Building New Models: Rethinking and Revising ODE Model Assumptions

Paul J. Hurtado

Abstract Ordinary Differential Equation (ODE) models are ubiquitous throughout the sciences, and form the backbone of the branch of mathematics known as applied dynamical systems. However, despite their utility and their analytical and computational tractability, modelers must make certain simplifying assumptions when modeling a system using ODEs. Relaxing (or otherwise changing) these assumptions may lead to the derivation of new ODE or non-ODE models and sometimes these new models can yield results that differ meaningfully in the context of a given application. The goal of this chapter is to explore some approaches to relaxing these ODE model assumptions to derive models which can then be analyzed in ways that parallel or build upon an existing ODE model analysis. To accomplish this, the first part of this chapter (Sect. 2) reviews some common methods for the application and analysis of ODE models. The next section (Sect. 3) explores various ways of deriving new models by modifying the assumptions of existing ODE models. This allows investigators to explore the extent to which ODE model results are robust to changes in model assumptions, and to answer questions that are better addressed using non-ODE models. The last part of this chapter suggests a few specific project ideas (Sect. 4) and encourages undergraduate researchers to share their results through presentations and publications (Sect. 5).

Suggested Prerequisites *Knowledge of topics from a standard calculus sequence, basic linear algebra, an introductory course in probability, and some familiarity with very basic differential equations models are assumed. Readers would also benefit from having some basic programming experience (in R, Python,* MATLAB*, or a similar high-level scientific programming language),*

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-33645-5_1) contains supplementary material, which is available to authorized users.

P. J. Hurtado (✉)
Department of Mathematics and Statistics, University of Nevada, Reno (UNR), Reno, NV, USA
e-mail: phurtado@unr.edu

H. Callender Highlander et al. (eds.), *An Introduction to Undergraduate Research in Computational and Mathematical Biology*, Foundations for Undergraduate Research in Mathematics, https://doi.org/10.1007/978-3-030-33645-5_1

and familiarity with calculus-based probability, stochastic processes (e.g., homogeneous and non-homogeneous Poisson processes), statistical theory of estimators, biology, or experience with mathematical modeling in an applied context.

1 Introduction

Ordinary Differential Equations (ODEs) models are ubiquitous throughout the sciences, and are a cornerstone of the branch of applied mathematics known as applied dynamical systems. Examples include the exponential and logistic growth models, Newton's laws of motion, Ohm's law, chemical kinetics models, infectious disease models for the control and treatment of diseases ranging from the common cold to HIV and Ebola, and models for large scale ecosystem dynamics.

The utility of ODEs as mathematical models is partly a reflection of their analytical and computational tractability, and also the relative ease of formulating ODE model equations from assumptions about the system being modeled. Modelers also have access to a well-developed set of mathematical approaches and computational tools for analyzing ODEs [4, 10, 39, 41, 45, 50, 55, 56, 84, 94–96, 119, 126, 127, 149, 155, 160, 165]. If there is a practically important or scientifically interesting real-world system that changes over time, odds are very good that someone has modeled it with one or more ODEs. However, despite their utility, modelers must also make certain simplifying assumptions when using ODEs to model a real-world system (e.g., that systems are well mixed, individuals behave identically, etc.), and sometimes alternative assumptions can lead to different outcomes that are very relevant in the context of the motivating application.

The goal of this chapter is to explore some approaches to relaxing these ODE model assumptions to come up with new models that can then be used in ways that parallel or build upon an existing ODE model application. This allows investigators to explore the extent to which results obtained from the analysis of an ODE model are robust to changes in certain model assumptions, and/or to address a broader range of questions, some of which might be better answered through the analysis of a similar ODE or non-ODE model (e.g., using a stochastic model of population growth and decline to assess the distribution of time to extinction under different population management scenarios [105]). Undergraduate students reading this chapter are encouraged to initially focus on understanding the broader context of the technical details presented in this chapter, and then later attempt to more fully understand specific details, with the help of a research mentor or advisor, as they work on a project.

A great recipe for an undergraduate research project is to (a) find an interesting paper in a reputable, peer-reviewed journal that includes some analysis of an ODE model, (b) repeat one or more analyses to confirm a specific published result, and then (c) answer similar questions via the analysis of a new model derived by modifying one or more assumptions of that original ODE model. Mathematically, these projects often go one of two ways: Changing ODE model assumptions will sometimes result in a new set of ODEs, while changing other assumptions can lead

to the formulation of a new non-ODE model, e.g., a model that is defined using partial differential equations (PDEs), stochastic differential equations (SDEs), or discrete-time difference equations.

This chapter is divided into two main sections. First, we review some common approaches to using ODEs models in Sect. 2, including methods of mathematical, computational, and statistical analysis. Second, in Sect. 3, we address some of the different ways that ODE model assumptions can be modified to derive new ODE or non-ODE models, and additional project suggestions are presented in Sect. 4.

Examples are used throughout to illustrate the process of deriving a new model, and some guidance is provided for analyzing or simulating these derived models. Some examples include computer code written in the R programming language [137]. R is a popular, free computing platform that is widely used, easy to install, and well supported in terms of free resources to get started programming in R. For readers that have no prior programming experience, or want to learn R, relevant resources for getting started with R are provided in the appendix. To help readers who are proficient in another language (e.g., Python, MATLAB, or Mathematica) but who are unfamiliar with R, the R code in this chapter is annotated with comments so that it can be more easily ported to other languages.

Lastly, many sections below include *exercises* that the readers should work through as they read the sections. These are intended to help solidify the topics being discussed, and should not require consulting outside resources. The *challenge problems* are more involved, and are intended to push the reader to dig deeper into the topic, or perhaps inspire a research project, by exploring additional resources including software tools and other peer-reviewed publications.

2 Techniques for Analyzing ODE Models

Before we explore some of the ways in which we can alter ODE model assumptions to obtain new (sometimes non-ODE) models, it is helpful to consider the kinds of questions investigators address using ODE models, and some related methods of ODE model analysis. This is done in Sects. 2.2–2.7. But first, for context, it helps to review how ODEs can often (but not always, e.g., see Newton's laws of motion) be viewed as *mean field* models of a stochastic process, as this is often the natural context in which to critically evaluate and revise the assumptions of biological models.

2.1 ODEs as Mean Field Models

First order systems of ODEs are systems of equations that can be written as

$$\frac{d\mathbf{x}}{dt} = f(t, \mathbf{x}, \theta), \quad \mathbf{x}(0) = \mathbf{x}_0, \tag{1}$$

where the *state variable* vector $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))^\mathsf{T} \in \mathbb{R}^n$, the *initial condition* vector $\mathbf{x}_0 \in \mathbb{R}^n$, the parameter vector $\theta = (\theta_1, \ldots, \theta_p) \in \mathbb{R}^p$, and the function[1] $f : \mathbb{R}^n \mapsto \mathbb{R}^n$. We call such systems *autonomous* when the right hand side f only depends on time through its dependence on the state variables, i.e., when the right side of the system of equations can be written $f(\mathbf{x}, \theta)$. While these are deterministic equations, in the context of mathematical models we often think of ODEs as *mean field* approximations of an underlying stochastic model. More specifically, biological and other physical systems made up of large numbers of discrete individuals can often be modeled as continuous-time, discrete-state stochastic models (e.g., where the model tracks $X(t)$, the integer number of individuals alive at time t, which changes stochastically to reflect births and deaths in the population). Thus, mean field models attempt to capture a particular kind of average behavior of a stochastic model.

As detailed below, mean field ODEs that approximate these stochastic models can often be derived from a stochastic model in two main steps: First, the continuous-time model is approximated with a discrete-time model (step size Δt), and the stochastic rule that describes how state variables change over one time step is replaced with a deterministic rule obtained by simply averaging over that stochastic model. Second, letting $\Delta t \to 0$, the discrete-time model converges to an ODE (e.g., see [5, 20, 92] and the similar approach to derive stochastic differential equations (SDEs) in [3]).

For example, consider a simple exponential growth model of the growing number of bacteria, $x(t)$, with *per capita* growth rate r, in a Petri dish that was inoculated with $x_0 > 0$ bacteria at time $t = 0$, given by

$$\frac{dx(t)}{dt} = r\, x(t), \quad x(0) = x_0. \tag{2}$$

Note that this is a continuous-state (i.e., $x(t) \in \mathbb{R}$) deterministic model, despite the fact that we envision the actual biological process as a integer-valued number of bacterial cells dividing and dying stochastically over time. *This dual perspective is very important in that it provides a foundation for thinking about model assumptions, and in some cases a starting point for deriving new models.*

To clarify this link between stochastic and deterministic approaches, consider the following derivation of Eq. (2) from the following stochastic model[2] of bacterial population growth. Let $X(t)$ be the number of cells at time t. Assume that, over a sufficiently small time period of duration $\Delta t > 0$, the probability p of each cell dividing to give rise to a new cell is assumed to be identical for all cells, independent of time, and given by

[1] This function is often referred to as the *right hand side*, abbreviated RHS, of the ODE.

[2] It is worth mentioning here that multiple different stochastic models can yield the same mean field ODE model.

$$p = r\,\Delta t, \quad r > 0. \tag{3}$$

A *stochastic, discrete-time map* can be used to model this process, as follows. If there are $X(t)$ cells at time t, then Δt time units later, at time $t + \Delta t$, there are

$$X(t + \Delta t) = X(t) + U(X(t), r\,\Delta t) \tag{4}$$

cells present, where $U(n, p)$ is a binomially distributed random variable where for each of the $n = X(t)$ cells there is a probability $p = r\,\Delta t$ that a given cell will give rise to a new cell by time $t + \Delta t$.

To derive the ODE model of exponential growth Eq. (2) as an approximation of the above stochastic model, we follow the steps described above: The discrete-time stochastic model is approximated by a *mean field*[3] deterministic model, and then we let the time step $\Delta t \to 0$ to yield the desired ODE.

In this first step, the number of new cells U is replaced by the expected value $E(U)$ as follows. Assume that the number of cells $X(t)$ is sufficiently large, and that we use an appropriately small choice of Δt so that U is well approximated by a binomial distribution. The expected value of this binomial random variable, where $n = X(t)$ trials and probability of success $p = r\,\Delta t$, is the product $n\,p = X(t)\,r\,\Delta t$, thus we make the approximation $U(X(t), r\,\Delta t) \approx E(U(n = X(t), p = r\,\Delta t)) = X(t)\,r\,\Delta t$. This yields the *mean field* discrete-time model

$$x(t + \Delta t) = x(t) + r\,x(t)\Delta t. \tag{5}$$

Note that we distinguish between these new state variable quantities $x(t)$ (which are $\mathbb{R}$-valued expectations) and their integer-valued counterparts $X(t)$, even though we interpret both as representing the number of bacteria present at time t. Lastly, rearranging this deterministic equation, and taking the limit as $\Delta t \to 0$, yields

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} \quad \to \quad \frac{dx(t)}{dt} = r\,x(t). \tag{6}$$

There are a few remarks on the above derivation that are worth mentioning. First, a more careful derivation, that explicitly tracks error terms in the above approximations, would yield the same mean field result, but those details were omitted for brevity and clarity (for a more detailed example, see [92]). Second, mean field models locally average how state variable change in time, and do not average

[3]The deterministic equation derived in this first step is called a *mean field* model because, for a given point $X(t)$ in state space, we take the distribution of possible steps the system might take on the next time step, and define a new model by replacing that random quantity with its mean—in this case, the mean of the binomial random variable $U(n = X(t), p = r\Delta t)$. One could visualize this on a grid of state space values with vectors pointing from $x(t)$ to $x(t + \Delta t)$, forming a vector field.

over full trajectories of a stochastic model. Thus, mean field model solutions can differ markedly from the average over multiple stochastic model trajectories. Third, for readers who are familiar with Poisson processes [145], this is a very useful context for thinking about these underlying stochastic processes associated with a mean field ODE. In fact, the approach above (i.e., discretizing time to derive a mean field model) very much parallels the typical approach to deriving properties of homogeneous and non-homogeneous Poisson processes by discretizing time then taking the continuum limit (e.g., see the book [40], the standard derivation of the binomial approximation of the Poisson distribution [106], and [92] for a some relevant aspects of Poisson processes). In the context of Poisson processes, what has been assumed in the stochastic model above can be rephrased as follows: each individual cell replicates itself according to a Poisson process with rate r where the events generated by the Poisson process mark the times at which replication events occur.

Exercise 1 It is often appropriate to make simplifying assumptions when formulating mathematical models of real-world phenomena. The art of modeling is striking a balance between simplicity, mathematical tractability, and capturing the important mechanism that drive real-world processes. Often, these simplifying assumptions are not explicitly stated as an exhaustive list as part of the model description, for example, above we have assumed all cells are basically identical in terms of their replication rates. What other *implicit* assumptions have been made in the above example? *Hint: What was assumed about resource limitation, cell mortality, whether or not cells influence one another in terms of death and cell division, and so on?*

Challenge Problem 1 Formulate a similar stochastic model to Eq. (4), but replace replication rate symbol r with b and include a mortality rate d. Show this yields an identical mean field model to Eq. (2) where $r = b - d$. *Hint: Show* $E(\Delta X) \approx X\, b\Delta t - X\, d\Delta t$.

This ability to reframe the stochastic model assumptions in terms of Poisson processes is important because the ODE model Eq. (2) reflects these underlying exponential distributions through the per capita growth rate r. Importantly, this pattern generalizes: when the appearance or loss of individuals in a given state follows *i.i.d.* Poisson process first event time distributions with rate $r(t)$ (i.e., exponential distributions if r is constant) then the per capita loss or growth rates in corresponding mean field ODEs are the coefficients $-r(t)$ or $r(t)$, respectively. This is illustrated by the following theorem, which is stated here without a proof (the proof follows from more general theorems in [92]).

Theorem 1 *Consider a continuous-time, stochastic, state transition model in which individuals spend independent and identically distributed amounts of time in a given state (X) and then transition to subsequent state Y. Suppose that individual dwell times are either exponentially distributed with constant rate r, or (more generally) follow the first event time distribution under a non-homogeneous Poisson process with rate* $r(t)$. *Further assume individuals enter X at rate* $\Lambda_X(t)$ *and the dwell time in state Y follows the first event time under a (non-homogeneous) Poisson process*

with rate $\mu(t)$. Then, if $x(t)$ and $y(t)$ are the expected number of individuals in each state X and Y, respectively, the mean field ODE model for this case is

$$\frac{dx}{dt} = \Lambda_X(t) - r(t)\,x(t) \tag{7a}$$

$$\frac{dy}{dt} = r(t)\,x(t) - \mu(t)\,y(t). \tag{7b}$$

To further illustrate how these underlying Poisson process rates are reflected in ODE model terms, consider the well-known SIR model of infectious disease transmission [98] (for more on SIR-type epidemic models, see [4, 7, 22, 23, 43, 98]). In this model, a closed population of size N is assumed to be made up of individuals who are either *susceptible* to an infectious disease, currently *infected* (and *infectious*), or *recovered* from infection and immune to reinfection by the pathogen. Accordingly, we let state variables $S(t)$, $I(t)$, and $R(t)$ correspond to the number of susceptible, infectious, and recovered individuals at time t. The SIR model is a simple ODE model of how these numbers of individuals in each state change over time, and is given by the equations

$$\frac{d}{dt}S(t) = -\lambda(t)\,S(t) \tag{8a}$$

$$\frac{d}{dt}I(t) = \lambda(t)\,S(t) - \gamma\,I(t) \tag{8b}$$

$$\frac{d}{dt}R(t) = \gamma\,I(t) \tag{8c}$$

where $\lambda(t) \equiv \beta\,I(t)$ is the per capita infection rate (also called the *force of infection* [7]), and γ is the per capita recovery rate. Note that, by the assumed dependence of $\lambda(t)$ on $I(t)$ this is a nonlinear system of ODEs. This model can also be viewed as the mean field model for an underlying stochastic state transition model of a large but finite number of individuals that each transition from state S to I following "event times" (the time spent in a given state; also called *dwell times* or *residence times*) that obey a non-homogeneous Poisson process first event time distribution with rate $\lambda(t)$. Each individual similarly transitions from I to R following a homogeneous Poisson process first event time distributions with rate γ, hence the time an individual spends in state I is exponentially distributed with mean $1/\gamma$ (see [98] for a derivation, and see [9, 13], and references therein for examples of the convergence of stochastic models to mean field ODEs).

Altering the assumptions of ODE models like the SIR model above is often best done when the modeler has adopted this mean field perspective. For example, if we would like to relax the assumption that individuals immediately become infectious and instead assume that they experience a *latent period* where they are infected but not yet infectious for a time period that is exponentially distributed with mean $1/\nu$, then we can introduce an *exposed* class and let $E(t)$ be the number in this class at time t. This yields the well-known SEIR model:

$$\frac{d}{dt}S(t) = -\lambda(t)\,S(t) \tag{9a}$$

$$\frac{d}{dt}E(t) = \lambda(t)\,S(t) - \nu\,E(t) \tag{9b}$$

$$\frac{d}{dt}I(t) = \nu\,E(t)) - \gamma\,I(t) \tag{9c}$$

$$\frac{d}{dt}R(t) = \gamma\,I(t). \tag{9d}$$

Exercise 2 Under the SIR model, the time spent infected and infectious is the same, and follows an exponential distribution with rate γ (i.e., with mean duration $1/\gamma$). This is not the case under the SEIR model. Describe and compare the mean time infected (i.e., between entering the exposed state and entering the recovered state) for both the SIR and SEIR models, in terms of the mean time spent in the exposed state and the mean time spent in the infectious state.

In summary, systems of ODEs that model individuals transitioning among different states are often best thought of as a mean field model for some underlying (often unspecified) continuous-time stochastic process. While in general it is true that multiple different stochastic models can yield the same mean field ODE model, it is still often very useful to associate a given ODE with the continuous-time stochastic state transition model defined in terms of Poisson processes where the Poisson process rates are given by the overall rates (i.e., ODE model terms) or the per capita rates (i.e., coefficients on those terms) that appear in the right hand side of the ODE model. We will explore one way of constructing such a model from a system of ODEs using the Stochastic Simulation Algorithm in Sect. 3.3.1, but throughout the remainder of this chapter we will often draw upon the intuition associated with this underlying stochastic model implied by a given mean field ODE model.

With this perspective in mind, we next review some standard ways of using ODE models in applications.

2.2 Equilibrium Stability Analysis

Many questions about dynamical systems can be answered by asking *What is the eventual state of* $\mathbf{x}(t)$ *as* $t \to \infty$? For example, Does the disease epidemic end with a local extinction of the pathogen, or does the disease become *endemic* and persist by establishing itself at low levels in the population? Will new harvest regulations in a fishery maintain a sustainable population of fish, or will the population be pushed to extinction? Sometimes knowing where a system is going helps inform the path it takes to get there, so these asymptotic results as $t \to \infty$ can also be useful for understanding short-term dynamics. Since many questions can be addressed

by mathematical and computational analyses that reveal the long-term behavior of a system, this is often a focus of an introductory course in mathematical biology [4, 50], and the main focus of courses in applied dynamical systems [10, 84, 160] and bifurcation theory [104, 165]. Here we review the basics of equilibrium stability analysis, and in the next section we discuss analyses that build on these concepts to further illuminate the behavior of ODE models in a dynamical systems context.

Let us revisit the simple exponential growth or decay model Eq. (2) where we assume a positive initial value, i.e., $x(0) = x_0 > 0$, and that $r \in \mathbb{R}$. Solutions to this ODE have the form

$$x(t) = x_0 e^{r t} \tag{10}$$

thus we can see that the sign of r determines whether solutions $x(t)$ diverge towards ∞ ($r > 0$) or converge towards zero ($r < 0$). Here we focus on solutions like the latter, where the long-term behavior of the system is to settle on a steady-state value, more commonly referred to as an *equilibrium point*.

Formally, an equilibrium point of a model $d\mathbf{x}/dt = f(\mathbf{x})$ is a state $\mathbf{x}_*$ such that $f(\mathbf{x}_*) = 0$. Starting at such points yields solutions that remain there for all time, which we refer to as *equilibrium solutions*.

An *equilibrium stability analysis* aims to accomplish two goals: First, to determine how many different *equilibrium points* exist and find expressions for those state values as a function of the model parameters, and second, to determine the *(local) asymptotic stability* of those equilibria. Identifying equilibria is straightforward: simply set the system of equations equal to zero, and solve for the state variable vector $\mathbf{x}_*$ that satisfies $f(\mathbf{x}_*) = 0$. The second step in a stability analyses requires us to first clarify what is meant by an equilibrium point being *asymptotically stable*.

In short, we call an equilibrium point *stable* if small perturbations away from that point in state space yield trajectories that converge back towards said equilibrium point, and *unstable* if small perturbations yield trajectories that diverge away from the equilibrium point. More precisely, we can use the formal definition of equilibrium stability given in [160]: If, for an equilibrium point $\mathbf{x}_*$, there is a $\delta > 0$ such that any trajectory $\mathbf{x}(t)$ that starts within a distance $< \delta$ from $\mathbf{x}_*$ eventually converges to $\mathbf{x}_*$, then we call $\mathbf{x}_*$ *attracting*. If for $\epsilon > 0$ there is a $\delta_\epsilon > 0$ such that whenever a trajectory $\mathbf{x}(t)$ starts closer than δ_ϵ to $\mathbf{x}_*$ the trajectory remains within a distance ϵ of $\mathbf{x}_*$, then we call $\mathbf{x}_*$ *Lyapunov stable*. If $\mathbf{x}_*$ is both attracting and Lyapunov stable, we say $\mathbf{x}_*$ is (locally) *asymptotically stable* and this is what is meant below when an equilibrium point $\mathbf{x}_*$ is called *stable*, i.e., for sufficiently small perturbations away from that point, trajectories $\mathbf{x}(t)$ stay within a small neighborhood of that point and asymptotically converge towards $\mathbf{x}_*$ (i.e., $\mathbf{x}(t) \to \mathbf{x}_*$) as $t \to \infty$.

In practice, determining the stability of an equilibrium point is fairly straightforward; however, having an intuitive understanding of why the standard approach

works requires some explanation. Because the *vector field*[4] $f(\mathbf{x})$ is usually smooth (i.e., because $f(\mathbf{x})$ is differentiable), zooming in very close to a given point (e.g., an equilibrium point) the vector field is increasingly well approximated by a linear vector field. Mathematically, this is a straightforward consequence of Taylor's Theorem: Recall that Taylor's Theorem for functions of one variable [112] states that if a function is differentiable at a then for x values in a neighborhood of a the function can be approximated by

$$f(x) \approx f(a) + f'(a)(x-a) + \ldots + \frac{1}{k!} f^{(k)}(a)(x-a)^k \tag{11}$$

where the difference between $f(x)$ and the approximation above vanishes as $x \to a$. This gives the linear approximation

$$f(x) \approx f(a) + f'(a)(x-a) \tag{12}$$

for values of x in a small neighborhood of a. Similarly, Taylor's Theorem for multivariate functions[5] $f : \mathbb{R}^n \to \mathbb{R}^n$ gives the linear approximation

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \mathbf{J}\,(\mathbf{x}-\mathbf{a}) \tag{13}$$

where $\mathbf{J}$ is the $n \times n$ *Jacobian* matrix whose entries are the partial derivatives of f with respect to each component of $\mathbf{x}$ evaluated at $\mathbf{x} = \mathbf{a}$, i.e., if $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))^{\mathsf{T}}$ then

$$J_{ij} = \left.\frac{\partial f_i}{\partial x_j}\right|_{\mathbf{x}=\mathbf{a}} \tag{14}$$

Since $f(\mathbf{x}_*) = 0$ (i.e., if $\mathbf{x}_*$ is an equilibrium point), Taylor's Theorem implies that the linear approximation near $\mathbf{x}_*$ is

$$\frac{d\mathbf{x}}{dt} \approx \mathbf{J}\,(\mathbf{x}-\mathbf{x}_*). \tag{15}$$

Moreover, if we let $\mathbf{u} = \mathbf{x} - \mathbf{x}_*$ be the perturbation of $\mathbf{x}$ away from $\mathbf{x}_*$, then

$$\frac{d\mathbf{u}}{dt} \approx \mathbf{J}\,\mathbf{u} \tag{16}$$

and thus the deviations ($\mathbf{u}$) of trajectories very near $\mathbf{x}_*$ are approximately solutions to this linear system of first order differential equations.

[4] For each point in state space, one can think of the derivative $\frac{d\mathbf{x}}{dt}$ as a vector pointing in a direction that is tangent to the trajectory passing through that point.

[5] See upper level dynamical systems texts and online resources for the Taylor expansion of multivariate functions.

The importance of this linear approximation in the context of determining the stability of an equilibrium point $\mathbf{x}_*$ is that the eigenvalues [159] of matrix $\mathbf{J}$ can tell us whether or not $\mathbf{x}_*$ is stable (e.g., see linear stability analysis sections in [45, 84, 160, 165] or related texts on nonlinear dynamics). Why? Small perturbations off of the equilibrium point will yield trajectories that are approximately $\mathbf{x}(t) \approx \mathbf{u}(t) + \mathbf{x}_*$ where $\mathbf{u}(t)$ is a solution to the above linearization Eq. (16). Importantly, solutions to linear ODEs can be written explicitly in terms of the eigenvalues and eigenvectors of the coefficient matrix $\mathbf{J}$ as shown for the 2-dimensional case in [45] and for higher dimensions in [84]. Specifically, these solutions look like sums of exponential growth and decay terms (sometimes multiplied by sine and cosine terms if $\mathbf{J}$ has complex eigenvalues [84]) along the different directions determined by the eigenvectors of $\mathbf{J}$, where the growth and decay rates along these eigenvectors are the *real parts of the corresponding eigenvalues of* $\mathbf{J}$. Thus, if the eigenvalues of $\mathbf{J}$ all have negative real parts, small perturbations off of $\mathbf{x}_*$ will exponentially decay back to that point, and if any eigenvalue has a positive real part, trajectories that start as perturbations off of $\mathbf{x}_*$ will diverge away from that point. The following stability theorem formally summarizes this intuition (for further details, see [84, 165] or similar texts):

Theorem 2 (Equilibrium Stability Criteria) *Suppose* $\mathbf{x}_*$ *is an equilibrium point of* $d\mathbf{x}/dt = f(\mathbf{x})$ *and* $\mathbf{J}$ *is the Jacobian evaluated at* $\mathbf{x}_*$. *Further assume the real parts of the eigenvalues of* $\mathbf{J}$ *are all non-zero (i.e.,* $Re(\lambda_i) \neq 0$ *for* $i = 1, \ldots, n$*). Then the equilibrium point is stable if all* $Re(\lambda_i) < 0$*, and unstable if any* $Re(\lambda_i) > 0$*. If any eigenvalues have* $Re(\lambda_i) = 0$ *the stability of* $\mathbf{x}_*$ *is determined by higher order terms in the Taylor expansion of* f *about* $\mathbf{x} = \mathbf{x}_*$*. If the system is one dimensional (i.e., if* $f : \mathbb{R} \to \mathbb{R}$*) then* x_* *is stable if* $f'(x_*)$ *is negative (and unstable if* $f'(x_*)$ *is positive).*

In computational applications, where it suffices to check stability of an equilibrium point for a specific set of parameter values, the above criteria are very practical. It is usually straightforward to compute the Jacobian and use standard approaches to find its eigenvalues (e.g., using the `eigen()` function in R). However, when looking to obtain analytical stability conditions, it is often cumbersome to find general expressions for eigenvalues and to find conditions under which they have a negative real part. This is especially true for higher dimensional nonlinear systems.

Importantly, we do not need to find these eigenvalues explicitly to determine if an equilibrium point is stable. The Routh–Hurwitz criteria (Theorem 3 below) provide a sometimes simpler way of checking whether or not all eigenvalues of Jacobian $\mathbf{J}$, evaluated at equilibrium point $\mathbf{x}_*$, have negative real part by checking a set of criteria that involve the coefficients of the Jacobian's characteristic polynomial[6] [159],

[6]This definition is used instead of the more common $p(\lambda) = \det(\mathbf{J} - \lambda\mathbf{I})$ since it ensures that the roots are the eigenvalues of $\mathbf{J}$ and that the polynomial is *monic, i.e., has a leading coefficient of 1 on the* λ^n *term.*

$$p(\lambda) = \det(\lambda \mathbf{I} - \mathbf{J}). \tag{17}$$

In short, since the eigenvalues are the roots of the characteristic polynomial of the Jacobian, satisfying the Routh–Hurwitz criteria implies that each of those eigenvalues has negative real part, and thus, the equilibrium point is stable. This is often the preferred approach when looking for general stability criteria (e.g., an inequality describing parameter value relationships that yield a stable equilibrium point) because in practice it typically yields meaningful stability criteria in fewer intermediate steps compared to finding the eigenvalues, and their real parts, directly.

The following statement of the Routh–Hurwitz criteria was adapted from Ch. 4 in [4][7] and pg 233–234 of [45] (see also [117]).

Theorem 3 (Routh–Hurwitz Criteria) *Consider the monic polynomial*

$$p(\lambda) = \lambda^n + a_1 \lambda^{n-1} + \cdots + a_{n-1} \lambda + a_n \tag{18}$$

with real coefficients a_i. All roots of Eq. (18) have negative real part if and only if all n principal minors of H_n (where $a_j = 0$ for $j > n$) have positive determinants.

$$H_n = \begin{bmatrix} a_1 & 1 & 0 & 0 & \cdots & 0 \\ a_3 & a_2 & a_1 & 1 & \cdots & 0 \\ a_5 & a_4 & a_3 & a_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{2n-1} & a_{2n-2} & a_{2n-3} & a_{2n-4} & \cdots & a_n \end{bmatrix} \tag{19}$$

The principal minors H_k, $k = 1, \ldots, n$, are

$$H_1 = a_1, \quad H_2 = \begin{bmatrix} a_1 & 1 \\ a_3 & a_2 \end{bmatrix}, \quad \ldots \tag{20}$$

Equivalently, the real parts of the roots of the characteristic polynomial (i.e., the real parts of each eigenvalue) all have negative real part if and only if the following hold (similar criteria can be derived using the matrices above for $n \geq 6$):

$$\begin{aligned} n = 2: &\quad a_2 > 0, \ \textit{and} \ a_1 > 0. \\ n = 3: &\quad a_1 > 0, \ a_3 > 0 \ \textit{and} \ a_1 a_2 > a_3. \\ n = 4: &\quad a_1 > 0, \ a_3 > 0, \ a_4 > 0, \ \textit{and} \ a_1 a_2 a_3 > a_3^2 + a_1^2 a_4. \\ n = 5: &\quad a_i > 0 \ (i = 1, \ldots, 5), \ a_1 a_5 + a_1 a_2 a_3 > a_3^2 + a_1^2 a_4 \ \textit{and} \\ &\quad (a_1 a_4 - a_5)(a_1 a_5 + a_1 a_2 a_3 - a_3^2 - a_1^2 a_4) > a_5 (a_1 a_2 - a_3)^2. \end{aligned}$$

[7] Here we give the correct $n = 5$ case, which is missing a few terms in [4].

Challenge Problem 2 Look up the analogues of Theorems 2 and 3 for determining the stability of *fixed points* for discrete maps.

Here we have only briefly touched upon equilibrium stability analysis. Interested readers are encouraged to consult standard applied dynamical systems texts, such as [4, 45, 160], for additional details and related topics like the asymptotic stability analysis of equilibria (also called *fixed points*) in discrete-time systems.

The following exercises illustrate the application of Theorems 2 and 3.

Exercise 3 The well-known logistic equation, where $r, K > 0$ and $x \geq 0$, is given by

$$\frac{dx}{dt} = r\,x\left(1 - \frac{x}{K}\right) \tag{21}$$

and has equilibria $x = 0$ and $x = K$. Use Theorem 2 to show that $x = 0$ is unstable and $x = K$ is stable. *Hint: What is the derivative of* $f(x)$ *with respect to* x*?*

Exercise 4 The following extension of the logistic equation includes a strong Allee effect: below a certain threshold ($\alpha > 0$) the population declines towards 0.

$$\frac{dx}{dt} = r\,x\left(1 - \frac{x}{K}\right)\left(\frac{x}{\alpha} - 1\right) \tag{22}$$

Find all three equilibria of this model, and determine their stability.

The next few exercises deal with the Rosenzweig–MacArthur predator–prey model (for background on this and related models, see [118, 125])

$$\frac{dN}{dt} = r\,N\left(1 - \frac{N}{K}\right) - \frac{a\,P}{k+N}N \tag{23a}$$

$$\frac{dP}{dt} = \chi\frac{a\,N}{k+N}P - \mu\,P \tag{23b}$$

where N and P are the prey and predator densities, respectively, $r > 0$ and $K > 0$ are the logistic growth rate and carrying capacity for the prey, $a > 0$ is the maximum per-predator rate of predation, k is the half-saturation value of the prey population in the Holling type-2 predation rate term (when $N = k$ the per-predator predation rate is $a/2$), $\chi > 0$ is a conversion factor between the number of prey consumed and predators born, and $\mu > 0$ is the predator mortality rate corresponding to having assumed exponentially distributed predator lifetimes with a mean lifespan $1/\mu$.

Exercise 5 Find equilibrium points of the Rosenzweig–MacArthur model above. Note that it may be helpful to write these equilibrium values where one state variable value is given as a function of parameter values, and the other is written as a function of parameter values as well as the equilibrium value of the other state variable. *Hint:*

There are three non-negative equilibria: one with zero organisms, one with only prey, and one with both predators and prey coexisting.

Exercise 6 Find the Jacobian for this system (recall Eq. (14)), and evaluate it at each equilibrium point. *Hint: The Jacobian for the case with zero organisms is*

$$\begin{bmatrix} r & 0 \\ 0 & -\mu \end{bmatrix}. \tag{24}$$

Challenge Problem 3 Find the characteristic polynomial of each Jacobian above and apply the Routh–Hurwitz criteria to determine their stability. *Hint: For the case with zero organisms, the characteristic polynomial is*

$$p(\lambda) = \lambda^2 + (\mu - r)\lambda - r\,\mu. \tag{25}$$

Challenge Problem 4 Repeat the above stability analysis for the Rosenzweig–MacArthur model using a *computer algebra system* like the free software wxMaxima [163] (an improved user interface for Maxima [148]), Sage, or commercial software such as Maple or Mathematica.

Challenge Problem 5 What are some other kinds of *attractors* besides equilibrium points (e.g., limit cycles) and how do we determine their stability?

2.3 Bifurcation Analysis

An equilibrium stability analysis often reveals parameter space *thresholds* that mark important qualitative transitions in the long-term behavior of solutions. These typically manifest as inequalities involving model parameters and equilibrium values.

For example, in our simple exponential decay equation, the coefficient $a = 0$ divides *parameter space* into cases of exponential decay ($a < 0$) and those leading to exponential growth ($a > 0$). In infectious disease models, one can often find an expression for the *basic reproduction number* of the pathogen (commonly denoted as $\mathscr{R}_0$ and interpreted as the expected number of new infections per infectious individual over the average duration of infectiousness when the number of susceptible individuals is at the disease-free equilibrium). For example, for the SIR model given by Eqs. (8) the number susceptible at the disease-free steady state is N, hence the rate of new infections per infectious individual, per unit time, is βN and the mean duration of infectiousness is $1/\gamma$. Thus[8] (multiplying rate times time) we have $\mathscr{R}_0 = \beta N/\gamma$. It is commonly the case that no epidemic will occur if $\mathscr{R}_0 < 1$

[8]These basic reproduction numbers are best derived from stability criteria, which often require some rearrangement using a similar interpretation to arrive at the proper form of $\mathscr{R}_0$.

(i.e., the disease-free equilibrium is stable) but an epidemic will occur if $\mathscr{R}_0 > 1$ (i.e., when infectious individuals more than replace themselves, on average). For more on $\mathscr{R}_0$ calculations and its role in epidemic thresholds, see [22, 23, 35, 43].

In addition to changes in the stability of equilibria (or in some cases, their appearance or disappearance) threshold phenomena also arise in non-equilibrium dynamics. For example, the transition between steady-state and oscillatory behavior is quite common in applications, e.g., it is exhibited by the Rosenzweig–MacArthur model mentioned above in which there is a transition from predator and prey coexisting at equilibrium to asymptotically periodic solutions where the predator and prey populations oscillate together (this example will be discussed in more detail below). In some higher dimensional models, these can include much more complex dynamics, including transitions to chaos.

Bifurcation theory deals with the mathematical study of these transitions [73, 84, 104, 160, 165]. Formally, *bifurcations* are the thresholds in parameter space that indicate where there are changes in the topological structure of the model behavior (i.e., in the vector field defined by f). For example, we can consider the set of all parameters for which predators and prey coexist at equilibrium as *topologically equivalent*,[9] and topologically distinct from the case where the predators are unable to live on the prey and go extinct. In essence, bifurcation thresholds partition parameter space into regions that yield the same qualitative behavior, making them an excellent tool for studying model dynamics in an applied setting.

The goals of a *bifurcation analysis* are to characterize where bifurcations occur in parameter space, and to identify the types of bifurcation(s) involved. Importantly, identifying the different types of bifurcations informs us of the long-term behavior of solutions for parameters on either side of these bifurcation thresholds. These boundaries are, fortunately, usually very well defined, e.g., they often involve an equilibrium point where one of the real parts of its eigenvalues passes through zero (recall that our stability criteria above say that an equilibrium point is stable if the eigenvalues of the corresponding Jacobian have negative real part). More complex bifurcations involving other types of attractors (e.g., limit cycles or chaotic attractors) can be similarly characterized by extending the above notions of equilibria and their stability to these other sets[10] of points, e.g., limit cycles, and their stability. Importantly, there are a small number of named bifurcations that we often see arise in applications, e.g., *Hopf* bifurcations (where an isolated equilibrium point changes stability and gives rise to a *limit cycle*—an isolated closed curve in state space that is a solution of the model—i.e., a change from steady-state to oscillatory dynamics), *transcritical* bifurcations (two equilibrium points pass through one another and exchange stability), *saddle-node* bifurcations (where

[9]For readers unfamiliar with topological equivalence, this essentially means that the gross qualitative features of solutions are the same across *equivalent* parameter sets, e.g., the number of equilibrium points and their stability will be the same, even though their exact numerical values may change. See [160, pg. 156], or more advanced treatments of topological equivalence in the texts mentioned above.

[10]See the definition of an *invariant set*, and different types of attractors, in texts like [73, 104].

two equilibria collide and vanish). For a more detailed treatment of the different bifurcations that commonly arise in applications, consult standard texts such as [73, 84, 104, 160, 165].

In practice, the first step of a bifurcation analysis is to conduct an equilibrium stability analysis. From there, stability thresholds can be identified and these can sometimes be useful for identifying specific types of bifurcations. This will often reveal equilibrium bifurcations like saddle-node and transcritical bifurcations, but this can also reveal bifurcations involving non-equilibrium attractors like Hopf bifurcations that involve limit cycles. For example, there is a very useful result [74, 75] that states that, when using the Routh–Hurwitz criteria to determine equilibrium stability, the loss of stability via the failure of the criterion listed last, for each n value considered at the end of Theorem 3, coincides with the loss of equilibrium stability via a Hopf bifurcation.

Computational methods also exist for conducting bifurcation analyses (e.g., see [41, 55, 56, 155]), and can be quite useful for exploring model dynamics for parameter values near a specific parameter set. A less sophisticated approach is to simply use brute force to simulate solutions over a range of parameter values, and analyze those simulation results to infer bifurcations (see Sect. 2.6 below for more on simulating solutions to ODEs on the computer). For example, Fig. 1 shows simulated asymptotic states of the Rosenzweig–MacArthur model over a range of maximum predation rates, a.

The preferred approach, however, is to use computational methods that are specifically designed to find bifurcations in parameter space. The reason? First, the above simulation-based approach will typically miss part of the picture in situations where different initial conditions lead to different attractors (i.e., cases of *bistability* or *multistability*), or in cases where a bifurcation involves the creation or disappearance of either an unstable equilibrium point or unstable limit cycle (which can be practically impossible to find by running simulations unless you know ahead of time where or how to look for them). Simulation based approaches also rely on the investigator to infer what bifurcations might or might not be involved, whereas software designed for bifurcation analysis includes rigorous mathematical criteria that are automatically checked to notify the investigator when certain types of bifurcations are found. Two popular software packages for conducting bifurcation analyses are MatCont [41], which is freely available software that runs in Matlab, and the software AUTO which is available in different forms including through the XPPAUT simulation software [55, 56] which includes an interface for AUTO. An introduction to using such software is beyond the scope of this book; however, various tutorials and examples can easily be found online (see also the list of software available on the SIAM Dynamical Systems activity group website [155]: https://dsweb.siam.org/Software).

Challenge Problem 6 Improve upon the bifurcation diagram in Fig. 1 with an equilibrium stability analysis and by using bifurcation continuation software such as MatCont or XPPAUT to do a computational bifurcation analysis of the model given by Eqs. (23). Use the parameter values given in the figure caption. *Hint: Are there any missing equilibria or other stable or unstable attractors not shown in Fig. 1?*

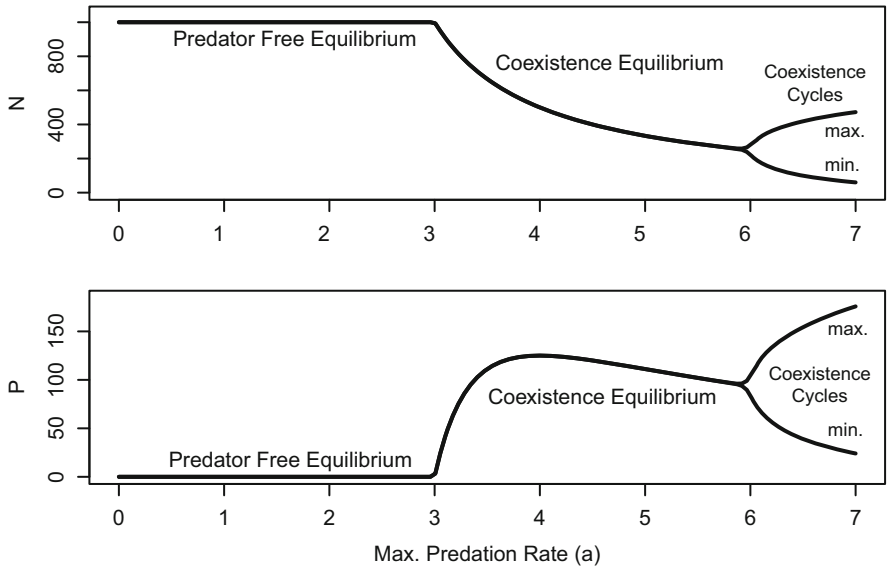

Fig. 1 A "brute force" bifurcation diagram for the Rosenzweig–MacArthur model, generated by numerically simulating solutions over very long time periods then plotting the long-term state variable values as a function of parameter a. For solutions that converge to limit cycle oscillations, the min and max values are plotted. Parameter values used: $r = 1$, $K = 1000$, $a = 5$, $k = 500$, $\chi = 0.5$, $\mu = 1$, and initial condition $(N_0, P_0) = (1000, 10)$. These results suggest a transcritical bifurcation near $a = 3$ and a Hopf bifurcation near $a = 6$

2.4 A Few Comments on Approximation

In applications, some questions cannot be appropriately answered using the asymptotic analyses described above, but might instead be addressed using careful approximations or simulations (numerical solutions to ODEs are discussed in Sect. 2.6).

Approximation methods can be very useful for gaining insight into specific phenomena. For example, in Sect. 2.2 linear approximation via Taylor's Theorem was used to clarify the behavior of trajectories near equilibria. Similar techniques exist that use other expansions to make local approximations, e.g., Fourier expansion can be used to approximate periodic solutions to quantify period–amplitude relationships (for more on investigating oscillations in nonlinear dynamical systems, see [140]). Whatever the technique, a good presentation of approximation-based results should balance intuition and mathematical rigor, and hopefully make clear the conditions for which the approximation is a good one.

For example,[11] consider this heuristic argument: the logistic growth model Eq. (21) should approximately obey an exponential growth model

$$\frac{dx}{dt} = r\,x \tag{26}$$

when x values are near 0, since the term $1 - x/K \approx 1$ when $x \ll K$. Alternatively, one could arrive at the same conclusion using a Taylor expansion of the logistic equation about $x = 0$, which would yield the same linear approximation, but would also yield a remainder that quantifies the approximation error.

Approximation methods can also be used to study *transient dynamics*, although the approaches used vary by both the specifics of the model, and the question(s) being asked. For example, one might ask when an epidemic might peak in the SEIR model, and answering this question might require a different approach than is used to determine the conditions under which the Rosenzweig–MacArthur model exhibits damped oscillations that converge to a steady-state coexistence between predator and prey. In addition to computational investigations (see Sect. 2.6), the approximation approach described next in Sect. 2.5 is often used to disentangle some transient dynamics and long-term asymptotic dynamics in systems with multiple time scales, which commonly arise in biological applications.

2.5 Fast–Slow Analysis of Systems with Multiple Time Scales

In some systems of ODEs, sometimes called *stiff* systems, dynamics occur very rapidly in certain directions in state space, while other parts of the system change more slowly. These types of systems are said to have *multiple time scales* and these can often be analyzed by separately considering the dynamics on the different time scales using appropriate approximating models. In short, the slow time scale state variables can often be treated as fixed constants, yielding an approximate model that includes only the fast time scale variables. Once the dynamics of that fast subsystem are understood, that information can be used to create a complementary set of equations for the slow subsystem that approximates the long-term behavior of the model. Often, the fast variables converge quickly to a quasi-steady-state (i.e., an equilibrium point of the fast subsystem) it can typically be assumed that, over the slow time scale, the fast time scale variables closely track these quasi-equilibrium values. Thus, the ODEs governing the slow time scale variables can be approximated by replacing all occurrences of the fast time scale variables with these quasi-equilibrium expressions, thus reducing the system to a smaller number of ODEs.

[11] Here, integration by parts will yield an equation for the solution curves $x(t)$, but we proceed assuming no such curves are available as this is typically the case in practice.

To clarify, suppose an ODE model with n state variables can be separated into slow ($\mathbf{x}$) and fast ($\mathbf{y}$) time scale variables and written as

$$\frac{d\mathbf{x}}{dt} = \epsilon\, f_{\text{slow}}(\mathbf{x}, \mathbf{y}, \theta) \tag{27a}$$

$$\frac{d\mathbf{y}}{dt} = f_{\text{fast}}(\mathbf{x}, \mathbf{y}, \theta). \tag{27b}$$

Here $0 < \epsilon \ll 1$ and f_{slow} and f_{fast} are roughly the same order of magnitude, hence $d\mathbf{x}/dt$ will be very small and $\mathbf{x}$ will change much more slowly than $\mathbf{y}$. If the $\mathbf{x}$ values are treated as constants in Eq. (27b) (i.e., if we let $\epsilon \to 0$) and there are quasi-equilibrium expressions $\mathbf{y}_*(\mathbf{x}, \theta)$ that satisfy $f_{\text{fast}}(\mathbf{x}, \mathbf{y}_*(\mathbf{x}, \theta), \theta) = 0$ so that the slow time scale dynamics can be approximated by

$$\frac{d\mathbf{x}}{dt} = \epsilon\, f_{\text{slow}}(\mathbf{x},\ \mathbf{y}_*(\mathbf{x},\ \theta), \theta) \tag{28}$$

where we see that the right hand side is purely a function of $\mathbf{x}$ and θ. For examples of such fast–slow system analyses, see [17, 36, 37, 90, 91, 134, 142, 153] or see the more thorough treatment of the topic in [103].

To illustrate how a fast–slow analysis can yield a simplified model, consider the well-known Michaelis–Menten equation, which was originally used to model a biochemical reaction in which an enzyme catalyzes the hydrolysis of sucrose into simpler sugars [95]. In more general terms, the model is of a substrate S that binds to enzyme E to form the complex ES which either dissociates back into free enzyme and substrate, or undergoes a reaction that releases the enzyme and a reaction product P. This can be written in chemical reaction equation notation as

$$E + S \underset{k_u}{\overset{k_b}{\rightleftharpoons}} ES \overset{k_{cat}}{\to} E + P$$

where k_b is the binding rate, k_u is the unbinding rate, and k_{cat} is the catalytic rate. Let the concentrations of enzyme E, substrate S, complex ES, and product P at time t be denoted $z(t)$, $s(t)$, $x(t)$, and $p(t)$, respectively. A corresponding (mean field) ODE model of this reaction is

$$\frac{dz}{dt} = -k_b\, z\, s + k_u\, x + k_{cat}\, x \tag{29a}$$

$$\frac{ds}{dt} = -k_b\, z\, s + k_u\, x \tag{29b}$$

$$\frac{dx}{dt} = k_b\, z\, s - k_u\, x - k_{cat}\, x \tag{29c}$$

$$\frac{dp}{dt} = k_{cat}\, x \tag{29d}$$

Note that there are two quantities in this model that remain constant over time: If z_0 is the initial amount of enzyme, and s_0 is the initial amount of substrate, then $z + x = z_0$ and $s + x + p = s_0$. Thus, we can omit the first equation (29a) from the model since we can always infer z from x since $z = z_0 - x$. Likewise, we can omit Eq. (29c) since $x = s_0 - s - p$. This yields the equivalent, but simpler, model

$$\frac{ds}{dt} = -k_b\,(z_0 - (s_0 - s - p))\,s + k_u\,(s_0 - s - p) \tag{30a}$$

$$\frac{dp}{dt} = k_{cat}\,(s_0 - s - p). \tag{30b}$$

Next, assume that the catalytic reaction is much faster than the other binding and unbinding reactions, so that Eqs. (30) become a slow-fast system where s very quickly reaches a quasi-equilibrium state that then slowly tracks the slower timescale changes in p. In short,[12] setting $ds/dt = 0$ and solving for that quasi-equilibrium relationship between s and p yields

$$p(s) = \frac{-k_b\,z_0\,s}{k_u + k_b\,s} + s_0 - s \tag{31}$$

which, when substituted into Eq. (30b), yields

$$\frac{dp}{dt} = \frac{V_{max}\,s}{k_d + s} \tag{32}$$

where the maximum velocity (i.e., maximum rate of change of p) $V_{max} = k_{cat}\,z_0$ and the dissociation constant $k_d = k_u/k_b$. Other assumptions can lead to similar approximations for this same model [24, 95], and similar analyses for mathematically related interaction processes have been used to derive similar nonlinear response curves, e.g., compare the right hand side of Eq. (32) with the predation rate in Eqs. (23) (for more details, see [38, 125] and references therein).

2.6 Computing Numerical Solutions to ODEs

It can often be useful to supplement analytical approaches with numerical simulation studies, e.g., to assess how well an approximate model compares to the original model, or to investigate transient dynamics when analytical approaches are not practical. In typical introductory ODE class, students are taught methods for finding (or sometimes approximating) analytical solutions to a given set of ODEs. Unfortunately, analytical solutions rarely exist for the kinds of nonlinear

[12] Here we are glossing over the formal details from singular perturbation theory [103] for deriving equations that approximate the fast- and slow-timescale dynamics of this model.

ODE models encountered in applications. This section introduces how to compute numerical solutions to ODEs.

It is relatively straightforward to compute approximate numerical solutions to ODEs using standard computational methods available in nearly all mathematical software [138], or by coding up your own solver if the more advanced method that you need is otherwise unavailable. In R, the `ode` function in the `deSolve` package [156] implements various numerical methods that are each tailored to provide reliable numerical solutions by addressing various numerical challenges posed by different categories of ODEs. Similar tools exist in Matlab [115] and additional solvers can sometimes be found in other software, e.g., in [78]. Here we provide a simple introduction to computing numerical solutions in R, but leave it to the reader to further investigate the conditions under which different methods should, and should not, be applied. See the Appendices for resources to get started using R, and for more on numerical methods see [115, 138, 155, 160] and references therein.

2.6.1 Euler's Method

To understand how various methods for computing numerical solutions are implemented on a computer, it is instructive to see the very simple algorithm known as Euler's method. Like other methods, it approximates a continuous curve using a discrete-time approximation where information about the derivative of the curve (i.e., the right hand side of the ODE) is used to determine where the state variables move to on each time step. Consider the generic system of ODEs

$$\frac{d\mathbf{x}}{dt} = f(t, \mathbf{x}, \theta), \quad \mathbf{x}(0) = \mathbf{x_0} \tag{33}$$

and note that, if we back off of the limit implicit in the derivative on the left, we get that for small Δt,

$$\frac{\mathbf{x}(t + \Delta t) - \mathbf{x}(t)}{\Delta t} \approx f(t, \mathbf{x}(t), \theta) \tag{34}$$

which can be rearranged to yield

$$\mathbf{x}(t + \Delta t) \approx \mathbf{x}(t) + f(t, \mathbf{x}(t), \theta)\, \Delta t. \tag{35}$$

Euler's method approximates trajectories starting at $\mathbf{x}(0) = \mathbf{x_0}$ by iterating the above discrete-time map Eq. (34). This method has been improved upon in various ways, which range from adaptively picking smaller or larger time steps when appropriate, to incorporating information about the curvature of the trajectory to obtain a better increment than the linear approximation $f(t, \mathbf{x}(t), \theta)\Delta t$.

Consider Eq. (34) starting at time $t = 0$. The following information is needed to implement Euler's method and similar algorithms for computing numerical solutions to ODEs: a starting vector of *initial condition* values $\mathbf{x}(0) = \mathbf{x_0}$, parameter values θ, and a way of computing the derivative $f(\mathbf{t}, \mathbf{x}, \theta)$.

The R code below illustrates how this is implemented using the `ode` function in the `deSolve` package is illustrated below.

2.6.2 Numerical Solutions in R

The first step is to create a function that computes $f(t, \mathbf{x}, \theta)$ given a specific value of time t, the state of the system $\mathbf{x}$, and a set of parameter values θ. Consider the SIR model given by Eqs. (8). The right hand side of the model (the *derivative function* f) is a function of the state variables S, I, and R and parameter values β and γ. According to the R documentation[13] for the `ode` function we must create an R function that computes these derivatives based on three inputs—time, a vector of state variable values, and a parameter vector, in that order, i.e., $f(t, \mathbf{x}(t), \theta)$—and returns the vector of derivative values in an R object known as a `list`. This is implemented in the following R code.

```
# We need to load the ode() function in package deSolve
# install.packages("deSolve") Install once, if needed
library(deSolve); # load  deSolve  into the workspace
# Define an SIR function to use with ode()
SIR = function(tval, X, params) {
        S = X[["S"]] # one could also have used X[1]
        I = X[["I"]] # ... X[2]
        R = X[["R"]] # ... X[3]
        B = params[["beta"]]  # or  params[1]
        g = params[["gamma"]] # ... params[2]
        # Now compute the SIR model derivatives
        dS = -B*I*S
        dI = B*I*S - g*I
        dR = g*I
        # Return the derivatives in a list to use with ode()
        return(list(c(dS,dI,dR)))
}
```

Next, the `ode` function requires that we give it a set of initial conditions, a vector of (user specified) time values for which to return state variable values, and a set of parameter values. Thus, we must specify $S(0)$, $I(0)$, $R(0)$ and parameters β and γ. The `ode` function returns a matrix whose first column is the vector of time values, and the subsequent columns are the corresponding state vector values (in this case,

[13] To view the documentation, load the package `deSolve` with the command `library(deSolve)` then type `?ode` into the R console. See the Appendix for additional resources to get started using R.

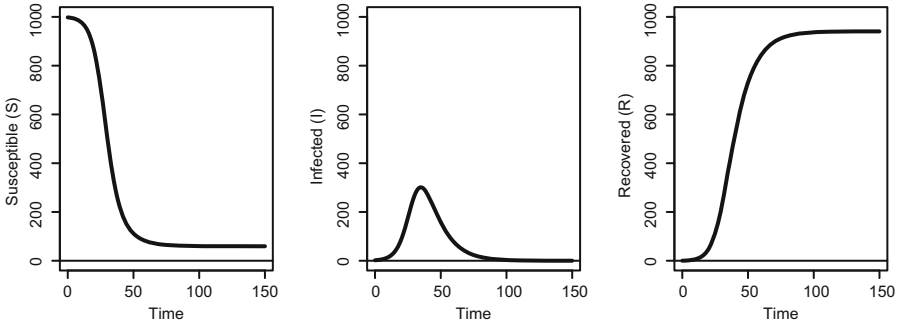

Fig. 2 Numerical solution to the SIR model, Eqs. (8). See the code in the main text for details

the values of S, I, and R). This is implemented in the following code, which also plots the three numerical solution curves as functions of time (Fig. 2).

```
X0 = c(S=998, I=2, R=0) # Initial state values
Pars = c(beta=0.0003, gamma=0.1)
Time = seq(0, 150, length=200) # time values for ode()
# Now we can compute a numerical solution
Xout = ode(X0, Time, SIR, Pars) # Default: method="lsoda"
head(Xout, 3) # Show the first three rows of output
```

```
          time        S        I         R
[1,] 0.0000000 998.0000 2.000000 0.0000000
[2,] 0.7537688 997.5131 2.324244 0.1626696
[3,] 1.5075377 996.9476 2.700737 0.3517011
```

```
# Plot the numerical solution curves ...
par(mfrow=c(1,3)) # ... in a figure with 1 row, 3 columns.
plot(Time, Xout[,2], ylab="Susceptible (S)", ylim=c(0,sum(X0)),
     type="l", lwd=2);  abline(h=0) # abline() draws the x-axis
plot(Time, Xout[,3], ylab="Infected (I)", ylim=c(0,sum(X0)),
     type="l", lwd=2);  abline(h=0)
plot(Time, Xout[,4], ylab="Recovered (R)", ylim=c(0,sum(X0)),
     type="l", lwd=2);  abline(h=0)
```

Exercise 7 Modify the example above so that the ODE solver only uses the equations for S and I, but not R. At the end of the code, after the output from `ode` has been obtained, add the R column to `Xout` using the relationship $R(t) = N(0) - S(t) - I(t)$ (to convince yourself $N = S + I + R$ is constant, note its derivative $dN/dt = 0$).

Challenge Problem 7 Modify the code above by adding an exposed class so that the code generates solutions of the SEIR model mentioned previously. Explore how solution change for different values of ν.

Challenge Problem 8 Modify the code above to correspond to changing the quantity β to a positive-valued function of time, e.g., using a periodic function such

as $\beta(t) = \beta_0 + A\ \sin(\omega\, t)$. This makes the previously autonomous system of ODEs into a non-autonomous system. To do this, modify the equations in `SIR` as well as the parameter vector (to assign values to the new parameters in the model). How do the solution curves change relative to the case where β is a constant?

Challenge Problem 9 Implement the Euler algorithm on your own, without using `ode` and the 4th order Runge–Kutta (RK4) algorithm (e.g., see page 33 in [160]).

Challenge Problem 10 Use integration by parts to show the θ-Logistic model

$$\frac{dx}{dt} = r\, x \left(1 - \left(\frac{x}{K}\right)^{\theta}\right) \tag{36}$$

has an analytical solution, then modify the code above to find a numerical solution using `ode` in R. Use initial conditions $x(0) = 10$, $r = 1$, and $K = 1000$, and explore θ values above, below, and at $\theta = 1$. Plot both curves together on the same plot to compare the exact and approximate solution curves.

2.6.3 Keeping Numerical Solutions Positive: The Log-Transform Trick

The above approach to computing numerical solutions to ODEs can sometimes be problematic for ODE models of biological systems. Often, in biological models (e.g., like the SIR model mentioned above) one or more equations are of the form

$$\frac{dx}{dt} = g(x)\, x. \tag{37}$$

Importantly, this implies that trajectories $x(t)$ slow as they asymptotically approach $x = 0$, and therefore can never pass through $x = 0$. However, numerical methods may take an approximated, discrete-time step that results in the simulated x trajectory erroneously crossing zero (which should be impossible!). This can lead to state variables running off to $\pm\infty$, and other undesirable behaviors. Fortunately, there is a clever trick that allows us to avoid these numerical errors: if we assume that $x(t) > 0$ over the time period for which we seek a numerical solution, then we can let $X = \log(x)$ and by the properties of the natural log function[14] it follows that

$$\begin{aligned} \frac{dX}{dt} = \frac{d}{dt}\log(x) &= \frac{1}{x}\frac{dx}{dt} \\ &= \frac{1}{x}\, g(x)x = g(\exp(X)) \end{aligned} \tag{38}$$

[14]Here the notation `log(x)` is used for the natural log function, instead of `ln(x)`, following the convention used in most modern scientific programming languages. Likewise, `exp(x)` $= e^x$.

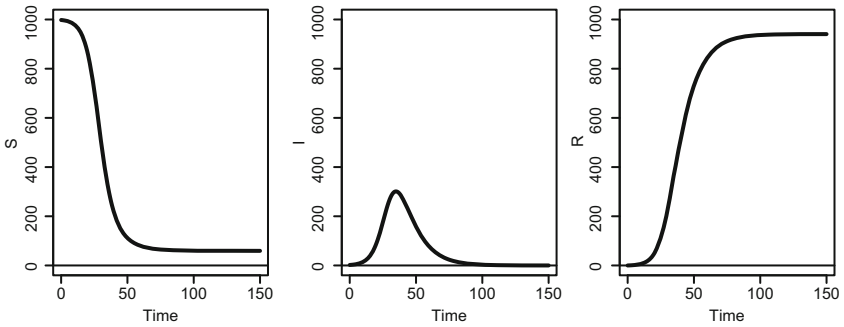

Fig. 3 A numerical solution to the log-transformed SIR model using the same parameters and initial conditions as in Fig. 2. See the code in the main text for details

That is, if we transform a strictly positive state variables x to a real-valued variable X using the natural log function, then numerical solutions of X can take on both positive and negative values that can later be transformed back to strictly positive x values, since $x = \exp(X)$. This transformation of variables is the default approach some modelers take to computing numerical solutions of models where trajectories should remain strictly positive over a finite time interval.

One caveat to this approach is that, since $X(t) \to -\infty$ as $x(t) \to 0$, it can lead to numerical errors due to finite limits on the size of floating point values on a computer. While it may seem contradictory to trade one numerical problem for another, it is usually far less common (and much more manageable) in practice to have a numerical solution diverging to $-\infty$ and hitting the computer's floating point limit. Thus, this approach should be reserved for situations where $x(t)$ remains bounded away from 0 or converges slowly to 0. A second caveat is that often times solutions that erroneously cross through 0 do so because of typos in the code that defines the derivative function. Thus, carefully checking that the equations, as implemented in the code, are free of errors is strongly advised before assuming that numerical methods are to blame for seemingly erroneous numerical solutions.

To illustrate this approach, here is the above SIR model code rewritten in terms of the log-transformed values $X = \log(S)$ and $Y = \log(I)$ where we have also omitted the R equation since the constant population size allows us to calculate $R(t)$ once the $S(t)$ and $I(t)$ curves are obtained using $R(t) = N(0) - S(t) - I(t)$ (Fig. 3).

```
# A function for ode() to compute numerical solutions
# of the log-transformed SIR model.
logSI = function(tval, Xs, params) {
   X = Xs[["X"]]        # X = log(S)
   Y = Xs[["Y"]]        # Y = log(I)
   B=params[["beta"]]
   g=params[["gamma"]]
   # Now compute the derivatives
   dX = -B*exp(Y)       # I = exp(Y)
   dY = B*exp(X) - g    # S = exp(X)
```

```
   # Return the derivatives in a list to use with ode()
   return(list(c(dX,dY)))
}

# Initial conditions, log-transformed
logX0 = c(X=log(X0[["S"]]), Y=log(X0[["I"]]))
logXout = ode(logX0, Time, logSI, Pars) # requires deSolve

# Convert back to S and I, and calculate R
Xout = cbind(Time,S=exp(logXout[,2]), I=exp(logXout[,3]),
             R=sum(X0) - exp(logXout[,2]) - exp(logXout[,3]))
head(Xout,3)
```

```
          Time        S        I             R
[1,] 0.0000000 998.0000 2.000000 -2.273737e-13
[2,] 0.7537688 997.5126 2.324242  1.631915e-01
[3,] 1.5075377 996.9464 2.700730  3.528366e-01
```

```
# Plot the numerical solution curves as before
par(mfrow=c(1,3)) # 3 panels in 1 row
plot(Time, Xout[,2], ylab="S", ylim=c(0,sum(X0)),
     type="l", lwd=2);   abline(h=0)
plot(Time, Xout[,3], ylab="I", ylim=c(0,sum(X0)),
     type="l", lwd=2);   abline(h=0)
plot(Time, Xout[,4], ylab="R", ylim=c(0,sum(X0)),
     type="l", lwd=2);   abline(h=0)
```

Exercise 8 Modify the code above to simulate the Rosenzweig–MacArthur model, Eqs. (23), using the parameter values from Fig. 1 and using both direct implementation and the log-transformation technique. Plot both results to compare approaches.

Challenge Problem 11 Look at the different methods available through the `ode` function in the `deSolve` package ([156]; type `?ode` into R to view the documentation). When should you use one method over the other? Compare these to the methods available in Matlab, Python, or other computing platforms (see also [115, 138]).

Challenge Problem 12 It can be challenging to obtain numerical solutions to stiff systems. Find resources like [116, 138] to help you better understand which numerical methods work well for stiff (multiple time scale) systems in the R package `deSolve` as well as in Matlab or similar software, and look up some stiff models to use as examples. Use the different methods to find numerical solutions for the same parameter values and time values, and compare these to methods that should perform poorly (e.g., Euler). How different are the results from different methods? How do R's stiff solvers compare to those in Matlab or other software?

2.7 ODEs as Statistical Models

This final section in Sect. 2 introduces some useful statistical concepts and basic approaches to using an ODE model as the basis for a statistical model.[15] Increasingly, ODE models are being used as components of statistical models, e.g., for making inferences about underlying mechanisms or for forecasting. Applications include making inferences from model parameters estimated from time series data (i.e., a sequence of state variable measurements taken at multiple time points, that in some way corresponds to a sequence of state variable values), quantifying uncertainty in those parameter estimates, forecasting, and conducting statistical tests to determine whether a certain parameter is significantly different from zero or whether one model fits the data better than another model (i.e., *model selection*).

2.7.1 A Brief Overview of Key Statistical Concepts

Parameter estimation can be approached in various ways, but to start off it is helpful to first introduce some concepts from the statistical theory of *estimators* (e.g., see [85, 106]). You may recall that certain formulas exist for estimating parameters in specific contexts, e.g., for a normal (Gaussian) distribution with mean μ and variance σ^2, an estimate of μ can be obtained by calculating the *sample mean* $\bar{y} = \sum_{i=1}^{n} y_i/n$ of a *random sample*, i.e., n data points $y_1, \ldots, y_n$ where each y_i are independent draws from the given normal distribution. Formally, we refer to $\widehat{\mu} \equiv \bar{y}$ as an *estimator* because it defines a function of our data that yields a parameter estimate. The "hat" notation indicates that $\widehat{\mu}$ is an estimator of parameter μ. Note that we could have chosen other estimators to compute an estimate of parameter μ, e.g., (a) the median of the sample, (b) the geometric mean instead of the arithmetic mean, or (c) the mean of the extreme values of the sample (i.e., the mean of $y_{\min}$ and $y_{\max}$).

Estimator theory in statistics deals in part with how to select an estimator with certain desired properties. For example, some estimators are more robust to outliers than others, and some may be more or less biased. A good place to begin discussing properties of estimators is to recognize that estimators are functions of data, and hence, we can think of them as *functions of random variables*. As such, estimators are themselves random variables and therefore follow some distribution, and it is that distribution that determines the properties of the estimator. For example, the sample mean $\bar{y}$ above is proportional to the sum of normal random variables, and is therefore itself normally distributed with a mean and variance that can be computed from the definitions of the y_i distribution [85, 106]. In a statistical context, a desirable property of an estimator is that it be *unbiased*, i.e., its expected value equals the true parameter value of interest (if not, we say the estimator is *biased*).

[15] For related resources in R, see the packages such as `CollocInfer` [87], `deBInfer` [19], or browse the relevant CRAN Task Views (https://cran.r-project.org/web/views/).

Estimators should also have as *small a variance as possible* so that estimates fall as close as possible to the mean (i.e., the true parameter value if the estimator is unbiased). Hence, we typically strive for *minimum variance*, *unbiased* estimators; however, in some circumstances we select estimators that may deviate from this ideal (e.g., because they may be more robust to outliers and thus in practice may perform better with "messy" data). In our simple normal distribution example, note that

$$E(\widehat{\mu}) = E\left(\sum y_i/n\right) = \sum E(y_i)/n = \sum \mu/n = \mu \tag{39}$$

so this estimator is *unbiased*. But is it a *minimum variance estimator*? To answer that question requires taking a deeper look into the theory of estimators that is beyond the scope of this chapter. For a more in depth treatment of estimator theory, minimum variance estimators, and a very nice result known as the Cramér–Rao lower bound, see texts such as [28, 85, 106].

Alternatively, on a case-by-case basis, when trying to decide among different estimation procedures or assessing properties of a given estimator, one could simulate data repeatedly effectively sample from the estimator distribution(s), and thus make these assessments computationally. By repeatedly estimating parameters from simulated data, where the "true" parameter values are known, we can sample from an estimator's distribution a large number of times. That large sample of estimates can then be used to assess properties like the mean and variance of the estimator distribution. Comparing multiple estimators in this way can reveal differences in their variance, and any bias can be quantified by calculating how well the mean of the estimates compares to the parameter value used to generate samples from the estimator distribution. For more on computational approaches in statistics, see [70].

To illustrate, the following R code compares the three candidate estimators of μ in the above example by reconstructing each estimator distribution by random sampling: the standard arithmetic mean ($\bar{y}$), the geometric mean $(y_1 \cdot y_2 \cdots y_n)^{1/n}$, and the average of the two extreme values $y_{\min}$ and $y_{\max}$ (which we would expect to perform poorly given how much data it omits from the calculation). This is accomplished by repeatedly sampling from each estimator distribution by iteratively drawing a random sample from the given normal distribution and then calculating parameter estimates from that random sample.

```
# Begin by setting parameters for our simulations
mu = 25  # define our mean parameter for our normal distribution
sd =  2  # define our variance (variance = sd^2)
N  = 10  # to simulate data, use a sample size of N
mu.est1 = c() # empty list for our arithmetic mean estimates
mu.est2 = c() # empty list for our geometric mean estimates
mu.est3 = c() # empty list for our mean(min,max) estimates
# Sample our estimators 100000 times each...
set.seed(6.28) # set seed to get reproducible random numbers
for(i in 1:100000) { # simulate and estimate 100000 times
```

```
  y = rnorm(N,mu,sd) # Simulate data. Read ?rnorm for details
  mu.est1[i] = mean(y) # calculate the standard estimator
  mu.est2[i] = prod(y)^(1/N)      # the geometric mean
  mu.est3[i] = (min(y)+max(y))/2 # the mean of ymin, ymax
}
# Plot results
par(mfrow=c(1,3)) # plots in 1 row, 3 columns
# First panel: standard estimator (population mean)
plot(density(mu.est1), # empirical density function
    main=expression(hat(mu)==sum(y[i])/N),
    xlab=expression(hat(mu)), xlim=c(21.5,29), ylim=c(0,0.65))
hist(mu.est1, 30, freq=FALSE, add=TRUE,
    col=rgb(.6,.6,.6,.5), border=rgb(0,0,0,0.5))
text(27.5,0.6,paste("Mean =",signif(mean(mu.est1),4),  ))
text(27.5,0.57,paste("SE =",signif(sd(mu.est1),3),  ))
abline(h=0) # draw the horizontal axis
# Second panel: geometric mean
plot(density(mu.est2), # empirical density function
   main=expression(hat(mu)==(prod(y[i]))^{1/N}),
   xlab=expression(hat(mu)), xlim=c(21.5,29), ylim=c(0,0.65))
hist(mu.est2, 30, freq=FALSE, add=TRUE,
   col=rgb(.6,.6,.6,.5), border=rgb(0,0,0,0.5))
text(27.5,0.6,paste("Mean =",signif(mean(mu.est2),4),  ))
text(27.5,0.57,paste("SE =",signif(sd(mu.est2),3),  ))
abline(h=0) # draw the horizontal axis
# Third panel: mean(ymin,ymax)
plot(density(mu.est3), # empirical density function
   main=expression(hat(mu)==(y[min]+y[max])/2),
   xlab=expression(hat(mu)), xlim=c(21.5,29), ylim=c(0,0.65))
hist(mu.est3, 40, freq=FALSE, add=TRUE,
   col=rgb(.6,.6,.6,.5), border=rgb(0,0,0,0.5))
text(27.5,0.6,paste("Mean =",signif(mean(mu.est3),4),  ))
text(27.5,0.57,paste("SE =",signif(sd(mu.est3),3),  ))
abline(h=0) # draw the horizontal axis
```

The simulation output in Fig. 4 illustrates how estimators can differ in terms of bias and their variance. In practice, it is not always possible to obtain unbiased estimators that are also minimum variance estimators. However, choosing likelihood based estimation procedures is usually a good starting point because they often yield minimum variance estimates that are asymptotically unbiased (i.e., the bias vanishes as the sample size grows).

2.7.2 Likelihood Based Parameter Estimation

Maximum likelihood based parameter estimation is a widely used approach in statistical applications. Since the approach is not restricted to parameter estimation for ODEs, and can also be used for the other kinds of models mentioned in this chapter, it is worth reviewing some maximum likelihood basics before applying this approach using ODE models. For further details, see [21, 28, 85, 106].

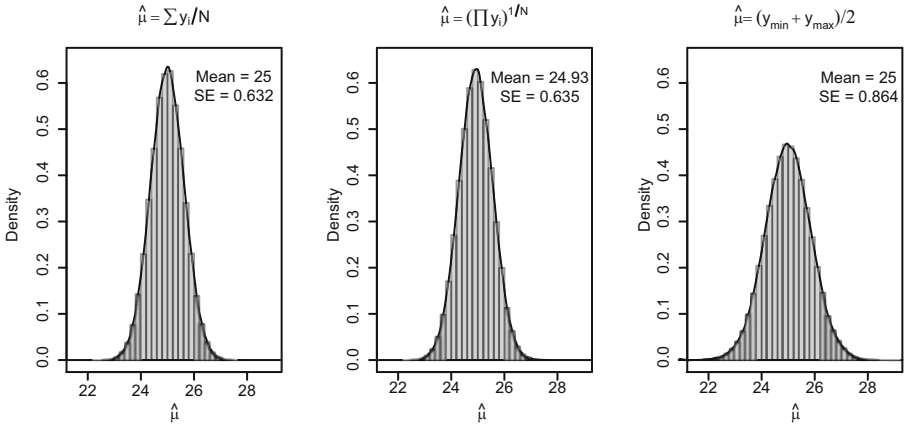

Fig. 4 A comparison to assess the bias and relative variability of three candidate estimators for the mean of a normal distribution: From left to right, the three panels show the result of simulating 100,000 samples (sample size of $n = 10$) and computing an estimate for the mean μ using either the standard arithmetic mean (left), the geometric mean (center), or the mean of the extreme values (right). The plots show a histogram with an empirical density function overlay. The empirical mean and standard deviation (standard error) of the estimator distribution are shown in the top right of each panel. The true value used to simulate the data was $\mu = 25$. These results suggest that the geometric mean may be slightly biased and have a slightly larger variance than the standard estimator, while the third option appears to be unbiased but has substantially higher variance. For more details, see the code in the main text

As mentioned above, statistical theory tells us that estimators derived by finding the parameter set that maximizes the *likelihood function* (defined next) are usually (but not always!) minimum variance estimators and often have little to no bias (again, we would like to check these properties hold to the extent it is possible to do so). We call these estimators *maximum likelihood estimators* (MLEs), and their desirable statistical properties, combined with the relative ease of computing MLEs, have made this a very popular approach to parameter estimation.

The likelihood function should be a familiar function to those who have taken a first course in probability: In short, the likelihood function is just the *joint probability density function* (for continuous data; or the *joint mass function* for discrete data), but where we have fixed the values of our random variables (i.e., our data) and instead treat the distribution parameters as the unknown independent variables. More specifically, for a set of independent observations $y = (y_1, \ldots, y_n)$ (i.e., our data) where the distribution of y_i is described by density function $f_i(y_i, \theta)$ with parameter vector θ, the likelihood of θ, given the data y, is given by

$$L(\theta|y) = \prod_{i=1}^{n} f_i(y_i, \theta). \tag{40}$$

That is, *both the likelihood and joint density functions have the same mathematical formulation*, but we think of them as distinct functions over two different domains: the familiar joint density function domain is the space of possible random variable values (e.g., $\mathbb{R}^n$ for a sample of size n from a normal distribution), while the domain of the likelihood function is the space of all possible parameter values θ (e.g., $\mu \in \mathbb{R}$ and $\sigma > 0$ in our normal distribution example).

In general, once we can write down a joint density function for our data, and thus can specify a likelihood function $L(\theta|y)$ for parameter vector θ given the data y, the MLE $\widehat{\theta}$ is given by the parameter set that maximizes the likelihood function:

$$\widehat{\theta} = \underset{\theta}{\text{argmax}} \; L(\theta|y). \tag{41}$$

In practice, it is often both analytically and computationally easier to minimize the *negative log likelihood* function $nLL(\theta|y) \equiv -\log(L(\theta|y))$ rather than maximize the likelihood function (the two yield equivalent estimators, due to the monotonicity of the log function). Thus, we also have the equivalent definition of the MLE

$$\widehat{\theta} = \underset{\theta}{\text{argmin}} \; -\log(L(\theta|y)). \tag{42}$$

In some cases (like the example below), formulas for the MLE can be found using the standard approach taught in calculus: take the partial derivatives of the negative log likelihood function nLL with respect to each unknown parameter, then set each partial derivative to zero and solve to find the parameter values (which will be functions of the data) that minimize the negative log likelihood (and thus maximize the likelihood). If finding the MLE analytically is not a viable option (e.g., see [101]), computational optimization approaches can be used to find parameter values that minimize nLL for a specific data set.

Continuing with our simple example from the previous section, suppose we would like to use MLEs to estimate the unknown mean μ_0 and standard deviation σ_0 for a normal distribution from a random sample of size n.[16] Let y_i denote the ith observation in our sample ($i = 1, \ldots, n$). The *likelihood function* in this case is given by the same formulation as the joint density function

$$L(\mu, \sigma|y) = \prod_{i=1}^{n} f(y_i, \mu, \sigma) \tag{43}$$

where function f is the normal density function evaluated at y_i using the given mean μ and standard deviation σ. Recall that our y_i values (our data) are now fixed constants, and the domain of this likelihood function is the upper half of $\mathbb{R}^2$ since our two independent variables (inputs) are $\mu \in \mathbb{R}$ and $\sigma > 0$. The maximum likelihood

[16] Here the subscript 0 is used to distinguish the true parameters values μ_0 and σ_0 from the candidate values μ and σ that one might plug in to the likelihood function.

principle states that the parameter values $\mu = \widehat{\mu}_{MLE}$ and $\sigma = \widehat{\sigma}_{MLE}$ that together maximize $L(\mu, \sigma | y)$ are the MLEs for unknown parameters μ_0 and σ_0^2. In this case, these can be found analytically, and are

$$\widehat{\mu}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} y_i, \qquad \text{and} \qquad \widehat{\sigma^2}_{MLE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{\mu}_{MLE})^2. \tag{44}$$

That is, the MLE for the mean is the familiar *sample mean* (recall Fig. 4), while the MLE for the variance is the *unadjusted sample variance*. Importantly, while the MLE for the mean μ_0 is *unbiased*, since $E(\widehat{\mu}_{MLE}) = \mu_0$, the MLE for the variance σ_0^2 is biased, since $E(\widehat{\sigma^2}_{MLE}) = \frac{n}{n-1}\sigma_0^2$, but asymptotically unbiased since the bias vanishes as sample size $n \to \infty$. This example nicely illustrates how, in practice, minimum variance unbiased estimators do not always exist, and we often prefer to use unbiased estimators, even if they require a small increase in the variance of our estimates.

In general, it is worth noting that there are no guarantees that an MLE exists, or that when an MLE exists that it is unique.[17] When no MLE exists, other approaches to parameter estimation must be used or the model must be reformulated. In cases where multiple MLEs exist, special considerations must be made to identify the appropriate course of action (this is discussed below in the context of parameter identifiability).

Exercise 9 Consider the normal distribution example above. Suppose we would like to analyze a long-term series of water depth data (each depth measurement x_i has an associated time value t_i) in a coastal bay to quantify a rise in sea level (which we assume might be increasing linearly over the time period for which we have data). Assume that the data collection site in the bay experiences tidal fluctuations and that the expected value for a depth measurement at time t is assumed to follow the curve $d(t)$ given by

$$d(t) = d_0 + m\,t + A\sin(2\pi\,t/T) \tag{45}$$

where d_0 is a baseline depth, m is the slope of the long-term increase (or decrease) in sea level, and the last term describes the smaller timescale rise and fall of the tide (which here has been simplified to have a fixed period T and fixed amplitude A). Assume that wave activity and other factors introduce noise into these measurements that roughly follow a normal distribution so that a measurement x_i at time t_i is normally distributed with mean $d(t_i)$ and standard deviation σ. Write down the likelihood function for this data set.

[17] Sufficient criteria for the existence of one or more MLEs are that the parameter space is compact and the log likelihood surface is continuous. In practice, non-unique MLEs are more commonly the problem, especially when working with nonlinear ODE models.

Next, we look at an approach to using likelihood based parameter estimation for ODE models where, in contrast to the exercise above, the mean value for our likelihood calculation depends on a solution to an ODE. Since many ODEs do not have a closed form solution, the mean therefore must be calculated using a numerical solution to the ODE.

2.7.3 Likelihood Framework for ODEs

A popular framework for parameter estimation using ODE models is the following (e.g., see [32, 33, 46, 47, 123, 141] and references therein). First, in order to distinguish between the modeled "true" state of the system $\mathbf{x}$ (e.g., the number of rabbits in a forest) versus the data $\mathbf{y}$ (e.g., the count of rabbits along multiple transects), we define the *process model* Eq. (46a) (i.e., what we typically think of as our mechanistic mathematical model of our real-world study system) and a set of one (or more) variables that we refer to as the *observation model* Eq. (46b).

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \theta), \quad \mathbf{x}(0) = \mathbf{x_0} \tag{46a}$$

$$\mathbf{y} = g(\mathbf{x}, \theta) \tag{46b}$$

Note that unknown parameters that one hopes to estimate may appear in both the process and observation models. Typically, the observations ($\mathbf{y}$ values) are assumed to be noisy, thus g above may represent a procedure for sampling $\mathbf{y}$ values from a probability distribution appropriately parameterized by the underlying state of the process model (i.e., its mean and/or variance, etc., might depend on $\mathbf{x}$).

To illustrate how this framework is used to estimate ODE model parameters, we use the following example of a logistic growth model where the observations y_i are assumed to be normally distributed with constant variance σ^2 and a mean that is proportional (by some factor c) to the state variable at time t_i, i.e., $E(y_i) = c\,x(t_i)$:

$$\frac{dx}{dt} = r\,x\left(1 - \frac{x}{K}\right), \quad x(0) = x_0, \tag{47a}$$

$$y_i \sim \text{Normal}(c\,x(t_i), \sigma^2). \tag{47b}$$

We may now consider estimating one or more of the parameters r, K, c, and x_0 within a *maximum likelihood* framework.[18] It is strongly advised that, when attempting a new-to-you parameter estimation procedure, you to implement it first

[18] It is worth pointing out that this model does not have a unique best-fit parameterization, unless certain parameters are held constant (i.e., are already known), even when estimating parameters from ideal data (e.g., a large sample with little or no noise)! This problem and the ways of resolving the issue are detailed in the latter half of this section.

on simulated data where the known "true" parameter values can be compared to estimates. Hence, for the above example, we will first simulate data by picking a series of time values (t_1, t_2, ..., t_n) and generating corresponding $x(t_i)$ values by computing a numerical solution to the above logistic equation, Eq. (47a), then the corresponding y_i values can be sampled from the normal distributions given by Eq. (47b). This is implemented in the following R code, and provides a data set with known "true" parameters that can be used to evaluate and verify that the parameter estimation procedure works as intended.

```
set.seed(6.28) # for repeatable random number generation
library(deSolve) # load the ode() function into the workspace

# Function to simulate x(t) using ode() in the deSolve package
odefunc = function(t,x,ps) { # see ?ode in R for details
  with(as.list(ps),{  # Use named values in ps as variables
    dx = r*x*(1-x/K); #    (compare to SIR code above).
    return(list(dx)); } )
}

# Simulate data. First, obtain an ODE solution
Time = seq(0,7,length=30) # 30 evenly spaced values from 0 to 7
params = c(r=2, K=500, c=0.1, sigma=2) # Model parameters
x0 = c(x=50) # labels our state variable  x  in xout below
xout = ode(y=x0, times=Time, func=odefunc, parms=params)
head(xout,2) # Display the first two rows of xout
```

```
          time        x
[1,] 0.0000000 50.00000
[2,] 0.2413793 76.29257
```

```
# Second, sample y values from the given normal distributions
mydata = cbind(Time, y = rnorm(nrow(xout),
               mean = params["c"]*xout[,2], params["sigma"]))
head(mydata,2)
```

```
          Time        y
[1,] 0.0000000 5.539212
[2,] 0.2413793 6.369287
```

```
tail(mydata,2)
```

```
          Time        y
[29,] 6.758621 52.78624
[30,] 7.000000 48.87896
```

```
# Plot process model trajectory x(t) and data (t_i, y_i)
par(mfrow=c(1,2)) # Plot results in 1 row, 2 columns
plot(Time, xout[,2], type="l", ylim=c(0,550),
     main="Process Model", ylab="x")
plot(mydata, pch=19, col="gray50", ylim=c(0,55),
     main="Data (Simulated)")
```

The remainder of this section illustrates how to carry out the parameter estimation procedure, and how to conduct additional analyses to address a commonly encountered problem where there exits multiple "best-fit" parameter estimates. This is known as an *identifiability* problem (also called an *estimability* problem), and will be discussed in greater detail below.

2.7.4 Parameter Estimation as an Optimization Problem

Continuing with the example above, let us assume that it is known that $x_0 = 50$, and we seek to estimate the other parameter values from our (simulated) data. The likelihood of a given parameter set $\theta = (r, K, c, \sigma)$ (given those n simulated pairs of data $y_{\text{data}} = (t_i, y_i)$) can be computed using the *likelihood function* defined as

$$L(r, K, c, \sigma \,|y_{\text{data}}) = \prod_{i=1}^{n} f(y_i, \text{mean} = c\,x(t_i), \text{sd} = \sigma) \tag{48}$$

where function f is the density function for a normal distribution[19] with the given mean and standard deviation, evaluated at y_i, and parameters x_0, r, and K are used to numerically find $x(t_i)$. Our goal is to computationally find the set of parameter values that maximizes L for a given data set. As stated above, it is often more practical to minimize the *negative log likelihood* function $nLL(\theta|y) \equiv -\log(L(\theta|y))$ rather than maximize the likelihood function. Thus, by properties of the natural log function,

$$\begin{aligned} nLL(\theta|y) &= -\log\Big(\prod_{i=1}^{n} f(y_i, \text{mean} = c\,x(t_i), \text{sd} = \sigma)\Big) \\ &= -\sum_{i=1}^{n} \log(f(y_i, \text{mean} = c\,x(t_i), \text{sd} = \sigma)). \end{aligned}$$

We can now frame our parameter estimation problem as a computational optimization problem. Since solutions $x(t)$ typically have no closed form, we will use numerical solutions of our process model in order to compute likelihoods.

Optimization algorithms essentially take a real-valued *objective function* (in our case, a negative log likelihood function) which defines a surface over parameter space, and from an initial set of parameter values (i.e., a point in parameter space) moves through parameter space by moving "downhill" (or "uphill") on the objective function surface until the minimum (or maximum) objective function value is found. Thus, to use generic optimization tools in software like R, we must construct a

[19] See `dnorm` in R.

negative log likelihood function that conforms to the requirements of our chosen optimization routines.

The objective functions defined below are written specifically to be used with the generic optimization methods available under `optim()` and `optimx()` in R [129]. These functions require that the first argument to the objective function be a parameter vector, and any additional arguments are for passing in additional fixed values (in this case, the data). See the help documentation for `optim` in R for further details. The examples below use the default Nelder–Mead algorithm, which can be relatively slow,[20] and does not allow for the specification of explicit constraints (e.g., we would like to require the standard deviation parameter σ be positive), but often gives good results where gradient-based methods perform poorly (for additional optimization considerations and resources, see [128, 129, 161]).

In the context of our specific application, we additionally require that all parameters are positive-valued. This is a common constraint; however, generic *unconstrained* optimization methods like Nelder–Mead often work best when allowed to freely consider all real numbers as candidate values for each unknown parameter. To impose constraints on those parameter ranges requires us to either modify our objective function—to coerce the optimization method into avoiding negative parameter values—or to instead use a constrained optimization method. When using a box-constrained optimization method in R (e.g., method `L-BFGS-B` in `optim`) the limits on the range of possible parameter values can be specified explicitly as arguments to `optim` or `optimx`. Here we only illustrate two variations on the first approach, using unconstrained optimization with parameter constraints built into the objective functions.

The first objective function (`nLL1`) implements this constraint by returning a very large value if any parameter value is not strictly positive. This introduces a discontinuity into the likelihood surface, which can cause problems for many optimization methods, especially gradient-based methods (i.e., those methods that use derivatives to "move downhill" to find objective function minima).

```
nLL1 = function(theta, tydat) {
  # To disallow negative parameter values, nLL1 returns a large
  # number if the optimization algorithm inputs any negatives.
  # (See the text and nLL() below for better alternatives.)
  if(any(theta<=0)) { return(1*10^10) }

  # Otherwise, calculate our likelihood of theta given the data.
  Times = tydat[,1] # 1st column are the times
  Ys = tydat[,2]    # 2nd column are the y values

  # Simulate a trajectory x(t) under the given parameters (theta)
  x0 = c(x=50) # For now, we assume x0 is known.
  xout = ode(y=x0 ,times=Times, func=odefunc, parms=theta)
```

[20] The performance can be improved somewhat, e.g., by setting the flag `kkt=FALSE` to avoid unnecessary computations or using `parscale` when parameter values vary by multiple orders of magnitude. See the documentation for `optim` and `optimx` for details.

```
  # Return the -log(Lik) value under a normal distribution. Here
  # we use the built-in ability to calculate log-density values
  # by setting the argument log=TRUE. See ?dnorm for details.
  return(-sum(dnorm(Ys,theta["c"]*xout[,2],
         theta["sigma"],log=TRUE)))
}
```

The second (preferred) approach, implemented in `nLL2` below, uses transformations (like those discussed in Sect. 2.6) to ensure parameter values remain strictly positive. Here, (positive-valued) parameters are transformed using the natural log and then unconstrained optimization is performed using these real-valued transformed quantities. The resulting log-transformed values can then be transformed back to positive parameter values using the exponential function $\exp(x) = e^x$. In the example above, if we want r to remain strictly positive, we can re-parameterize `nLL1` using the transformed quantity $q = \log(r)$ (where $r = \exp(q)$ is positive for all $-\infty < q < \infty$). Likewise, defining $k = \log(K)$, $C = \log(c)$, and $S = \log(\sigma)$ so that $K = \exp(k)$, $c = \exp(C)$, and $\sigma = \exp(S)$, we can rewrite `nLL1` above as a function of these new parameters $P = (q, k, C, S)$ where $\theta = \exp(P)$.

```
nLL2 = function(P, tydat) {
  Times = tydat[,1] # 1st column are the times
  Ys = tydat[,2]    # 2nd column are the y values
  # Next, obtain x(t) under the given parameters
  x0 = c(x=50) # As above, we assume x0 is known.
  xout = ode(x0, times=Times, func=odefunc,
             parms = c(r=exp(P[["q"]]), K=exp(P[["k"]])))
  # Return the -log(Lik) value under a normal distribution
  return(-sum(dnorm(Ys,
         exp(P[["C"]])*xout[,2], exp(P[["S"]]), log=T)))
}
```

While, in theory, the optimal parameter values obtained from these two approaches should be identical, in practice round-off error and other factors can lead to different outcomes. Optimization methods tend to perform better using the log-transformation with unconstrained optimization, or using an explicit constrained optimization method instead of the approach implemented in `nLL1` above.

To illustrate this, the following code obtains parameter estimates using both approaches described above. This requires that initial values are provided for the parameters we would like to estimate, as required by `optim`.

```
# Initial parameter values for optimization using nLL1
theta0 = c(r=1,  # Next, use the last y value for K, but
           # divide that y value by c to get an x value...
           K=as.numeric(mydata[nrow(mydata),2])/0.6,
           c=0.6, sigma=0.1)

# The log transformed initial values for nLL2
P0 = c(q = log(theta0[["r"]]), k = log(theta0[["K"]]),
       C = log(theta0[["c"]]), S = log(theta0[["sigma"]]) )
```

```
# The true values, for comparison (impossible with real data!)
Ptrue = c(q = log(params[["r"]]),  k = log(params[["K"]]),
          C = log(params[["c"]]),  S = log(params[["sigma"]]) )

# Fit the model both ways, then compare.
fit1 = optim(theta0, nLL1, tydat=mydata,
             control=list(maxit=1e9, reltol=1e-10))
fit2 = optim(P0, nLL2, tydat=mydata,
             control=list(maxit=1e9, reltol=1e-10))
```

The results from these two optimization runs are summarized below by showing the lowest negative log likelihood value found, the corresponding parameter values, the number of calls of the objective function (counts), and the convergence code[21] indicating that the optimization routine reported no errors.

```
# Combine rows into a single object to display the results
rbind(fit1 = c(fit1$par, nLL=fit1$value,
          counts=fit1$counts[[1]], conv.code=fit1$convergence),
      fit2 = c(exp(fit2$par), nLL=fit2$value,
          counts=fit2$counts[[1]], conv.code=fit2$convergence),
      true = c(params, nLL1(params,mydata), NA, NA) );
```

```
          r      K       c  sigma    nLL counts conv.code
fit1 1.8623 430.47 0.11738 2.2161 66.448    779         0
fit2 1.8619 430.29 0.11744 2.2171 66.448    295         0
true 2.0000 500.00 0.10000 2.0000 67.859     NA        NA
```

Observe that the estimates are effectively the same, but the number of function calls indicates that the optimization run on the log-transformed parameter values (`fit2`) converged much faster. Furthermore, notice that the best estimate is slightly off from the known ("true") values used to simulate these data, due to the added noise and finite sample size. For more on implementing and debugging this optimization approach to parameter estimation, see [128, 129, 161].

Exercise 10 In the example above, we may wish to further assume c is a proportion that cannot exceed 1. What other invertible functions might one use to map the open interval $(0, 1)$ to $\mathbb{R}$?

Challenge Problem 13 Read through R's help documentation for the `optim` function (and the `optimx` function in the `optimx` package; see also [161]), and identify one or more constrained optimization methods that can be used as an alternative to Nelder–Mead. Also, modify the code above to include a third objective function identical to `nLL1` but with no checks on the sign of the parameter values. Apply one or more constrained optimization method(s), and provide a comparative summary of your results.

[21] Convergence code 0 indicates no errors, and convergence code 10 indicates degeneracy of the Nelder–Mead simplex. The code 10 often occurs when the model is not identifiable and reaches a "flat spot" in the objective surface.

2.7.5 Recognizing Identifiability (Estimability) Problems

In practice, before we attempt using optimization to find our best-fit parameter values, we must ask one important question that was neglected in the above example: *Does this model and estimation procedure have a unique best-fit parameter set, or might there be multiple sets of parameter values that give equally good fits?* That is, phrased in terms of the geometry of the negative log likelihood surface: Is it roughly bowl shaped (with a distinct minimum) or is it more like a valley with a connected line of minima (or worse!)? In the first case, we call the model *identifiable*—it has a unique likelihood-maximizing parameter set. In the second case, we say the model is *unidentifiable*. While such problems may not affect forecasting of your observable quantities, it is often the case that addressing any such identifiability problems is essential to properly performing and drawing inferences from the parameter estimates (e.g., see [11, 33, 46, 47, 57, 141]). Below, we introduce some ways of assessing whether identifiability problems exist, and some approaches to correcting the problem.

There are two predominant causes of identifiability problems that arise in practice. First, sometimes our data lack the information needed to estimate one or more parameters. For example, if our logistic model data were restricted to just the early exponential growth phase of the trajectory shown in Fig. 5, we might be able to estimate growth rate r, but our data would lack information about the carrying capacity K. This can make K very difficult to estimate, and may result in wildly different estimates of K depending on where the estimation procedure was initialized. This can sometimes (but not always) lead to numerical problems if the optimization methods used require the existence of a unique optimum. Perhaps a worse outcome is that no numerical errors are reported, and a seemingly good (but in reality, arbitrary) estimate of K is obtained. Without any further identifiability or uncertainty analysis, the conclusions drawn from that flawed estimate would be

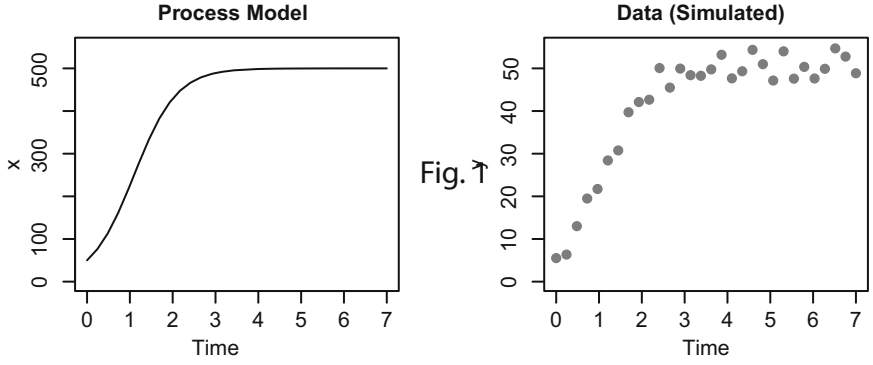

Fig. 5 A trajectory $x(t)$ of logistic model Eq. (47) (left) and a corresponding data set where the simulated data values y_i are normally distributed with a mean that is proportional to $x(t_i)$. For more details, see the corresponding code in the main text

spurious (e.g., if K being very large or very small were of great importance). This would be a *practical identifiability* problem, since it results from a shortcoming in the data, not our model.

The second cause for identifiability problems arises from an overparameterized model, e.g., where there are multiple parameter sets that yield the same outcomes for the deterministic part of our model used to define the mean of observation distributions. We call this a *structural identifiability* problem, and unlike practical identifiability problems, these persist even when estimating parameters from ideal data (e.g., full trajectories with no noise and arbitrarily large sample size).

For a simple example of a structurally unidentifiable model, consider the simple linear regression model where the slope parameter m is replaced with two a parameter expression $m = m_1 + m_2$, i.e., consider

$$E(y_i) = (m_1 + m_2)\, x_i + b. \tag{49}$$

It is no surprise that this model has infinitely many best-fit parameter sets! Specifically, if the standard regression model had a best-fit slope of say $\widehat{m} = 5$ and intercept of $\widehat{b} = 2$ then for our overparameterized model we would get equally good fits out of any set of parameters where $\widehat{m_1} + \widehat{m_2} = 5$. Notice that in this simple example, the problem has nothing to do with the amount of noise in the data, or the sample size, but it is a fundamental property of the deterministic part of our statistical model.

Since, in practice, both structural and practical identifiability problems manifest in the form of an objective function that does not have a unique and/or well defined optimum, a *practical identifiability analysis* is a good approach to assessing whether or not such problems exist. This can then be complemented by a *structural identifiability analysis* to look for identifiability problems that are caused by the deterministic part of our model, if needed.

Next, we look at some approaches to conducting these identifiability analyses to assess presence and extent of these problems, and see some ways to correct them.

2.7.6 Practical Identifiability Analysis

In practice, to assess whether or not an identifiability problem exists, the first step is often to perform a practical identifiability analysis. *Likelihood profiling* is a common way to perform a practical identifiability analysis when using a likelihood based estimation procedure [32, 46, 141], because this is often the method used for quantifying uncertainty in the parameter estimates (i.e., to approximate confidence intervals) and thus requires no additional analyses. A more simplistic alternative that can be used in other estimation frameworks (e.g., least-squares regression) is to run the estimation procedure using multiple different initial parameter values, then check to see if each converges to the same, or different, best-fit parameter estimates.

To illustrate this approach, we again consider estimating parameters for our logistic example above, but we now assume x_0 is unknown and thus a quantity

we would like to estimate. To implement this in R, the code below first defines a negative log likelihood function `nLL` (an extension of `nLL2`) that takes in the additional free parameter $X_0 = \log(x_0)$. Unlike the case above, where x_0 was a known quantity, this additional free parameter makes the model structurally unidentifiable.

Below, the estimation procedure described above was run for the full model (i.e., for `nLL`) using 1000 different initial parameter values. The pairwise relationships among the best estimates out of those 1000 runs (based on `nLL` values) are plotted in Fig. 6 to show their range of variation as well as any obvious patterns indicating trade-offs between pairs of parameters. Here x_0 is initialized using the first value in our data set, and the 1000 different initial parameter sets are obtained by random sampling from a uniform distribution so that each parameter value is within $\pm 30\%$ of the initial parameter values specified above.

```
# nLL for the full example model with log-transformed parameters
nLL = function(P, tydat) {
     Times = tydat[,1] # 1st column are the times
     Ys = tydat[,2]    # 2nd column are the y values
     xout = ode(c(x=exp(P[["X0"]])), times=Times, func=odefunc,
                parms=c(r=exp(P[["q"]]), K=exp(P[["k"]])))
     # Return the -log(Lik) value under a normal distribution
     return(-sum(dnorm(Ys, mean=exp(P[["C"]])*xout[,2],
                       sd=exp(P[["S"]]), log=TRUE)))
}

# Use the first y value for a crude initial estimate of x0
P0 = c(X0 = log(as.numeric(mydata[1,2])/0.6),
       q = log(theta0[["r"]]), k = log(theta0[["K"]]),
       C = log(theta0[["c"]]), S = log(theta0[["sigma"]]) )

# initialize a data frame to store optimization output
fits=data.frame(x0=NA,r=NA,K=NA,c=NA,sigma=NA,nLL=NA,code=NA)
set.seed(123)  # for repeatable random number generation
for(i in 1:1000) {
    # Random parameter values are within +/- 30% of true values
    P0i  = log( exp(P0) * runif(length(P0), 0.7, 1.3) )
    fiti = optim(P0i, nLL, tydat=mydata, control=list(maxit=1e5))
    fits[i,] = c(exp(fiti$par), nLL=fiti$value,
                 code=fiti$convergence)
}
# select the estimates within 0.01% of the best fit
fits = fits[order(fits$nLL,decreasing=FALSE), ]
fits2 = fits[(fits$code==0 | fits$code==10) &
             fits$nLL <= 1.0001 * min(fits$nLL, na.rm=T), ]
# Omit some extremes to clean up the graphical output below
fits2 = fits2[fits2$K != max(fits2$K), ]
fits2 = fits2[fits2$K != min(fits2$K), ]
# log-scale plots to show pairwise tradeoffs, parameter ranges
pairs(fits2[,1:5], log="xy",
      labels=c(expression(x[0]),"r","K","c",expression(sigma)))
```

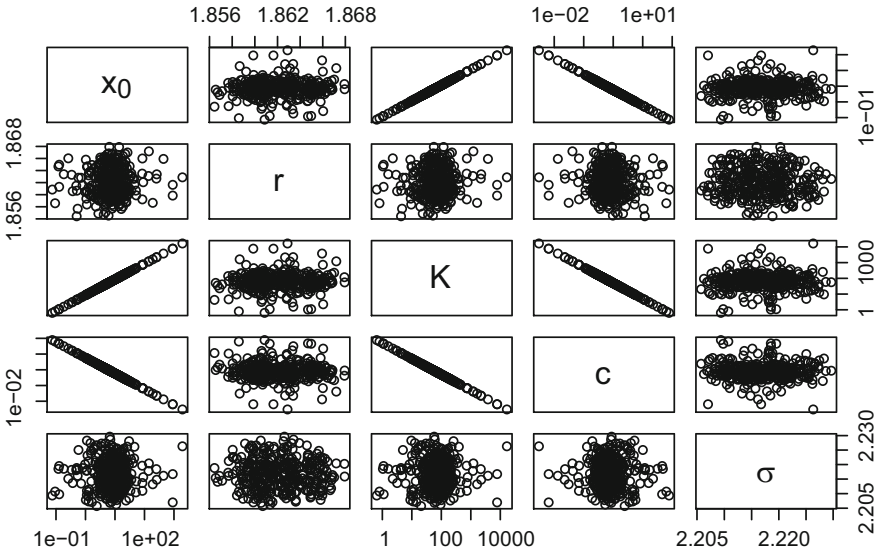

Fig. 6 Parameter estimates, on a log scale, obtained by running the optimization routine in our estimation procedure for 1000 different starting parameter sets, then taking the estimates within 0.01% of the best estimate's nLL value and plotting their pairwise relationships (note the best-fit negative log likelihood value of these 350 top fits is 66.4476 and the worst is 66.4488). These estimates (which exclude some extreme values to clarify patterns in the plots; see code for details) have the following coefficients of variation: $cv_{x_0} = 5$, $cv_r = 0.0013$, $cv_K = 5.1$, $cv_c = 3.2$, and $cv_\sigma = 0.0023$. The very close negative log likelihood values, combined with the large ranges of variation and correlations between some parameters indicate that there is an identifiability problem (i.e., there is apparently no unique best-fit parameter set, either due to there being more parameters than estimable quantities in the model, or the optimization method lacks the sensitivity to find an existing optimum due to a nearly flat objective function near that optimum). It seems only r and σ appear to be identifiable. Linear relationships on these log–log plots indicate linear, inverse, or power law relationships between best-fit parameters. These estimates are slightly better (lower negative log likelihood values) than the previous figure, where x_0 was held constant at a non-optimal value. See the code in the main text for further details

```
rownames(fits2)=c() # remove row numbers for the output below
head(fits2[,1:6],5) # Show the 5 best parameter estimates
```

```
        x0      r        K       c  sigma    nLL
1 19.35396 1.8624 166.5768 0.30336 2.2170 66.448
2 11.51700 1.8623  99.1655 0.50959 2.2196 66.448
3  5.25086 1.8615  45.1620 1.11897 2.2166 66.448
4  3.55392 1.8610  30.5587 1.65359 2.2161 66.448
5  0.79901 1.8616   6.8779 7.34744 2.2188 66.448
```

```
summary(fits2[,1:3])
```

```
       x0                r                K         
 Min.   :   0.07   Min.   :1.86   Min.   :    0.6  
 1st Qu.:   4.31   1st Qu.:1.86   1st Qu.:   37.2
```

```
Median :   7.25   Median :1.86   Median :   62.6
Mean   :  25.21   Mean   :1.86   Mean   :  217.2
3rd Qu.:  13.79   3rd Qu.:1.86   3rd Qu.:  119.1
Max.   :1934.69   Max.   :1.87   Max.   :16685.9
```

```
summary(fits2[,4:6])
```

```
       c                sigma           nLL
Min.   : 0.003   Min.   :2.21   Min.   :66.4
1st Qu.: 0.424   1st Qu.:2.21   1st Qu.:66.4
Median : 0.807   Median :2.22   Median :66.4
Mean   : 2.278   Mean   :2.22   Mean   :66.4
3rd Qu.: 1.359   3rd Qu.:2.22   3rd Qu.:66.4
Max.   :78.369   Max.   :2.23   Max.   :66.4
```

Note that, while the estimates for r and σ are tightly clustered, the others vary by multiple orders of magnitude! These wildly different parameter estimates, each fitting equally well to the data (i.e., they have nearly identical negative log likelihood values), are a strong indication of an identifiability problem.

Next, we conduct a structural identifiability analysis to assess whether or not these problems arise from a structural identifiability problem (and would therefore persist even for data with a very large sample size and little to no noise). Importantly, even though this does not always yield the desired results (see [46, 121] and references therein), it can often provide equations that exactly describe the parameter trade-offs evident in Fig. 6.

2.7.7 Structural Identifiability Analysis

Structural identifiability problems can often be identified (and sometimes, corrected for) by a structural identifiability analysis (e.g., see [11, 33, 46, 57, 122, 123, 141, 146] and references therein) that determines whether or not there is a one-to-one mapping between the model parameters and the quantities in the model that can be estimated (e.g., in our simple linear regression example above, Eq. (49), there is no one-to-one mapping between parameters m_1, m_2, and b and the estimable slope m and intercept b). These estimable quantities are also often called (identifiable) *parameter combinations*. A commonly used method of conducting a structural identifiability analysis is the *input–output equation approach* detailed in the references above, and the benefit of this approach is that it can give you generic analytical relationships (e.g., like the $m = m_1 + m_2$ relationship in Eq. (49)) that can be used to correct the problem, e.g., by fixing one of the problem parameters in order to make one or more others uniquely estimable.

Since the above practical identifiability analysis indicates an identifiability problem, we here apply the input–output equation approach to determine whether or not Eq. (46) is a structurally unidentifiable model, and if so, to discover the identifiable parameter combinations.

To do this, we first assume a zero-variance (deterministic) observation variable $y(t) = c\,x(t)$ and differentiate it with respect to t. Second, we substitute the process model Eq. (47a) and substitute $x = y/c$ to obtain a differential equation that is only in terms of the observation variable y. The resulting equation is referred to as the *input–output equation*, and for our logistic example it is given by

$$\frac{dy}{dt} - r\,y - \frac{r}{c\,K}y^2 = 0, \quad y(0) = c\,x_0. \tag{50}$$

Since this equation is a monic polynomial in terms of y and derivatives of y, it follows that are our identifiable parameter combinations (i.e., the quantities we can estimate from data) are the two coefficients in the input–output equation and the initial condition, i.e., one can only estimate values for

$$r, \quad \frac{r}{c\,K}, \quad \text{and} \quad c\,x_0. \tag{51}$$

What does this imply about the estimability of our model parameters? If there were a one-to-one mapping between these three identifiable parameter combinations and our four parameters, then we would expect there to be a unique best parameter estimate (given ideal data). But, note that we have three identifiable parameter combinations and four model parameters, hence we cannot have a one-to-one mapping between them. Thus, our example logistic model is structurally unidentifiable.

Obtaining these expressions for the trade-offs between parameters are a major benefit of conducting a structural identifiability analysis. From the above example, we have learned that we should expect a unique estimate for r, but any parameter values for c, x_0, and K that yield the same "best-fit" values of $c\,K$ and $c\,x_0$ will fit equally well. Thus, the pairwise plots in Fig. 6 should show an inverse relationship between c and x_0 and between c and K (or, a linear relationship with negative slope when viewed on a log–log plot), and a linear relationship with positive slope between x_0 and K. Also, by fixing one parameter—e.g., c or K or x_0 (but not r)—there would then be a one-to-one mapping between the three identifiable parameter combinations and the three unknown model parameters. Often, in applications, some parameters can be assigned values based on independent experiments, so holding constant the most well known of these parameter values might correct these identifiability problems and thus permit investigators to estimate the other unknown parameters from data.

Exercise 11 In the logistic example above, prove that fixing c, K, or x_0 corrects the identifiability problem, whereas fixing r does not. To do this, suppose there are two different sets of best-fit parameter values (r_1, K_1, x_{01}, c_1) and (r_2, K_2, x_{02}, c_2) that yield the same identifiable quantities (51), i.e., $c_1\,x_{01} = c_2\,x_{02}$, $r_1 = r_2$, and $\frac{r_1}{c_1\,K_1} = \frac{r_2}{c_2\,K_2}$. Fix each parameter in turn (e.g., fix $c_1 = c_2 = c$), and show whether or not the resulting system of equations permits two distinct parameter sets.

Exercise 12 Structural identifiability results can provide explicit functional forms describing trade-offs among unidentifiable parameters, like those shown in Fig. 6. These can be found by writing our identifiable quantities explicitly and solving for the implied relationships between parameters. For the example above, we have the identifiable quantities $\alpha_1 = c\,x_0$, $\alpha_2 = r$, and $\alpha_3 = r/(c\,K)$, which are uniquely estimable from an ideal data set. Use these expressions to show that the ratio K/x_0 is constant, and thus K and x_0 have a positive linear relationship consistent with Fig. 6. Verify the other pairwise relationships suggested by the log–log plots in Fig. 6.

Let us now use these results to correct our estimation procedure. Suppose that, in our full logistic growth example, we had good reason to believe that our sampling design captured roughly 10% of the population at any given time. That is, we can reasonably assume $c = 0.10$. In that case, our negative log likelihood function could be modified so that it takes in a vector of parameter values that does not include c (and instead holds it fixed at 0.10), and likewise c would be omitted from initial parameter value vector provided to `optim`.

These modifications are implemented in the practical identifiability analysis below, which replicates the parameter estimation procedure 500 times for different initial parameter sets. The results are shown in Fig. 7, where pairwise scatterplots of the best-fit parameter estimates can be used to assess whether or not replicates seem to be converging to a unique best-fit parameter set.

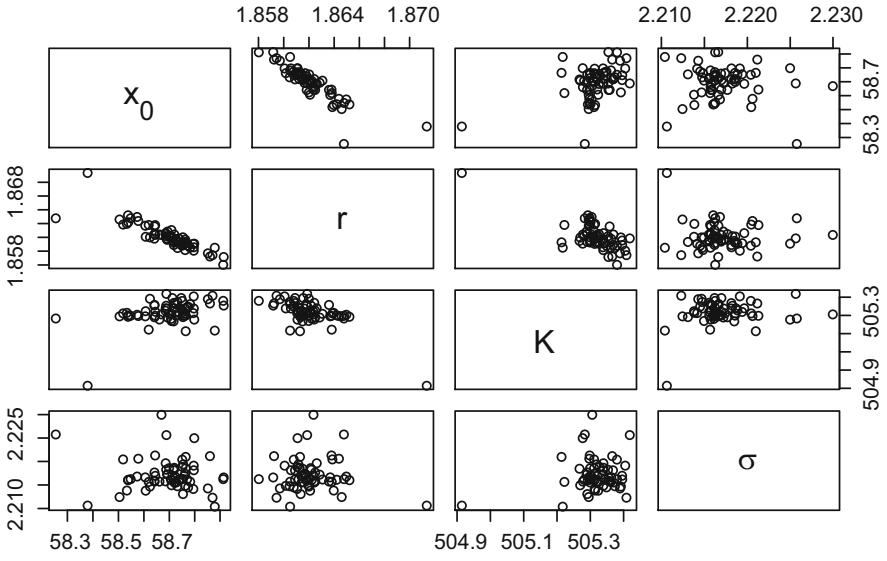

Fig. 7 Pairwise plots of parameter estimates for the logistic model Eq. (47) where $c = 0.1$ is fixed to correct the structural unidentifiability of the model (see Fig. 6). To illustrate the improved convergence of estimates, 500 different initial parameter sets were used, and for the 65 of those estimates with a negative log likelihood value within 0.01% of the best estimate, these had the coefficients of variation: $cv_{x_0} = 0.002$, $cv_r = 0.001$, $cv_K = 0.00013$, and $cv_\sigma = 0.0015$

```
nLLc = function(P, tydat) {
   Times = tydat[,1] # 1st column = times; 2nd = y values
   Ys = tydat[,2]
   x0 = c(x=exp(P[["X0"]]))
   xout = ode(x0, times=Times, func=odefunc,
              parms = c(r=exp(P[["q"]]),K=exp(P[["k"]])))
   # Return -log(Lik) value as in previous examples
   return(-sum(dnorm(Ys, 0.1*xout[,2], exp(P[["S"]]),log=T)))
}

# initial P values for all but C=log(c)
P00 = P0[-4] # exclude the 4th element, C=log(c), from P0

# Pairwise plots to verify the identifiability problem is fixed
fits = data.frame(x0=NA,r=NA,K=NA,sigma=NA,nLL=NA,code=NA)
for(i in 1:500) {
   # Random parameter values are within +/- 20% of true values
   P00i = log( exp(P00) * runif(length(P00),0.8, 1.2) )
   fiti = optim(P00i, nLLc, tydat=mydata,
                control=list(maxit=1e6, reltol=1e-10))
   fits[i,] = c(exp(fiti$par), nLL=fiti$value,
                code = fiti$convergence)
}
# Display the best-fit estimates and the pairwise scatterplots
fits  = fits[order(fits$nLL, decreasing=FALSE), ]
# Only include those with a nLL withing 0.01% of the best fit
fits2 = fits[(fits$code==0 | fits$code==10) &
             fits$nLL <= 1.0001 * min(fits$nLL, na.rm=T), 1:5]
summary(fits2[,-5]) # Omit nLL quartiles (all are very close)
```

```
       x0               r               K              sigma
 Min.   :58.25   Min.   :1.858   Min.   :504.9   Min.   :2.210
 1st Qu.:58.64   1st Qu.:1.861   1st Qu.:505.3   1st Qu.:2.216
 Median :58.72   Median :1.862   Median :505.3   Median :2.216
 Mean   :58.70   Mean   :1.862   Mean   :505.3   Mean   :2.217
 3rd Qu.:58.76   3rd Qu.:1.863   3rd Qu.:505.4   3rd Qu.:2.219
 Max.   :58.91   Max.   :1.871   Max.   :505.4   Max.   :2.230
```

```
# Estimates should be close to params, used to simulate the data,
# but not exact due to noise and finite sample size.
c(x0=50, params)  # The values used to simulate the data.
```

```
   x0     r     K     c sigma
 50.0   2.0 500.0   0.1   2.0
```

```
rownames(fits2)=c() # erase the old row numbers from `fits
head(fits2[,1:5], 4) # The best estimates obtained are...
```

```
        x0        r        K    sigma      nLL
1 58.68502 1.862500 505.3260 2.215778 66.44762
2 58.69906 1.862127 505.3115 2.218590 66.44763
3 58.73725 1.861839 505.3141 2.216148 66.44763
4 58.72716 1.861420 505.3199 2.215563 66.44764
```

```
pairs(fits2[,1:4], log="xy",
      labels=c(expression(x[0]), "r", "K", expression(sigma)))
```

From these pairwise plots of parameter estimates, we see there is still some small variation in our estimates, but the best estimates are now all clustered tightly around the same value, and close to the true values used to simulate the data (note the estimates would deviate more from the true values if we had fixed c at a value away from 0.1). Some deviation from the true values is always expected when the data set is noisy and has a small to moderate number of data points. Note that some parameters show correlations, e.g., r and x_0, indicating that there is some slight trade-off between them. This is often the case when the objective surface (i.e., the negative log likelihood surface) has a unique optimum but it lies along a nearly flat trough with only a slight amount of curvature near the optimum, which can cause the optimization routine to end prematurely. This suggests that our estimates might be refined, if needed, using a different optimization method or by modifying the control parameters so that the criteria for when to conclude that an optimum was found are more conservative.

Exercise 13 Suppose data for the θ-logistic model (see Eq. (36)) were simulated only for a short period of time starting with $x_0 \ll K$ so that only the exponential growth phase was reflected in the data. Would you expect to be able to estimate K or the exponent θ with much certainty? Likewise, for the logistic or θ-logistic models, would you expect to be able to estimate r with any certainty if the data were only sampled from a trajectory that started at carrying capacity $x_0 = K$? Discuss these in the context of *structural* versus *practical identifiability*.

Challenge Problem 14 There are still some correlations in Fig. 7 suggesting some "trade-off" between estimated parameter values. Explore different optimization routines and control settings. Are the correlations and variation in estimates improved by a more finely tuned optimization procedure? How does increasing or decreasing the noise, or the sample size, of our simulated data affect these estimates?

Challenge Problem 15 (Standardizing Parameters for Optimization) The Nelder–Mead algorithm used in the examples above (as well as many other optimization methods) may not perform well in situations where the parameter values in question span many orders of magnitude. A simple way to correct this problem is to standardize the optimization parameters. To do this, given p unknown parameter values, one can introduce a new set of p scalars (initially set at 1) that can be multiplied by a base parameter vector to create a new parameter vector. Optimization is then done on these multipliers using a new objective function that takes these new parameters as inputs, multiplies them by a base parameter set, and then computes a negative log likelihood value for the resulting set of model parameters using the original `nLL` function. To do this, begin with `nLL` above and a base parameter set `params0` such that the usual optimization call would be `optim(params0, nLL, ...)`. Next, write a new function `nLLscaled` that takes a vector of parameter values `M` (initially all set to 1) as an input, and also a second parameter vector `basepars` which can be passed in as an argument to

`optim` using `optim(M, nLLscaled, basepars=params0, ...)`. The optimum parameter values returned by `optim` can be multiplied by `params0` to obtain the best-fit model parameters.

Implement this approach for the logistic example above, where $c = 0.1$ is assumed to be known, and compare its performance (e.g., in terms of rates of convergence and goodness of fit) to the unstandardized approach. Finally, note that `optim` and `optimx` have a built-in way of achieving this scaling using the `parscale` option (see the documentation for `optim` and `optimx`). Add this built-in option to your comparison of approaches.

2.7.8 Statistical Analyses Beyond Parameter Estimation

Once we are happy with our parameter estimation procedure, other statistical analyses can and should be conducted. For example, we typically should conduct some sort of *uncertainty quantification* to assess how much certainty we have in our parameter estimates. Uncertainty quantification is almost always necessary, as it provides an essential context for evaluating any inferences drawn from estimates obtained by fitting models to data. In the context presented above, uncertainty quantification can be done by approximating confidence intervals for the parameter estimates using *likelihood profiling* techniques, which, as mentioned above, can also be used for practical identifiability analysis [47]. Resampling methods like bootstrapping or other approaches to uncertainty quantification may also be used, e.g., see [1, 2, 19, 21, 32, 70, 79, 114] and references therein. Model comparisons can also be made, e.g., using likelihood ratio tests, relative AIC values, or similar criteria (e.g., see [21, 123] and references therein). While uncertainty quantification is not treated here in detail, readers fitting models to data are strongly encouraged to quantify the level of uncertainty in their parameter estimates, forecasts, etc. as part of any presentation or discussion of their final results.

2.7.9 Alternative Approaches to Parameter Estimation and Uncertainty Quantification

New and improved methods for parameter estimation and uncertainty quantification for ODE-based statistical models are still being actively developed, but in practice the above approach is a good place to start. As mentioned at the start of this section, there are other approaches than the one detailed above and some have even been implemented as R packages. For example, the `deBInfer` package can be used to implement Bayesian parameter estimation for ODE models [19]. A Bayesian framework offers many advantages, including a straightforward avenue to uncertainty quantification using the posterior parameter distributions. Sometimes likelihood surfaces that involve solutions to ODEs can deviate greatly from the ideal "hump shaped" likelihood surface with a distinct optimum, and thus make for a difficult computational optimization problem. Alternative methods have been

developed that instead use a different measure of how well the model fits the data by comparing the right hand side of the ODE model (i.e., derivatives of the solution curves) to derivatives of a non-parametric smooth curve fit to time series data [139]. The package `CollocInfer` implements one of these *functional data analysis* methods in R [87, 88], and there is also Matlab software for use with the FDA package [86]. See also [14, 32] and the *probe matching* approach discussed in [50, 97, 151].

2.7.10 Closing Remarks on Fitting ODE Models to Data

In general, it is strongly advised that any parameter estimation procedure be assessed using data simulated from the exact statistical model being used, to verify that the implementation was done correctly and that the estimates look reasonable (additionally, this simulation-estimation check can be repeated for many simulated data sets to generate a "boot-strapped" empirical distribution for the estimates, which can then be used to assess estimator properties, e.g., to quantify bias). An identifiability analysis also further helps to ensure the parameter estimation procedure works as intended. If optimization challenges arise, other methods are available by selecting a different `method` option in `optim`, or multiple methods can be compared using `optimx` in the R package `optimx`. For even more optimization options in R, see the CRAN Task View page on Optimization [161] which lists additional methods available in various contributed R packages.

3 Identifying and Modifying Model Assumptions

The sections above introduce some useful methods for analyzing ODE models or otherwise using them in an applied scientific setting. They also introduce some of the simple ways in which new ODE models can be derived by altering the assumptions of an existing model. In the sections that follow, we take a closer look at the task critically evaluating ODE model assumptions, and formulating new models that in some cases lead to new non-ODE models.

To aid in identifying assumptions associated with a given ODE model that might be altered to yield a new ODE (or non-ODE) model, it helps to consider a few general features of ODE models. First, there is the continuous (vs. discrete) nature of ODE models, both in time and state space, as well as the deterministic (vs. stochastic) nature of ODE models. Modifying one or both of these assumptions may be both scientifically fruitful and mathematically interesting to investigate. Relaxing the memoryless (vs. history-dependent) nature of ODEs can often yield different results from a corresponding set of ODEs, as can altering the assumptions that a system is autonomous (vs. driven by external forcing terms), non-spatial (vs. being explicitly spatial), well-mixed, and/or composed of identical (vs. heterogeneous) individuals. Modifying model assumptions like those listed above will often lead to

opportunities to formulate new non-ODE models, while altering other assumptions is perhaps more likely to yield new ODE models [4, 6, 45, 51, 100].

To illustrate, let us revisit the SIR model Eqs. (8) and its assumptions. First, because ODEs are inherently memoryless (i.e., the future state of the system depends only on the current state of the system, not the past), introducing any dependence on past states of the system may yield a non-ODE model, unless the necessary information about that history can be incorporated into the state variable values at the present time (e.g., the addition of an exposed class and the resulting SEIR model is a way of accounting for history by assuming an exponentially distributed lag that manifests as an intermediate exposed state). As stated above, the dwell times in states S and I follow Poisson process first event time distributions (e.g., the duration if infectiousness is exponentially distributed with rate γ), and altering those distributions (as shown below) can yield ODE and non-ODE models.

Second, the SIR model is seemingly agnostic of any spatial relationships between individuals, but (like many ODE models) it actually implicitly assumes that the system is well mixed as opposed to assuming a contact process between individuals that is, e.g., limited to nearest neighbors over some spatial domain. This implicit assumption also implies no other sources of spatial heterogeneity, e.g., areas where transmission rates are higher or lower than other areas.

Third, in the basic SIR model, individuals are assumed to be identical (or identically distributed) with respect to infectiousness and recovery times, as opposed to assuming multiple categories of individuals differing in their transmission risk (e.g., in modeling sexually transmitted infections, modelers typically acknowledge that sexual interactions are not random, so individuals are often partitioned into age classes or into groups that engage in low- versus high-risk behavior) or differ in disease progression (e.g., vaccinated versus unvaccinated individuals, or individuals that do or do not have a certain genotype that confers some protection from disease).

Fourth, the SIR model also assumes constant rates of per capita infection and recovery, versus rates that vary over time. However, the transmission rate for many childhood diseases is strongly driven by whether or not school is in session, as this increases the contact rate among infected and susceptible children.

Fifth, the SIR model is also continuous in both time and state space, which may not be desirable if one seeks to quantify the time it takes for an epidemic to end (i.e., for $I(t) = 0$, which cannot happen in finite time under the ODE model) or if one wants to use the SIR model as part of a statistical model for the analysis of weekly case count data, which might be more tractable with a stochastic discrete-time, discrete-state model.

Lastly, the SIR model makes very specific assumptions about the functional forms of the transmission and recovery rates. It is possible to instead define models that include more mathematically general terms (e.g., instead of defining the force of infection as $\lambda(I) \equiv \beta I$ one could more generally assume that the force of infection λ is instead some unknown strictly increasing function of I—and possibly a function of S and R too—where $\lambda(0) = 0$) which are quite amenable to mathematical analyses to explore what results can be obtained that apply across this broader class of models. This kind of model generalization based on more specific models can go

a long way to draw more general conclusions from specific study systems, and can also provide important context for drawing attention to the full range of dynamics that one might observe in similar real systems (e.g., see [35]).

Modelers strive to find an appropriate balance between simplicity and reality. As you can see from the SIR model assumptions listed above, there are often myriad ways of altering the assumptions of an existing ODE model. Some of these may be important to investigate, e.g., to better understand how a certain simplifying assumption might impact a key results obtained from the model. On the other hand, incorporating too many complicating assumptions can quickly make models too complex to analyze, diminishing their utility for applications.

Students can often find an interesting research project by focusing on a single ODE model assumption (e.g., the presence or absence of an Exposed class in the SIR model) and that might be altered to yield a new set of ODEs that are still simple enough to analyze and compare to the original model. But, students are particularly encouraged to consider altering those simplifying assumptions that primarily serve to ensure the model is a system of ODEs, especially when such assumptions might seem to be an egregious oversimplification given the research question at hand, and given the specifics of the real-world system(s) being modeled.

In the next few sections, we briefly introduce some approaches to generalizing, or otherwise altering, these ODE model assumptions. To give a full treatment of the analytical and computational aspects of deriving and analyzing these models is beyond the scope of this chapter, so instead an effort has been made to provide references to resources that can be consulted to dig deeper into these topics.

3.1 Autonomous ODEs to Non-autonomous ODEs

Autonomous ODE models, which only depend on the current state of the system (i.e., that could be written with $f(\mathbf{x}, \theta)$ in the context of Eq. (1)), assume that the system dynamics are not governed by any other time-varying quantities. This is often a major simplifying assumption for biological systems where, for example, the true rates of growth, reproduction, and survival may vary greatly over time following changes in temperature, light levels, or evolved seasonal patterns of behavior. Accounting for this time dependence is often accomplished by incorporating time-varying terms into ODEs, especially when incorporating strong seasonal or other periodically varying quantities into the model (e.g., seasonal transmission patterns in infectious diseases; e.g., see [34, 71] and references therein). Aperiodic and/or stochastic environmental forcing terms can also often be represented by continuous functions of time, e.g., to include temperature dependencies in a population growth model using hourly temperature data interpolated to create a continuous function of time.

While the choice to use a non-autonomous ODE model may preclude something like an equilibrium stability analysis (since the notion of an equilibrium point is often irrelevant when the system is being forced), however such extensions can often

make for a great follow-up study of a system previously modeled with autonomous ODEs [44, 143, 158]. Mathematical analysis can be done in some cases, e.g., where the forcing functions are bounded around some mean or are periodic (e.g., see [15, 108]), or where they can be approximated by step functions thus allowing one to represent the model with autonomous ODEs over specific time intervals [89].

For a simple example of a seasonally forced model, consider a logistic growth model where we have separated out the birth and death processes in the form

$$\frac{dx}{dt} = b\,x - (d + \mu\,x)\,x = (b-d)\,x\left(1 - \frac{x}{(b-d)/\mu}\right) \tag{52}$$

where the birth rate is modeled by an exponential-growth-like term with rate b, and the per capita mortality rate $d + \mu\,x$ is *density dependent* in that it increases from a baseline level d at rate $\mu > 0$ with increasing population size x. Rewritten in the standard logistic equation form (Eq. (21)), we see the model has an exponential growth rate $r = b - d$ (we may assume $b > d$ to ensure population growth) and a carrying capacity $K = (b-d)/\mu = r/\mu$.

The following exercises illustrate how one might modify such a model to include time-varying processes.

Exercise 14 Suppose Eq. (52) was to be used to model a population with density-dependent growth, where temperature was either to be held constant in one set of experiments or cycled between high and low temperatures to mimic a day–night temperature cycle. To make this model non-autonomous, one could make b a positive-valued function of time, and/or d a positive-valued function of time, and/or the quantity $b - d$ a function of time (e.g., by multiplying it by a function that oscillates between values a little above and a little below 1, and whose long-term average is 1), and/or μ a positive-valued function of time. For what scenarios might one of these be more appropriate than the other?

Challenge Problem 16 Numerically investigate two or more instances of the modified logistic models described in the exercise above.

Challenge Problem 17 Write down equations for a modified SIR model with a periodic transmission rate $\beta(t)$, e.g., using sine, cosine, or step functions (i.e., piecewise constant functions) that alternate periodically between two positive values. Modify the R code in previous sections to find numerical solutions to this seasonally forced SIR model, and compare model trajectories under different scenarios.

3.2 Deriving Deterministic Discrete-Time Models

For some applications, models that change according to discrete-time steps are sometimes desirable, thus one may want to modifying the assumptions of an existing ODE model to derive an analogous deterministic discrete-time model (or, as

discussed in the next section, a stochastic discrete-time model). For example, one might want to take an existing model of a continuously breeding organism and modify it for use with an annual plant or an animal population that has a very distinct breeding period each year (e.g., salmon or various insects, like mayflies). Another example may be a desire to consider a discrete-time analog of an ODE model as part of a statistical model to use with, e.g., weekly or monthly data. This is often accomplished using an exact or approximate representation of how the (continuous-time) ODE model trajectories change over a fixed time interval. For example, in some cases a discrete-time map can be obtained when ODE model solutions can be found analytically as in [89]. Alternatively, over short time steps, Euler's method gives an approximate discrete-time map $x(t+\Delta t) = x(t) + f(x(t), \theta)\Delta t$. However, there is some merit in taking a different approach by going back to a set of system-specific assumptions and re-deriving an appropriate model from first principles.

For examples of biological applications of discrete-time models, readers are encouraged to explore the literature. Many discrete-time models exist for studying population dynamics, and various texts address formulating and analyzing discrete-time models, including [4, 45, 50, 100, 126].

Exercise 15 Take the standard logistic model (or the form above, written in terms of b, d, and μ) and use integration by parts to verify the analytical solution curve

$$x(t) = \frac{x_0\, K}{x_0 + (K - x_0)\exp(-r\, t)}, \tag{53}$$

starting at $x(0) = x_0$. Next, use that solution to find α and β such that this system has a discrete-time map

$$x(k + \Delta t) = \frac{x(k)}{\alpha + \beta\, x(k)}. \tag{54}$$

Hint: Consider the solution curve from time $t = k$ to $t = k + \Delta t$.

Challenge Problem 18 Use a `for` loop to simulate a trajectory of the above discrete map analogue of the continuous-time logistic equation. Use code from previous sections to obtain a numerical solution to the continuous-time logistic model, and also implement Eq. (53) and overlay these three trajectories in one plot.

Challenge Problem 19 Construct a discrete-time model (using the results from the exercise above, and Eq. (52)) for the following scenario. Assume a migratory population spends a portion of the year ($0 < T_1 < 1$) on the breeding grounds, where the population grows according to a logistic equation, then a proportion ρ_1 survive a quick (assume it is instantaneous) migration to the wintering grounds. Once there, natural mortality leads to an exponential decline in population size over the remainder of the year. Then, of the remaining individuals, a fraction ρ_2 survive the return trip to the breeding grounds to start another annual cycle. Construct a map $x(t+1) = f(x(t))$ that reflects all of these processes described above. Use resources

like one of the references given above to do an analytical (or simulation-based) equilibrium stability analysis of this map (see also [89, 150] for similar models).

3.3 Deriving Stochastic Models

Revising the assumptions of an ODE model to can sometimes result in the need to derive a stochastic model, e.g., to investigate the role of demographic stochasticity arising from individual births and deaths in a population, or to alter implicit assumptions like the probability distribution describing the duration of time individuals are infectious in an SIR-type disease model, or to use in deriving a new deterministic model. Stochastic models can take various forms, ranging from Markov chains to discrete maps to stochastic differential equations (SDEs), just to name a few. This section introduces some common types of the stochastic dynamic models found in applications. For further reading, see [3, 6, 21, 45, 131] and references therein.

3.3.1 Continuous-Time, Discrete-State Stochastic Models

A natural stochastic model analogue of a given ODE model (if it can be viewed as a mean field model of some unspecified stochastic process) is the particular continuous-time stochastic process based on Poisson processes mentioned in Sect. 2.1. This is often used to add demographic stochasticity to a model, or alter implicit individual-level assumptions to derive a new mean field model. This process can often be derived via the approaches outlined in texts like [3, 6] and references therein.

Another way to construct this model, especially for simulation purposes, is the Stochastic Simulation Algorithm (SSA; also known as the Gillespie algorithm) [68, 69, 133]. In short, the SSA is a straightforward algorithm for simulating the stochastic analog of a mean field ODE using the Poisson process intuition illustrated in the exponential growth example in Sect. 2.1. There are various extensions and refinements of the SSA that an interested reader may wish to pursue, e.g., for time-varying processes [162] or the simulation of large systems where computational efficiency is important [26, 27, 31, 124]. Here we introduce the basic SSA algorithm.

To implement the SSA we first must identify all state transition events that lead to a change in state variables, the corresponding state vector changes for each type of event, and the corresponding rates for each event type. For example, in our SIR example, the two basic event types are the transition from susceptible to infected and the transition from infected to recovered, as detailed in Table 1. This information is used to iterate the following two main steps of the SSA: first, simulate the time to the next event, then determine which event type occurs.

Table 1 Possible changes to the state vector $\mathbf{x} = (S,\ I,\ R)$ in a stochastic discrete-state discrete-time SIR model with time step Δt, and their approximate probabilities

Event	Update vector	Probability = Rate$\cdot\Delta t$
Transmission	$(\Delta S,\ \Delta I,\ \Delta R) = (-1,\ 1,\ 0)$	$\beta\, S\, I\, \Delta t$
Recovery	$(\Delta S,\ \Delta I,\ \Delta R) = (0,\ -1,\ 1)$	$\gamma\, I\, \Delta t$

First, to simulate the time to the next event we need to specify its probability distribution. As mentioned in Sect. 2.1, the rate terms in our ODE that correspond to these different transition events can be interpreted as Poisson process rates. In an autonomous system, the time to the next event will be exponentially distributed with a rate given by the sum of each event rate, since these rates depend only on the state of the system, and so between events those rates are constant.[22] Once this overall event rate is calculated, the time to the next event is sampled from the given exponential distribution.

The second step in the SSA is to determine which event type occurred, so the state variables can be updated accordingly. To do this requires using the fact that the probability of the ith event type occurring, p_i, is given by the ratio of that event rate (i.e., the corresponding term from the ODE) divided by the total rate. To clarify this step, consider again the SIR model where our two events have corresponding rates βSI (infection event) and $\gamma\, I$ (recovery event), i.e., the probabilities of a new infection or a recovery are approximately $\beta SI\Delta t$ and $\gamma\, I\Delta t$, respectively. Here we specify the overall rate of a transmission event occurring, as opposed to the per capita rate. Then the probability p that the next event is a transmission event is given by $p = \beta SI/(\beta SI + \gamma\, I)$. Thus, the event type is determined by sampling from this discrete probability distribution defined by these relative rates. This sampling is done using the inverse transform sampling method [40] (see example below), and then the state variables are updated accordingly. This process is then repeated to obtain the desired simulated trajectory of the stochastic model.[23]

To illustrate how to apply the Stochastic Simulation Algorithm, it is implemented below in R for the basic SIR model. The overall rate r that defines the time until the next event is $r = \beta SI + \gamma I$ as mentioned above, thus the probabilities of an event being a transmission or recovery event are $\beta SI/r$ and $\gamma I/r$, respectively. To sample which event occurs, we simply sample from a Uniform(0,1) distribution and check whether the sampled value falls below $\beta SI/r$ (indicating a transmission event) or above that value (indicating a recovery event).

These two steps are repeated until a specified time is reached, or until $I(t) = 0$ (at which point the state values become fixed for all future times).

[22] Since the time to the next event under multiple different Poisson processes—one for each event type—is the minimum of the corresponding exponentially distributed event times, it is itself exponentially distributed with a rate that is the sum of the individual rates.

[23] Additional intuition for this algorithm can be found in the section in [92] regarding "competing clocks" in a Poisson process framework.

```
# Using the same parameters and initial conditions above...
set.seed(8675309) # Repeatable random number generation
X = cbind(Time=c(0), S=c(998), I=c(2), R=c(0)) # Store output
B=0.0003 # parameter values
g=0.1

# Next a matrix of "update vectors" to add to the
# current state of the system, i.e., x(t), when either a
# transmission (row 1) or recovery (row 2) event occurs.
update = matrix(c(-1,1,0,  0,-1,1), nrow=2, ncol=3, byrow=TRUE)
# display the two row vectors
update
```

```
     [,1] [,2] [,3]
[1,]   -1    1    0
[2,]    0   -1    1
```

```
# Stochastic Simulation Algorithm iteration using a while loop
i = 1 # a counter for our while loop iterations
while(X[i,1] < 150 & X[i,3] > 0) { # while Time < 150 and I>0 ...
   rates = c(B*X[i,2]*X[i,3] , g*X[i,3]) # indiv. event rates
   r = sum(rates) # total rate of some event happening
   tstep = rexp(1,r) # time to the next event is Exponential(r)

   # Determine which event occurs
   p = rates/r  # Probability vector for each event type
   u = runif(1) # Sample a Uniform(0,1)
   j = which(u <= cumsum(p))[1] # Sample which event occurred
   # The cumulative sum defines intervals [0,p1], [p1,p1+p2] ...
   # with interval lengths p1, p2, ... The function which(...)
   # returns the index for each interval including and above u.
   # Index [1] returns the 1st of these -- the one containing u.
   # Thus, Prob(j=1)=p1, Prob(j=2)=p2, etc.

   # Store updated time and the state variables, in new row of X
   X = rbind(X, c(X[i,1] + tstep, X[i,2:4] + update[j,]))
   i = i+1 # increment our row index for X
} # end of while() loop
if(X[i,1] < 150) { # If I reached 0, extend simulation to t=150
   X = rbind(X, c(150, X[nrow(X),2:4]))
}
head(X,3) # Display first and last few simulated values
```

```
         Time   S I R
[1,] 0.000000 998 2 0
[2,] 2.080826 997 3 0
[3,] 2.523152 997 2 1
```

```
tail(X,3)
```

```
            Time  S I   R
[1884,] 119.9396 57 1 942
[1885,] 135.8091 57 0 943
[1886,] 150.0000 57 0 943
```

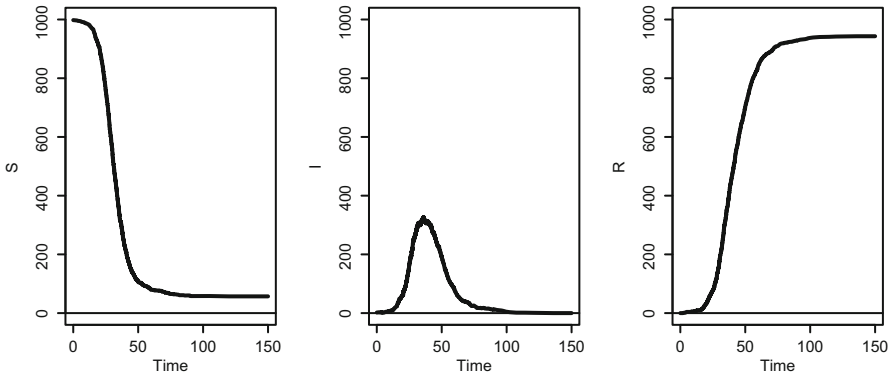

Fig. 8 One realization of the Stochastic Simulation Algorithm applied to the SIR model, which yields a simulated trajectory under the discrete-state, continuous-time analogue of the ODE-based SIR model defined by Eqs. (8). Compare to the ODE model trajectory in Figs. 2 and 3. See the code in the main text for further details

```
# Plot the stochastic trajectories as before
Time = X[,1]
par(mfrow = c(1,3)) # three panels in 1 row
plot(Time, X[,2], ylab="S", ylim=c(0,sum(X[1,2:4])),
     type="l", lwd=2);   abline(h=0) # Add a horizontal axis
plot(Time, X[,3], ylab="I", ylim=c(0,sum(X[1,2:4])),
     type="l", lwd=2);   abline(h=0)
plot(Time, X[,4], ylab="R", ylim=c(0,sum(X[1,2:4])),
     type="l", lwd=2);   abline(h=0)
```

These simulation results in Fig. 8 illustrate how this stochastic model captures the random fluctuations induced by the discrete nature of individuals in the system, and the probabilistic nature of the transmission and recovery processes. We call this kind of stochasticity *demographic stochasticity* to distinguish it from *environmental stochasticity* which might, for example, take the form of a non-constant transmission coefficient β fluctuating in time. This may be modeled using a pre-specified curve as discussed in the previous section, or by approximating the combined demographic and environmental noise within a stochastic differential equation (SDE) framework as discussed in the next section.

Exercise 16 Rerun the code above multiple times, but omit the `set.seed()` call so that a different stochastic simulation is generated each time. Do all simulations look the same, or not? How do these compare to Figs. 2 and 3?

Exercise 17 Combine the above SSA code with the code for plotting the numerical solution curve to the ODE version of the SIR model, and overlay the ODE output with the stochastic model output. How do the two trajectories compare?

Challenge Problem 20 Repeat the stochastic simulation above a few hundred times, store each of the simulations, and plot them together in one graph. How do

they compare? Plot or otherwise summarize the distribution (across these different simulations) of any interesting quantities you can think of, e.g., peak number infected, total number infected, time to end of epidemic, etc.

Challenge Problem 21 Iterate these stochastic simulations many times and for each iteration store the first time at which $I(t) = 0$. Plot a histogram of these extinction times.

Challenge Problem 22 Implement the Stochastic Simulation Algorithm (SSA) for the Rosenzweig–MacArthur predator–prey model, Eqs. (23). Carefully consider the dependence or independence of different events (e.g., loss of prey via predation versus the birth of a new predator). As above, compare stochastic and ODE model output for the same parameter values and initial conditions. Increasing a through the Hopf bifurcation (see Fig. 1), noise may sometimes cause the predators or prey to go extinct, especially as the amplitude of the oscillations increases with increasing a. How does the average time to extinction vary with a above the threshold value that gives rise to predator–prey oscillations?

3.4 Stochastic Differential Equations (SDEs)

A common way to approximate a continuous-time discrete-state stochastic model is to derive a continuous-time, continuous-state model known as a *stochastic differential equation* (SDE) which retains the key average (mean) changes over time (like ODEs) as well as some of the covariance structure of the stochastic fluctuations about that average behavior. There are multiple ways to derive SDEs, but the derivation below follows Chapter 5 of [3].

First, consider the discrete-time approximation of the underlying continuous-time stochastic model for our SIR equation, and the possible update vector values corresponding to transmission and recovery events. These are summarized in Table 1 where as above we only focus on the S and I equations since R can be found later using $R(t) = N_0 - S(t) - I(t)$. Let the update vector $\Delta\mathbf{x}$ be a random vector that takes on these possible values according to their corresponding probabilities. The mean and second moment of the update vector $\Delta\mathbf{x}$ are, respectively,

$$E(\Delta\mathbf{x}) = \begin{bmatrix} -\beta S I \\ \beta S I - \gamma \end{bmatrix} \Delta t \tag{55}$$

$$E(\Delta\mathbf{x}(\Delta\mathbf{x})^{\mathsf{T}}) = \begin{bmatrix} \beta S I & -\beta S I \\ -\beta S I & \beta S I + \gamma I \end{bmatrix} \Delta t. \tag{56}$$

If we define $\boldsymbol{\mu} = E(\Delta\mathbf{x})/\Delta t$ and $\mathbf{B}$ such that $\mathbf{B}^2 = E(\Delta\mathbf{x}(\Delta\mathbf{x})^{\mathsf{T}})/\Delta t$ then the SDE for this scenario—using *drift coefficient* $\boldsymbol{\mu}$ and *diffusion coefficient* $\mathbf{B}$—is given by

$$d\mathbf{x} = \boldsymbol{\mu}\, dt + \mathbf{B}\, d\mathbf{W} \tag{57}$$

where $\mathbf{W}(t)$ is a vector of two independent Wiener processes (i.e., standard Brownian motion where for $t > s$ each $W_i(t) - W_i(s)$ is normally distributed with mean 0 and variance $t - s$). Since $\mathbf{W}(t)$ is not time-differentiable, the notation used in Eq. (57) implicitly assumes these quantities exist as the integrand of an integral over time [3, 6, 131]. Thus, an SDE approximation of our stochastic SIR model is

$$\begin{bmatrix} dS(t) \\ dI(t) \end{bmatrix} = \begin{bmatrix} -\lambda(t)\, S(t) \\ \lambda(t)\, S(t) - \gamma\, I(t) \end{bmatrix} dt + \mathbf{B}\, d\mathbf{W}. \tag{58}$$

Matrix $\mathbf{B}$ can be found analytically in this case, but in general may need to be computed numerically. Here, the covariance matrix $\mathbf{B}^2$ has multiple square roots, but only one is a symmetric positive definite matrix [159], thus we use

$$\mathbf{B} = \begin{bmatrix} \frac{\sqrt{BSI}\sqrt{gI}+BSI}{\sqrt{\left(\sqrt{gI}+\sqrt{BSI}\right)^2+BSI}} & -\frac{BSI}{\sqrt{\left(\sqrt{gI}+\sqrt{BSI}\right)^2+BSI}} \\ -\frac{BSI}{\sqrt{\left(\sqrt{gI}+\sqrt{BSI}\right)^2+BSI}} & \frac{gI+\sqrt{BSI}\sqrt{gI}+BSI}{\sqrt{\left(\sqrt{gI}+\sqrt{BSI}\right)^2+BSI}} \end{bmatrix}. \tag{59}$$

3.4.1 Numerical Solutions to SDEs

Numerical solutions to SDEs can be computed using algorithms similar to those discussed in the section above for simulating ODEs, e.g., see [3, 16, 76, 83, 138, 147]. Below, the stochastic fourth-order Runge–Kutta algorithm from [76] is used to numerically solve the above SDE version of the SIR model (Fig. 9). For additional details on deriving, analyzing, and generating numerical solutions of SDEs and related models (e.g., stochastic delay differential equations) see [3, 6, 93, 131].

```
#############################################################
# Stochastic Runge-Kutta-4 for Stratonovich calculus,
# as described in Hansen and Penland 2006.

# A function to calculate the deterministic part of our SDE,
dSI = function(X, ps) { # i.e., the drift coefficient mu.
   S = X[["S"]]      # Unpack our state variable values
   I = X[["I"]]
   b = ps[["beta"]]  # Unpack parameters
   g = ps[["gamma"]]
   return(matrix(c(-b*S*I, b*S*I - g*I), nrow=2, ncol=1))
}

sdeRK4 = function(X, dt, ps) {
   S = X[["S"]];       I = X[["I"]]; # As in dSI above.
   b = ps[["beta"]];  g = ps[["gamma"]];
```

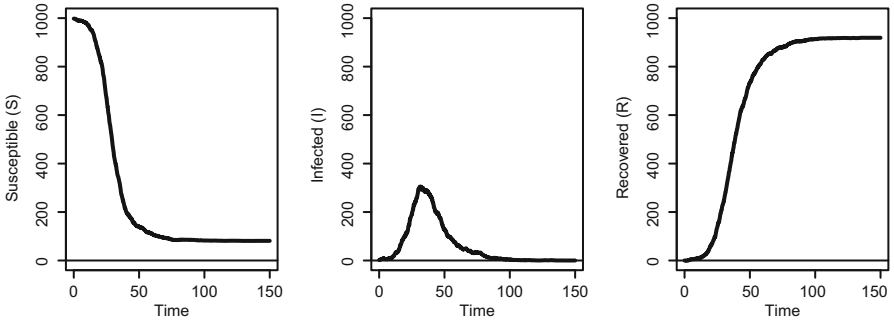

Fig. 9 One numerical solution to the SDE version of the simple SIR model defined by Eqs. (58) and (59), simulated using a stochastic fourth-order Runge–Kutta algorithm [76]. Compare to the ODE model trajectory in Fig. 2 and the Stochastic Simulation Algorithm (SSA) implementation of that model in Fig. 8. See the code in the main text for further details

```
   # Standard normal values with variance dt
   dW = matrix(rnorm(2,mean=0,sd=sqrt(dt)), nrow=2, ncol=1)

   # Square root of the second moment matrix B^2
   B = matrix(c(sqrt(b*S*I)*sqrt(g*I)+b*S*I, -b*S*I,
                -b*S*I, g*I + sqrt(b*S*I)*sqrt(g*I)+b*S*I)/
                sqrt((sqrt(g*I) + sqrt(b*S*I))^2+b*S*I),
                nrow=2, ncol=2)

   # Define the terms of the stochastic RK4 scheme
   # See Hansen and Penland 2006 for details.
   k1 = dSI(X,ps)*dt + B %*% dW ## %*% is matrix multiplication
   k2 = dSI(X + .5 * k1, ps)*dt + B %*% dW
   k3 = dSI(X + .5 * k2, ps)*dt + B %*% dW
   k4 = dSI(X + k3, ps)*dt      + B %*% dW
   return(X + (1 / 6) * (k1 + 2*k2 + 2*k3 + k4))
}

# Parameters for our numerical solution
Pars = c(beta=0.0003, gamma=0.1)
N=1000
X = data.frame(S=998, I=2) # Assume that R(0)=0
Time = c(0)
tstep = 0.1

# Iterate sdeRK4() steps to simulate a full trajectory
set.seed(246) # For repeatable random number generation
i=2 # counter for our while loop below
while(Time[i-1] < 150 & X[i-1,2]>0) { # while t<150, I>0
   Time[i] = Time[i-1] + tstep
   X[i,] = sdeRK4(X[i-1,],tstep,Pars)
   i = i+1
}
```

```
# Combine columns into a single data frame
Xout = cbind(Time, X, R = N - rowSums(X))
tail(Xout,4)
```

```
      Time        S           I        R
1417 141.6 81.05222  0.02383346 918.9239
1418 141.7 81.05465  0.04914261 918.8962
1419 141.8 81.06478  0.02691306 918.9083
1420 141.9 81.07035 -0.01539963 918.9450
```

```
# If I(t) dropped below zero on the last iteration...
if(Xout[i-1,3] < 0) { # ... set I=0 and extend to t=150:
        Xout[i-1,3]  = 0  # First set negative value to 0, then
        Xout[i,]     = Xout[i-1,] # duplicate that last row and
        Xout[i,1]    = 150        # set the final time to 150.
}

# Plot the numerical solution curves as before
par(mfrow=c(1,3)) # Plot 3 panels in 1 row
plot(Xout[,1], Xout[,2], ylab="Susceptible (S)", ylim=c(0,N),
     xlab="Time", type="l", lwd=2); abline(h=0)
plot(Xout[,1], Xout[,3], ylab="Infected (I)", ylim=c(0,N),
     xlab="Time", type="l", lwd=2); abline(h=0)
plot(Xout[,1], Xout[,4], ylab="Recovered (R)", ylim=c(0,N),
     xlab="Time", type="l", lwd=2); abline(h=0)
```

Challenge Problem 23 Derive and implement SSA- and SDE-based stochastic simulations of the logistic equation with explicit birth and death rates given by Eq. (52). How do their assumptions differ? What similarities and differences can you identify between these two stochastic implementations of the ODE-based logistic growth model?

3.5 *Distributed Delay Equations*

Another widespread ODE model assumption that can be altered is the assumed exponentially distributed *dwell times*[24] that are so often encountered as simplifying assumptions in mean field ODEs. Choosing other dwell time assumptions can sometimes alter model behavior in ways that are important for a given application (e.g., [63, 64]). Typically, models that relax this assumption must be derived from stochastic model first principles, and often the end result is a set of *integral equations* or *integro-differential equations* rather than ODEs (e.g., [80]). Under certain distributional assumptions, these integral equations can also be realized as a

[24]The *dwell time*, or *passage time*, is the duration of time an individual spends in a given state. As discussed in Sect. 2.1, ODEs with linear loss rate terms implicitly assume that—in the context of Sect. 3.3.1—the times individuals spend in that state are exponentially distributed.

system of ODEs (e.g., see [64, 65, 92, 154]), and in some cases it is possible to write down those ODEs directly from the stochastic model assumptions (see Sect. 3.5.2 and references therein). The following sections introduce these *distributed delay* equation models, as well as the linear chain trick (LCT) and a generalized linear chain trick (GLCT) for approximating or exactly representing these with systems of ODEs. *Discrete delay* differential equations (DDEs), which can be thought of as the zero-variance limit of a distributed delay equation, are also introduced in Sect. 3.5.4.

3.5.1 Integral and Integro-Differential Equations

Integral and integro-differential equations arise as a more general way of writing mean field models for continuous-time stochastic processes, especially when the dwell time distributions and related assumptions do not permit an ODE representation of the mean field model (e.g., when the model is not memoryless [92]). Specific forms, known as *Volterra integral equations* and *Volterra integro-differential equations* (or sometimes just referred to as *Volterra equations*) are named after the prolific mathematician and physicist Vito Volterra (born 3 May 1860, died 11 October 1940) who, building upon his work on functionals in functional analysis, laid the groundwork to use integral and integro-differential equations to study biological problems.

One approach to derive a mean field distributed delay model in the form of an integral equation is to first derive a discrete-time mean field model that accounts for all the past movements into and out of the different states. This can be done for an arbitrary dwell time distribution, as shown in the following example.

For an example of such a derivation, consider again our SIR model (see also [92]). Suppose we seek to model $I(t)$—the number of infected individuals at time t—with an integral equation. To begin, the time interval $[0, t]$ is partitioned into a large number of N sub-intervals, each of duration Δt. We can refer to each sub-interval by indexing the start time of each interval as t_i, i.e., the ith sub-interval is $(t_i,\ t_i + \Delta t]$. Assume the number of new infections during the ith time interval is binomially distributed where there are $n = S(t_i)$ individuals who might become infected, each with probability $p = \beta I(t_i)\Delta t$, during that time period of duration Δt. Thus, by properties of binomial distributions, the expected number of new infections during the ith time interval is $n\,p = \beta S(t_i)I(t_i)\Delta t$. For each of these cohorts entering the infectious state during the same time interval, assume all follow the same dwell time distribution with domain $[0, \infty)$, cumulative distribution function (CDF) $F(t)$, and *survival function* $G(t) \equiv 1 - F(t) = P(X > t)$. It then follows that, for the cohort of individuals who entered the infectious state during the ith sub-interval of time, the expected proportion that remains in that state at time t (after $t - t_i$ time units) is $G(t - t_i)$.

With those details in hand, a discrete-time mean field model of the number of infected individuals at time t is obtained by summing over all N of the sub-intervals of time (where $t = N\Delta t$), accounting for (a) the expected number entering in each

time interval, and (b) the proportion of each cohort[25] remaining in I at time t:

$$I(t) = \overbrace{I(0)\,G(t)}^{\text{initial cohort}} + \sum_{i=1}^{N} \overbrace{\beta S(t_i) I(t_i) \Delta t}^{\text{expected influx}} \underbrace{G(t - t_i)}_{\text{proportion remaining at time } t}. \tag{60}$$

Recognizing this as a Riemann sum, we can take the limit as $\Delta t \to 0$ ($N \to \infty$) and replace the times t_i with the integration variable s to yield the continuous-time mean field model given by the (Volterra) integral equation

$$I(t) = I(0)\,G(t) + \int_0^t \beta S(s) I(s) G(t - s)\, ds. \tag{61}$$

Similarly, one can go on to derive the full generalization of the SIR model, which incorporates general distributions for the time spent in the susceptible state, and the time spent in the infectious state (i.e., the *duration of infectiousness* also called the *infectious period*) which each have the respective CDFs $F_S(t)$ and $F_I(t)$ (i.e., survival functions $G_i(t) = 1 - F_i(t)$). Written as above, the equations are

$$S(t) = S(0) G_S(t) \tag{62a}$$

$$I(t) = I(0) G_I(t) + \int_0^t \beta\, I(s)\, S(s)\, G_I(t - s)\, ds \tag{62b}$$

$$R(t) = N - S(t) - I(t) \tag{62c}$$

where $N = S(0) + I(0) + R(0)$. The ODE model given by Eqs. (8) is a special case of the above model. This can be shown by assuming the time spent in the susceptible state obeys the 1st event time under a Poisson process with rate $\lambda(t) = \beta\, I(t)$, and thus $G_S(t) = \exp\big(-\int_0^t \beta\, I(u)\, du\big)$, and further assuming $G_I(t) = \exp\big(-\gamma\, t\big)$. However, in this more general framework one could consider various other choices for the dwell time distribution that describes the duration of infectiousness. For more details and similar models see [29, 63, 92, 102, 113].

One can undertake analyses (analogous to those mentioned above for ODE models) to investigate integral equation dynamics; however, even basic equilibrium stability analyses and computing numerical solutions requires more careful approaches than for ODEs [12, 25, 80, 111]. In practice, a combination of analytical and numerical methods can be used to study these types of distributed delay equations. However, another alternative is to take advantage of the fact that, for some choices of the dwell time distribution, differentiating these integral equations can yield equivalent systems of ODEs, as detailed in the next section.

[25] For simplicity, we here assume the initial cohort enters state I at $t = 0$ and thus follows the same dwell time distribution G.

3.5.2 Linear Chain Trick

The *linear chain trick* (LCT; also known as the *gamma chain trick*) allows us to derive a new ODE model from an existing ODE model that assumes an exponential dwell time distribution in one or more states by replacing that assumption with an Erlang[26] distributed dwell time. It also allows for the derivation of a system of ODEs that approximates a distributed delay equation, or a delay differential equation (as illustrated in Sect. 3.5.4). Erlang distributions are the special set of gamma distributions that can be thought of as the sum of k independent exponential distributions, each with the same rate [85, 106, 145]. Importantly, the LCT allows modelers to write down this new ODE model directly, without going through the process of constructing a more general integral equation model (as in the above example), and/or without explicitly deriving ODEs from an integral equation (as discussed below).

The LCT works as follows. We seek to replace a target state and corresponding state variable x in an ODE—which has an implicit exponential dwell time distribution with a mean dwell time of $1/r$ (or equivalently, rate r, i.e., the loss rate in the dx/dt equation is $-r\,x$)—with a series of substates x_i that partition state x into k substates. Individuals enter the state into the first substate x_1, then transition to the second substate, and so on, in series. This ensures that the overall passage time through these substates is Erlang distributed, since the sum of k exponentially distributed random variables with rate rk yields an Erlang distribution with mean $k/(rk) = 1/r$, as desired.[27]

The LCT is applied to specific model equations as follows. Suppose a state x in the model has an input rate $\Lambda(t)$ and an exponential dwell time with rate μ (i.e., mean $1/\mu$), and thus the mean field ODE equation is of the form

$$\frac{dx}{dt} = \Lambda(t) - \mu\, x(t). \tag{63}$$

To instead assume an Erlang distributed duration of time with the same mean $(1/\mu)$, the equation for $x(t)$ can be replaced with equations for k substates $x_i(t)$, where $x(t) = \sum x_i(t)$, and the equation above is replaced by

$$\frac{dx_1(t)}{dt} = \Lambda(t) - \mu k\, x_1(t) \tag{64a}$$

$$\frac{dx_j(t)}{dt} = \mu k\, x_{j-1}(t) - \mu k\, x_j(t), \quad \text{for } j = 2, \ldots, k. \tag{64b}$$

[26] Erlang distributions are gamma distributions with integer-valued shape parameters. Compared to exponential distributions, the Erlang density function is more hump shaped and the variance can be made arbitrarily small, as is sometimes desired in applications.

[27] Readers familiar with Poisson processes may recall that homogeneous Poisson processes have inter-event times that are exponentially distributed, and the time to the kth event under a homogeneous Poisson process with rate r is Erlang distributed with rate r and shape k.

Furthermore, one can choose a value of k in order to implement the desired variance (increasing k decreases the variance). Often, it is preferable to specify the desired *coefficient of variation*[28] instead, since the coefficient of variation (cv) for an Erlang distribution is determined solely by shape parameter k, where $cv = 1/\sqrt{k}$. Note cv can be specified to yield an integer-valued $k = 1/cv^2$.

Other model equations may also need to be modified when implementing the LCT, for example you may need to replace expressions involving state variable $x(t)$ with the sum of the new substate variables $x_i(t)$, and any input rates into other states that depend on the loss rate from x, formerly $\mu\, x(t)$, must instead depend on the new loss rate $\mu k\, x_k(t)$ (for an example, see below).

To illustrate, suppose that for our original SIR model Eqs. (8) we wanted to replace our exponentially distributed dwell time (rate γ; mean $1/\gamma$) in the infected state with an Erlang distribution with mean $1/\gamma$ and coefficient of variation of 1/3 (i.e., $k = 9$). To do this using the LCT, we simply replace the ODE

$$\frac{dI}{dt} = \beta\, S\, I - \gamma\, I \tag{65}$$

as indicated above, and modify the transmission and recovery rate terms to yield

$$\frac{dS}{dt} = -\beta \left(\sum_{i=1}^{9} I_i(t) \right) S(t) \tag{66a}$$

$$\frac{dI_1}{dt} = \beta \left(\sum_{i=1}^{9} I_i(t) \right) S(t) - \gamma k\, I_1(t) \tag{66b}$$

$$\frac{dI_2}{dt} = \gamma k\, I_1(t) - \gamma k\, I_2(t) \tag{66c}$$

$$\vdots$$

$$\frac{dI_9}{dt} = \gamma k\, I_8(t) - \gamma k\, I_9(t) \tag{66d}$$

$$\frac{dR}{dt} = \gamma k\, I_9(t). \tag{66e}$$

In practice, the LCT may be more challenging to apply if there are additional model complexities like transitions to multiple states or substate transitions that should not "reset the clock" for the overall dwell time distribution in a state. For guidance on applying the LCT in those situations, see [92].

[28] The *coefficient of variation* is the standard deviation divided by the mean.

Challenge Problem 24 As illustrated above, the LCT can quickly increase the number of state variables. Explore the consequences of this increase in dimensionality either in terms of model dynamics, and/or the accuracy of numerical solutions.

3.5.3 Mathematical Foundations of the Linear Chain Trick

For readers interested in the mathematical machinery behind the LCT, the following is a brief overview of how to explicitly derive ODEs from integral equations.

The Erlang density and survival functions are given (respectively) by

$$g_r^k(t) = r\,\frac{(r\,t)^{k-1}}{(k-1)!}e^{-rt} \tag{67a}$$

$$G_r^k(t) = \sum_{j=1}^{k}\frac{1}{r}\,g_r^j(t). \tag{67b}$$

The following recursion relationship between Erlang density functions and their derivatives is the linchpin of the LCT, as summarized in Theorem 4 (adapted from [92], which follows Eqs. 7.11 in [154]); see [92] for extensions).

Theorem 4 *The Erlang density functions $g_r^j(t)$, with rate r and shape j, satisfy*

$$\frac{d}{dt}g_r^1(t) = -\,r g_r^1(t), \quad \textit{where} \quad g_r^1(0) = r, \tag{68a}$$

$$\frac{d}{dt}g_r^j(t) = r\,[g_r^{j-1}(t) - g_r^j(t)], \quad \textit{where} \quad g_r^j(0) = 0 \ \textit{for} \ j \geq 2. \tag{68b}$$

This recursion relation allows one to differentiate integral equations like Eqs. (62) with assumed Erlang dwell times, and formally derive equivalent mean field ODEs like Eqs. (66). The linear chain trick can also be generalized to a much broader family of dwell time distributions that includes the *phase-type distributions* as detailed in [92]. More specifically, these phase-type distributions describe the distribution of times it takes to hit an absorbing state in a continuous-time Markov chain, and there is a straightforward formula for writing the ODEs for such cases based on the corresponding transition matrix and rate vector for the continuous-time Markov chain representation of the assumed phase-type distribution. Interested readers should consult [92, 154] and references therein for further details, as here we only introduce the standard linear chain trick.

Challenge Problem 25 Take the $I(t)$ equation from the Volterra integral form of the SIR model (Eqs. (62)) and assume an Erlang distribution survival function with rate r and shape k. Use the Leibniz rule for differentiating integrals to prove that the

above set of equations for I_i are correct. *Hint: After differentiating the integral and applying Theorem 4, use the substitution*

$$I_j(t) = I_0 \frac{1}{r} g_r^j(t) + \int_0^t \beta\, S(s)\, I(s) \frac{1}{r} g_r^j(t-s) ds. \tag{69}$$

3.5.4 Delay Differential Equations

Another approach to modifying the exponential delay assumption implicit in many ODE models is to consider taking the limit of the dwell time distribution's variance to zero while the mean is fixed at some value τ. This yields equations of the form[29]

$$\frac{d\mathbf{x}}{dt} = f(t, \mathbf{x}(t), \mathbf{x}(t-\tau), \theta) \tag{70}$$

where $\tau > 0$ thus allowing the derivative of the current state (at time t) to depend on the past state of the system τ time units into the past. These are known as (discrete) *delay differential equations* (DDEs) [154], and while DDEs are slightly more difficult to analyze than ODEs (e.g., in terms of equilibrium stability analysis), they are easier to mathematically analyze and to compute numerical solutions for, relative to integral and integro-differential equations. It is also worth noting that DDEs often exhibit more complex dynamics than similar (non-delayed) ODE models (e.g., see the simple example in the Appendix of [67]).

There are many resources that address the analysis of DDEs (e.g., [8, 18, 154, 155]), and applications of DDEs are common in the scientific literature, so the various methods of mathematical analysis will not be reviewed here. Numerical solutions can be obtained using the `dede` function in the `deSolve` package in R [156], which is used in a similar fashion to the `ode` function but with a few caveats. First, unlike ODEs (which are memoryless and only depend on the current state) DDEs depend on the history of the system. Thus, to begin a numerical solution and compute forward in time starting at time $t = 0$ requires that $x(t)$ be defined over the time interval $t \in [-\tau, 0]$. That is, unlike the vector of initial state values needed to simulate trajectories for ODEs, DDEs require *initial functions* over $[-\tau, 0]$ to obtain numerical solutions (for example code in R, see the bottom of the help documentation for `dede` which illustrates how to specify these initial function values).

DDEs with a finite number of state variables are actually, in a sense, infinite dimensional in that all of the values of $x(t)$ between time $t - \tau$ and t can be thought of as state variables. To provide some intuition for why this is the case (and thus provide a means of approximating DDEs with ODEs, or vice versa) consider the

[29] More generally, if there are multiple lag values $\tau_i, i = 1, \ldots, k$, the right hand side is of the form $f(t, \mathbf{x}(t), \mathbf{x}(t-\tau_1), \ldots, \mathbf{x}(t-\tau_k), \theta)$.

following construction of the SIR model with a discrete delay based on Eqs. (66), the SIR model with an Erlang distributed duration of infection. Recall that, for a fixed mean duration of infectiousness $1/\gamma$, the shape parameter k represents the number of substates I_i necessary to include in the ODE when assuming an Erlang distributed delay between acquiring an infection, and recovery. Recalling that the coefficient of variation (and the variance) decreases as k increases, it follows that taking $k \to \infty$ should yield a model that has a discrete delay of exactly $\tau = 1/\gamma$. For all k, $\sum I_i(t) = I(t) = N_0 - S(t) - R(t)$, thus in the limit as $k \to \infty$ the number entering the recovered state during an infinitesimally small period of time approaches exactly the same number who became infected $\tau = 1/\gamma$ time units ago (i.e., the rate of recovery at t is $\beta\, I(t-\tau)\, S(t-\tau)$), one could write a DDE version of this model as

$$\frac{dS(t)}{dt} = -\beta\,(N_0 - S(t) - R(t))\,S(t) \tag{71a}$$

$$\frac{dR(t)}{dt} = \beta\,(N_0 - S(t-\tau) - R(t-\tau))\,S(t-\tau). \tag{71b}$$

Exercise 18 Differentiate $I(t) = N_0 - S(t) - R(t)$ and use Eqs. (71) to show that the full version of the model above can be written as

$$\frac{dS(t)}{dt} = -\beta\, I(t)\, S(t) \tag{72a}$$

$$\frac{dI(t)}{dt} = \beta\, I(t)\, S(t) - \beta\, I(t-\tau)\, S(t-\tau) \tag{72b}$$

$$\frac{dR(t)}{dt} = \beta\, I(t-\tau)\, S(t-\tau). \tag{72c}$$

Exercise 19 Consider repeating the above derivation of a DDE for the basic SEIR model, where we would like to change the assumption of an exponentially distributed latent period (with mean $\tau = 1/\nu$) to a discrete delay with the same mean, i.e., we would like to assume that the incubation period before an exposed individual becomes infectious is exactly τ units of time. Justify whether or not this model should be of the form

$$\frac{dS(t)}{dt} = -\beta\, I(t)\, S(t) \tag{73a}$$

$$\frac{dI(t)}{dt} = \beta\, I(t-\tau)\, S(t-\tau) - \gamma\, I(t) \tag{73b}$$

$$\frac{dR(t)}{dt} = \gamma\, I(t) \tag{73c}$$

by first generalizing the SEIR model using the linear chain trick, and then letting the variance of the latent period go to zero (i.e., letting $k \to 0$).

Lastly, the above procedure can also be reversed as a way of constructing a system of ODEs that approximates a DDE by relaxing the "zero-variance" assumption to instead assume an Erlang distributed delay with mean τ. This well-known approximation method is described at the end of Chapter 7 in [154].

For example, consider the following DDE version of the Rosenzweig–MacArthur model, where we assume that there is some period of time τ between predators consuming prey and the subsequent arrival of new predators in the system [167].

$$\frac{dN(t)}{dt} = r\,N(t)\left(1 - \frac{N(t)}{K}\right) - \frac{a\,P(t)}{k + N(t)}N(t) \tag{74a}$$

$$\frac{dP(t)}{dt} = \chi\frac{a\,N(t-\tau)}{k + N(t-\tau)}P(t-\tau) - \mu\,P(t) \tag{74b}$$

This may be reasonable if, for example, food resources are quickly turned into offspring and their overall maturation time has a small coefficient of variation (compare Eq. (74) to the model in §6.2 of [118]). Here we may treat the delayed rate of new predators entering the system (i.e., the *recruitment rate*)

$$\chi\frac{a\,N(t-\tau)}{k + N(t-\tau)}P(t-\tau) \tag{75}$$

as the input rate into this previously unwritten count of offspring that will eventually become adults. Approximating with the LCT introduces a series of intermediate state variables Y_i, $i = 1, \ldots, k$ reminiscent of the linear chain trick results above, where we assume the value of k such that it gives the desired coefficient of variation (recall $k = 1/c_v^2$) for an assumed Erlang distribution of incubation times. Relaxing the discrete delay assumption to include some non-zero variance in the incubation period therefore yields the following ODE model.

$$\frac{dN(t)}{dt} = r\,N(t)\left(1 - \frac{N(t)}{K}\right) - \frac{a\,P(t)}{k + N(t)}N(t) \tag{76a}$$

$$\frac{dY_1(t)}{dt} = \chi\frac{a\,N(t)}{k + N(t)}P(t) - \frac{k}{\tau}\,Y_1(t) \tag{76b}$$

$$\frac{dY_j(t)}{dt} = \frac{k}{\tau}\,Y_{j-1}(t) - \frac{k}{\tau}\,Y_j(t), \quad \text{for } j = 2, \ldots, k \tag{76c}$$

$$\frac{dP(t)}{dt} = \frac{k}{\tau}\,Y_k(t) - \mu\,P(t) \tag{76d}$$

It should be noted that these finite-state approximations of DDE models, especially for large k (i.e., small coefficient of variation), can have different dynamics relative to the original DDE, and they can also be numerically problematic when computing numerical solutions for large k values.

Challenge Problem 26 Use the `dede` and `ode` functions in `deSolve` to generate comparable numerical solutions to the two SIR models (and/or the two predator–prey models) above, and compare those solution curves for various values of k (i.e., for various values of the implicitly assumed variance about the mean delay value τ). Think carefully about the initial function values for the DDE model solutions (e.g., consider using ODE model solution curves for these values) so that the simulated results can be fairly compared.

3.6 Individual Heterogeneity

Different approaches exist for relaxing the ODE model assumption of homogeneity among individuals in a given state. This may be a reasonable assumption for the individual molecules in a solution of chemical reactants, but may be an oversimplification for a population of plants or animals. The most straightforward way to relax this assumption is to simply assume a finite number of distinct types of individuals. For example, this has been done with models of sexually transmitted infections to account for low- and high-risk groups [22, 81], and in multispecies interaction models where prey can occur in forms that are either undefended (a good food resource) or well-defended (a poor food resource) [36, 49, 168, 169].

Methods for modeling a continuum of individual heterogeneity also exist, e.g., to allow something like individual size to vary along a continuum as opposed to a finite set of possible size classes. For example, in a discrete-time case, this scenario can be modeled using *integral projection models* [53, 54, 120], which can offer benefits over discrete-state models when used for parameter estimation since smooth kernels with only a few parameters may have fewer parameters to estimate than a model with a larger number of discrete states, each with their own related parameters. In continuous-time, a PDE model can be used to describe the time-evolution of a continuous trait distribution across a population, and there are a variety of analytical and computational resources available for such models, e.g., [45, 52, 100, 127].

Another approach to constructing a model that incorporates individual heterogeneity is to implement an *agent based* (also known as *individual based*) *model* [72]. These computational models allow for the inclusion of more complex rules governing individual behavior than can typically be incorporated into a tractable mathematical model. These approaches have been used quite successfully to better understand phenomena like animal movements in groups or through heterogeneous environments (e.g., [50, 66, 72, 144]). While these typically do not allow much mathematical analysis, they can be useful for modeling specific systems that are data rich or are otherwise well understood, and can form the basis for subsequently deriving mathematical models that aim to address very specific phenomena observed in simulations.

3.7 Spatially Explicit Models

Spatial effects are perhaps one of the most important components of real-world systems that frequently get oversimplified by using ODE models, and some of the other types of models discussed above. ODE models typically assume that systems are well mixed and not constrained by spatial distances. However, incorporating these spatial dependencies can have a significant impact on how these models behave. There is a sizable body of literature on spatial models and methods for analyzing various types of spatially explicit models. Here we just scratch the surface of this topic and provide only a very brief introduction to a few of these modeling approaches, and leave it to the reader to seek out a more in depth treatment of spatially explicit models.

Readers in search of a research project that includes spatial effects are encouraged to derive a spatially explicit model (like one of the model types listed below) from an existing ODE model, or to slightly modify the assumptions of an existing spatially explicit model, for example a PDE model. One might also consider the more ambitious option of deriving a multipatch or network model based on an existing spatial (e.g., PDE) model, or vice versa.

One approach to deriving a spatially explicit model from an existing non-spatial model is to simply consider dividing the system up across multiple "patches." A well-known example of such models are the *metapopulation models* in population ecology [77, 109, 110, 132]. In these contexts, the dynamics within each patch can be governed by a simple model, such as an ODE model, and some explicit rules can be imposed that govern movement or other interactions between patches. Constructing these sorts of *multipatch models* is an easy way to explore how spatially dependent processes affect dynamics. These models can also be used to explore the effects of spatial heterogeneity by allowing model parameters to vary from patch to patch. In recent decades, such models have been revisited in the context of the burgeoning field of network science [130], allowing investigators to bring the tools and insights from the study of networks to bear on understanding how the patterns of patch connectedness (i.e., the network properties) shape system-wide dynamics.

Many examples of multipatch and network models exist in the literature, and some of the standard methods of ODE model analysis can be applied to these models. However, these methods must sometimes be adapted for the high dimensionality of these systems (e.g., see [30] and references therein; also compare methods in the literature for calculating epidemic thresholds on networks with the next generation operator approach to calculating the basic reproduction number $\mathscr{R}_0$ for small to medium sized model, as detailed in [43]). For an introduction to methods for network models, see [130, 135, 136] and also [58–62, 152].

Another approach to explicitly incorporating space into a dynamical systems model is to discretize a continuous 1-, 2-, or 3-dimensional spatial domain into a finite number of contiguous patches. This special case of the previously described patch models can then be analyzed using approximation methods such as *pair approximation* [48, 82, 110], or by taking the limit as the patch size shrinks to zero

to derive a PDE model. As mentioned above, there are other resources available that cover methods for deriving, analyzing, and numerically simulating PDE models (including the methods available in the `deSolve` package in R). Interested readers may find more detailed treatments of these topics in [45, 50, 100, 127] and related publications (also, see the list of "books in related areas" in the preface of [45]).

4 Choosing a Research Project

I hope the preceding sections have inspired you to reevaluate a familiar ODE model, or to seek out an ODE model related to a topic of particular interest to you, and take a critical look at the explicit and implicit assumptions of that model. The sections above provide an overview of some commonly used methods for analyzing ODE models (or otherwise using them in a research setting), and an overview of some ways that ODE model assumptions can be altered to derive new models in pursuit of new avenues of research.

From the perspective of conducting research to advance our collective knowledge of how the world works, research projects that parallel an existing ODE model analysis (using a model derived by meaningfully altering the assumptions of that original model) can help incrementally build upon an existing body of work. These types of research projects can be great for student researchers, since they provide some guidance in terms of which questions are worth asking, and which types of analyses can be used to address those questions. Additionally, these alternative models can also provide opportunities to ask new questions about important real-world systems that may not be as practical (or possible) to answer using an existing ODE model (e.g., the distribution of extinction times is more appropriately addressed in a discrete-state stochastic model). For these reasons, the relationships between different models and their differing assumptions are a wonderful place to probe and challenge our understanding of the real-world systems we seek to understand through mathematical modeling.

4.1 Additional Project Topics

Scientifically, many open problems exist in the study of specific systems that are being investigated using ODE models. As stated in the introduction, studying multiple mathematical models of the same system, each with slight variations in assumptions, can yield a more clear understanding of which results are robust across models (e.g., see [1, 2, 99, 107]) and across similar real-world systems. These studies can also identify which results might be meaningfully influenced by simplifying assumptions. Below are a few additional project ideas to consider that have some potential to lead to yield meaningful contributions to the very broad field of mathematical biology.

Research Project 1 (Timing in Infectious Disease Models) Recently, increased attention has been given to the importance of properly modeling the presence or absence of latent periods in infectious disease models [164], and the importance of making proper assumptions about the distributions that describe the duration of time spent in different model states [64, 65]. Many models exist in the literature for various diseases in wildlife and humans (and for other types of multispecies interactions that do not involve infectious parasites), and improving a model in this manner can make a great project.

One approach to tackling this project is to do the following:

1. Search the literature for relatively recent publication using an SIR-type ODE model to study a specific infectious disease. Diseases in non-human hosts or diseases that are otherwise not heavily researched (e.g., not HIV or Malaria, Dengue, or Rabies) are perhaps more likely to be modeled using relatively simple or otherwise poorly refined models.
2. Identify key model assumptions and the research aims addressed in the paper, paying careful attention to timing and the presence or absence of latent periods. Which assumptions would be the most important to reconsider?
3. Identify a question you plan to answer, and deriving a new model based on modifying just one (or maybe two) relevant assumptions, e.g., adding a latent period.
4. If possible, consult with a public health or infectious disease expert who has some expert knowledge of the system(s) being modeled.
5. Focusing on one question asked in the original paper, recreate the published analyses to verify those results.
6. Conduct an analogous analysis of your new model to address the same question under your revised assumptions, and/or conduct additional analyses to address other important questions.

From a more mathematical or methodological perspective, there are also some open questions related to the analytical and computational tools that exist for the various model frameworks mentioned in the sections above. While these can be more challenging problems to tackle, and may require a preparation beyond the typical undergraduate curriculum, some of these questions are still accessible to undergraduate researchers and could make interesting and impactful research projects.

Research Project 2 (Numerical Solutions for Distributed Delay Equations) Methods to compute numerical solutions to integral and integro-differential equations (often called *Volterra* integral and integro-differential

(continued)

Research Project 2 (continued)
equations) are currently being developed and refined. Review the literature, and select two or more methods and compare their performance under different modeling scenarios (e.g., periodically forced systems or systems with multiple time scales). To provide a reference point for evaluating the accuracy of numerical solutions, the example models used are typically simple enough to have analytical solutions. A nice spin on this project would be to use distributional assumptions that allow integral or integro-differential equations to be fully reduced to an equivalent system of ODEs according to the linear chain trick (LCT, see Sect. 3.5.2) or the generalized linear chain trick (GLCT) [92]. How well do numerical solutions to those ODEs perform in these comparisons?

A project like this could be conducted as follows:

1. Identify some numerical methods for one or more types of distributed delay equations (e.g., just integral equations).
2. Do a literature search to find appropriate models to use as examples. Include some with analytical solutions, and some that could be written as ODEs via the LCT and therefore simulated using appropriate ODE solvers.
3. Establish a set of criteria by which to compare the different methods (e.g., computing time per numerical solution, implementation time or some other measure of the effort required leading up to computing numerical solutions, the accuracy relative to some baseline solution such as an analytical solution, and so forth).
4. Implement the planned comparisons.

Students interested in exploring statistical projects may choose to focus on ODE-based models or projects that involve other modeling frameworks.

As discussed in Sect. 2.7, investigating statistical properties of dynamic models can be done using simulated data so that the known "true" parameter values can be used to evaluating the parameter estimation procedure. Therefore one can take almost any dynamic model and assess the statistical properties of that model under certain observation process assumptions or using different estimation techniques. This is especially meaningful when done for models that were used in evaluating real data where it is unclear that models have identifiable parameters, and/or where additional analyses (e.g., a power analysis) might provide important context for interpreting the analysis of real data.

Research Project 3 (Estimator Properties) Investigate the statistical properties of a system-specific ODE model that has previously been (or might someday be) fit to empirical data.

This project could be approached as follows:

1. First, as above, find a peer-reviewed publication that includes an ODE model that was fit to data. Focus on searching in journals in application areas where such analyses are unlikely to be included in the publication.
2. Conduct a structural and/or practical identifiability analysis of the model.
3. Conduct a simulation-based study to characterize what if any bias exists for the estimation procedure used in the publication.
4. Assess what (if any) *uncertainty quantification* was done, and consider further analyses if warranted.

Research Project 4 (Fitting Models with Delays to Data) It is not well understood whether (or under which conditions) distributed delay equations can be fit to data from comparable discrete delay equations (and vice versa) and yield correct parameter inferences or predictions about future states of the system. While deriving general statements about fitting one model type to another may be challenging, these questions can be addressed in the context specific cases, perhaps laying the groundwork for identifying more general patterns.

For this project, consider doing the following:

1. Find a published distributed or discrete delay model, and derive the complementary model so that you have a discrete delay model paired with an analogous distributed delay model.
2. Use the linear chain trick to write a set of ODEs for the distributed delay model.
3. Design a simulation-based experimental comparison, thinking carefully about the differences in assumptions between models, then simulate data under each model. Fit each model to each data set.
4. How biased are the estimates in each case? Which quantities can or cannot be estimated well (e.g., other parameter values in the model unrelated to the form of the delays), and/or how well does each model forecast a continuation of those simulated data sets?

5 The Importance of Publishing

Finally, perhaps the single most important part of conducting research is to share that research with your colleagues, the broader scientific community, and the public. This might include preparing and presenting a poster at a local poster session or professional conference, or preparing a final manuscript for submission to a peer-reviewed journal, so that others may benefit from your work.

Once a project is underway, work with your research mentor to begin drafting your manuscript in LaTeX using templates and guidelines provided by journals for authors. Write up your results as if you were planning to submit the manuscript for publication. To select a journal, start with the one that published the paper you are basing your work on, and talk to your mentor about which journal might be most appropriate for you to submit your work to.

In addition to standard peer-reviewed journals, there are also a number of options available for publishing undergraduate research through the peer-review process. Examples include journals like *Spora: A Journal of Biomathematics*, *SIAM Undergraduate Research Online (SIURO)*, and regional or university-related journals like *The Minnesota Journal of Undergraduate Mathematics*, *The Journal of Undergraduate Research at Ohio State*, or the *Nevada State Undergraduate Research Journal*.

Acknowledgements My outlook on undergraduate research at the interface of biology, mathematics, and statistics has been shaped by many influential mentors and peers that I was lucky to have known as an undergraduate at the University of Southern Colorado (now Colorado State University-Pueblo), at the Mathematical and Theoretical Biology Summer Institute (MTBI), as a graduate student in the Center for Applied Mathematics at Cornell University, and as a postdoctoral researcher in the Aquatic Ecology Laboratory and the Mathematical Biosciences Institute at The Ohio State University. It has truly been a privilege to know and learn from such an outstanding collection of people. I thank the students in my courses for exposing me to a broad array of project topics that have further influenced my outlook on undergraduate research across the sciences. I thank my colleague and wife Dr. Deena Schmidt; my colleagues Michael Cortez, Marisa Eisenberg, Colin Grudzien, and Andrey Sarantsev; my students Amy Robards, Jace Gilbert, Adam Kirosingh, Narae Wadsworth, and Catalina Medina; and reviewers Andrew Brouwer and Kathryn Montovan for many helpful comments, criticisms, and suggestions that ultimately improved this chapter. Finally, I thank my children Alex (age 7), Ellie (age 3), and Nat (age 3) for their unyielding companionship and boundless energy, which were instrumental to my preparing this chapter over such an extended period of time.

Appendix

Getting Started Writing in LaTeX and Programming in R

Installing the free software LaTeX and R should be straightforward, but here are some installation tips for Microsoft Windows and Mac OS X users (Linux users can find similar installation instructions using the resources mentioned below). Readers are encouraged to ask other students and faculty at their institution about additional resources.

There are two pieces of software each, for LaTeX and R, that should be installed: the basic LaTeX and R software, and an enhanced user interface that facilitates learning for new users and helps established users with their day-to-

day workflow (e.g., helpful menus, code autocompletion and highlighting, custom keyboard shortcuts, advanced document preparation capabilities, etc.). Educators will appreciate that both TeXstudio and R Studio have a consistent user interface across operating systems, making them ideal for group or classroom environments where students may be running a mix of operating systems.

If installing both LaTeX and R (recommended), install the base software first (in either order) before installing TeXstudio and/or R Studio (in either order). More detailed instructions and resources are provided below.

Installing and Using LaTeX

There are different implementations of LaTeX available: *MiKTeX* is a popular Microsoft Windows option (http://miktex.org/), and *TeX Live* a popular Mac OS X option. *TeX Live* comes as part of a full 2 gigabyte installation called *MacTeX* (www.tug.org/mactex/; which includes the popular editors TeXstudio and TeXShop) or can be installed through a smaller 110 megabyte bundle *BasicTeX* (www.tug.org/mactex/morepackages.html). Configure LaTeX to install packages "on the fly" without prompting you for permission. This can be done during (preferred) or after installation. Also download and install Ghostscript (www.tug.org/mactex/morepackages.html).

Next, install the TeXstudio editor (www.texstudio.org/), preferably after R is installed. Various settings can be changed after installation, including color themes, and configuring TeXstudio to compile a type of LaTeX document that includes blocks of R code known as a `Sweave` or `knitr` document (use `knitr`).

For additional LaTeX resources, see the author's website (www.pauljhurtado.com/R/), the LaTeX wikibook [166], the AMS Short Math Guide for LaTeX [42], and references and resources listed therein.

Installing and Using R

Download R from www.r-project.org/ and use the default installation process. Once R is installed (and, preferably once LaTeX is installed), install R Studio from www.rstudio.com. By installing R Studio after LaTeX, you will be able to create multiple document types to generate PDFs, including *R Markdown* documents and `knitr` documents. Helpful online resources include the "cheat sheets" on the R Studio website, introductory courses by DataCamp (www.datacamp.com) and Software Carpentry (www.software-carpentry.org), [157] for a gentle introduction to R and some applications to population modeling, and R resources on the author's website (www.pauljhurtado.com/R/).

References

1. Adamson, M.W., Morozov, A.Y.: When can we trust our model predictions? Unearthing structural sensitivity in biological systems. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **469**(2149) (2012). https://doi.org/10.1098/rspa.2012.0500
2. Adamson, M.W., Morozov, A.Y.: Defining and detecting structural sensitivity in biological models: Developing a new framework. Journal of Mathematical Biology **69**(6–7), 1815–1848 (2014). https://doi.org/10.1007/s00285-014-0753-3
3. Allen, E.: Modeling with Itô Stochastic Differential Equations, *Mathematical Modelling: Theory and Applications*, vol. 22. Springer Netherlands (2007). https://doi.org/10.1007/978-1-4020-5953-7
4. Allen, L.: An Introduction to Mathematical Biology. Pearson/Prentice Hall (2007)
5. Allen, L.J.S.: Mathematical Epidemiology, *Lecture Notes in Mathematics*, vol. 1945, chap. An Introduction to Stochastic Epidemic Models, pp. 81–130. Springer Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78911-6_3
6. Allen, L.J.S.: An Introduction to Stochastic Processes with Applications to Biology. Chapman and Hall/CRC (2010)
7. Anderson, R.M., May, R.M.: Infectious Diseases of Humans: Dynamics and Control. Oxford University Press (1992)
8. Arino, O., Hbid, M., Dads, E.A. (eds.): Delay Differential Equations and Applications, *NATO Science Series*, vol. 205. Springer (2006). https://doi.org/10.1007/1-4020-3647-7
9. Armbruster, B., Beck, E.: Elementary proof of convergence to the mean-field model for the SIR process. Journal of Mathematical Biology **75**(2), 327–339 (2017). https://doi.org/10.1007/s00285-016-1086-1
10. Arrowsmith, D.K., Place, C.M.: An Introduction to Dynamical Systems. Cambridge University Press (1990)
11. Audoly, S., Bellu, G., D'Angio, L., Saccomani, M., Cobelli, C.: Global identifiability of nonlinear models of biological systems. IEEE Transactions on Biomedical Engineering **48**(1), 55–65 (2001). https://doi.org/10.1109/10.900248
12. Baker, C.T.: A perspective on the numerical treatment of Volterra equations. Journal of Computational and Applied Mathematics **125**(1–2), 217–249 (2000). https://doi.org/10.1016/S0377-0427(00)00470-2
13. Banks, H.T., Catenacci, J., Hu, S.: A Comparison of Stochastic Systems with Different Types of Delays. Stochastic Analysis and Applications **31**(6), 913–955 (2013). https://doi.org/10.1080/07362994.2013.806217
14. Banks, H.T., Cintrón-Arias, A., Kappel, F.: Mathematical Modeling and Validation in Physiology: Applications to the Cardiovascular and Respiratory Systems, chap. Parameter Selection Methods in Inverse Problem Formulation, pp. 43–73. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-32882-4_3
15. Barrientos, P.G., Rodríguez, J.Á., Ruiz-Herrera, A.: Chaotic dynamics in the seasonally forced SIR epidemic model. Journal of Mathematical Biology **75**(6–7), 1655–1668 (2017). https://doi.org/10.1007/s00285-017-1130-9
16. Bayram, M., Partal, T., Buyukoz, G.O.: Numerical methods for simulation of stochastic differential equations. Advances in Difference Equations **2018**(1) (2018). https://doi.org/10.1186/s13662-018-1466-5
17. Bertram, R., Rubin, J.E.: Multi-timescale systems and fast-slow analysis. Mathematical Biosciences **287**, 105–121 (2017). https://doi.org/10.1016/j.mbs.2016.07.003. 50th Anniversary Issue
18. Beuter, A., Glass, L., Mackey, M.C., Titcombe, M.S. (eds.): Nonlinear Dynamics in Physiology and Medicine. Interdisciplinary Applied Mathematics (Book 25). Springer (2003)
19. Boersch-Supan, P.H., Ryan, S.J., Johnson, L.R.: deBInfer: Bayesian inference for dynamical models of biological systems in R. Methods in Ecology and Evolution **8**(4), 511–518 (2016). https://doi.org/10.1111/2041-210X.12679

20. Bolker, B.M.: Ecological Models and Data in R, chap. Dynamic Models (Ch. 11). Princeton University Press (2008). https://ms.mcmaster.ca/~bolker/emdbook/chap11A.pdf
21. Bolker, B.M.: Ecological Models and Data in R. Princeton University Press (2008)
22. Brauer, F., Castillo-Chavez, C.: Mathematical Models in Population Biology and Epidemiology, 2nd (2012) edn. Texts in Applied Mathematics (Book 40). Springer-Verlag (2011)
23. Brauer, F., van den Driessche, P., Wu, J. (eds.): Mathematical Epidemiology. Lecture Notes in Mathematics: Mathematical Biosciences Subseries. Springer-Verlag Berlin Heidelberg (2008). https://doi.org/10.1007/978-3-540-78911-6
24. Briggs, G.E., Haldane, J.B.S.: A note on the kinetics of enzyme action. Biochemical Journal **19**(2), 338–339 (1925). https://doi.org/10.1042/bj0190338
25. Burton, T., Furumochi, T.: A stability theory for integral equations. Journal of Integral Equations and Applications **6**(4), 445–477 (1994). https://doi.org/10.1216/jiea/1181075832
26. Cao, Y., Gillespie, D.T., Petzold, L.R.: Avoiding negative populations in explicit Poisson tau-leaping. The Journal of Chemical Physics **123**(5), 054,104 (2005). https://doi.org/10.1063/1.1992473
27. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. The Journal of Chemical Physics **124**(4), 044,109 (2006). https://doi.org/10.1063/1.2159468
28. Casella, G., Berger, R.: Statistical Inference, 2nd edn. Cengage Learning (2001)
29. Champredon, D., Dushoff, J., Earn, D.: Equivalence of the Erlang-Distributed SEIR Epidemic Model and the Renewal Equation. SIAM Journal on Applied Mathematics **78**(6), 3258–3278 (2018). https://doi.org/10.1137/18M1186411
30. Chapman, A., Mesbahi, M.: Stability analysis of nonlinear networks via M-matrix theory: Beyond linear consensus. In: 2012 American Control Conference (ACC). IEEE (2012). https://doi.org/10.1109/ACC.2012.6315625
31. Chatterjee, A., Vlachos, D.G., Katsoulakis, M.A.: Binomial distribution based τ-leap accelerated stochastic simulation. The Journal of Chemical Physics **122**(2), 024,112 (2005). https://doi.org/10.1063/1.1833357
32. Cintrón-Arias, A., Banks, H.T., Capaldi, A., Lloyd, A.L.: A sensitivity matrix based methodology for inverse problem formulation. Journal of Inverse and Ill-posed Problems pp. 545–564 (2009). https://doi.org/10.1515/JIIP.2009.034
33. Cobelli, C., DiStefano, J.J.: Parameter and structural identifiability concepts and ambiguities: A critical review and analysis. American Journal of Physiology-Regulatory, Integrative and Comparative Physiology **239**(1), R7–R24 (1980). https://doi.org/10.1152/ajpregu.1980.239.1.R7
34. Conlan, A.J., Grenfell, B.T.: Seasonality and the persistence and invasion of measles. Proceedings of the Royal Society B: Biological Sciences **274**(1614), 1133–1141 (2007). https://doi.org/10.1098/rspb.2006.0030
35. Cortez, M.H.: When does pathogen evolution maximize the basic reproductive number in well-mixed host–pathogen systems? Journal of Mathematical Biology **67**(6), 1533–1585 (2013). https://doi.org/10.1007/s00285-012-0601-2
36. Cortez, M.H.: Coevolution-driven predator-prey cycles: Predicting the characteristics of eco-coevolutionary cycles using fast-slow dynamical systems theory. Theoretical Ecology **8**(3), 369–382 (2015). https://doi.org/10.1007/s12080-015-0256-x
37. Cortez, M.H., Ellner, S.P.: Understanding rapid evolution in predator-prey interactions using the theory of fast-slow dynamical systems. The American Naturalist **176**(5), E109–E127 (2010). https://doi.org/10.1086/656485
38. Dawes, J., Souza, M.: A derivation of Hollings type I, II and III functional responses in predator–prey systems. Journal of Theoretical Biology **327**, 11–22 (2013). https://doi.org/10.1016/j.jtbi.2013.02.017
39. Dayan, P., Abbott, L.F.: Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. Computational Neuroscience. The MIT Press (2005)
40. Devroye, L.: Non-Uniform Random Variate Generation, chap. General Principles in Random Variate Generation: Inversion Method (§2.2). Springer-Verlag. (1986). http://luc.devroye.org/rnbookindex.html

41. Dhooge, A., Govaerts, W., Kuznetsov, Y.A.: MATCONT: A MATLAB package for numerical bifurcation analysis of ODEs. ACM Transactions on Mathematical Software **29**(2), 141–164 (2003). https://doi.org/10.1145/779359.779362
42. Downes, M., Beeton, B.: Short Math Guide for LaTeX. American Mathematical Society (2017). https://www.ams.org/tex/amslatex. (Accessed: 22 April 2019)
43. van den Driessche, P., Watmough, J.: Reproduction numbers and sub-threshold endemic equilibria for compartmental models of disease transmission. Mathematical Biosciences **180**(1–2), 29–48 (2002). https://doi.org/10.1016/S0025-5564(02)00108-6
44. Earn, D.J.: A simple model for complex dynamical transitions in epidemics. Science **287**(5453), 667–670 (2000). https://doi.org/10.1126/science.287.5453.667
45. Edelstein-Keshet, L.: Mathematical Models in Biology. Classics in Applied Mathematics (Book 46). Society for Industrial and Applied Mathematics (2005). https://doi.org/10.1137/1.9780898719147
46. Eisenberg, M.: Generalizing the differential algebra approach to input–output equations in structural identifiability. ArXiv e-prints (2013). https://arxiv.org/abs/1302.5484
47. Eisenberg, M.C., Hayashi, M.A.: Determining identifiable parameter combinations using subset profiling. Mathematical Biosciences **256**, 116–126 (2014). https://doi.org/10.1016/j.mbs.2014.08.008
48. Ellner, S.P.: Pair approximation for lattice models with multiple interaction scales. Journal of Theoretical Biology **210**(4), 435–447 (2001). https://doi.org/10.1006/jtbi.2001.2322
49. Ellner, S.P., Becks, L.: Rapid prey evolution and the dynamics of two-predator food webs. Theoretical Ecology **4**(2), 133–152 (2011). https://doi.org/10.1007/s12080-010-0096-7
50. Ellner, S.P., Guckenheimer, J.: Dynamic Models in Biology. Princeton University Press (2006)
51. Ellner, S.P., Guckenheimer, J.: Dynamic Models in Biology, chap. Building Dynamic Models (Ch. 9). Princeton University Press (2006). http://assets.press.princeton.edu/chapters/s9_8124.pdf
52. Ellner, S.P., Guckenheimer, J.: Dynamic Models in Biology, chap. Spatial Patterns in Biology (Ch. 7). Princeton University Press (2006). http://assets.press.princeton.edu/chapters/s7_8124.pdf
53. Ellner, S.P., Rees, M.: Integral projection models for species with complex demography. The American Naturalist **167**(3), 410–428 (2006). https://doi.org/10.1086/499438
54. Ellner, S.P., Rees, M.: Stochastic stable population growth in integral projection models: theory and application. Journal of Mathematical Biology **54**(2), 227–256 (2006). https://doi.org/10.1007/s00285-006-0044-8
55. Ermentrout, B.: XPP/XPPAUT Homepage. http://www.math.pitt.edu/~bard/xpp/xpp.html. (Accessed: April 2019)
56. Ermentrout, B.: Simulating, Analyzing, and Animating Dynamical Systems: A Guide to XPPAUT for Researchers and Students. Software, Environments and Tools (Book 14). Society for Industrial and Applied Mathematics (1987)
57. Evans, N.D., Chappell, M.J.: Extensions to a procedure for generating locally identifiable reparameterisations of unidentifiable systems. Mathematical Biosciences **168**(2), 137–159 (2000). https://doi.org/10.1016/S0025-5564(00)00047-X
58. Feinberg, M.: Complex balancing in general kinetic systems. Archive for Rational Mechanics and Analysis **49**(3), 187–194 (1972). https://doi.org/10.1007/BF00255665
59. Feinberg, M.: On chemical kinetics of a certain class. Archive for Rational Mechanics and Analysis **46**(1), 1–41 (1972). https://doi.org/10.1007/BF00251866
60. Feinberg, M.: Chemical reaction network structure and the stability of complex isothermal reactors—i. The deficiency zero and deficiency one theorems. Chemical Engineering Science **42**(10), 2229–2268 (1987). https://doi.org/10.1016/0009-2509(87)80099-4
61. Feinberg, M.: Foundations of Chemical Reaction Network Theory. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-03858-8
62. Feinberg, M., Ellison, P., Ji, H., Knight, D.: Chemical reaction network toolbox. https://crnt.osu.edu/CRNTWin. (Accessed: April 2019)

63. Feng, Z., Thieme, H.: Endemic models with arbitrarily distributed periods of infection I: Fundamental properties of the model. SIAM Journal on Applied Mathematics **61**(3), 803–833 (2000). https://doi.org/10.1137/S0036139998347834
64. Feng, Z., Xu, D., Zhao, H.: Epidemiological models with non-exponentially distributed disease stages and applications to disease control. Bulletin of Mathematical Biology **69**(5), 1511–1536 (2007). https://doi.org/10.1007/s11538-006-9174-9
65. Feng, Z., Zheng, Y., Hernandez-Ceron, N., Zhao, H., Glasser, J.W., Hill, A.N.: Mathematical models of Ebola—Consequences of underlying assumptions. Mathematical biosciences **277**, 89–107 (2016)
66. Fiechter, J., Rose, K.A., Curchitser, E.N., Hedstrom, K.S.: The role of environmental controls in determining sardine and anchovy population cycles in the California Current: Analysis of an end-to-end model. Progress in Oceanography **138**, 381–398 (2015). https://doi.org/10.1016/j.pocean.2014.11.013
67. Ghil, M., Zaliapin, I., Thompson, S.: A delay differential model of ENSO variability: Parametric instability and the distribution of extremes. Nonlinear Processes in Geophysics **15**(3), 417–433 (2008). https://doi.org/10.5194/npg-15-417-2008
68. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics **22**(4), 403–434 (1976). https://doi.org/10.1016/0021-9991(76)90041-3
69. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry **81**(25), 2340–2361 (1977). https://doi.org/10.1021/j100540a008
70. Givens, G.H., Hoeting, J.A.: Computational Statistics, 2nd edn. Computational Statistics. John Wiley & Sons (2012)
71. Grassly, N.C., Fraser, C.: Seasonal infectious disease epidemiology. Proceedings of the Royal Society B: Biological Sciences **273**(1600), 2541–2550 (2006). https://doi.org/10.1098/rspb.2006.3604
72. Grimm, V., Railsback, S.F.: Individual-based Modeling and Ecology. Princeton University Press (2005)
73. Guckenheimer, J., Holmes, P.: Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields, corr. 6th printing, 6th edn. Applied Mathematical Sciences. Springer (2002)
74. Guckenheimer, J., Myers, M.: Computing Hopf Bifurcations. II: Three Examples From Neurophysiology. SIAM Journal on Scientific Computing **17**(6), 1275–1301 (1996). https://doi.org/10.1137/S1064827593253495
75. Guckenheimer, J., Myers, M., Sturmfels, B.: Computing Hopf Bifurcations I. SIAM Journal on Numerical Analysis **34**(1), 1–21 (1997). https://doi.org/10.1137/S0036142993253461
76. Hansen, J.A., Penland, C.: Efficient approximate techniques for integrating stochastic differential equations. Monthly Weather Review **134**(10), 3006–3014 (2006). https://doi.org/10.1175/MWR3192.1
77. Hanski, I.A.: Metapopulation Ecology. Oxford Series in Ecology and Evolution. Oxford University Press (1999)
78. Heitmann, S.: Brain Dynamics Toolbox (2018). http://bdtoolbox.org. (Accessed: 20 June 2019) Alt. URL https://github.com/breakspear/bdtoolkit/
79. Helton, J., Davis, F.: Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. Reliability Engineering & System Safety **81**(1), 23–69 (2003). https://doi.org/10.1016/S0951-8320(03)00058-9
80. Hethcote, H.W., Tudor, D.W.: Integral equation models for endemic infectious diseases. Journal of Mathematical Biology **9**(1), 37–47 (1980). https://doi.org/10.1007/BF00276034
81. Hethcote, H.W., Yorke, J.A.: Gonorrhea Transmission Dynamics and Control. Lecture Notes in Biomathematics. Springer Berlin Heidelberg (1984). https://doi.org/10.1007/978-3-662-07544-9
82. Hiebeler, D.E., Millett, N.E.: Pair and triplet approximation of a spatial lattice population model with multiscale dispersal using Markov chains for estimating spatial autocorrelation. Journal of Theoretical Biology **279**(1), 74–82 (2011). https://doi.org/10.1016/j.jtbi.2011.03.027

83. Higham, D.J.: An algorithmic introduction to numerical simulation of stochastic differential equations. SIAM Review **43**(3), 525–546 (2001). https://doi.org/10.1137/S0036144500378302
84. Hirsch, M.W., Smale, S., Devaney, R.L.: Differential Equations, Dynamical Systems, and an Introduction to Chaos, 3rd edn. Elsevier (2013). https://doi.org/10.1016/C2009-0-61160-0
85. Hogg, R.V., McKean, J.W., Craig, A.T.: Introduction to Mathematical Statistics, 7th edn. Pearson (2012)
86. Hooker, G.: MATLAB Functions for Profiled Estimation of Differential Equations (2010). http://faculty.bscb.cornell.edu/~hooker/profile_webpages/
87. Hooker, G., Ramsay, J.O., Xiao, L.: CollocInfer: Collocation inference in differential equation models. Journal of Statistical Software **75**(2), 1–52 (2016). https://doi.org/10.18637/jss.v075.i02
88. Hooker, G., Xiao, L., Ramsay, J.: CollocInfer: An R Library for Collocation Inference for Continuous- and Discrete-Time Dynamic Systems (2010). http://faculty.bscb.cornell.edu/~hooker/profile_webpages/
89. Hurtado, P.J.: The potential impact of disease on the migratory structure of a partially migratory passerine population. Bulletin of Mathematical Biology **70**(8), 2264 (2008). https://doi.org/10.1007/s11538-008-9345-y
90. Hurtado, P.J.: Within-host dynamics of mycoplasma infections: Conjunctivitis in wild passerine birds. Journal of Theoretical Biology **306**, 73–92 (2012). https://doi.org/10.1016/j.jtbi.2012.04.018
91. Hurtado, P.J., Hall, S.R., Ellner, S.P.: Infectious disease in consumer populations: Dynamic consequences of resource-mediated transmission and infectiousness. Theoretical Ecology **7**(2), 163–179 (2014). https://doi.org/10.1007/s12080-013-0208-2
92. Hurtado, P.J., Kirosingh, A.S.: Generalizations of the 'Linear Chain Trick': Incorporating more flexible dwell time distributions into mean field ODE models. Journal of Mathematical Biology **79**, 1831–1883 (2019). https://doi.org/10.1007/s00285-019-01412-w
93. Iacus, S.M.: Simulation and Inference for Stochastic Differential Equations. Springer New York (2008)
94. Izhikevich, E.M.: Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting. Computational Neuroscience. MIT Press (2010)
95. Keener, J., Sneyd, J.: Mathematical Physiology I: Cellular Physiology, 2nd edn. Springer (2008)
96. Keener, J., Sneyd, J.: Mathematical Physiology II: Systems Physiology, 2nd edn. Springer (2008)
97. Kendall, B.E., Briggs, C.J., Murdoch, W.W., Turchin, P., Ellner, S.P., McCauley, E., Nisbet, R.M., Wood, S.N.: Why do populations cycle? A synthesis of statistical and mechanistic modeling approaches. Ecology **80**(6), 1789–1805 (1999). https://doi.org/10.1890/0012-9658(1999)080[1789:WDPCAS]2.0.CO;2
98. Kermack, W.O., McKendrick, A.G.: A contribution to the mathematical theory of epidemics. Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character **115**(772), 700–721 (1927)
99. Koopman, J.: Modeling infection transmission. Annual Review of Public Health **25**(1), 303–326 (2004). https://doi.org/10.1146/annurev.publhealth.25.102802.124353
100. Kot, M.: Elements of Mathematical Ecology. Cambridge University Press (2014)
101. Kozubowski, T.J., Panorska, A.K., Forister, M.L.: A discrete truncated Pareto distribution. Statistical Methodology **26**, 135–150 (2015). https://doi.org/10.1016/j.stamet.2015.04.002
102. Krylova, O., Earn, D.J.D.: Effects of the infectious period distribution on predicted transitions in childhood disease dynamics. Journal of The Royal Society Interface **10**(84) (2013). https://doi.org/10.1098/rsif.2013.0098
103. Kuehn, C.: Multiple Time Scale Dynamics, *Applied Mathematical Sciences*, vol. 191, 1 edn. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-12316-5
104. Kuznetsov, Y.: Elements of Applied Bifurcation Theory, 3 edn. Applied Mathematical Sciences. Springer New York (2004)

105. Lande, R., Engen, S., Saether, B.E.: Stochastic Population Dynamics in Ecology and Conservation. Oxford University Press (2003). https://doi.org/10.1093/acprof:oso/9780198525257.001.0001
106. Larsen, R.J., Marx, M.L.: Introduction to Mathematical Statistics and Its Applications, 5th edn. Pearson (2011)
107. Lee, E.C., Kelly, M.R., Ochocki, B.M., Akinwumi, S.M., Hamre, K.E., Tien, J.H., Eisenberg, M.C.: Model distinguishability and inference robustness in mechanisms of cholera transmission and loss of immunity. Journal of Theoretical Biology **420**, 68–81 (2017). https://doi.org/10.1016/j.jtbi.2017.01.032
108. Lee, S., Chowell, G.: Exploring optimal control strategies in seasonally varying flu-like epidemics. Journal of Theoretical Biology **412**, 36–47 (2017). https://doi.org/10.1016/j.jtbi.2016.09.023
109. Levin, S.A., Powell, T.M., Steele, J.W. (eds.): Patch Dynamics. Springer Berlin Heidelberg (1993). https://doi.org/10.1007/978-3-642-50155-5
110. Liao, J., Li, Z., Hiebeler, D.E., Iwasa, Y., Bogaert, J., Nijs, I.: Species persistence in landscapes with spatial variation in habitat quality: A pair approximation model. Journal of Theoretical Biology **335**, 22–30 (2013). https://doi.org/10.1016/j.jtbi.2013.06.015
111. Linz, P.: Analytical and Numerical Methods for Volterra Equations. Society for Industrial and Applied Mathematics (1985). https://doi.org/10.1137/1.9781611970852
112. Liu, Y., Khim, J.: Taylor's theorem (with Lagrange remainder). https://brilliant.org/wiki/taylors-theorem-with-lagrange-remainder/. (Accessed 15 April 2019)
113. Ma, J., Earn, D.J.D.: Generality of the final size formula for an epidemic of a newly invading infectious disease. Bulletin of Mathematical Biology **68**(3), 679–702 (2006). https://doi.org/10.1007/s11538-005-9047-7
114. Marino, S., Hogue, I.B., Ray, C.J., Kirschner, D.E.: A methodology for performing global uncertainty and sensitivity analysis in systems biology. Journal of Theoretical Biology **254**(1), 178–196 (2008). https://doi.org/10.1016/j.jtbi.2008.04.011
115. Mathworks: MATLAB Documentation: Choose an ODE solver. https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html. (Accessed: April 2019)
116. Mathworks: MATLAB Documentation: Solve Stiff ODEs. https://www.mathworks.com/help/matlab/math/solve-stiff-odes.html. (Accessed: April 2019)
117. May, R.: Stability and Complexity in Model Ecosystems. Landmarks in Biology Series. Princeton University Press (2001)
118. McCann, K.S.: Food Webs. Monographs in Population Biology (Book 57). Princeton University Press (2011)
119. Meiss, J.D.: Differential Dynamical Systems, Revised Edition. Society for Industrial and Applied Mathematics, Philadelphia, PA (2017). https://doi.org/10.1137/1.9781611974645
120. Merow, C., Dahlgren, J.P., Metcalf, C.J.E., Childs, D.Z., Evans, M.E., Jongejans, E., Record, S., Rees, M., Salguero-Gómez, R., McMahon, S.M.: Advancing population ecology with integral projection models: a practical guide. Methods in Ecology and Evolution **5**(2), 99–110 (2014). https://doi.org/10.1111/2041-210X.12146
121. Meshkat, N., Eisenberg, M., DiStefano, J.J.: An algorithm for finding globally identifiable parameter combinations of nonlinear ODE models using Gröbner bases. Mathematical Biosciences **222**(2), 61–72 (2009). https://doi.org/10.1016/j.mbs.2009.08.010
122. Meshkat, N., zhen Kuo, C.E., Joseph DiStefano, I.: On finding and using identifiable parameter combinations in nonlinear dynamic systems biology models and COMBOS: A novel web implementation. PLoS ONE **9**(10), e110,261 (2014). https://doi.org/10.1371/journal.pone.0110261
123. Miao, H., Dykes, C., Demeter, L.M., Wu, H.: Differential equation modeling of HIV viral fitness experiments: Model identification, model selection, and multimodel inference. Biometrics **65**(1), 292–300 (2008). https://doi.org/10.1111/j.1541-0420.2008.01059.x
124. Moraes, A., Tempone, R., Vilanova, P.: Hybrid Chernoff Tau-Leap. Multiscale Modeling & Simulation **12**(2), 581–615 (2014). https://doi.org/10.1137/130925657

125. Murdoch, W.W., Briggs, C.J., Nisbet, R.M.: Consumer–Resource Dynamics, *Monographs in Population Biology*, vol. 36. Princeton University Press, Princeton, USA (2003)
126. Murray, J.D.: Mathematical Biology: I. An Introduction. Interdisciplinary Applied Mathematics (Book 17). Springer (2007)
127. Murray, J.D.: Mathematical Biology II: Spatial Models and Biomedical Applications. Interdisciplinary Applied Mathematics (Book 18). Springer (2011)
128. Nash, J.C.: On best practice optimization methods in R. Journal of Statistical Software **60**(2), 1–14 (2014). http://www.jstatsoft.org/v60/i02/
129. Nash, J.C., Varadhan, R.: Unifying optimization algorithms to aid software system users: optimx for R. Journal of Statistical Software **43**(9), 1–14 (2011). http://www.jstatsoft.org/v43/i09/
130. Newman, M.: Networks, 2nd edn. Oxford University Press (2018)
131. Øksendal, B.: Stochastic Differential Equations: An Introduction with Applications, 6th ed. (6th corrected printing 2013) edn. Springer-Verlag Berlin Heidelberg (2003). https://doi.org/10.1007/978-3-642-14394-6
132. Ovaskainen, O., Saastamoinen, M.: Frontiers in Metapopulation Biology: The Legacy of Ilkka Hanski. Annual Review of Ecology, Evolution, and Systematics **49**(1), 231–252 (2018). https://doi.org/10.1146/annurev-ecolsys-110617-062519
133. Pineda-Krch, M.: GillespieSSA: Gillespie's Stochastic Simulation Algorithm (SSA) (2010). https://CRAN.R-project.org/package=GillespieSSA. R package version 0.5-4
134. Poggiale, J.C., Aldebert, C., Girardot, B., Kooi, B.W.: Analysis of a predator–prey model with specific time scales: A geometrical approach proving the occurrence of canard solutions. Journal of Mathematical Biology (2019). https://doi.org/10.1007/s00285-019-01337-4
135. Porter, M.A., Gleeson, J.P.: Dynamics on Networks: A Tutorial. ArXiv e-prints (2015). http://arxiv.org/abs/1403.7663v2
136. Porter, M.A., Gleeson, J.P.: Dynamical Systems on Networks: A Tutorial, *Frontiers in Applied Dynamical Systems: Reviews and Tutorials*, vol. 4. Springer (2016). https://doi.org/10.1007/978-3-319-26641-1
137. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2019). https://www.R-project.org/
138. Rackauckas, C.: A Comparison Between Differential Equation Solver Suites In MATLAB, R, Julia, Python, C, Mathematica, Maple, and Fortran. The Winnower (2018). https://doi.org/10.15200/winn.153459.98975
139. Ramsay, J.O., Hooker, G., Campbell, D., Cao, J.: Parameter estimation for differential equations: A generalized smoothing approach. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **69**(5), 741–796 (2007). https://doi.org/10.1111/j.1467-9868.2007.00610.x
140. Rand, R.H.: Lecture notes on nonlinear vibrations. Cornell eCommons (2012). https://hdl.handle.net/1813/28989
141. Raue, A., Kreutz, C., Bachmann, J., Timmer, J., Schilling, M., Maiwald, T., Klingmüller, U.: Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. Bioinformatics **25**(15), 1923–1929 (2009). https://doi.org/10.1093/bioinformatics/btp358
142. Reynolds, A., Rubin, J., Clermont, G., Day, J., Vodovotz, Y., Ermentrout, G.B.: A reduced mathematical model of the acute inflammatory response: I. Derivation of model and analysis of anti-inflammation. Journal of Theoretical Biology **242**(1), 220–236 (2006). https://doi.org/10.1016/j.jtbi.2006.02.016
143. Rinaldi, S., Muratori, S., Kuznetsov, Y.: Multiple attractors, catastrophes and chaos in seasonally perturbed predator-prey communities. Bulletin of Mathematical Biology **55**(1), 15–35 (1993). https://doi.org/10.1007/BF02460293
144. Rose, K.A., Fiechter, J., Curchitser, E.N., Hedstrom, K., Bernal, M., Creekmore, S., Haynie, A., ichi Ito, S., Lluch-Cota, S., Megrey, B.A., Edwards, C.A., Checkley, D., Koslow, T., McClatchie, S., Werner, F., MacCall, A., Agostini, V.: Demonstration of a fully-coupled end-to-end model for small pelagic fish using sardine and anchovy in the California Current.

Progress in Oceanography **138**, 348–380 (2015). https://doi.org/10.1016/j.pocean.2015.01.012
145. Ross, S.: Introduction to Probability Models, 11th edn. Elsevier (2014). https://doi.org/10.1016/C2012-0-03564-8
146. Saccomani, M.P., Audoly, S., DAngiò, L.: Parameter identifiability of nonlinear systems: The role of initial conditions. Automatica **39**(4), 619–632 (2003). https://doi.org/10.1016/S0005-1098(02)00302-3
147. Sauer, T.: Computational solution of stochastic differential equations. Wiley Interdisciplinary Reviews: Computational Statistics **5**(5), 362–371 (2013). https://doi.org/10.1002/wics.1272
148. Schelter, W.F.: Maxima (2000). http://maxima.sourceforge.net/. (Accessed: April 2019)
149. Segel, L., Edelstein-Keshet, L.: A Primer on Mathematical Models in Biology. Society for Industrial and Applied Mathematics, Philadelphia, PA (2013). https://doi.org/10.1137/1.9781611972504
150. Shaw, A.K., Binning, S.A.: Migratory recovery from infection as a selective pressure for the evolution of migration. The American Naturalist **187**(4), 491–501 (2016). https://doi.org/10.1086/685386
151. Shertzer, K.W., Ellner, S.P., Fussmann, G.F., Hairston, N.G.: Predator–prey cycles in an aquatic microcosm: Testing hypotheses of mechanism. Journal of Animal Ecology **71**(5), 802–815 (2002). https://doi.org/10.1046/j.1365-2656.2002.00645.x
152. Shinar, G., Alon, U., Feinberg, M.: Sensitivity and Robustness in Chemical Reaction Networks. SIAM Journal on Applied Mathematics **69**(4), 977–998 (2009). https://doi.org/10.1137/080719820
153. Shoffner, S., Schnell, S.: Approaches for the estimation of timescales in nonlinear dynamical systems: Timescale separation in enzyme kinetics as a case study. Mathematical Biosciences **287**, 122–129 (2017). https://doi.org/10.1016/j.mbs.2016.09.001
154. Smith, H.: An introduction to delay differential equations with applications to the life sciences, vol. 57. Springer (2010)
155. Society for Industrial and Applied Mathematics: DSWeb Dynamical Systems Software. https://dsweb.siam.org/Software. (Accessed: 1 April 2019)
156. Soetaert, K., Petzoldt, T., Setzer, R.W.: Solving differential equations in R: Package deSolve. Journal of Statistical Software **33** (2010). https://doi.org/10.18637/jss.v033.i09
157. Stieha, C., Montovan, K., Castillo-Guajardo, D.: A field guide to programming: A tutorial for learning programming and population models. CODEE Journal **10**, Article 2 (2014). https://doi.org/10.5642/codee.201410.01.02. https://scholarship.claremont.edu/codee/vol10/iss1/2/
158. Stone, L., Olinky, R., Huppert, A.: Seasonal dynamics of recurrent epidemics. Nature **446**(7135), 533–536 (2007). https://doi.org/10.1038/nature05638
159. Strang, G.: Introduction to Linear Algebra, 5th edn. Wellesley – Cambridge Press (2016). https://math.mit.edu/~gs/linearalgebra/
160. Strogatz, S.H.: Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering, 2nd edn. Studies in Nonlinearity. Westview Press (2014)
161. Theussl, S., Schwendinger, F., Borchers, H.W.: CRAN Task View: Optimization and mathematical programming (2019). https://CRAN.R-project.org/view=Optimization. (Accessed: April 2019)
162. Vestergaard, C.L., Génois, M.: Temporal Gillespie Algorithm: Fast simulation of contagion processes on time-varying networks. PLOS Computational Biology **11**(10), e1004, 579 (2015). https://doi.org/10.1371/journal.pcbi.1004579
163. Vodopivec, A.: wxMaxima: A GUI for the computer algebra system maxima (2018). https://github.com/wxMaxima-developers/wxmaxima. (Accessed: April 2019)
164. Wearing, H.J., Rohani, P., Keeling, M.J.: Appropriate models for the management of infectious diseases. PLOS Medicine **2**(7) (2005). https://doi.org/10.1371/journal.pmed.0020174
165. Wiggins, S.: Introduction to Applied Nonlinear Dynamical Systems and Chaos, *Texts in Applied Mathematics*, vol. 2, 2nd edn. Springer-Verlag New York (2003). https://doi.org/10.1007/b97481

166. Wikibooks: LaTeX — Wikibooks, The Free Textbook Project (2019). https://en.wikibooks.org/w/index.php?title=LaTeX&oldid=3527944. (Accessed: 22 April 2019)
167. Xia, J., Liu, Z., Yuan, R., Ruan, S.: The effects of harvesting and time delay on predator–prey systems with Holling type II functional response. SIAM Journal on Applied Mathematics **70**(4), 1178–1200 (2009). https://doi.org/10.1137/080728512
168. Yoshida, T., Hairston, N.G., Ellner, S.P.: Evolutionary trade–off between defence against grazing and competitive ability in a simple unicellular alga, *Chlorella vulgaris*. Proceedings of the Royal Society of London. Series B: Biological Sciences **271**(1551), 1947–1953 (2004). https://doi.org/10.1098/rspb.2004.2818
169. Yoshida, T., Jones, L.E., Ellner, S.P., Fussmann, G.F., Hairston, N.G.: Rapid evolution drives ecological dynamics in a predator–prey system. Nature **424**(6946), 303–306 (2003). https://doi.org/10.1038/nature01767