CHAPTER

Introduction to statistical inference[⊛] 2

pter outline

In this chapter, we give a gentle introduction to the key statistical concepts and computational techniques used in statistical modeling. That is, we explain what goes on under the hood when we fit a statistical model to data to estimate parameters and assess their associated uncertainty, for example, by

[⊛]This book has a companion website hosting complementary materials, including all code for download. Visit this URL to access it: https://www.elsevier.com/books-and-journals/book-companion/9780443137150.

computing standard errors (SEs) and confidence intervals (CIs). We do this for the two dominant schools of model-based inference, classical, or maximum likelihood (ML), and Bayesian posterior inference. They have far more in common than what we are sometimes made to believe. For instance, they are both model- as opposed to design-based (Little, 2004) and they have a strong focus on the likelihood function (see below). We think that to be a productive researcher you should be versed in both. However, these are truly vast topics. Hence, we can necessarily only scratch the surface.

We describe the main underlying concepts and techniques involved in the fitting of a parametric statistical model. Often, this model is hidden under the hood when we use software such as R, SPSS, or SAS, but it is nevertheless the basis of our analysis. Thus, it is essential that we understand what our software does. More importantly, an understanding of these basics allows us to apply more general model-fitting engines in order to fit statistical models in a completely flexible manner. This greatly extends our modeling options compared to what we can do with canned functions.

We begin by briefly describing probability as the foundation on which statistical modeling rests. Then, we introduce parametric models for statistical inference, that is, for parameter estimation, uncertainty assessment, hypothesis testing, goodness-of-fit, model selection, prediction, or missing value estimation. The likelihood function is a central concept in both the classical, or frequentist, and in the Bayesian schools of statistical inference. Hence, we spend most of the chapter covering parameter estimation and uncertainty assessment using either the likelihood function alone (that is, by ML) or by combining the likelihood function with prior distributions and obtaining Bayesian posterior inference.

2.1 Probability as the basis for statistical inference

In Chapter 1, we argued that there are only a few things in nature or life that are perfectly known, completely invariable, and thus fully understandable and predictable. That is, hardly anything is entirely deterministic; rather, there is an element of chance in almost everything. But on the other hand, very few things are completely irregular and unpredictable; rather, most things can be predicted at least in some broad, average sense. Thus, we need a way of drawing conclusions from noisy and incomplete observations and of making decisions in the face of the resulting uncertainty. Our methods for learning from data in an uncertain world must be able to describe phenomena both in an average sense as well as in terms of their variability, which is a major source of our uncertainty about any random phenomenon. A powerful basis for such a description is *probability*: the branch of mathematics that deals with chance processes and their outcomes (Pishro-Nik, 2014; Blitzstein & Hwang, 2019). Probability is a fascinating subject in its own right, because it enables you to better understand and navigate an uncertain world. Indeed, probability is the extension of logic from certain (i.e., true or false) events, which are extremely rare in life, to all events (Lindley, 2006). Probability is also the basis for statistical modeling and inference, the topics of this book.

A statistical model is a description of a probability mechanism that could have produced as one possible outcome the data set we have in hand. Thus, the same chance mechanism could also have produced a different data set or indeed an infinity of potential data sets. This view of our particular data set as being just one out of many possible data sets is the conceptual basis of the frequentist school of inference in statistics. It defines probability as the frequency of an event in hypothetical replicates (Casella & Berger, 2002). For now, we won't discuss where our models come from and how we build them; we say more about this in Chapters 3 and 18. Instead, we will cover inference, that is, the use of formal methods to infer unknown things once we have a model and some relevant data. "Statistical modeling" comprises both the development of a statistical model and its analysis using some method of inference, such as maximum likelihood or Bayesian posterior inference.

Statistical inference is based on probability, but in a sense it goes "the other way round" (Link & Barker, 2010). While probability assumes a stochastic process and then looks at its possible outcomes, that is, the data it may produce, in statistical inference we start with a data set and make assumptions about the stochastic process(es) that could have produced it. Combining data and model enables us to learn about features of the stochastic process (Fig. 2.1). This inductive process is called *statistical inference*. Its key task is the use of the data to infer the likely values of parameters that govern a stochastic process and to assess our uncertainty in these estimates. Statistical modeling is a powerful formal way of learning from data. In this sense, statistics could be called a meta-science that overarches all empirical sciences. Statisticians have also been called the "*custodians of the scientific method*" (Hooke, 1980).

A key concept in probability is that of the *random experiment*: the process of observing or measuring something unpredictable. A particular observation made in such an experiment is called an *outcome* (Pishro-Nik, 2014). Associated with each random experiment is the set of all possible outcomes, called a *sample space* or S. Favorite examples of random experiments for statisticians include the toss of a coin and the throw of a die, which have sample spaces with two or six possible outcomes, respectively. In ecology, more directly relevant random experiments include the following:

- We choose an animal or plant population and select, or capture, an individual and take some measurement on it, such as the body mass of peregrines, snout-vent length or body mass of snakes or box turtles, wing-length of butterflies, color morph of snakes, or the number of ectoparasites on dragonflies. In all these cases, we often assume that sampling is at random, that is, that every individual in the biological population has the same chance to be selected and to appear in our sample.
- We select a sample of survey sites or "local populations" and make an assessment of whether a study species is present or absent, of its population size, or the proportion of pairs that

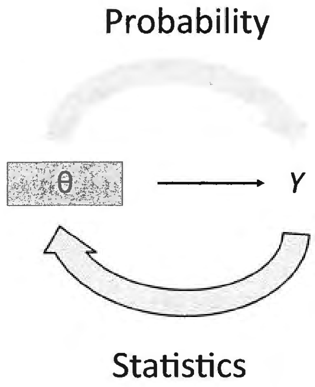

FIGURE 2.1

The duality between probability and statistics (compare with Fig. 1.1, which shows an analogous duality between data simulation and data analysis). The gray box denotes a stochastic process, or model, governed by parameter θ, and Y denotes some output, that is, observed data. In statistical modeling, we treat the observed data as the output from a stochastic process and use probability to describe this process in the form of a statistical model. Combining data and model lets us infer features of the data-generating process, such as parameter θ.

reproduce successfully, or take any other measurement at the scale of that spatial sample site. The sampled sites will typically again be assumed to be selected at random from a larger statistical population of sites that could have been selected and about which we want to make an inference. That could be the mean proportion or the total number of sites in the statistical population of sites that are inhabited or the average number of individuals per site.

In the former case, where we select individuals in a population, the sample space of these random experiments may be all possible measurements of: body mass of peregrines (e.g., some value between 400 and 1600 g), snout-vent length of snakes (e.g., some value in the range of 20–100 cm), butterfly wing length (e.g., 6–10 cm), adder color morph ("black" vs "gray"), or the number of dragonfly ecto-parasites (a non-negative integer). In the latter case, where we select a number of study sites or local populations, our sample space may consist of the values "presence" or "absence" of our target species or a count such as 0, 1, ... for an abundance at each site.

In all these examples, there will be chance involved in the particular outcomes observed. Thus, the actual outcome of a random experiment will vary when we repeat it (because we happen to select a different individual or because of measurement error) or when different people conduct it. This *sampling variability* creates an important part of the uncertainty that we have to deal with in statistical modeling. We note in passing that in probability the term *experiment* has a different meaning than in experimental design, where it always involves some manipulation in the form of a treatment (Snedecor & Cochran, 1980; Cochran & Cox, 1992; Quinn & Keough, 2002; Mead et al. 2012). In probability, the term *experiment* simply denotes the formal manner in which we take some observation or measurement in a chance process.

Often we are not interested in all possible outcomes of a random experiment, but only in some summary, or in an *event*, which is a subspace of the entire sample space. For instance, we may want to distinguish heavy from lean peregrines and take 800 g as a threshold for such a classification, or we simply add up the number of black and gray adders observed in a population. Similarly, we may summarize an assessment of presence and absence of a species at a set of sites by the total number of sites where a species is detected. Working with suitably defined events will be more directly relevant to our questions, and the randomness in the outcome of the original random experiment carries over to the events such defined.

To describe the effects of chance on the particular outcomes obtained in a random experiment, and therefore on the events observed, we use probability. A *probability function* for an event A is defined by the following three axioms (Pishro-Nik, 2014; Blitzstein & Hwang, 2019):

- Probability is a number between 0 and 1, where $P(A)=0$ denotes an impossible event and $P(A)=1$ is a certain event.
- The probability of the entire sample space $P(S)$ is 1, since by definition the outcome of a random experiment must be part of the set S.
- The probability of the union of mutually exclusive events (e.g., either or both of A and B) is the sum of the probability of the individual events, that is, $P(A \cup B)=P(A)+P(B)$. This is also called the addition rule.

Often the multiplication rule is also added, which describes the probability of the intersection of two events, i.e., that both of them occur or are true: $P(A \cap B)=P(A)P(B|A)$. Statistics is built on these axioms of probability, regardless of the interpretation that we attach to probability (Lindley, 2006; Spiegelhalter, 2019). Hence, these axioms and a host of further probability rules that can be deduced from them hold regardless of whether we interpret probability as a relative frequency in a classical, or frequentist, analysis, or as a measure of uncertainty or an expression of the amount of knowledge about something uncertain in a Bayesian analysis (see Sections 2.5–2.8).

2.2 Random variables and their description by probability distributions

Typically, in a study we are interested in a particular aspect of the sample obtained, that is, of the particular event recorded in a random experiment. Examples may include the proportion of black versus gray adders in our study populations, the number of occupied versus unoccupied grid cells, the number of plants in each of a number of populations, or the body mass recorded on each of the captured peregrines. All of these features of the studied samples are examples of *random variables*, abbreviated RVs. A RV is a real-valued function defined on the sample space of a random experiment (Pishro-Nik, 2014), and it is a key concept in probability and statistical modeling. While RVs may hardly even feature in the statistical training of most ecologists, they are the fundamental manner in which statisticians conceptualize a data set at the start of a statistical analysis. That is, the observed data (and, for hierarchical models, also unobserved data or random effects; see Section 3.5) will be treated as the outcomes of RVs. Our statistical model then describes the stochastic mechanisms that gave rise to the particular observed outcomes of these random processes.

The RV concept is critical for our understanding of statistical modeling, but meaning and use of the term may be confusing at first. Statisticians often distinguish RV Y, when meant as a name for a stochastic process, from the realized value or outcome of that RV, y. That is, they denote the abstract process in upper case and the realized value produced by the process in lower case. The realized value y denotes a single measurement, or observation, produced by the random process denoted Y. For instance, the body mass of a male peregrine falcon may be our RV Y, but when we measure the mass of our first peregrine, then we treat that measurement as our RV y. Worse yet, when we take multiple measurements of Y, say one from each of n examined peregrines, we will distinguish RVs $Y_1, Y_2, \ldots, Y_n$, and each will have an associated realized value $y_1, y_2, \ldots, y_n$. Moreover, we may distinguish RV Y from RV X, where Y may be the body mass of a captured peregrine and X its sex. The outcome of a random experiment consisting of capturing and examining n peregrines can be summarized by one random vector $\{x_1, x_2, \ldots, x_n\}$ for the sex of each bird and another, $\{y_1, y_2, \ldots, y_n\}$, for its mass.

As we will see below, for inference about non-hierarchical models we construct a likelihood function from the joint probability of all observed RVs. By this we mean for instance the set of all peregrine body mass measurements in our study, that is, the vector $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$. But in a hierarchical model (see Chapter 3), say, in a study of the detection or nondetection of a species in an occupancy species distribution model (Chapter 19), we may distinguish two RVs Z and Y: the former is the latent, i.e., unobserved, presence or absence of a species at a site, while the latter is the observed detection or nondetection at a site during one particular survey (see Chapter 19 for why presence/absence and detection/nondetection are not in general the same thing). Together, they produce the observed data $\{y_1, y_2, \ldots, y_n\}$. A probability expression defining our likelihood function will at first have to include all RVs, both unobserved/latent (i.e., Z) and observed (i.e., Y), see Kéry & Royle (2016: Section 2.4.6). See Section 3.5 for more about what is considered a RV in a hierarchical model and how this affects the likelihood construction. In summary, the specific meaning of "random variable" may vary somewhat depending on the context.

RVs are the essential building blocks for parametric statistical models. It is therefore paramount that you obtain a sound working knowledge of them. We strongly advocate thinking about your data and observations from the "random variable perspective." This is very natural in the context of learning about and understanding most things in nature, which can almost always be conceptualized as the outcome of a chance process, and hence as some RV Y, producing one or multiple observations y or $\mathbf{y}$, respectively.

2.2.1 Discrete and continuous random variables

We can distinguish two classes of RVs: discrete-valued and continuous. To some degree, this distinction coincides with the nominal, ordinal, and metric or interval measurement scales (Zar, 1998): the first two produce discrete-valued RVs, and the latter two continuous RVs.

Examples of discrete RVs include descriptions, labels, or names of classes, such as sex; color of hair, fur or plumage; geographic strata such as "Population A", "Population B" and "Population C"; or the identity of the field technician doing a bird point count. There is no sense of order in these class labels. In contrast, the other typical case of a discrete RV is a count, which does have a natural ordering. Examples of counts include the number of female nestlings in a nest with a known number of young, the number of asp vipers observed per visit to a site, or the number of whales migrating past a point on the coast during some time interval.

Continuous RVs can take on any value within some range, at least up to measurement accuracy. Measurements of length, distance, area, duration or of mass provide the typical examples of a continuous RV. Some measurements may be either discrete or continuous, depending on the specific type. For instance, the location of an animal in a movement study may be either continuous, when you record its exact coordinates at a point in time, or discrete, when you just note the identity of a grid cell wherein the animal is located. In general, you may translate between continuous and discrete RVs by "binning" or grouping outcomes of the former (see Fig. 2.3). Note that measurement accuracy is always finite. Hence, strictly, every continuous RV is measured in a finite number of discrete classes, but in practice this can usually be ignored when modeling continuous RVs.

Many types of RVs have natural bounds. For instance, a count cannot be negative, nor can most measurements, but a RV formed as the difference between two counts or measurements can. The range of a RV is also called its *support* and can be an important consideration when choosing a suitable probability model for a data set (see Section 3.2).

2.2.2 Description of random variables by probability distributions

A parametric statistical model (see Section 2.3) for a data set must describe all of its RVs, and for this we use probability distributions. They describe how the total probability of 1 in a random experiment is distributed across the sample space, that is, among all possible realizations of a RV. We use a *probability mass function* (PMF) for discrete RVs and a *probability density function* (PDF) for continuous RVs. Note that in practice we often use the terms "PDF" or "density" interchangeably for either a PMF or a PDF and the context determines which is meant. We begin by describing a PMF because this is arguably simpler to grasp.

A PMF $f(Y)$ gives the probability of every possible observable value y of a discrete RV Y:

$$f(Y) = P(Y = y) \tag{2.1}$$

The function value of a PMF lies always between 0 and 1, since it's a probability, and the sum of the PMF over all possible values is equal to 1. Indeed, these are the defining features of a PMF. Some

RVs are binary and have only two possible values, for example, a survival process and its associated RV Y can only produce an individual that is either dead or alive. Therefore, if we define $y=1$ as an animal that is alive and $y=0$ as one that is dead, then it is sufficient to define the probability $P(y=1)=\phi$, i.e., survival probability, to fully describe the binary RV (Royle & Dorazio, 2008). Since the sample space of the underlying random experiment contains only two values, and since the probability of the entire sample space is equal to 1, we know that $P(y=0)=1-\phi$. To describe a discrete RV with more than two possible outcomes such as hair color, where there are n mutually exclusive events in a finite set, we need only n-1 free probabilities, since the probability of one event can always be expressed as 1 minus the sum of the probabilities of all others. The binomial, multinomial, or categorical distributions are examples of such PMFs (Royle & Dorazio, 2008: chapter 2; Schaub & Kéry, 2022: chapter 2).

A PDF gives the *probability density* $f(Y)$ of every possible observable value of a continuous RV Y. This density is a non-negative value that is the limit of the area of a rectangle with height $f(Y)$ and base $(y-\delta, y+\delta)$, as the rectangle half-width δ goes to 0. A PDF has the peculiar feature that the probability (but not the density) of any particular value of Y is 0, and that the density may be >1 for some values of Y. To obtain the probability of a range of values of Y, say for (y_1, y_2) for density $f(Y)$, we must integrate f between y_1 and y_2, that is, $P(y_1 < Y < y_2) = \int_{y_1}^{y_2} f(Y)dY$. The integral over the entire support of a PDF is again equal to 1 by definition.

In statistics, there is a large number of stochastic processes that occur so frequently that their associated probability distributions have been given names. These named *statistical distributions* include the Bernoulli, binomial, Poisson, uniform, and the normal (or Gaussian), which we describe in more detail in Chapter 3. They are the typical building blocks of our statistical models; hence, it is important that you obtain a good understanding of them. All distributions are governed by a small number of constants, called *parameters*, which determine the specific form of a distribution (see also Fig. 2.5). For instance, a Poisson distribution is a typical stochastic model for unbounded counts. The PMF of a Poisson RV is as follows (remember that the exclamation mark is the factorial function):

$$f(y|\lambda) = P(Y=y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!} \tag{2.2}$$

A PMF gives the probability of observing any possible value of the RV Y as a function of some parameter(s). For a Poisson PMF, λ (lambda) is the only parameter, and it is both the mean and the variance of Y. In the expressions for f and P we see λ on the right side of a conditioning bar (|). This means that the values of these functions depend on the specific value chosen for λ. We suspect that the eyes of many ecologists will glaze over when confronted with this explicit way of writing a statistical distribution. However, it is extremely valuable to learn to understand what distribution functions are and how we use them. Play around with them by filling in some values and see what comes out, and also produce plenty of graphs, i.e., adopt the experimental approach to statistics (see Section 1.8). You should note that λ in the Poisson PMF is just a customary name, and you might just as well denote the Poisson mean with any other Greek or Roman letter or write it out as a word, as we do in our model-fitting engines (see Chapter 4) (In contrast, e is a mathematical "system letter"; it is is Euler's number 2.7183.).

Thus, the Poisson PMF enables you to compute the probability of observing any of the possible outcomes of a random experiment for which we define a RV that can be adequately approximated by the model of a Poisson distribution. This assumes that λ is known. So what if we assume for now $\lambda=2$? Then, we can for instance obtain the probability of a count of 1, which is $\lambda^y e^{-\lambda}/y! = 2^1 e^{-2}/1! \approx 0.27$. What about the probability of a count of zero? We can evaluate the PMF for $Y=0$ and obtain $2^0 e^{-2}/0! \approx 0.14$. Also, we can use this probability model to determine that the probability of a count of either 2 or of 3 is $2^2 e^{-2}/2! + 2^3 e^{-2}/3! \approx 0.27 + 0.18 = 0.45$ (that we can add these probabilities follows directly from the axioms of probability).

The definition of the Poisson PMF is Eq. 2.2, but we will frequently use shorthand. Thus, you will see a Poisson RV written in many different forms, including these (and remember the comment about the somewhat arbitrary naming of the Poisson parameter from above):

- $y \sim f(Y|\lambda)$
- $y \sim f_Y(Y|\lambda)$
- $y \sim Poisson(\lambda)$ or $y \sim Pois(\lambda)$
- $y \sim Poisson(Y|\lambda)$ or $y \sim Pois(Y|\lambda)$

The BUGS language is extremely simple and powerful for describing statistical models (Gilks et al., 1994). We use it to specify statistical models in software JAGS and NIMBLE and in a dialect also in Stan (see Chapter 4). In BUGS we define a Poisson RV in a similar way:

```
y ~ dpois(lambda)
```

In addition, the simplest statistical model for a set of Poisson RVs is a Poisson generalized linear model (see Chapter 3) which in R can be fit by issuing the following command:

```
glm(y ~ 1, family = poisson, data = list(y = y))
```

Although not the definition of a Poisson PMF, this comes close to it: it defines the values in vector `y` to be Poisson RVs and estimates a mean parameter for them.

In R, we have four functions for each of a large number of probability distributions. They are all named with a short form of the distribution name, and have a letter d, p, q, or r added in front. These letters denote, in this order, the probability density (or probability mass) function, the cumulative distribution function (or CDF), the quantile function, and the random number generation (RNG) function. What do these mean? Here's an illustration for all four with our example of a Poisson RV with `lambda` equal to 2:

```
# Look up help text for the Poisson
?dpois                                # Also '?Poisson'

# Poisson probability mass function (PMF) evaluated for y = 0
dpois(0, lambda = 2)                  # Probability of getting a value 0
dpois(0, lambda = 2, log = TRUE)      # Same on log scale

# Get density 'by hand' to emphasize dpois() is just shorthand!
lam <- 2; y <- 0
(lam^y)*exp(-lam) / factorial(y)      # Probability of getting a value 0

# Cumulative distribution function (CDF), or just 'distribution function'
ppois(3, lambda = 2)                  # Probability of getting a value of 0, 1, 2 or 3
sum(dpois(0:3, 2))                    # Same, summing up probs 'by hand'
plot(ppois(0:9, 2), type = 'h', lend = 'butt', lwd = 20, ylim = c(0, 1),
  xlab = 'Value of Y', ylab = 'P(Y <= y)', main = "CDF of Y")     # not shown

# Quantile function: value of y for which CDF(y) has a certain value
qpois(0.85, lambda = 2)
```

```
# Random number generator (RNG) function
set.seed(2016)                        # Set seed if want same numbers as we have
rpois(n = 10, lambda = 2)             # 10 Poisson(lambda = 2) random numbers

[1] 0.1353353                         # P(y = 0)
[1] -2                                # log(P(y = 0))
[1] 0.1353353                         # same 'by hand'
[1] 0.8571235                         # P(y <= 3)
[1] 0.8571235                         # same 'by hand'
[1] 3                                 # Value of y for which CDF evaluates to 0.85
[1] 1 1 3 0 2 0 2 4 0 0               # 10 random Poisson numbers with lambda = 2
```

We have emphasized how `dpois(..., log = TRUE)` yields the log-probability, since this will become very important in model fitting and for many methods of doing model selection. Note also that the CDF is an alternative to the PMF or PDF for defining a probability distribution and works equally for discrete and continuous RVs. The CDF is often denoted $F(Y)$ and it gives the probability that Y takes on a value less or equal than y, that is, $P(Y \leq y)$. We don't cover the CDF in this book but will briefly encounter it when discussing quantile residuals (Dunn & Smyth, 1996) in Chapters 5 and 18.

We can easily produce a plot of a PMF, and we highly encourage you to do this to train your intuition about probability distributions. Fig. 2.2 shows the PMF of our Poisson(lambda = 2) RV.

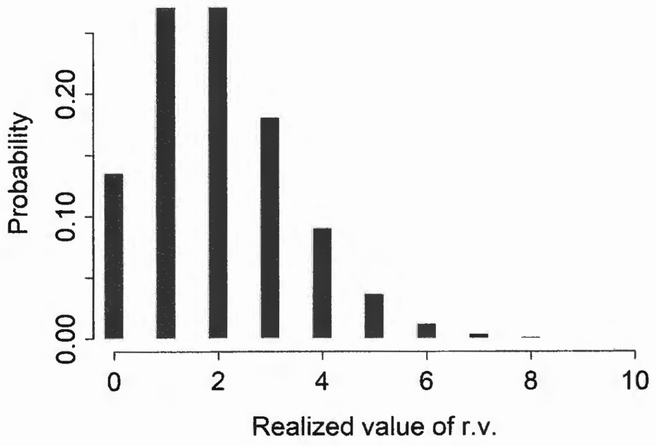

FIGURE 2.2

PMF of a Poisson RV with parameter $\lambda = 2$. The y-axis shows the probability of each possible value for this discrete RV. The probability of values greater than about 10 is not 0, but is simply too small to be seen at the scale of the y-axis.

```
# Plot PMF of Poisson with lambda = 2 (Fig. 2.2)
par(mar = c(6,8,5,4), cex.lab = 1.5, cex.axis = 1.5, cex.main = 1.5)
plot(0:10, dpois(0:10, lambda = 2), xlab = 'Realized value of RV',
  ylab = 'Probability', main = 'Poisson(2) PMF', type = 'h', lend = 'butt',
  lwd = 20, col = 'gray30', frame = FALSE, las = 1)
```

As an example for the probability distribution of a continuous RV, let's take the peregrine body mass example for which a normal distribution is our usual statistical model. The PDF of a normal, or Gaussian, RV is as follows:

$$f(y|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \tag{2.3}$$

This distribution has two parameters, μ and σ, which are the mean and the standard deviation (SD) of a normal RV. If we pick a value for each, then the PDF enables us to compute the probability density (or alternatively, the log-density) for any possible observation and, by integration over a range of values, to obtain the probability that our RV will take on a value within this range. We next give an illustration for all four variants of the normal distribution functions in R with a Normal(600, 30) RV, as in Chapter 4 where we assume this describes body mass of male peregrine falcons.

```
?dnorm                                         # Also '?Normal': Look up the help
                                               # text for the Normal

# Gaussian, or normal, probability density function (PDF) for y = 650
dnorm(650, mean = 600, sd = 30)
dnorm(650, mean = 600, sd = 30, log = TRUE)    # Same on log scale

# Get the density 'by hand' to emphasize dnorm() is just a shortcut
mu <- 600; sig <- 30; y <- 650
1/(sig*sqrt(2*pi)) * exp(-(y - mu)^2 / (2 * sig^2))

# Cumulative distribution function (CDF), or just 'distribution function'
pnorm(600, mean = 600, sd = 30)                # Prob. of value between -Inf and 600
# Next is same, but integrating the PDF 'by hand'
integrate(dnorm, lower = -Inf, upper = 600, mean = 600, sd = 30)
plot(pnorm(seq(400, 800, by = 1), 600, 30),
  type = 'l', lwd = 5, ylim = c(0, 1),
  xlab = 'Value of Y', ylab = 'Prob. density',
  main = "CDF of Y", frame = FALSE)            # not shown

# Quantile function: value of y for which CDF(y) has a certain value
qnorm(0.95, mean = 600, sd = 30)

# Random number generator (RNG) function
set.seed(2016)
rnorm(n = 10, mean = 600, sd = 30)             # 10 Normal(600, 30) random numbers
```

```
[1] 0.003315905                                   # f(y = 650)
[1] -5.709025                                     # log-density: log(f(y = 650))
[1] 0.003315905                                   # same 'by hand'
[1] 0.5                                           # P(y <= 600)
0.4995709 with absolute error < 1.4e-14           # same 'by hand'
[1] 649.3456                                      # value for which 95% of values
                                                  # are smaller
[1] 572.5577 630.0374 598.3073 608.8994 516.2559  # random numbers
[6] 591.5178 577.0947 579.4497 611.0122 605.4899
```

We can't directly use the PDF to obtain the probability of a particular measurement, but we can do so for a range of values. For illustration, let's compute the probabilities for a large number of 5g-bins of body mass. We will produce two plots, one for the density and the other for the probabilities of the binned body mass RV (Fig. 2.3). You can imagine the density plot on the left as what we would get if, in the plot on the right, we make the bins smaller and smaller, for example, by choosing a bin width of 0.1 g instead of 5 g. As always, we highly encourage you to adopt the experimental approach to statistics and try this out.

```
# Compute probabilities for classes of binned Normal RV
limits <- seq(500, 700, by = 5)
midpts <- seq(500 + 5/2, 700 - 5/2, by = 5)
cumProb <- pnorm(limits, mean = 600, sd = 30)
probs <- diff(cumProb)

# Plot probability density function (PDF) of Normal(600, 30)
par(mfrow = c(1, 2), mar = c(6,6,5,2), cex.lab = 1.5, cex.axis = 1.5,
  cex.main = 1.5)                                              # Fig. 2.3
curve(dnorm(x, mean = 600, sd = 30), 500, 700, xlab = 'Body mass (g)',
  ylab = 'Probability density', main = 'PDF of Normal(600, 30) RV', type = 'l',
  lwd = 3, col = 'gray30', frame = FALSE, las = 1)
```

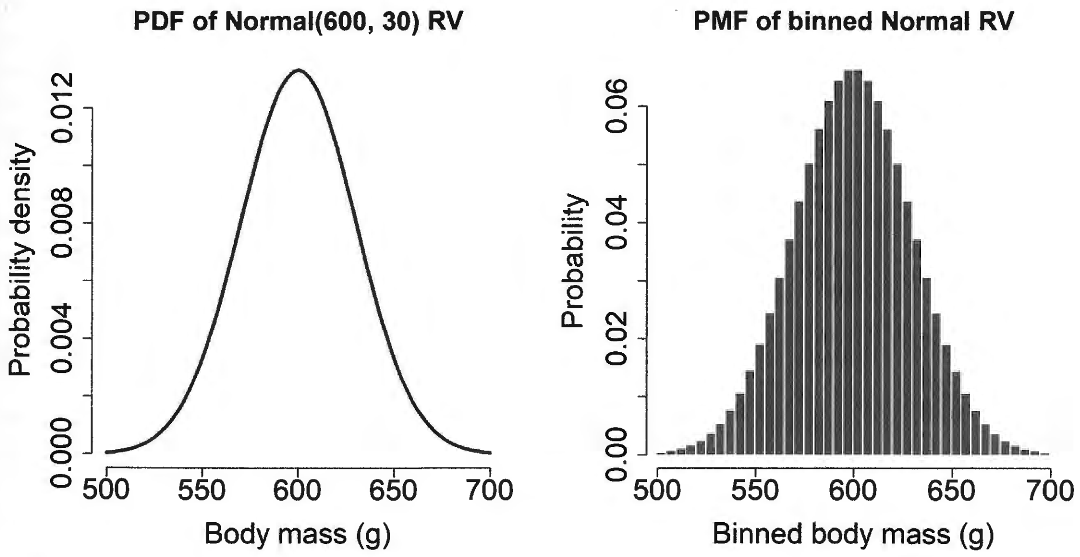

FIGURE 2.3

Left: PDF of a normal random variable with parameters $\mu = 600$ and $\sigma = 30$. Right: Corresponding PMF of binned body mass, with a bin width of 5 g. Note the different scales of the y-axis. When you recompute and draw the plot on the right with much narrower bins, you will approximate the density plot on the left.

```
# Plot probabilities for 5g-binned mass random variable
plot(midpts, probs, xlab = 'Binned body mass (g)', ylab = 'Probability',
  main = 'PMF of binned Normal RV', type = 'h', lend = 'butt', lwd = 5,
  col = 'gray30', frame = FALSE, las = 1)
```

2.2.3 Location and spread of a random variable, and what "modeling a parameter" means

The probability density (PMF or PDF) yields a complete description of a RV, but we may want to characterize it in a simpler way. Two important properties of a RV are its central tendency, or location, and its dispersion, or spread. The former is also called the expectation E (or mean) and can be envisioned as the average over a large number of realizations of the RV. Thus, for a discrete RV the expectation is a weighted mean of all possible outcomes, where the weights correspond to the probability of each: $E(Y) = \sum_{i=1}^{\infty} p(y_i)y_i$. For a continuous RV, we replace the sum by an integral. The variance of the RV is a function that quantifies its dispersion or variability. It is defined as another expectation $Var(Y) = E[(Y - E(Y))^2]$. Expectation and variance can be obtained for most RVs, and for some distributions, they correspond to a parameter of the distribution.

For instance, for both Poisson and normal distribution, the expectation corresponds to a parameter of the density: it is λ in the Poisson and μ in the normal distribution. In most statistical models we will want to "model" a response variable, that is, an observed or possibly an unobserved RV, by expressing its mean as some function of a covariate (Section 3.4). This simply means that we replace the parameter for the mean in the PDF/PMF by a function of these covariates, with new parameters that replace the old parameter, that is, the mean. Such a function relating the mean to covariates may be nonlinear or linear. In this book, we focus on mathematically linear relationships between the mean of a response or some transformation of it in the case of non-normal generalized linear models (GLMs; Section 3.3) and a set of covariates, that is, on linear models. However, conceptually, the step towards the modeling of nonlinear functions in the mean is straightforward; see Pinheiro & Bates (2000), Bolker (2008), Hobbs & Hooten (2015) or Hooten & Hefley (2019). Note that the formal definition of a linear model is a model where a parameter can be expressed as the sum of a series of explanatory variables (i.e., covariates) multiplied by their effects, and "linear" refers to the effects of these explanatory variables. We can easily describe very wiggly curves by what mathematically are linear models; thus, a linear model does not restrict us to mere straightline relationships. Models with higher-order polynomial terms (i.e., squares, cubes, etc.) of a covariate offer a good example of wiggly relationships represented by a model that is mathematically linear.

In this book, we focus on the normal, Poisson, and Bernoulli/binomial distributions; see Chapter 3. They all have one parameter which represents the mean response. Obviously, this makes it straightforward to model the response by modeling the mean of the distribution as a function of covariates. However, in many other distributions there is no correspondence between the mean and any of the parameters. Then, to express the mean response as a function of covariates, you must first reparameterize such that the mean becomes a parameter by what is known as *moment matching* (Hobbs & Hooten, 2015; Kéry & Royle, 2016).

A normal distribution has a parameter for the dispersion (i.e., the variance, inverse precision, or SD), but in many other distributions there is no free parameter for the dispersion. Rather, the dispersion is a function of the mean, as in the binomial and the Poisson distributions. The dispersion of a RV is often treated as a sort of a nuisance in many statistical models: that is, a model must account for the unexplained variability in the RVs to get the uncertainty assessments right, but the dispersion is rarely a parameter of interest in its own right. However, there may be exceptions to this, and it is also possible to specify a model for the dispersion, that is, to explain variability in the variance with covariates (see also Section 6.4). In addition, in a hierarchical model (Chapter 3) we also model structure in the variance of a response, but do so in a more implicit manner.

2.2.4 Short summary on random variables and probability distributions

This concludes our introduction to that foundational topic of the RV and its probability distribution. RVs are the defining features of statistical models, because they represent the stochastic building blocks of our models. Since explanation is often easier with concrete examples, we have singled out the Poisson and the normal distributions. In Chapter 3, we will say more about when these statistical models are suitable for a data set and also discuss a small number of other commonly used probability distributions. You can find long lists of discrete and continuous distribution functions in almost any statistics book, including Bolker (2008) or Royle & Dorazio (2008), and of course on the internet. In addition, many of these distributions are related to each other such that one arises as a special or limiting case of another; see the famous flowchart at the end of the *Table of Common Distributions* section in Casella & Berger (2002). Please read up on these fundamental pieces of statistical modeling because the small list of distributions in this book is far from sufficient to cover all stochastic systems that you are likely to encounter in your research.

We make one final remark on terminology: typically, "probability distribution" can mean either a CDF, PMF, or PDF. "Probability density" is frequently used synonymously for a PMF or a PDF. In contrast, "distribution function" is normally reserved for the CDF.

2.3 Statistical models and their usages

What is a parametric statistical model? According to Millar (2011):

"*A parametric statistical model is a collection of joint density functions*, $f(\mathbf{y}; \boldsymbol{\theta})$ ...."

The bold face font for data $\mathbf{y}$ and parameter(s) $\boldsymbol{\theta}$ defines both as vectors or matrices. In addition, Lee et al. (2017) say that "*the model ... should specify how the data could have been generated probabilistically.*" Hence, a statistical model is a probabilistic representation of a data-generating mechanism which provides a density for every datum in an analysis. A shorter and more approximate definition is "*a parametric statistical model is a collection of density functions for its random variables.*"

To clarify, let's define a model for our Poisson example above, where we drew 10 RVs that we interpret as counts, by constructing the joint density for this data set. In the simplest case, we assume that these 10 numbers were produced independently from stochastic processes with identical parameter `lambda`. In statistics, the resulting RVs are called *independent and identically distributed*, or i.i.d. Statistical independence means that the value of one count contains no information about the value of any other, once we account for the Poisson parameter `lambda` (for more on independence, you can look up *statistical independence* in any book on probability ... and while you're at it, you may also read about *conditional independence*). The "identical" in i.i.d. means that all 10 numbers were produced by a Poisson RV with the same value of the intensity parameter. Under i.i.d. assumptions, the joint density of the 10 RVs is a simple product of the densities of each constituent RV in the vector **y**.

$$f(\mathbf{y}|\boldsymbol{\theta}) = f(\{y_1, \ldots, y_{10}\}|\boldsymbol{\theta}) = \prod_{i=1}^{10} f(y_i|\theta) \tag{2.4}$$

A heuristic depiction of a statistical model is shown in Fig. 2.4. It emphasizes that the data at hand are assumed to be the outcome from a stochastic process, whose main characteristics we try to emulate in our model, which is the gray box. Of course, we will never be able (nor even want) to describe the data-generating processes exactly; a useful model must be an abstraction and hence a simplification. But we aim for a representation that is useful given the goals of our modeling, which may be prediction, explanation, or summarization (Tredennick et al., 2021; see also Section 1.1).

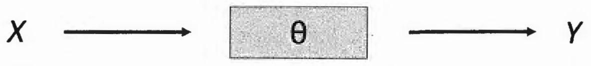

FIGURE 2.4

Schematic of a model (gray box) with parameter *θ*; *X* and *Y* denote some input (e.g., covariates) and output (i.e., observed data), respectively. In parametric statistical modeling, the model typically represents the main processes that generate the observed data (modified from Breimann (2001)).

Parametric modeling, that is, the act of building a model and then fitting it to a data set, is at once a rigid and a flexible process. It has a rigid component, because we make some very specific choices. For example, we must choose the distribution assumed for the observed RVs, and in more complex (i.e., hierarchical or mixed) models, we also must make choices about latent RVs and the distributions that govern them. Furthermore, we typically have covariates and will assume specific functional forms for the linkage with the parameters in the model (although this may be relaxed in "semi-parametric" models such as generalized additive models, or GAMs; Wood, 2017).

But in another sense, parametric modeling is very flexible, since an important part of the model will be adjusted (i.e., fitted) to the specific data as closely as possible. This is where the parameters come in. Parameters give the distributions assumed for the RVs their specific form. It is astonishing how different the same distribution can look depending on the values of its parameters. Fig. 2.5 illustrates this for a beta distribution, which is used to model continuous variables between 0 and 1. These arise for instance as the ratio of two continuous variables, such as the proportion of a leaf that is eaten by an insect.

```
# Look how different a beta can look depending on the values chosen for its parameters
par(mfrow = c(2,2))                                  # Fig. 2.5
curve(dbeta(x,10,10), 0, 1, lwd = 3, col = 1, main = "beta(10, 10)")
curve(dbeta(x,0.1,0.1), 0, 1, lwd = 3, col = 1, main = "beta(0.1, 0.1)")
curve(dbeta(x,1,100), 0, 1, lwd = 3, col = 1, main = "beta(1, 100)")
curve(dbeta(x,100,1), 0, 1, lwd = 3, col = 1, main = "beta(100, 1)")
```

A parametric statistical model has many uses. Most importantly, we can apply the model to our observed data *y* to infer the key features of the process that we suppose has produced these data (at least as an approximation, which is why Royle & Dorazio (2008) write about "approximating models"). The most immediate such inference will be estimation of the parameter(s) $\boldsymbol{\theta}$ and assessment of the associated uncertainty of these estimates, typically in the form of some quantification of error

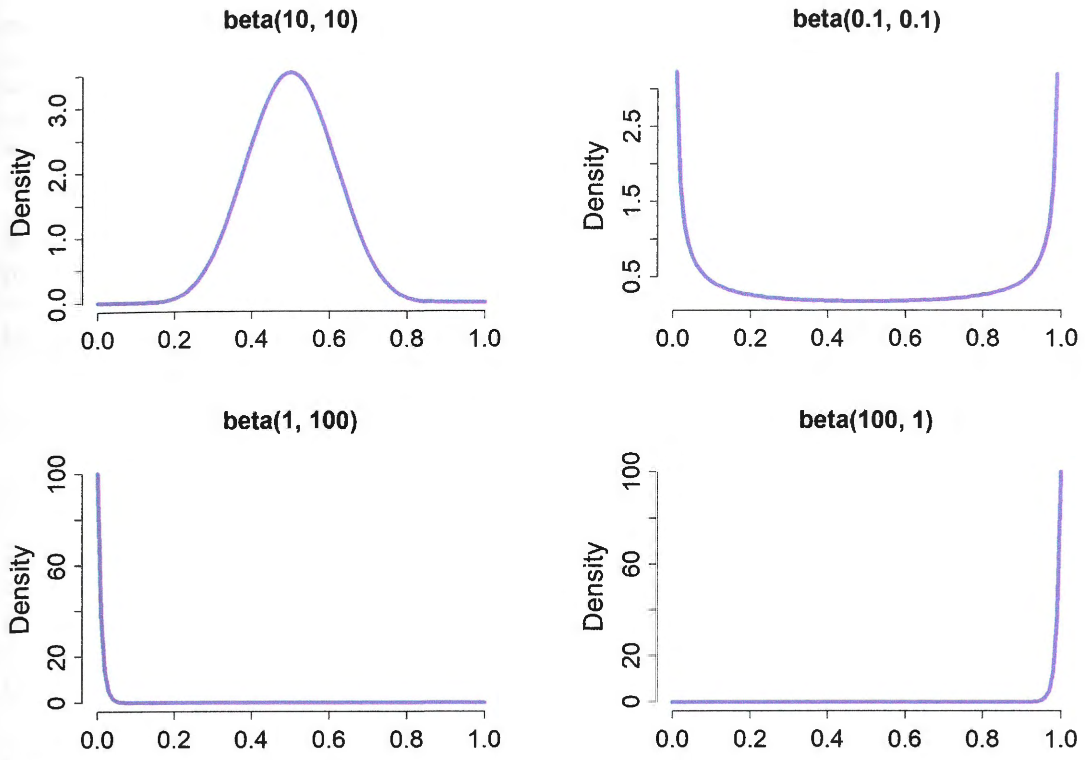

FIGURE 2.5

Four variants of a beta distribution which differ solely in terms of the values of its two parameters.

(e.g., SEs, or posterior SDs) or uncertainty intervals, such as CIs or credible intervals (CRIs). An example might be a capture-recapture study where a key parameter of interest is population size, which we want to estimate along with a SE and 95% CI. In addition, we may also want to test hypotheses about likely values of the parameters, or assess the goodness-of-fit to gauge the degree to which the observed data agree with the key assumptions made by our model. Yet another use is to compare one model variant with another, that is, to do model selection. Further, sometimes we may have observed some output and we may have estimated some model parameters, but we lack any information about the inputs, that is, about the covariates X in Fig. 2.4. We may then use the model to estimate (or "impute") the values of the missing covariates. And finally, we may have observed the input and then along with parameter estimates use the model to estimate (or "predict") missing responses from a model. A typical example of such "missing response estimation" is in species distribution modeling (SDM) (Elith & Leathwick, 2009), where we fit a model to a smaller data set that includes both environmental and species occurrence or abundance data. If we know covariate values also for some larger area, we may use them along with the estimated parameters to extrapolate the expected species occurrence or abundance to this wider area to yield a species distribution map.

All this is known as *statistical inference* and, to summarize, includes: parameter estimation, uncertainty assessment of the estimates, goodness-of-fit assessment, model selection, missing value estimation (imputation), and missing response estimation (prediction). Fundamentally, these are all inductive goals: we take a sample of observations and from them and a model infer features of the wider statistical population from which our data was drawn. Models are our vehicles for inference.

There are two main schools in statistics for making inferences: classical or frequentist, and Bayesian inference. We cover both, because we are convinced that to become an effective statistical modeler you need to understand and be able to use both. Furthermore, their apparent differences are often exaggerated. In an important way they are closely related: both are clearly based on a parametric statistical model and for both likelihood is a central concept. Therefore, we next introduce the likelihood function, and then in Sections 2.5 and 2.6, describe statistical inference that is directly based on the likelihood function, that is, maximum likelihood estimation (MLE). In Sections 2.7 and 2.8, we describe Bayesian posterior inference, which adds to the likelihood function a probability statement about how likely parameter values are by specification of so-called prior distributions and computes the posterior distribution of a parameter.

2.4 The likelihood function

The likelihood function is a key concept for statistical inference. It is a main ingredient of both the classical and the Bayesian schools of learning from data. The likelihood function represents the formal connection between what you observe (the data) and what you don't observe (the unknowns) and especially the parameters you want to estimate.

To quote Millar (2011) again, "*the likelihood function is the (joint) density function evaluated at the observed data, and regarded as a function of* $\boldsymbol{\theta}$ . . ." *That is,*

$$L(\boldsymbol{\theta}) \equiv L(\boldsymbol{\theta}; \mathbf{y}) = f(\mathbf{y}; \boldsymbol{\theta}) \tag{2.5}$$

The likelihood function provides a measure of the relative support, in terms of the probability of the observed data under a model, for different possible values of $\boldsymbol{\theta}$ (Edwards, 1992; Pawitan, 2013).

Likelihood is conceptually simple, yet it can be surprisingly hard to grasp. As ecologists, we are not normally used to thinking about models as the joint densities of the data (Eq. 2.4). Furthermore, algebraically the likelihood function is the exact same thing as the joint density function of the data, but in likelihood the density is "read in the opposite direction," that is, the role of data and parameters is reversed. While for the joint density we take the parameters as given, in the likelihood we take the data as given and evaluate the density as a function of varying values of the parameters.

One consequence of this "turning the joint density around" is that the likelihood function is *not* a PDF over the parameter space. Remember the two defining features of a probability density: it evaluates to a non-negative value everywhere and its sum (for a PMF) or integral (for a PDF) is equal to 1. This is usually not the case when a joint density is used as a likelihood function.

The likelihood function of an i.i.d. sample of n data points is constructed as a product of n densities, which usually (but not always) each evaluate to a number between 0 and 1. Thus, the value of a likelihood function when evaluated for a data set will be relatively small, and the likelihood will be smaller for a larger data set (e.g., with $n = 100$) than for a smaller one (e.g., with $n = 10$) . . . which however does not mean that a larger data set is less likely than a smaller. To make calculation of the likelihood easier, we usually work with the log (more specifically, the natural log) of the likelihood function, where the product in the joint density function of the data (Eq. 2.4) is replaced by a sum:

$$\log L(\theta) = l(\theta) = \sum_{i=1}^{n} \log(f(y_i|\theta)) \tag{2.6}$$

The likelihood function is easiest to understand in the simple case of i.i.d. data. However, in most of our analyses, you won't have data that are strictly i.i.d., but instead will have covariates or random effects. How do we then construct the likelihood functions for our data? The simpler of the two cases is when we have covariates ... which is virtually always. In this case, we simply replace the parameter θ with a function of these covariates plus a new set of parameters. We will see this in most other chapters in the book where we have covariates in a model, and also in Section 2.6.

Often we will have unobserved RVs, or random effects, in a model. In that case, the construction of a likelihood function is a little more involved. In a frequentist analysis the likelihood is now constructed by starting with the *joint density function of both the observed and the unobserved RVs*. From this, the unobserved RVs (i.e., the random effects) are removed by summation (for discrete random effects) or by integration for the more typical case where random effects are continuous; see Berger et al. (1999), Hobert (2000), Pinheiro & Bates (2000), Royle & Dorazio (2008: chapter 2); Royle et al. (2014: Chapter 2) and Kéry & Royle (2016: chapter 2). The result is the integrated, or marginal, likelihood, wherein the latent variables or random effects are replaced by one or two parameters of the distribution that we assume for them. Typical examples would be a mean and a variance for random effects for which we assume a normal distribution, as in all linear or generalized linear mixed models; see Chapters 10, 14, and 17. However, the random-effects distribution might also be a Bernoulli in an occupancy model (see Chapter 19) or a Poisson in an *N*-mixture model (Chapter 19B).

The likelihood (or alternatively, the log-likelihood [LL]) function contains most or all the information about the parameters that is contained in our data set. Look up the *likelihood principle* for more details. Both statistical inference methods that we will apply use likelihood, but in a different manner. We next cover them both, beginning with classical inference and followed by Bayesian posterior inference.

2.5 Classical inference by maximum likelihood and its application to a single-parameter model

The crucial difference between classical and Bayesian statistics is their different use of probability. In classical inference, probability is used to describe variability in the data among hypothetical replicate data sets. As a consequence, probability is also used to describe the variability of things that we compute from the data, such as estimates of parameters or CIs. In classical statistics, probability is *never* directly used to make an inferential statement involving uncertainty about likely values of an estimated parameter.

That is, classical statistics defines probability as a long-run relative frequency of data and functions of data in hypothetical replicates of a study. This relative-frequency interpretation of probability has also given rise to the label "frequentist." An important concept in classical statistics is that of the *estimator*: a method for computing an estimate ($\hat{\theta}$, the hat indicates an estimate) of a parameter (θ) from data. Conceptually, the RV characteristic of the data carries over to that of the estimator. Hence, in classical inference we imagine that a single estimate of a parameter obtained from application of an estimator to our data set is just one out of an infinitely large population of such estimates that we could have obtained in hypothetical replicates of our study. This hypothetical distribution of estimates across replicates is called the *sampling distribution* of the estimator. Often, there is theory that tells us about the form of this distribution. How wide or narrow the sampling distribution is determines the precision of our estimates as assessed by the SE or CI. The SE of an estimate is simply the SD of the hypothetical replicate estimates, while the CI is given by appropriate percentiles of the sampling distribution of the estimator. Statistical theory enables us to make a statement about that distribution of the estimates even though the only thing we may ever have in hand is a single estimate. As a result, the concept of a SE or CI can be extremely elusive to nonstatisticians. Resampling approaches such as the bootstrap (Section 2.5.4) can be very helpful in this respect to build intuition.

In much of classical statistics, we work directly with the likelihood function when we do ML inference. The likelihood based on the density of the data, or the LL based on the log-density, provides a numerical measure for how likely the observed data are under different assumed values of the parameters in a statistical model. By picking those values of the parameters that maximize the value of the likelihood function when evaluated for our data set, we hope to obtain a good guess for these unknown parameters. This is the principle of maximum likelihood estimation (Edwards, 1992; Millar, 2011; Pawitan, 2013; Lee et al., 2017). The resulting maximum likelihood estimates (MLEs) make the observed data set the most probable. ML is the most widely used estimation method in all of statistics. In addition, many other estimation methods such as least-squares for normal linear models (see Chapters 4–9) or iterative reweighted least-squares (IRLS) for generalized linear models (see Chapter 11–15) yield MLEs for these particular model classes.

Next, we show how to obtain the MLEs for a given model and data set and illustrate three methods for assessing the uncertainty associated with these estimates. We illustrate ML with two examples. First, in the rest of Section 2.5, we use the counts simulated in Section 2.2.2 and assume a one-parameter Poisson model for them. In this simple setting the principle of ML will be easiest to understand. Our second example, in Section 2.6, will use an actual data set for which we will assume a two-parameter Poisson model. Both models can be viewed as a Poisson regression or Poisson GLM; see Chapters 3 and 11–14. The first example is the Poisson equivalent of the simplest normal model in Chapter 4, while the second is the Poisson equivalent to ordinary linear regression in Chapter 5.

2.5.1 Introduction to maximum likelihood

We start with the simple case where our model has only a single parameter that we want to estimate from our data using a likelihood function. For illustration, we work with two data sets with different sample sizes, to see how the different amount of information affects our inferences. We create these data sets by respectively drawing 10 and 100 Poisson RVs in R. Owing to sampling variability, the likelihood function evaluated for these two data sets typically will not have its maximum at the same place. More importantly, the shape of the likelihood function around the maximum will be different, and this is crucial for the assessment of the uncertainty of the estimates.

```
# Two data sets representing counts with expected count lam
set.seed(2)
lam <- 2
y1 <- rpois(10, lambda = lam)     # Small data set (n = 10)
y2 <- rpois(100, lambda = lam)    # Large data set (n = 100)
table(y1) ; table(y2)

y1                                # Small data set
0 1 2 3 4
1 2 3 2 2

y2                                # Large data set
 0  1  2  3 4 5 6
12 36 21 16 7 7 1
```

To use the method of ML, we need to choose a suitable probability model for our data. We will assume that we have Poisson RVs, which is a standard assumption for counts without any natural upper bound (Chapter 3). That is, we make the strong claim that the data-generating process for our data is adequately described by the Poisson PMF, that is, $p(y|\lambda)=\exp(-\lambda)\lambda^y/y!$, and that the only thing unknown is the value of λ (this claim is exactly true for our simulated data, but for real data it is never exactly true). The likelihood function is algebraically identical to the joint density of the data under that model. Under i.i.d. assumptions, the joint density is a product of the densities of each single datum. To obtain the MLE, we combine the density function and the data and see which parameter value leads to the highest likelihood function value when evaluated for our data set.

In practice, we will always work with the natural log of the likelihood function, because that takes us from a product of likelihoods to a sum of log-likelihoods. This tends to avoid numerical problems with very small or very large values when the function is evaluated for a data set. In addition, we will use *function minimization* to obtain the MLEs; hence, instead of the log-likelihood function, we will work with the negative of the log-likelihood function. All three functions lead to the same MLEs and actually, we may often say "we maximize the likelihood" when in fact we are minimizing the negative log-likelihood (NLL).

```
# Define likelihood function in R
L <- function(lambda, y) {
  Li <- dpois(y, lambda)            # likelihood contribution of each data point i
  L <- prod(Li)                     # Likelihood for entire data set is a product
  return(L)
}

# Define log-likelihood function in R
LL <- function(lambda, y) {
  LLi <- dpois(y, lambda, log = TRUE) # log-likelihood contribution of i
  LL <- sum(LLi)                    # Log-likelihood for entire data set is a sum
  return(LL)
}

# Define negative log-likelihood function in R
NLL <- function(lambda, y) {
  LL <- dpois(y, lambda, log=TRUE)  # log-likelihood contribution of i
  NLL <- -sum(LL)                   # *neg* log-likelihood for entire data set is a sum
  return(NLL)
}
```

Next, we evaluate all three R functions for the smaller data set `y1` and for a large range of possible values for the Poisson parameter `lambda`, draw a plot of each function, and determine the MLEs by hand, or by eye. Fig. 2.6 shows that all three variants of the likelihood function lead to the same value of `lambda` that either maximizes the function (for the likelihood and the log-likelihood) or minimizes it (for the negative log-likelihood). Thus, we can use any of the three functions to find the MLEs. However, it becomes obvious from looking at the very small values along the y-axes that working with the likelihood directly can be numerically challenging. Hence, we usually work with the log-likelihood or the negative log-likelihood function when doing ML inference.

```
# Evaluate all three functions for large range of possible values for lambda
possible.lam <- seq(0.001, 4, by = 0.001)
like1 <- loglike1 <- negloglike1 <- numeric(length(possible.lam))
for(i in 1:length(like1)){
  like1[i] <- L(possible.lam[i], y1)                # Likelihood
  loglike1[i] <- LL(possible.lam[i], y1)            # log-Likelihood
  negloglike1[i] <- NLL(possible.lam[i], y1)        # negative log-likelihood
}

# Plot profiles (this is Fig. 2.6)
par(mfrow = c(1, 3), mar = c(6,6,5,3), cex.lab = 1.5, cex.axis = 1.5, cex.main = 2)
plot(possible.lam, like1, xlab = 'lambda', ylab = 'L', main = 'Likelihood', frame = FALSE)
abline(v = possible.lam[which(like1 == max(like1))], col = 'gray', lwd = 3, lty = 3)
plot(possible.lam, loglike1, xlab = 'lambda', ylab = 'LL', main = 'log-Likelihood',
  frame = FALSE)
abline(v = possible.lam[which(loglike1 == max(loglike1))], col = 'gray', lwd = 3, lty = 3)
plot(possible.lam, negloglike1, xlab = 'lambda', ylab = '-LL', main = 'Negative log-Likelihood',
  frame = FALSE)
abline(v = possible.lam[which(negloglike1 == min(negloglike1))], col = 'gray', lwd = 3, lty = 3)
```

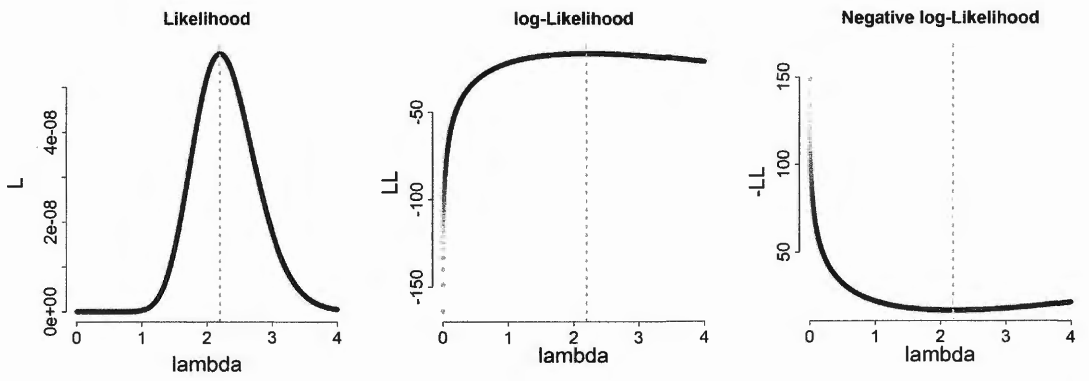

FIGURE 2.6

Curves of the likelihood (left), the log-likelihood (middle), and the negative log-likelihood (right) functions when assuming our 10 counts are i.i.d. Poisson RVs. Vertical gray line shows the value of lambda of 2.2 which is the extremum in all three profiles. We call this the *maximum likelihood estimate*, or MLE. This figure shows a very large range of values along both axes, see Fig. 2.7 for a zoomed-in version of the log-likelihood. Be prepared to be astonished at how the shape can look different for a different scaling of the *y*-axis.

```
# Determine MLE 'by hand'
possible.lam[which(like1 == max(like1))]          # Maximum likelihood
possible.lam[which(loglike1 == max(loglike1))]    # Maximum log-likelihood
possible.lam[which(negloglike1 == min(negloglike1))]  # Minimum negative log-likelihood
```

To get acquainted with the principle of ML, it is good to show things in a super-simple case where we can evaluate the functions essentially "by hand," that is, by trying out a large number of possible values for a parameter and then picking the one that leads to the highest function value of the LL or the lowest for the NLL. But it's hard or impossible to do this with more complex models, for which we use function minimization instead, for example, through `optim()`. This is the R workhorse for function minimization and thus for obtaining our MLEs (see also Section 4.7).

Next, we repeat our search for the MLE for our data set `y1` using `optim()`. We can directly use our R function for the NLL and provide it as an argument to `optim()`, along with an initial value for the start of the iterative search. Function miminization is a huge topic, and indeed function `optim()` enables you to choose among several optimization methods. Here, we simply work with the function's default method. Function minimisation is an iterative process, so we have to initialize it somewhere.

```
# Determine MLE using function minimization in R with optim()
inits <- c('lambda' = 1)
out <- optim(inits, NLL, y = y1)   # Optimize function for y1 over lambda
out
(MLE <- out$par)                   # Grab the MLE

> out
$par
lambda
   2.2

$value
[1] 16.67301

$counts
function gradient
      30       NA

$convergence
[1] 0

$message
NULL

> (MLE <- out$par)                 # Grab the MLE
lambda
   2.2
```

We get a warning message (not shown), which we can ignore here. And we see that minimizing the NLL yields the same value for the MLE as did our earlier search "by hand" in the computations for Fig. 2.6.

ML is the gold standard in statistical estimation, and has become extremely well known during the first 100 years since its discovery (Fisher, 1922; Aldrich, 1997). In a sense, it is an automatic inference method: one simply defines a likelihood function for a model and data set and then uses function minimization to obtain the MLEs (Efron, 1986). Importantly, MLEs have several desirable properties, such as invariance to transformation, efficiency, consistency, and asymptotic normality; see Royle & Dorazio (2008): chapter 2, and Kéry & Royle (2016): also chapter 2. Transformation invariance means that we can maximize a Poisson likelihood in terms of parameter `lambda` or in terms of the log of `lambda`, and when we exponentiate the MLE of the latter, we will get the same MLE as in the former case. This is useful, since it is good to avoid constraints on parameters by transforming them in a suitable way when minimizing a likelihood function. Examples of parameters with natural constraints are the Poisson mean or a variance, both of which we may log-transform, or a probability, which we may logit-transform (see Chapter 3). Efficiency and consistency can simply be subsumed under the claim that MLEs are "good" when your sample size is big enough (which is often, but not always). Asymptotic normality is a big deal, since it is the basis for the most common method of computation of SEs and CIs around MLEs, as we will see below.

Any estimate without an associated uncertainty assessment, such as a SE or a CI, is of dubious value. Maximization of the (log-)likelihood yields point estimates that are easy to get, but how can we obtain uncertainty assessments for our MLEs? We next illustrate three such methods:

- CIs based on a likelihood ratio test (LRT): These are based on LRT theory and use the actual shape of the likelihood function around its maximum.
- Asymptotic normality of the MLE ("inverting the Hessian"): SEs and Wald-test-based CIs can be obtained under the assumption that the MLEs are distributed as normal around the true parameter value.
- The bootstrap (parametric or nonparametric).

The information about estimation uncertainty (or put in another way, about the precision of an estimate) comes from the shape of the likelihood function in the vicinity of the MLEs in the case of the first two, and from the spread of the experimental (re-)sampling distribution of the MLEs when using the bootstrap.

Turning to the first two methods, we note that the maximum values of the LL function are about –17 for the smaller data set and –170, that is, 10 times more, for the 10 times larger data set. This does *not* mean that the small data set is more likely than the larger one, but is simply a consequence of the different sample sizes: the sum of a larger number of negative values will be more negative than the sum of a smaller number of such values. The important thing is how much the LL *changes* as we vary the values assumed for the parameters. That is, proportional likelihoods are equivalent and carry the same information about the parameters (Pawitan, 2013; Lee et al., 2017).

Let us first zoom in to the middle panel in Fig. 2.6 and plot the LL function when evaluated for both the small and large data set (Fig. 2.7). When comparing LLs it is customary to set the maximum to 0 and to check a range of the parameter such that the range of LL is roughly between –4 and 0 (Lee et al., 2017). We already computed these values for the smaller data set, but we yet have to do this for the large data set.

```
# Evaluate L, LL and NLL for large sample
like2 <- loglike2 <- negloglike2 <- numeric(length(possible.lam))
for(i in 1:length(like2)){
  like2[i] <- L(possible.lam[i], y2)                    # Likelihood
  loglike2[i] <- LL(possible.lam[i], y2)                # log-Likelihood
  negloglike2[i] <- NLL(possible.lam[i], y2)            # negative log-likelihood
}
```

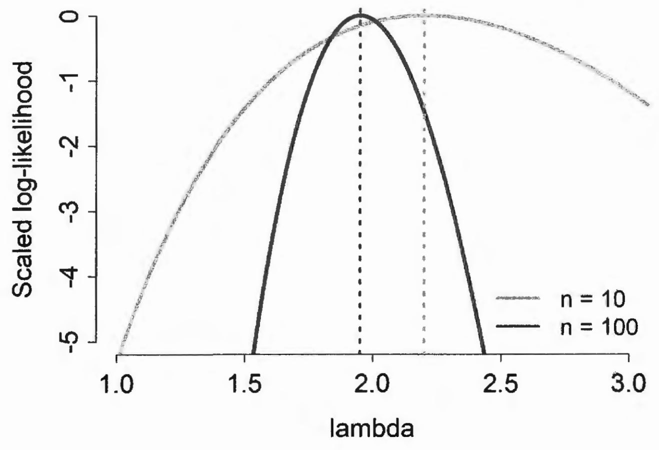

FIGURE 2.7

Curves of the scaled log-likelihood in the vicinity of the MLE for the small ($n = 10$) and the large ($n = 100$) variant of our data set. Both curves were scaled to a maximum of 0 to make them comparable. Vertical gray lines show the two MLEs. Note that the gray curve is just a zoom into the middle panel of Fig. 2.6.

```
# Compute value of the maximized log-likelihood for both data sets
(lmax1 <- max(loglike1, na.rm = TRUE))
(lmax2 <- max(loglike2, na.rm = TRUE))

# Plot LL in the vicinity of the MLE for both samples (Fig. 2.7)
ylim <- c(-5, 0)                                          # Choose scaling of y axis
plot(possible.lam, loglike1-max(loglike1), xlab = 'lambda',
  ylab = 'Scaled Log-likelihood', type = 'l', frame = FALSE, col = 'gray',
  lwd = 5, ylim = ylim, xlim = c(1, 3))
lines(possible.lam, loglike2-max(loglike2), col = 'gray40', lwd = 5)
abline(v = possible.lam[which(loglike1 == max(loglike1))],
  lwd = 3, lty = 3, col = 'gray')                         # MLE for small data set
abline(v = possible.lam[which(loglike2 == max(loglike2))],
  lwd = 3, lty = 3, col = 'grey40')                       # MLE for large data set
legend('bottomright', col = c('gray', 'gray40'), lwd = 4,
  bty = 'n', legend = c("n = 10", "n = 100"), cex = 1.5)
```

We note that the MLE obtained from the larger sample happens to be closer to the known truth in our case than the MLE from the smaller sample. But let us instead focus on the shape of the two likelihood functions. Remember that a likelihood function tells us how much support by the observed data different assumed values of a parameter have under a given model. The more concentrated the likelihood around its maximum, the better will be our "guess" about the parameter value, that is, the more precise will be our estimate. Thus, the curvature and, in general, the shape, of the LL near its maximum is a key for uncertainty assessments of MLEs. Thus, we clearly see there is much less uncertainty about the estimate of λ when $n = 100$ than when $n = 10$.

The first method for characterizing uncertainty of MLEs that we present is based directly on the shape of the LL function near its maximum and uses theory about the distribution of differences of LLs (called a likelihood ratio test) when sample sizes are large. This method is more accurate, but (much) more cumbersome than the subsequent two. The second method uses the expected curvature of the LL function near its maximum and is based on the assumption of asymptotic normality of the MLEs. In practice, it is by far the most common of these approaches. For moderate to large sample sizes and for likelihood functions that are symmetric and approximately quadratic, the resulting estimates of SEs and CIs are typically completely adequate. Moreover, the CIs will then be very similar under the first two methods. Finally, the bootstrap is an under-used method which is extremely flexible and can often be very easy to use. We illustrate it for uncertainty assessment of MLEs, using the two variants of a parametric and a nonparametric bootstrap.

2.5.2 Confidence intervals by inversion of the likelihood ratio test

The LRT is the basic significance test in the likelihood framework (Millar, 2011). When we are interested in comparing an estimate $\hat{\theta}$ with some hypothetical parameter value, for example, $\theta_0 = 0$, theory says that for large samples $2(LL(\hat{\theta}) - LL(\theta_0)) \sim \chi^2_r$, where r is 1 for a scalar parameter. This expression can be inverted to obtain a CI around the estimate at some test level α, that is, for a $100(1-\alpha)\%$ CI. For instance, an approximate 95% CI is defined by $2(LL(\hat{\theta}) - LL(\theta_0)) < 3.84$ and therefore by $LL(\hat{\theta}) - LL(\theta_0) < 3.84/2 = 1.92$ (Bolker, 2008; Millar, 2011; Pawitan, 2013). Note that 3.84 is the 95th quantile of a Chi-squared distribution with 1 d.f., as you can check out by typing in R `qchisq(0.95, 1)`.

To apply this method for obtaining a 95% CI, we choose the two values of `lambda` which lie vertically below the intersections of the LL curve with a horizontal line drawn at 1.92 units below the maximum (Fig. 2.8). For a 99% CI we would place that horizontal line at 6.635/2 = 3.32 units below the maximum. For our one-parameter model, the resulting CI is equivalent to a so-called profile interval (Bolker, 2008).

```
# Method 1 for CIs: profile-likelihood- or LRT-based 95% CI
# -----------------------------------------------------------
(lmax <- max(loglikel, na.rm = TRUE))              # Maximized ll
(llim <- lmax - 1.92)                              # value of that is 3.84/2 units below

# Plot profile focused on some range around MLE (Fig. 2.8)
par(mar = c(6, 6, 6, 3), cex.lab = 1.5, cex.axis = 1.5, cex.main = 1.5)
ooo <- which(abs(loglikel-lmax) < 3.5)
xlim <- c(0.9 * possible.lam[min(ooo)], 1.1*possible.lam[max(ooo)])
ylim <- c(-20, -16)
plot(possible.lam[ooo], loglikel[ooo], type = 'l', lwd = 3,
  xlab = 'lambda', ylab = 'Log-likelihood', frame = FALSE, xlim = xlim,
  ylim = ylim, main = 'Log-likelihood curve in vicinity of the MLE\n with LRT-based 95% CI')

# Add to graph vertical line that is chisq(1) / 2 + max(ll)
idx <- which(loglikel == max(loglikel))            # order which contains max
arrows(possible.lam[idx], llim, possible.lam[idx], lmax,
  length = 0.25, angle = 30, code = 3, lwd = 2)
abline(h = llim, lty = 2, lwd = 3)
text(2.4, mean(c(llim, lmax)), "1.92", cex = 2, srt = 90)

# Compute two CI limits separate for left and right branch of profile
l.left <- loglikel[1:idx]                          # Left branch
l.right <- loglikel[(idx+1):length(loglikel)]      # Right branch
```

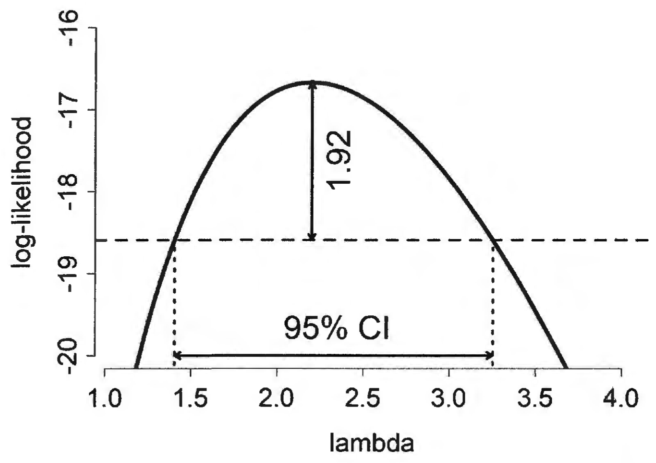

FIGURE 2.8

A likelihood curve showing a LRT-based 95% CI for lambda computed for the smaller data set with $n = 10$. The value of 1.92 is one half of the 0.95 quantile of a Chi-squared distribution with 1 d.f., that is, 3.84/2. Adapted from Millar (2011). The resulting interval goes from 1.404 to 3.251. Note also the slight asymmetry of the curve.

```
# Compare lower limit (llim) with values in l.left and then with those in l.right
idx.left <- which(abs(l.left - llim) == min(abs(l.left - llim), na.rm = TRUE))
idx.right <- which(abs(l.right - llim) == min(abs(l.right - llim), na.rm = TRUE))

# Compute profile confidence limits...
(LCL <- possible.lam[1:idx][idx.left])
(UCL <- possible.lam[(idx+1):length(loglike1)][idx.right])

# add the confidence limits into the plot
segments(possible.lam[idx.left], -21, possible.lam[idx.left], llim, lwd = 2, lty = 3)
segments(possible.lam[idx+idx.right], -21, possible.lam[idx+idx.right], llim, lwd = 2, lty = 3)
arrows(possible.lam[idx.left], -20, possible.lam[idx+idx.right], -20, length = 0.25,
  angle = 30, code = 3, lwd = 2)
text(2.4, -19.5, "95% CI", cex = 2, srt = 0)

# Compare with canned profile CI method in R for our Poisson GLM
exp(confint(glm(y1 ~ 1, family = poisson)))

Waiting for profiling to be done...
    2.5%    97.5%
1.403978 3.251454
```

This code is long-winded and the method fairly complicated, but we show it here to give you an intuition for how such a profile interval is computed. For the specific example with our simple Poisson counts, we can also use R's `confint()` function which gives the same result in a more concise but far less transparent way.

2.5.3 Standard errors and confidence intervals based on approximate normality of the maximum likelihood estimates

Statistical theory tells us that asymptotically, that is, as sample size goes to infinity, the sampling distribution of the MLE becomes normal (Royle & Dorazio, 2008: chapter 2, Pawitan, 2013: chapter 2, Lee et al., 2017: chapter 1), that is:

$$\hat{\theta} \sim Normal(\theta, I(\hat{\theta})^{-1}) \tag{2.7}$$

Thus, with increasing sample sizes, our MLEs will get closer and closer to the true value of the parameter they're estimating, and their variance will be the inverse of $I(\hat{\theta})$, which is called the observed Fisher information (Chapter 2 in Millar, 2011; Royle & Dorazio, 2008). $I(\hat{\theta})$ is the negative of the second partial derivative of the LL function with respect to the parameters in the model, when evaluated at the MLE and for our data set. For a single-parameter model $I(\hat{\theta})$ is a scalar, but more generally, for a model with s parameters it will be an $s \times s$ matrix and becomes the observed Fisher information matrix. To see this, note that the first derivative of the LL function with respect to the parameters gives the slope in each direction, while the second partial derivative quantifies the curvature of the LL around its maximum, that is, the rate of change of the slope at each point of the curve. At the MLEs, the latter function must be negative, since the slope changes from positive to negative. The Fisher information is expressed as the negative of the function value, so information increases when the curvature is greater.

The matrix of second partial derivatives of the LL is called the Hessian matrix. Hence, when obtaining the MLEs from the LL in a model with two or more parameters, we get the variance-covariance (VC) matrix of the MLEs by taking the inverse of the negative of the Hessian matrix ***H*** when evaluated for our data set at the MLEs.

$$\mathbf{VC}(\hat{\boldsymbol{\theta}}) = [\mathbf{I}(\hat{\boldsymbol{\theta}})]^{-1} = -\mathbf{H}(\hat{\boldsymbol{\theta}})^{-1} \tag{2.8}$$

The variances are on the diagonal of that matrix. When using `optim()` to get the MLEs, the argument `hessian = TRUE` asks for the Hessian to be computed. Interestingly, since we optimize the negative of the LL function with `optim()` to obtain the MLEs, what `optim()` reports as "the Hessian" is in fact the negative of the Hessian with respect to the LL. Hence, to get the VC matrix of the estimated parameters, we then simply take the inverse of what is reported as the Hessian. To obtain SEs, we take the square root of the inverse of the diagonal of the Hessian matrix, see Section 2.6.

In this way it is very easy to compute asymptotic SEs, where "asymptotic" should always remind us that this is a large-sample approximation. Moreover, we can use the resulting SEs for a so-called Wald-based $100 * (1-\alpha)\%$ CI as the MLE plus/minus $z_{1-\alpha/2}$ times the asymptotic SE of the estimate, that is, $\hat{\theta} \pm z_{1-\alpha/2} SE(\hat{\theta})$. Here, $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of a standard normal distribution, so for a 95% CI, we will use $z = 1.96$. We next demonstrate all of this in R.

```
# Method 2 for SEs and CIs: 'inverting the Hessian'
# -----------------------------------------------------
inits <- c(lambda = 1)

# Small data set (n = 10)
(out1 <- optim(inits, NLL, y = y1, hessian=TRUE))
(MLE1 <- out1$par)                  # Grab MLE
(VC1 <- solve(out1$hessian))        # Get variance-covariance matrix
(VC1 <- 1/out1$hessian)             # Same for a one-parameter model !
(ASE1 <- sqrt(diag(VC1)))           # Extract asymptotic SEs
```

```
# Large data set (n = 100)
(out2 <- optim(inits, NLL, y = y2, hessian=TRUE))
(MLE2 <- out2$par)                 # Grab MLE
(VC2 <- solve(out2$hessian))       # Get variance-covariance matrix
(VC2 <- 1/out2$hessian)            # Same for a one-parameter model!
(ASE2 <- sqrt(diag(VC2)))          # Extract asymptotic SEs

# Compare with the truth MLEs and ASE's for small and large data set
print(cbind('truth' = lam, MLE1, ASE1, MLE2, ASE2), 5)

       truth MLE1    ASE1 MLE2    ASE2
lambda     2  2.2 0.46904 1.95 0.13964
```

We note a decent agreement between the estimates and the truth, that is, the value used to simulate our data, and we see that the estimated SEs are much smaller for the larger data set. We can use them to get Wald-type CIs, which we compare with the LRT-based intervals computed by R.

```
# Do-it-yourself Wald-type CI for smaller data set
print(CI1 <- c(MLE1 - 1.96 * ASE1, MLE1 + 1.96 * ASE1), 4)

# LRT-based CI for smaller data set (from R)
print(exp(confint(glm(y1 ~ 1, family = poisson))), 4)

# DIY Wald-type CI for larger data set
print(CI2 <- c(MLE2 - 1.96 * ASE2, MLE2 + 1.96 * ASE2), 4)

# LRT-based CI for larger data set (from R)
print(exp(confint(glm(y2 ~ 1, family = 'poisson'))), 4)

> # Do-it-yourself Wald-type CI for smaller data set
lambda lambda
 1.281  3.119

> # LRT-based CI for smaller data set (from R)
 2.5% 97.5%
1.404 3.251

> # DIY Wald-type CI for larger data set
lambda lambda
 1.676  2.224

> # LRT-based CI for larger data set (from R)
 2.5% 97.5%
1.689 2.237
```

We note a better agreement between the more precise LRT-based intervals and the Wald-type intervals for the larger sample size. This is not surprising, because the distribution of the MLEs is more approximately normal for larger than for small samples.

2.5.4 Obtaining standard errors and confidence intervals using the bootstrap

The bootstrap (Efron, 1979) is a very useful statistical method that is not nearly as well known among ecologists as it should be. Millar (2011) succinctly describes it as follows:

"Frequentist inferential procedures are based on the notion of repeat sampling and so, in the likelihood context, it is necessary to determine the properties and behavior of ML-based inference under repetition of the experiment. The general tools and techniques ... have been obtained from a well established

body of theory that required large doses of calculus, probability theory and mathematical statistics... The bootstrap effectively replaces this calculus and theory with pure computational effort. The essential concept of bootstrapping is to emulate repetition of the experiment by simulating new data on the computer, followed by recalculation of the MLE using the simulated data."

The bootstrap is perhaps most useful for obtaining uncertainty assessments of estimates and derived quantities such as predictions and can also be used for goodness-of-fit assessments (Chapter 18, Kéry & Royle, 2016: chapter 2). Here, we first bootstrap the SEs and CIs parametrically, and then nonparametrically. We will first conduct all the computations for the two bootstraps and then compare the resulting estimates.

For a *parametric bootstrap* with the smaller data set we repeat the following a large number of times (remember that our MLE in the Poisson model is 2.2 for the small, and 1.95 for the large data set):

(1) We simulate another data set of size *n* (i.e., the same size as the original dataset) by drawing from a Poisson(2.2) or from a Poisson(1.95),
(2) we fit a Poisson model with an intercept only using R function `glm()` and
(3) we save the resulting new MLE.

For a *nonparametric bootstrap*, we repeat the following a large number of times:

(1) We draw another data set of size *n from our actual data* (i.e., `y1` or `y2`). This sampling is with replacement, hence some values will occur more than once, while others will appear not at all in the resampled data sets.
(2) We fit a Poisson model with an intercept only using R function `glm()`.
(3) We save the resulting new MLE.

In either case, once we're done, we take the SD of the bootstrapped sampling distribution of the MLEs as the bootstrapped SE of the estimate, and we take the 2.5 and 97.5th percentile of the distribution as our bootstrapped 95% CI.

The following code produces a histogram of the bootstrapped sampling distribution of the MLE. We don't show this for both, but Fig. 2.9 shows this for the parametric bootstrap. To get the bootstrapped SEs and CIs, we simply summarize these two distributions by their sample SD and by a symmetric 95% percentile interval.

```
# Method 3.1 for SEs and CIs: Parametric bootstrap
# ------------------------------------------------
simrep <- 10000                                    # Number of bootstrap samples:
                                                   # a large number
estiPB <- array(NA, dim = c(simrep, 2))            # Array to hold estimates
colnames(estiPB) <- c('Small data set', 'Large data set')

for(i in 1:simrep){
  if(i %% 500 == 0) cat(paste("iter", i, "\n"))    # Counter
  yb1 <- rpois(10, lambda = MLE1)                  # Draw another small data set
  fm1 <- summary(glm(yb1~1, family = 'poisson'))   # re-fit the same model
  estiPB[i,1] <- exp(fm1$coef[1])                  # Save estimate on natural scale
  yb2 <- rpois(100, lambda = MLE2)                 # Draw another large data set
  fm2 <- summary(glm(yb2~1, family = 'poisson'))   # re-fit the same model
  estiPB[i,2] <- exp(fm2$coef[1])                  # Save estimates
}
```

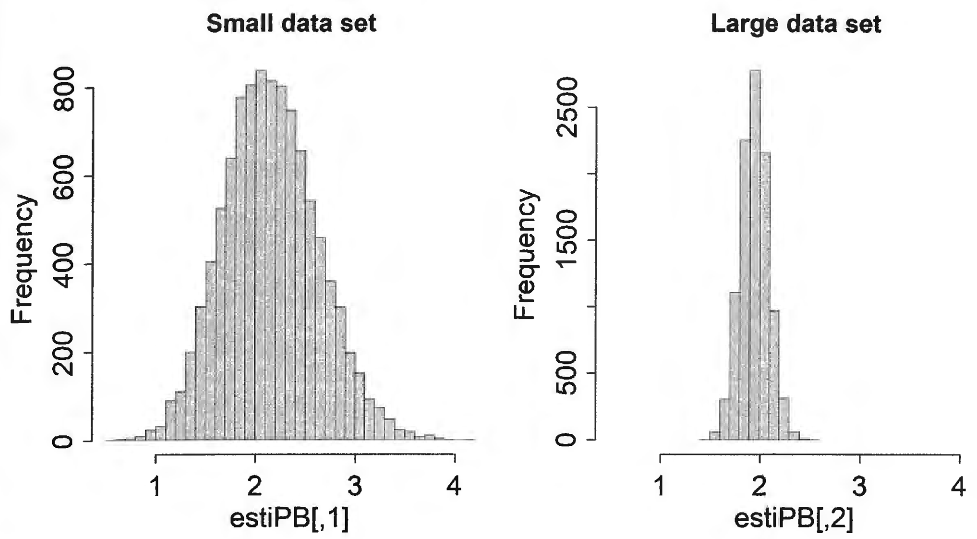

FIGURE 2.9

Two examples of a bootstrapped sampling distribution of the MLE of a model (this is for the parametric bootstrap); left ($n=10$), right ($n=100$).

```
# Parametrically bootstrapped sampling distribution of the MLE (Fig. 2.9)
par(mfrow = c(1, 2), mar = c(5,5,4,3), cex.lab = 1.5, cex.axis = 1.5, cex.main = 1.6)
xlim <- range(estiPB[,1])
hist(estiPB[,1], col = 'gray', main = 'Bootstrapped sampling distribution of MLE
  of lambda (small data set; parametric bootstrap)', xlim = xlim)
hist(estiPB[,2], col = 'gray', main = 'Bootstrapped sampling distribution of MLE
  of lambda (large data set; parametric bootstrap)', xlim = xlim)

# Parametric-bootstrapped standard errors
se1.pb <- sd(estiPB[,1])                            # Small data set
se2.pb <- sd(estiPB[,2])                            # Large data set

# Parametric-bootstrapped CIs
CI1.pb <- quantile(estiPB[,1], c(0.025, 0.975))     # Small data set
CI2.pb <- quantile(estiPB[,2], c(0.025, 0.975))     # Large data set

# Method 3.2 for SEs and CIs: Nonparametric bootstrap
# ------------------------------------------------
simrep <- 10000                                      # Number of bootstrap samples:
                                                     # a large number
estiNPB <- array(NA, dim = c(simrep, 2))             # Array to hold estimates
colnames(estiNPB) <- c('Small data set', 'Large data set')

for(i in 1:simrep){
  if(i %% 500 == 0) cat(paste("iter", i, "\n"))      # Counter
  yb1 <- sample(y1, 10, replace = TRUE)              # Re-sample a small data set
  fm1 <- summary(glm(yb1~1, family = 'poisson'))     # Re-fit model
  estiNPB[i,1] <- exp(fm1$coef[1])                   # Save estimate
  yb2 <- sample(y2, 100, replace = TRUE)             # Re-sample a large data set
  fm2 <- summary(glm(yb2~1, family = 'poisson'))     # Re-fit model
  estiNPB[i,2] <- exp(fm2$coef[1])                   # Save estimate
}
```

```
# Non-parametrically bootstrapped sampling distributions of the MLE
par(mfrow = c(1, 2), mar = c(5,5,4,3), cex.lab = 1.5,
  cex.axis = 1.5, cex.main = 1.6)                          # not shown
xlim <- range(estiNPB[,1])
hist(estiNPB[,1], col = 'gray', main = 'Bootstrapped sampling distribution of MLE
  of lambda (small data set; non-parametric bootstrap)', xlim = xlim)
hist(estiNPB[,2], col = 'gray', main = 'Bootstrapped sampling distribution of MLE
  of lambda (large data set; non-parametric bootstrap)', xlim = xlim)

# Nonparametric-bootstrapped standard errors
se1.npb <- sd(estiNPB[,1])                                 # Small data set
se2.npb <- sd(estiNPB[,2])                                 # Large data set

# Nonparametric-bootstrapped CIs
CI1.npb <- quantile(estiNPB[,1], c(0.025, 0.975))          # Small data set
CI2.npb <- quantile(estiNPB[,2], c(0.025, 0.975))          # Large data set
```

Once we're done, we compare the SEs and the 95% CIs using all methods in Sections 2.5.2–2.5.4.

```
# Standard errors
# ---------------
# Small data set: asymptotic normality and the two bootstraps
print(cbind(ASE1, 'PB-SE' = se1.pb, 'NPB-SE' = se1.npb), 4)

# Large data set: asymptotic normality and the two bootstraps
print(cbind(ASE2, 'PB-SE' = se2.pb, 'NPB-SE' = se2.npb), 4)

# 95% Confidence intervals
# ------------------------
# Quickly re-calculate LRT-based CI
CIp1 <- exp(confint(glm(y1 ~ 1, family = 'poisson')))
CIp2 <- exp(confint(glm(y2 ~ 1, family = 'poisson')))

# CIs for small data set: LRT-based, normal approximation, bootstraps
print(cbind(CIp1, CI1, CI1.pb, CI1.npb), 4)

# CIs for large data set: LRT-based, normal approximation, bootstraps
print(cbind(CIp2, CI2, CI2.pb, CI2.npb), 4)
```

```
> # Small data set: asymptotic normality and the two bootstraps
> print(cbind(ASE1, 'PB-SE' = se1.pb, 'NPB-SE' = se1.npb), 4)
        ASE1  PB-SE NPB-SE
lambda 0.469 0.4725 0.3926

> # Large data set: asymptotic normality and the two bootstraps
> print(cbind(ASE2, 'PB-SE' = se2.pb, 'NPB-SE' = se2.npb), 4)
         ASE2  PB-SE NPB-SE
lambda 0.1396 0.1398 0.1455

> # 95% Confidence intervals
> # ------------------------
> # CIs for small data set: LRT-based, normal approximation, bootstraps
> print(cbind(CIp1, CI1, CI1.pb, CI1.npb), 4)
       CIp1   CI1 CI1.pb CI1.npb
2.5%  1.404 1.281    1.3     1.4
97.5% 3.251 3.119    3.2     3.0
```

```
> # CIs for large data set: LRT-based, normal approximation, bootstraps
> print(cbind(CIp2, CI2, CI2.pb, CI2.npb), 4)
        CIp2    CI2 CI2.pb CI2.npb
2.5%  1.689  1.676   1.69    1.67
97.5% 2.237  2.224   2.23    2.24
```

Despite the very small size of the smaller sample, we see remarkable agreement between the uncertainty assessments, with the exception of those from the nonparametric bootstrap, for which the SE is smaller and the CI narrower for the smaller data set. This may illustrate a limitation of the nonparametric bootstrap when applied to very small sample sizes. When comparing the LRT-based CIs with the others, we find the former a little wider for the small data set, because it properly accounts for the asymmetry of the likelihood profile in that case. In general, we can't say that one type of bootstrap is always better than another. A parametric bootstrap may be preferable if the parametric model is adequate, but a nonparametric bootstrap may be more robust, exactly because it does not make any parametric assumptions to create replicate data sets. If you're interested to learn more about this powerful method, read up some in the vast literature on it (e.g., Efron, 1979; Efron & Tibshirani, 1993; Dixon, 2006; Manly, 2006).

2.5.5 A short summary on maximum likelihood estimation

This concludes our first exposition of that most widely used method of inference for parametric statistical models in classical or frequentist statistics: maximum likelihood estimation. The likelihood function is simply the joint density of the data when viewed as a function of the parameters. Both point estimation and uncertainty assessments are based directly on the likelihood function: the parameter values associated with the maximum of the likelihood function evaluated for the observed data set are taken as a point estimate (i.e., are the MLEs), and the shape and the curvature of the function around its maximum provides the information about the uncertainty around the estimates. The bootstrap offers a powerful alternative method of uncertainty assessment (Manly, 2006).

We have omitted the delta method (Dorfman, 1938) for variance estimation of functions of MLEs, including transformations of an MLE, or when forming predictions based on the linear predictor of a GLM (see Chapter 3). You may need this when you obtain the MLE for a transformed parameter, such as the log of the Poisson mean or the logit of a binomial probability, and then want to obtain the SE not on the log-scale, but on the natural scale of the parameter. You can read up on the delta method, also called "Taylor series expansion," at many places, including Powell (2007), Millar (2011): Chapter 4, ver Hoef & Boveng (2015), Cooch & White (2021): Appendix B. For an example application, see Chapter 19B. For the two specific transformations mentioned, see Kéry & Royle (2016: p. 35).

2.6 Maximum likelihood estimation in a two-parameter model

In practice you will rarely ever have a statistical model with just a single parameter. To illustrate a slightly more complex case, this time with 2 (!) parameters, we repeat some of the preceding analyses. Specifically, we will *model a parameter* with a covariate (Section 2.2.3), in the context of a Poisson model as in Section 2.5. In so doing, we get a bit ahead of ourselves, because we will develop a likelihood implementation of a Poisson log-linear model, or Poisson GLM. We will cover this type of model in depth in Chapters 3 and 11–14. For once in the book, we will use a real data set. We describe every step that underlies our approach, both conceptually and procedurally. This will be a re-cap of most of what we have covered so far in this chapter. You will find more extensive code for the analyses in this section on the book website https://www.elsevier.com/books-and-journals/book-companion/9780443137150.

We use a time series of counts of the Swiss population of Bee-eaters (Fig. 2.10) between 1990 and 2020 from Müller (2021); see Fig. 2.11. Bee-eaters live predominantly in Southern Europe and in Switzerland have benefitted greatly from climate warming over the last several decades, when their known population increased from 0 to 200 pairs. In our analysis, we will gloss over several potentially important issues in this data set. First, we ignore any possible sampling issues having to do with detection probability (see Section 11.2 and Chapters 19 and 19B). Second, there is strong evidence for overdispersion in these counts (see additional analyses on the website). And third, in Fig. 2.11 we see some intriguing patterns of temporal autocorrelation in the form of periodicities. Imperfect detection, when not taken account of, will lead to an underestimation of population size, while neglecting to accommodate overdispersion or serial correlation in a model typically leads to an underestimate of the uncertainty, that is, too small SEs and too narrow CIs. Here, we use these data for a simple procedural illustration of ML and ignore these issues. For more on overdispersion, see Chapter 12.

FIGURE 2.10

Two sensational Bee-eaters (*Merops apiaster;* photo by Dominique Delfino). This is one of the most colorful bird species in Europe.

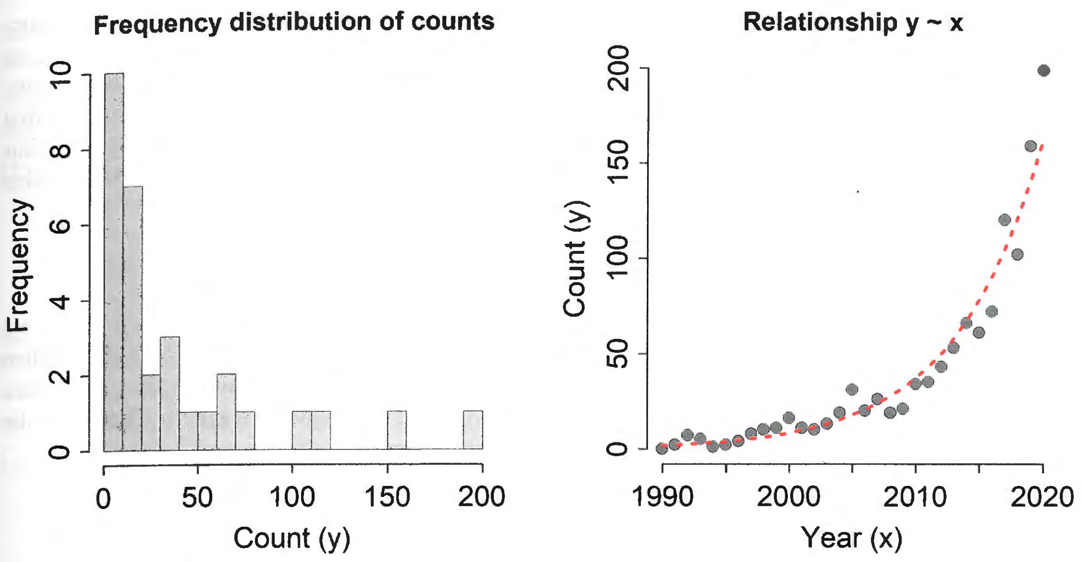

FIGURE 2.11

Two summaries of counts of Swiss Bee-eaters 1990–2020. Red curve on the right is the line of best fit from a Poisson GLM fit with `glm()` in R (Data from Müller, 2021).

```
# Swiss Bee-eater data set (national counts of pairs 1990-2020)
# -------------------------------------------------------------
# Counts of known pairs in the country 1990-2020
y <- c(0, 2, 7, 5, 1, 2, 4, 8, 10, 11, 16, 11, 10, 13, 19, 31, 20, 26, 19, 21, 34,
  35, 43, 53, 66, 61, 72, 120, 102, 159, 199)
year <- 1990:2020                        # Define year range
x <- (year-1989)                         # Scaled, but not centered, year covariate
x <- x-16                                # Now year covariate is also centered

# Plot bee-eater data (Fig. 2.11)
par(mfrow = c(1, 2), mar = c(5,5,5,2), cex.lab = 1.5, cex.axis = 1.5, cex.main = 1.5)
hist(y, xlab = 'Count (y)', ylab = 'Frequency', breaks = 20,
  col = 'gray', main = 'Frequency distribution of counts')
plot(year, y, xlab = 'Year (x)', ylab = 'Count (y)', frame = FALSE, cex = 1.5,
  pch = 16, col = 'gray20', main = 'Relationship y ~ x')
fm <- glm(y ~ x, family = poisson)      # Add Poisson GLM line of best fit
lines(year, predict(fm, type = 'response'), lwd = 3, col = 'red', lty = 3)
```

To use parametric statistical inference to learn about the Swiss bee-eater population trend based on our sample of noisy data from 31 years, we view these 31 numbers as outcomes of a stochastic process that we must describe. The counts are non-negative integers without any obvious ceiling. Among the probability distributions covered in this book (see Chapter 3), a Poisson distribution looks like the right choice. Thus, we work with the same basic model as in Section 2.5, but will now relax the assumption of a constant mean. Fig. 2.11 makes it clear that the counts cannot have been produced by a Poisson random process with the same mean λ. Instead, we want to fit a model where the counts, through their expectation, depend on the year x_i. That is, we will estimate a population trend, or growth rate, for Swiss bee-eaters during 1990–2020.

We will *model the Poisson parameter* λ, that is, we allow the Poisson mean to differ as a function of year x_i for every observation i, thus becoming λ_i. But we can't just assume $\lambda_i = \alpha + \beta x_i$ as in a simple linear regression, since this might get us into trouble with negative expected counts, which are of course impossible. Hence, we do what we often do when modeling a parameter that cannot be negative: we model it on the log-transformed scale (and we mean the natural log). Our model for Swiss bee-eaters can be described in algebra as follows, where $i = 1, 2..., 31$ is an index for year in count y_i and in the year covariate x_i:

$$y_i \sim Poisson(\lambda_i)$$

$$\log(\lambda_i) = \alpha + \beta x_i$$

Note that we could also write the second expression as $\lambda_i = e^{\alpha+\beta x_i}$ or as $\lambda_i = \exp(\alpha + \beta x_i)$. When defining the joint density of our data set, we replace λ_i by $e^{\alpha+\beta x_i}$. Thus, we can say that we replace the old parameter, the constant λ from Section 2.5, with two new ones, α and β. These are the intercept and the slope of a linear regression for the Poisson mean on the log scale.

Thus, we treat each count y_i as a realization from a Poisson RV with PMF:

$$P(Y = y_i | \lambda_i) = \lambda_i^{y_i} e^{-\lambda_i} / y_i!$$

Replacing λ_i by $e^{\alpha+\beta x_i}$, we get

$$P(Y = y_i | \alpha, \beta, x_i) = (e^{\alpha+\beta x_i})^{y_i} e^{-(e^{\alpha+\beta x_i})} / y_i!,$$

where on the left side we make explicit the dependency of the PMF on the two new parameters, α and β, and on the values of the year covariate x. Under our independence assumptions, the joint density of the data set is a product of 31 such terms, one for each pair of count y_i and year covariate x_i. Remember that the R function `dpois()` greatly simplifies our life because it saves us from having to type the explicit expression for the Poisson PMF.

In practice, we always work with the log-density when doing likelihood inference. For a single pair of y_i and x_i this yields

$$\log P(Y = y_i | \alpha, \beta, x_i) = y_i(\alpha + \beta x_i) - e^{\alpha+\beta x_i} - \log(y_i!).$$

The joint density and therefore the LL of the entire data set is a sum of 31 such terms. Taking the negative of the sum yields the NLL, which is what we minimize with `optim()`. In R, we get the LL by using function `dpois()`, setting the argument `log = TRUE`.

Next, we define the NLL function for our Poisson GLM as an R function that we minimize for the bee-eater data over the parameter space of α and β. We will also compare our do-it-yourself MLEs with estimates obtained by fitting the same model with the R function `glm()`. This uses a different algorithm, but for GLMs it yields the MLEs. It is always a good idea to double-check your solutions for more complex models or when you start to write your own code for fitting models.

When preparing the material in this section, we discovered that `optim()` is unable, at least with default settings, to find the global minimum of the NLL when working directly with the year covariate in the form of $x = \{1, 2, ..., 31\}$. The reason is that the LL of this model is almost flat over a large range of values of α near the maximum (you might want to try that out!). When working with continuous covariates it is often a good idea to transform them in some way since this reduces numerical challenges in likelihood maximization. Therefore, we will work with the centered version of the year covariate x, for which we did not encounter any problems (although we still select a non-default optimization algorithm; see below). Here's a summary of what we do:

(1) We start by defining an R function for the NLL for the Poisson GLM, where we will use R's density function for the log-density of the Poisson `dpois(..., log=TRUE)`. To emphasize once more that this is only a shorthand for an explicit algebraic expression which should not make us panic, we also write the log-density explicitly in comments on the function definition.

(2) Then, we will use that function in a brute-force, or grid, search for the maximum of the likelihood surface. It is instructive to plot the resulting likelihood surface, or landscape, in part because you can view a 2d likelihood surface as a conceptual model also for higher-dimensional likelihood "clouds" and later for joint posterior densities (see below).

(3) Finally, we call `optim()`, providing as arguments our new NLL function along with a vector of starting values and possibly other arguments, too. The function internally uses one of a number of optimization algorithms. The default is `method="Nelder-Mead,"` which often works well, but sometimes setting `method="BFGS"` works better; see the function's help file. We also set `hessian=TRUE`, which lets `optim()` report a numerical estimate of the Hessian matrix. Remember that the Hessian is the second partial derivative matrix of the LL function evaluated at the MLEs, and that the negative of the Hessian matrix is called the observed Fisher information matrix. However, since we work with the NLL function, what `optim()` reports as "the Hessian" is in fact directly the information matrix. Inverting it yields our usual estimate of the VC matrix of the estimated parameter vector. Taking square roots of the diagonal gives us SEs of the estimates, and the MLE plus/minus 1.96 times these SEs provides a Wald-type 95% CI.

For comparison, we will fit the same model with `glm()` and also produce 95% LRT-based CIs with function `confint()`, which for a fitted GLM uses profile likelihood analogous to what we did in Section 2.5.2. On the website you will find further code for bootstrapping (parametric and nonparametric) both the SEs and CIs.

We start by defining the NLL function for the Poisson log-linear model and organize the parameters in a vector which we call `param`. Then, we calculate the expected abundance as a function of a log-linear model and use it for evaluating the LL of every single observation in data vector `y`; this is stored in vector `LLi` on executing the function. Taking the sum and negating the result yields the NLL for the entire data set.

```
# Define an R function for the NLL of a Poisson log-linear model
NLL <- function(param, y, x) {
  alpha <- param[1]                          # Intercept
  beta <- param[2]                           # Slope
  lambda <- exp(alpha + beta * x)
  LLi <- dpois(y, lambda, log=TRUE)          # log-likelihood for an observation
# LLi <- y*log(lambda)-lambda-lfactorial(y)  # Same 'by hand' !
  LL <- -sum(LLi)                            # NLL for all observations in vector y
  return(LL)
}
```

```
# Try out, evaluate a few candidates for alpha, beta (lower is better)
NLL(c(0, 0), y, x)
NLL(c(0, 1), y, x)
NLL(c(1, 0), y, x)
```

```
> NLL(c(0, 0), y, x)
[1] 3930.635
> NLL(c(0, 1), y, x)
[1] 5164763
> NLL(c(1, 0), y, x)
[1] 2803.901
```

We see that among the three pairs of trial values, assuming `alpha = 1` and `beta = 0` comes closest to the MLEs that we're looking for. We continue this same approach of simply trying out the NLL function for different pairs of values for `alpha` and `beta`, and conduct a systematic search across a regular grid. Our choice in the next code box was informed by an earlier search over a much wider range of possible values for the two parameters. Thus, here we are "zooming in" into that part of the likelihood landscape where we know the maximum is located.

```
# Brute-force search for the MLEs in a grid
nside <- 1000                                  # Grid resolution governs
                                               # quality of approximation
try.alpha <- seq(2.8, 2.96, length.out = nside) # intercept
try.beta <- seq(0.135, 0.16, length.out = nside) # slope
nll.grid <- array(NA, dim = c(nside, nside))

# Evaluate NLL over the grid: try 1 Million of (alpha, beta) pairs
for(i in 1:nside){
  for(j in 1:nside){
    nll.grid[i,j] <- NLL(c(try.alpha[i], try.beta[j]), y, x)
  }
}

# Get pair of values associated with minimum function value
(best <- which(nll.grid == min(nll.grid)))
(best.point <- cbind(rep(try.alpha, nside), rep(try.beta, each = nside))[best,])

[1] 2.8868068 0.1472372                        # Best pair of values for (alpha, beta)

# Plot the likelihood surface, or likelihood landscape (Fig. 2.12)
par(mar = c(6,6,5,3), cex.lab = 1.5, cex.axis = 1.5, cex.main = 1.5)
mapPalette <- colorRampPalette(c("gray", "yellow", "orange", "red"))
image(x = try.alpha, y = try.beta, z = nll.grid, col = mapPalette(100),
    axes = FALSE, xlab = "Intercept (alpha)", ylab = "Slope (beta)")
contour(x = try.alpha, y = try.beta, z = nll.grid, add = TRUE, lwd = 1.5, labcex = 1.5)
axis(1, at = seq(min(try.alpha), max(try.alpha), by = 0.05))
axis(2, at = seq(min(try.beta), max(try.beta), by = 0.005))
box()
points(best.point[1], best.point[2],
  col = 'black', pch = 'X', cex = 1.5)         # Best point from our grid search
```

We identify the minimum over the grid as `alpha = 2.8868` and `beta = 0.1472`, respectively. The map of the likelihood surface in Fig. 2.12 shows that the two estimates are negatively

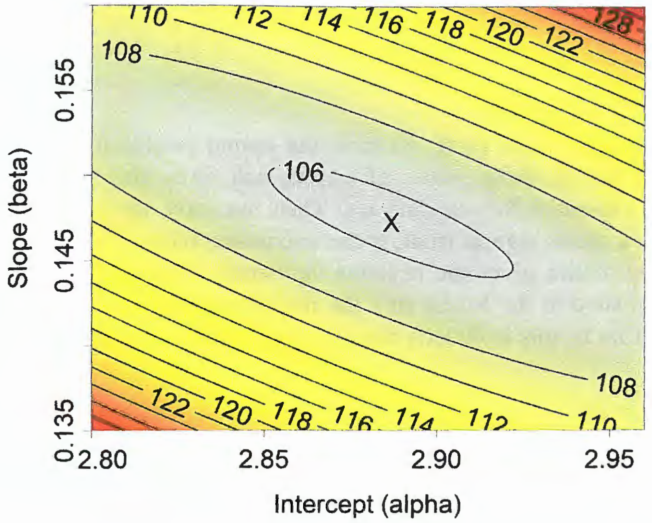

FIGURE 2.12

(Negative log-) Likelihood surface for the two parameters of a Poisson log-linear regression fit to the Swiss bee-eater data obtained by evaluating the NLL function over a grid with a total of 1 million pairs of possible values of the intercept and the slope. The minimum is marked with an X. These are the MLEs of the two parameters as obtained by this fairly crude method (compare with Fig. 2.17 for Bayesian inference for the same model).

correlated, since there is a ridge going from north-west to south-east. Moreover, the curvature around the maximum is greater in direction of the y-axis than in that of the x-axis. This indicates greater absolute precision of the estimate of the slope than of the intercept.

Finally, we use `optim()` to minimize the NLL function for our data set over the parameter space. This is the customary method with which we produce what we call the "DIY-MLEs" (see Section 4.7). We choose `method="BFGS,"` since the default choice in `optim()` does not quite find the true MLEs.

```
# Minimize NLL and get VC matrix by inverting the Hessian
inits <- c(alpha = 0, beta = 0)           # Inits
(out <- optim(inits, NLL, y = y, x = x, hessian = TRUE, method = 'BFGS'))
MLE <- out$par                            # Grab MLEs
(VC <- solve(out$hessian))                # Get variance-covariance matrix
ASE <- sqrt(diag(VC))                     # Extract asymptotic SEs
LCL <- MLE - 1.96 * ASE                   # Lower limit of Wald-type CI
UCL <- MLE + 1.96 * ASE                   # Upper limit of Wald-type CI
print(cbind(MLE, ASE, LCL, UCL), 4)       # Print MLEs, SEs and CI
```

```
# Compare with function glm() and get 95% profile CIs
fm <- glm(y~x, family = 'poisson')
summary(fm)
confint(fm)
```

There is a lot of output here. First, we have the output produced by function `optim()`. We have the MLEs first, that is, those values of `alpha` and `beta` that lead to the highest function value of NLL when evaluated for our data set. Then, we have the minimum value of the NLL, which, when you put a minus sign in front, is the maximum value of the LL for the data set. What is labeled the Hessian matrix gives the negative curvature of the LL function with respect to the parameters when evaluated at the MLEs (it's the negative curvature since we work with the NLL rather than the LL). That is, this is directly the observed information matrix.

```
$par
    alpha       beta
2.8868266  0.1472298

$value
[1] 105.774

$counts
function gradient
      62       13

$convergence
[1] 0

$message
NULL

$hessian
            alpha      beta
alpha   1180.003  10646.29
beta   10646.288 138329.53
```

This last table just above gives the observed Fisher information matrix. Notice how the information is much larger in absolute terms for `beta` than for `alpha`, something that we saw already in Fig. 2.12. Inverting the information matrix yields the VC matrix of the estimates. The off-diagonal is negative, illustrating what we saw in Fig. 2.12: the two estimates are negatively correlated.

```
> (VC <- solve(out$hessian))   # Get variance-covariance matrix
            alpha             beta
alpha 0.002772921   -2.134130e-04
beta -0.000213413    2.365407e-05
```

And here is a summary of our DIY ML inference, showing the MLEs, the asymptotic SEs, and the lower and upper limits of a Wald-type 95% CI for both parameters.

```
         MLE        ASE     LCL     UCL
alpha 2.8868   0.052659  2.7836  2.9900
beta  0.1472   0.004864  0.1377  0.1568
```

Comparing these estimates with those from function `glm()` we find an excellent agreement. The normality assumption of the MLEs, on which the Wald-type CIs rely upon, appears to be fairly adequate here, when judged by a comparison with the profile intervals produced by `confint()`.

```
Coefficients:
             Estimate  Std. Error  z value  Pr(>|z|)
(Intercept)  2.886821   0.052659    54.82    <2e-16 ***
x            0.147230   0.004864    30.27    <2e-16 ***

> confint(fm)
                 2.5%      97.5%
(Intercept) 2.7814779 2.9879605
x           0.1378238 0.1568931
```

This concludes our demonstration of what is probably the most common way of conducting statistical inferences for a parametric statistical model, using the ML method. We find the MLEs by numerical minimization of the NLL and invert the Hessian matrix to obtain an estimate of the VC matrix. This yields asymptotic SEs, which enable us to construct Wald-type CIs. Throughout the book, we use this method to produce our "DIY-MLEs."

Ah, and before we forget, we have also learned something biological in this statistical exercise. We estimate that over the last 31 years, the Swiss bee-eater population has grown every year by a factor of `exp(0.1472)=1.169`, that is, by about 17% annually. Compounded over the 30 interannual intervals, this corresponds to a total population growth of `exp(0.1472)^30=82.76`. That is, the population has multiplied by a factor of 83 during 1990–2020. A smooth representation of the trajectory of the population is shown by the red line in Fig. 2.11.

2.7 Bayesian inference using posterior distributions

Next, we discover the Bayesian way of learning from a data set via a statistical model. Conceptually, the start of any Bayesian analysis is exactly the same as in a frequentist analysis with ML: a parametric statistical model. That means the data we want to analyze are viewed as the outcome of a stochastic process, and we describe this data-generating process in a manner that is adequate for the aims of our analysis, for example, description, prediction, or mechanistic understanding (Tredennick et al., 2021). We use probability to describe the data-generating process. Hence, we treat all observed data, and for hierarchical models (see Chapter 3) also all unobserved variables, as RVs endowed with a probability distribution governed by a small number of unknown constants called parameters. This is our parametric statistical model wherein every RV is fully described by a probability distribution that we claim to know except for its parameters. We can do a number of things with the model, such as estimating unknown inputs (missing covariates) or unknown outputs (making predictions). Most importantly, when we combine a data set with a model, we can estimate parameters and assess the uncertainty associated with these estimates. It is only from this last point on that Bayesian inference differs fundamentally from frequentist statistical inference.

2.7.1 Probability as a general measure of uncertainty, or of imperfect knowledge

Bayesians divide the world into things that are known and things that are at least partially unknown and therefore uncertain (Hooten & Hefley, 2019). The defining feature of Bayesian inference is the use of probability as a measure of knowledge about these uncertain things. In Bayesian inference, as with inference by ML, we do use probability as a measure of variability in a stochastic process, that is, we use probability distributions to characterize RVs and to build our statistical models. But in addition, probability is used in a more general way to quantify the amount of knowledge about anything that is not perfectly known to us. Spiegelhalter (2019) calls these two sources of uncertainty 'stochastic' and 'epistemic', both of which are quantified by probability when using Bayesian inference. Often, the Bayesian interpretation of probability is described as a *degree of belief probability*. This is perhaps a little unfortunate, since "belief" has a connotation of being subjective and therefore perhaps unscientific. A better description of the Bayesian usage of probability is arguably that it is a *degree of personal knowledge* or *a measure of uncertainty*.

Lindley (2006) stresses the fact that under this view, probability becomes personal, since what is well known to one person may be partly or wholly unknown to another. For instance, as a conservation biologist you may wonder whether the asp viper (*Vipera aspis*) still occurs at some site where it was known to occur 30 years ago. This is uncertain to you, and therefore as a Bayesian you will use a probability distribution to quantify your incomplete knowledge about the current presence or absence of the viper. But imagine another biologist who happens to have recently seen an asp viper at that site, so to him there is no more uncertainty. That the probability of an event, when probability is used as a measure of knowledge, may be different for one person than for another does not mean that such use of probability is unscientific.

The world is full with things that we don't know for certain. In fact, if you think about it, there are very few things that you know for certain (see Section 2.1). Unknown things include the following:

- In conservation, whether the asp viper still occurs at a site it was earlier known to occur;
- in a survival analysis, the fate and body mass of an individual during an occasion when it is not observed or caught;
- in a SDM exercise, presence/absence or abundance of a study species at an unsurveyed site;
- in another SDM application, the number of grid cells occupied by a study species in a region;
- and in a population viability analysis (PVA), the fate (extant vs. extinct) and the size of a population some time into the future.

In the context of modeling, the value of a parameter is an outstanding example of something we are uncertain about. Remember that in our statistical models we claim to be at once omniscient and completely ignorant: we know exactly the type of the data-generating process, for example, we claim that our bee-eater counts were produced exactly by a random process known as a Poisson distribution. But we feign complete ignorance about the value of the unknown constants in this process, that is, about parameter λ or, more specifically, α and β that replace λ in the Poisson GLM above.

When we conduct a Bayesian analysis, we use probability as a measure of our knowledge about anything that is not known for certain, by placing a probability distribution on it. Hence, we deal with our uncertainty about unknown things in the same way as with the variability of observable things such as our data. As a consequence, we treat parameters and other unknown quantities *as if* they were RVs. This does not mean that *a priori* the value of a parameter may not be a constant also to a Bayesian. In fact most parameters in a Bayesian analysis are fundamentally identical to parameters in a frequentist analysis: they are simply unknown constants and we are 'epistemically uncertain' about their values (Spiegelhalter, 2019). It is only due to our use of probability as a measure of uncertainty that they are treated as RVs, but their true values are actually fixed. Link & Barker (2010) cite the example of the millionth digit of the number π, which is an unknown constant to most of us, but which we would treat as a RV in a Bayesian analysis that aimed at guessing it.

In the context of our examples above, this means:

- In the asp viper example, if we denote as A the event that there still are snakes at the site 30 years after the last known sighting, then we can write $p(A)$ as the probability that there is still at least one snake living there.
- In the survival analysis, for an individual i that is not detected during occasion t, we will place a probability distribution on both its fate (which is binary: dead or alive) and on its body mass (which is continuous), for example, $p(z_{i,t} = 1)$ and $p(x_{i,t})$ where z is the alive state and x is mass.
- In the two SDM examples, we place probability distributions on the unknown presence/absence or on the abundance of the modeled species at an unsurveyed site, and on the unknown number of occupied grid cells in the region.
- Similarly, in a PVA we place a probability distribution on the fate or the size of our study population at every time step up to our prediction horizon (Petchey et al., 2015).

In the context of our second SDM example, Fig. 2.13 summarizes the use of probability as a general measure of uncertainty. A probability density enables you to describe both complete lack of knowledge (that's the blue horizontal line) and complete certainty (that's the red vertical line), as well as anything in between (e.g., the gray curves). When we learn, our probability distribution for the value of some parameter will become more and more pointed. Thus, how wide or how narrow

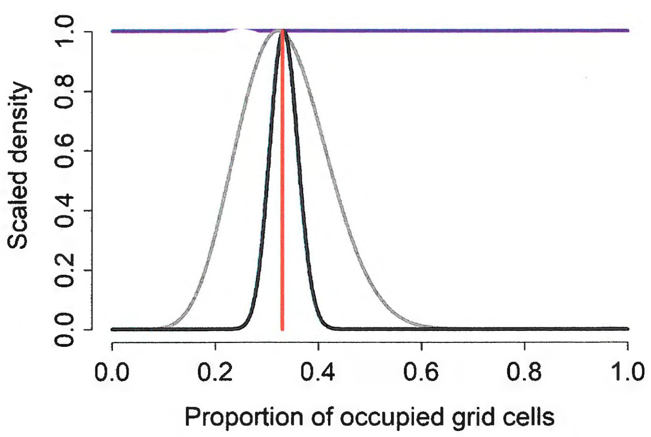

FIGURE 2.13

The use of probability as a measure of knowledge about uncertain things, in the context of a hypothetical study that asks about the proportion of grid cells in a region that are occupied by some species. With probability we can describe any state of knowledge ranging from ignorance (this is the blackish-blue horizontal line) to certainty (red) as well as any state of knowledge in between (gray). The amount of our information is represented by how narrow the probability density is (note densities are scaled for ease of comparison).

our probability distribution is quantifies the amount of our knowledge about a parameter. Again, this use of probability as a measure of uncertainty is *the* defining feature of Bayesian inference. Interestingly, this coincides with how we humans like to express uncertainty, when we say things such as "*I am 99% certain that tonight it's going to rain,*" or "*I am 100% certain that this or that is going to happen.*" The only difference is that Bayesians do calculations with such probability assessments.

2.7.2 The posterior distribution as a fundamental target of inference

In an inference problem with a data set y and a model with unknown parameter(s) θ, a Bayesian would argue that the most sensible thing to do is to estimate θ based on the information in the data using the conditional probability expression $p(\theta|y)$. That is, to go after the probability of what is unknown (i.e., the value of parameter θ) given what is known (i.e., data y). Hence, even though the data are considered as the outcome of a random process, once we have observed them, they are perfectly known and there is no longer any uncertainty about them. So how should this conditional probability of the parameters given the data be computed?

That's where Bayes rule comes in, which for two events A and B can be written as

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}. \tag{2.7}$$

Bayes rule (which for aesthetic reasons we prefer to write without an apostrophe) states that the probability of A, given that B is true (or has been observed), is the product of the probability that B is true, given that A is true, times the probability that A is true, divided by the probability that B is true. This is the same as $p(A|B) = p(A,B)/p(B)$, that is, the joint probability of A and B (or that both A and B are true), divided by the probability of B.

Bayes rule is a mathematical fact that can easily be derived from the definition of conditional probability (Blitzstein & Hwang, 2019; Pishro-Nik, 2014), which itself is given by a re-arrangement of the multiplication rule (Section 2.1). There is nothing intrinsically Bayesian about it and the rule has many non-Bayesian applications. These include the assessment of how likely you have some disease when you receive a positive test result, and the estimation of the realized values of random effects in a non-Bayesian analysis, such as the site-specific population size in an N-mixture model (Kéry & Royle, 2016: chapter 6) and the presence or absence at a site where a species was not detected in an occupancy model (Kéry & Royle, 2016: chapter 10). Indeed, Eq. 2.7 is the general statistical expression by which we update knowledge in the light of new information (Blitzstein & Hwang, 2019).

The English minister and mathematician Thomas Bayes (1702–1761) and others such as Pierre-Simon Laplace (1749–1827) used what later became to be known as Bayes rule for inference about unobservable things, such as parameters in statistical models. In this setting, Bayes rule can be written as follows:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} = \frac{p(y,\theta)}{p(y)}. \tag{2.8}$$

The four components of Bayes rule in the context of Bayesian statistical inference are as follows:

- the posterior distribution $p(\theta|y)$ is the probability (or probability density) of the parameters given the data and the prior,
- the likelihood $p(y|\theta)$ is the probability (or probability density) of the data (i.e., a PMF or a PDF) viewed as a function of the parameters,

- the prior distribution $p(\theta)$,
- the marginal distribution of the data $p(y) = \int p(y|\theta)p(\theta)d\theta$ is the integral of the numerator over the parameter(s) θ, that is, what we get when we average the numerator over all possible values of parameter θ, weighted by its plausibility given in $p(\theta)$.

Note that the posterior distribution is also given by the ratio of the joint distribution of the data and the parameters, and the marginal distribution of the data.

You can see that hallmark of Bayesian inference, the use of probability as a measure for uncertain knowledge, at two places in Bayes rule: in the prior and in the posterior. Because of this we can make formal probability expressions about specific values of parameters in a model. For instance, in a PVA we may rightfully say things like "*we are 95% certain that the rate of decline of the population is steeper than 20% per year.*" Such expressions are impossible in non-Bayesian ways of learning from data, or at least cannot be obtained in such a straightforward way.

The formal steps underlying every Bayesian analysis are the following:

- We use probability as a measure of uncertainty about unknown quantities such as parameter θ.
- We treat all statistical inference (e.g., parameter estimation, testing, imputation of missing values, or predictions) as a probability calculation by application of Bayes rule.
- We start by expressing our initial, or prior, knowledge about parameter θ by one probability distribution, which is the prior $p(\theta)$. Importantly, no information from the data must be used to form the prior; this would be using the data twice and amount to cheating.
- Then, we use Bayes rule to update our prior knowledge with the information contained in the data and embodied by the likelihood function $p(y|\theta)$.
- The result is another probability distribution, the posterior distribution $p(\theta|y)$, which represents our new state of knowledge about θ.
- All these steps can be repeated and Bayes rule is a natural way for updating knowledge in a sequential manner.

You see that the likelihood is a central piece also of Bayesian inference, since it is the likelihood (via the data y) which transforms our prior knowledge into our posterior knowledge about parameter θ. However, to do straight ML inference, we use *only* the likelihood function along with the data and optimize it over the parameter space to find the MLEs. In contrast, in Bayesian inference our target is always a distribution for every unknown.

There is a lot of heuristic appeal to Bayes rule as a method for learning about uncertain things. First, it is based on a "human concept" of probability, that is, we formally use probability as a measure of knowledge similar to what we humans like to do in everyday language. Second, Bayes rule is often described as $p(\theta|y) \propto p(y|\theta)p(\theta)$, that is, the posterior is proportional to the product of the likelihood and the prior. And this is exactly how we humans learn: when faced with a new piece of information we hardly ever consider it in isolation. Rather, we almost always consider it in the context of what we know already and then base our conclusion or decision on a combination of both the new and the old information. Thus, the new information updates our old information, and the result is a new and (hopefully) improved state of our knowledge.

Bird identification provides a good illustration of the Bayesian nature of human thinking. When we see a bird and are unsure about its identity, our final conclusion may be "it's species A" or "it's very probably species B," or perhaps "it's more likely to be species C than species A." In coming to this conclusion, in addition to the features noted in the field or visible on a photograph that we took, we will *always* also use external (i.e., *prior*) knowledge about how likely a species is to occur at the given place and time. For instance, when seeing a large soaring raptor in Europe it is exceedingly unlikely that this is a condor, since the two condor species are endemic to the Americas. Likewise, it will be highly unlikely to see a neotropical long-distance migrant in the Canadian arctic during winter, when such a species is expected to be in its winter quarters in Latin America.

Also, when we learn according to Bayes rule, it is clear that every scientific position or every opinion, as embodied by the prior, can be modified by the arrival of new information. Therefore, in science we should avoid 0/1 priors, which place a probability of 0 (representing impossibility) or 1 (representing absolute certainty) on some event, or on the specific value of a quantitative hypothesis (Lindley, 2006). This would be the end of learning and would go counter that defining activity of science, which is to test our theories (loosely represented by the prior distribution in Bayes rule) with empirical data, which we can similarly loosely imagine as equivalent to the likelihood. The confrontation between thoughts about how the world might be, and observations about how it actually is, will up- or downweigh our trust in our theories and thus lend them more or less credibility. Thus, over time, this procedure lets us eliminate less apt explanations of the world and keep the better ones instead.

When estimating parameters we are used to having some number as a point estimate (i.e., as our best guess of what the parameter value is) and then one or two other numbers that quantify the likely margin of error of this estimate, see Sections 2.5–2.6. How can we get these when the result of our analysis is an entire (posterior) probability distribution? The answer is that we can simply use any measure of location and spread, as in the most basic descriptive statistics. As a point estimate we can use the mean, median, or mode of the posterior distribution, and as an uncertainty assessment we may use the SD or a range given by some percentile, such as 95 or 99 ... or even 89 if you *really* want to insist on being different from the rest.

2.7.3 Prior distributions

In any Bayesian analysis we must specify prior distributions for all unknown parameters. Historically, the use of prior information in statistical analyses has been controversial. Arguments in favor of the use of priors focus on their value as a formal way of incorporating information from outside your study. You can think of the priors almost as a sort of plug with a sign attached to it saying "*If you have external information and want to use it in your estimation then put it in here.*" This looks like a good thing, because often you may have quite good contextual or even quantitative knowledge about a parameter. Bayes rule makes it relatively straightforward to combine this information with the information in your data set. For instance, in the examples of uncertain things in Section 2.7.1, we may perhaps think that the proportion of recently visited historic asp viper sites in the same region where the viper was still found to survive represents a sensible expectation also for the site in question. In a survival analysis, available estimates for similarly sized species may provide a reasonable expectation for our study species (McCarthy & Masters, 2005). In general, results from "similar studies nearby in space, time or taxonomy" may provide sensible *a priori* information about the likely value of a parameter. Bayes rule then lets us use that information as long as we can express it as a probability distribution.

Priors that are meant to bring into estimation some amount of external information are called *informative*. Arguably their use makes a lot of sense, since you can imagine them as if they were additional data. In any estimation task it wouldn't be sensible to ignore part of the available data. Hence, it does not make sense to ignore *a priori* information when available. There can be major benefits in the use of informative priors. Exactly as additional data will make your posterior distributions narrower (i.e., increase your knowledge about a parameter), so will informative priors tend to increase your knowledge about a parameter. You will then get smaller posterior SDs and narrower Bayesian credible intervals. In addition, you may be able to estimate additional parameters which may not be estimable without the use of informative priors.

On the other hand, those who resist the formal use of prior information (and by extension, Bayesian inference) are motivated by the belief that Bayesian inference is subjective, since priors must be chosen actively and are not suggested by seemingly objective considerations. This appears to contradict the tenet that science must be objective and ought to minimize personal influences by the experimenter or analyst. While it is probably true that we ought to try and make science as objective as possible, we feel that the issue of the subjectivity of Bayesian inference has often been oversold dramatically: the choice of prior is only one out of literally dozens or even hundreds of arbitrary choices that we make when we study a scientific question. For instance, when we are interested in the effects of density dependence on population dynamics, which species should we choose to work with? A plant or an animal? Should we study it in the field, in a cage or in the greenhouse? In North America or in Australia? During which set of years? And on and on and on. Every single one of our subjective choices to these questions will be guaranteed to influence our results to some degree, but nobody seems to get nervous about that. When we use Bayesian inference, we just add one more such choice to our study. We really don't think that this jeopardizes the scientific value of our study.

However, it is true that in a Bayesian analysis the priors always affect our estimates at least to some small degree; there is no escaping that. It is true that most of the time, the influence of the prior will be small unless we use informative priors, and unless the amount of information in the data is really small. But this is not always so. In addition, having to specify priors does induce an additional ambiguity, since one prior may lead to one result, while another may lead to a slightly different result. This is of course not wrong, but it may make reporting a study more complex, because we might have to report two or even more different sets of results, one for each prior.

For better or worse, what people do in the vast majority of applied Bayesian analyses is to use so-called vague priors, which are meant to minimize their influence on the posteriors. That is, we choose low-information priors which are almost uniform over the range of parameter values where the likelihood has non-negligible support. Vague priors give similar *a priori* weight to a large range of plausible parameter values. For instance, we might specify a uniform(0, 1) prior for the probability that a historic asp viper location is still occupied, as we might do also for any parameter representing a probability, such as survival or occupancy. In practice, we often use a uniform prior within some suitable range or a "flat normal" distribution, that is, a normal distribution with a large variance chosen again so that the *a priori* probability of parameter values are approximately similar over a wide relevant range. For another view, wherein the choice of weakly or moderately informative priors merits much more thought, see Gelman et al. (2020).

Another issue with Bayesian inference that we think is often oversold is the lack of transformation invariance with respect to the choice of prior (Royle & Dorazio, 2008). For instance, in a Poisson GLM we will typically model the mean on the log-transformed scale. In an analysis with vague priors, should we then put a uniform on log(lambda) or on lambda? This choice will have some effect on the resulting posterior distribution, though most times it will be minor and negligible. Another famous example is the choice of a uniform prior on the logit-scale intercept of a logistic regression (see Chapter 3 and 15–17). This implies a very informative prior for the intercept on the probability scale, with far higher mass placed on values near 0 and 1 (Link & Barker, 2010; Northrup & Gerber, 2018). In this book as in most of our research, we follow Royle & Dorazio (2008) and specify vague priors on a "natural scale," which is one we like to think about a parameter. That would be lambda in a Poisson GLM and the probability scale in a binomial GLM, because most of us don't like to think in terms of the log number of bee-eaters or logit probabilities.

With vague priors, we will find that the Bayesian estimates (both point and uncertainty) are usually extremely similar numerically to the the same estimates using ML for a model, unless sample sizes are really small. We find this comforting and see it as a strong argument in favor of an opportunistic use of ML and Bayes. Technically, it is the mode of the posterior distribution (sometimes called the maximum *a posteriori*, or MAP, estimate) in an analysis with vague priors which corresponds to the MLE. However, with approximately symmetric posteriors, the mode, median, and mean all co-incide. In practice, the posterior mean is used most often as a Bayesian point estimate, along with the posterior SD as a Bayesian analog of the SE and the 95% percentiles of the posterior which give a 95% Bayesian credible interval. This is called a credible interval, or CRI, to distinguish it from the frequentist CI. Note though that for more complex models (specifically those with parameters that have posteriors with non-Gaussian shapes, for example, one-tailed distributions), the median is often a preferred point estimate, since it is better than the mean and mode at representing the middle of the distribution (Jeff Doser, pers. comm.).

Interestingly, the use of prior information in learning from data is not unique to Bayesian inference. For instance, penalized likelihood (Lele et al., 2012; Moreno & Lele 2010; see also Chapter 18) is a conceptually equivalent frequentist estimation method in which we assume that some values of a parameter are *a priori* less likely than others. Nevertheless, prior distributions are a salient feature of Bayesian inference because we can't do Bayesian inference without them. And as a final remark we note that in spite of our preference for vague priors, we often use what are also called weakly informative priors. Such priors will be a bit more informative than very vague ones, but still produce estimates that numerically strongly resemble estimates obtained without use of prior information (e.g., MLEs). But weakly informative priors often produce Markov chains with much better convergence and mixing (see the next section).

2.8 Bayesian computation by Markov chain Monte Carlo (MCMC)

When we're doing Bayesian inference we're always after the posterior distribution of the unknowns in our statistical models, especially the posterior distributions of the parameters, $p(\theta|y)$. How can we obtain these posterior distributions? For centuries, this was the Achilles' heel of Bayesian inference. Mathematically, Bayes rule can only be evaluated in exceptional cases because the required integrals over every parameter in the model are usually intractable (Brooks, 2003; Hobbs & Hooten, 2015; Plummer, 2023b). Thus, Bayesian inference needed to either rely on often restrictive approximations or (in special cases) on the use of so-called conjugate priors. These priors are of a similar "form" as the likelihood and this combination leads to posterior distributions of a known form. The famous textbook example is the combination of a binomial likelihood with a beta prior when observing r successes and

n failures for a Bernoulli RV such as 'presence' or 'absence' in a SDM context. Use of a beta(a,b) prior leads to a posterior that is also a beta distribution with parameters $a+r$ and $b+n$. But such "tricks" were never sufficient to make Bayesian inference a general-purpose inference tool for widespread use.

This changed dramatically in the early 1990, when statisticians re-discovered work conducted by physicists in the 1950s (Metropolis & Ulam, 1949; Metropolis et al., 1953). This work showed how one can develop simulation algorithms to explore the shape of unknown distributions (Geman & Geman, 1984; Tanner & Wong, 1987; Gelfand & Smith, 1990), for example, of posterior distributions that could not be derived analytically. The rediscovery and subsequent further development of what is now called Markov chain Monte Carlo (MCMC) algorithms has caused a statistical revolution since then (Brooks, 2003). MCMC algorithms coupled with greatly increased computing power have been primarily responsible for the wide adoption of Bayesian inference both in the statistical sciences and, with the advent of user-friendly software such as WinBUGS and OpenBUGS (Lunn et al., 2013), also in ecology. At least conceptually, these ground-breaking software engines represent the precursors and inspiration for all three Bayesian inference engines covered in this book: JAGS, NIMBLE, and Stan (Chapter 4).

For several reasons, we would not advocate learning how to code up your own MCMC algorithms to everyone in ecology. First, the learning curve for these methods for complex models may be too steep for many. Second, debugging your custom MCMC software would take a lot of time which might be better spent elsewhere. And third, the available general-purpose engines such as JAGS, NIMBLE, and Stan are incredibly powerful and let an expert statistician do virtually everything that he or she might be able to achieve with custom-written MCMC code.

Nevertheless, it is important to develop an intuition about what goes on under the hood when running such engines because it may help you diagnose causes for problems, interpret MCMC output, and simply be intellectually satisfying. Finally, some of you may come into the situation where you have to use some custom MCMC code that somebody else wrote. And a small proportion of you may need and want to learn how to write their own MCMC algorithms in R or some other statistical programming language. For these reasons we want to give you a flavor of "how MCMC works." We will show one of the simplest possible variants of MCMC: a random-walk Metropolis algorithm. We believe that even just understanding this simple prototypical MCMC algorithm will help you achieve some of the aims just formulated. Read Hooten & Hefley (2019) and Zhao (2024) when you want to learn much more about MCMC in the context of models used in ecology. Chapter 17 in Royle et al. (2014) is a neat introduction for ecologists to several classes of MCMC algorithms.

To help understand what you do when you use JAGS, NIMBLE, or Stan, we focus on two topics: Monte Carlo integration and a simple MCMC algorithm for implementation of the Poisson log-linear regression model for the Swiss bee-eater data from Section 2.6. In the former, we use sample statistics computed from random numbers drawn from a distribution to learn about features of that distribution. An MCMC algorithm can be viewed as a highly customized random number generator (RNG) for the posterior distributions of the parameters in our models using Bayesian inference.

2.8.1 Monte Carlo integration

An MCMC algorithm produces random numbers from a posterior distribution. Once we have a large amount of these random numbers, for example, some 100s or 1000s, we can describe them to learn about the features of the posterior distribution. Drawing a larger number of values will provide a better approximation. For instance, we can take the sample mean as an approximation of the mean of the distribution and the sample SD as an approximation of its SD. Technically, in these and other cases we use simulation to solve an integral. For instance, for the mean of the distribution, we use simulation to solve the integral $E(x) = \int p(x)xdx$.

To see how a sample of random numbers from a distribution can be used to learn about the mean and the SD or other features of that distribution, consider the following artificial case. We use the Gaussian RNG `rnorm()` and draw an increasing number of values from a normal distribution with known parameters for the mean and the SD. Then we see how well we can estimate these two parameters from the samples of values produced.

```
# Illustration of the concept of Monte Carlo integration (Fig. 2.14)
set.seed(2)
par(mfrow = c(2,2), mar = c(3,5,4,2))

# 10 draws from distribution
out1 <- rnorm(10, mean = 5, sd = 2)
hist(out1, breaks = 50, col = "gray", main = "Can't believe it's normal !")
mean(out1) ; sd(out1)

# 100 draws from distribution
out2 <- rnorm(100, mean = 5, sd = 2)
hist(out2, breaks = 50, col = "gray", main = 'Well, perhaps ...')
mean(out2) ; sd(out2)

# 1000 draws from distribution
out3 <- rnorm(1000, mean = 5, sd = 2)
hist(out3, breaks = 50, col = "gray", main = 'Oh !')
mean(out3) ; sd(out3)

# 1,000,000 draws from distribution
out4 <- rnorm(10^6, mean = 5, sd = 2)
hist(out4, breaks = 100, col = "gray", main = 'Oh, wow!')
mean(out4) ; sd(out4)
```

With increasingly large samples we can get an essentially arbitrarily good approximation for the distribution from which these random numbers were drawn (Fig. 2.14). But perhaps even more astonishing is how close the numerical summaries may come to the true parameters even in small samples. Look at this:

```
>  # 10 draws from distribution
[1] 5.422303
[1] 1.969975
>
>  # 100 draws from distribution
[1] 4.969041
[1] 2.345771
>
>  # 1000 draws from distribution
[1] 5.134862
[1] 1.982934
>
>  # 1,000,000 draws from distribution
[1] 5.000644
[1] 1.999467
```

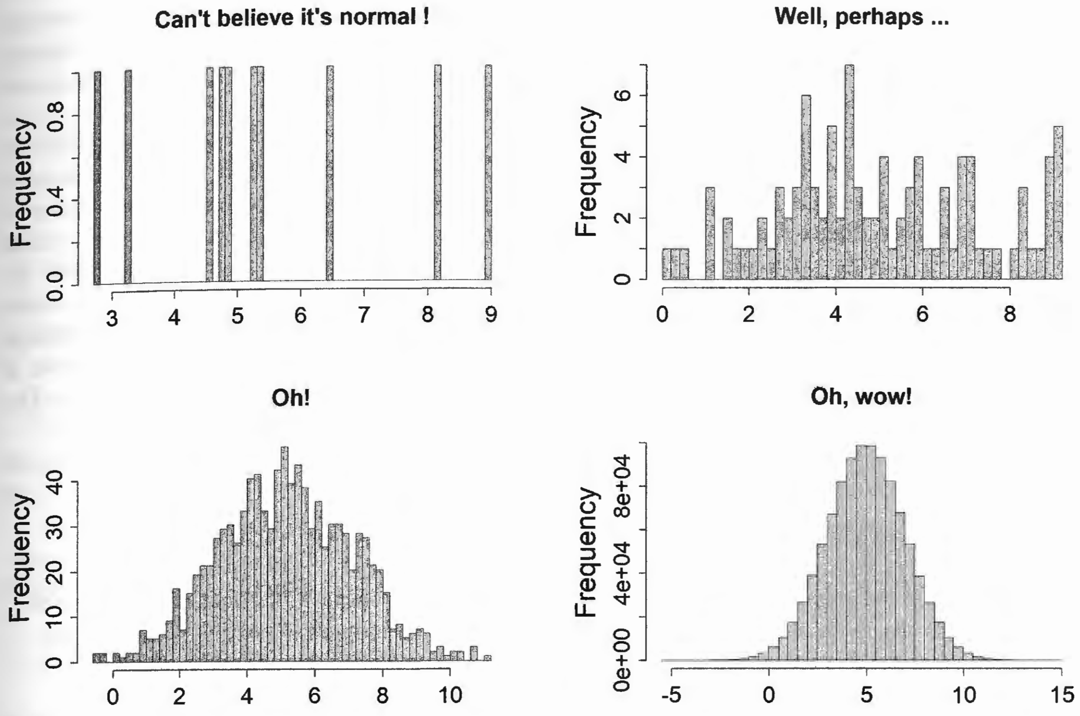

FIGURE 2.14

The principle of Monte Carlo integration illustrated with four samples of different size (10, 100, 1000, and 10^6) from the same normal(mean = 5, sd = 2) distribution. The sample mean and the sample SD provide an increasingly good estimate of the true mean and standard deviation of the distribution.

Thus, with as few as 100 or 1000 draws, we can obtain a pretty good picture of what the parent distribution looks like, either graphically or numerically. This is what we always do with Bayesian inference based on iterative simulation algorithms: we produce a large number of random numbers and then summarize them in suitable ways to infer the distributions from which they were drawn. But this is of course not all, because with real MCMC we must ensure that the distributions we draw random numbers from are in fact the posterior distributions for our parameters of interest. In the next section, we develop our own MCMC algorithm for fitting a Poisson log-linear regression to the Swiss bee-eaters from Section 2.6.

2.8.2 MCMC for the Swiss bee-eaters

We will now develop a random-walk Metropolis algorithm for our two-parameter model. This algorithm serves essentially as a RNG for the joint posterior distribution of the two parameters, that is, the intercept (`alpha`) and the slope (`beta`) of the log-linear regression of the Swiss bee-eater population counts on the years 1990–2020. We will see that in this magic algorithm the denominator of Bayes rule vanishes. Thus, one need no longer evaluate any integrals and Bayesian inference becomes possible even for very complex models such as those in Royle & Dorazio (2008), Kéry & Royle (2016, 2021), Hooten & Hefley (2019), Schaub & Kéry (2022) or Zhao (2024). For many of these, inference would have been impossible before the advent of MCMC.

We will construct a simple MCMC algorithm that in essence is nothing but a glorified RNG for the joint posterior distribution of the two parameters in our log-linear Poisson model. But in contrast to the RNG functions such as `rnorm()` in R, our "posterior distribution RNG" will not produce independent draws from a distribution, but instead the draws produced have a serial dependence by design. In addition, the draws from multiple parameters may be correlated. An MCMC algorithm iteratively produces a series of numbers where the output at each iteration is stochastically dependent on the output at the immediately preceding iteration: that's the "Markov chain" in MCMC. A Markov chain of order 1 is a stochastic process, or a series of RVs, where at each time t the output is independent from the past when the output at time t-1 is known. The "Monte Carlo" in MCMC simply indicates a simulation algorithm.

Here is the recipe for a random-walk Metropolis algorithm for the posterior distribution of parameter θ (note we use superscripts for indexing only).

(1) Starting at some arbitrary initial value for the parameter, θ^0, we use a stochastic rule for obtaining a proposed new value for it. We call the new value θ^*.

(2) We compare the proposed θ^* with the current value of θ in the chain. At the start of the chain, this means that we compare θ^* and θ^0, while at iteration t we will compare θ^* with θ^{t-1}. Specifically, we compare the two values of θ in each iteration t by calculating their posterior density ratio:

$$r = \frac{p(y|\theta^*)p(\theta^*)/p(y)}{p(y|\theta^{t-1})p(\theta^{t-1})/p(y)} \tag{2.9}$$

(3) We accept the proposed new value into our sample of draws with probability $\min(1, r)$. That is, if the posterior density of the proposed new value of the parameter, θ^*, is greater than the posterior density of the current value, θ^{t-1}, that is, if $r > 1$, we accept the proposed new value right away: we add θ^* to our sample and it becomes θ^t, which will be the "current value" in the next iteration. In contrast, if the posterior density of the proposed new value is less than that of the current value, that is, if $r < 1$, we may still accept the proposed new value, but only with probability equal to r, while with probability $1 - r$ we make a copy of the current value of θ and add that to our sample as our new current value θ^t.

And that's all there is for a very basic MCMC algorithm! We note four important things.

First, the stochastic rule in (1) is called the proposal or candidate-generating distribution h. It is often taken as a normal density, such that the proposed new value at iteration t is drawn as $\theta^* \sim Normal(\theta^{t-1}, \sigma_h)$, that is, from a normal density centered on the current value and with some SD σ_h. This normal density is NOT part of the model, rather the normal RNG is simply a convenient device to produce a stochastic proposal for parameter θ. It is the centering of the proposal distribution on the current values which builds into the output some degree of temporal autocorrelation. Thus, the proposal SD σ_h is NOT a parameter in the model, rather, it is best considered a tuning parameter or a "setting" of the algorithm. We will see that with a larger value of σ_h, successive values in the Markov chain will be less similar to each other than with a smaller value of the proposal SD. We may also say that σ_h governs the step length of the algorithm. The

MCMC algorithm is a random walk, since the proposals are produced in a "blind" manner by randomly disturbing the current value. This makes for a simple algorithm, but can lead to reduced efficiency compared to more complex algorithms such as Hamiltonian Monte Carlo (HMC), which use as information for their proposals the local gradient of the posterior in question (Section 2.10).

Second, in Eq. 2.9 the marginal distribution of the data $p(y)$ cancels from the numerator and the denominator, that is, it drops out from the equation! That is, we don't need to evaluate any integrals. These integrals were the main stumbling blocks for implementing practical Bayesian inference for most interesting models for centuries, so this feature of the MCMC algorithm is a huge relief.

Third, if we repeat this simple recipe a large number of times (e.g., 100, 1000, or more), then after some initial transient phase, the Markov chain of values produced will move towards some equilibrium distribution. There, values of θ are drawn in proportion to their posterior density $p(\theta|y)$. In other words, at equilibrium our algorithm produces serially correlated random numbers from the target posterior distribution.

Fourth, you may wonder why we may also accept proposals resulting in a lower posterior density. This is because we also want to characterize the tails of the posterior distribution, and not just the region with the highest density.

For illustration, we now fit the Poisson GLM to the Swiss bee-eater data using this type of algorithm. As for the MLE algorithms earlier, we again work in the log space. We start by defining two R functions. The first function is for the LL of the model (see also Chapter 13 for a similar model).

```
# Define R function for log-likelihood of Poisson GLM
LL <- function(param, y, x) {
  alpha <- param[1]                   # Intercept
  beta <- param[2]                    # Slope
  lambda <- exp(alpha + beta * x)     # Log-linear model
  LLi <- dpois(y, lambda, log=TRUE)   # LL contribution of each datum i
  LL <- sum(LLi)                      # LL for all observations in the data vector y
  return(LL)
}
```

The second function is for the unnormalized log-posterior density, which means it calculates only the numerator in Bayes rule, ignoring the denominator. We can ignore the marginal distribution of the data, as we have just seen, since it drops out from the posterior density ratio. As for the likelihood, working with the log posterior means that we sum LL and log-priors instead of getting products of likelihood and priors. As priors, we specify zero-mean normal densities with SD = 100. These are meant to be vague in this analysis.

```
# Define R function for un-normalized log-posterior of the model
log.posterior <- function(alpha, beta, y, x){
  loglike <- LL(c(alpha, beta), y, x)
  logprior.alpha <- dnorm(alpha, mean = 0, sd = 100, log = TRUE)
  logprior.beta <- dnorm(beta, mean = 0, sd = 100, log = TRUE)
  return(loglike + logprior.alpha + logprior.beta)
}
```

Next, we make some additional preparations: At each iteration we will keep track of acceptance/rejection of the proposed new values of `alpha` and `beta` and store them as binary indicators in an R object. The proportion of proposals that are accepted is an important indicator of the efficiency of an MCMC algorithm. The efficiency is related to the SD of the normal proposal distribution, σ_h, which we call `sd.tune` here.

```
# Choose number of iterations for which to run the algorithm
niter <- 10000

# Create an R object to hold the posterior draws produced
out.mcmc <- matrix(NA, niter, 2, dimnames = list(NULL, c("alpha", "beta")))

# Initialize chains for both parameters: these are the initial values
alpha <- rnorm(1)               # Init for alpha
beta <- rnorm(1)                # Init for beta

# R object to hold acceptance indicator for both params
acc <- matrix(0, niter, 2, dimnames = list(NULL, c("alpha", "beta")))

# Evaluate current value of the log(posterior density) (for the inits)
logpost.curr <- log.posterior(alpha, beta, y, x)

# Choose values for the tuning parameters of the algorithm
sd.tune <- c(0.1, 0.01)         # first is for alpha, second for beta
```

We are ready now to run the MCMC algorithm for the desired number of iterations. You will observe how we cycle through the parameters in the model. At every iteration we will first update the intercept and then the slope, but in this model the order does not matter.

```
# Run MCMC algorithm
for(t in 1:niter){
  if(t %% 1000 == 0)                       # counter
    cat("iter", t, "\n")
# First, update log-linear intercept (alpha)
# ------------------------------------------
# Propose candidate value of alpha
alpha.cand <- rnorm(1, alpha, sd.tune[1])  # note tuning 'parameter'

# Evaluate log(posterior) for proposed new alpha and
#  for current beta for the data y and covs x
logpost.cand <- log.posterior(alpha.cand, beta, y, x)
# Compute Metropolis acceptance ratio r
r <- exp(logpost.cand - logpost.curr)
# Keep candidate alpha if it meets criterion (u < r)
if(runif(1) < r){
  alpha <- alpha.cand
  logpost.curr <- logpost.cand
  acc[t,1] <- 1                            # Indicator for whether candidate alpha accepted
}

# Second, update log-linear slope (beta)
# ----------------------------------------
beta.cand <- rnorm(1, beta, sd.tune[2])    # note again tuning parameter
# Evaluate the log(posterior) for proposed new beta and
#  for the current alpha for the data y with covs x
logpost.cand <- log.posterior(alpha, beta.cand, y, x)
# Compute Metropolis acceptance ratio
r <- exp(logpost.cand - logpost.curr)
# Keep candidate if it meets criterion (u < r)
if(runif(1) < r){
  beta <- beta.cand
  logpost.curr <- logpost.cand
  acc[t,2] <- 1                       # Indicator for whether candidate beta accepted
}
out.mcmc[t,] <- c(alpha, beta)        # Save samples for iteration i
# NOTE: if proposed new values not accepted, then in this step
# we will copy their 'old' values and insert them into object 'out'
}
```

In reading this code, remember that `exp(log(a) - log(b))` is equal to `a/b`, but less likely to cause calculation problems, i.e. more numerically stable. In addition, the line `if(runif(1) < r)` is a shortcut for the condition in the third step of the algorithm as outlined above.

For our small data set and simple model, the algorithm runs very swiftly. We get two main outputs: one is a matrix containing 10,000 posterior draws for `alpha` and `beta`, and the other is a matrix containing the two acceptance indicators. We compute the acceptance ratio over all iterations of the algorithm and produce two time-series plots of the Markov chains (Fig. 2.15). These are called traceplots.

```
# Compute overall acceptance ratio separately for alpha and beta
(acc.ratio <- apply(acc, 2, mean))          # Should be in range 25%-40%

# Check the traceplots for convergence of chains
# Plot all MC draws right from the start (Fig. 2.15)
par(mfrow = c(2, 1))
plot(1:niter, out.mcmc[,1], main = paste('alpha (acc. ratio = ', round(acc.ratio[1], 2),')'),
  xlab = "MC iteration", ylab = 'Posterior draw', type = 'l')
plot(1:niter, out.mcmc[,2], main = paste('beta (acc. ratio = ', round(acc.ratio[2], 2),')'),
  xlab = "MC iteration", ylab = 'Posterior draw', type = 'l')
```

We see that during the initial couple of 100 iterations, the chains have not yet reached their equilibrium distribution. For inference, we discard this initial, transient phase as a "burnin," and produce another plot which takes account of a burnin of 500 iterations. Fig. 2.16 indicates that both chains have nicely converged after iteration 500. The acceptance ratio is at about one third, which is well within the range of 25%–40% that is a rule of thumb for MCMC algorithms with high efficiency (Gelman et al., 2014).

```
# Based on Fig. 2.15 choose the amount of burnin
nb <- 500                    # Discard this number of draws at the start

# Compute post-burnin acceptance ratio
(acc.ratio.pb <- apply(acc[nb:niter,], 2, mean))
```

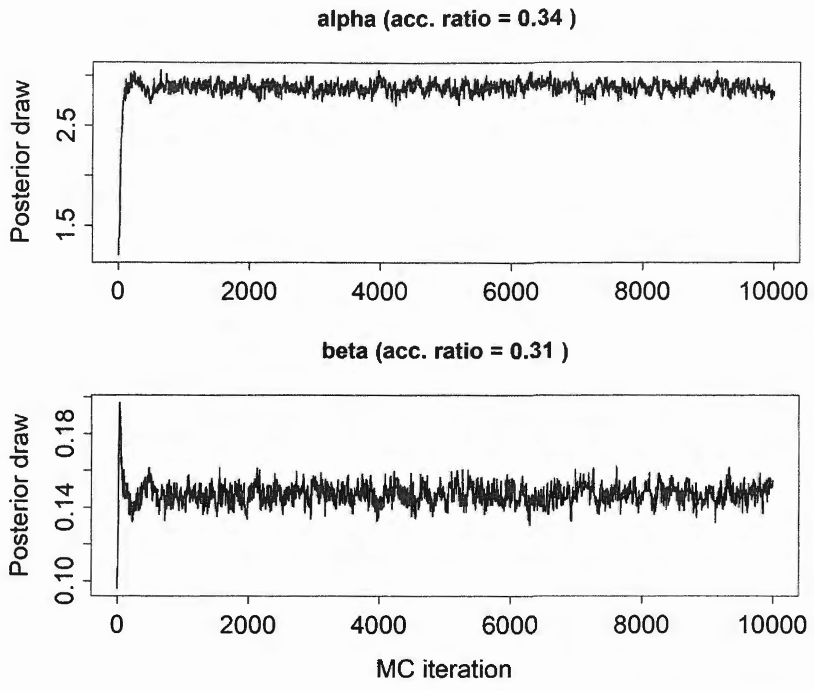

FIGURE 2.15

Traceplots of the posterior draws produced for `alpha` and `beta` right from the start of the MCMC algorithm. Acceptance ratio for each is given in the header.

```
# Repeat traceplots only for post-burnin draws (Fig. 2.16)
par(mfrow = c(2, 1))
plot(nb:niter, out.mcmc[nb:niter,1], main = paste('alpha (post-burnin acc. ratio = ',
  round(acc.ratio.pb[1], 2),')'), xlab = "Iteration", ylab = 'Posterior draw', type = 'l')
plot(nb:niter, out.mcmc[nb:niter,2], main = paste('beta (post-burnin acc. ratio = ',
  round(acc.ratio.pb[2], 2),')'), xlab = "Iteration", ylab = 'Posterior draw', type = 'l')
```

We go on to summarize the post-burnin draws from `alpha` and `beta` to obtain inferences. We compute posterior means as point estimates and posterior SDs and percentiles as the Bayesian analog of a SE and a 95% CI. We also compare our Bayesian inferences on the intercept and the slope of the Poisson GLM fit to the Swiss bee-eater data with the inferences from ML in Section 2.6. (For this you need to have the result from Section 2.6, from the call to `optim()` called `out`, in your R workspace.)

```
# Get posterior mean, sd and 95% CRI (all post-burnin)
post.mn <- apply(out.mcmc[nb:niter,], 2, mean)     # posterior mean
post.sd <- apply(out.mcmc[nb:niter,], 2, sd)       # posterior sd
CRI <- apply(out.mcmc[nb:niter,], 2,
  function(x) quantile(x, c(0.025, 0.975)))        # 95% CRI
```

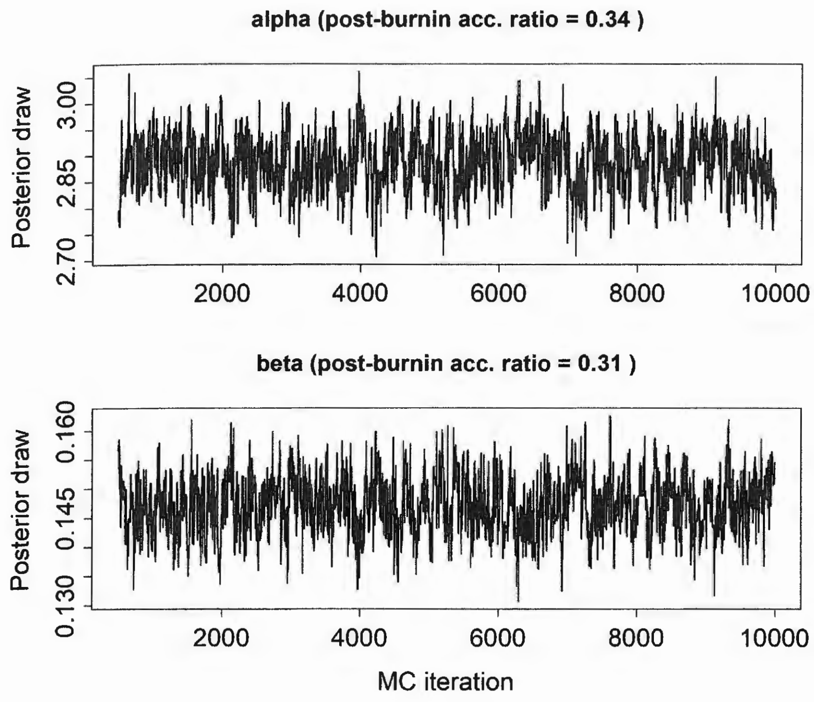

FIGURE 2.16

Post-burnin traceplots of the posterior draws produced for `alpha` and `beta`.

```
# Produce marginal posterior summary for post-burnin part of chains
tmp <- cbind(post.mn, post.sd, t(CRI))
print(tmp, 4)

# Compare Bayesian with DIY-MLE inferences
library(ASMbook)
get_MLE(out, 4)
```

```
> # Produce posterior summary for post-burnin part of chains
       post.mn  post.sd   2.5%  97.5%
alpha    2.889 0.053663 2.7795 2.9910
beta     0.147 0.004989 0.1373 0.1568
>
> # Compare with DIY-MLE inferences
         MLE       ASE  LCL.95 UCL.95
alpha 2.8868  0.052659 2.7836 2.9900
beta  0.1472  0.004864 0.1377 0.1568
```

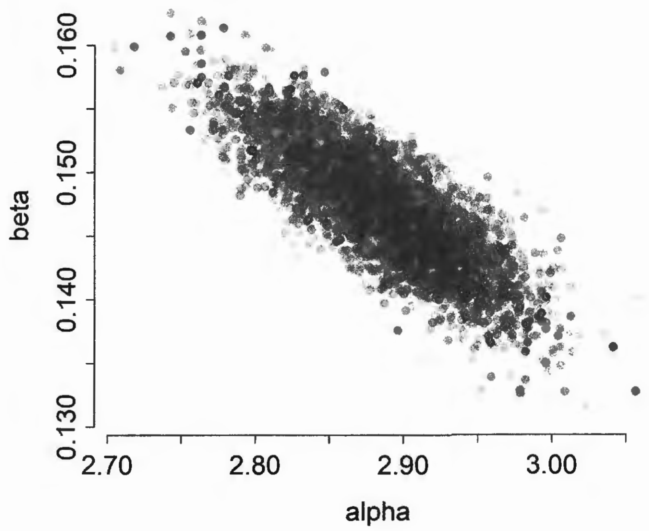

FIGURE 2.17

Joint posterior density for the two parameters of a Poisson log-linear regression fit to the Swiss bee-eater data (compare with likelihood surface in Fig. 2.12).

As is typical for a Bayesian analysis with vague priors, we find that inferences based on ML and posterior distributions are numerically practically identical. This holds both for the point estimates and for the uncertainty assessment in form of SE versus posterior SD and for 95% CI versus 95% CRI.

Finally, we want to emphasize that fundamentally what we get in the Bayesian analysis is the joint posterior density of the two parameters in the model (Fig. 2.17). Owing to our use of vague priors, this joint density will be driven by the likelihood and thus resemble very much the likelihood surface that we plotted in Fig. 2.12. We see once more that the estimates of the two parameters are negatively correlated.

```
# Plot the joint posterior distribution of alpha and beta (Fig. 2.17)
par(mar = c(5,5,4,3), cex.lab = 1.5, cex.axis = 1.5)
plot(out.mcmc[nb:niter,1], out.mcmc[nb:niter,2], xlab = 'alpha', ylab = 'beta',
   frame = FALSE, cex = 1.2, pch = 16, col = rgb(0,0,0,0.1))
```

Summarizing the posterior distribution of one parameter in a multiparameter model involves an integration over the distribution of the other(s) (Royle & Dorazio, 2008: chapter 2). With MCMC-based inferences, we can easily obtain such marginal posterior inferences by simply summarizing the joint posterior for one of its dimensions, while ignoring the others. This is simply another instance of Monte Carlo integration. Fig. 2.18 provides what is the most complete depiction of the inference about each parameter alone. For `alpha` and `beta` this is a probability distribution that conveys our knowledge about the magnitude of these parameters after we have used the information in our bee-eater data set with the Poisson GLM used as an inferential vehicle. Comparing the

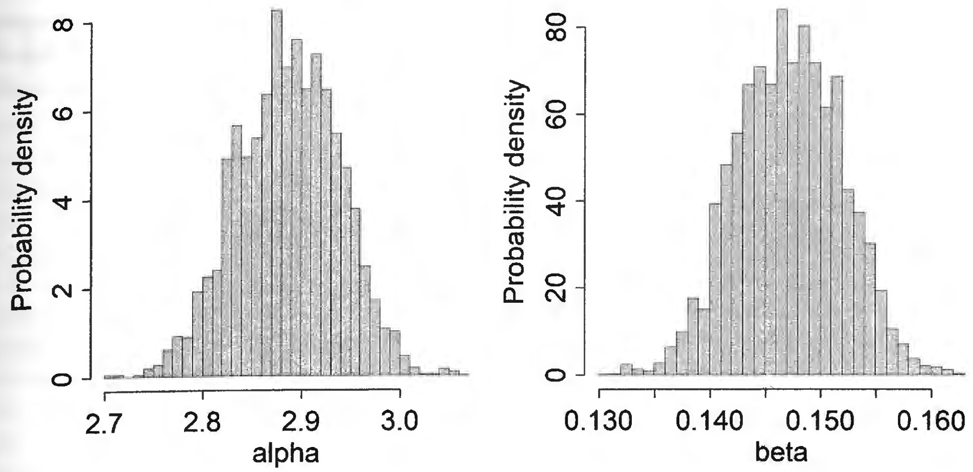

FIGURE 2.18

Marginal posterior distributions for the two parameters of a Poisson log-linear regression fit to the Swiss bee-eater data. These are achieved by integrating the joint distribution in Fig. 2.17 over the other dimensions for each parameter.

figure with the numerical marginal posterior summaries above, you can probably recognize posterior mean, SDs, and the 95% CRIs.

```
# Plot the marginal posterior distributions of alpha, beta (Fig. 2.18)
par(mfrow = c(1, 2), mar = c(5,5,4,3), cex.lab = 1.5, cex.axis = 1.5)
hist(out.mcmc[nb:niter,1], xlab = 'alpha', col = 'gray', main = '')
hist(out.mcmc[nb:niter,2], xlab = 'beta', col = 'gray', main = '')
```

2.8.3 A practical in MCMC for the bee-eaters with function demoMCMC() (*)

We have packaged in the R function demoMCMC() the code for the MCMC algorithm in Section 2.8.2 so that we can play around with it and explore some features of these algorithms. We don't have any real exercises in this book, and yet here is a practical which we ask you to do for yourself. Please do it: it's very simple and what we ask is that you just execute the code and observe what happens ... and there is a really cool animation. Here is the function with all its arguments at their defaults.

```
str(tmp <- demoMCMC(y=y, x=x, true.vals = c(2.5, 0.14), inits = c(0, 0),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.1, 0.1),
  niter = 10000, nburn = 1000, quiet = FALSE, show.plots = TRUE))
```

The inits are the initial values, the two arguments `prior.sd.alpha` and `prior.sd.beta` are the SDs of zero-mean normal priors for the two parameters, while the `tuning.params` are the SDs of the normal proposal distributions, called σ_h above. Finally, `niter` and `nburn` specify to the total length of the single chain and the number of iterations discarded as a burnin. When you execute the code in the next box, close the graphics window after each model run. Also,

*For this section, RStudio users are advised to switch to base R to experience the full power of R.

don't initialize at more extreme places than (−40, 40), since otherwise you will get a numerical failure. Note that this function just implements an MCMC algorithm for a Poisson log-linear model with a single continuous covariate; thus you could of course use it to fit this model to other data sets than just the bee-eaters.

```
# Shorter run than default
# (can use for true.vals the MLEs from above for graphics)
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(0, 0),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.1, 0.1),
  niter = 2000, nburn = 1000))          # Bad mixing, strong autocorrelation

# Smaller tuning param to increase acceptance
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(0,0),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.1, 0.01),
  niter = 2000, nburn = 1000))

# Larger step length (=tuning.params)
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(0, 0),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.5, 0.5),
  niter = 2000, nburn = 1000))          # Ugly Manhattan traceplots, not efficient,
                                        # no convergence after 1000 steps

# Try to break by picking far-out inits (pretty amazing!)
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(40, 40),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.1, 0.1),
  niter = 5000, nburn = 2000) )

# Try to break some more ... (perhaps even more amazing)
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(-40, 40),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.1, 0.1),
  niter = 5000, nburn = 2000) )

# ...and more
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(40, 0),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.1, 0.1),
  niter = 5000, nburn = 2000) )         # Remarkable search trajectory!

# Same with bigger step length (and shorter chain)
str(tmp <- demoMCMC(y = y, x = x, true.vals = MLE, inits = c(40, 0),
  prior.sd.alpha = 100, prior.sd.beta = 100, tuning.params = c(0.5, 0.2),
  niter = 3000, nburn = 2000) )
```

We believe that by experimenting with this function you can develop some valuable intuition for how basic MCMC algorithms work, and how settings such as the initial values, priors, tuning parameters, and MCMC settings (here, `niter` and `nburn`) affect what they produce. We also hope that you share our own astonishment at how incredibly powerful such a very simple algorithm can be for exploring posterior densities. In this respect, especially the last few calls to the function are really interesting, where we make the job of the function much harder by initializing the search at far-out places in the parameter space. And yet the algorithm finds the solution, that is, that part of the parameter space where most of the posterior density is located.

2.9 So should you now be a Bayesian or a frequentist?

Our short answer is: you should be both! Here is why. Maximum likelihood and Bayesian posterior inference are both very powerful and general methods for fitting models to data, that is, for conducting statistical inferences based on a model and a data set. There are a number of advantages and of disadvantages to both ML and Bayes, which we summarize next.

ML is an extremely well-understood method with a very long history (Fisher, 1922). It is "automatic" since you simply define the joint density of the data, which links the unknowns (the parameters) with the knowns (the data). Then, you treat that density as a likelihood function and minimize the NLL to obtain the MLEs. There exist several methods to obtain uncertainty assessments such as SEs and CIs associated with the MLEs. In addition, MLEs have a number of desirable properties which in general make them "good." ML is arguably the most widely used estimation method.

But ML has disadvantages, too. First, MLEs can be hard or impossible to obtain for complex models, such as those with lots of random effects, especially when we model correlations as in spatial or time-series models. Second, SEs and CIs are only valid asymptotically, that is, for "large" samples, and we rarely have truly large samples in ecology. Third, we often want to compute functions of parameters, such as transformations of one parameter or ratios of two parameters. The delta rule and the bootstrap may be used to compute SEs for those (Kéry & Royle, 2016: chapter 2), but both may become hard or impossible for more complicated functions and models. Fourth, ML is fundamentally based on a probability statement about the data, not about the parameters. That is, a probability such as the "95%" for a CI is *not* about the parameter in question, but about the reliability of the method used to compute the interval. Also, you don't have the right to say things such as "*I am 95% certain that my population is declining*" if 0 is outside of a 95% CI for a population growth rate parameter. And finally, the appeal to hypothetical replicate data may appear ridiculous: for instance, what could we possibly mean with "1 million replicate populations of panda bears"?

There are at least three major advantages of using Bayesian inference using tools such as JAGS, NIMBLE, and Stan. Interestingly, they are fairly independent reasons for choosing Bayesian inference. First is the Bayesian paradigm for inference, which interprets probability exactly like most humans use it: as a measure of knowledge about uncertain things. Also, when we want to introduce into our model fitting external information, then it is very clear how to do this via the prior distributions. The second reason for wanting to go Bayes has to do with Bayesian computation, which typically means use of MCMC and other simulation algorithms. These make it easy to fit hierarchical models to accommodate multiple sources of uncertainty and correlations in the data (see Chapter 3). These days a large part of the models fit in applied statistics fall under this category. Also, simulation-based algorithms make it really easy to compute derived quantities or functions of parameters along with a full assessment of their uncertainty. Thus, you no longer have to worry about the right delta-rule approximation for the SE of functions of parameters. The third good reason for going Bayes has to do with the model-definition language adopted in JAGS and NIMBLE, and earlier in BUGS, WinBUGS, and OpenBUGS (Lunn et al., 2013), and to a lesser degree also in Stan. The BUGS language has proven to be superbly accessible to statisticians and nonstatisticians alike for specifying even very complex statistical models. It is the BUGS language that brings within the reach of ecologists the implementation of really complex, custom models. With BUGS you can develop models that you would never have been able to fit if you had to use ML. Thus, we like to say that BUGS frees the modeler in you.

But Bayesian inference also has some disadvantages. First, the prior will always exert some influence on your results, and in data-poor situations, this influence will be greater. We will see this in several chapters later in the book. This may make reporting of your results more complex since you may need to tell multiple stories. Second, for complex models and big data sets simulation-based methods such as MCMC may take a very long time to run, for example, days or even weeks. Worse yet, sometimes you may never achieve convergence within a reasonable time. Third, the final disadvantage is that BUGS engines may be so flexible and powerful that it may be easy to fit nonsensical models. By this we mean especially models with parameters that are unidentifiable, that is, for which your data contain no information. True, parameter identifiability is a very big issue for any complex statistical model and it may plague inferences by ML just as well. However, lack of identifiability of a parameter may be even harder to diagnose when using Bayesian inference than when using ML. There are formal methods to confront this (Cole, 2021), but they may be hard to implement for complex models and for nonstatisticians. In practice, the best method to judge whether a parameter is identifiable is simulation (Kéry & Royle, 2016: chapter 4): you simulate a large number of data sets with varying values of all parameters, especially of the suspect parameter, fit your model, and compare estimates and true values. For an indentifiable parameter, there ought to be a clear numerical resemblance between estimates and true values.

In summary, we completely agree with Royle & Dorazio (2008) that ecologists need a good working knowledge of both ML and Bayesian methods in statistical modeling. Indeed, this whole book is predicated on this idea. Sometimes, or for some of you, one is the better choice, while another time or for others it's the other. In addition, the two methods of inference have far more in common than what one is sometimes made to believe: ML and Bayes share their basis as model-based approaches to inference, with a central role played by the likelihood function. In addition, by far the most common way in which Bayesian analyses are conducted in practice is such that we use the powerful computing machinery associated with Bayesian MCMC methods to solve difficult estimation problems, while minimizing the amount of information contained in our priors. That is, we work with vague priors. As a result, at least numerically we are then almost always back to something that greatly resembles the same old MLEs. We demonstrate this throughout the book.

2.10 Summary and outlook

In this chapter, we have seen that probability is the foundation of statistical modeling and inference. We then covered the key concept of a RV, which is defined as a real-valued function defined on the sample space of a random experiment. RVs are described by PMFs when they have discrete values only or by PDFs when continuous. A statistical model is a collection of PDFs or PMFs for all your response variables. We have also seen that models have multiple uses in statistical inference, but that a key thing we use them for is to "fit them to data" to estimate parameters (we note in passing that we always fit a model to data, but we never fit data to a model). The two dominant methods of parameter estimation are both based on the likelihood function, which is algebraically identical to the joint density (or probability) of all RVs, but viewed as a function of the parameters. The likelihood function links what we observe (the data) with what we don't observe but would like to estimate (the parameters). It carries the information about the parameters contained in our data set.

With ML we work directly with the likelihood function for statistical inference, by maximizing it over the parameter space to obtain the MLEs and using the curvature of the likelihood function around the MLEs for uncertainty assessments. The concept of probability in ML is based on a relative-frequency interpretation in a large number of hypothetal replicates of our data set, that is, as a measure of the variability of observable quantities or essentially of the variability of data. In contrast, in addition to the use of probability as a measure of variability of observable things, Bayesians adopt a more general definition of probability as a metric for our knowledge about any kind of uncertain things, including parameters in a model. They use the likelihood function to update their prior knowledge about the parameter(s) via Bayes rule, in a simple application of conditional probability to statistical inference. The result is another probability distribution, the posterior distribution, which contains all information contributed by the data (via the likelihood) and our prior assumptions about a parameter (represented by the prior). We can then summarize the posterior by its mean for a point estimate and by its spread for an uncertainty assessment of that estimate. We have also seen that with vague priors, Bayesian point estimates and uncertainty assessments are typically very similar, numerically, to those from ML.

In practice, Bayesian inference is most often obtained with MCMC methods, for which we have described and studied the prototypical case of a random-walk Metropolis algorithm. Even with this simple algorithm, we can obtain a lot of useful intuition about the workings of more complex simulation-based algorithms which are used in JAGS, NIMBLE, Stan, and also other Bayesian model-fitting software. MCMC algorithms are incredibly powerful and yet conceptually simple algorithms. For plenty of inference problems, they work extremely well if you are prepared to wait some, but for others, convergence may be harder to achieve or even never at all.

At the end of the chapter, we have argued that both ML and Bayes have a number of advantages and disadvantages. We are convinced that to become a successful quantitative ecologist, you need to understand both ML and Bayesian inference, and this is indeed a major reason for why we wrote this book. But even if you want to go Bayes only, you still need to develop a sound understanding of concepts that underlie both methods. Most of all, you need to understand the concepts of the joint density, likelihood and log-likelihood, which are essential ingredients of every Bayesian analysis. Arguably, these may be easier to grasp in the context of ML, and this is one of the reasons for which we emphasize DIY-MLEs (see Section 4.7) so much throughout the book.

Much of the material in this chapter is conceptual and meant to demonstrate the workings of ML and Bayesian inference, along with their typical computational implementations. However, you could probably use the simple MCMC code in this chapter as a starting point for writing your own code for more complex or different models, for example, by translating the Poisson into a binomial likelihood, or by adding more covariates. As always, it's usually by making such small incremental changes to existing code that you learn and write new code. For much deeper and comprehensive introductions to MCMC methods for ecologists, see the recent books by Hooten & Hefley (2019) and Zhao (2024). For a deeper introduction to ML geared at biologists, excellent references include Bolker (2008), Royle & Dorazio (2008: chapter 2), and Millar (2011).

Now that we have covered two powerful methods for fitting statistical models to data, we will first (in Chapter 3) cover some of the most typical parametric statistical models applied in ecology. Then, in Chapter 4 we will give an overview of several extremely powerful, general-purpose model-fitting engines that we use throughout the book, namely JAGS, NIMBLE, and Stan for Bayesian inference (mostly) and R and TMB for ML inference (mostly). In all later chapters, we will also continue getting our own MLEs "by hand", because we believe it is so important that ecologists get a better understanding of ML. For Bayesian inference we will no longer write our own algorithms, but use JAGS, NIMBLE, and Stan.

Further reading

- Probability and statistics: Casella & Berger (2002), Pishro-Nik (2014), Blitzstein & Hwang (2019); with focus on biology/ecology: Royle & Dorazio (2008: chapter 2), Royle et al. (2014: chapter 2), Kéry & Royle (2016: chapter 2).
- Bayesian inference: Lindley (2006), Carlin & Louis (2009), Gelman et al. (2014a), Kruschke (2015), Spiegelhalter (2019), McElreath (2020); with focus on biology/ecology: Royle & Dorazio (2008: chapter 2), King et al. (2009), Link & Barker (2010), Royle et al. (2014: chapter 2), Hobbs & Hooten (2015), Kéry & Royle (2016: chapter 2), Inchausti (2023), Gimenez (2025).
- Maximum likelihood (with focus on biology/ecology): Bolker (2008), Royle & Dorazio (2008: chapter 2), Millar (2011), Lee et al. (2017: chapter 1 [without biology]), Inchausti (2023).
- Markov chain Monte Carlo (with focus on biology/ecology): Royle & Dorazio (2008: chapter 2), Royle et al. (2014: chapter 2), Kéry & Royle (2016: chapter 2), Hooten & Hefley (2019), Zhao (2024).