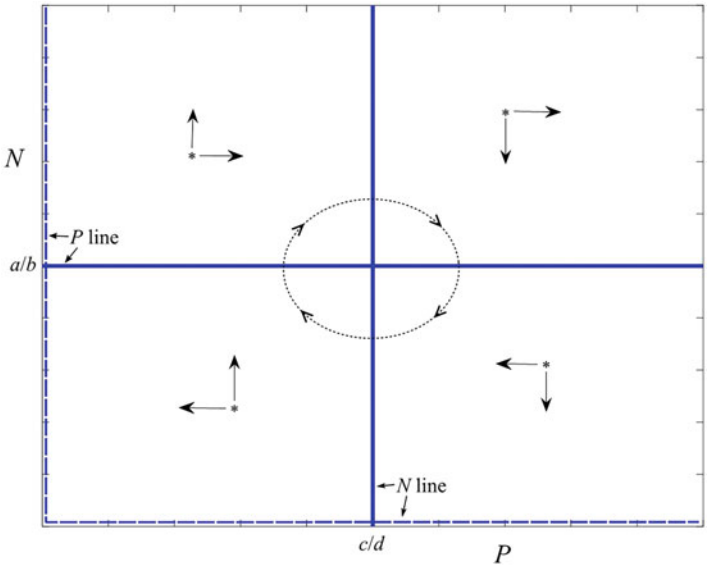


Fig. 8.2 Phase-plane diagram for the predator-prey model

8.4 Mutualism

For a mutualism model, the interaction terms for both organism groups will be positive to reflect the increase in population size that may occur due to the interaction. Our mutualism model would be expressed as

$$\begin{aligned} \frac{dN_1}{dt} &= -aN_1 + bN_1N_2 \\ \frac{dN_2}{dt} &= -cN_2 + dN_1N_2 \end{aligned} \tag{8.6}$$

with $a, b, c, d > 0$. Note that if we also used a positive term for the population growth (a positive α and γ in Eq. 8.1), the combination of positive growth and positive interaction would lead to both populations to grow without limit, so we will not consider that example here.

For the N_1 population, the two nullclines are found as follows:

$$\frac{dN_1}{dt} = 0 = -aN_1 + bN_1N_2$$

$$N_1 = 0 \quad \text{and} \quad N_2 = \frac{aN_1}{bN_1} = \frac{a}{b}$$

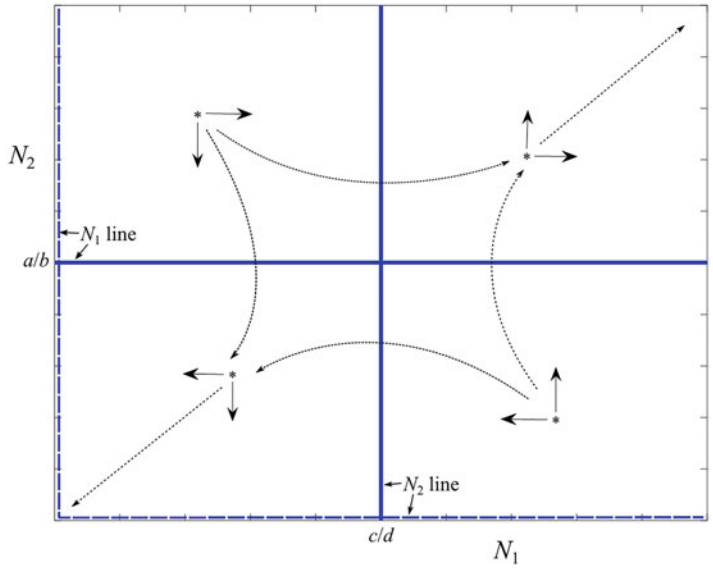


Fig. 8.3 Phase-plane diagram for the mutualism model

For the N_2 population, the two nullclines are found as follows:

$$\frac{dN_2}{dt} = 0 = -cN_2 - dN_1N_2$$

$$N_2 = 0 \quad \text{and} \quad N_1 = \frac{cN_1}{dN_1} = \frac{c}{d}$$

The above nullclines are then plotted in a graph (Fig. 8.3). We then identify points in each of the four regions in the figure and determine the directions each population would move if located at each of the four points.For this model with the two populations having a mutualistic interaction, large numbers of both populations lead to growth of both populations (upper right of Fig. 8.3). Whereas small numbers of both populations lead to a decline in both populations because of the reduced mutual interaction (lower left of Fig. 8.3). The steady-state with both populations at zero is a stable steady-state in this case. A weakness of this model can be seen if both populations are at large numbers, because they will both grow without limit.

8.5 Summary

Here, we developed a system of two ODEs that could be used as a general form for interactions between two populations of organisms. Phase-plane analysis was introduced as a method for finding steady-states and assessing their stability. Three forms of interaction models were considered. Competition between the two populations led

to one species winning while the other goes to zero, but the winning species grew without limit. A predator-prey interaction was considered next, leading to the two populations oscillating. Lastly, mutualism, an interaction that benefits both populations, was treated in which case both populations approach zero or grow without limit. These weaknesses will be addressed in Chap. 10.

8.6 Exercises

8.6.1 Consider the case of competition (Eq. 8.2). Use one of the ODE solvers (see Appendices for the tutorials) found in either Mathematica, MATLAB, or Python to develop a numerical solution of this ODE system giving N_1 and N_2 as functions of time. Use these values for the parameters and initial conditions: a = 0.5, b = 0.01, c = 0.5, d = 0.025, $N_1(0) = 25$, $N_2(0) = 60$. Run the solution for around 10 time steps. Draw a plot of the two population sizes versus time. Your results should make sense based on the earlier phase-plane diagram (Fig. 8.1).

8.6.2 Consider the case of predator-prey interaction (Eq. 8.5). Use one of the ODE solvers (see Appendices for the tutorials) found in either Mathematica, MATLAB, or Python to develop a numerical solution of this ODE system giving N_1 and N_2 as functions of time. Use these values for the parameters and initial conditions: a = 0.5, b = 0.01, c = 0.5, d = 0.03, $P_0 = 5$, and $N_0 = 10$. Run the solution for around 50 time steps. Draw a plot of the two population sizes versus time and also a plot of predator versus prey population sizes showing the orbits that would develop. Your results should make sense based on the earlier phase-plane diagram (Fig. 8.2).

8.6.3 Consider the case of mutualism (Eq. 8.6). Use one of the ODE solvers (see Appendices for the tutorials) found in either Mathematica, MATLAB, or Python to develop a numerical solution of this ODE system giving N_1 and N_2 as functions of time. Use these values for the parameters: a = 0.5, b = 0.01, c = 0.5, and d = 0.025. Run your program for two cases: (i) $N_1(0) = 25$, $N_2(0) = 20$ and (ii) $N_1(0) = 25$, $N_2(0) = 75$. Run the solutions for around 20 time steps. Draw a plot for each case of the population sizes versus time. Your results should make sense based on the earlier phase-plane diagram (Fig. 8.3).

References

Beltrami E (1987) Mathematics for dynamic modeling. Academic, New York
Murray JD (2002) Mathematical biology. I. An introduction. Springer, Berlin

Chapter 9
Host-Parasitoid Models

We can also develop models for the interaction between hosts and parasitoid insects. We will see these are related to our previous predator-prey models in Chap. 8, although the host-parasitoid model presented here is for a discrete system.

9.1 Beddington Model

Our equation system was developed by Beddington et al. (1975):

$$\begin{aligned} H_{t+1} &= H_t e^{r(1-H_t/K)-aP_t} \\ P_{t+1} &= \alpha H_t\left(1-e^{-aP_t}\right) \end{aligned} \tag{9.1}$$

where H is the host population and P is the parasitoid population. The parameter K is the carrying capacity for the host insects, r is the prey reproductive rate, and the positive parameters a and α are controlling the interaction. Note that Beddington et al. (1975) described this as an improvement over the earlier Nicholson-Bailey model (Nicholson and Bailey 1935), where the host population could grow as density-independent without the presence of the parasitoid (Hassell and May 1973). In the Beddington model (Eq. 9.1), the host population includes an exponential term for its growth relative to a carrying capacity (K).

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_9

9.2 Some Solutions of the Beddington Model

Case 1
We can examine some possible behaviors of this model by choosing values for the parameters of the model (Eq. 9.1), in this case $a = 0.005$, $\alpha = 3.0$, $r = 1.1$, $K = 200$, and solve using a MATLAB program shown in Sect. 9.3 of this chapter.

In this scenario (Fig. 9.1), the two populations oscillate for some time but eventually the oscillations dampen and they appear to be stabilizing at constant values of both populations.

For an alternative view of the solution, we can plot a graph with the parasitoid population against the host population (Fig. 9.2). The population oscillations now appear as orbits, which gradually collapse because of their dampening as shown in the previous plot (Fig. 9.1).

Case 2
Here, we increased the value of K by 25% (from 200 to 250) and kept the other parameter values the same.

In this scenario (Fig. 9.3), the host and parasitoid populations oscillate steadily, although there is a curious slight variation in the maximum values of the parasitoid population with each oscillation.

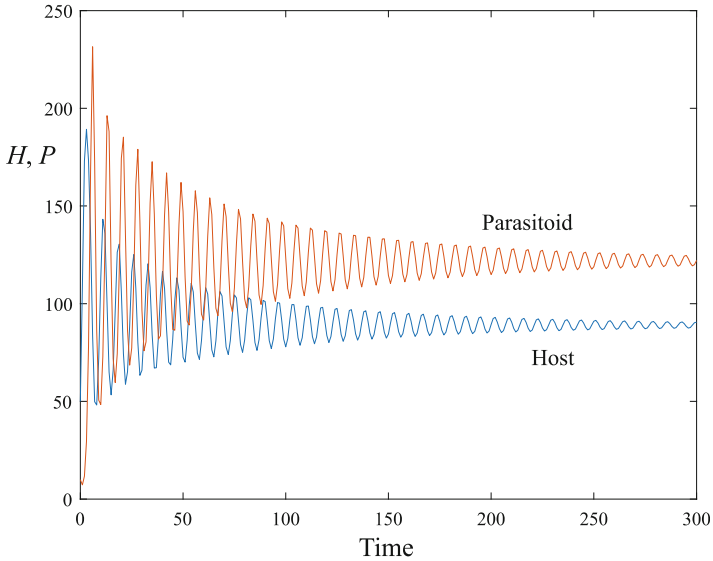


Fig. 9.1 Beddington model solution for Case 1

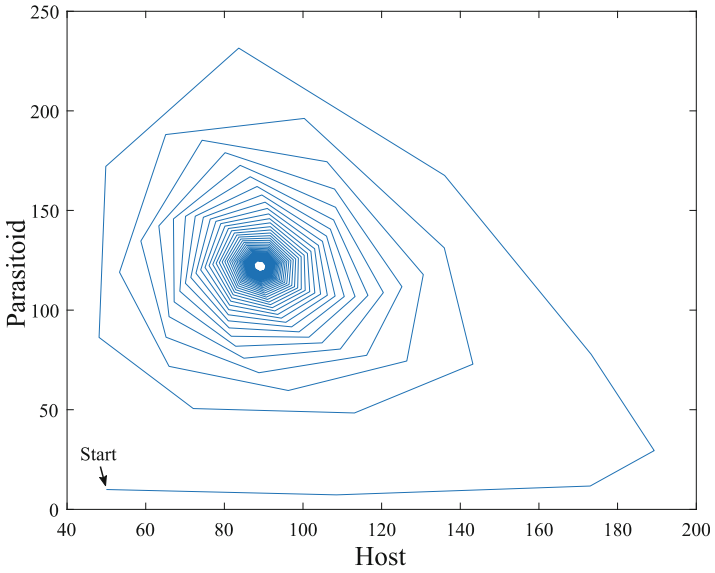


Fig. 9.2 Alternative plot of the solution for Case 1. The initial values for the two populations are indicated with "Start" and the population sizes move in a counter-clockwise fashion for the advancing time values

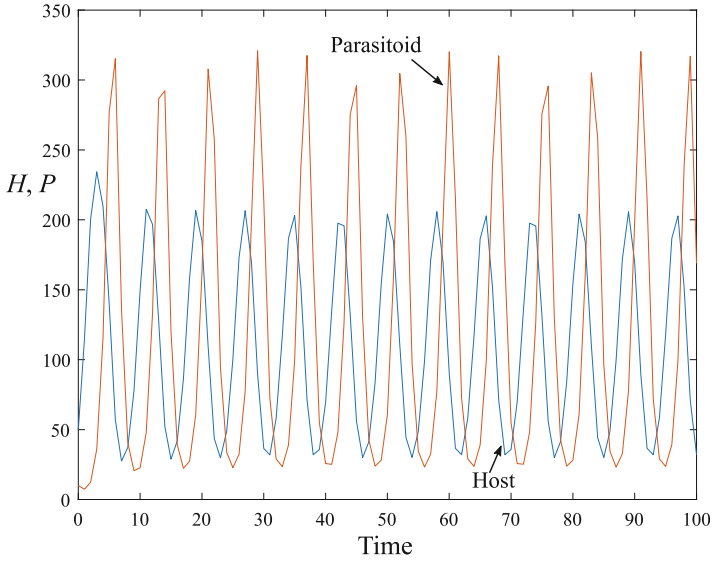


Fig. 9.3 Beddington model solution for Case 2

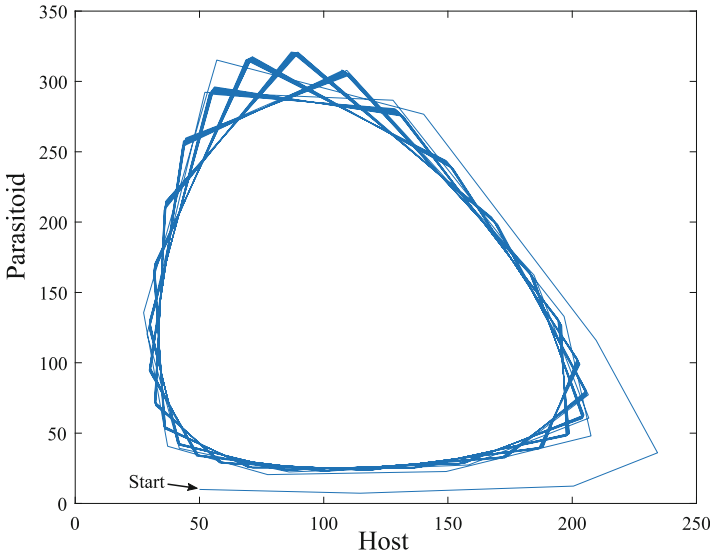


Fig. 9.4 Alternative plot of the solution for Case 2. The initial values for the two populations are indicated with "Start" and the population sizes move in a counter-clockwise fashion for the advancing time values

We can also plot the solution with the parasitoid population against the host population (Fig. 9.4). The population oscillations still arrear as orbits, but they do not collapse reflecting the continued steady oscillations as shown in the previous plot (Fig. 9.3).

9.3 MATLAB Solution for the Host-Parasitoid Model

One method for solving these host-parasitoid models uses MATLAB. The function for the host and parasitoid populations could be set up as:

```
function [h] = Host(h, p)
%HOST population
global alpha a r K;
h = h*exp(r*(1-h/K)-a*p);
end

function [p] = Parasitoid(h, p)
    %PARASITOID function
    global alpha a r K;
    p = alpha * h * (1- exp(-a*p));
end
```

A script was used to solve the model and draw the plot:

```
%  Host-parasitoid model solution
clear H P T
global alpha a r K;
a = 0.005;
alpha = 3;
r = 1.1;
K = 200;
H(1) = 50;   % Set the pop initial sizes
P(1) = 10;
T(1) = 0;
for i = 1:300   % Calculate pop sizes for 300 time steps
    H(i+1) = Host(H(i), P(i));
    P(i+1) = Parasitoid(H(i), P(i));
    T(i+1) = T(i) + 1;
end
plot(T, H, T, P)   % Draw graph with both populations versus time
```

For the above MATLAB objects, the model parameters were declared as "global" so they could be shared simply among all three components.

9.4 Python Solution for the Host-Parasitoid Model

Python could also be used to solve this model using the program shown here. The plot generated will show the two populations with time on the horizontal axis.

```
# Host-parasitoid model

import numpy as np
import matplotlib.pyplot as plt
import math

# Set parameter values
a = 0.005
alpha = 3.0
r = 1.1
K = 200
steps = 300
h = np.zeros(steps+1)
p = np.zeros(steps+1)
t = np.zeros(steps+1)

#Set initial values
h[0] = 50
p[0] = 10
t[0] = 0

for i in range(0,steps):
  h[i+1] = h[i] * math.exp(r*(1-h[i]/K)-a*p[i])
  p[i+1] = alpha * h[i] * (1-math.exp(-a*p[i]))
  t[i+1] = t[i] + 1
```

```
# plot results
plt.plot(t,h, label = "Host")
plt.plot(t,p, label = "Parasitoid")
plt.xlabel('Time')
plt.ylabel('h(t), p(t)')
plt.legend()
plt.show()
```

9.5 Summary

The basic Beddington model was expressed for describing host-parasitoid interactions. As we saw for our predator-prey models in Chap. 8, the host and parasitoid insect populations can show oscillatory behavior. In some cases the oscillations dampen and the populations approach stable values. However, for other cases, the two populations may continue to oscillate without dampening.

9.6 Exercises

9.6.1 The Beddington host-parasitoid model, like many models for interacting populations, can display a range of behaviors depending on the values of model parameters. Case 1 shown above displays populations that initially oscillate, but gradually show an approach to stable values. Try out one of the solution approaches (MATLAB or Python) to show how doubling the value of a for the host-parasitoid interaction results in undamped oscillations of the populations.

9.6.2 The Beddington host-parasitoid model, like many models for interacting populations, can display a range of behaviors depending on the values of model parameters. Case 1 shown above displays populations that initially oscillate, but gradually show an approach to stable values. Try out one of the solution approaches (MATLAB or Python) to show how decreasing the value of K for the host population has a strong effect on the time it takes for the populations to stabilize.

References

Beddington JR, Free CA, Lawton JH (1975) Dynamic complexity in predator-prey models framed in difference equations. Nature 255:58–60

Hassell MP, May RM (1973) Stability in insect host-parasite models. J Anim Ecol 42:693–726

Nicholson AJ, Bailey VA (1935) The balance of animal populations – part I. Proc Zool Soc London 105:551–598

Chapter 10
Competition Models with Logistic Term

Recall from the competition model in Chap. 8 that the form used for the model lead to one population declining to zero while the other grew without limit, a decidedly unrealistic result from a biological perspective. In a model of exponential growth of a single population, we could approach the similar problem of unlimited growth by adding a term for competition within the population (intraspecific), that term including a carrying capacity. Here, we will try a similar approach (see Beltrami 1987; Murray 2002).

10.1 Addition of Logistic Term to Competition Models

The extended model is expressed as:

$$\begin{aligned} \frac{dN_1}{dt} &= a_1 N_1 \left(\frac{K_1 - N_1 - \alpha N_2}{K_1} \right) \\ \frac{dN_2}{dt} &= a_2 N_2 \left(\frac{K_2 - N_2 - \beta N_1}{K_2} \right) \end{aligned} \tag{10.1}$$

Note that we can rewrite the above equations as:

$$\begin{aligned} \frac{dN_1}{dt} &= aN_1 - bN_1N_1 - cN_1N_2 \\ \frac{dN_2}{dt} &= dN_2 - gN_2N_2 - hN_1N_2 \end{aligned} \tag{10.2}$$

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_10

These forms are equivalent if the coefficients in Eq. 10.2 are related to the coefficients in Eq. 10.1 as follows:

$$a = a_1 \quad b = \frac{a_1}{K_1} \quad c = \frac{a_1\alpha}{K_1}$$
$$d = a_2 \quad g = \frac{a_2}{K_2} \quad h = \frac{a_2\beta}{K_2}$$

However, with the form in Eq. 10.2, we can perhaps readily see that the first interaction terms (with N_1N_1 and N_2N_2) represent an intraspecific competition (within each population) and the second interaction terms (with N_1N_2) represent an interspecific competition (between the two populations).

Next, we calculate the nullclines for this system.

For N_1:

$$\frac{dN_1}{dt} = 0 = a_1N_1\left(\frac{K_1 - N_1 - \alpha N_2}{K_1}\right)$$
$$N_1 = 0 \quad \text{and} \quad K_1 - N_1 - \alpha N_2 = 0$$

For N_2:

$$\frac{dN_2}{dt} = 0 = a_2N_2\left(\frac{K_2 - N_2 - \beta N_1}{K_2}\right)$$
$$N_2 = 0 \quad \text{and} \quad K_2 - N_2 - \beta N_1 = 0$$

Note that for this system, the nullclines are not necessarily vertical or horizontal lines given by constants as was the case in Chap. 8. We will plot these lines by finding the intercepts on the two axes. For example for the nullclines of N_1, the intercept on the horizontal axis (when $N_2 = 0$) is K_1 and the intercept on the vertical axis (when $N_1 = 0$) is K_1/α. Similarly for the nullclines of N_2, the intercept on the vertical axis (when $N_1 = 0$) is K_2 and the intercept on the horizontal axis (when $N_2 = 0$) is K_2/β.

We will consider three cases based on the relative values of the constants in our system of equations.

Case (i): For this scenario, we choose values for the model parameters such that $K_1/\alpha > K_2$ and $K_2/\beta < K_1$ (Fig. 10.1).

In this case, N_1 wins as indicated by the circle at K_1, which is a stable steady-state.

Case (ii): For this scenario, we choose values for the model parameters such that $K_1/\alpha > K_2$ and $K_1 < K_2/\beta$.

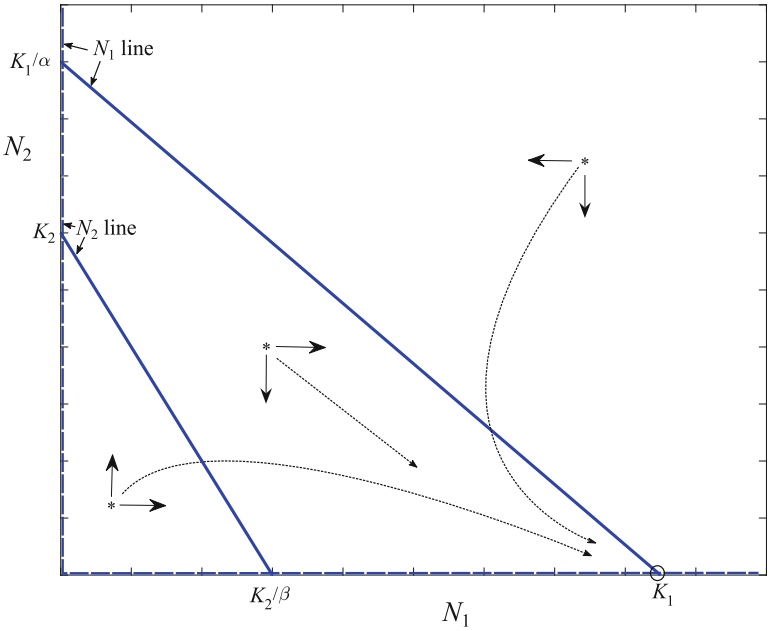


Fig. 10.1 Phase-plane diagram for Case (i). Arrows show trajectories from the starting points indicated by the asterisk symbols

In this case, both species coexist at intermediate sizes. The circle (Fig. 10.2) indicates the stable steady-state.

Case (iii): For this scenario, we choose values for the model parameters such that $K_2 > K_1/\alpha$ and $K_2/\beta < K_1$ (Fig. 10.3).

In this case, one species wins but which species is the winner is not clear from the start. The steady-state where the two nullclines cross in the middle of the figure is an unstable steady-state. The two steady-states with either N_1 winning at K_1 *or* N_2 winning at K_2 are stable.

10.2 Predator-Prey-Prey Three Species Model

We have been considering models of two interacting groups of populations. There is no reason that we have to be limited to only two groups. For one extension of these approaches for interacting populations, let's consider a model of two prey organisms that interact competitively and one predator population that can prey upon the other two. Hopefully, you will see an interesting and meaningful biological result from this model.

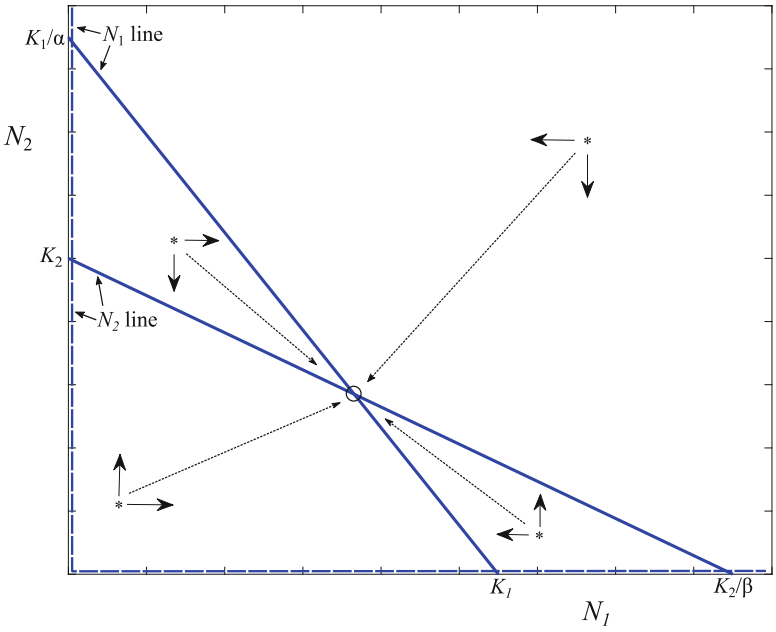


Fig. 10.2 Phase-plane diagram for Case (ii). Arrows show trajectories from the starting points indicated by the asterisk symbols

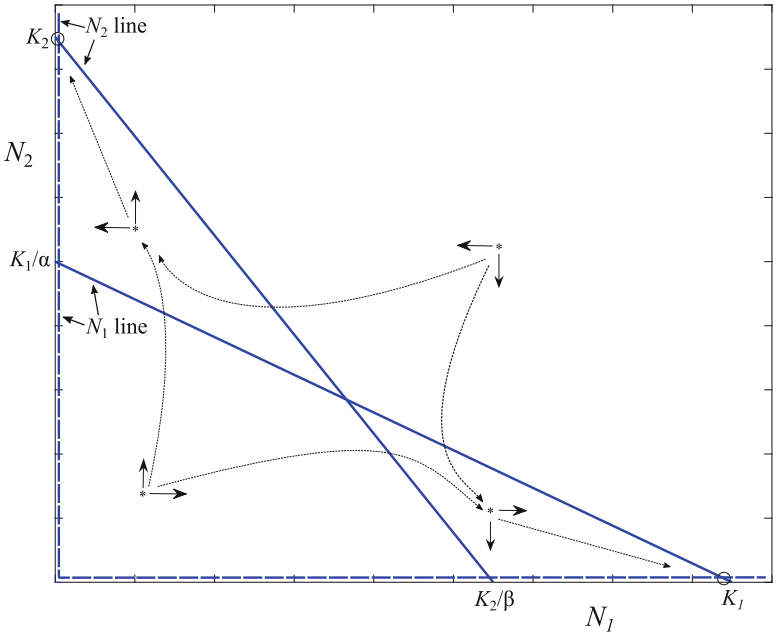


Fig. 10.3 Phase-plane diagram for Case (iii). Arrows show trajectories from the starting points indicated by the asterisk symbols

$$\frac{dN_1}{dt} = a_1 N_1 \left(\frac{k_1 - N_1 - \alpha N_2}{k_1} \right) - p_1 N_1 P$$
$$\frac{dN_2}{dt} = a_2 N_2 \left(\frac{k_2 - N_2 - \beta N_1}{k_2} \right) - p_2 N_2 P$$
$$\frac{dP}{dt} = \gamma_1 p_1 N_1 P + \gamma_2 p_2 N_2 P - mP$$

The first two equations are for the prey organisms (N_1 and N_2) that compete with each other, including intra- and inter-specific competition terms. The third organism is the predator (P) that can consume either prey organism. The prey equations also include a term for predation, with p_1 and p_2 as predation rate coefficients. The predator equation includes terms for predation of the two prey species. The terms have constants (γ) that reflect the idea that typically more than one predation event ($1/\gamma$) is needed to generate one more predator. Lastly, predators die at a rate described by the last term in the predator equation.

This model is more complicated than our previous interaction models. It becomes difficult to use the visual approaches exemplified by phase-lines and phase-planes. We would need a 3D phase volume approach. Instead, we will focus on using our computer software to develop actual solutions to the system.

10.3 Predator-Prey-Prey Model Solutions

Let's try using MATLAB to solve this ODE system (see the tutorial in Appendix 4). The equation system is contained in the MATLAB function:

```
function yprime = predpreyprey(t,y)
a1 = 0.2;
K1 = 150;
alpha = 0.5;
p1 = 0.008;
a2 = 0.4;
K2 = 100;
beta = 0.75;
p2 = 0.002;
gamma1 = 0.2;
gamma2 = 0.2;
m = 0.1;
yprime = [a1*y(1)*(K1-y(1)-alpha*y(2))/K1 - p1*y(1)*y(3);...
          a2*y(2)*(K2-y(2)-beta*y(1))/K2 - p2*y(2)*y(3);...
          gamma1*p1*y(1)*y(3) + gamma2*p2*y(2)*y(3) - m*y(3)];
end
```

The model would be solved using these MATLAB statements:

```
>> tspan = [0 100];
>> yzero = [10; 10; 10];
>> [t y] = ode45(@predpreyprey, tspan, yzero)
>> plot(t,y)
```

Note that the above statement for the initial values (yzero = ...) allows the model to run with all three organisms. To leave out the predators, use this statement instead:

```
>> yzero = [10; 10; 0];
```

Model solutions without the predator

When the predator is not present in the model, the two competing prey organisms show a behavior similar to case (i) earlier in this chapter, where one group of organisms wins and the other goes to zero (Figs. 10.1 and 10.4).

Model solutions with the predator present

In this scenario the predator is present at nonzero initial numbers (Fig. 10.5). The three populations oscillate at first before stabilizing. Ultimately all three organisms coexist. This result suggests that under some circumstances, the presence of a predator in an ecosystem might allow competing organisms to coexist, even if they are at smaller numbers than the single winning species for the case where a predator was not present (Fig. 10.4).

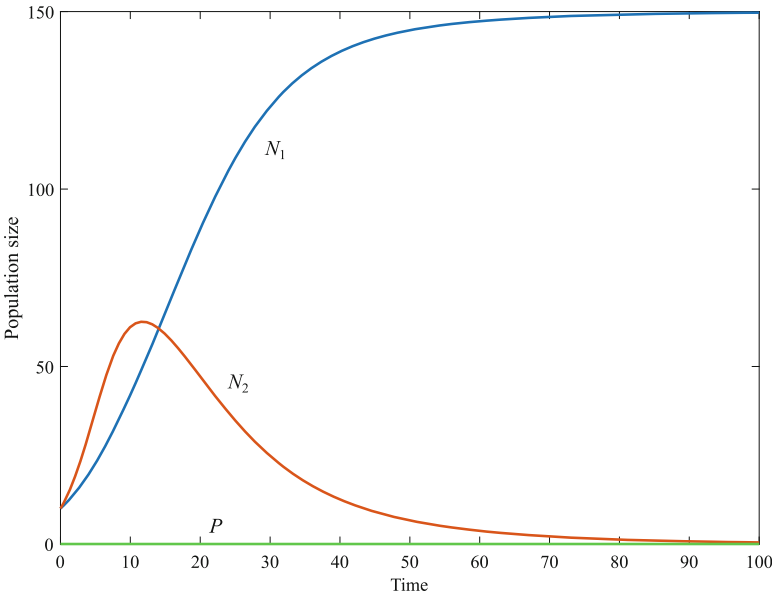


Fig. 10.4 Predator-prey-prey model solutions for the case with zero predators

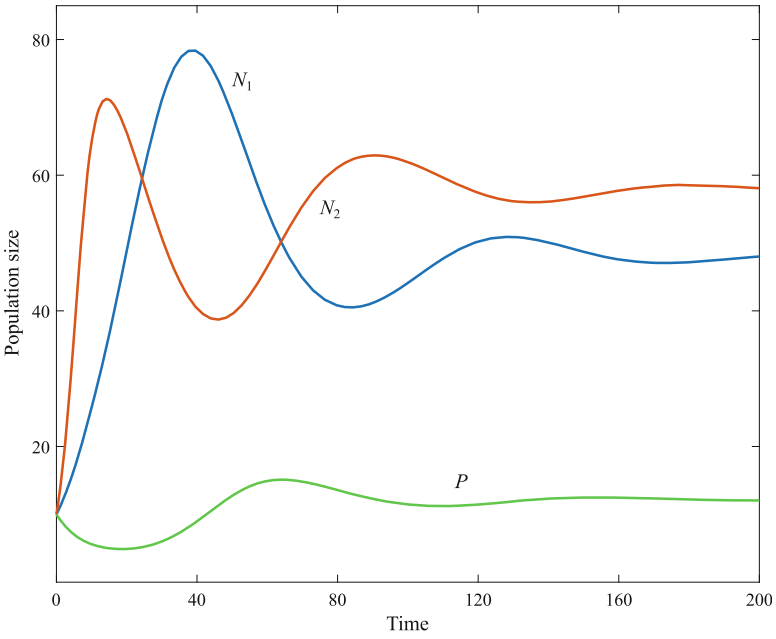


Fig. 10.5 Predator-prey-prey model solutions for the case with predators present

10.4 Summary

In this chapter, we modified the competition model from Chap. 8 so as to prevent the populations from increasing without bound. Depending on the position of the nullclines in the first quadrant (positive N_1 and N_2) we can get three possible kinds of solutions. In one case, either N_1 or N_2 wins but growing to its carrying capacity instead of without bound. In a second case, a stable steady-state exists where the nullclines cross and indicate a point where the two populations will coexist. In the last case, the nullclines cross at an unstable steady-state where they intersect, leading to one population winning (growing to its carrying capacity) with the other population going to zero. However, in this last case, either population can win depending on their initial values, unlike the first case where the winning population does not depend on the starting point. The predator-prey-prey model was used to show that the presence of a predator could allow the two prey populations to co-exist, whereas without the predator, one of the two competing prey organisms disappears.

10.5 Exercises

10.5.1 Consider a simplified form of the competition model shown in Eq. 10.1:

$$\frac{dN_1}{dt} = N_1 - N_1^2 - \frac{1}{2}N_1N_2$$
$$\frac{dN_2}{dt} = N_2 - \frac{N_2^2}{4} - \frac{1}{2}N_1N_2$$

Find the equations for the nullclines and draw a phase-plane graph of N_1 and N_2 including those nullclines. On the phase-plane graph, for each region determined by the nullclines, show the trajectory and final outcome for the two populations.

10.5.2 Using either MATLAB, Python, or Mathematica, solve the competition model illustrated in case (i) above, drawing a plot of N_1 and N_2 versus time. Possible parameter values for this scenario of case (i) would be: $a_1 = 0.2$, $a_2 = 0.4$, $\alpha = 0.9$, $\beta = 0.9$, $K_1 = 150$, and $K_2 = 100$. The initial values are not critical, but you could try $N_1(0) = 150$ and $N_2(0) = 150$. Note how these parameter values satisfy the case (i) scenario.

10.5.3 Using either MATLAB, Python, or Mathematica, solve the competition model illustrated in case (i) above, drawing a plot of N_1 and N_2 versus time. Possible parameter values for this scenario of case (ii) would be: $a_1 = 0.2$, $a_2 = 0.4$, $\alpha = 0.9$, $\beta = 0.9$, $K_1 = 100$, and $K_2 = 100$. The initial values are not critical, but you could try $N_1(0) = 150$ and $N_2(0) = 150$. Note how these parameter values satisfy the case (ii) scenario.

References

Beltrami E (1987) Mathematics for dynamic modeling. Academic, New York
Murray JD (2002) Mathematical biology. I. An introduction. Springer, Berlin

Chapter 11
Infectious Disease Models

With the current pandemic associated with the SARS-Cov-2 virus, the interest among the public and particularly among students toward models of infectious diseases is likely to be quite enhanced. Therefore, this chapter will provide an introduction to some of the mathematical modeling approaches to the spread and dynamics of infectious diseases among the human population.

11.1 Basic Compartment Modeling Approaches

Compartment models divide a system into groups and each group has an ordinary differential equation describing how the members of that group change with time. For example, a simple model of susceptible (S), infected (I), and recovered (R) organisms could be represented with the diagram in Fig. 11.1.

11.2 SI Model

The flow of individuals from the susceptible group into the infected (and therefore infectious) group can be treated as a simple product of S and I (analogous to a law of mass action). If we assume that the recovery process is proportional in a simple linear manner to the number of infected members, an ODE system for S and I could be written as:

$$\begin{aligned} \frac{dS}{dt} &= -\beta SI \\ \frac{dI}{dt} &= \beta SI - \delta I \end{aligned} \tag{11.1}$$

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_11

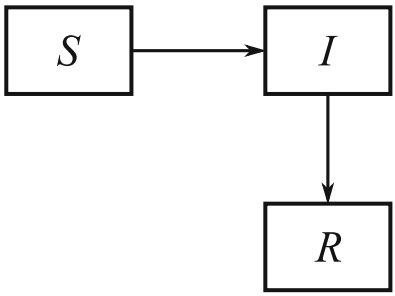


Fig. 11.1 Compartment diagram for a simple SIR model

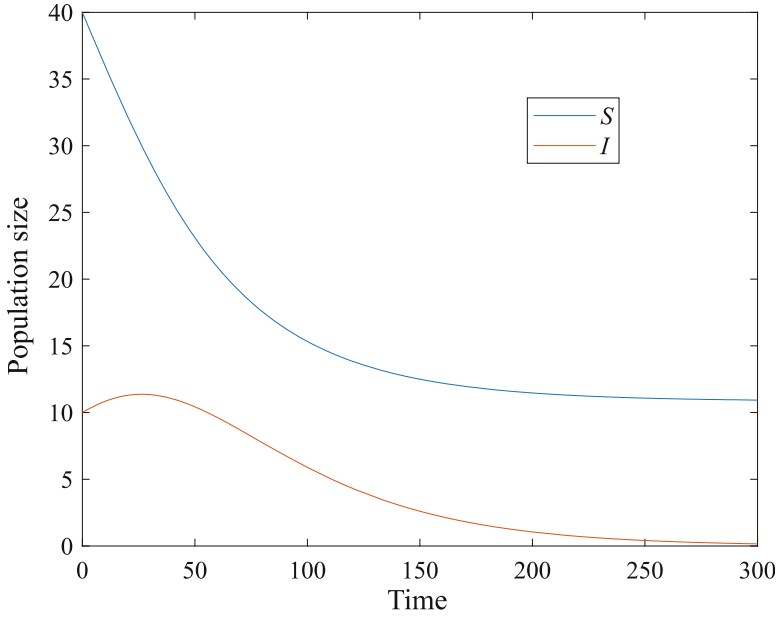


Fig. 11.2 Solution of the simple epidemic model showing S and I for parameter values $\beta = 0.001$ and $\delta = 0.03$ and initial values $S_0 = 40$, $I_0 = 10$

We can consider one possible solution for the above model given parameter values $\beta = 0.001$ and $\delta = 0.03$ and initial values $S_0 = 40$. $I_0 = 10$. In this case, the number of infected people grows slightly before declining to zero and the number of susceptible people that are never infected gradually declines to about 11 (Fig. 11.2). Although we do not show this, decreasing δ significantly still shows the number of infected declining to zero, but in that case essentially all susceptible in the population become infected.

A common question with infectious disease models of this form would be: Does an epidemic develop? In other words, given some initial numbers of susceptible and infected individuals, does the infected quantity grow with time?

$$\frac{dI}{dt} > 0 \quad ??$$

Yes, an epidemic does develop if the initial number of susceptible individuals is greater than the quantity shown here:

$$S_0 > \frac{\delta}{\beta} \quad (\text{and } I_0 > 0)$$

For the model solution shown above, if S_0 is greater than 0.03/0.001 = 30, the number of infected individuals would grow and this is indeed the case (Fig. 11.2), where S_0 was 40.

A common expression from mathematical models of epidemics is the basic reproduction number, R_0, which provides an estimate of the number of infections created by one individual in the infectious population (I). For the SIR model shown above, we can calculate:

$$R_0 = \frac{\beta S_0}{\delta}$$

Given the parameters used in the above example, $R_0 = 1.33$. If this quantity is greater than 1, the model would predict that an epidemic would occur (I would grow initially) and this is indeed what can be seen (Fig. 11.2).

11.3 SI Model with Growth in S

What if the population (S) can grow, separate from the disease process?

$$\begin{aligned} \frac{dS}{dt} &= \alpha S - \beta SI \\ \frac{dI}{dt} &= \beta SI - \delta I \end{aligned} \tag{11.2}$$

This model includes an ability for growth in S that is just proportional to S – a simple approach but admittedly one that would lead to exponential growth in S if no infectious individuals are present ($I_0 = 0$). One possible solution for this model (Fig. 11.3) given parameter values $\alpha = 0.06$, $\beta = 0.001$, and $\delta = 0.03$ and initial values $S_0 = 40$, $I_0 = 10$ shows a very different behavior than the previous case with no growth of the susceptible population. Here the population appears to go through epidemic cycles, whereby the susceptible population grows leading to a rapid increase in infections and a rapid decline in the susceptible uninfected individuals, followed by a collapse of the epidemic and another growth cycle.

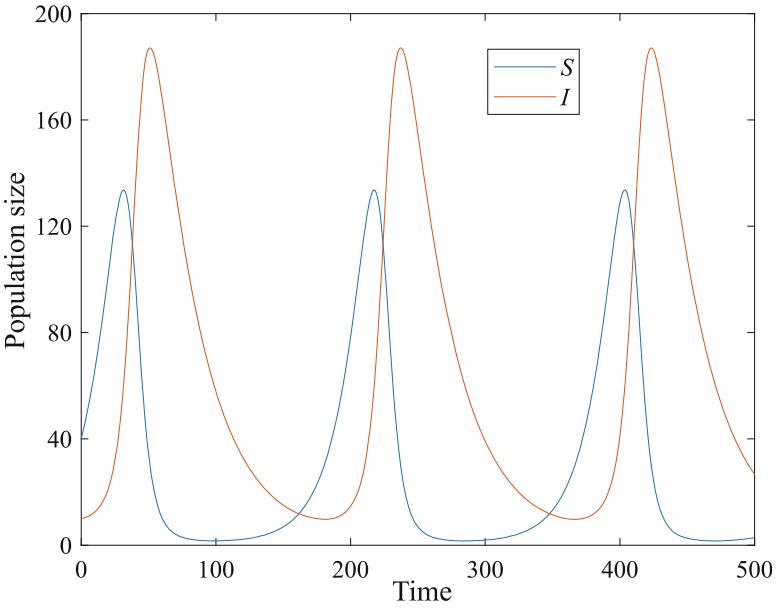


Fig. 11.3 Solution of the simple epidemic model that allows for growth in the susceptible population showing S and I for parameter values $\alpha = 0.06$, $\beta = 0.001$, and $\delta = 0.03$ and initial values $S_0 = 40$, $I_0 = 10$

Mathematical models of epidemics with vaccination have also been useful for estimating the fraction of individuals among the population that must be immunized to confer a herd immunity. An earlier expression (see Hethcote 2000; Elbasha and Gumel 2021) for this fraction leading to a herd immunity (υ^*) assuming the vaccine is perfectly effective and confers a lasting immunity gives:

$$\upsilon^* = 1 - \frac{1}{R_0}$$

where R_0 is the basic reproduction number or the number of infections created by each infected individual. For example, if R_0 is 2.0, we would expect that herd immunity would arise from 50% immunization. An R_0 value for measles of 18 would lead to an estimated critical vaccination proportion of 94% (Scherer and McLean 2002). This approach has been expanded to allow for an imperfect response to the vaccine and for an immunity that wanes over time (Elbasha and Gumel 2021):

$$\upsilon^* = \left(\frac{\mu + \omega}{\mu}\right)\frac{1}{\varepsilon}\left(1 - \frac{1}{R_0}\right)$$

where ω is the rate at which the immunity wanes with time, $1/\mu$ is the average life expectancy of the population, and ε is the efficacy of the vaccination. The ratio:

$$\left(\frac{\mu+\omega}{\mu}\right)^{-1}$$

gives the fraction of a lifetime for which protection exists by the vaccine. Considering a case where R_0 is 2.0, a vaccine has an efficacy of 80% ($\varepsilon = 0.8$), and the vaccine protection lasts 80% of a lifetime, gives a required threshold for herd immunity of 78% as compared to 50% for the case of a perfect vaccine with lifetime immunity.

More advanced models add additional compartments such as groups in the population that are exposed (E) to the infectious agent but not yet infectious, or capable of spreading the disease. Such models are often termed SEIR models. Additional examples include vaccinated groups or break all the groups into age-structured sections. For further exploration of epidemic disease modeling, the reader may wish to examine references such as Hethcote (2000) or Li (2018).

11.4 Applications Using Mathematica

Here is an example of a Mathematica notebook that could be used to generate the solution shown for the SIR model above with no growth in the susceptible population. Readers not familiar with Mathematica might wish to consider the tutorial in Appendix 3.

Infection model without pop. growth

Model equations

$$\frac{dS}{dt} = -\beta SI$$

$$\frac{dI}{dt} = \beta SI - \delta I$$

Set parameters

```
β = 0.001;
δ = 0.03;
```

Solve numerically

```
sol = NDSolve[ {s'[t] == - β s[t] × i[t], i'[t] == β s[t] × i[t] - δ i[t], s[0] == 40, i[0] == 10},
   {s, i}, {t, 0, 2000}, AccuracyGoal → 30, PrecisionGoal → 12, MaxSteps → 100000];
```

Plot solution

```
Plot[Evaluate[{s[t], i[t]} /. sol], {t, 0, 300}, AxesOrigin → {0, 0}, PlotRange → Full]
```

11.5 Applications Using MATLAB

The following list of a MATLAB script and function could be used to generate the solution shown for the SIR model with growth of the susceptible population. Readers not familiar with MATLAB might wish to consider the tutorial in Appendix 4.

```
function SI
%Script for SI model
tspan = [0 20000];
yzero = [40 10];
options = odeset('RelTol',1e-13,'AbsTol',1e-13);
[t y] = ode45(@SIode, tspan, yzero, options);
plot(t,y)
xlabel('Time')
ylabel('Population size')
legend('S', 'I')
end

function [ yprime ] = SIode( t, y )
%SI Infection model
%   Detailed explanation goes here
alpha = 0.0003;
beta = 0.001;
d = 0.03;
yprime = [alpha*y(1) - beta*y(1)*y(2); ...
            beta*y(1)*y(2) - d*y(2)];
end
```

11.6 Summary

Here, we have introduced examples of infectious disease models. Simple models with susceptible and infected populations, not including the growth in the susceptible population, can show an increase in the number of infected individuals depending on the size of the susceptible population. Eventually the number of infected declines towards zero, but depending on the recovery rate (δ), some of the susceptible are never infected or all may be infected before the disease runs its course (I declines to zero). If a growth term is included in the susceptible population, solutions can show a repeating cycle of disease outbreaks.

11.7 Exercises

11.7.1 Consider the simple SI model shown in Eq. 11.1, with these values for the parameters and initial conditions: $\beta = 0.001$, $\delta = 0.03$, $S_0 = 40$, and $I_0 = 10$. Determine if an epidemic will ensue under these conditions.

11.7.2 Consider the simple SI model shown in Eq. 11.1, with these values for the parameters and initial conditions: $\beta = 0.001$, $\delta = 0.03$, $S_0 = 50$, and $I_0 = 10$.

Solve this model using either Mathematica, MATLAB, or Python and draw a phase-plane plot of S versus I. The time span can run from 0 to 500.

11.7.3 Consider the SI model with growth in S as expressed in Eq. 11.2. Solve this model using either Mathematica, MATLAB, or Python and draw a phase-plane plot of S versus I. Use these values for the parameters and initial conditions: $\alpha = 0.06$, $\beta = 0.001$, $\delta = 0.03$, $S_0 = 40$, and $I_0 = 10$. The time span can run from 0 to 500. Describe the behavior of the system as time progresses.

References

Elbasha EH, Gumel AB (2021) Vaccination and herd immunity thresholds in heterogeneous populations. J Math Biol 83:73

Hethcote HW (2000) The mathematics of infectious diseases. SIAM Rev 42:599–653

Li MY (2018) An introduction to mathematical modeling of infectious diseases. Springer Nature, Cham

Scherer A, McLean A (2002) Mathematical models of vaccination. Br Med Bull 62:187–199

Chapter 12
Organism Environment Interactions

12.1 Introduction to Energy Budgets

Organisms exist in a state of continuous exchange of energy with their environments. These energy forms include electromagnetic radiation (e.g., sunlight absorbed or radiation emitted at infrared wavelengths), exchange with the air or other fluids in convection, heat conducted between the organism and a physical surface, and energy associated with the evaporation of water through its latent heat of vaporization and formed by metabolism in the organism. In this chapter, we will consider these energy exchanges as ultimately determining the temperature of a leaf. Mathematical models of the processes are often described as energy budgets in the sense of an accounting of energy inputs and outputs. The various components of these budgets are summarized in Table 12.1. Note that each term will have units of watts per meter squared: a watt is a joule per second and so these are energy flow rates per unit of a surface.

An early pioneer in the consideration of organism energy budgets was David M Gates, who combined his work in the text Biophysical Ecology (Gates 1980) published while he was a professor at the University of Michigan. One of us (PJS) was fortunate enough to have been a student when this text was being written and took the course by Dr. Gates, who regaled the class with fascinating stories based on his observations of organisms and how they appeared in various settings with extreme high air temperatures and sunlight intensities.

For our consideration of energy budgets in this chapter, we will make the simplification of steady-state. In other words, the flows are constant and we do not consider heat storage by our leaf. In addition, we will not consider metabolism as a source of energy in our leaf flows and also not consider any heat flow from conduction to other objects. For an energy budget of an animal, particularly a large mammal, metabolism would be an important component (see the text by Gates for examples).

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6_12

Table 12.1 Components of the energy budget model

Energy component	Variable in budget	Units
Radiation input	Q_a	W m^{-2}
Radiation output	Q_{out}	W m^{-2}
Convection	C	W m^{-2}
Metabolism	M	W m^{-2}
Evaporation	E	W m^{-2}

12.2 Radiation

The exchange of energy through electromagnetic radiation involves certain components that we can readily observe visually, such as absorbed sunlight which may also include light from the sky or reflected from the ground or other objects. But there are also important radiation components at longer wavelengths that we cannot see directly: in the infrared. All objects, including the surfaces of our leaf, give off radiation, whose quantity can be estimated from the Stefan-Boltzmann law:

$$Q_{out} = \varepsilon\sigma {T_{leaf}}^4$$

where Q_{out} is the energy emitted with units of W m^{-2}, ε is the emissivity (with values between 0 and 1), σ is the Stefan-Boltzmann constant (approximately 5.67 E-8 W m^{-2} K^{-4}), and T_{leaf} is the leaf surface temperature with kelvin units. The reader may be familiar with the term "black-body," which refers to an object that is a perfect absorber and emitter at all wavelengths: for this object the emissivity (ε) would be 1.0.

12.3 Convection

The exchange of energy with a fluid flowing around an object (the air in this case for our leaf) is termed convection (C). This exchange can occur either as an outflow from the leaf if it is warmer than the air or as an inflow if the air is warmer than the leaf.

$$C = h_c\left(T_{leaf} - T_{air}\right) \qquad h_c = k_1\left(\frac{\sqrt{V}}{\sqrt{D}}\right)$$

where T_{air} is the air temperature, h_c is the convection coefficient that depends on wind speed (V), object size (D), and a shape factor $k_1 = 9.14$ for a flat surface.

12.4 Transpiration

The evaporation of water from the leaf (E) can cool the leaf because the process requires energy.

$$E = \frac{w_{sat,leaf} - h\,w_{sat,air}}{\left(r_{leaf} + r_{air}\right)}$$

$$w_{sat} = \frac{2.17 P_{sat}}{273 + T}$$

$$P_{sat} = 0.61078\ e^{\left(\frac{17.269\,T}{T+237.3}\right)}$$

$$r_{air} = 200\left(\frac{\sqrt{D}}{\sqrt{V}}\right)$$

where w_{sat} is the saturated vapor concentration in the air inside the leaf stomata or in the surrounding air. For this expression, evaporation is proportional to the vapor concentration difference between the inside of the leaf and the surrounding air, whose vapor concentration is the saturated concentration multiplied by the air humidity (h). Evaporation also depends on the resistance to diffusion due to the leaf stomata (r_{leaf}) and that of the boundary layer above the leaf surface (r_{air}). The boundary layer resistance term has wind speed in the denominator and so V cannot be exactly zero. Therefore to simulate a lack of wind, we will choose some small value such as 0.01 m s^{-1}. The formulae above give transpiration in mass units (kg m^{-2} s^{-1}) and this must be converted to energy units by multiplying E by the heat of vaporization for water (λ), where λ is 2.427 $E6$ J/kg. (This value does change slightly with temperature of the water but we will not consider this aspect.)

12.5 Total Energy Budget

Combining all the budget components together, with positive inputs and negative outputs gives:

$$Q_a - \varepsilon\sigma\left(T_{leaf} + 273.15\right)^4 - h_c\left(T_{leaf} - T_{air}\right) - \lambda\left(\frac{w_{sat,leaf} - h w_{sat,air}}{r_{leaf} + r_{air}}\right) = 0$$

Note that it is possible for the convection term to be either an input or an output. The evaporation (transpiration) term can in principle also be an energy input if water is condensing as dew or frost on the leaf. An example of this last possibility will be discussed later. The budget terms must add to zero as indicated above because this is a steady-state energy budget and so the flows must balance such that leaf temperature is constant for the conditions supplied (heat storage not considered).

Notice that T_{leaf} is present in the budget in various forms such that we cannot readily rearrange the equation analytically to solve for T_{leaf}. We will therefore need to use a numerical method, often termed root finding methods because we are looking for roots of the function on the left-hand side of the budget equation. There are many such methods, but we will focus on the one called Newton's Method (Table 12.2).

Table 12.2 Possible values for the various model parameters. If the model is solved for these test case values, the resulting leaf temperature should be 27.59 °C

Variable	Test case value	Range for experimenting	Units
Q_a	800	200–900	W m^{-2}
h	0.5	0–1	
V	1.0	0.01–5	m s^{-1}
D	0.05	0.001–0.25	m
ε	0.95	0–1	
T_{air}	25.0	5–40	°C
r_{leaf}	100	100–2000	s m^{-1}

12.6 Solving the Budget: Newton's Method for Root Finding

The Python module scipy has a function for root finding using Newton's method that is applied in the program shown here:

```
from scipy import optimize
import math

# Enter parameter values
Qa = 800.0
h = 0.5
V = 1.0
D = 0.05
rleaf = 100.0
emiss = 0.95
Tair = 25.0
lamb = 2.427e6
x1 = 25.0

def wsat(T):
    psat = 0.61078 * math.exp( 17.269 * T / ( T + 237.3 ) )
    return 2.17 * psat / ( 273.15 + T )

def f(Tleaf):
    global Qout, C, E
    Qout = emiss * 5.67e-8 * (Tleaf+273.15)**4
    hc = 9.14 * math.sqrt(V) / math.sqrt(D)
    C = hc * ( Tleaf - Tair )
    rair = 200 * math.sqrt(D) / math.sqrt(V)
    E = lamb * ( wsat( Tleaf ) - h * wsat( Tair ) ) / ( rleaf + rair )
    return Qa - Qout - C - E

root = optimize.newton(f, x1)
print ('Leaf temperature = %.4f ' % (root) )
print ('Energy terms')
print ('     Radiation absorbed = %.1f ' % (Qa))
print ('     Radiation out = %.1f ' % (Qout))
print ('     Convection = %.1f ' % (C))
print ('     Evaporation = %.1f ' % (E))
```

If you run the above Python program, the results should be displayed as:

Leaf temperature = 27.5939
Energy terms
  Radiation absorbed = 800.0
  Radiation out = 440.7
  Convection = 106.0
  Evaporation = 253.3

A related programming form for MATLAB is shown here:

```
function [ Sum ] = Budget( Tleaf )
global Qa h V D rleaf emiss Tair lamb x1 Qout C E
Qout = emiss * 5.67e-8 * (Tleaf+273.15)^4;
hc = 9.14 * sqrt(V) / sqrt(D);
C = hc * (Tleaf - Tair);
rair = 200 * sqrt(D) / sqrt(V);
E = lamb * (Wsat(Tleaf) - h * Wsat(Tair)) / (rleaf + rair);
Sum = Qa - Qout - C - E;
end


function [ wsat ] = Wsat( T )
Psat = 0.61078 * exp( 17.269 * T / (T + 237.3));
wsat = 2.17 * Psat / (273.15 + T);
end


global Qa h V D rleaf emiss Tair lamb x1 Qout C E
Qa = 800;
h = 0.5;
V = 1.0;
D = 0.05;
rleaf = 100;
emiss = 0.95;
Tair = 25;
lamb = 2.427e6;
x1 = 25;
root = fzero( @Budget, x1 );
fprintf('Leaf temperature = %.4f\n', root)
fprintf('    Radiation absorbed = %.1f\n', Qa)
fprintf('    Radiation out = %.1f\n', Qout)
fprintf('    Convection = %.1f\n', C)
fprintf('    Evaporation = %.1f\n', E)
```

The first component shown above is the function containing the leaf energy budget (saved in the file Budget.m). The second component above is the function calculating the saturation water vapor concentration (saved in a file Wsat.m). The third component is a script (saved as the file LeafBudget.m) that enters some values for the budget variables, calls the root finding function (fzero), and then displays results from the budget solution. Note that several variables are treated as global variables to simplify the printing of the results. If run with MATLAB, this program generates the following results:

>> LeafBudget
Leaf temperature = 27.5939
  Radiation absorbed = 800.0

Radiation out = 440.7
Convection = 106.0
Evaporation = 253.3

Here is a Mathematica notebook that could be used to solve the leaf energy budget:

Leaf energy budget

Enter variable values

```
In[•]:= Qa = 800.0;
        h = 0.5;
        v = 1.0;
        d = 0.05;
        rleaf = 100.0;
        emiss = 0.95;
        Tair = 25.0;
        lamb = 2.427*^6;
        x1 = 25.0;
        hc = 9.14 * Sqrt[v] / Sqrt[d];
        rair = 200 * Sqrt[d] / Sqrt[v];
```

Create the functions for saturation water vapor conc.

```
In[•]:= psat[T_] := 0.61078 * Exp[ 17.269 * T / (T + 237.3)]
        wsat[T_] := 2.17 * psat[T] / (273.15 + T)
```

Create the functions for the energy budget

```
In[•]:= Qout[t_] := emiss * 5.67*^-8 * (t + 273.15)^4;
        conv[T_] := hc * (T - Tair);
        evap[T_] := lamb * (wsat[T] - h * wsat[Tair]) / (rleaf + rair);
        sum[T_] := Qa - Qout[T] - conv[T] - evap[T];
```

Solve the energy budget

```
In[•]:= root = FindRoot[sum[TLeaf], {TLeaf, x1}]
Out[•]= {TLeaf → 27.5939}
```

Show the results

```
In[•]:= Print["Leaf temperature = ", NumberForm[TLeaf /. root, {6, 4}]]
        Print["   Radiation absorbed = ", NumberForm[Qa, {6, 1}]]
        Print["   Radiation output = ", NumberForm[Qout[TLeaf /. root], {6, 1}]]
        Print["   Convection = ", NumberForm[conv[TLeaf /. root], {6, 1}]]
        Print["   Evaporation = ", NumberForm[evap[TLeaf /. root], {6, 1}]]
        Leaf temperature = 27.5939
           Radiation absorbed = 800.0
           Radiation output = 440.7
           Convection = 106.0
           Evaporation = 253.3
```

12.7 Experimenting with the Leaf Energy Budget

Once we have a way to solve the leaf energy budget, we can do many experiments to see how various environmental conditions affect the leaf temperature. Here, we will consider two examples, but one could design many others. For these two cases, we will consider a bright sunny day with Q_a of 800 W m^{-2} and a nighttime with Q_a of 250 W m^{-2}. Notice that the chosen nighttime value is not zero; the leaf surface still absorbs radiation at infrared wavelengths from the sky and other surroundings, like the ground. As described in detail by Gates (1980) these nighttime values can vary considerably depending on air temperature, but we have chosen one representative possible value.

It is worth remembering that all of our energy budget terms express energy flows per unit surface area. Students in our classes have sometimes thought that a large leaf will get hotter because it is absorbing more total energy due to its large size. But the other energy terms would also increase in total energy. As we will see below, if a large leaf gets hotter in bright sunlight, it is because of the effects of leaf size on the other budget terms like convection.

1. Effect of wind speed on leaf temperature

Solving the leaf energy budget for a range of wind speeds from 0.05 to 5.0 m/s (and assuming the test case values for other variables (Table 12.2)) gives the leaf temperatures as shown in Fig. 12.1.

The wind speed appears in the leaf energy budget in two terms: the convection coefficient and the leaf boundary layer resistance. As wind speed increases, the

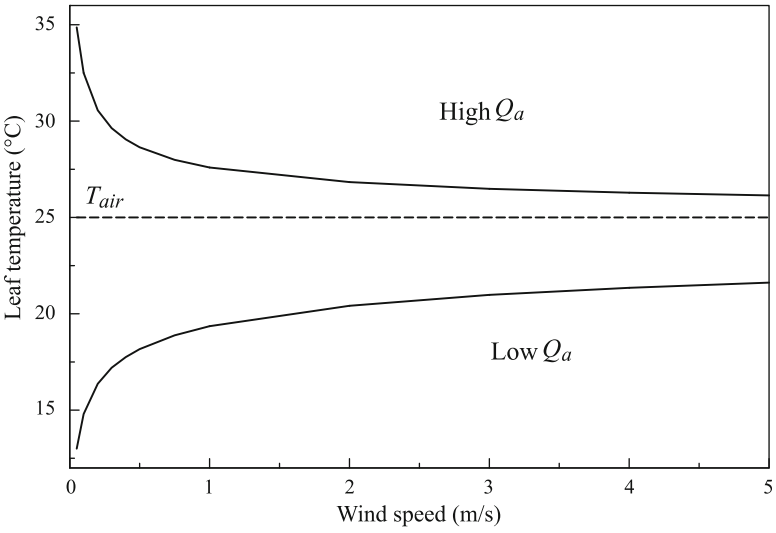


Fig. 12.1 Leaf temperature for a range of wind speeds under high radiation (bright daylight) and low radiation (nighttime) conditions

convection coefficient increases, leading to a closer coupling of the leaf to the air temperature. Whether this would warm or cool the leaf depends on the radiation absorbed. For the bright sunny day in this case, the leaf is above air temperature and so convection cools the leaf. For the nighttime case, the leaf is cooler than air temperature and so convection warms the leaf. The leaf in these scenarios is also being cooled by evaporation and increasing wind speed increases evaporation by reducing the boundary layer resistance (r_{air}). The experiment can be extended further by changing the leaf stomatal resistance (r_{leaf}) to reduce evaporative cooling. This could be relevant here because many plants will close their stomata at night when there is no sunlight to drive photosynthesis, thus conserving water.

Another interesting result from these solutions becomes apparent if you pay attention to the energy budget components along with the leaf temperature. For the low radiation condition at nighttime and at the lowest wind speed, the evaporation term becomes negative. This means that dew would be forming on the leaf surface, whereby the process is leading to a slight heat input to the leaf.

2. Effect of leaf size on leaf temperature

Solving the leaf energy budget for a range of leaf size from 0.001 to 0.5 m (assuming a wind speed of 0.5 m/s and the test case values for the other variables) gives the leaf temperatures as shown in Fig. 12.2.

Leaf size appears in the energy budget in two terms: the convection coefficient and the leaf boundary layer resistance. Larger leaves will have a reduced convection coefficient and reduced convection, thus leading to a leaf that can deviate considerably from air temperature. On the other hand, larger leaves will have an increased boundary layer resistance leading to reduced evaporative cooling. Notice that very

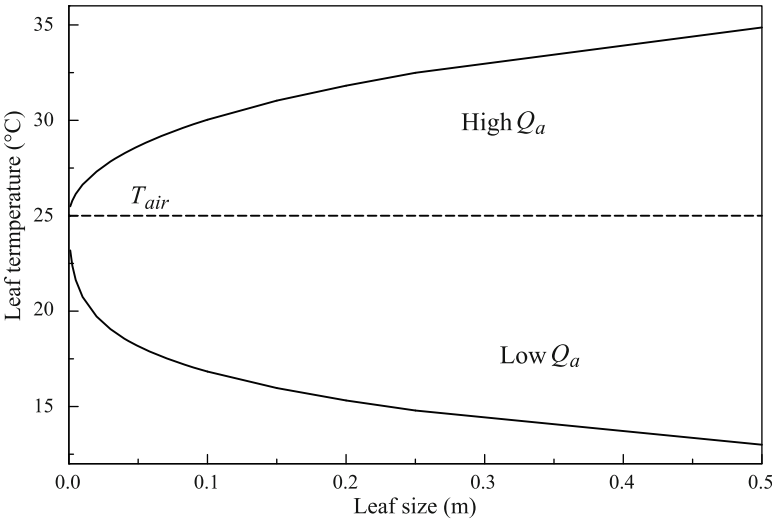


Fig. 12.2 Leaf temperature for a range of leaf sizes under high radiation (bright daylight) and low radiation (nighttime) conditions

small leaves of a few millimeters size are very close to air temperature regardless of the radiation absorbed value.

As shown in this scenario for the nighttime light condition, a larger leaf can cool (by radiative output) well below air temperature. One can also play with the air humidity, which broadly speaking tends to increase at night when air temperature declines. If we increase the air humidity from 0.5 to 0.8 (80% relative humidity), a leaf of 0.25 m size shows a negative evaporation term, again signifying that dew is forming on the leaf.

The effects of wind speed and leaf size shown here apply to our leaf's energy budget, but similar effects can be apparent in other circumstances. If we want to measure air temperature accurately, especially with low wind speed, we need a thermometer with a small size that will have a high convection coefficient and be closely coupled to the air. If you have a chance to be out at night with a dark clear sky and nearly zero wind, you might notice that your face looking up at the sky feels colder than when the wind picks up. Or that on a night with a high humidity and low wind speed, dew (or frost) tends to form on large surfaces (like the roof of a car) because it has cooled radiatively below the air temperature unlike a small surface that is more closely coupled to the air temperature.

12.8 Summary

An energy budget model was developed to describe the interactions between a leaf and its environment, giving rates of energy and water vapor exchange and determining the temperature of the leaf. Examples were given of an implementation of the budget with Mathematica, MATLAB, and Python programs. A couple of interesting scenarios were examined: the effects of wind speed and the effects of leaf size. Wind speed appears in the budget within the convection term, having a nonlinear relationship between the convection coefficient and wind speed. The effect of wind speed is complicated because it depends on the radiation absorbed (a bright sunny day as opposed to at night). Higher wind speed increases convection and this can either decrease the leaf temperature if it is above air temperature, illustrated by a bright sunny day, or increase the leaf temperature if it is below air temperature, illustrated by the nighttime case. Leaf size also has a complicated effect on leaf temperature. On a bright sunny day where the leaf temperature was above air temperature, increasing leaf size leads to an increase in leaf temperature. At nighttime where the leaf temperature was below air temperature, increasing leaf size leads to a further decrease in leaf temperature. This last observation can help explain why large surfaces are more likely to show dew or frost condensation at night, particularly if the humidity is high.

12.9 Exercises

12.9.1 The leaf energy budget includes the relative humidity of the air. Check the budget equations to see where the relative humidity appears. Show how this parameter affects the temperature of the leaf. Choose a bright sunny day, like the test case values in Table 12.2 and pick several values of humidity from dry, such as 5% or 10% (entered as 0.05 or 0.10) up to a humid day with 80% or 90%.

12.9.2 Included in the leaf energy budget is a parameter called r_{leaf} that describes the resistance for water diffusing out of the leaf stomata. Increasing resistance implies that the stomata are closing. Run a program for the energy budget to see how different stomatal resistances affect the leaf temperature, choosing a range of values for r_{leaf}. Use the test case (Table 12.2) for most budget values. Why would your results depend on the relative humidity of the air?

12.9.3 Our example of using the leaf energy budget to study the effects of leaf size considered the case of nighttime with low radiation absorbed. Under some conditions, we could see a negative evaporation term, indicating condensation was occurring. Solve the energy budget for those conditions and show how the occurrence of dew formation depends on the humidity of the air, tending to occur when the humidity is high.

Reference

Gates DM (1980) Biophysical ecology. Springer, Berlin

Appendix 1: Brief Review of Differential Equations in Calculus

Here we shall start by giving a brief review of what is minimally needed to recall the basic material about first-order ordinary differential equations (ODEs) used in this text. In fact, we will not bother going over all the rules of differentiation which you may have forgotten by now (except, perhaps for $(x^n)' = nx^{n-1}$), since you can review that by yourself or ask the instructor. However, we find important that we recall the following basic concepts:

Definitions

1. A nice (say, "continuous") function $y = f(t)$ defined on an "interval" I has a derivative at t if

$$\lim_{\Delta t \to 0} \frac{f(t+\Delta t) - f(t)}{\Delta t} \quad \text{exists.}$$

   Such limit is usually denoted by $\frac{dy}{dt}$ or $f'(t)$[1]

2. A first-order ODE is an equation involving an unknown function y of one variable x and its derivative y' with respect to (w.r.t.) x. In explicit form, it is written

$$y' = f(x, y), \ (x, y) \text{ in D} \qquad \text{(A.1.1)}$$

   Remarks: (i) D denotes a "nice" domain of points (x,y) in the plane of the given function f of two variables.

   (ii) A solution for the ODE is a function

[1] Recall that both the independent variable t and the dependent variable y could be denoted by other letters. In fact, it is common to use t as independent variable to indicate time, in which case, the derivative $\frac{dy}{dt}$ represents the instantaneous rate of change of y w.r.t. t

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6

$$y = h(x)$$

defined on an interval I of real numbers having its graph contained in the domain D and satisfying

$$h'(x) = f(x, h(x)) \quad \text{for all } x \text{ in I} \tag{A.1.2}$$

(iii) When an initial condition is prescribed in (ii), i.e., a point (x_0, y_0) in D in the graph of f (see A.1.1) is given, then a unique solution of A.1.1 can be obtained. Then we say that an Initial Value Problem (IVP) is given and we write it as:

$$y'(x) = f(x, y(x)) \quad \text{for all } x \text{ in I} \tag{A.1.3}$$

$$y(x_0) = y_0 \tag{A.1.4}$$

A solution for the IVP is a function defined on an interval I of real numbers having its graph contained in the domain D and with (x_0,y_0) belonging to D that satisfies the conditions

$$h'(x) = f(x, h(x)) \quad \text{for all } x \text{ in I}$$
$$h(x_0) = y_0$$

Example 1 (IVP 1)

$$\begin{cases} y' = 3y \\ y(0) = 2 \end{cases}$$

This differential equation is of type called separable, because we can separate the variables on each side of the equation, as follows.

$$\frac{1}{y}dy = 3\ dx$$
$$\int \frac{1}{y}dy = \int 3\ dx$$
$$\ln|y| = 3x + C_0$$

Therefore, taking the exponential of each side, we get

$$e^{\ln|y|} = e^{3x+C_0}$$
$$|y| = e^{3x}e^{C_0}$$

In other words, we get

$$y = Ce^{3x} \quad \text{where } C = \pm e^{C_0}$$

For the IVP on this example,

$$Ce^{3x_0} = y_0$$

Using $x_0 = 0$ and $y_0 = 2$ (our initial condition) gives

$$y = 2e^{3x}$$

Example 2 The Initial Value Problem (IVP 2)

$$\begin{cases} \dfrac{dx}{dt} = x^2 \\ x(2) = 1/2 \end{cases}$$

Using separation of variables, we find a general solution[2]:

$$x = \frac{1}{C - t}$$

Using the given initial condition x(2) = ½, we find C = 4, so that the solution of our IVP 2[3] is

$$x = \frac{1}{4 - t}$$

[2] You should try to find this general solution!

[3] Note here that had the initial condition been given as $x(t_0) = 0$ for a given t_0, the unique solution of this new IVP would necessarily be the zero solution $x \equiv 0$, regardless of the value of t_0.

Geometric Interpretation of the IVP

Let us think about what (A.1.1) and (A.1.4) mean for having a unique solution (although we won't prove this math result but it can be shown under suitable hypotheses!). First, the initial condition (A.1.4) says that the solution has its graph passing through the point (x_0, y_0). Then, (A.1.1) says that the "slope" of the graph at that point must be the "value" $f(x_0, y_0)$. Now, imagine that we could draw tiny little arrows at all the possible points (x_0, y_0) with their directions given by their corresponding slopes $f(x_0, y_0)$. And then connect those points to get curves that follow their directions [i.e., connect the dots!]. The two resulting pictures would be what is called in mathematics a "vector field" and its corresponding "solution curves". As a matter of fact, any CAS software would allow us to see those vector fields and their solution curves! Let us do that for the two Examples considered above. With the help of (say) Mathematica, we obtain Figs. A.1.1 and A.1.2 below corresponding to Examples 1 and 2, respectively.

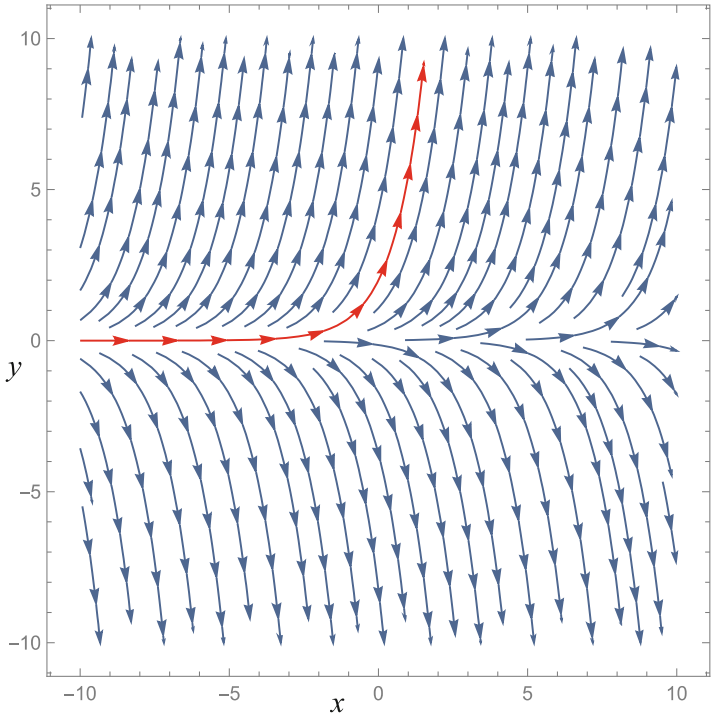


Fig. A.1.1 Vector field and its solution curves for Example 1. The highlighted (red) curve shows the unique solution for the given IVP

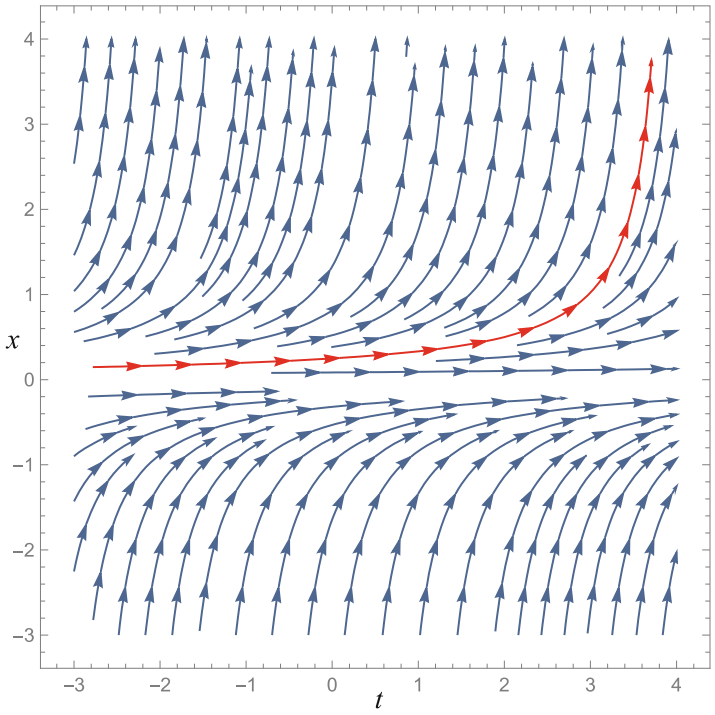


Fig. A.1.2 Vector field and its solution curves for Example 2. The highlighted (red) curve shows the unique solution for the given IVP

We should understand that the solution curves displayed in Figs. A.1.1 and A.1.2 indicate their flows in relation to each other and single out the particular solution curve passing through a given point (x_0, y_0) in the display. In this manner, those graphical displays can be used to suggest possible steady-states (namely where $y = 0$ in both Figs. A.1.1 and A.1.2). Moreover, notice that the flows in each of the figures are well suited to show that the steady-state$y = 0$ is unstable in both cases: nearby flows diverge away from $y = 0$ on both sides in Fig. A.1.1, and on one of the sides of $y = 0$ in Fig. A.1.2, hence the instability of $y = 0$ in each situation.

A related plot of the vector field and solution curves could be drawn for the Logistic Equation described in Chap. 7:

$$\frac{dN}{dt} = N - N^2$$

This plot is shown in Fig. A.1.3.

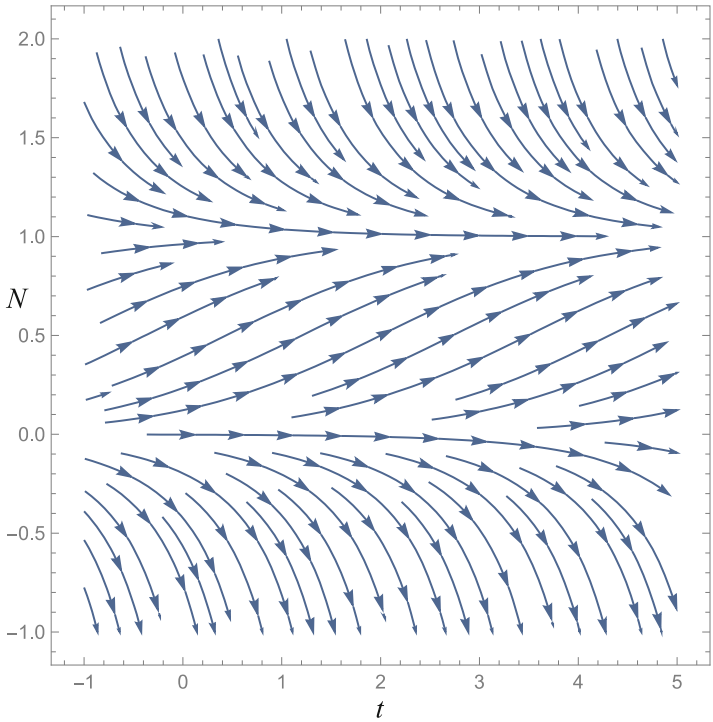


Fig. A.1.3 Vector field and its solution curves for the Logistic Equation discussed in Chap. 7. Note here that the steady-state$N = 0$ is unstable whereas $N = 1.0$ is stable

Appendix 2: Numerical Methods for Solving ODEs

One computational approach for solving an ODE (or system of ODEs) relies on a finite difference method as applied to the model equation. For a simple example, consider an exponential growth model for a population of organisms:

$$\frac{dN}{dt} = rN$$

where N is the population size, t is time, and r is the growth rate parameter. We then translate this equation into a finite difference approximation:

$$\frac{\Delta N}{\Delta t} \approx rN$$
$$\Delta N \approx rN\Delta t$$

Starting with an initial condition of N_0 at $t = 0$, we can advance our solution for time steps with the spacing of Δt:

$$N_{t+1} = N_t + \Delta N$$

A graphical explanation for this method (often referred to as Euler's Method) shows how errors arise at each step because the slope of the function at the starting time is assumed to apply for the entire interval (Fig. A.2.1). Note that reducing the size of the time step (Δt) would result in less error, a point that we will come back to a bit later.

A more accurate method with less error could be termed the Runge-Kutta second-order method and uses two estimates of the change in N after one time step. Graphically we can see (Fig. A.2.2) that the resulting error is less than that obtained with only the initial slope estimate as used in Euler's Method. In this case, two estimates are made for the change in N, termed $\Delta N'$ and $\Delta N''$. The final estimate for ΔN then relies on the average of the two estimates.

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6

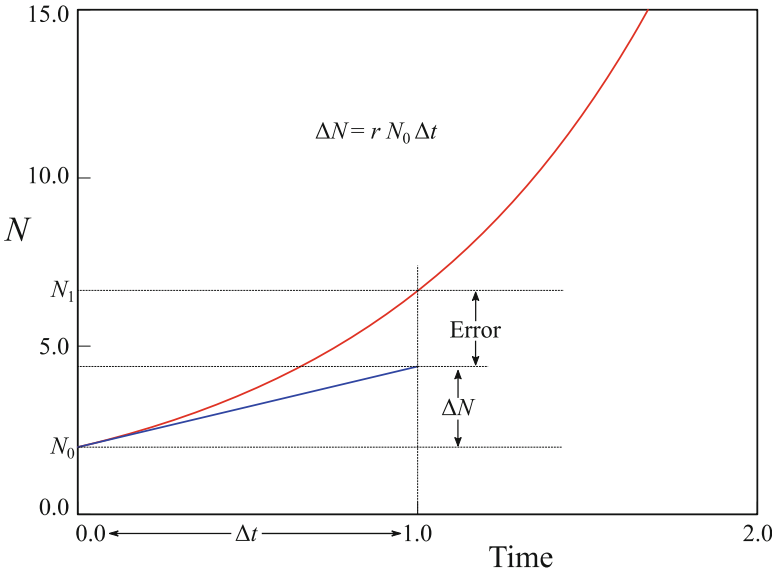


Fig. A.2.1 Graphical representation of the simple Euler's Method for solving an ODE based on a finite difference representation. The curving red line shows the actual exponential function that could be developed as an analytical solution to this exponential growth model

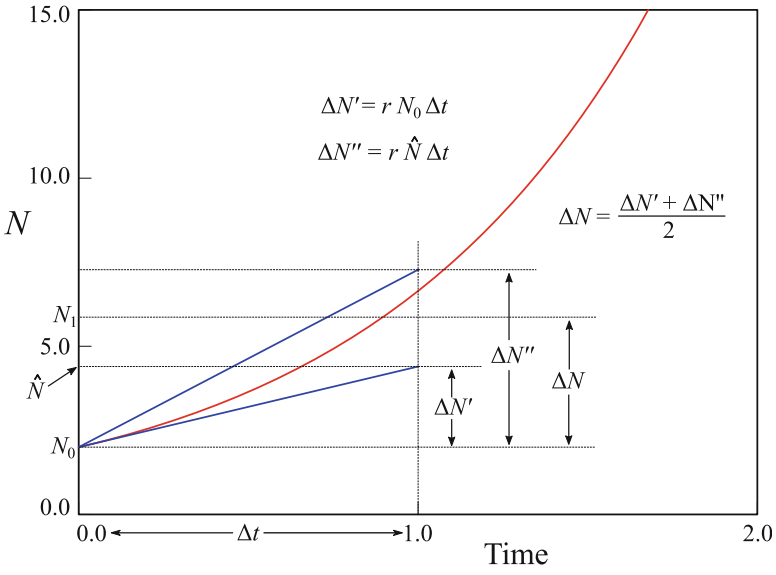


Fig. A.2.2 Graphical representation of the Runge-Kutta second-order method. The estimate for the change in N is based on an average of two estimates ($\Delta N'$ and $\Delta N''$)

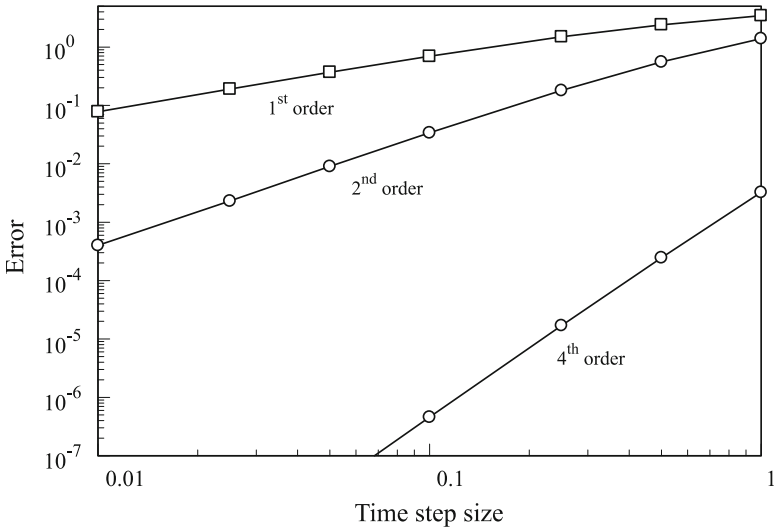


Fig. A.2.3 Errors in the numerical solution of the logistic population ODE (discussed in Chap. 7) for three different finite difference methods, based on the solution at time 3.0 for values of $r = 0.5$ and $K = 100$

A still better approach called the Runge-Kutta fourth-order method uses four estimates of the change in N over the time step. The reason that we call this method "better" is illustrated in Fig. A.2.3. Note that for all three methods, the error can be made smaller by reducing the time step size (which of course increases the computational effort). But the higher order methods result in smaller errors, but even more importantly, errors that decrease faster with reduction in time step size.

Appendix 3: Mathematica Tutorial

A few introductory notes. This tutorial assumes that you have Mathematica (www.wolfram.com) running on your computer. The program is generally used by creating what are called "notebooks," which are documents containing statements to Mathematica and the results generated by the program. Your Mathematica installation should have access to a full set of documentation including more complete tutorials, particularly in the documentation called "Fast introduction." Here, we provide a simpler introduction that our students have typically completed within an hour or so. You can also find many tutorial introductions on YouTube.

Start your Mathematica program and open a new notebook. One aspect of Mathematica that students often forget at the beginning is that if you have entered a statement that you want Mathematica to execute, you have to press the shift-enter keys or the enter key on your keyboard numeric pad. Try typing "2+2" and then shift-enter. You should see:

In[1]= 2 + 2
Out[1]= 4

The labels "In" and "Out" are provided by Mathematica along with a number indicating that particular input and output statement (these numbers are left off in the statements below). Mathematica has many functions like a calculator:

In[.]= 2^8
Out[.]= 256

In[.]= Sin[Pi/2]
Out[.]= 1

Note that the functions built into Mathematica start with an upper-case letter and that Pi is available as a special quantity. The arguments to the built-in functions are provided in square brackets. This is often a stage where students learn that syntax is important! You cannot generally substitute parentheses where square brackets are

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6

called for and if you enter "sin" without the first letter in upper-case, Mathematica will not know you are calling for the sine function.

The usual standard arithmetic operations are available like in a calculator:

In[.]= 6.8+5.1
Out[.]= 11.9

Priorities exist for operations, such that multiplication and division have higher priority than addition and subtraction:

In[.]= 2+3*4
Out[.]= 14

Note that 3 is multiplied by 4 before adding 2. Exponentiation has a still higher priority:

In[.]= 1+2^3
Out[.]= 9

Note that 2 is cubed before adding 1. You can also use parentheses to control priority:

In[.]= (4+2)/3
Out[.]= 2

This forces the addition before dividing by 3.

One interesting aspect of Mathematica is that it tries to maintain perfect accuracy in operations such as:

In[.]= 2/3*2
Out[.]= 4/3

Whereas a calculator might display 1.3333...

You can always request a numerical answer using the "N" function:

In[.]= N[2/3*2]
Out[.]= 1.33333

Or with some specified number of digits:

In[.]= N[Pi,20]
Out[.]= 3.1415926535897932385

Try this above input statement by replacing the 20 with a larger value like 200.

Working with Variables

Generally, letters or words starting with lower-case are used for the names of variables, such as in:

In[.]= x=2.5
Out[.]= 2.5

Now you can use that variable x in other calculations

In[.]= 3 * x
Out[.]= 7.5

Working with Your Own Functions

Mathematica comes with many built-in functions such as Sqrt[x], Exp[x], Log[x], Sin[x], Arcsin[x], and many others. But you can also define your own. Let's create a function called "f" (you can give them any name).

In[.]= f[x_]=2 * x ^ 2

Note that this is a function of x and that the brackets are required, the underscore after the first x is required, as is the "=". Now we can use this function.

In[.]= f[2]
Out[.]= 8

Finding Derivatives

We can use the "D" function to find derivatives.

In[.]= D[2 x^2, x]
Out[.]= 4 x

Note that we entered the function without the explicit multiply symbol *. This works because Mathematica assumes that two variables are to be multiplied if there is a space character in between them. If you want x multiplied by y, you can enter x*y or x y (with the space). If you just enter xy, Mathematica will assume that you are referring to a variable named xy.

If you are following this tutorial completely, you might get an error message from that last call to the D function. This is because we earlier set the variable x to the value 2.5. Mathematica will keep that value forever (until you restart the program). In this case, we want Mathematica to treat x as an unknown variable. So we need to clear the value of x:

In[.]= Clear[x]

Now, the D function should give the result shown above. You can also save the result of the derivative function as another function.

As an aside, three of the errors noted above are very common when students first start using this program:

- Typing xy when we mean x y or x*y
- Inconsistent use of upper-case and lower-case (sin instead of Sin, for example)
- Expecting a symbol to be treated as an unknown variable when it has a value set (need to clear the variable).

We can also create a function using the result of a function like D for calculating a derivative

In[.]= deriv[x_] = D[3 * x^3, x]
Out[.]= $9\ x^2$

Note that this definition uses just = instead of :=. Now you can evaluate that derivative for some value:

In[.]= deriv[1.2]
Out[.]= 12.96

Solving Equations

The Solve function can be used to solve many expressions.

In[.]= Solve[a x^2 + b x + c = 0, x]
Out[.]= $\left\{\left\{x \to \frac{-b-\sqrt{b^2-4ac}}{2a}\right\}, \left\{x \to \frac{-b+\sqrt{b^2-4ac}}{2a}\right\}\right\}$

In[.]= Solve[y = 4 x + 2, x]
Out[.]= $\left\{\left\{x \to \frac{1}{4}(-2+y)\right\}\right\}$

In[.]= Solve[y = 3 Exp[x], x, Reals]
Out[.]= $\left\{\left\{x \to \text{ConditionalExpression}\left[\text{Log}\left[\frac{y}{3}\right], y > 0\right]\right\}\right\}$

The "ConditionalExpression" indicates that the solution is valid for positive values of y. If you leave out the words "Reals," you will get a more complicated solution that is also valid for complex numbers.

Symbolic Math Quick Tour

In[.]= Expand[(x + 2) ^ 2]
Out[.]= $4+4\ x + x^2$

In[.]= Factor[x^2 + x – 6]
Out[.]= $(-2 + x)(3 + x)$

In[.]= Collect[a x + 4 y + c x, x]
Out[.]= (a + c) x + 4 y

In[.]= Together[x^2 / (x^2 – 1) + x / (x^2 – 1)]
Out[.]= $\frac{x}{-1+x}$

In[.]= Cancel[(x^2 – 1) / (x – 1)]
Out[.]= 1 + x

Sharing of Variables

By default, if you have several notebooks open at once, Mathematica notebooks share symbols such as variables across all of those notebooks. You can change this

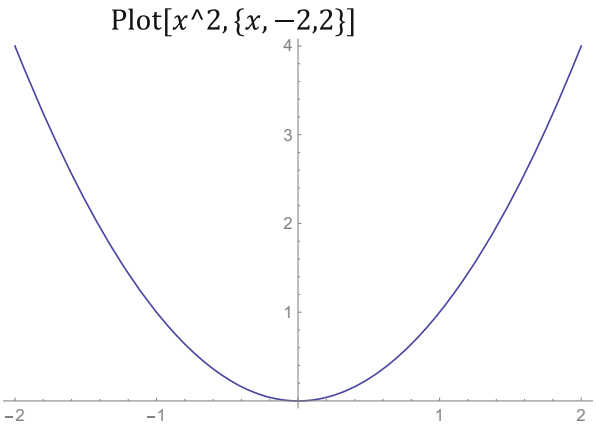
behavior such that symbols will be local to individual notebooks and not shared in common.

To make notebook symbols local on an individual notebook basis, see the Mathematica program menu item "Evaluation -> Notebook's Default Context. . ." and then choose "Unique to this notebook."

If you want every notebook that you work with to have local symbols, you need to set an option called "CellContext." Follow the menu item route "Edit. . .Preferences. . .Advanced" and press "Open Option Inspector.". Look under "Cell Options. . .Evaluation Options" and find the option called "CellContext" and set it to "Notebook." You can also find this option by searching for "context" in the Lookup field near the top of the Options dialog.

Creating Plots in Mathematica

A simple plot of a function can be created. This statement should generate the plot as shown.

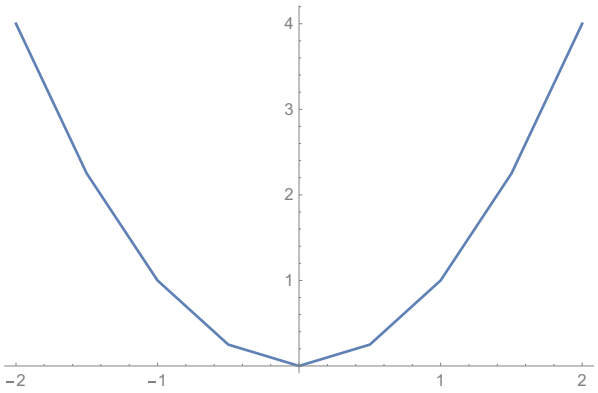


You can also create a plot from a series of data points. Here, we will use the Table function to enter the points and then draw a plot with lines connecting the points.

mytable = Table[{x, x^2}, {x, −2, 2, 0.5}]

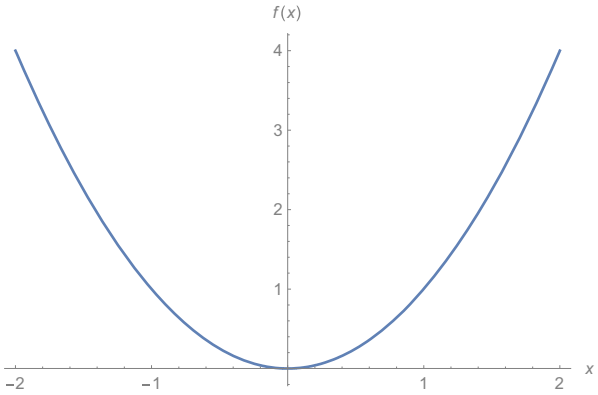
This will create a table with pairs of points having x values from −2 to 2 with a spacing of 0.5 along with the corresponding squares of those x values. Entering this statement will draw the plot:

ListLinePlot[mytable]



Next, we can create a plot of a quadratic function, specifying the x-axis range from −2 to 2 and labels for the axes.

```
a = 1; b = 0, c = 0;
Plot[a x^2 + b x + c, {x, −2, 2}, AxesLabel -> {x, f[x]}]
```



Using Mathematica to Solve the Discrete Logistic Population Model

The following set of statements comes from a notebook that was used to work with the discrete logistic population model that was described in Chap. 3.

Discrete logistic using functions

Create model function and derivative

```
f[x_]:=r x (1 - x)
fprime[x_]=D[f[x], x]
r (1-x)-r x
```

Set parameters

```
r = 1.5;
```

Find fixed points

```
Solve[f[x] - x==0, x]
{{x->0.},{x->0.333333}}
```

Assess stability of fixed points

```
fprime[0]
1.5
```

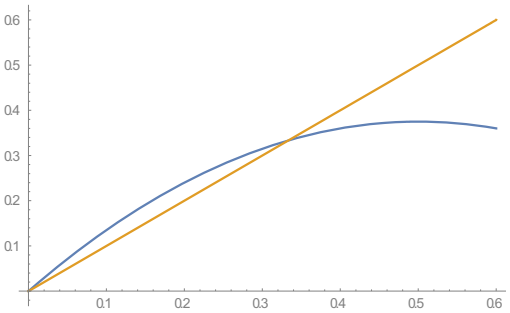
This fixed point is unstable.

```
fprime[1/3]
0.5
```

This fixed point is stable.

Plot results for cobwebbing

```
Plot[{f[x], x}, {x, 0, 0.6}]
```



Using Mathematica with OrdinaryDifferential Equations

Mathematica has functions for working with ordinary differential equations. The notebook shown below considers the logistic population model and how you can solve this equation with an analytical approach (using the DSolve function in Mathematica) giving an explicit solution expressing the population size as a function of time. Also shown is a numerical approach (using the NDSolve function in

Mathematica) that gives a numerical solution that is used here for a plot of the population size with time.

The logistic ODE:dn/dt=rn (1-n/k)

In[ ]:=
```
Clear[k, r, N0]
```

Enter the ODE (as dn/dt=f(n))

In[ ]:=
```
f[n_] := r n (1 - n / k)
```

Find the steady-states

In[ ]:=
```
Solve[f[n] == 0, n]
```

Out[ ]=
```
{{n → 0}, {n → k}}
```

Directly solve ODE using analytical solution

In[ ]:=
```
Clear[r, k, n0];
```

In[ ]:=
```
sol1 = DSolve[{n'[t] == f[n[t]], n[0] == n0}, n[t], t]
```

Solve: Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information.

Out[ ]=

$$\left\{\left\{n[t] \to \frac{i^{r t}\, k\, n0}{k - n0 + i^{r t}\, n0}\right\}\right\}$$

In[ ]:=
```
g[t_] := Evaluate[n[t] /. sol1]
```

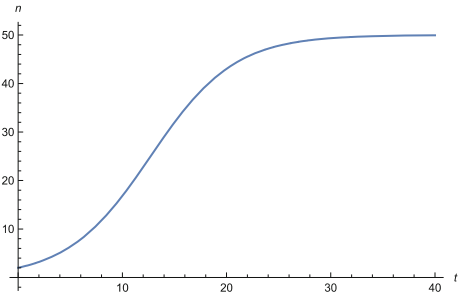
In[ ]:=
```
n0 = 2;
```

In[ ]:=
```
r = 0.25;
```

In[ ]:=
```
k = 50;
```

In[ ]:=
```
Plot[g[t], {t, 0, 40}, AxesLabel→{t, n}]
```

Out[ ]=



Directly solve ODE using numerical solution

In[ ]:=
```
sol2 =NDSolve[{n'[t]== f[n[t]], n[0]==n0}, n, {t, 0, 40}]
```

Out[ ]=

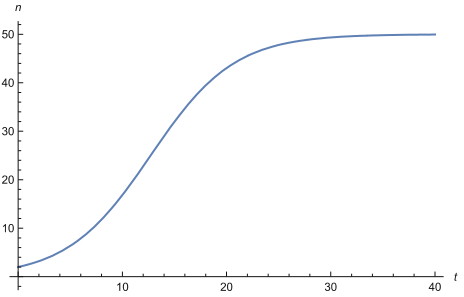
```
~~ n  InterpolatingFunction[x]
```


```
. .
```

In[ ]:=
```
h[t_] :=Evaluate[n[t] /. sol2]
```

In[ ]:=
```
Plot[h[t], {t, 0, 40}, AxesLabel→{t, n}]
```

Out[ ]=



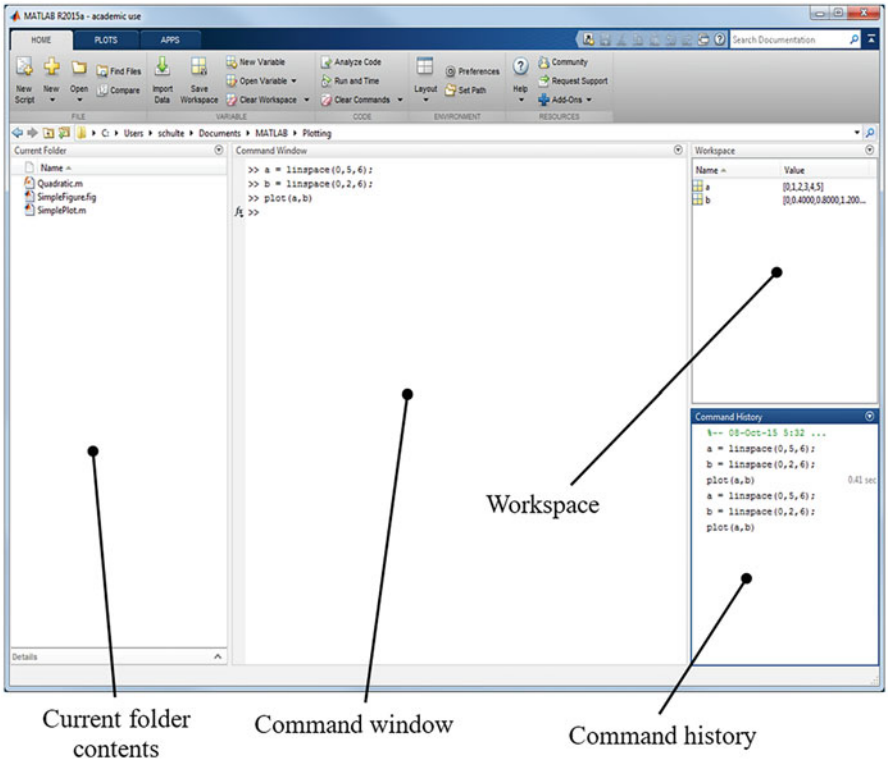
Appendix 4: MATLAB Introductory Tutorial

MATLAB is one of the programs that we will be using for numerical computations. This tutorial is an introduction to some of the features of the program. You should follow through the tutorial while actually in front of the computer running the program so that you can enter commands and see the results for yourself.

1. The Main MATLAB Window

Below is a view of the MATLAB window. Each component is actually a separate window that can be undocked from the main window frame and moved around on your screen, but by default they are usually docked and arranged as shown.

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6



The Command window is where you will type in many of the statements or calculations shown below. You can use the arrow keys on your keyboard to scroll back through previously entered statements.

The Current folder shows a list of files present in the folder on your computer as shown in the line above the window.

The Workspace shows a list of variables which contain data that you are using. You can double-click on these to open a spreadsheet view of the data and edit those values.

The Command history shows a list of commands that have been entered in the command window. You can double-click on these to re-enter and execute them to avoid having to retype previous commands.

2. Entering Simple Commands

Try entering this statement into the Command window (type as shown followed by "Enter," the >> prompt is supplied by MATLAB):

```
>> x = 1 + 2
```

You should see the following in the command window below the line that you just entered:

```
x =

 3
```

Also notice that the Workspace window now contains an entry for this new variable. If you double-click on that entry in the Workspace, you will see a spreadsheet view containing only one cell for that value of "3". If you entered that command above with a semicolon at the end, MATLAB would have suppressed the output but still entered the new value for x.

MATLAB contains many built-in functions such as sine and cosine:

```
>> x = sin(pi/2)
```

You should see the resulting value of 1. Notice that "pi" is available as a constant. Like many computing environments, trig functions here assume radian measure and not degree measure. You can also display the value of a variable by entering its name alone:

```
>> x

x =
 1
```

- Upper and Lower Case

An important point to remember with names in MATLAB (and many other computational programs) is that MATLAB is case-sensitive. This means that a variable called x is not the same variable as one called X. The same case-sensitivity applies to the function names that we will be using below.

- Numerical Formats

MATLAB can store numbers as integers, real numbers, or complex numbers. If you enter a real number without a decimal, it will be stored as an exact integer. Numbers entered with a decimal will be displayed in a long or short format. Try the statements:

```
>> format short
>> x = 1.23456

x =
 1.2346
```

```
>> format long
>> x

x = 1.234560000000000
```

Notice how the number of digits displayed depends on the format that you specified.

Here are the basic math operations that can be entered:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^
Negation	-

Note that the "-" symbol is used to indicate subtraction but also for indicating negative numbers. The exponentiation symbol is used for cases like:

x = 2 ^ 3 meaning $x = 2^3$

- Operator Precedence or Priority

Math operations in computers generally follow this priority:

( )
^, - (negation)
*, /
+, -

Why is this important? Suppose you see in a textbook:

$$2x + 3$$

We can enter this in a computer as 2 * x + 3 and get the expected answer because multiplication has a higher priority than addition. But in this case:

$$\frac{2 + x}{y}$$

you cannot enter 2 + x/y because division has a higher priority than addition. You would need to use parentheses because they have a still higher priority and will force the addition to be done first before the division: (2 + x)/y.

- The Help System

You can access help in MATLAB through a variety of mechanisms. The little "question mark" icon near the top of the main window will start an online help

session. But you can also obtain help for specific commands or functions from the Command Window:

```
>> help sin
 sin Sine of argument in radians.
 sin(X) is the sine of the elements of X.

 See also asin, sind.

 Reference page in Help browser
 doc sin
```

As the output shown above suggests, you can get more extensive documentation by typing "doc sin". You can also get help for a specific function like "sin" by putting the command window cursor in the text "sin" and pressing the F1 key.

3. Arrays in MATLAB

One of the strengths of MATLAB is the ease in which it can work with arrays of numbers. One can think of an array as a table of numbers in rows and columns

If you enter

```
x = 2
```

a single value will be stored in the variable. But we can also store a whole set of values and MATLAB will treat variables as either scalar (single-valued) or as an array:

```
>> x = [1,2,3,4];

>> y = [0,1,2,3];
```

Try entering these with the semicolon so that output will be suppressed. Now enter just the variable names to see what has been stored:

```
>> x

x =
 1 2 3 4

>> y

y =
 0 1 2 3
```

These variables are sometimes thought of as "row vectors" because they can be represented as a single row of values.

Try entering:

```
>> r = [1;2;3;4]

r =
 1
 2
 3
 4
```

In this case, you have created a "column vector".

To see how MATLAB can work transparently with arrays, enter:

```
>> x + y

ans =

 1 3 5 7
```

Notice how all the elements of those two variables have been summed to give the results. Also notice that when you enter an operation like addition without specifying a variable in which to store the result (as in saying "r = x + y"), MATLAB will display the result and store the answer in a temporary variable named "ans".

Arrays can be two-dimensional, like a table with rows and columns:

```
>> x = [1,2,3;3,2,1]

x =

 1 2 3
 3 2 1

>> y = [0,1,2;1,0,2]

y =

 0 1 2
 1 0 2
```

Now you have created two 2D arrays for the variables x and y.

Now try adding x and y:

```
>> x + y
```

You should see that MATLAB has added the two arrays element-by-element.

You can try other operations like subtraction. But try multiplying x and y:

```
>> x * y
```

You will get an error message that may only make sense if you have learned some matrix algebra in a math course. By the rules of matrix algebra, when you multiply two matrices their matrix dimensions (numbers of rows and columns) must satisfy certain requirements. In this case, in order to be able to multiply x * y, the array x must have the same number of columns as the number of rows in array y. We can "transpose" the array y to show how this might work. Transposing is like flipping the rows and columns and the single quote symbol after the variable name applies this transpose operation. So try:

```
>> y'
```

and you will see the rows and columns of y flipped. Now you can try:

```
>> x * y'
```

and get a two-by-two array for the answer. We are leaving out the actual procedures for matrix multiplication as a topic for another math course.

- Element-wise Array Operations

Now for the above case with the original arrays x and y, you can multiply them element-wise (each array element multiplied by the corresponding element in the same row and column) by using a special symbol with a decimal in front of the multiplication symbol:

Try:

```
>> x .* y

ans =

 0 2 6
 3 0 2
```

Look back at the original arrays and see that this result makes sense.

4. Files in MATLAB – Creating and Using Scripts and Functions

- Scripts

All of the command window statements that you have been entering could be stored in a simple text file called a "script". In this way you could re-execute all these commands without having to retype them in the command window.

Place your mouse cursor in the "Current folder" window and right-click. Now move your mouse pointer over "New File" and look through the sub-menu and select "Script". A file will be created in that current folder with a default name that you would normally change to something meaningful. Do this to create a script file called "Script.m". The filename extension of "m" is associated with MATLAB and such files are often called "m-files".

Next double-click on this new file and an editor window will pop-up with a blank set of lines. Enter some commands, like:

```
% This is my new script
format long
x = sin(pi/4)
```

Push the "Save" button on the toolbar above. Now go back to the Command window and just type "Script". You should see the results of this script just as if they were entered in the command window. But now you could go back and edit the script and re-execute whatever commands it contained without having to retype the actual commands. This ability could be quite useful if you have scripts with many, many lines. You can also execute the script by selecting the "Run" button in the toolbar at the top of the editor window.

Note that lines that start with "%" are comment lines that are not processed by MATLAB.

- Functions

Another type of file that you can create is useful for creating "functions" in MATLAB. A MATLAB function is related to a function in the mathematical sense – a series of operations and so typically a MATLAB function is a set of operations. You "call" that function, passing it some data, and then the function returns some data. Try the following example of using functions:

Suppose we want to use a function that carries out:

$$y = 2x^2 + 1$$

Let's create a function called "Quad"

Go back to that "Current folder" window; right-click, but this time under "New File" select "Function". MATLAB will create a file. Rename the file "Quad.m". Common practice is to name the file the same as the name of the function within the file.

Double-click on the filename. This time there will be some default text present in the editor. The function is defined between the words "function" and "end" in the file. First, you will want to check that the function name is correct (after the equal sign in the first row).

Functions are usually supplied some data in the form of "input arguments" and return calculated results as "output arguments". In this case, we want to create a function for the equation above that expresses the variable y as a function of the variable x. So change the default "input_args" and the default "output_args" so that the function looks like this:

```
function [ y ] = Quad( x )
%QUAD Summary of this function goes here
%   Detailed explanation goes here


end
```

Now add the actual function definition is the space before the "end" of the function. Your final file text should look like this:

```
function [ y ] = Quad( x )
%QUAD Summary of this function goes here
%   Detailed explanation goes here

y = 2 * x^2 + 1;

end
```

Note that in general, the spaces added within the line starting with "y = " are optional and sometimes only added for readability.

Save the m-file that you just created. Now we can use that new function. Incidentally, one of the reasons that we might create a function within an m-file instead of directly in the command window is that those files can be saved and exchanged with other users. Also, they are simple text files that can be printed or viewed from outside of MATLAB.

Now return to the command window and enter:

```
>> Quad(2)
```

Here you have called your function, passed it the value of 2 and it should return the value 9 (2 times 2 squared plus 1).

Earlier, I said that MATLAB can work transparently with arrays as well as scalar variables:

Enter these values for x:

```
>> x = [1,2,3,4,5];
```

Now try using our Quad function to calculate the function values for all of these elements at once:

```
>> Quad(x)
```

Most likely this did not work and you received an error message. What's the problem? Go back and look at the file Quad.m in the editor. The function that you entered is trying to square the variable x. Works fine if x is a scalar quantity, but not an array. For array variables, squaring is similar to multiplying two arrays; they must have the proper dimensions. In this case the array must have the same number of rows as the number of columns. What you really want is an element-by-element multiplication. So edit your function, changing the line:

```
y = 2 * x ^ 2 + 1;
```

to:

```
y = 2 * x .^ 2 + 1;
```

where you are using the element wise form .^ instead of just ^. This is a case where the period has to be right next to the ^ without an extra space. Save the function and call it again from the command window:

```
>> Quad(x)
```

and now you should get those original five values applied to the function. It will still work with scalar forms of variables.

5. Drawing Graphs

There are several functions that can be used to draw 2D or 3D graphs. First, enter some data to be graphed:

```
>> x = [0,1,2,3,4];

>> y = [1,3,4,2,5];
```

Now create the plot with:

```
>> plot(x,y)
```

You should see a graph window pop-up.

There are many graph editing features that can be accessed from this graph window. Click on the "Tools" menu item at the top and select "Edit plot". Double click in the middle of the graph and the window will change to show a set of editing possibilities at the bottom. Try these out until you see how to:

- Add or change the labels on the x or y axis.
- Change the scaling of the x or y axes (uncheck or check the Auto box).
- Change the fonts used for labels
- Add a Title to the graph
- Change the line used in the graph: Double click on the actual line in the figure, now you can change the line style, thickness, or color. You can also add symbols of various sizes for each data point with the "Marker" settings

Using the "Insert" menu at the top of the window (or under "View" enabling the Plot edit toolbar), you can add many other objects to the figure including text and various types of lines or arrows.

Lastly, take a look under the "File" menu at the top. You can save the figure (by default given a .fig extension to the name). You can also print the figure.

If you still have that earlier function called "Quad" showing in the Current folder list, try this command to draw the function:

`>> ezplot(@Quad)` or you can use this form: `>> ezplot('Quad')`

You can also draw a similar plot without creating the function. Try this:

```
>> ezplot('2*x^2+1')
```

6. Using Built-in Functions to Solve ODEs

MATLAB has several built-in functions that can be used to solve an ODE or system of ODEs by numerical methods that are expressed as an IVP (Initial Value Problem). In general, these equations are of the form:

$$\frac{dy}{dt} = f(t, y) \quad y(t_0) = y_0$$

Solving this ODE or system of ODEs numerically means to determine numerical values of y for any future time t. The following examples utilize the MATLAB function "ode45".

Typically, there are two steps in this process:

(a) Create a MATLAB function in an m-file which contains the ODE.
(b) Enter the command statements to set up data, call the ode45 function, and then plot the results.

Example 1 The simple exponential growth model

This population model is described by:

$$\frac{dN}{dt} = rN$$

First, create a function in an m-file with the file name "expODE.m". Edit the function so that it looks like this:

```
function [ dndt ] = expODE( t,n )
%expODE Simple exponential growth model
%   Detailed explanation goes here
r = 0.5;
dndt = r * n;
end
```

Here are a few important points about this function file:

- You can pick other names for the function (here named “expODE”) and variables as long as you are consistent. This means that if you want to use something other than "dndt" you have to be consistent in both places that the variable appears.
- You could also pick different names for the variables t and n, although in this case we do not use the t as part of the ODE definition because the ODE is autonomous and the right-hand side does not depend on t.
- Ordinarily, the name of the function and the name of the m-file should be the same.

- Also, MATLAB is case-sensitive, so these names should match with respect to upper and lower case letters. In general, you have to be consistent with names regarding upper and lower case.
- The statements following the "%" are comments that are not executed.

Now enter the following statements in the MATLAB command window:

```
>> tspan = [0 5];
>> nzero = 2;
>> [t n] = ode45(@expODE, tspan, nzero);
>> plot(t,n)
```

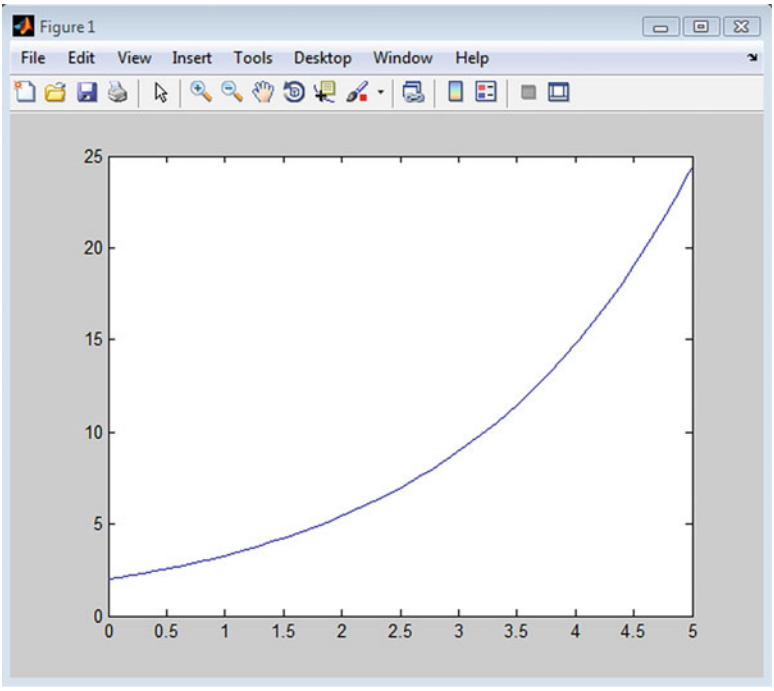
The variable "tspan" now contains the start and final times for which we want a solution of the ODE.

The variable "nzero" contains the initial value for the population size at time zero.

The ode45 function is called and we pass it the function handle for our model function, the time range, and the initial value. The function could also be specified like this:

```
>> [t n] = ode45('expODE', tspan, nzero);
```

You should see a plot like this:



Notice that the curve drawn in the figure looks fairly smooth. This is because the MATLAB ode45 function has chosen small enough time steps to maintain an accurate solution and also allow us to see a smooth curve.

You are strongly advised to play with some of the plot editing tools described in Part 1 of the tutorial at the bottom of page 9.

Next, let's examine the actual solution. Back at the MATLAB command window; display a table of solution values by typing:

```
>> [t n]
```

You will see numbers like this (with some rows left out in the middle):

```
0 2.0000
0.1005 2.1030
0.2010 2.2114
0.3014 2.3253
0.4019 2.4451
0.5269 2.6028
0.6519 2.7707

--- ---

4.9019 23.1988
4.9264 23.4850
4.9510 23.7747
4.9755 24.0681
5.0000 24.3650
```

Notice how the time steps at the beginning near time zero (left column) are further apart in time than the time steps at the end of the table. Looking at the graph, the solution curve is steeper at the later time values. The ODE solver (ode45) is an "adaptive" solver that picks smaller step sizes where the solution is changing more rapidly.

If you want to specify exactly the values of time where your solution will be provided, you can use one of these statements for the variable tspan:

```
>> tspan = [0 1 2 3 4 5];
>> tspan = linspace(0,5,11);
```

The first alternate choice will provide for solutions at the six time steps shown. The second alternate choice will provide for solutions at 11 points ranging from 0 to 5. In either case, the ODE solver will still use an adaptive time step process but the resulting numerical data that are returned will include only the requested time steps.

Example 2 Competition model of two species.

The MATLAB functions for solving ODEs, like the ode45 function shown above, can also solve systems of equations. In considering this example, remember that variables in MATLAB can be scalar quantities, but also vector or array quantities.

Suppose that we have two groups of organisms (called N_1 and N_2) that are competing for resources. Our model of competition could be written as:

$$\frac{dN_1}{dt} = \alpha N_1 - \beta N_1 N_2$$
$$\frac{dN_2}{dt} = \gamma N_2 - \delta N_1 N_2$$

Solve this model by setting up an m-file with the following function:

```
function [ nprime ] = Competition( t, n )
%COMPETITION Summary of this function goes here
%   Detailed explanation goes here
alpha = 0.11;
beta = 0.01;
gamma = 0.1;
delta = 0.01;
nprime = [alpha*n(1) - beta*n(1)*n(2);...
          gamma*n(2) - delta*n(1)*n(2)];
end
```

Note that "..." at the end of a line is a "continuation" which extends a function line beyond the single line and can be used to make the text more readable.

Recall that the model is a pair of ODEs. The variable n is a column vector with population sizes for N_1 and N_2, and so the function above must specify which row in n (which population) is being used in a calculation. Similarly, the variable "nprime" which is returned by the function is a column vector containing dN_1/dt in the first row and dN_2/dt in the second row.

An alternate approach for the above function could use (substituting for the line "nprime = ..."):

```
nprime(1,1) = alpha*n(1) - beta*n(1)*n(2);
nprime(2,1) = gamma*n(2) - delta*n(1)*n(2);
```

Another alternate approach could use (again, substituting for the line "nprime = ..."):

```
nprime = zeros(2,1);
nprime(1) = alpha*n(1) - beta*n(1)*n(2);
nprime(2) = gamma*n(2) - delta*n(1)*n(2);
```

As noted above, "nprime" must be a column vector and so these alternates must either directly specify the column number (as in the first alternate) or first setup "nprime" as a vector with two rows and one column by use of the "zeros" function, which just fills an array with zeros.

Solve the model with the following command window statements:

```
>> tspan = [0 30];
>> nzero = [5; 5];
>> [t n] = ode45(@Competition, tspan, nzero);
```

Next, display a graph of the two populations as a function of time:

```
>> plot(t,n)
```

Remember that "n" is an array which contains the sizes of both populations in the first and second columns of that array, and so the above statement will draw both curves in the plot.

Lastly, you can display the solution as a phase-plane diagram showing N_1 versus N_2:

```
>> plot(n(:,1), n(:,2))
```

Here, the " : " symbol effectively means "all rows" in a particular column of the array.

Your plots should look like those shown below (Figs. A.4.1 and A.4.2), except that in this case some additions were made using the MATLAB plot editing tools. You can also use the following statements after the plot function:

```
>> xlabel('Population size')
>> ylabel('Time')
>> legend('N_1', N_2')
```

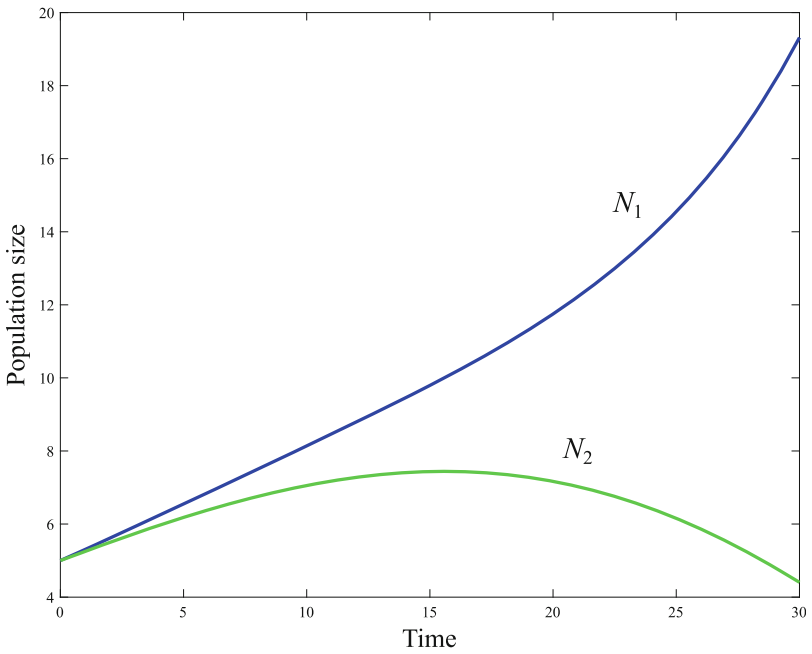


Fig. A.4.1 Plot showing the sizes of the two populations versus time

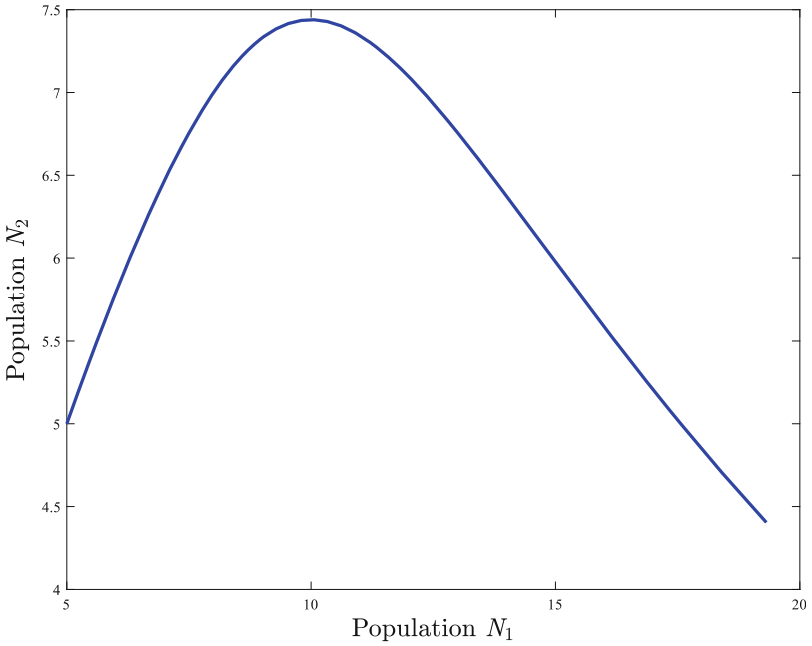


Fig. A.4.2 Plot showing the two population sizes in a phase-plane diagram

Appendix 5: Introduction to Python Programming

Python programming has been growing in popularity and our intention here is to provide a brief introduction to this system. The chapters in the text will show examples of using Python for particular models. Our emphasis here is to show how Python can be used for some of the numerical methods used in solving our models. The Python programming language is a full language in the sense that, like the C language, it provides features for the general writing of programs such as declaring variables, inputting and outputting data, and program flow with similar statements such as "for" loops and conditional execution of statements with the "if" construct. Our purpose in this introduction to the language is to focus more on the numerical method functions that are provided with Python add-ins such as the "numpy" and "scipy" components.

Python is a freely available programming environment. There are several complete texts that introduce this language, such as Programming for Computations – Python by Svein Linge and Hans Petter Langtangen, that would be useful for a more complete introduction. As suggested in the text above, we recommend installing the necessary software using a free Python distribution produced by Continuum Analytics, called Anaconda and available from http://continuum.io/downloads in versions for MS Windows, Apple Mac, and Linux-based computers. This package includes the basic Python components plus about 200 Python add-ins. Also included is a program called Spyder, which provides an integrated development environment that we find very useful for writing and executing Python programs. Python programs can also be written with a simple text editor and then run from a terminal window.

Assuming you have installed Python using the Anaconda distribution, a simple "Hello" program can be written by entering this text in the editor window in Spyder:

```
Print ("Hello!")
```

Then select Run from the Run menu (or click on the triangular run symbol on the toolbar). The program output (just the word Hello) will appear in the console window at the lower right.

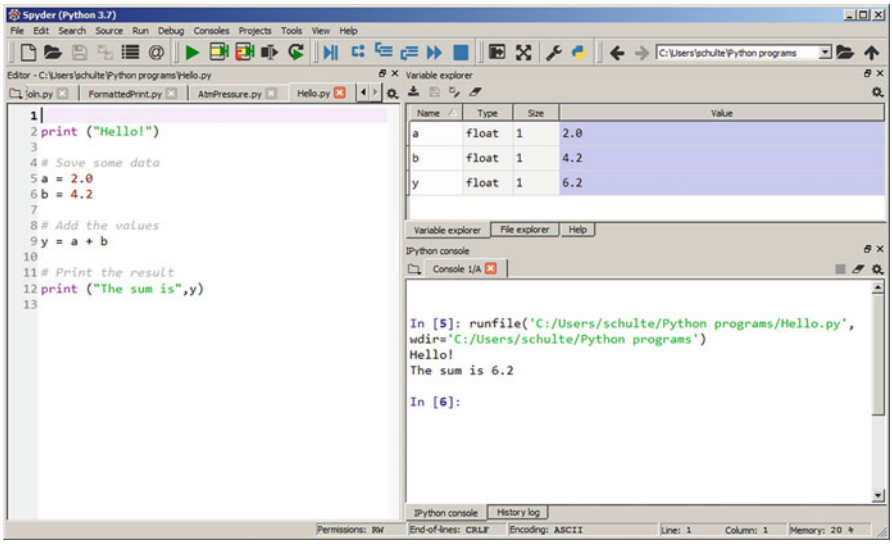
D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6

A slightly more involved program might be entered as:

```
print ("Hello!")

# Save some data
a = 2.0
b = 4.2

# Add the values
y = a + b

# Print the result
print ("The sum is",y)
```

Note that text after the # character is a comment ignored by Python but useful for us as readers.

A view of the Spyder window at this point is shown here:



The editor window is on the left, the program output is available (along with messages about errors that might be found in the program) at the lower right. The upper right can show a list of program variables, the files in the current directory, or the help system depending on the tab selected below this view.

As noted earlier, our goal in introducing and using the Python programming language relates to solving the models that are presented in our text. For a fuller introduction to Python as a programming language, the reader may wish to consult a text such as A Primer on Scientific Programming with Python (Hans Petter Langtangen) or numerous tutorials available online. One of our goals in this text will be to introduce a number of models based on an ordinary differential equation

(ODE) or system of ODEs. Here is an example of a Python program that uses the function solve_ivp from the Python scipy package to provide a numerical solution to the logistic growth ODE model (see Eq. 7.1, Chap. 7). The line numbers at the left are not typed when entering the program, they are shown here for reference.

```
1 # Logistic growth model
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.integrate import solve_ivp
5
6 # n - population size
7 def LogisticModel(t, n):
8 r = 1.1
9 k = 100
10 return r*n*(1-n/k)
11
12 n0 = 2.0
13 t0 = 0
14 tf = 10
15 steps = 10*tf
16 sol = solve_ivp(LogisticModel, [t0, tf], [n0], method='RK45',
17 rtol=1e-6, atol=1e-9, dense_output=True)
18 t = np.linspace(t0,tf,steps+1)
19 n = sol.sol(t)
20
21 # Display data
22 #print (np.transpose([t,n[0]]))
23
24 # plot results
25 plt.plot(t, n[0])
26 plt.xlabel('Time')
27 plt.ylabel('Population size')
28 plt.savefig('Logistic.svg')
29 plt.show()
30 # Save results to a file
31 f = open('Logistic.txt', "w")
32 for i in range(0, steps+1, 1):
33 print("%5.1f, %6.2f" % (t[i], n[0][i]), file=f)
34 f.close()
```

The statements in lines 2–4 are for including needed components such as numpy for numerical functions, scipy for numerical methods, and matplotlib for drawing graphs. Again, note that statements starting with # are comments that are included only for the reader of the program and are not acted upon.

The model to be solved in this case is the logistic growth ODE:

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right)$$

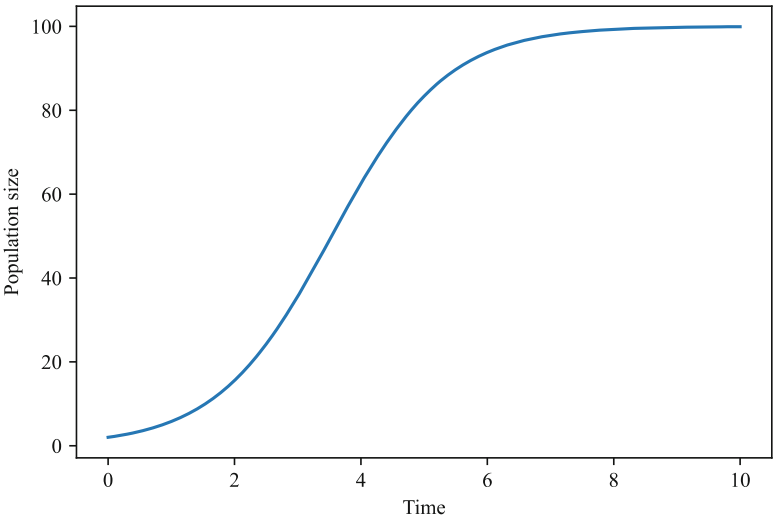


Fig. A.5.1 Plot of the python solution for the logistic population model

This equation is incorporated into the program function "LogisticModel" in lines 7–10. The model variables to be passed to the function are shown as t and n. Next, the values for parameters r and k are set in lines 8 and 9. The right-hand side of the ODE is entered in line 10.

Next the program sets an initial value for the dependent variable n at time zero, here called n0. For our solution to this model, we want values of n at a series of time points set in line 17, using the linspace function (part of numpy). We specify a set of 101 values linearly spaced between 0 and 10.

The actual numerical solver for this ODE, solve_ivp, is called in line 16 and passed the model, the range of time values desired for the solution, the initial value of n, the ODE solver method selected, and a setting asking for a continuous solution.

Line 18 sets a range of time values that we want for the solution and the number of steps for those time values. Line 19 extracts the values of n from the solution (saved as sol). The remainder of the program (lines 25–28) plots the solution as values of t and n and also saves a scalable vector graphics (svg) form of the plot. Lines 31–34 save the data to a text file that could be used as an input into another graphics program.

The graphical results should look like the plot shown in Fig. A.5.1. Note that one can set the font used in the figure (Times New Roman in Fig. A.5.1) with this statement to be entered before the plt.plot statement in the program:

```
plt.rcParams['font.family']='Times New Roman'
```

Index

D. G. Costa, P. J. Schulte, *An Invitation to Mathematical Biology*,
https://doi.org/10.1007/978-3-031-40258-6