Joseph E. Hightower

A Bayesian Introduction to Fish Population Analysis

To my son,

without whom I should have finished this book two years earlier

Contents

List of Figures **vii**

List of Tables **ix**

Preface **xi**

About the Author **xv**

1 Introduction **1**

2 A Brief Introduction to R and RStudio **1**

3 Probability Distributions **1**

3.1 Discrete 1

3.1.1 Bernoulli 1

3.1.2 Binomial 4

3.1.3 Multinomial 6

3.1.4 Poisson 9

3.1.5 Negative binomial 10

3.2 Continuous 12

3.2.1 Uniform 12

3.2.2 Beta 13

3.2.3 Normal 14

3.2.4 Lognormal 14

3.2.5 Gamma 16

3.3 Exercises 17

4 Model Fitting **1**

4.1 Introduction 1

4.2 Using simulation 2

4.3 Bayesian approach 4

4.3.1 Debugging code 10

4.3.2 Judging convergence 14

4.3.3 What is a likelihood? 15

4.3.4 Deviance as a measure of model fit 17

4.3.5 Model checking 17

4.3.6 Model selection 22

4.4 Exercises 25

5 Abundance **29**

5.1 Removal 29

5.2 Mark-recapture 32

5.3 Binomial-mixture 34

5.4 Exercises 39

6 Survival **41**

6.1 Age-composition 41

6.1.1 Accounting For Gear Selectivity 44

6.1.2 Cohort model 51

6.2 Telemetry-based 54

6.3 Tag-based 58

6.4 Exercises 63

7 Mortality Components **65**

7.1 Exploitation rate 65

7.2 Completed tagging study 67

7.3 Multi-year tagging study 71

7.4 Telemetry estimates of F and M 76

7.5 Exercises 83

8 Growth **85**

8.1 Growth in length 85

8.1.1 Age-length 86

8.1.2 Using tagging data 88

8.2 Growth in weight 92

8.3 Exercises 96

9 Abundance trends **97**

9.1 Linear model 97

9.2 Exponential growth model 100

9.2.1 Forecasting 103

9.3 Exercises 106

10 Recruitment **107**

10.1 Fitting a stock-recruitment curve 108

10.2 Does spawning stock size affect recruitment? 113

10.3 Exercises 115

11 Integrated Population Models **117**

11.1 Exercises 121

Appendix **123**

A Title Placeholder **123**

Bibliography **125**

List of Figures

3.1 Summary bar plot of coin tosses. 3

3.2 Summary bar plot of die rolls. 8

3.3 Negative binomial frequency distributions for k=1 (left) and 10 (right). 11

3.4 Histogram for 200 random draws from a beta distribution, with alpha=1 and beta=0.5 (left), 1 (center), and 5 (right). 13

3.5 Histogram for 200 lognormally-distributed values, with ln-scale mean of 0 and variance of 0.5. Red vertical line denotes expected mean. 15

3.6 Histogram for 200 gamma-distributed values, with alpha=3 and beta=1 (left) and 3 (right). 16

4.1 Frequency distributions for number of recaptured fish (left) and estimated population size (right), based on simulation using a binomial distribution for the number of recaptures. 3

List of Tables

Preface

This book is based in large part on material I developed while teaching (1991-2014) at NC State University. My hope is that the book will be a bridge between traditional fisheries analytical methods and Bayesian approaches that offer many advantages in ecological modeling. The book might be useful as an upper-level undergraduate or early graduate text, or for a working fisheries biologist interested in a hands-on introduction to Bayesian methods.

The general approach for this book follows that used by **?** and **?**, in that sample data for each Bayesian analysis are produced by simulation. Analyzing simulated data is helpful for learning because the expected results are known. Simulating a field study also provides the flexibility to vary study parameters such as the number of samples or the within-sample variability. Running the simulation and analysis code multiple times under different settings will aid in understanding the methods and their limitations. This brings up a general point about this book: reading about an analysis is not enough – it is essential to run the code in order to understand the methods. Exercises included at the end of each chapter should also aid in understanding the strengths and weaknesses of each analysis. A solutions manual is available from the author.

Software information and conventions

This book has been prepared using the **knitr** (Xie, 2015) and **bookdown** (Xie, 2024) packages. Package names (e.g., **bookdown**) will be indicated by bold text; typewriter font will indicate inline code (e.g., `x^y`) or a function name (followed by parentheses, e.g., mean()). My R session information is shown below:

```
xfun::session_info()
```

```
## R version 4.2.2 (2022-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Locale:
##   LC_COLLATE=English_United States.utf8
##   LC_CTYPE=English_United States.utf8
##   LC_MONETARY=English_United States.utf8
##   LC_NUMERIC=C
##   LC_TIME=English_United States.utf8
##
## Package version:
##   base64enc_0.1.3   bookdown_0.38
##   bslib_0.6.2       cachem_1.0.8
##   cli_3.6.2         compiler_4.2.2
##   digest_0.6.35     ellipsis_0.3.2
##   evaluate_0.23     fastmap_1.1.1
##   fontawesome_0.5.2 fs_1.6.3
##   glue_1.7.0        graphics_4.2.2
##   grDevices_4.2.2   highr_0.10
##   htmltools_0.5.7   jquerylib_0.1.4
##   jsonlite_1.8.8    knitr_1.45
##   lifecycle_1.0.4   memoise_2.0.1
##   methods_4.2.2     mime_0.12
##   R6_2.5.1          rappdirs_0.3.3
##   rlang_1.1.3       rmarkdown_2.26
##   rstudioapi_0.16.0 sass_0.4.9
##   stats_4.2.2       tinytex_0.50
##   tools_4.2.2       utils_4.2.2
##   xfun_0.42         yaml_2.3.8
```

Acknowledgments

My first exposure to Bayesian statistical methods was via Matthew Krachey, who took the time as a PhD student to teach a few informal sessions on WinBUGS. I soon came to agree with **?** that the BUGS language "frees the modeler in you" and would be a valuable tool for research and especially teaching. I incorporated the BUGS language into my graduate fisheries course and soon into research population models. Other colleagues who provided valuable assistance on statistical methods include Beth Gardner, Ken Pollock, and Brian Reich. I also benefited from some of the many excellent reference books on Bayesian methods for ecology (e.g., McCarthy, 2007; **?**; **?**).

Anonymous reviewers of previous drafts made a number of helpful suggestions

that I have attempted to accommodate. Any remaining errors in the book are mine.

About the Author

Joseph E. Hightower is Professor Emeritus in the Department of Applied Ecology at NC State University in Raleigh, North Carolina.

1

Introduction

Survey a group of fishery biologists and many would say that they entered the field because of a love of the outdoors. They might also admit that, despite a preference for being outside, much of their work is done at a computer. There are mundane tasks like email correspondence, but of greater importance (and relevance for this book) are the steps in getting the information needed for effective management. Those steps include planning and conducting field studies, carrying out correct analyses, and presenting results in reports and presentations. A strong statistical foundation is essential for accomplishing these tasks. Otherwise, management ends up being a trial-and-error process that is inefficient and often ineffective. The purpose of this book is to provide an introduction to these quantitative methods.

The models that we will investigate have a statistical foundation but our interest is primarily in parameter estimation rather than hypothesis testing. This is the opposite of most university statistics courses, but it reflects the reality of fishery management. For example, we might conduct a tagging study in order to estimate the exploitation rate, or the fraction of the legal-sized fish that are harvested in a year. We are not interested in a trivial hypothesis test ("Is the exploitation rate 0?") but just in getting a reliable estimate. If a substantial fraction of the population is being harvested, then the fishery manager might consider harvest restrictions such as a closed season, bag limit, or size limit. The tagging study could be done using red tags that have a reward of \$100 and yellow tags that have a reward of \$5. Anglers typically return high-reward tags at a higher rate than low-reward tags, so one part of the planning process would be to decide how many tags of each type to use. Once the study is complete, the model used for planning can often be modified for analysis of the tag-return data. One step in the process would be to determine whether the model needs separate parameters for the return rates of high- and low-reward tags. That could be viewed as a hypothesis-testing situation, but can also be thought of as model selection (simpler model with one rate versus a more complex model with two). Ultimately, we simply want to produce the best possible information about the effect of fishing on the population. This would include not only a point estimate but also measures of uncertainty that aid in decision making.

Examples in the chapters that follow use Bayesian statistical methods to illustrate the steps in planning and analysis of field studies. Fisheries analyses (and university statistics courses) have traditionally been based on so-called frequentist statistical methods, that are based on the expected frequency of observing the data in hand, if the study were repeated many times (McCarthy, 2007). Until recently, an important advantage of frequentist methods was that parameter estimation, hypothesis testing, etc. could be implemented with commonly available computing resources (Dorazio, 2016). In contrast, Bayesian methods of statistical inference were not practical until fairly recently, when generic algorithms known as Markov Chain Monte Carlo (see Section 4.3) became available (Dorazio, 2016). Another important factor in the increased adoption of Bayesian methods is the availability of public-domain software to fit the models (**?**). The earliest version was known as BUGS (Bayesian inference Using Gibbs Sampling; Lunn et al. (2009)), followed by a Windows version WinBUGS (Lunn et al., 2000), then an open version OpenBUGS (Lunn et al., 2009), as well as JAGS (Plummer, 2003) which was developed independently but uses essentially identical code. The BUGS language is an important development in that it "frees the modeler in you" (**?**). Rather than trying to force the data into a rigid black box for analysis, study-specific code can be produced that is readable and biologically meaningful. Another advantage of BUGS software is that it forces you to think about the biological and field-sampling processes that generated your data. Thus it is a great tool both for learning and for getting work done. We will use JAGS in this book, but the code should work without modification in other versions of the BUGS family, and could be converted to run under other Bayesian software such as NIMBLE (**?**) or Stan (Gelman et al., 2015). The JAGS code used in this book will be run from R (R Core Team, 2022), a general software environment that is briefly described in Chapter 2.

2

A Brief Introduction to R and RStudio

R (R Core Team, 2022) is a free software environment for data manipulation<, statistical analyses, and graphics. It is easily extended through add-on software packages including some for commonly used fisheries models. There are a number of software interfaces for working in R and you should use the interface with which you are most comfortable. For this book, we use the free software RStudio[1], which can be run in either the Windows, Macintosh, or Unix environment. There are many online resources for installing and learning R and RStudio, and even a recent book for fisheries analysis using R (Ogle, 2018). We assume in this book that R and RStudio have been installed. We get started by examining a few basic R commands but will primarily illustrate the use of R through code examples in the remaining chapters. We will also describe some general methods for getting help and learning more about R.

As a first example using R, let's estimate population size through a two-sample mark-recapture study. Consider a pond with N=100 fish. If you take a first sample (n_1=50) and mark (tag) those fish, then half the fish in the pond are tagged. If you take a second sample (say n_2=30), how many tagged fish (m_2) would you expect in the second sample? That's right, 15! Of course, in practice, we don't know N but we solve for it. We begin by setting the two proportions equal: $n_1/N = m_2/n_2$. Solving for N gives us our estimator (N_{hat}): $\hat{N} = \frac{n_1 * n_2}{m_2}$

We can carry out the calculations using R code as follows. First, open a new RStudio window (menu commands File : New File : R Script). This Source window will contain your R script, or the sequence of R commands you intend to execute. Copy and paste the following code into the Source window (from the online version[2] if using a print copy of the book):

```r
setwd("Q:/My Drive/FishAnalysis") # Edit to show your working directory

# Two-sample population estimate
n1 <- 26 # Number caught and marked in first sample
n2 <- 14 # Number caught and examined for marks in second sample
```

[1]https://posit.co/products/open-source/rstudio/
[2]https://bookdown.org/Joseph_Hightower/bayesianfish/

```
m2 <- 7 # Number of marked fish in second sample
N_hat <- n1*n2/m2 # Store estimate as variable N_hat
N_hat # Prints N_hat in Console window
```

Any text following a pound sign (#) is a comment, whether on a line by itself or following executable R code. Comments do not affect the program's execution, but are extremely helpful in documenting your code. Taking the time to enter comments is always worthwhile, whether for your own information or when code is to be shared with others.

Next, position the cursor on the first line and execute one line at a time by clicking on the Run icon. This allows you to see the effect of each statement. The first line calls the `setwd()` function to establish the working directory for any files saved. Doing this routinely will ensure that files end up in the proper location. You can find out your working directory path by using the RStudio menu command (Session | Set Working Directory | Choose Directory). That allows you to navigate to the correct location, and the `setwd()` code will be printed to the Console. You can then save that line of code for future use.

The next three executable lines assign (<-) values to variables, which have arbitrary names. Here, I have used conventional fisheries mark-recapture names for the sample sizes and number of recaptures, but more descriptive names could be used (e.g., Mark_Sample) for clarity. Each variable is a new object that will appear in the Environment window once that line of code has been run. The next line of code calculates the estimate, which is stored as N_hat. The final line prints the value of N_hat in the Console window. The population estimate (52) makes sense because you marked 26 and half the fish in the second sample were marked.

After you have run this code in RStudio, be sure to save the file. This is a good general practice – never create code from scratch when you can modify existing code! It cuts down on typing errors and allows you to build on existing code when doing new but similar things. The code could be from your prior work or from a journal article or book. The ability to share R and JAGS code is very helpful in learning and working efficiently. Saving the file to a working directory specified in the code helps in keeping together all files related to a specific project. Use a descriptive name for the code (R script) so that it can be readily located in the future.

Much of the work done in R is accomplished by using functions, which are a set of statements that perform a specific task. Typing the name of an R function into the search box in the RStudio Help window is one quick way to get help (e.g., function defaults, arguments, examples). However, the R documentation is sometimes overly detailed and cryptic, so another good option is an online search (e.g., "R function max" or "how to load csv file R"). Another helpful alternative is the R Reference Card[3], which provides a brief summary of some of the more frequently used R commands. For example, the reference card lists a number of math functions such as `log(x, base)`, which returns the log of x to the specified base. The statement `log(x)` returns the default natural-log base. This is typical of R, in that functions have default arguments that need not be specified. (It can also be a bit dangerous if the default action is not what you intended!) Another important aspect of R is that the object returned by the `log()` function depends on the argument. Consider the following four lines of code:

```r
x <- 2.72
log(x)
x <- c(4, 7)
log(x)
```

We can test the code in a new R script window. If you have worked through this section in a single session, then objects from the mark-recapture example will be in the Environment window. Start with a clean slate by clicking on the broom icon in the Environment window. Like most things in R, this can also be accomplished through menu commands (Session : Clear Workspace) or R code `rm(list=ls())`. The first R statement assigns the value 2.72 to x, and the second statement prints the natural log of 2.72 in the Console window. The third line uses the `c()` (combine) function to create a vector of length two. Because we used the same variable name to store the result, R replaces the original integer value of x with a vector containing two integers. (It is done here as an example, but in the future, we will avoid this bad practice of reusing a variable name for a different purpose!) In the final line of code, the `log()` function returns a vector by applying the `log()` function element-by-element. This flexibility (allowing x to be defined first as a scalar then vector, and allowing either type as an argument to the `log()` function) is a strength of R, but it also increases the risk of coding errors compared to programming languages that are more structured. The keys for success in R programming are to check and test code carefully and to reuse existing code whenever possible. The chapters that follow will provide many more examples of R code, with comments and explanation as needed as new features are introduced. As you encounter R code, you should always run it to make sure that you understand how it works and its purpose.

[3]https://cran.r-project.org/doc/contrib/Short-refcard.pdf

3

Probability Distributions

When a biologist conducts a field study, the usual purpose is to collect data that will be useful for management. Examples of data that could be collected include (i) whether a fish with a radio transmitter was detected, (ii) how many tags were reported by anglers over a fishing season, (iii) how many fish of each age were observed in a sample, or (iv) number of fish caught in a single tow of a net. Each of these examples arises from a different statistical distribution. The distribution affects the properties of the data and how those data might be analyzed. For example, observations in the first example are either Yes or No, which could be represented as a 1 or 0. Having only those two possible values certainly affects characteristics of that distribution as well as methods for analysis.

In this chapter, we will examine some of the more common and important statistical distributions, and learn how to simulate observations from those distributions. Simulation will be an important skill in the chapters that follow, in that simulated data with known parameters provide a valuable way of evaluating field study designs and methods of analysis. The first few distributions that we consider are discrete (Section 3.1); that is, non-negative whole numbers that are appropriate for counts like the number of fish in a single haul of a net. The second category is continuous distributions (Section 3.2), for real numbers such as a catch *rate* of 0.52.

3.1 Discrete

3.1.1 Bernoulli

The most basic discrete distribution is the Bernoulli. There are only two outcomes, often referred to as success and failure. The classic example of a Bernoulli distribution is a coin toss, where "heads" might be considered success and "tails" failure. This distribution has a single parameter p, representing the probability of success. Its complement (1-p) is the probability of failure. Searching a lake for a fish with a radio transmitter is a fisheries example of a Bernoulli trial, where p represents the probability of detecting the fish's signal (success). Other biological examples are whether a fish with a transmitter is dead or alive and whether or not a study site is occupied.

Let's begin with R code for a hands-on example, where a coin was tossed ten times. It is convenient in R to use 0 for tails and 1 for heads:

```
y <- c(0,0,1,0,0,1,0,0,0,1)  # Coin toss observations, 0=tails
N.obs <- length(y) # R function to count number of observations
```

As usual, you should paste the code into an RStudio script window. You can select both lines and click the Run icon in order to run both at once. The first line assigns the vector of 0s and 1s to the variable y. The next line uses the `length()` function to count the number of observations in the vector y. We could have used the code N.obs <- 10 but there are two related reasons not to do so. First, hard-coding a constant for sample size means that we would need to remember to change that line if the y vector was later updated with a different number of observations. Second, you should never do anything manually that computer code can do for you. Letting R count the observations means that you will never miscount, and your code will still work correctly if you change the y vector and end up with a different sample size.

Another critical step in data analysis is to plot the data whenever possible, in order to look for patterns. Here, plotting the sequence of 0s and 1s is not that useful. A better summary plot is to examine the total number of 0s and 1s. We can use the `table()` function, which returns a vector with the number of 0s and 1s in the y vector. The `barplot()` function plots the two counts (Figure 3.1):

```
Counts <- table(y)  # Summary table of 0 vs 1
barplot(Counts)
```

Running these lines of code in a script window causes this plot to appear automatically under the Plots tab in the lower right RStudio window. We can save the plot image or copy and paste it into a report using menu option Export under the Plots tab. Examining the plot shows the unusual result of seven "tails" and three "heads". We would expect the counts to be about equal for the two categories, given the assumed probability of success (p) for a fair coin of 0.5.

Can you think of a physical process (hands-on method) that would produce Bernoulli data using other values for p, such as 0.25 or 0.33? What value for p might you expect for a fisheries examples, such as an annual exploitation

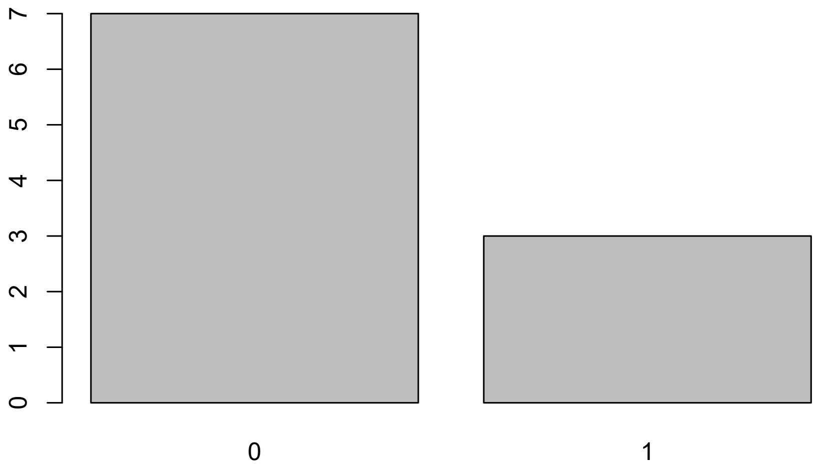

FIGURE 3.1 Summary bar plot of coin tosses.

rate (probability of being harvested), tag reporting rate (probability of angler returning a tag), or survival rate of adult fish in a lake (probability of surviving from year i to i+1)? These are the kinds of parameters that a field biologist might need to estimate. In the chapters that follow, we will examine some methods for collecting the data and estimating those probabilities.

We can generate simulated observations from a Bernoulli distribution using the `rbinom()` function:

```
N.obs <- 30
p <- 0.4
y <- rbinom(n=N.obs, prob=p, size=1)
Counts <- table(y)  # Summary table of 0 vs 1
barplot(Counts)
```

The `rbinom()` function generates values from a binomial distribution (described in Section 3.1.2). If we set the trial size to 1 (e.g., a single coin toss), we simulate draws from a Bernoulli distribution. The other two arguments in the `rbinom()` function are how many Bernoulli trials to simulate (n) and the probability of success (prob). How many values of 1 would you expect the vector y to contain in a vector of 30 replicate observations? As above, the final two lines can be used to count 0s and 1s and view a plot for these simulated data.

One of the most important benefits of using simulation is the ability to examine results from large sample sizes. For example, the following code shows that the `rbinom()` function produces an outcome very close to the expected proportion of 0s and 1s:

```r
rm(list=ls()) # Clear Environment

N.obs <- 1000
p <- 0.4
y <- rbinom(n=N.obs, prob=p, size=1)
SumOnes <- sum(y)
SumOnes
SumOnes/N.obs # Proportion of trials that are successful
Counts <- table(y)
barplot(Counts)
```

The first line of code just clears the Environment, ensuring that we start from a clean slate. The `sum()` function provides a total for the y vector, which is equal to the number of successful trials. Dividing by the number of observations is an empirical estimate of p. The plot (Plot window) shows the result for one simulation (with N.obs replicates). How close to the expected count of 1s (400) did you observe? How much does the result vary when you do multiple simulations? (Highlight all the statements and click Run several times to see the variation.)

3.1.2 Binomial

The binomial distribution is simply the combined total from multiple Bernoulli trials. One physical example would be the number of heads in five tosses of a coin. Each toss is an independent Bernoulli trial, but we record the total number of successes (heads). We could obtain anything between 0 and 5 heads in a single trial. We assume that the probability of success (p) is constant for each Bernoulli trial (e.g., each coin toss).

One fisheries example of a binomially distributed process would be the number of recaptures in a two-sample mark-recapture study. In that case, the number of fish in the second sample is the size of the trial, and fraction of the population that is marked is the probability p. Another fisheries example would be the number of tag returns in a study to estimate the exploitation rate (rate of harvest). The number of tagged fish is the size of the trial and the exploitation rate is p. A telemetry study to estimate fish survival rate is a third example. The initial number of fish with transmitters is the size of the trial. The survival rate is p and the number of survivors at the start of the next period is binomially distributed.

Let's start with a physical example of drawing five playing cards (with replacement) and recording the number of clubs. The probability of success in this case (drawing a club) is 0.25 and there are six possible outcomes (0-5 clubs). The expected number of clubs is 1.25 (trial size * p).

```r
rm(list=ls()) # Clear Environment

Clubs <-c(0, 0, 3, 3, 1, 1)  # Drawing five cards, with replacement
N.trials <- length(Clubs) # Function for getting number of trials
Counts <- table(Clubs)
barplot(Counts)
barplot(Counts, xlab="Number of clubs")
```

Again we let R count the number of trials (6 in this case) and use the `table()` function to summarize the outcomes. Our sample size is small and the plot only shows the three observed outcomes. The plot can be made more understandable (especially by someone else) by adding a plot option (xlab) to label the x axis. Doing more than one plot produces a sequence of plots which you can view by using the blue left and right arrows in the Plot window. Practice modifying the above code by adding a label for the y axis (e.g., "Frequency"). You can find the full list of bar plot options by entering "barplot" in the search box in the RStudio help window (or simply type ?barplot) in the Console).

We can alleviate our small sample size issue by turning to a simulation version. We again use the `rbinom()` function from the Bernoulli example, but now size is greater than 1:

```r
rm(list=ls()) # Clear Environment

N.trials <- 30
p <- 0.25 # Probability of drawing a club
Trial.size <- 5
Clubs <- rbinom(n=N.trials, prob=p, size=Trial.size)
Clubs
Counts <- table(Clubs)
barplot(Counts)
```

Each of the thirty observations now represents a trial size of 5 (in total, a simulation equivalent to 150 Bernoulli trials). Only part of the Clubs vector is visible in the Environment window but we can put Clubs as a line of code to print the whole vector in the Console. We again summarize the counts and plot the results using the `barplot()` function. Highlight the code in the Source window and run it multiple times to get some insight into the variability of the simulated system. For example, how often do you observe 0 or 5 Clubs? Which outcome(s) occurs most often (mode) and how frequently does the mode change?

We would expect to observe very few trials with 5 successes, because the probability is quite low (0.25^5=0.001). We can calculate it in the Console, by entering the expression 0.25^5. There is a higher probability of 0 successes (0.24), which can be calculated as 0.75^5 or $(1\text{-p})^5$, because 1-p is the probability of failure (not getting a club). It is more involved to calculate the probabilities of 1-4 successes because they can be obtained multiple ways (e.g., 1 success can be obtained five ways (10000, 01000, 00100, 00010, 00001). We could calculate those probabilities using the binomial probability distribution: $f(k) = \binom{n}{k}p^k(1-p)^{n-k}$, where k is the number of successes in a trial of size n. R code for this calculation uses the `choose()` function; for example, `choose(Trial.size,0) * p^0 * (1-p)^5` calculates the probability of 0 clubs in five draws. We could get an approximate ("brute force") estimate of the probabilities by increasing N.trials to a large value (say 10,000) and dividing the Counts vector by N.trials to estimate at the proportion of trials resulting in 0, 1,...5 successes:

```r
rm(list=ls()) # Clear Environment

N.trials <- 10000
p <- 0.25 # Probability of drawing a club
Trial.size <- 5
Clubs <- rbinom(n=N.trials, prob=p, size=Trial.size)
#Clubs
Counts <- table(Clubs)
barplot(Counts)
Counts/N.trials # Vector division - probabilities of 0-5 Clubs
```

Note that we used a pound sign to turn off ("comment out") the line for printing the Clubs vector to the Console, and print to the Console the estimated probabilities. The mode of our simulated distribution is in agreement with the expected value (trial size * p), as are the estimated probabilities for 0 and 5 clubs.

3.1.3 Multinomial

The Bernoulli and binomial distributions return a single value: success or failure for the Bernoulli and the number of successes (out of a specified trial size) for the binomial. The multinomial distribution extends that to a vector of results. A classic physical example would be to roll a die several times and record how many rolls were a 1, 2, …, 6. A fisheries example would be to collect a sample of fish and determine how many are age 1, 2,… The focus in that example is on estimating the age distribution, or proportions of fish by age. An age distribution contains valuable information about the rates of mortality and recruitment (year-class strength). Another fisheries example would be to collect a sample of fish (e.g., from electrofishing or a trawl) and determine the number caught by species. Species composition can be an important indicator of the ecological health of a study site (e.g. if there is a high relative abundance of species considered tolerant of poor water quality).

Let's begin with a hands-on example. I roll a die ten times, and record as a vector the number of times I get a 1, 2, …, 6:

```
rm(list=ls()) # Clear Environment

x <-c(2,3,3,0,2,0)  # Generated using die, equal cell probabilities
SampleSize <- sum(x)
barplot(x)
```

Here R calculates the sample size (SampleSize), even though we know in this instance that there were 10 rolls. The `barplot()` function generates the plot but without any labels is not very appealing. We can dress it up a bit by adding some options (Figure 3.2). We first create a vector with values 1-6 to serve as x-axis labels (`seq()` function, from 1 to 6 by 1). Then we use the names.arg option to add the labels, and add x and y axis titles using xlab and ylab respectively.

```
x.labels <- seq(from=1, to=6, by=1)  # x axis labels for plotting
barplot(x, names.arg=x.labels, xlab="Die side", ylab="Number of rolls")
```

We can simulate a multinomial distribution as follows:

```
rm(list=ls()) # Clear Environment

SampleSize <- 30  # e.g., sample of 30 fish assigned by age
TrueP <- c(0.1, 0.3, 0.4, 0.1, 0.05, 0.05) # probability of being ages 1-6
Reps <- 1 # How many replicate multinomial trials to carry out

# Simulate a single trial using rmultinom function
rmultinom(n=Reps, prob=TrueP, size=SampleSize) # Prints matrix to Console
```

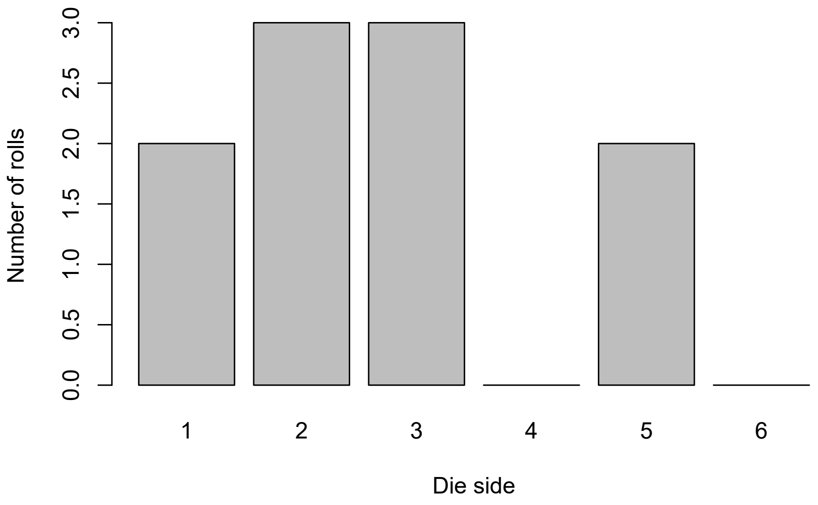

FIGURE 3.2 Summary bar plot of die rolls.

```r
x <- rmultinom(n=Reps, prob=TrueP, size=SampleSize)
# x contains matrix of counts (one column per Rep)
x.vec <- as.vector(t(x)) # Transpose matrix then convert to vector

age.vec <- seq(1,6, by=1)
barplot(x.vec, names.arg=age.vec, xlab="Age", ylab="Number of fish")
```

In this simulation, the age proportions (TrueP) are arbitrarily chosen (but could be based on field data). Run statements one-by-one to see how each line of code works. Note that the `rmultinom()` function returns a matrix with six rows and one column (n determines how many columns are returned). Rather than a column, we want a row vector, with the number of fish by age in successive columns. We use the `t()` (transpose) function to transpose the matrix and the `as.vector()` function to convert the matrix (with one row) into a row vector. We could have done this transformation in two steps but instead have nested the two functions. That results in more compact code but it is always a fine option to do the steps separately for greater clarity.

Run the code beginning with the `x <- rmultinom()` line several times to see how much the age distribution varies from one random sample to the next. The underlying age proportions are fixed, but the counts vary quite a bit for this small sample size. This is an example of how simulation is helpful in gaining experience and intuition in judging pure sampling variation versus a real biological result (e.g. good or bad year-class strength).

3.1.4 Poisson

Like the Bernoulli, binomial, and multinomial distributions, the Poisson and negative binomial distributions are used for counts (non-negative whole numbers). The first three cases have an upper bound (1 for the Bernoulli, trial size for the next two) whereas the Poisson and negative binomial distributions do not (in theory, if not in practical terms). The Poisson is simpler than the negative binomial in that it is described by a single parameter, usually λ, which represents the mean and variance of the distribution. A fisheries example of a Poisson distribution could be the number of fish per day moving upstream via a fish ladder. A model for this process could be useful for designing new fish ladders to have an appropriate capacity. Another example could be the number of fish caught per trawl tow. In this case, emphasis might be placed on how trawl catches vary from year to year, as an indication of population status.

Our simulation uses an arbitrary value for λ:

```r
rm(list=ls()) # Clear Environment

N <- 30
lambda <- 4
Count <- rpois(n=N, lambda=lambda)
Freq <- table(Count)  # Distribution of simulated counts
barplot(Freq, main="", xlab="Count", ylab="Frequency")

mean(Count)
var(Count)
```

Run the lines of code multiple times to gain experience with the Poisson distribution. How often is the mode of the distribution equal to λ? Are the estimated mean and variance (printed to the Console) close to λ? Increase the sample size to get a smooth distribution and reliable estimates of the two sample statistics.

It is also useful to run the code using other values for λ. What value for λ causes the mode to shift to 0? At what value does the sample distribution look like a bell curve (i.e., symmetrical)? One of the key benefits of simulation is that we can easily try different scenarios to build understanding about the system being modeled.

The plot can sometimes be misleading because it only includes observed levels. We can modify the code to include levels with a frequency of zero:

```r
rm(list=ls()) # Clear Environment

N <- 30
lambda <- 4
Count <- rpois(n=N, lambda=lambda) # Counts drawn from Poisson distribution
Freq <- table(factor(Count, levels = 0:max(Count)))
barplot(Freq, main="", xlab="Count", ylab="Frequency")

mean(Count)
var(Count)
```

Now the `table()` function operates on counts transformed into factors (categories or levels) by the `factor()` function. The levels= argument forces `factor()` to use a range of levels from 0 to `max(Count)`, in order to include levels with a frequency of zero. Try the following partial code in the Console to see the levels ("0", "1", etc) created by the `factor()` function: `myfact <- factor(Count, levels = 0:max(Count))`.

3.1.5 Negative binomial

As noted above, the negative binomial distribution is used in similar situations to the Poisson except that it has two parameters and is therefore more flexible. This is particularly helpful for ecological data, where counts are often more variable than would be expected under a Poisson distribution (Bolker, 2008; Link and Barker, 2009). The R function for generating negative binomial random variates (`rnbinom()`) can be parameterized in different ways; we use the "ecological" parameterization recommended by Bolker (2008). This uses the mean (termed μ) but not the variance, instead using an overdispersion parameter k. Lower values of k result in a more heterogeneous distribution, and in ecological settings, k is often less than the mean (Bolker, 2008). The distribution approximates a Poisson when k is large. The code includes calculation of the variance as a function of mu and k (Bolker, 2008). Alternatively the mean and variance (V) can be specified and used to solve for k: $k = \mu^2/(V - \mu)$.

```r
rm(list=ls()) # Clear Environment

N <- 30
mu <- 4
k=1
variance <- mu+(mu^2)/k
Count <- rnbinom(n=N, mu=mu, size=k)
```

```
Freq <- table(factor(Count, levels = 0:max(Count)))
barplot(Freq, main="", xlab="Count", ylab="Frequency")

mean(Count)
var(Count)
```

This distribution has the same mean as in the Poisson example but tends to have a longer tail (occasional large counts). Rerun the code using different values for k; for example, Figure 3.3 uses a sample size of 200 to compare results for k=1 and k=10, each with μ=4.

```
N <- 200
mu <- 4
k=1
Count <- rnbinom(n=N, mu=mu, size=k)
Freq <- table(factor(Count, levels = 0:max(Count)))
barplot(Freq, main="", xlab="Count", ylab="Frequency")

k=10 # Reduce degree of overdispersion
Count <- rnbinom(n=N, mu=mu, size=k)
Freq <- table(factor(Count, levels = 0:max(Count)))
barplot(Freq, main="", xlab="Count", ylab="Frequency")
```

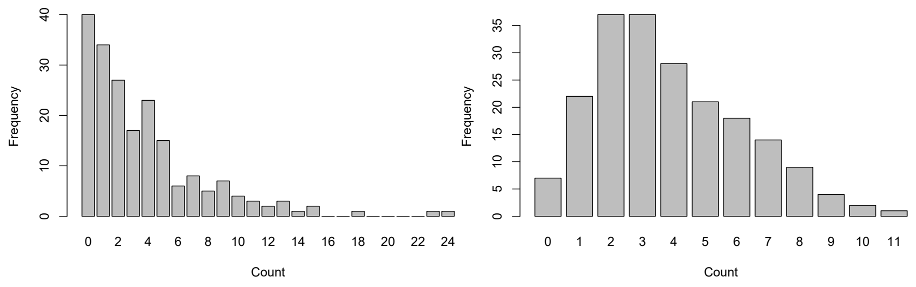

FIGURE 3.3 Negative binomial frequency distributions for k=1 (left) and 10 (right).

3.2 Continuous

3.2.1 Uniform

A uniform distribution (bounds a, b) is flat in shape, indicating that all values between a lower and upper bound are considered equally likely. It is also useful as a "first" model for situations where relatively little is known other than the lower and upper bounds (Law and Kelton, 1982). Probabilities have default bounds of 0 and 1, so a fish population model might use a uniform 0-1 distribution for probabilities such as survival rate. More narrow bounds could be used in cases where prior information was available; for example, a pilot study might indicate that the tag-reporting rate would be expected to vary between 0.4 and 0.6. Another fisheries example would be a uniform distribution between 60 and 70, for simulating individual variation in maximum size for a growth curve.

We begin with a simulation example, using the default range of 0-1 for the `runif()` function:

```r
rm(list=ls()) # Clear Environment

Reps <- 30 # Replicate draws from a uniform distribution
p <- runif(n=Reps)# Default range of 0-1. Can specify using min, max
hist(p, breaks=5, main="")
```

Try running the code several times and note the variation in pattern from one simulation run to the next. We use the breaks=5 option in the `hist()` function so that the histogram bin intervals are stable among runs. Specifying a null character string for "main=" prevents the plot from including a main title. Next, increase the sample size and note the increased smoothness of the observed distribution. What would be the expected mean of the distribution? Estimate the mean and compare the observed and expected value among runs.

As noted above, the `runif()` function allows for a uniform distribution using any lower and upper bounds. For example, we might simulate uncertainty about an exploitation rate (fraction of the population that is harvested) by allowing randomly drawn values in a specified interval; for example, `runif(n=Reps, min=0.3, max=0.4)`. The `runif()` function is not constrained to positive values; for example, it could be used with bounds -0.5 and 0.5 to simulate uncertainty about a parameter that is close to 0 but can be positive or negative. The chapters that follow will include many examples where a uniform distribution is relevant for fisheries modeling.

3.2.2 Beta

The beta distribution has a lower bound of 0 and an upper bound of 1, so it is useful for modeling parameters with those bounds (i.e., probabilities). A fisheries model might use a beta distribution to simulate variation in a survival probability or the probability of a tag being returned. The uniform distribution (Section 3.2.1) can also be used for probabilities, but only for the case where all values are equally likely. The beta distribution is more flexible and can not only be flat but also u-shaped, bell-shaped, or left- or right-skewed. Figure 3.4 shows sample distributions for shape parameters α=1 and β=0.5, 1, and 5.

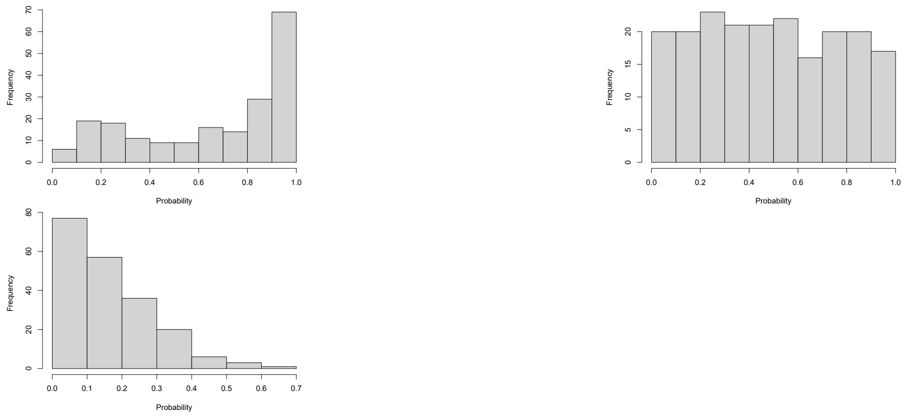

FIGURE 3.4 Histogram for 200 random draws from a beta distribution, with alpha=1 and beta=0.5 (left), 1 (center), and 5 (right).

We simulate draws from a beta distribution using the `rbeta()` function. Starting with α and β set to 1 would be appropriate for a situation where all probabilities are equally likely:

```r
rm(list=ls()) # Clear Environment

Reps <- 200 # Replicate draws from a beta distribution
alpha <- 1
beta <- 1
x <- rbeta(n=Reps, shape1=alpha, shape2=beta)
hist(x, main="", xlab="Probability")
```

After running the above code multiple times, try a beta(8, 4) distribution. These parameter values could result from a pilot study with seven successes in ten trials (α=7+1, β=3+1). This shifts the distribution toward a mode of 0.7, which is the observed proportion of successes in ten trials. Try other arbitrary values for the two parameters to vary the shape of the distribution. Can you think of a situation where probabilities would be expected to be close to 0 or close to 1?

3.2.3 Normal

The normal distribution is the well-known bell curve from traditional statistics courses. The distribution is unbounded (bounds -∞, ∞). The shape of the distribution is determined by the mean (location) and standard deviation (spread). A fisheries example might be a simulation where size-at-age varied among individuals according to a normal distribution.

Let's look at simulation code for length of individual fish, assuming a mean of 30 and standard deviation of 2:

```r
rm(list=ls()) # Clear Environment

Reps <- 30 # Replicate draws from a normal distribution
Len <- rnorm(n=Reps, mean=30, sd=2)
hist(Len, main="")
```

Run the code a few times to look at the variation in shape and bounds at a relatively small sample size. Next, increase the sample size and determine through trial and error the smallest sample size that provides a relatively smooth bell-curve shape. Estimate the mean and standard deviation (function `sd()`) and compare the observed and true values.

3.2.4 Lognormal

The lognormal distribution (bounds 0, ∞) is useful in ecological settings because it excludes negative values and allows for a long upper tail. A classic fisheries example of a lognormal distribution is annual recruitment for marine fish species (number of young entering the population). Most years result in low recruitment but there are occasional extreme values when environmental conditions are right.

We begin with simulation code:

```r
rm(list=ls()) # Clear Environment
```

```r
Reps <- 200
ln.m <- 0 # Ln-scale mean
ln.v <- 0.5 # Ln-scale variance
y <- rlnorm(n=Reps, mean=ln.m, sd=sqrt(ln.v))

hist(y, main="Lognormal distribution")
Exp.mean <- exp(ln.m+(ln.v/2)) # Calculate arithmetic scale expected mean
abline(v=Exp.mean, col="red")
Exp.var <- (exp(2*ln.m+ln.v))*(exp(ln.v)-1) # Arithmetic scale expected variance
mean(y)
var(y)
```

The `rlnorm()` function generates random lognormal values, using an arbitrarily chosen mean and standard deviation specified in natural log-scale. Those values can be used to calculate the expected mean and variance in arithmetic scale. The `abline()` function can be used to add to the histogram a vertical (v=) line for the expected arithmetic-scale mean (Figure 3.5). The line color is specified as col="red"; other options including line weight and pattern can be found in the Help window. Compare the estimates of arithmetic-scale mean and variance for your (relatively large) sample of lognormal random variates to the expected values (visible in the Environment window).

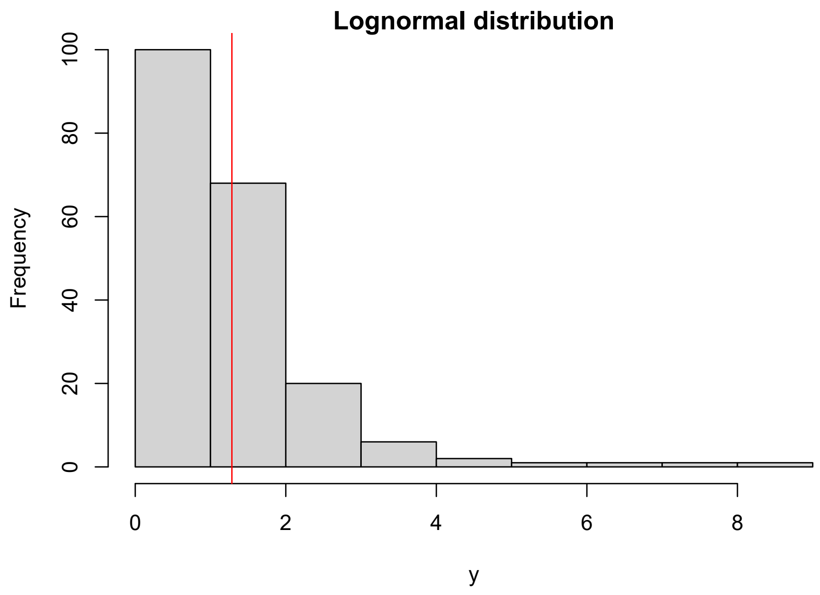

FIGURE 3.5 Histogram for 200 lognormally-distributed values, with ln-scale mean of 0 and variance of 0.5. Red vertical line denotes expected mean.

3.2.5 Gamma

The gamma distribution is similar to the lognormal in that has a lower bound of 0 and an upper bound of ∞. Thus it also excludes negative values and allows for a long upper tail. This distribution is often used to simulate waiting times (how long until some number of events has occurred, based on a specified average time between events). We consider it here simply because it can take on a variety of shapes depending on its parameters (Bolker, 2008). Figure 3.6 shows sample distributions for shape parameter α=3 and scale or spread parameter β=1 and 3.

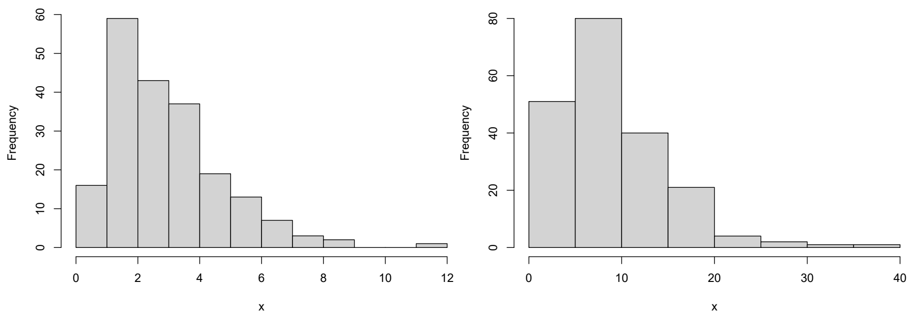

FIGURE 3.6 Histogram for 200 gamma-distributed values, with alpha=3 and beta=1 (left) and 3 (right).

We simulate draws from a gamma distribution using the `rgamma()` function, with arbitrary values for α and β:

```
rm(list=ls()) # Clear Environment

Reps <- 200 # Replicate draws from a gamma distribution
alpha <- 3 # Shape parameter, > 0
beta <- 3 # Scale parameter, > 0
x <- rgamma(n=Reps, shape=alpha, scale=beta)
hist(x, main="")
Exp.mean <- alpha*beta
Exp.var <- alpha*beta^2
mean(x)
var(x)
```

A sample size of 200 replicates produces a relatively smooth pattern for the distribution. How close are the estimated mean and variance to the expected values for this sample size? Try adjusting the two parameters to vary the shape of the distribution. What values for α and β produce a right-skewed distribution (long right tail), or a pattern somewhat like a normal distribution (bell-curve)?

3.3 Exercises

1. Use a physical process (i.e., not simulated) to generate twenty Bernoulli observations with probability of success 0.25. Include in your code comments describing how you obtained the data. Produce a summary plot of the total number of 0s and 1s. How does your observed number of successes compare to what you would expect?
2. Use a physical process to generate ten binomially distributed observations, with a trial size of at least four. What is the probability of success and the expected number of successes per trial? Produce a histogram showing the frequencies for different numbers of success per trial.
3. Use a physical process to generate 10 outcomes from a multinomial process with at least three possible outcomes. Describe in your code comments the physical process for obtaining the data and include a plot of the frequencies. How do the frequencies compare to what you would expect?
4. Assume that the following vector contains trawl catches: y=c(7, 5, 1, 5, 0, 3, 8, 2, 14, 5, 0, 6, 9, 1, 2, 2, 1, 1, 0, 9, 2, 2, 4, 11, 1, 1, 7, 3, 1, 1). Based on your analysis (including plot), were these data likely generated from a Poisson or negative binomial distribution?
5. Provide simulation code for generating normally-distributed length data for fish ages 1 (n.1=40, age-1 mean 30, sd 5) and 2 (n.2=25, age-2 mean=50, sd=7). Plot the combined length sample. In multiple simulations using these default values, how frequently is the length distribution visibly bimodal?

4 Model Fitting

4.1 Introduction

What is a model in a fisheries context? It typically means one or more equations that describe a biological process of interest. There are two good reasons for developing a model. A model that simulates a field study can be used for planning purposes, before field work is conducted. Such a model might be used to determine the number of fish to tag or the level of electrofishing effort, in order for the field work to produce a useful result. A model can also be used to test an analytical approach. For example, we could generate simulated data and see whether our analysis agrees with the known true value(s). The other main use of a model is to analyze data from field work. One example from Chapter 2 is a two-sample mark-recapture study to estimate population size. Fish in the first sample are tagged, and the fraction of marked fish in the second sample provides the information needed to estimate population size. The model in that case is the equation for population size (N_{hat}), along with some assumptions about sampling. The equation is based on the assumption that the marked fraction of the second sample should (on average) be the same as the marked fraction in the underlying population. This method also requires some other assumptions that are more practical in nature, to help ensure that our field methods approximate the underlying model structure. For example, we assume that tagged and untagged fish are well mixed. In practice, this means that tagging effort should be well distributed throughout the study area so that our recapture effort does not encounter pockets with too many or too few tagged fish. We also assume that tags are not shed and that there are no deaths due to tagging. The mark-recapture model assumes that we know how many fish are in the tagged subpopulation. Tag loss or tagging mortality would affect that number and result in a biased estimate.

4.2 Using simulation

The equation for a two-sample mark-recapture study provides a point estimate of population size, but we could consider an improved version of the model that would allow us to more fully characterize the study results. For example, we could assume that the number of marked fish in the second sample comes from a binomial distribution. That distributional assumption allows us to generate estimates of uncertainty as well as the point estimate. The key to this sort of extended model is to think about the process generating the field data and determine which statistical distribution is appropriate. Let's start with a simulation then consider an alternative (Bayesian) approach.

```r
rm(list=ls()) # Clear Environment

# Two-sample population estimate
N.true <- 400  # Arbitrary assumed population size
p.true <- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true * p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample
m2 <- n2 * p.true # Number of marked fish in second sample
N.hat <- n1*n2/m2 # Store estimate as variable N_hat
```

This example uses the same estimator as in Chapter 2. The two samples could be fixed values and could differ, but here we assume they are equal in size, based on the capture probability (p.true). The number of recaptures in the second sample equals the expected number based on the fraction of the population that is marked (p.true). Because n_1, n_2, and m_2 take on their expected values based on p.true, the estimate (N_{hat}) equals the true value (see Environment window).

Now, let's extend the model by introducing some randomness (stochasticity) in the number of recaptures. We assume that recaptures are the successes in a binomial trial. The size of the binomial trial is n_2. The probability of success equals p.true here but we calculate it as n_1/N.true in case n_1 and n_2 were given other fixed values.

```r
# Extend model to allow for stochasticity
Reps <- 1000
m2.vec <- rbinom(n=Reps, prob=n1/N.true, size=n2)
N.hat <- n1 * n2 / m2.vec
```

The rbinom() function produces a vector of recaptures, simulating a mark-recapture study repeated Reps times. Using that vector in the equation for N_{hat} now produces a vector of population estimates. Plotting those results (Figure 4.1) shows that estimates are generally close to the true value but tend to have a long tail to the right (overestimates N_{hat} when a low value is drawn for the number of recaptures).

```
hist(m2.vec, xlab="Number of recaptures", main="")
hist(N.hat, xlab="Population estimate", main="")
```

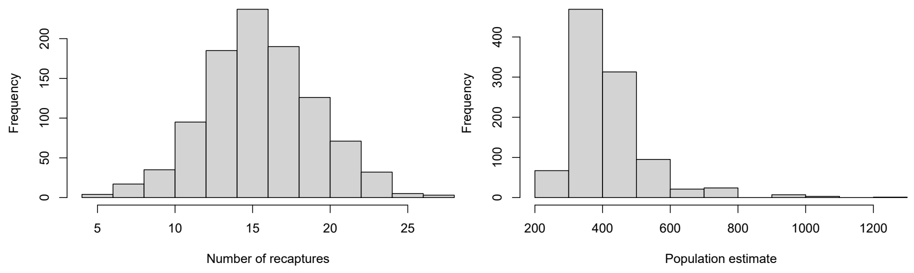

FIGURE 4.1 Frequency distributions for number of recaptured fish (left) and estimated population size (right), based on simulation using a binomial distribution for the number of recaptures.

```
mean(N.hat)
quantile(N.hat, probs=c(0.025, 0.5, 0.975), na.rm=TRUE)
# quantile function provides empirical confidence bounds and median
```

Results printed to the Console show that the mean is higher than the true value (due to the tail of extreme values) but the median is very close to N.true. The estimated median and 95% bounds are obtained using the quantile() function. We provide a vector of probabilities and the function returns the estimates at those points. The 0.025 and 0.975 points provide the empirical 95% bounds, and the 0.5 point is the median. There are less biased modifications to the basic two-sample estimator but this version is more intuitive and sufficient for our purposes. When interpreting a field study of this sort, the confidence bounds and overall shape of the distribution should be given more emphasis than the point estimate (which reduces concern about any statistical bias of the estimator).

4.3 Using Bayesian methods

In Section **??**, we used a simulation-based or numerical approach to estimate confidence limits for the mark-recapture study design. This is an example of parametric bootstrapping (**?**), where replicate samples (number of recaptured fish in this case) are drawn from a parametric distribution, in this case binomial. We can compare this simulation approach to a formal statistical method, using a Bayesian framework. We use the same binomial distribution to describe the process of obtaining recaptures. Our results will be similar, and hopefully the comparison will provide some intuition about how a Bayesian analysis works. Our use of simulation in the remainder of the book will be limited to generating "sample" data. Model fitting will be done using Bayesian methods that work well for simple or complex models.

The two paradigms for statistical methods are frequentist and Bayesian. Frequentist methods are the classical approach for estimating model parameters and making inferences (**?**). In a frequentist framework, model parameters are assumed to be fixed but unknown values. For example, we could conduct a two-sample mark-recapture study to estimate population size in a lake. If we repeated the mark-recapture study many times, we would expect the 95% confidence limit for each estimate to contain the true population size 95 percent of the time (hence the term frequentist).

A Bayesian analysis is based on the data in hand and does not rely on the idea of hypothetical replicate studies (McCarthy, 2007; **?**). The results are exact for any sample size (**?**), which is an important advantage for fisheries studies because sample sizes are often small. Bayesian statistics also differs from frequentist statistics in that it takes into account what is known before the study is conducted (McCarthy, 2007). The requirement to specify prior knowledge can be viewed as either an advantage or disadvantage, depending on the availability of prior data and one's statistical philosophy (Ellison, 2004; McCarthy, 2007; **?**; **?**; Dorazio, 2016; **?**). The prior information takes the form of a statistical distribution that defines the range and relative likelihood of possible values (prior to collecting the new data). The prior distribution can be informative or non-informative. For example, a uniform 0-1 distribution would be a non-informative prior for a tag-reporting rate, because probabilities are by definition constrained to be between 0 and 1. Results from a pilot study might allow for an informative prior distribution; for example, a uniform distribution between 0.2 and 0.4.

When prior information is available, there can be uncertainty about whether it is still valid. For example, the tag reporting rate might now be lower than when a pilot study was done because of increased angler dissatisfaction with the current management approach. Another example could be whether to use growth information from a prior year when population size was much higher. The big advantage of using prior information is that it can improve the precision (and sometimes accuracy) of study results (e.g., ?).

Bayes' rule provides the formal structure for combining the prior information and new data in order to estimate the posterior distributionDistribution, posterior). For our purposes, it is sufficient to know that the posterior distribution is proportional to the product of the likelihood (based on the new data; Section 4.3.3) and the prior distribution:

$posterior \propto likelihood \cdot prior$

The details in applying Bayes' rule are handled automatically by the software used in this book. Nevertheless, knowing this relationship helps in understanding how the two components work together. Being forced to specify the prior distribution makes a Bayesian analysis very transparent (?). Anyone reviewing the results can decide about the appropriateness of the prior distribution(s). This would be particularly important in cases where a prior distribution was based on belief (e.g., expert opinion) rather than prior studies. The ideal situation is for the likelihood term to dominate, which is often referred to as the "data overwhelming the prior." This would mean that the sample size for new data was sufficiently large that the prior had little or no influence on results.

For most analyses in this book, we follow the approach of previous authors in using non-informative (flat or vague) prior distributions [royle.dorazio_2009; ?]. The simplest example is using a uniform(0,1) distribution for probabilities. In cases where choosing a non-informative prior distribution is more subjective (e.g., average maximum size for a growth curve), the analysis can be repeated using narrower and wider priors to ensure that results are insensitive to the bounds (?). Typically the results we obtain will be very similar to results of a frequentist analysis (McCarthy, 2007; ?; ?). In situations where an informative prior distribution can be justified, it is straightforward to include that in the Bayesian analysis (see Section x.x (****)).

It is reasonable to ask why a Bayesian approach would be recommended, if we are going to avoid informative prior distributions? The answer is a practical one – to take advantage of the "BUGS" family of software (Lunn et al., 2000, 2009). This open-source software began with BUGS, followed by WinBUGS (for Windows PCs) then OpenBUGS (intended for multiple operating systems). There are several newer sources of Bayesian software that use the same or very similar code. Here we use R packages for JAGS (Plummer, 2003), a popular and well-supported version. The code should transfer with little or no modification to other BUGS-family software versions.

The following code provides for a Bayesian model fit to the mark-recapture data:

```
rm(list=ls()) # Clear Environment

# Arbitrary 'observed' values for analysis
N.true <- 400  # Population size
p.true <- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true * p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample
m2 <- n2 * p.true # Number of marked fish in second sample

# Load necessary library packages
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# JAGS code
sink("TwoSampleCR.txt")
cat("
    model {

  # Prior
    N.hat ~ dlnorm(0, 1E-6) # Non-informative prior (N.hat>0)
    MarkedFraction <- n1/N.hat

  # Likelihood
    # Binomial distribution for observed recaptures
    m2 ~ dbin(MarkedFraction, n2)
    }

    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("n1", "n2", "m2")

# Initial values.
jags.inits <- function(){ list(N.hat=runif(n=1, min=max(n1,n2), max=20000))}

model.file <- 'TwoSampleCR.txt'

# Parameters monitored
jags.params <- c("N.hat", "MarkedFraction")

# Call JAGS from R
jagsfit <- jags(data=jags.data, inits=jags.inits, jags.params,
                n.chains = 3, n.thin = 1, n.iter = 2000, n.burnin = 1000,
                model.file)
print(jagsfit, digits=3)
plot(jagsfit)
```

The first section of code sets up the experiment. The "observed" values for n_1, n_2 and m_2 could be replaced by real data from a field study; in that case, N would be unknown.

Before the first time of running JAGS code, you will need to download and install JAGS[1]. Next, use RStudio to install the packages for running JAGS. There are several options; this book uses **rjags** (**?**) and **R2jags** (**?**). Both of these steps are done only once. Then each time that you want to access JAGS from R, you load the two packages using the library() function.

The cat() function concatenates (merges together) the lines of JAGS code, which are written out as a text file using the sink() function. The name of the external file is arbitrary but just needs to match the model.file code further below. It can be helpful to use an informative name for the text file rather than something generic that might get overwritten or be hard to locate at a later date. The JAGS code can be thought of as consisting of two main parts: prior information and analysis of the new data. These two parts can come in any order but it seems logical to put the prior information first. In this case, our prior distribution is an uninformative lognormal (McCarthy, 2007) for the population estimate N.hat. This ensures that the population estimate cannot be negative. The analysis of new data uses the likelihood (see Section 4.3.3) and is calculated in the final line of JAGS code. The observed number of recaptures is assumed to be binomally distributed. The parameter N.hat is used to estimate the fraction of the population that is marked, which is the probability for the binomial function dbin().

The next few lines of R code provide settings for the JAGS analysis (data to pass in, initial values for parameters, file name for the JAGS code, parameters to return). JAGS is sometimes tolerant of a wide range of initial values, but in this case, we need to provide a large enough initial value to avoid a run-time JAGS error. Population size must be at least as large (and likely much larger than) the sizes of the two samples, so for simplicity we use a broad uniform prior distribution ranging from max(n1,n2) to an arbitrarily large value (20,000). Parameters (true model parameters and functions of those parameters) for which we want JAGS to monitor and return results are listed in jags.params. The function call to jags() links to the data, initial values, etc., and provides some settings for the estimation process.

BUGS software uses an iterative method of parameter estimation called Markov Chain Monte Carlo or MCMC (McCarthy, 2007). The MCMC method

[1] https://sourceforge.net/projects/mcmc-jags/files/

produces a sequence of autocorrelated estimates. Each sequence or chain uses a different stream of pseudo-random numbers, so having multiple independent chains is necessary for judging convergence. Three chains is a good practical choice. Thinning (e.g., n.thin=10 to retain only every tenth MCMC result) is sometimes done to reduce the autocorrelation, but is not generally necessary (**?**) and is not done here. The number of MCMC iterations is arbitrary but it is better to err on the side of having too many (and can provide a valuable opportunity to stretch legs or get coffee). R returns a measure of convergence (Rhat) that can be useful in deciding if a larger number of iterations is needed. Rhat values less than 1.05 can indicate acceptable convergence (**?**), with values closer to 1 being preferred. More details about judging convergence are in Section 4.3.2. The final MCMC setting is for the "burn-in" phase, where initial values are discarded so that the retained values are considered to be unaffected by the initial values. The default if n.burnin is not specified is to discard half the total number of updates. The best way to develop experience and intuition about MCMC settings (and Bayesian analysis in general) is to run and re-run code using different settings. For example, are the results different if n.iter is set to 10000?

Results for this case are contained in the jagsfit object, which can be printed to the Console and viewed as plots. Summary statistics are produced for the two monitored parameters and deviance (See Section 4.3.4), which is a measure of fit. Convergence was very good based on estimated Rhat values less than 1.01. The 95% Bayesian credible interval (comparable to a frequentist confidence interval) from one run was 277 to 670, with a median of 410 and mean of 425. Your results will differ slightly because of the pseudo-randomness of the MCMC process. The credible interval was not too different from the estimate obtained through simulation (three simulation runs produced 278-711, 278-640, and 278-640). (The upper bound estimates of 640 and 711 occur with ten and nine recaptures, respectively.)

More information about Nhat can be obtained by plotting values from the jagsfit object:

```
hist(jagsfit$BUGSoutput$sims.list$N.hat, main="", xlab="Population estimates")
```

The jagsfit object is a data frame that can be inspected in the Environment window. Note that sims.list arrays contain 3000 elements, which is the combined length of 1000 retained values (2000-n.burnin) from each of three chains. (When entering a jagsfit object in the Source window or Console, typing $ after each level of the object will bring up an RStudio prompt for the next choice.)

The model results include pD, which is an estimate of the number of effective parameters in the model. In simple models, we can often count the number of nominal parameters, but the number of effective parameters may be less; for example, parameters may be correlated or constrained by an informative prior distribution (McCarthy, 2007). JAGS uses numerical methods to estimate pD. For this model, we would expect pD to be close to 1 for the estimate of N.hat; MarkedFraction is a calculated value and not a model parameter.

JAGS also provides an estimate of the Deviance Information Criterion (DIC), which is a relative measure of how well a model fits the data (McCarthy, 2007; Lunn et al., 2009). It is the Bayesian analog of the Akaike's Information Criterion (AIC) that is widely used in a frequentist setting to compare alternative models. Like the AIC, the DIC results in a trade-off between model fit (likelihood) and complexity (number of parameters) (McCarthy, 2007). The DIC score is calculated using a measure of fit (deviance at the mean of the posterior distribution) and a measure of model complexity (pD); lower values are better when comparing alternative models. DIC scores should be interpreted with caution because of the difficulty in estimating pD for some models (Lunn et al., 2009).

It is worth noting that this model may also be fit by using MarkedFraction as the model parameter. It is easy to specify the uninformative prior distribution (and initial values) in this case, as MarkedFraction is a proportion between 0 and 1. Now N.hat is a calculated value. One major advantage of Bayesian software is that we automatically get full information about parameters that are calculated functions of model parameters. In this case, the calculated value (N.hat) is of primary interest, rather than the actual model parameter MarkedFraction. Results for Nhat characterize its posterior distribution, which takes into account the prior information (none here) and the new data.

```r
rm(list=ls()) # Clear Environment

# Load necessary library packages
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# Arbitrary 'observed' values for analysis
N.true <- 400  # Population size
p.true <- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true * p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample
m2 <- n2 * p.true # Number of marked fish in second sample

# JAGS code
sink("TwoSampleCR.txt")
cat("
    model {

    # Priors
    MarkedFraction ~ dunif(0, 1)

    # Calculated value
    N.hat <- n1 /MarkedFraction

    # Likelihood
    # Binomial distribution for observed recaptures
    m2 ~ dbin(MarkedFraction, n2)
    }

    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("n1", "n2", "m2")

# Initial values.
jags.inits <- function(){ list(MarkedFraction=runif(1, min=0, max=1))}

model.file <- 'TwoSampleCR.txt'

# Parameters monitored
jags.params <- c("N.hat", "MarkedFraction")

# Call JAGS from R
jagsfit <- jags(data=jags.data, inits=jags.inits, jags.params,
                n.chains = 3, n.thin = 1, n.iter = 2000, n.burnin = 1000,
                model.file)
print(jagsfit, digits=3)
plot(jagsfit)
```

Another advantage of this way of parameterizing the model is that we avoid specifying an arbitrary upper bound for initial values for N.hat. Results of the two approaches appear to be comparable.

4.3.1 Debugging code

Anyone who develops new code for fisheries data analysis will wrestle with the inevitable coding errors, not only in JAGS but also the R code that surrounds it. Let's begin by looking at a few examples of R coding errors:

```r
rm(list=ls()) # Clear Environment

#1
x <- 2
lg(x) # Incorrect function name

#2
y <- 4
z <- x+y # Typo (should have been x*y)

#3
RandLength <- rnorm(1, 10, 2) # Correct but poorly documented
RandLength <- rnorm(10,2) # Sample size omitted
RandLength <- rnorm(1,10) # sd omitted so default used (incorrectly)
RandLength <- rnorm(n=1, mean=10, sd=2) # Correct and fully documented

#4
x2 <- rnorm(n=1000, mean=10, sd=5)
x3 <- rnorm(n=1000, mean=20, sd=10)
x4 <- x2+x3
mean(x4)
var(x4)
```

The first example is easy to resolve, as we get a helpful error message in the Console. Rechecking the code, or doing an internet search for the correct function name, are approaches for addressing this type of error. The second example, if our intent was to assign x*y to z, is much more dangerous. The code will run but will give the wrong answer because of a typing error. It is always essential to review new code, line by line. Having a colleague (fresh set of eyes) check code can also be productive. R code can often be executed line by line, and checking calculated values in the Environment window can help in finding this sort of logic error, at least for simpler analyses.

In the third example, we are simulating a random length observation, drawn from a normal distribution with a mean of 10 and standard deviation of 2. The first version of the code works correctly but is poorly documented. The second example runs but does not produce the intended result. We omitted the first argument and did not include parameter names, so R assumes incorrectly that the first argument is the sample size and the second one is the mean. The third argument is missing so R uses the default value (sd=1), which in this case is incorrect. Even if we wanted to use an sd of 1, it is better to specify it for clarity. RStudio is helpful in working with functions in that it displays the function arguments and any defaults, once the function name and "("

have been entered. The fourth version works as intended, and makes clear the values that are being used.

The fourth example is coded correctly but illustrates another way of checking code. Simulation makes it easy to use a large sample size, to check sample statistics for x4. We verify that the mean is close to the expected value (10+20). The variance of a sum equals the sum of the variances so we confirm that the sample variance is close to the expected value (25+100).

How many errors do you find in the following section of code? Review line by line and correct errors before executing the code. Once the code is working correctly, confirm that the sample median is very close to the expected population size:

```r
rm(list=ls()) # Clear Environment

# Two-sample population estimate
N.true <- 400  # Arbitrary assumed population size
p.true <-- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true X p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample

Reps <- 1000
m2.vec <- rbinom(n=Rep, prob=n1/N.true, size=n2)
N.hat <- n1 * n2 / m2.vec
hist(N.hat, xlab="Population estimate", main="")
mean(Nhat)
quantile(N.hat, probs=c(0.5), na.rm=TRUE))  # Sample median
```

Debugging JAGS code can be more challenging. There is no option for executing single lines of code as is possible with R. It is usually a matter of fixing one error at a time until the code runs, then checking and testing the code to be confident that it is working correctly. Consider the following (correctly coded) example for a tagging study to estimate the exploitation rate (fraction of the population harvested). Our simulation generates the single observation for the number of tags returned (e.g., over a year). This observation is drawn from a binomial distribution and we use the same JAGS function (dbin()) as in the population estimate examples:

```r
rm(list=ls()) # Clear Environment

# Arbitrary 'observed' values for analysis
N.tagged <- 100
```

```
Exp.rate <- 0.4
Tags.returned <- rbinom(n=1, prob=Exp.rate, size=N.tagged)

# Load necessary library packages
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# JAGS code
sink("ExpRate.txt")
cat("
    model {

    # Priors
    Exp.rate.est ~ dunif(0,1)

    # Likelihood
    Tags.returned ~ dbin(Exp.rate.est, N.tagged)
}
    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("N.tagged", "Tags.returned")

# Initial values.
jags.inits <- function(){ list(Exp.rate.est=runif(n=1, min=0, max=1))}

model.file <- 'ExpRate.txt'

# Parameters monitored
jags.params <- c("Exp.rate.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, inits=jags.inits, jags.params,
                n.chains = 3, n.thin = 1, n.iter = 2000, n.burnin = 1000,
                model.file)
print(jagsfit, digits=3)
plot(jagsfit)
```

Begin by running the code as is, then try substituting each of the following (incorrect) lines, one at a time.

- `ExpRate.est ~ dunif(0,1)`
- `Exp.rate.est ~ unif(0,1)`

- `Tags.returned ~ dbin(Exp.rate.est)`

The first error message points to line 8. JAGS correctly reports that the variable Exp.rate.est has not been previously defined. This is correct, but the error is actually on line 5, where the intended parameter name (Exp.rate.est) was mistyped. The second error is easy to diagnose because it specifies that the "unif" distribution on line 5 is unknown. The third error indicates that a discrete-valued parameter for function dbin() is missing. This makes sense because the size of the experiment (N.tagged) was omitted.

As with R code, the first step is always to review the code line by line. Fitting a model to simulated data is always useful (even if there are field observations) because the correct answer is known. Another strategy is to start with the simplest possible analysis, then add complexity one piece at a time. For example, **?** discuss Cormack-Jolly-Seber models for estimating survival, beginning with constant parameters then gradually adding more complexity (e.g., models that allow for time and individual variation).

Skill in debugging comes with experience. It is hoped that the above strategies will be useful in working through the inevitable coding errors.

4.3.2 Judging convergence

The MCMC process is iterative so it is important to ensure that the number of updates (iterations) is sufficient to achieve convergence. One practical approach is to fit the model with some arbitrary (but reasonably large) number of updates, then refit the model using larger numbers of updates. Results that are stable suggest that convergence has been achieved. The convergence statistic Rhat (**???**) is also generally reliable as a measure of convergence. It compares between- and within-chain variance; their ratio should be close to 1 once chains have converged. Especially for models for which convergence appears to be slow, it can also be helpful to view "trace" plots that show how estimates change as updating occurs (**?**). Add the following lines of code to the two-sample mark-recapture model used in Section 4.3:

```r
par(mfrow=c(3,1)) # Multi-frame plot, 3 rows, 1 col
matplot(jagsfit$BUGSoutput$sims.array[,,1], type = "l", ylab="Marked fraction")
matplot(jagsfit$BUGSoutput$sims.array[,,2], type = "l", ylab="N hat")
matplot(jagsfit$BUGSoutput$sims.array[,,3], type = "l", ylab="Deviance")

jagsfit.mcmc <- as.mcmc(jagsfit) # Creates an MCMC object for plotting
plot(jagsfit.mcmc) # Trace and density plots
```

The first approach follows an example from **?**. We use the par() graphical function to set up a multi-frame plot (by row), with three rows and one column. This results in a compact set of three plots within a single RStudio plot window. The next three lines use the built-in matplot() function to plot columns of a matrix as overlaid lines. The Environment window can be viewed to see the structure of the sims.array, a matrix of type num[1:1000, 1:3, 1:3]. The first dimension is for the 1000 retained updates, the second is for chains 1-3, and the third is for the variable (MarkedFraction, Nhat, Deviance). Omitting the ranges for updates and chains results in a plot containing all updates, with a different colored line for each chain. We could also specify a subset of the range to see the initial pattern (e.g., ...sims.array[1:100,,1], ...). For this simple model, the estimates converge almost immediately so there is no obvious transitory pattern. Note that there is a lot of wasted space and it may be necessary to increase the size of the Plots window to see the plots adequately.

The next two of code provide a more automated way of generating trace and density plots, using methods contained in the **coda** (**?**) package (loaded automatically by rjags). The as.mcmc() function converts the original output object (jagsfit) into an MCMC object. The plot function here produces a nicely formatted set of trace and density plots that are very helpful in confirming convergence. If convergence has been achieved, the trace plots should appear to be a "grassy lawn" or "fuzzy caterpillar" without any trend, with overlapping (fully mixed) chains. Density plots show the estimated posterior distributions, and given convergence, should have an essentially identical pattern for the three chains. Convergence occurs very quickly here but we will see in later chapters with more complex models that convergence can sometimes be more difficult to achieve.

4.3.3 What is a likelihood?

As mentioned above, Bayesian software uses the likelihood and prior information to estimate the posterior distribution. But what exactly is a likelihood? It is an expression or function that is used to estimate the *likelihood* of obtaining the sample observation(s), given some specific value for each model parameter. As an example, let's return to the second version of the mark-recapture analysis, with the single model parameter MarkedFracton (the fraction of the population that is marked). We use the binomial distribution to model the number of recaptures. The likelihood function for the binomial is the same as the probability distribution: $L(p|n,k) = \binom{n}{k} p^k (1-p)^{n-k}$, except that here, we estimate the likelihood of a certain value for p, given k successes in a trial of size n. We can look at likelihoods using the same parameter values as in the simulation:

```r
rm(list=ls()) # Clear Environment

# Arbitrary 'observed' values for analysis
N.true <- 400  # Population size
p.true <- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true * p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample
m2 <- n2 * p.true # Number of marked fish in second sample

# Calculate likelihood for specific value of p
p <- 0.2
choose(n2,m2) * p^m2 * (1-p)^(n2-m2) # Likelihood for single value of p

dbinom(x=m2, size=n2, prob=p) # x successes, built-in function to obtain same result

# Create vector of possible p values, determine likelihood for each
p <- seq(from=0.02, to=1, by=0.02)
l.vec <- choose(n2,m2) * p^m2 * (1-p)^(n2-m2)
par(mfrow=c(1,1)) # Reset plot frame
plot(p, l.vec, ylab="Likelihood", col="red")

# Examine change in likelihood at higher number of recaptures
new.m2 <- 18
new.l.vec <- choose(n2,new.m2) * p^new.m2 * (1-p)^(n2-new.m2)
points(p, new.l.vec, col="blue")
```

The likelihood calculation at p=0.2 uses the choose() function, to determine how many ways m2=16 recaptures can be drawn from a sample of size 80. Type `choose(n2,m2)` in the Console to see that it is a very large number! The remainder of the expression determines the probability of 16 successes and 80-16 failures. We can obtain the same result from the built-in function dbinom(), although it seems more instructive to write out the full expression.

The calculated likelihood (0.11) is really only of interest as a relative value, compared to other potential values for p. We use the seq() function to create a vector of potential p values, and obtain a vector of likelihoods from the p vector. Plotting the points shows that the likelihood peaks sharply at 0.2 (not surprisingly since that is the value that generated the 16 recaptures) and decreases to near 0 at about 0.08 and 0.32.

We have assumed thus far that we obtained the expected number of recaptures (80*p), but in reality, the number of recaptures would be a binomially-distributed random variate. A different observed value for m_2 would shift the likelihood function. The next lines of code calculate the likelihood for a slightly higher number of recaptures. The points function adds the new likelihood values to the plot, using the col="blue" plot option. The new values are shifted to the right because the number of recaptured fish was higher.

If multiple independent samples are taken, the combined likelihood is a product. For our two assumed samples of 80 fish, with 16 recaptures in one and 18 in the other, the combined likelihood at each level of p would be the product l.vec * new.l.vec.

We wrote out the expression for the likelihood for this example, but that is not needed when using the BUGS language. We need only specify the function name for the distribution describing our sample data. JAGS does the behind-the-scenes work to provide the code for the likelihood function. Working at a higher level allows us to focus on the study design and assumptions, which guide us to our choice of sample distribution.

4.3.4 Deviance as a measure of model fit

Likelihood calculations were introduced in Section 4.3.3. Here we show two related measures of model fit (log-likelihood and deviance) using the same mark-recapture example. There can be computational advantages to working with the log-likelihood compared to the likelihood (McCarthy, 2007).

```r
rm(list=ls()) # Clear Environment

# Arbitrary 'observed' values for analysis
N.true <- 400  # Population size
p.true <- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true * p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample
m2 <- n2 * p.true # Number of marked fish in second sample

# Create vector of possible p values, determine likelihood for each
p <- seq(from=0.02, to=1, by=0.02)
l.vec <- choose(n2,m2) * p^m2 * (1-p)^(n2-m2)
plot(p, l.vec, ylab="Likelihood", col="red")

log_l.vec <- log(l.vec) # Log-likelihood
plot(p, log_l.vec, ylab="log-Likelihood", col="red")
dev.vec <- -2 * log_l.vec # Deviance (-2 * ln_L)
plot(p, dev.vec, ylab="Deviance", col="red")
```

Compared to the likelihood, the log-likelihood curve has a different shape but the same peak at p=0.2 (the most likely value given 16 successes in a trial size of 80). The deviance is obtained by multiplying the log-likelihood by -2.

The plotted pattern is now inverted so that now the curve has a *minimum* at p=0.2; i.e., a larger value for deviance indicates a poorer fit (McCarthy, 2007). JAGS uses the deviance to obtain parameter estimates, but it is nice to know that we would obtain the same estimates from either the maximum likelihood or log-likelihood or the minimum deviance (McCarthy, 2007).

4.3.5 Model checking

It is easy to fit the correct model to simulated data, but that is not the case for real (field) data. Any model fit to field data should be viewed as an approximation of the underlying ecological processes. We cannot prove that the model is correct, but we can examine whether it is a useful approximation. One method for doing this is the posterior predictive check (**????**). This method compares a measure of fit for replicated data generated under the fitted model compared to observed data. If the measure of fit (e.g., sum of squares or chi-square) is markedly different (usually higher) for the observed data, that suggests that the observed data contain some features not captured by the fitted model.

Our first example uses simulated catch data from a Poisson distribution, as might be obtained by standardized electrofishing or trawling. Keep in mind that these are our "observed" catches, to be compared with replicate data obtained by simulation. We arbitrarily set the mean for the Poisson distribution at 7 and the sample size at 30. Before viewing the extended JAGS code that includes the posterior predictive check, let's examine a version that simply fits the model:

```
# Fit model to catch data from Poisson distribution

rm(list=ls()) # Clear Environment

N <- 30
mu <- 7
Catch <- rpois(n=N, lambda=mu) # Catches drawn from Poisson distribution
Freq <- table(Catch)  # Distribution of simulated catches
barplot(Freq, main="", xlab="Catch", ylab="Frequency")

# Load necessary library
library(rjags)
library(R2jags)

sink("PoissonFit.txt")
cat("
model {
```

```
# Prior
 lambda.est ~ dunif(0, 100)

# Likelihood
    for(i in 1:N) {
      Catch[i] ~ dpois(lambda.est)
      } #y
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("N", "Catch")

# Initial values
jags.inits <- function(){ list(lambda.est=runif(n=1, min=0, max=100))}

model.file <- 'PoissonFit.txt'

# Parameters monitored
jags.params <- c("lambda.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

There are only a few lines of JAGS code. We use an uninformative uniform prior distribution for the single Poisson parameter lambda.est. The catches are assumed (correctly) to be drawn from a Poisson distribution, and the likelihood calculations use the Poisson distribution function dpois(). The initial value is obtained from the same uniform distribution as the prior. JAGS returns the estimated mean, which varies around the true value due to the relatively modest sample size. Try rerunning the code using a large sample size to see the improvement in the sample catch plot and the estimated mean.

Next we add a few lines of code to carry out the posterior predictive check:

```
# Fit model to catch data from Poisson distribution
# Includes code for posterior predictive check
```

```r
rm(list=ls()) # Clear Environment

N <- 30
mu <- 7
Catch <- rpois(n=N, lambda=mu) # Counts drawn from Poisson distribution
Freq <- table(Catch)  # Distribution of simulated catches
barplot(Freq, main="", xlab="Catch", ylab="Frequency")

# Load necessary library
library(rjags)
library(R2jags)

sink("PPC_Example.txt")
cat("
model {

# Prior
 lambda.est ~ dunif(0, 100)

# Likelihood
    for(i in 1:N) {
      Catch[i] ~ dpois(lambda.est)
      Catch.new[i] ~ dpois(lambda.est) # Generating replicated new data for PPC
      chi.obs[i] <- pow(Catch[i]-lambda.est,2)/lambda.est # chi-square discrepancy for obs
      chi.new[i] <- pow(Catch.new[i]-lambda.est,2)/lambda.est # chi-square for replicate new data
      } #y
  Total.obs <- sum(chi.obs[])
  Total.new <- sum(chi.new[])
  Bayesian.p <- step(Total.new - Total.obs)
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("N", "Catch")

# Initial values
jags.inits <- function(){ list(lambda.est=runif(n=1, min=0, max=100))}

model.file <- 'PPC_Example.txt'

# Parameters monitored
jags.params <- c("lambda.est", "Bayesian.p")
```

```
# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

Within the likelihood looping section, we draw new replicate observations from the posterior distribution (Catch.new), and calculate chi-square values for observed and replicate data. The Bayesian p value is obtained using the step() function and the total chi-square values for the observed and replicate data sets. The Bayesian p values tend to be quite variable; ten trials produced values of 0.34, 0.42, 0.18, 0.16, 0.32, 0.30, 0.32, 0.67, 0.69, and 0.18. We would expect the total chi-square for observed and replicate data to be the same on average as the fitted model is known to be correct (i.e., our "observed" catches and the replicate data sets are drawn from Poisson distributions).

Next consider a case where the "observed" data are obtained from something other than a Poisson distribution. The negative binomial distribution is a good choice for ecological data because it can allow for a skewed distribution (e.g. mostly low but a few large catches). The two negative binomial parameters are mu (mean catch) and the overdispersion parameter k (Bolker, 2008). A small value for k produces more clumping or aggregation, whereas extreme values of k (>10*mu) produce a Poisson-like distribution (Bolker, 2008). Our JAGS code for model fitting remains unchanged but we replace the Poisson simulation code with the following:

```
# Fit model to negative binomial data
# Includes code for posterior predictive check

rm(list=ls()) # Clear Environment

N <- 30
mu <- 7
k=1
variance <- mu+(mu^2)/k
Catch <- rnbinom(n=N, mu=mu, size=k) # Catches drawn from negative binomial
Freq <- table(Catch)  # Distribution of simulated counts
barplot(Freq, main="", xlab="Count", ylab="Frequency")
```

Now Bayesian p values are consistently extreme (0.00 for five replicate trials). The total chi-square for the replicate data sets (which *are* Poisson distributed) is always less than for the observed data (which are not). The fitted Poisson model lacks the flexibility to mimic the skew of the observed catch distribution. More intermediate Bayesian p values are only obtained if k takes on very large values (e.g., 1000), which results in a Poisson-like catch distribution.

? list some concerns about the posterior predictive check. One is that the observed data are used twice: in fitting the model and in producing the Bayesian p value. Another is that the method is qualitative, in that there is not an objective p-value range that indicates an acceptable fit. Despite those concerns, we agree with **?** that the approach is a useful way of detecting cases where a model fits poorly. Simulation is helpful in gaining experience with the approach, because it is straightforward to obtain Bayesian p values for correct and incorrect fitted models.

4.3.6 Model selection

It is often the case that more than one model may be considered as plausible. For example, survey catches could be modeled using either a Poisson or negative binomial distribution. A model for a tag-return study could include separate parameters for reporting rate of tags from commercial and recreational fishers, or just a single parameter if the sector reporting rates were similar. A third example is a model for survey catch rate, which might include covariates such as location, year, depth, or water temperature. Determining which covariates affected catch rate would make it possible to produce an improved survey index that was adjusted for environmental variation. For example, were low survey catches in the current year due to anomalously low water temperatures on sampling dates or to a real decline in abundance? The suite of candidate models should be chosen before data analysis begins (**?**) to avoid the tendency to examine the study results and to choose candidate models that match observed patterns. For example, observing low recruitment in a recent year might spur us to search for environmental covariates with a similar temporal pattern.

Choosing a preferred model among the suite of candidates (model selection) is a vast and complex topic (**?**), and there is no consensus among statisticians as to the best approach (**?**). **?** suggest that one practical approach is to decide on a model that is biologically plausible and stick to that. They also sometimes eliminate candidate parameters that have credible intervals that include zero, similar to a backward stepwise regression. **?** recommend model selection based on predictive ability, either in-sample (data used in fitting the model) or out-of-sample (new data). Out-of-sample validation is considered the gold standard (**?**). The main disadvantage of that approach is the requirement for additional data not used in fitting the model.

We illustrate here a simple approach of separately fitting candidate models and comparing DIC scores. Our “observed” data are simulated counts from a negative binomial distribution. Next we fit Poisson and negative binomial candidate models to the same observed data and compare DIC scores. The negative binomial distribution in JAGS uses the probability of success (p) and size (our overdispersion parameter k). We return the estimated mean for the negative binomial distribution, which is calculated internally using p.est and k.est.

```
# DIC comparison for model selection, using "observed" data from a negative binomial distribution

rm(list=ls()) # Clear Environment

N <- 30
mu <- 7
k=1
variance <- mu+(mu^2)/k
Count <- rnbinom(n=N, mu=mu, size=k) # Counts drawn from negative binomial
Freq <- table(Count)  # Distribution of simulated counts
barplot(Freq, main="", xlab="Count", ylab="Frequency")

# Load necessary library
library(rjags)
library(R2jags)

sink("PoissonFit.txt")
cat("
model {

# Prior
 lambda.est ~ dunif(0, 100)

# Likelihood
    for(i in 1:N) {
      Count[i] ~ dpois(lambda.est)
      } #y
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("N", "Count")

# Initial values
jags.inits <- function(){ list(lambda.est=runif(n=1, min=0, max=100))}
```

```
model.file <- 'PoissonFit.txt'

# Parameters monitored
jags.params <- c("lambda.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
#plot(jagsfit)

Poisson.DIC <- jagsfit$BUGSoutput$DIC

sink("NBFit.txt")
cat("
model {

# Prior
 p.est ~ dunif(0, 1)  # JAGS uses p (probability of success) for negative binomial
 k.est ~ dunif(0, 1000)
 mu.est <- k.est*(1-p.est)/p.est

# Likelihood
    for(i in 1:N) {
      Count[i] ~ dnbinom(p.est, k.est)
      } #y
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("N", "Count")

# Initial values
jags.inits <- function(){ list(p.est=runif(n=1, min=1E-6, max=1),
                               k.est=runif(n=1, min=0, max=1000))}

model.file <- 'NBFit.txt'

# Parameters monitored
jags.params <- c("mu.est", "k.est")

# Call JAGS from R
```

```
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
#plot(jagsfit)
NB.DIC <- jagsfit$BUGSoutput$DIC
Poisson.DIC - NB.DIC
```

The difference in DIC scores (printed to the Console for convenience) ranged from 50.4 to 90.4 in five trials using the default simulation settings. The substantially higher DIC score for the (incorrect) Poisson model is clear evidence of a poorer fit. **?** suggested that an alternative model with an AIC score within 1-2 of the best model merits consideration whereas a difference of 3-7 suggests much less support. **?** noted that the **?** criteria appear to work well for DIC scores.

Comparing DIC scores worked well for this case. Also, using simulated counts from a negative binomial distribution is convenient because we can adjust the overdispersion parameter (k) to obtain a more "Poisson-like" distribution. At what value for k do you obtain similar DIC scores for the two models? Observe the change in shape of the count frequency distribution as you vary k.

If considering this approach to model selection, keep in mind that DIC scores are not reliable for some more complex models; for example, ones with categorical parameters (Lunn et al., 2009). This is an advantage of starting with simulated data, in that the underlying distribution is known. Confirming that DIC scores reliably identify the correct model with simulated data is a useful first step before any analysis of field data.

4.4 Exercises

1. Estimate uncertainty around a two-sample mark-recapture estimate Nhat using the simulation and Bayesian approaches for a capture probability of 0.4. How similar are the results and how do they compare to results using the original capture probability of 0.2?
2. **?** provided practical advice on acceptable levels of accuracy and precision. They suggested that estimates within 10% of the true value be acceptable for research studies, versus 25% for management and 50% for preliminary studies. Bayesian credible intervals could be used as a proxy for their accuracy and precision targets. What cap-

ture probabilities (approximate) would correspond to those targets for the mark-recapture example?

3. How many errors can you locate before running the following code? Fix all errors that you spotted then run the code to fix any remaining errors.

```
rm(list=ls()) # Clear Environment

# Load necessary library packages
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# Arbitrary 'observed' values for analysis
Reps <- 20
AveC <- 3
Count <- rpois(n=Reps lambda=AveC)

# JAGS code
sink("PoissonSim.txt")
cat("
    model {

    # Priors
    AveC.est ~ dunif(0, MaxCount)

    # Likelihood
    for (i in 1:Rep){
    Count[i] ~ dpois(AveCest)
    } #i
}
    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("Count")

# Initial values.
jags.inits <- function(){ list(AveC.est=runif(n=1, min=0, max=100))}

model.file <- 'PoisonSim.txt'

# Parameters monitored
jags.params <- c("AveC.est")
```

```
# Call JAGS from R
jagsfit <- jags(data=jags.data, inits=jags.inits, jags.params,
                n.chains = 3, n.thin = 1, n.iter = 2000, n.burnin = 1000,
                model.file)
print(jagsfit, digits=3)
plot(jagsfit)
```

4. Consider a mark-recapture study with $n_1 = 60$ and two independent samples for recaptures (n_2=90, m_2=40; n_3=55, m_3=30). Use a vector for p at 0.01 intervals and produce a vector and plot of joint likelihoods (vector given the name “both”). Assume that only the initial sample (n_1) is marked; samples n_2 and n_3 provide two independent snapshots of the marked fraction. Explain in detail how p[which.max(both)] works and what it produces.
5. Use the approach from Section @**??**Model-selection) to compare model fits for a more “Poisson-like” negative binomial distribution (e.g., k=10).

5

Abundance

Information about abundance can be invaluable for making policy decisions. For fish populations that support commercial or recreational fishing, population estimates are highly useful for deciding about the appropriate level of harvest. For rare or declining populations, abundance data can guide the restoration process. In this chapter, we will consider a few simple field approaches for estimating population size. These methods can be used in freshwater or marine systems, but they do require that the study area be closed (no migration in or out, no recruitment (additions through reproduction), and no mortality for the duration of the study). Later chapters will consider open models that jointly estimate population abundance and mortality over longer time frames.

5.1 Removal

Removal models are a good option when a substantial fraction of the study population can be captured in a single sample. One common example is use of backpack electrofishing in streams that can be blocked on both ends to provide a temporarily closed system. Fish captured in each sample (or "pass") are held out temporarily and the declining catches from the diminishing population provide the data for estimating abundance. Consider a case with a true (study area) population of 100 and a capture probability (fraction of the population captured in each sample) of 0.4. The expected sequence of catches would be 40 (100 * 0.4), 24 ((100-40) * 0.4), 14.4 ((60-24)*0.4), etc. A model fit to these data provides estimates of initial population size and capture probability that best mimic the pattern of catches. The fit would be very good if we used the above expected values, but it is more challenging (and realistic) when the catch sequence includes random variation.

We illustrate the removal method using a simulation to generate sample data. Here the true values are arbitrary, but roughly similar to estimates obtained by **?** for Alaska streams. If planning a field study, your assumed population level should be based on prior work or literature review. The assumed capture probability should also be realistic, but it is a bit different from population size in that it is under the investigator's control. Thus, a better estimate can be achieved by increasing sampling effort; for example, by using more electrofishing units or setting more traps. The other study design variable controlled by the investigator is the number of samples. Here we use five removal samples, which should provide high quality results.

```
rm(list=ls()) # Clear Environment

# Generate simulated removals, assuming constant catchability
N.removals <- 5 # Number of removals
N.true <- 100
CapProb <- 0.4
N.remaining <- N.true # Initialize N.remaining, before removals begin
Catch <- numeric() # Creates empty vector for storing catch data
for (j in 1:N.removals){
  Catch[j] <- rbinom(1, N.remaining, CapProb)
  N.remaining <- N.remaining - Catch[j]
  }
```

Running the above lines of code generates the sequence of simulated catches. (Note that these lines of simulation code would be replaced by an assignment statement (e.g., Catch <- c(14, 4, 6, 2, 3)) if we were analyzing data from an actual field study). A 'for' loop is used to step through the simulated sampling sequence. For each value of j, we generate a binomially-distributed random catch using the rbinom() function, then remove that number of fish from the remaining population. One example run produced a catch vector of 36, 25, 20, 6 and 6. The high assumed value for capture probability produces very good data; thus, these values are close to what would be expected. Try running the code using other values for capture probability. Is there a lower bound below which you would not expect the method to work well?

The next step is to fit the removal model using JAGS. The prior distribution for capture probability is straightforward (uniform 0-1) but the choice is a bit more subjective for population size (N.est). We can use the total catch as a lower bound and an arbitrary high value for the upper bound. That should ensure that the prior distribution does not affect the solution, but that could be confirmed by comparing fits using different upper bounds. Knowledge of your study system and the literature would be useful in setting the upper bound for analyzing a real data set.

```
# Load necessary library
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# JAGS code for fitting model
sink("RemovalModel.txt")
  cat("
model{

# Priors
 CapProb.est ~ dunif(0,1)
 N.est ~ dunif(TotalCatch, 2000)

 N.remaining[1] <- trunc(N.est)
 for(j in 1:N.removals){
      Catch[j]~dbin (CapProb.est, N.remaining[j]) # jth removal
      N.remaining[j+1] <- N.remaining[j]-Catch[j] # Remaining population after removal
  } #j

}
      ",fill=TRUE)
  sink()

     # Bundle data
  jags.data <- list("Catch", "N.removals", "TotalCatch")

  TotalCatch <- sum(Catch[])

  # Initial values
  jags.inits <- function(){ list(CapProb.est=runif(1, min=0, max=1),
                                 N.est=runif(n=1, min=TotalCatch, max=2000))}

  model.file <- 'RemovalModel.txt'

  # Parameters monitored
  jags.params <- c("N.est", "CapProb.est")

   # Call JAGS from R
  jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                  n.chains = 3, n.thin = 1, n.iter = 40000,
                  model.file)
  print(jagsfit)
  plot(jagsfit)
```

JAGS requires an integer size variable in dbin() so we use the trunc() function to truncate the starting value. The looping code is the same as in the simulation, except that JAGS does not allow a value to to be repeatedly changed in a loop, so we set up remaining population size as a vector. We provide to JAGS the simulated catch vector, the number of removal samples, and the total catch. Initial values are generated using the same distributions as the priors. We monitor the estimates of population size and capture probability.

Initial runs using fewer updates occasionally resulted in poor convergence so n.iter was increased to 40000. Removing n.burnin uses the default of discarding half the updates. Running the simulation and analysis code multiple times for these simulation parameters shows that the estimates are consistently good, with narrow credible intervals.

As a fishery biologist, you could use this combination of simulation and analysis to plan your field study. You could look at different choices for the two design variables (number of removal samples, capture probability) to decide what would produce reliable results. Try runs using the above code, modified to test different numbers of removal samples and assumed capture probabilities. It is important to acknowledge that this is an optimistic case, in that the data exactly follow binomial sampling. In a real field study, the model will be an approximation so any design choices should be viewed as lower bounds. For example, if four removal samples appear to be sufficient, consider that a minimum or perhaps low. Another consideration is that we have made the simplifying assumption that capture probability is constant across samples. **?** found that capture probability was generally constant when using traps, but it may decline with disruptive methods such as electrofishing or seining, or if vulnerability varies among individuals (**?**). If capture probability declines as additional samples are taken, a more complex function would be needed for capture probability.

5.2 Mark-recapture

We briefly revisit the two-sample mark-recapture experiment from Sections 2 and 4. In this section, we change a single line of code to produce a random number of recaptured fish rather than the expected value. Now the combined code (simulation plus analysis) can be rerun multiple times to build intuition about the reliability of the experimental results. This is especially useful when the assumed population size or capture probability is low. The true value can be compared to the 95% credible interval (2.5 and 97.5 percentiles, printed to the Console) or 80% credible interval shown in the plots. You can look at either the marked fraction or Nhat, as they covary (low estimate for marked fraction produces high estimate for Nhat). The last four lines of code illustrate that it can be instructive to examine not only the posterior distributions but also the underlying relationship(s) (see bivariate plot). Here it is clear that Nhat is inversely related to the model parameter for marked fraction, but it may be less obvious in situations with more complex models.

```r
rm(list=ls()) # Clear Environment

# Simulate experiment
N.true <- 400  # Population size
p.true <- 0.2 # Capture probability (fraction of population sampled)
n1 <- N.true * p.true # Number caught and marked in first sample
n2 <- N.true * p.true # Number caught and examined for marks in second sample
m2 <- rbinom(1, n2, p.true) # Number of marked fish in second sample

# Load necessary library
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# JAGS code
sink("TwoSampleCR.txt")
cat("
    model {

    # Priors
    MarkedFraction ~ dunif(0, 1)

    # Calculated value
    Nhat <- n1 /MarkedFraction

    # Likelihood
    # Binomial distribution for observed recaptures
    m2 ~ dbin(MarkedFraction, n2)
    }

    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("n1", "n2", "m2")

# Initial values.
jags.inits <- function(){ list(MarkedFraction=runif(1, min=0, max=1))}

model.file <- 'TwoSampleCR.txt'
```

```
# Parameters monitored
jags.params <- c("Nhat", "MarkedFraction")

# Call JAGS from R
jagsfit <- jags(data=jags.data, inits=jags.inits, jags.params,
                n.chains = 3, n.thin = 1, n.iter = 2000, n.burnin = 1000,
                model.file)
print(jagsfit, digits=3)
plot(jagsfit)

par(mfrow=c(3,1))
hist(jagsfit$BUGSoutput$sims.array[,,1], main="", xlab="Marked fraction")
hist(jagsfit$BUGSoutput$sims.array[,,2], main="", xlab="Population estimate")
plot(jagsfit$BUGSoutput$sims.array[,,1], jagsfit$BUGSoutput$sims.array[,,2],
     xlab="Marked fraction", ylab="Population estimate")
```

If you were managing the fishery for this lake population, how might you use the estimate of abundance? Try a few runs (simulation and model fitting) to see whether this level of uncertainty would be tolerable for management. In doing so, keep in mind that this assumed capture probability (0.2) is quite high and would be difficult to achieve in most field situations. Is there a rough lower bound for capture probability, below which the cost and effort of doing a study would not be justified?

5.3 Binomial-mixture

A population estimate can sometimes be obtained from count data when replicate survey samples are taken (?). One fisheries example is the use of side-scan sonar to count adult Atlantic sturgeon, which are large and distinctive enough to be identifiable on side-scan images (??). Sites were assumed to be closed (no movement in or out, no mortality) for the brief duration of the survey. Each site had an unknown number of sturgeon, but it was not assumed that every sturgeon was counted (detected). Counts could vary among survey passes due to a variety of factors, including a fish's orientation, overlap with other fish or structure, or distortion in the image. Thus the counting process was modeled using a binomial distribution, where "size" is the number of sturgeon present at the site and "prob" is the probability of being detected on that survey pass. The probability is comparable to a capture probability but here sampling was unobtrusive. This type of model is referred to as a binomial mixture model (or N-mixture model). It attempts to separate the biological parameter (site abundance) from the observational parameter (detection probability) (**?**).

Consider a site with twenty individuals and a detection probability of 0.5. We can simulate a large sample of replicates to get a smooth distribution of counts, although in an actual field survey, there would typically be only a few replicate passes per site.

```r
rm(list=ls()) # Clear Environment

# Simulate experiment
N.true <- 20  # Number of individuals at site
p.true <- 0.5 # Detection probability (fraction of individuals detected)
Replicates <- 1000
Counts <- rbinom(n=Replicates, size=N.true, prob=p.true) # Vector of replicate counts

par(mfrow=c(1,1)) # Ensure plot window is reset to default setting
hist(Counts, main="")
table(Counts)
```

The expected count is 10 (N.true * p.true); the histogram and table show the frequency of different values. One run produced minimum and maximum counts of 3 and 16. Counting 20 would be possible but extremely unlikely. How might you estimate empirically the probability of counting 15 or more? Experiment with different detection probabilities to find a level that produced occasional counts of 20 individuals.

It is instructive to compare the default distribution (N.true=20 and p.true=0.5) to other combinations that would have the same expected count (e.g., N.true=40 and p.true=0.25). In the following code, we use even more replicates, and add upper and lower limits for the x axis in order to compare the two histograms. The differences in the shape of the distribution are subtle, illustrating that it can be challenging to get a reliable abundance estimate using this approach.

```r
N.true <- 20  # Number of individuals at site
p.true <- 0.5 # Detection probability (fraction of individuals detected)
Replicates <- 10000
Counts <- rbinom(n=Replicates, size=N.true, prob=p.true) # Vector of replicate counts
table(Counts)

par(mfrow=c(2,1)) # Two rows, one column
hist(Counts, main="N.true=20", xlim=c(0,25))
```

```
N.true <- 40  # Number of individuals at site
p.true <- 0.25 # Detection probability (fraction of individuals detected)
Counts <- rbinom(n=Replicates, size=N.true, prob=p.true) # Vector of replicate counts
hist(Counts, main="N.true=40", xlim=c(0,25))
table(Counts)
```

Try a more extreme second case (N.true=100 and p.true=0.1) to see if there is a more pronounced difference compared to the default (remember to change the second plot title!). It may be necessary to change the x-axis limits if N.true is increased.

To simulate a full field study, we need to add code for multiple sites. We assume 30 sites and 3 replicate visits to each site. **?** recommended at least 10-20 sites and at least two observations per site. It is assumed that the number of sturgeon per site varies about an underlying mean, so having a large sample of sites will aid in estimating that mean. There are a variety of ways to model the variation among sites. Here we chose the simplest option of using a Poisson distribution, which has only one parameter to estimate:

```
rm(list=ls()) # Clear Environment

Sites <- 30
Replicates <- 3
lam.true <- 20 # Mean number per site
p.true <- 0.5 # Detection probability

Counts <- matrix(NA, nrow=Sites, ncol=Replicates)
Site.N <- rpois(n = Sites, lambda = lam.true) # True abundance at each site
for (i in 1:Sites){
  Counts[i,] <- rbinom(n = Replicates, size = Site.N[i], prob = p.true)
  } #i
head(Site.N)
head(Counts)
```

The matrix function creates an empty matrix, which will hold the replicate counts for each site. Next, the function rpois generates the Poisson-distributed random number of sturgeon at each site. We then loop over sites to generate the binomially-distribution counts at each site. This illustrates the hierarchical nature of this model in that site abundance is set first, then replicate observations are drawn, given the site abundance. We can use the head function to look at the relationship between true abundance (Site.N) and counts for the first few sites. In one example run, the first site had 25 sturgeon and produced replicate counts of 11, 14, and 13.

```
# Load necessary library
library(rjags)
library(R2jags)

# Specify model in BUGS language
sink("N-mix.txt")
cat("
model {

# Priors
  lam.est ~ dunif(0, 100)  # uninformative prior for mean abundance at each site
  p.est ~ dunif(0, 1)

# Likelihood
# Biological model for true abundance
for (i in 1:Sites) {
   N.est[i] ~ dpois(lam.est)
   # Observation model for replicated counts
   for (j in 1:Replicates) {
      Counts[i,j] ~ dbin(p.est, N.est[i])
      } # j
   } # i
totalN <- sum(N.est[])     # Calculated variable: total pop. size across all sites
}
",fill = TRUE)
sink()

# Bundle data
jags.data <- list("Counts", "Sites", "Replicates")

# Initial values
Nst <- apply(Counts, 1, max) + 1    # Kery and Schaub
jags.inits <- function(){ list(lam.est=runif(n=1, min=0, max=100),
                               N.est=Nst, p.est=runif(1, min=0, max=1))}

model.file <- 'N-mix.txt'

# Parameters monitored
jags.params <- c("lam.est", "p.est", "totalN")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 100000,
                model.file)
```

```r
print(jagsfit)
par(mfrow=c(1,1)) # Reset plot panel
plot(jagsfit)

# Estimated posterior distribution for total population size
hist(jagsfit$BUGSoutput$sims.list$totalN, xlab="Estimated total pop size", main="")
abline(v=sum(Site.N), col="red")
```

The JAGS code includes uninformative prior distributions for mean site abundance and the detection probability. The choice for mean abundance is more subjective but can be based on the observed counts and knowledge or experience about the possible range for true abundance or detection probability. The JAGS code is again structurally similar to the simulation code. The first step is to generate estimates of the true (unknown) abundance at each site. The observations are binomially distributed samples, given the estimated abundance at each site. It is sometimes of interest to look at total abundance over sites, so the site estimates are summed (totalN).

It can be helpful to provide initial values for site abundance. JAGS will terminate the run if a nonsensical event occurs (e.g., count greater than the current estimate of abundance at a site). **?** provide example code using the apply() function, which is a generic function for applying (hence the name) other functions. In this case, the max() function is applied to the Count matrix by row (argument=1). The returned maximum value for each row is incremented by 1, implying that there is likely to be at least one more sturgeon present than the maximum observed count. The remaining parameters are initialized using settings that match the prior distributions. The final lines of R code show the estimated posterior distribution for total abundance (summed over sites). The abline function is used to add to the plot the true total abundance (vertical red line).

Estimates are generally reliable for these assumed parameter values and study design settings, although convergence was sometimes slow so a large number of updates was used. Note that the model is hierarchical in structure, in that the overall (Poisson) mean is estimated using all site data, then site-specific estimates are drawn using that overall mean. It is worthwhile to run the combined code (simulation plus analysis) several times to see variation in the estimates and their credible intervals. Median estimates of mean site abundance and detection probability tend to be close to the assumed values although credible intervals are fairly wide. The posterior distribution for total abundance appears to be reliable although it is skewed and has a high upper bound for the credible interval (451-1294 for one example run).

This model is a good illustration of the potentially problematic nature of pD, the estimated number of effective parameters. One would expect an estimate around 32, to account for detection probability, mean site abundance, and the site-specific abundance estimates using that mean. However, estimates from several runs were around 60. **?** note that hierachical models (such as this one) can be problematic for estimating the number of effective parameters (and therefore for DIC). We can look at pD for the various models in this book, to get some experience about when to trust pD and DIC estimates.

As usual, keep in mind that this is a very optimistic scenario. We are estimating mean abundance and detection probability using 30 sites that function as spatial replicates. The data are generated using a detection probability that is high and constant over sites. In an actual field study, there would be additional variation in detection probability among sites, and a binomial distribution would only be an approximation to the field sampling process that generates the observed counts. We have also assumed that variation among sites follows a Poisson distribution. Other distributions that allow for more variation among sites, such as a negative binomial, are possible and can be much more challenging to fit.

5.4 Exercises

1. Create a table of estimated credible intervals for population size, using different numbers of removal samples and a range of values for capture probability. You will need several replicate simulations to get a clear picture of results for each setting. What are the lower limits below which estimates are judged not to be reliable?
2. For a two-sample mark-recapture experiment with a true population size of 1000, produce a table of credible intervals for Nhat for different assumed capture probabilities. What sampling intensity would produce an estimate with uncertainty close to the Robson and Regier target for management (+/- 25%)?
3. Modify the binomial-mixture example to compare credible intervals for totalN using fewer sites but more replicates, preserving the same total number of observations (e.g., 10 sites and 9 replicate survey visits). Can you suggest potential advantages and disadvantages of using fewer sites?

6

Survival

Information about survival rate is valuable for managing exploited stocks and working to rebuild rare or declining populations. Survival is affected by two broad categories of threats: fishing and natural mortality. Fishing obviously refers to harvest, from commercial or recreational sectors, but could include mortality associated with fish that are caught and released. Catch-and-release fishing primarily occurs in the recreational sector, but can occur in the commercial sector due to regulations or lack of markets (sometimes referred to as discard). Natural mortality is a generic term for mortality sources other than fishing. There is typically a strong negative correlation between natural mortality rate and body size, indicating the predominant role of predation (**?**). Other sources of natural mortality include starvation, disease and senescence. It is important as a first step to estimate overall survival reliably, but management can be improved by partitioning total mortality into fishing and natural component rates. We begin with approaches for estimating overall survival, then in Section 7 consider tagging and telemetry methods that provide insights about sources of mortality.

6.1 Age-composition

The simplest approach for estimating survival is to examine the age composition of a sample of fish. Age would typically be determined through laboratory work using otoliths or other hard parts that show annual patterns (**?**). This approach is usually applied to adult fish that would be expected to have similar survival rates. We begin with simulation code for generating the age distribution:

```
rm(list=ls()) # Clear Environment

# Simulate age composition sampling
MaxAge <- 10
```

```r
S.true <- 0.7 # Annual survival probability

# Simulated age vector at equilibrium (expected values)
N <- array(data=NA, dim=MaxAge)
N[1] <- 1000
for (j in 2:(MaxAge)){
  N[j] <- N[j-1] * S.true
} #j
age.vec <- seq(1,MaxAge, by=1)  # Sequence function to provide x axis values for plotting
barplot(N, names.arg=age.vec, xlab="Age", ylab="Number")

SampleSize <- 30
AgeMatrix <- rmultinom(n=1, size=SampleSize, prob=N) # prob vector is rescaled internally
AgeSample <- as.vector(t(AgeMatrix)) # Transpose matrix then convert to vector
barplot(AgeSample, names.arg=age.vec, xlab="Age", ylab="Frequency")
```

The simulation tracks fish up to age 10 (maximum age tracked, not lifespan) with an arbitrary annual survival rate of 0.7. The looping code for age composition starts with 1000 fish at age 1 and reduces that number at each successive age by the annual survival rate. This creates an equilibrium age vector, based on assumptions of constant recruitment (1000 each year) and survival. For example, the 700 age-2 fish would have been the survivors from the previous year's 1000 age-1 recruits. The initial number is arbitrary; the shape of the distribution and age proportions would be the same if we began with 1 or 1000 fish. The next section of code uses the equilibrium age vector as proportions and draws a sample of size 30 using a multinomial distribution (Section 3.1.3). To build some intuition about age sampling, try running the simulation code multiple times with different values for sample size, maximum age, and survival rate. As usual, keep in mind that this is a very optimistic scenario, with constant recruitment and survival and unbiased survey sampling. The only variation reflected in these samples is due to sampling from the multinomial distribution (as is evident if a large age sample is drawn).

The JAGS code for analysis is straightforward and mirrors the simulation code. Note that the equilibrium age vector used as proportions in the dmulti() function must be rescaled manually to sum to 1. Convergence is fast and the estimate of survival rate is moderately precise, despite the small (but perfectly unbiased) sample. For example, the 95% credible interval for one run was 0.57-0.80.

```r
# Load necessary library
library(rjags)
library(R2jags)
```

```
# JAGS code
sink("AgeComp.txt")
cat("
model{
    # Priors
    S.est ~ dunif(0,1)  # Constant survival over sampled age range

    N[1] <- 1000
    # Generate equilibrium age proportions
    for (j in 2:MaxAge){
      N[j] <- N[j-1]*S.est
    } # j
    N.sum <- sum(N[]) # Rescale so that proportions sum to 1
    for (j in 1:MaxAge){
    p[j] <- N[j]/N.sum
    } #j
    # Likelihood
    AgeSample[] ~ dmulti(p[], SampleSize)  # Fit multinomial distribution based on constant surviv
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("AgeSample", "SampleSize", "MaxAge")

# Initial values
jags.inits <- function(){ list(S.est=runif(n=1, min=0, max=1))}

model.file <- 'AgeComp.txt'

# Parameters monitored
jags.params <- c("S.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 4000, n.burnin=2000,
                model.file)
print(jagsfit)
plot(jagsfit)

# Look at posterior distribution versus true value
hist(jagsfit$BUGSoutput$sims.list$S.est, xlab="Estimated survival rate", main="")
abline(v=S.true, col="red")
```

Note that we are not estimating the age proportions as parameters but rather the single parameter for survival rate (which is used to predict the equilibrium age proportions). Thus the number of effective parameters (pD) should be close to one. A few lines of code can be added to compare the predicted and observed age distributions:

```
# Add code to compare observed and predicted age comp
N.hat <- array(data=NA, dim=MaxAge)
N.hat[1] <- 1000
for (j in 2:(MaxAge)){
  N.hat[j] <- N.hat[j-1] * jagsfit$BUGSoutput$median$S.est
} #j
par(mfrow=c(2,1))
barplot(AgeSample, names.arg=age.vec, xlab="Age", ylab="Number", main="Observed")
barplot(N.hat, names.arg=age.vec, xlab="Age", ylab="Number", main="Predicted")
par(mfrow=c(1,1)) # Reset plot window
```

The predicted age distribution looks like a smoothed version of the observed, which makes sense given that we are estimating average survival over the sampled range of ages. When a poor estimate of the survival rate occurs, you will likely notice an odd pattern in the simulated age composition sample. Try running the code several times, starting from where the random age sample is drawn. Increasing the sample size will bring the observed pattern closer to the equilibrium age composition.

Thus far, we have used small sample sizes for these simulated studies. They are sufficient here because the simulation only includes variation due to sampling from the multinomial distribution. It is also convenient to work with a small sample size when studying the code and simulation data. Sample sizes are sometimes small in field studies, and it is important to establish that a study design will provide reliable results when using an achievable sample size. But it can also be informative to test the analytical approach with a very large sample size. This provides a best-case scenario for study results, and ensures that estimates approach the true values (are essentially unbiased). For example, three replicate median estimates of the survival rate were 0.71, 0.69, and 0.70 for a sample size of 2000 fish.

6.1.1 Accounting For Gear Selectivity

The traditional approach for estimating survival from age composition data was known as catch curve analysis. It could be done using linear regression (thus doable by hand in the pre-computer era) by log-transforming the age sample (**?**). Gear selectivity was addressed subjectively, by viewing a plot of the log-transformed catches. **?** described the catch curve pattern as having ascending and descending limbs, with the ascending limb assumed to reflect selectivity and survival. The descending limb was assumed to reflect only survival (age range of "fully selected" fish). The recommended approach was to begin the catch curve analysis at the age of peak catch or the next oldest age, in order to limit the analysis to fully selected fish. For catch curve analysis or the approach given here, declining selectivity for older fish would cause bias in the estimate of survival.

Given sufficient data, it is possible to use age composition data to estimate jointly the survival rate and gear selectivity. A practical example might be a trawl survey where small (younger) fish are not retained by the net. Fish larger than the net mesh size might be assumed to be fully selected, so a logistic function could be used to model gear selectivity that increases up an asymptote: $S_a = \frac{1}{1+e^{k*(a-a_0)}}$

```
# Add in selectivity
k <- 2 # Slope for logistic function
a_50 <- 4 # Age at 0.5 selectivity

# Age selectivity
Sel <- array(data=NA, dim=MaxAge)
for (j in 1:(MaxAge)){
  Sel[j] <- 1/(1+exp(-k*(j-a_50)))
} #j
age.vec <- seq(1,MaxAge, by=1)  # Sequence function to provide x axis values for plotting
par(mfrow=c(1,1))
plot(age.vec, Sel, xlab="Age", ylab="Gear selectivity")
```

Values for the slope (k) and age at 50% selectivity ($a_{0.50}$) are arbitrary, but chosen to increase to a well-defined asymptote before age 10. Try different values for the slope and $a_{0.50}$ to understand how the two parameters affect the shape of the curve. Note that any analysis of gear selectivity will only be successful if all ages share the survival rate and there is a sufficient age range to fully characterize the asymptote. For example, imagine the outcome of an analysis using data only for ages 1-5.

Now let's extend the age composition simulation to include less than fully selected fish:

```
rm(list=ls()) # Clear Environment

# Simulate age composition sampling with age-specific selectivity
```

```
MaxAge <- 10
S.true <- 0.7 # Annual survival probability

# Parameters for selectivity curve
k <- 2 # Slope for logistic function
a_50 <- 4 # Age at 0.5 selectivity

# Age selectivity
Sel <- array(data=NA, dim=MaxAge)
for (j in 1:(MaxAge)){
  Sel[j] <- 1/(1+exp(-k*(j-a_50)))
} #j
age.vec <- seq(1,MaxAge, by=1)  # Sequence function to provide x axis values for plotting
par(mfrow=c(1,1))
plot(age.vec, Sel, xlab="Age", ylab="Gear selectivity")

# Simulated age vector, including gear selectivity
N <- array(data=NA, dim=MaxAge)
N[1] <- 1000
for (j in 2:(MaxAge)){
  N[j] <- N[j-1] * S.true
} #j
N_Sel <- N * Sel
age.vec <- seq(1,MaxAge, by=1)  # Sequence function to provide x axis values for plotting
barplot(N_Sel, names.arg=age.vec, xlab="Age", ylab="Number", main="Expected age distribution")

SampleSize <- 100
AgeMatrix <- rmultinom(n=1, size=SampleSize, prob=N_Sel) # prob vector is rescaled internally
AgeSample <- as.vector(t(AgeMatrix)) # Transpose matrix then convert to vector
barplot(AgeSample, names.arg=age.vec, xlab="Age", ylab="Frequency", main="Sample age distribution"
```

The sample age distribution now reflects both gear selectivity and survival. We use a relatively large sample size and test whether it is possible to estimate all three parameters (survival rate and two parameters for selectivity):

```
# JAGS analysis
library(rjags)
library(R2jags)

sink("AgeComp.txt")
cat("
model{
    # Priors
    S.est ~ dunif(0,1)  # Constant survival over sampled age range
    a50.est ~ dunif(0, MaxAge)
    k.est ~ dlnorm(0, 1E-6) # Slope > 0 for increasing selectivity

    N[1] <- 1000
    # Generate equilibrium age proportions
    for (j in 2:MaxAge){
      N[j] <- N[j-1]*S.est
    } # j
    for (j in 1:MaxAge){
    Sel[j] <- 1/(1+exp(-k.est*(j-a50.est)))
     N.Sel[j] <- N[j] * Sel[j]
    }
    N.sum <- sum(N.Sel[]) # Rescale so that proportions sum to 1
    for (j in 1:MaxAge){
    p[j] <- N.Sel[j]/N.sum
    } #j
    # Likelihood
    AgeSample[] ~ dmulti(p[], SampleSize)  # Fit multinomial distribution based on constant surviv
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("AgeSample", "SampleSize", "MaxAge")

# Initial values
jags.inits <- function(){ list(S.est=runif(n=1, min=0, max=1),
                               a50.est=runif(n=1, min=0, max=MaxAge),
                               k.est=rlnorm(n=1, meanlog=0, sdlog=1))} # Arbitrary low initial val

model.file <- 'AgeComp.txt'

# Parameters monitored
jags.params <- c("S.est", "a50.est", "k.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 20000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

The age at 50% selectivity should be less than the maximum age so a uniform prior distribution is appropriate. The slope of the selectivity curve does not have an obvious upper bound, so following McCarthy (2007), we use an uninformative lognormal prior distribution. Restricting the slope to positive values is consistent with a survey gear that underrepresents smaller fish. Convergence can be slow when using a lognormal prior distribution so check Rhat values and increase the number of updates (n.iter) if needed.

There are several reasons why this analysis works (at least for these assumed parameter values). We have an unbiased sample of size 100 from the expected age distribution (which reflects both survival and selectivity). The most important reason is that survival is constant over the entire age range. Selectivity and survival are confounded for young fish but not for older fish that are fully vulnerable to the survey. Thus the shape of the selectivity function matters, as does the age range included in the analysis. It would not be possible to jointly estimate selectivity and survival if selectivity increased across the surveyed age range. And as before, the analysis is a best-case scenario in that the only variation is due to sampling from the multinomial distribution. Selectivity varies by age but all ages share the same selectivity parameters.

One way to improve on this analysis would be to include auxiliary data on selectivity. We could accomplish this by conducting a short-term tagging study with (for simplicity) equal numbers tagged by age. Having a tagged subpopulation with a known age structure provides direct information on selectivity, unlike the age composition data where selectivity and mortality are partially confounded. The logistical challenge in adding this auxiliary study is in obtaining sufficient tag returns through survey sampling.

The following version of the code includes auxiliary tag-return data:

```r
rm(list=ls()) # Clear Environment

# Simulate age composition sampling with age-specific selectivity
MaxAge <- 10
S.true <- 0.7 # Annual survival probability

# Parameters for selectivity curve
k <- 2 # Slope for logistic function
a_50 <- 4 # Age at 0.5 selectivity

# Age selectivity
Sel <- array(data=NA, dim=MaxAge)
for (j in 1:(MaxAge)){
  Sel[j] <- 1/(1+exp(-k*(j-a_50)))
} #j
```

```
age.vec <- seq(1,MaxAge, by=1)  # Sequence function to provide x axis values for plotting
par(mfrow=c(1,1))
plot(age.vec, Sel, xlab="Age", ylab="Gear selectivity")

# Simulated age vector, including gear selectivity
N <- array(data=NA, dim=MaxAge)
N[1] <- 1000
for (j in 2:(MaxAge)){
  N[j] <- N[j-1] * S.true
} #j
N_Sel <- N * Sel
age.vec <- seq(1,MaxAge, by=1)  # Sequence function to provide x axis values for plotting
barplot(N_Sel, names.arg=age.vec, xlab="Age", ylab="Number", main="Expected age distribution")

SampleSize <- 100
AgeMatrix <- rmultinom(n=1, size=SampleSize, prob=N_Sel) # prob vector is rescaled internally
AgeSample <- as.vector(t(AgeMatrix)) # Transpose matrix then convert to vector
barplot(AgeSample, names.arg=age.vec, xlab="Age", ylab="Frequency", main="Sample age distribution"

# Auxiliary tag-return data
TagReturns <- 30
RetMatrix <- rmultinom(n=1, size=TagReturns, prob=Sel) # prob vector is rescaled internally
RetVector <- as.vector(t(RetMatrix)) # Transpose matrix then convert to vector
barplot(RetVector, names.arg=age.vec, xlab="Age", ylab="Frequency", main="Tag returns")

# JAGS analysis
library(rjags)
library(R2jags)

sink("AgeComp.txt")
cat("
model{
    # Priors
    S.est ~ dunif(0,1)  # Constant survival over sampled age range
    a50.est ~ dunif(0, MaxAge)
    k.est ~ dlnorm(0, 1E-6) # Slope > 0 for increasing selectivity

    N[1] <- 1000
    # Generate equilibrium age proportions
    for (j in 2:MaxAge){
      N[j] <- N[j-1]*S.est
    } # j
    for (j in 1:MaxAge){
    Sel[j] <- 1/(1+exp(-k.est*(j-a50.est)))
```

```
     N.Sel[j] <- N[j] * Sel[j]
    }
    N.sum <- sum(N.Sel[]) # Rescale so that proportions sum to 1
    for (j in 1:MaxAge){
    p[j] <- N.Sel[j]/N.sum
    } #j
    # Likelihood
    AgeSample[] ~ dmulti(p[], SampleSize)  # Fit multinomial distribution based on constant surviv

    # Additional likelihood for tag-return data
    Sel.sum <- sum(Sel[])
    for (j in 1:MaxAge){
    Sel.p[j] <- Sel[j]/Sel.sum
    } #j
    RetVector[] ~ dmulti(Sel.p[], TagReturns)
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("AgeSample", "SampleSize", "MaxAge", "TagReturns", "RetVector")

# Initial values
jags.inits <- function(){ list(S.est=runif(n=1, min=0, max=1),
                               a50.est=runif(n=1, min=0, max=MaxAge),
                               k.est=rlnorm(n=1, meanlog=0, sdlog=1))} # Arbitrary low initial val

model.file <- 'AgeComp.txt'

# Parameters monitored
jags.params <- c("S.est", "a50.est", "k.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 20000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

There are three changes to the code in this version. First, the vector of random tag returns is generated just below the random survey catch vector. You can examine the tag-return vector in the Environment window to get a sense of what the auxiliary study contributes to the overall analysis. The next code change is adding a second likelihood component to make use of the tag-return data. The third change is to pass these additional data (tag return total and by age) to JAGS.

Having direct information on gear selectivity improves estimates of survival and selectivity, especially if the age composition sample size is small. And more importantly, this modification illustrates how auxiliary data can be added in JAGS to refine an analysis. Writing code to fit the specifics of a field study is valuable for planning and analysis, and is a major advantage over using canned software. The remaining chapters will include other examples where multiple data sources can be combined to improve estimates.

6.1.2 Cohort model

One limitation of the above analysis (Section 6.1) or a traditional catch curve is that recruitment is assumed to be constant (or can be averaged over). Catch curve analyses are generally robust to this assumption (**?**), but year-class variation can sometimes be quite extreme. The following approach addresses that concern by modeling each year-class (or cohort) separately. The approach is similar to a virtual population analysis (VPA) or cohort analysis (**?**) in that a catch matrix provides age- and year-specific information that is used to track cohorts across years. A VPA is traditionally done using harvest data, but here, we assume the catch-at-age matrix contains survey data. Survey information allows for modeling relative abundance across time, whereas a traditional VPA is used to estimate absolute abundance.

The following simulation code provides the survey catch-at-age matrix. The number of ages and years is arbitrary, but we are estimating survival rate for each year so there should be enough ages to provide a decent sample size. The annual survival rate is drawn from a uniform distribution between arbitrary lower and upper bounds. Having a fair degree of annual variation in survival makes for a better test of our ability to estimate those values.

```
rm(list=ls()) # Clear Environment

# Simulation to create survey catch-at-age matrix
# Assume ages included in matrix are fully selected
A <- 4 # Arbitrary number of ages in survey catch matrix
Y <- 10
LowS <- 0.2 # Arbitrary lower bound
HighS <- 0.9 # Arbitrary upper bound
MeanS <- (LowS+HighS)/2
AnnualS <- array(data=NA, dim=(Y-1))
AnnualS <- runif((Y-1), min=LowS, max=HighS)
```

```
MeanR <- 400 # Arbitrary mean initial cohort size (relative)
N <- array(data=NA, dim=c(A, Y))
N[1,] <- rpois(Y, MeanR) # Starting size of each cohort at first age
for (a in 2:A){
  N[a,1] <- rpois(1,MeanS^(a-1)*MeanR) # Starting sizes for ages 2+, year 1
  for (y in 2:Y){
    N[a,y] <- rbinom(1, N[a-1, y-1], AnnualS[y-1]) # Initial number yr 1, older ages
    } #y
  } #a

SurveyCapProb <- 0.05 # Capture probability
SurveyC <- array(data=NA, dim=c(A, Y))
# Generate survey catches (account for survey catchability)
for (a in 1:A){
  for (y in 1:Y){
    SurveyC[a,y] <- rbinom(1, N[a, y], SurveyCapProb)
  } #y
} #a
```

The model tracks coded ages 1:A, which could represent a range of older ages with comparable survival rates. True abundance at age 1 (N[1,]) is drawn from a Poisson distribution, based on an arbitrary mean initial cohort size. Initial numbers for ages 2:A in the first year are also obtained from a Poisson distribution, where the expected number at age a is obtained by a product of survival rates multiplied by mean recruitment. For example, the expected number is MeanR * S at age 2 (adjusting for survival over one year), MeanR * S^2 for age 3, etc. The code then loops over years (columns) and ages (rows) to fill in the rest of the matrix, using a binomial distribution to simulate the survival process (and using the year-specific survival rate). The survey catch-at-age matrix includes variation due to sampling (binomally-distributed using an assumed capture probability of 0.05), as well as year-class strength and survival. Click on N and SurveyC in the Environment window to compare the two matrices in Source windows.

The JAGS code for estimating cohort size and annual survival is generally similar to the simulation code. Prior distributions and initial values for predicted survey catches use lognormal distributions. The log-scale mean for the initial values was set at a high value to reduce the risk of a runtime JAGS error due to an inconsistent parent node (e.g., estimated survey "abundance" at time y-1 less than observed survey catch at time y). The likelihood for the survey catch matrix uses a Poisson distribution.

```
# Load necessary library packages
library(rjags)   # Package for fitting JAGS models from within R
library(R2jags)  # Package for fitting JAGS models. Requires rjags

# JAGS code
sink("CatchAge.txt")
cat("
model {
    # Priors
    for(y in 1:Y) {
      Recruits[y] ~ dlnorm(0, 1E-6) # Non-negative values
    SurveyC.est[1,y]<-Recruits[y] }

    for(a in 2:A){
      Initials[a-1] ~ dlnorm(0, 1E-6) # Non-negative; offset index to run vector from 1 to A-1
    SurveyC.est[a,1]<-Initials[a-1] }

  meanS <- prod(S.est)^(1/(Y-1))
  for (y in 1:(Y-1)) {
    S.est[y] ~ dunif(0,1)
    } #y

    # Population projection
    for(a in 2:A) {
   for(y in 2:Y) {
     SurveyC.est[a,y] ~ dbin(S.est[y-1], trunc(SurveyC.est[a-1,y-1])) } }

    #Likelihood
    for (a in 1:A) {
   for (y in 1:Y) {
      SurveyC[a,y]~dpois(SurveyC.est[a,y])
     } }
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("Y", "A", "SurveyC")

# Initial values
jags.inits <- function(){ list(Recruits=rlnorm(n=Y, meanlog=10, sdlog=1),
                               Initials=rlnorm(n=(A-1), meanlog=10, sdlog=1))}

model.file <- 'CatchAge.txt'
```

```
# Parameters monitored
jags.params <- c("S.est", "meanS")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
plot(jagsfit)

plot(seq(1:(Y-1)), jagsfit$BUGSoutput$mean$S.est, ylab="Survival", xlab="Year",
     type="b", pch=19, ylim=c(0,1))
points(AnnualS, col="blue", type="b")
```

The final three lines of code compare the annual survival estimates and true values. The plot option "type=b" produces a line of connected dots (i.e., *both* points and line segments), and the "pch=19" option selects for a filled dot. An online search for "r plot options pch" will provide the various choices for plotting symbols. The range for the y axis is fixed at 0-1 to ensure that the plot will contain the full range of both the estimates and the true values. The points() function adds to the plot the true values. The model provides reliable estimates of annual (and mean) survival and initial cohort size for the simulated data. Run the simulation and model fitting code multiple times to observe model performance. It would be worthwhile to test the approach using more variability in initial cohort sizes; for example, by using a negative binomial distribution rather than a Poisson.

It is important to keep in mind our predicted survey catches are not estimates of true abundance but simply scaled to the survey catches. **?** illustrate a Bayesian approach for estimating true abundance from survey *and* harvest catch-at-age matrices.

6.2 Telemetry-based

The focus for this section will be on estimating survival using fixed receiver stations and telemetry tags such as transmitters or PIT tags. Letting fish "detect themselves" eliminates the need for costly field surveys that often result in low detection probabilities. For example, **?** showed that survival estimates using fixed stations and automated detections were much better than those obtained using only survey recaptures. Similarly, **?** estimated survival of sonic-tagged Atlantic sturgeon which were detected as they migrated past receivers located in estuaries and rivers. The key in studies of this type is either to have many monitoring sites or to position sites in areas where tagged fish are likely to pass.

These studies produces detections that can be organized as a capture history matrix. Each row of the matrix is a tagged fish, and each column is a time period. The period can be chosen based on whatever time scale is relevant and practical; for example, monthly or quarterly detections would provide insights about seasonal survival. Each matrix cell indicates whether a fish was detected (1) or not (0). A fish not detected could be dead or simply did not move within range of a receiver during that period. Interpreting the capture history for an individual depends on the pattern. There is no ambiguity about an individual with a capture history of 11011. It was alive to the end of the study but was not detected on the third occasion. A sequence such as 11000 is not as straightforward because its fate is uncertain for the final three occasions. A model is essential for making inferences in such cases.

The capture history data can be analyzed using a Cormack-Jolly-Seber (CJS) model (**?**). The model is based only on captures or detections of tagged fish, which are assumed to be representative of the population as a whole. Technically, the model provides an estimate of apparent survival because we are not able to distinguish between mortality and permanent emigration. If the study area is closed to migration, then the estimate can be interpreted as a true survival rate.

Our simulation is for the most basic case, with survival and detection probabilities that are constant over time. Extended models that allow for time variation are presented by **?**. The number of occasions and sample size are arbitrary but they serve as useful replicates here because the underlying parameters are constant over time and individuals. We assume a moderate survival rate (0.5) and a high detection probability (0.8). Detection probabilities of that magnitude are difficult to achieve by survey sampling but are reasonable when using an automated monitoring system (**?**).

```r
rm(list=ls()) # Clear Environment

# Modified from Kery and Schaub 2012, Section 7.3
# Define parameter values
n.occasions <- 5           # Number of capture occasions
marked <- 50               # Number marked (all at time 1)
S.true <- 0.5
p.true <- 0.8
```

```r
# Define function to simulate a capture-history (CH) matrix
simul.cjs <- function(S.true, p.true, marked, n.occasions){
  Latent <- CH <- matrix(0, ncol = n.occasions, nrow = marked)
  # Fill the CH matrix
  for (i in 1:marked){
    Latent[i,1] <- CH[i,1] <- 1    # Known alive state at release occasion
    for (t in 2:n.occasions){
      # Bernoulli trial: does individual survive occasion?
      Latent[i,t] <- rbinom(1, 1, (Latent[i,(t-1)]*S.true))
      # Bernoulli trial: is individual recaptured?
      CH[i,t] <- rbinom(1, 1, (Latent[i,t]*p.true))
    } #t
  } #i
  return(CH)
}

# Execute function
CH <- simul.cjs(S.true, p.true, marked, n.occasions)
```

Our code, modified from **?**, introduces the creation of a user-defined function, which is a way of setting off a discrete block of code that has a specific purpose. We pass to the function the necessary arguments (S.true, p.true, marked, n.occasions) and the function returns the desired result (capture history matrix CH). The latent (true) state and the capture history observation are modeled using Bernoulli trials. The latent state at time t is based on the survival rate between times t-1 and t, but multiplied by the true state at time t-1. That results in live fish having a survival probability of 1 * S.true, whereas the probability for dead fish is 0 * S.true (ensuring that dead fish remain dead). A similar trick is used to generate the observations, given the known true states. Detection probability is 1 * p.true for live fish and 0 * p.true for dead fish. The matrix for latent state is internal to the function, but (for instructional purposes) can be compared to the observed matrix by selecting and running all lines between the second and next-to-last lines of the function. You should see the connection between the true latent state and observations (e.g., a true sequence of 11100 versus an observed sequence of 10100).

```r
# Load necessary library
library(rjags)
library(R2jags)

# Specify model in JAGS
```

```
sink("CJS_StateSpace.txt")
cat("
model {

# Priors and constraints
S.est ~ dunif(0, 1)
p.est ~ dunif(0, 1)

# Likelihood
for (i in 1:marked){
   z[i,1] <- 1 # Latent state 1 (alive) at time of release (time 1 here)
   for (t in 2:n.occasions){
      # State process. If z[t-1]=1, prob=S. If z=0, prob=0
      z[i,t] ~ dbern(z[i,t-1]*S.est)

      # Observation process. If z[t]=1, prob=p. If z=0, prob=0
      CH[i,t] ~ dbern(z[i,t]*p.est)
      } #t
   } #i
}
",fill = TRUE)
sink()

# Bundle data
jags.data <- list(CH = CH, marked = marked, n.occasions = n.occasions)

# Initial values for latent state z
init.z <- function(ch){
  state <- ch
  for (i in 1:dim(ch)[1]){
    n1 <- min(which(ch[i,]==1)) # Column with first detection (col 1 here)
    n2 <- max(which(ch[i,]==1)) # Last detection (so known to be alive between n1 and n2)
    state[i,n1:n2] <- 1
    state[i,n1] <- NA
  }
  state[state==0] <- NA
  return(state)
}

jags.inits <- function(){list(S.est = runif(1, 0, 1), p.est = runif(1, 0, 1),
                              z = init.z(CH))}

# Parameters monitored
jags.params <- c("S.est", "p.est")
```

```
model.file <- 'CJS_StateSpace.txt'

jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 4000, n.burnin=2000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

The JAGS code for fitting the model uses our standard uninformative prior distribution for the two probabilities (survival, detection probability). The likelihood mirrors the simulation code. The matrix z contains the latent state, which is sometimes known but inferred in situations where the pattern is ambiguous. JAGS requires good initial values (or will terminate execution if it encounters an initial value that is inconsistent with the observations, such as detection of a fish that the model considers to be dead). To avoid that, we use the **?** approach where the latent state is set to 1 between the first and last detections of an individual (user-defined function init.z()).

Estimates appear to be very reliable for this scenario, where capture histories for 50 fish are used to estimate only two time-independent parameters. Would you obtain usefully precise results with a sample size of 25 or 30 fish? How does reliability change as you reduce the assumed capture probability?

One important advantage of this telemetry method is that estimates are essentially produced in real-time and on any time scale. For example, the method could be used to estimate average annual survival over the first few years after a regulation change. Even better would be to modify the code to generate annual survival estimates, so that monitoring could be done over a block of years with a management action taken midway.

6.3 Tag-based

Survival can be estimated by a multi-period study using tag returns. An important advantage of this approach is that field effort is limited to the tagging process because the fishery generates the tag returns. The design requires a release of tagged fish at the start of each period. The releases need not be the same but here we assume a constant number released for simplicity. Like the telemetry method (Section 6.2), estimates can be produced for whatever time interval is relevant (e.g., monthly, yearly).

This study design is similar to band-recovery models used in wildlife studies

(**??**). Those methods were pioneered for migratory birds, with bands returned from individuals that were harvested. In fisheries tagging studies, tags may be also reported from fish that are caught and released but here we assume that returned tags reflect only harvest. Catch-and-release complicates the analysis because the tag "dies" but the fish is not killed. **?** provide analytical methods for addressing the combination of harvest and catch-and-release.

We begin by setting up matrices for tags-at-risk (tagged fish still alive) and the outcomes from each release. The matrix of outcomes or fates is analyzed row-by-row using a multinomial distribution. For example, a fish tagged in period 1 could be caught in any period up to the end of the study. It could also end up in the final column for fish not seen again, which would account for fish that were still alive or died of natural causes or were harvested but the tag was not reported. We do not assume here any knowledge of the tag-reporting rate. If that information were available (e.g., through use of high-reward tags that provide an assumed reporting rate of 100%), then it would be possible to partition total mortality into fishing and natural component rates (**?**). Here we keep things simple and estimate only the tag-recovery rate, which is the product of the exploitation rate and tag-reporting rate. In planning a tag-return study, the best results will be obtained by ensuring a tag-reporting rate close to 100%. Non-reporting results in lost information (a reduced sample size) and poorer precision.

```r
rm(list=ls()) # Clear Environment

# Parameter values for three-yr experiment
Periods <- 3  # Equal number of release and return periods assumed
NTags <- 1000 # Number released each period
S.true <- 0.8 # Survival rate
r.true <- 0.1 # Tag recovery rate (exploitation rate * tag-reporting rate)

# Calculated value(s), array creation
TagsAtRisk <- array(data=NA, dim=c(Periods, Periods))
TagFate <- array(data=NA, dim=c(Periods, Periods+1))   # Extra column for tags not seen again

for (i in 1:Periods){ # Expected values for tags-at-risk, returned/not-seen-again
    TagsAtRisk[i,i] <- NTags
    TagFate[i,i] <-  TagsAtRisk[i,i] * r.true # Returns in release period
  j <- i+1
  while(j <= Periods) {
      TagsAtRisk[i,j] <- TagsAtRisk[i,(j-1)]*S.true
      TagFate[i,j] <- TagsAtRisk[i,j] * r.true
      j <- j+1
     } #while
```

```
   TagFate[i, Periods+1] <- NTags-sum(TagFate[i,i:Periods]) # Tags not seen again
  } #i

#Random trial (tags returned by period or not seen again)
RandRet <- array(data=NA, dim=c(Periods, Periods+1))
  for (i in 1:Periods) # Create upper diagonal matrix of random outcomes
  {
    RandRet[i,i:(Periods+1)] <- t(rmultinom(1, NTags, TagFate[i,i:(Periods+1)]))
  } #i
```

Tags at risk are contained in an upper triangular matrix, with the diagonal element being the number released (NTags) and subsequent elements obtained as the previous number at risk multiplied by the survival rate. We arbitrarily assume Periods=3, a true survival rate (S.true) of 0.8 and a tag-recovery rate of 0.1. You should inspect the matrix by clicking on the name in the Environment window. The above parameters result in a vector of 1000, 800, and 640 tags at risk from a release in period 1. Cells in the matrix of fates are obtained as the number at risk times the tag-recovery rate, plus a final column for fish not seen again. The number released (1000) is arbitrary (and ambitious), but using a large value is helpful for reviewing the calculations that produce the two matrices. The parameter values can be changed to mimic any planned study design.

This section of code introduces a while loop. This different looping construct is needed because all rows except the last one include periods after the release. We need to be able to skip the additional processing in the last row, which has only the diagonal element. The final column for each row is the not-seen-again calculation. That value is highest for the most recent release, because most of those fish will still be alive. That uncertainty about whether fish are still at risk or have died is reduced by extending the number of periods for tag returns. Our example has the same number of releases and return periods, but the code could be modified to allow for additional periods of tag returns.

Inspecting the matrix of fates is helpful for understanding how survival is estimated. The expected tag returns within any column form ratios that equal the survival rate. For example, we would expect 80 tags returned in the second period from the first release compared to 100 from the second release, because the first release has been at large for one additional period. Similarly the ratio from successive rows of returns in period 3 also equal the survival rate. Thus we have several cells (and ratios) that provide information about survival, which we assumed to be constant over the three periods.

The final lines of code in this section introduce the stochasticity associated with a multinomial distribution. The matrix of fates (expected values) provide the cell probabilities (once rescaled, internal to the rmultinom() function).

It is instructive to compare the expected values and random realizations to see the level of variability associated with the multinomial distribution. Now the ratios of random values within a column no longer equals (but will vary around) the true survival rate.

```
# Load necessary library
library(rjags)
library(R2jags)

# Specify model in JAGS
sink("TagReturn.txt")
cat("
model {
  # Priors
  S.est ~ dunif(0,1) # Time-independent survival rate
  r.est ~ dunif(0,1) # Time-independent tag-recovery rate

  # Calculated value
  A.est <- 1-S.est # Rate of mortality

# Cell probabilities
for (i in 1:Periods) {
   p.est[i,i] <- r.est
   for (j in (i+1):Periods) {
      p.est[i,j] <- p.est[i,(j-1)] * S.est
      } #j
    p.est[i,Periods+1] <- 1 - sum(p.est[i, i:Periods])   # Prob of not being seen again
    } #i

# Likelihood
  for (i in 1:Periods) {
    RandRet[i,i:(Periods+1)] ~ dmulti(p.est[i,i:(Periods+1)], NTags)
    }#i
 } # model

",fill=TRUE)
sink()

     # Bundle data
  jags.data <- list("RandRet", "Periods", "NTags")

  S.init <- runif(1,min=0,max=1)
  r.init <- 1-S.init
```

```
# Initial values
jags.inits <- function(){list(S.est = S.init, r.est=r.init)}

model.file <- 'TagReturn.txt'

# Parameters monitored
jags.params <- c("S.est", "r.est", "A.est")

 # Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 4000, n.burnin=2000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

The analysis begins by establishing the usual uninformative prior distributions for the two rates. We calculate the total mortality rate (A.est) as the complement of the survival rate (1-S.est). One important advantage of a Bayesian analysis is that we automatically obtain measures of uncertainty for functions of model parameters. Here the function is simply 1-S.est, but this advantage applies regardless of the function's complexity.

The likelihood calculations (dmulti()) requires cell probabilities for the multinomial distribution, so we calculate those probabilities directly rather than keeping track of numbers of tags by fate. The probability of a tag from release i being returned in period j is p.est[i,j] <- p.est[i,(j-1)] * S.est. Make sure that you understand this expression for specific values of i and j. Can you think of another way of coding this calculation? Note that for loops work differently in JAGS compared to R, so we are able to use them for row and column processing. In JAGS, the 'j' loop is not executed when i=3 because j>Periods. Note that our initial value for the tag-recovery rate is obtained as 1-S.init. The tag recovery rate cannot exceed 1-S.init, and would typically be less due to the other possible outcome, natural death.

Estimates are good for this case, which is not surprising given the very large releases and only two time-independent parameters to estimate. The estimated number of effective parameters was 1.9 for one run, which is consistent with the two apparent model parameters (S.est, r.est) and A.est which is simply a function of S.est.

If you think about a local system with which you are familiar, how practical would it be to tag 1000 fish at the start of each period (e.g., annually)? If you were responsible for planning the tagging operation, what sampling gear would you recommend? Would there be any potential concern about mortality due to handling? What size field crew would be needed and how much time would the tagging operation require? Could you use simulation to explore different sample sizes before recommending a specific field design?

6.4 Exercises

1. For the default age-composition analysis without gear selectivity (Section 6.1), compare credible intervals for survival rate at sample sizes of 30 versus 200 fish.
2. For the telemetry method of estimating survival (Section 6.2), approximately what sample size would produce an estimate with the levels of precision recommended by **?** for management (25%) or preliminary studies (50%)?
3. For the tag-return method (Section 6.3), compare credible intervals for survival rate at releases of 50, 100, and 200 fish. How do the results compare to the **?** recommendations? In what ways is this simulated field study optimistic about the reliability of results?

7

Mortality Components

7.1 Exploitation rate

A tagging study can provide a real-time estimate of the exploitation rate, or the fraction of the population being harvested. Similar to the tag-return method of estimating survival (Section 6.3), the only field work required is the initial tagging effort. Here there is a single release of fish with external tags at the start of the study, and the number of returned tags is recorded over a specified interval. The study duration could be an entire year, or a fishing season for a population managed by seasonal harvest. A high exploitation rate might indicate that harvest regulations such as a size limit or creel limit could be worthwhile. Alternatively a low estimate might indicate that other factors such as natural mortality or recruitment may be more important in regulating population size. With only a single release, we cannot estimate survival (or its complement, mortality) but it is nevertheless an important first step in judging the relative impact of fishing.

There are several practical issues to consider in planning an exploitation rate study. We want to ensure that essentially all tags of harvested fish are reported, so high-reward tags (e.g., $100) could be used (**?**). Tag loss is another potential source of bias. It can be estimated by double-tagging (giving some or all fish two tags, and recording returns of two versus only one tag), but it greatly complicates the analysis. For this example, we will keep things simple and assume that tag loss is negligible. We also assume that there is no immediate tagging mortality (i.e., due to capture, handling and tagging). This can be challenging to investigate. It is sometimes estimated by holding a sample of fish in a cage after tagging but this can itself be a source of mortality (**?**). For now, we assume that there is no short- or long-term mortality associated with tagging. We assume that all caught fish are harvested. If catch-and-release was occurring, our model could be extended to estimate rates of harvest and catch-and-release. The decision of how many fish to tag will depend on the required precision of the study, and as usual can be examined through simulation. Finally, it is important that tagged fish be representative of the entire population. This refers not only to size or age, but also the spatial pattern. Tagging could be done at randomly selected locations, or when the entire study population is concentrated within a single area (e.g., winter holding area or spawning location). The key is for all fish, including the tagged subset, to have the same probability of being harvested.

Our simulation code uses an assumed exploitation rate (u) of 0.4. The value could be based on a literature review or prior work from the study area or similar areas regionally. The number of returned tags has a binomial distribution:

```r
rm(list=ls()) # Clear Environment

# Tagging study to estimate exploitation rate
u.true <- 0.4
n.tagged <- 100
n.returned <- rbinom(n=1, prob=u.true, size=n.tagged)
```

We need only a single observation for the returned number of tags. What two lines of code could you add to look at the distribution of returned tags (using a new variable for the vector of tag returns)?

The JAGS code is also extremely simple. We have one line of code for the uninformative prior distribution for the exploitation rate, and one for the likelihood.

```r
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code
sink("ExpRate.txt")
cat("
model{
    # Priors
    u.est ~ dunif(0,1)  # Exploitation rate

    # Likelihood
    n.returned ~ dbin(u.est, n.tagged)
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("n.returned", "n.tagged")
```

```r
# Initial values
jags.inits <- function(){ list(u.est=runif(n=1, min=0, max=1))}

model.file <- 'ExpRate.txt'

# Parameters monitored
jags.params <- c("u.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 4000, n.burnin=2000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

A single run with Ntags=100 provided a credible interval of 0.27-0.46 (from a random return of 36 tags). Experiment with different sample sizes and consider what level of precision would be adequate for management purposes. As always, this is an optimistic scenario, with the number of returned tags perfectly conforming to a binomial distribution.

7.2 Completed tagging study

A single release of tags (Section 7.1) can also be used to estimate the rate of natural death. **?** provide a method that can be used in completed tagging studies; that is, a study of sufficient duration that no live tagged fish remain (and tag returns have ceased). Their method requires only that fishing mortality in each period be greater than zero, in order to establish the study's endpoint. It is assumed that tags of harvested fish are always reported, there is no mortality associated with tagging, and there is no tag loss. The model assumes a constant rate of natural mortality, so the estimated rate would in practice represent the average over the study period.

The following Bayesian version is a modification of the **?** method. We allow for zero harvest but assume that tag returns are monitored for a sufficiently long time to confirm the true study length (i.e., to allow for the possibility of an interspersed 0). The **?** model uses instantaneous rates, which we have to this point avoided. The relationship between probabilities (survival, death, exploitation, natural death) and instantaneous rates is as follows. The instantaneous rate of population decline is $dN/dt = -Zt$, where N is population size and Z is the instantaneous total mortality rate (rate of decline due to fishing and natural sources). An advantage of instantaneous rates is that they are additive. We can partition Z into whatever categories are useful and estimable; for example, fishing versus natural mortality, commmercial versus recreational fishing, longline versus trawl, etc. The most common partitioning is for fishing (F) versus natural (M) sources: $dN/dt = -(M + F)t$. Integrating this equation provides us with an expression for population size at any time t: $N_t = N_0 * exp(-(M + F) * t)$ where N_0 is initial abundance and $S = exp(-(M + F) * t)$ is the survival rate. The probability of dying from all causes is A=1-S. The exploitation rate is obtained as u=FA/Z, which makes sense as the fraction of total deaths due to fishing (F/Z). Similarly the probability of natural death is v=MA/Z. These expressions make it possible to go back and forth between probabilities and instantaneous rates, depending on which is more useful or traditional for a particular model. These expressions apply for the most common situation where fishing and natural mortality occur simultaneously (referred to by **?** as a Type 2 fishery). **?** provides alternate expressions for the less common case (a Type 1 fishery) where natural mortality occurs after a short fishing season.

Unlike the probabilities for survival, harvest, etc., instantaneous rates are unbounded. In a Bayesian context, this makes them more challenging to estimate compared to probabilities, which are conveniently bounded by 0 and 1. Instantaneous rates are also less intuitive in terms of their magnitude compared to probabilities. There is typically more uncertainty about M than F because natural deaths are almost never observed. The joke about M is that it is often assumed to be 0.2, because .2 looks like a distored version of a question mark. In a review of actual published natural mortality estimates, **?** found that a majority were less than 0.5 but values exceeding 2.0 were occasionally reported.

We begin with simulation code:

```r
rm(list=ls()) # Clear Environment

n.tagged <- 100
Periods <- 20 # Set to high value to ensure that end of study is observed
F.true <- 0.4
M.true <- 0.2
S.true <- exp(-F.true - M.true)
u.true <- F.true * (1-S.true) / (F.true + M.true)

TagsAtRisk <- array(data=NA, dim=Periods)
TagsAtRisk[1] <- n.tagged
TagFates <- array(data=NA, dim=Periods) # Returns by period
```

```
for (j in 2:Periods){
  TagsAtRisk[j] <- rbinom(n=1, size=TagsAtRisk[j-1], prob=S.true)
  TagFates[j-1] <- rbinom(n=1, size=(TagsAtRisk[j-1]
                      -TagsAtRisk[j]),(F.true/(F.true+M.true))) # Random realization: fishing de
  } #j
TagFates[Periods] <- rbinom(n=1, size=(TagsAtRisk[Periods-1]
                      -TagsAtRisk[Periods]),(F.true/(F.true+M.true)))
t.Completed <- max(which(TagFates[]>0)) # Locate period when study completed (last return)
NatDeaths <- n.tagged - sum(TagFates)
TagFates <- c(TagFates[1:t.Completed],NatDeaths) # Returns + all natural deaths
```

The vectors for tags-at-risk and fates are binomially distributed random realizations. Tags at risk are obtained using the number at risk in the previous period and the survival rate. Tag returns (stored in tag fates vector) are obtained from the number of deaths between periods and the fraction of deaths due to fishing (F/Z). The which() function provides a vector of periods when tag returns are greater than 0, and the max function chooses the final period with at least one tag return. Try which(TagFates[]>0) in the Console to make clear how this expression works. The final version of the tag fates vector contains tag returns for the completed study length plus a final column for natural deaths (any tagged fish not seen in the completed tagging study).

The JAGS code for fitting the model is similar to that for the tag-based method of estimating survival (Section 6.3), except that our parameters now are instantaneous rates. We use an arbitrary uniform prior distribution, with an upper bound (2) set high enough to ensure that the interval contains the true value. The likelihood uses a multinomial distribution, where cell probabilities represent the fraction returned in each period plus the final column for fish not seen again (natural deaths in our completed study).

```
# Load necessary library
library(rjags)
library(R2jags)

######### JAGS code for estimating model parameters
sink("Hearn.txt")
cat("
  model {
  # Priors
  M.est ~ dunif(0, 2)  # Instantaneous natural mortality rate
  for (i in 1:t.Completed) {
     F.est[i] ~ dunif(0,2) # Instantaneous fishing mortality rate
     S.est[i] <- exp(-M.est - F.est[i])
     u.est[i] <- F.est[i] * (1-S.est[i])/(F.est[i]+M.est) # FA/Z
  } #i

# Cell probabilities
  p.est[1] <- u.est[1]
  for (i in 2:t.Completed) {
    p.est[i] <- prod(S.est[1:(i-1)])*u.est[i]
      } #i
    p.est[t.Completed+1] <- 1 - sum(p.est[1:t.Completed])   # Prob of not being seen again
  TagFates[1:(t.Completed+1)] ~ dmulti(p.est[1:(t.Completed+1)], n.tagged)
 }
  ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("TagFates", "t.Completed", "n.tagged")

# Initial values
jags.inits <- function(){list(M.est=runif(1, 0, 2), F.est=runif(t.Completed, 0, 2))}

model.file <- 'Hearn.txt'

# Parameters monitored
jags.params <- c("M.est", "F.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.iter = 4000, n.burnin=2000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

This approach works because there are only two fates for tagged fish: caught and reported or natural death. Using a completed tagging study means that the number of natural deaths is known exactly (number of tag releases - total returns). The model adjusts the estimate of M in order to account for (get rid of) all the natural deaths by the end of the completed study. The length of the completed study depends on the assumed values for F and M. The estimate of M will be higher when the study length is short and when many tagged fish are not seen again.

For the default settings, estimates of M tend to be reliable, which is unsurprising given that it is modeled as a constant rate and estimated from multiple years of data. How does uncertainty about M change as you increase the number of tags released (e.g., 200 or 1000)?

Estimates of F are reliable in the first few years but uncertainty increases dramatically towards the end of the completed study. This makes sense because the estimates of tags at risk are further and further from the known initial tagged population. There would likely be some improvement from simplifying the model to estimate only the average F rather than period-specific estimates, given that the true simulated F values vary stochastically around a fixed value.

7.3 Multi-year tagging study

We return to the multi-year tagging study (Section 6.3) but now assume that we have information about the tag-reporting rate. This information allows us to partition mortality into fishing and natural sources. Tag-reporting rate is fixed at 1.0 in the following code, as could be appropriate when using high-reward tags. In a later section of code, we estimate it internally, using planted tags (**?**) and an additional likelihood component. Information about the tag-reporting rate is very powerful. For example, a 100% tag-reporting rate means that (assuming no other assumptions are violated) you ultimately know the number of natural deaths because all other tags will be reported (as in Section 7.2). There is uncertainty in the short term because tags not returned could belong to fish still alive and at risk, so information about natural deaths is clear only after sufficient time has passed.

Our simulation code is similar to that of Section 6.3 except that we now specify the probabilities of harvest (u.true) and natural death (v.true). We use vectors so that the rates can vary by period. We have arbitrarily set the probabilities for harvest higher than for natural death, so it is useful to investigate whether we can reliably detect the predominant threat in our simulated study design.

```r
rm(list=ls()) # Clear Environment

# Parameter values for three-yr experiment
Periods <- 3  # Equal number of release and return periods assumed
NTags <- 1000 # Number released each period
NumTagged <- rep(NTags, Periods) # Release NTags fish each period

u.true <- c(0.4, 0.25, 0.3) # Probability of fishing death
v.true <- c(0.1, 0.15, 0.2) # Probability of natural death
```

```
lam.true <- 1  # Reporting rate

# Calculated value(s), array creation
S.true <- 1 - u.true - v.true
TagsAtRisk <- array(data=NA, dim=c(Periods, Periods))
TagFate <- array(data=NA, dim=c(Periods, Periods+1))   # Extra column for tags not seen again

for (i in 1:Periods){ # Expected values for tags-at-risk, returned/not-seen-again
  TagsAtRisk[i,i] <- NumTagged[i]
  TagFate[i,i] <-  TagsAtRisk[i,i] * lam.true * u.true[i] # Returns in release period
  j <- i+1
  while(j <= Periods) {
    TagsAtRisk[i,j] <- TagsAtRisk[i,(j-1)]*S.true[j-1]
    TagFate[i,j] <- TagsAtRisk[i,j] * lam.true * u.true[j]
    j <- j+1
   } #while
  TagFate[i, Periods+1] <- NTags-sum(TagFate[i,i:Periods]) # Tags not seen again
  } #i

RandRet <- array(data=NA, dim=c(Periods, Periods+1))  # Extra column for fish not seen again
#Random trial
  for (i in 1:Periods) # Create upper diagonal matrix of random tag returns
  {
    RandRet[i,i:(Periods+1)] <- t(rmultinom(1, NumTagged[i], TagFate[i,i:(Periods+1)]))
  } #i
```

The analysis is again similar to Section 6.3 except that now we now have a multinomial distribution with three possible fates (survival, harvest, natural death). Following **?** (their Section 9.6), we use three gamma-distributed parameters that are scaled by their sum, thus ensuring that they sum to 1. The parameters of interest (u, v, S) are calculated functions of the “a” parameters, which are used internally but not themselves of interest.

```
# Load necessary library
library(rjags)
library(R2jags)

  ######### JAGS code for estimating model parameters
  sink("TagReturn.txt")
  cat("
  model {
  # Priors
  lam.est <- 1 # Reporting rate assumed known (e.g. high-reward tags only)
  for (i in 1:Periods) {
     for (j in 1:3) # Rows are return periods, columns are fates
       {
         a[i,j] ~ dgamma(1,1)
       } #j
     S.est[i] <- a[i,1]/sum(a[i,]) # Probability of survival
     u.est[i] <- a[i,2]/sum(a[i,]) # Probability of harvest (exploitation rate)
     v.est[i] <- a[i,3]/sum(a[i,]) # Probability of natural death
     # v.est[3] could also be obtained as 1-S.est[3]-u.est[3]
  } #i

# Cell probabilities
  for (i in 1:Periods) {
    p.est[i,i] <- lam.est * u.est[i]
    for (j in (i+1):Periods) {
      p.est[i,j] <- prod(S.est[i:(j-1)])*lam.est*u.est[j]
      } #j
    p.est[i,Periods+1] <- 1 - sum(p.est[i, i:Periods])   # Prob of not being seen again
    } #i
  for (i in 1:Periods) {
  RandRet[i,i:(Periods+1)] ~ dmulti(p.est[i,i:(Periods+1)], NumTagged[i])
  }#i
 }
  ",fill=TRUE)
  sink()

# Bundle data
  jags.data <- list("RandRet", "Periods", "NumTagged")

# Initial values
  a.init <- array(data=NA, dim=c(Periods, 3))
  for (i in 1:3){
    a.init[i,] <- runif(n=3, min=0, max=1)
    a.init[i,] <- a.init[i,]/sum(a.init[i,]) # Rescale to sum to 1
  }
  jags.inits <- function(){ list(a=a.init)}

  model.file <- 'TagReturn.txt'

  # Parameters monitored
  jags.params <- c(#"lam.est",
                  "u.est", "v.est")
  #, "S.est")
  # Call JAGS from R
  jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                  n.chains = 3, n.thin = 1, n.iter = 4000, n.burnin=2000,
                  model.file)
  print(jagsfit)
  plot(jagsfit)
```

We provide arbitrary starting values for the “a” parameters by rescaling uniform (0,1) random variates. Using a large release on each occasion (1000), our estimates are generally close to the true values except that v.est[3] is imprecise and typically overestimated. It is the most difficult time-dependent parameter to estimate. We get direct information (from returned tags) about the final exploitation rate but there is much uncertainty about whether fish not seen were still at risk or were natural deaths. This is especially true for the final release because most fish are still at large (examine the TagFate and RandRet matrices). Note that this differs from a completed tagging study (Section 7.2), where the study continues until no live fish remain.

The analysis can be extended to include estimation of the tag-reporting rate. We use the simplest approach of a binomial experiment using planted tags. The fraction of planted tags that are returned is a direct estimate of the reporting rate. The only change required in the simulation code is to replace the assignment lam.true <- 1.0 with the following three lines:

```
lam.true <- 0.9  # Reporting rate
PlantedTags <- 30
PlantReturns <- rbinom(n=1, size=PlantedTags, prob=lam.true)
```

The assumed reporting rate and number of planted tags are arbitrary and can be varied to see the effect on parameter uncertainty.

```
# Load necessary library
library(rjags)
library(R2jags)

  ######### JAGS code for estimating model parameters
  sink("TagReturn.txt")
  cat("
  model {
  # Priors
  lam.est ~ dunif(0,1) # Reporting rate
  for (i in 1:Periods) {
     for (j in 1:3) # Rows are return periods, columns are fates
       {
         a[i,j] ~ dgamma(1,1)
       } #j
     S.est[i] <- a[i,1]/sum(a[i,]) # Probability of survival
     u.est[i] <- a[i,2]/sum(a[i,]) # Probability of harvest (exploitation rate)
     v.est[i] <- a[i,3]/sum(a[i,]) # Probability of natural death
     # v.est[3] could also be obtained as 1-S.est[i]-u.est[i]
  } #i

# Cell probabilities
  for (i in 1:Periods) {
    p.est[i,i] <- lam.est * u.est[i]
    for (j in (i+1):Periods) {
      p.est[i,j] <- prod(S.est[i:(j-1)])*lam.est*u.est[j]
      } #j
    p.est[i,Periods+1] <- 1 - sum(p.est[i, i:Periods])   # Prob of not being seen again
    } #i
  for (i in 1:Periods) {
  RandRet[i,i:(Periods+1)] ~ dmulti(p.est[i,i:(Periods+1)], NumTagged[i])
  }#i
  PlantReturns ~ dbin(lam.est, PlantedTags) # Additional likelihood component
 }
  ",fill=TRUE)
  sink()

# Bundle data
  jags.data <- list("RandRet", "Periods", "NumTagged", "PlantedTags", "PlantReturns")

# Initial values
  a.init <- array(data=NA, dim=c(Periods, 3))
  for (i in 1:3){
    a.init[i,] <- runif(n=3, min=0, max=1)
    a.init[i,] <- a.init[i,]/sum(a.init[i,])  # Rescale to sum to 1
  }
  jags.inits <- function(){ list(a=a.init)}

  model.file <- 'TagReturn.txt'

  # Parameters monitored
  jags.params <- c("lam.est", "u.est", "v.est")
  #, "S.est")
 # Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin = 1, n.iter = 10000, n.burnin=5000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

The auxiliary planted tag experiment requires only a few changes in the JAGS code. We provide an uninformative prior distribution for the now estimated parameter lam.est, and pass in the number of planted tags and returns. We let JAGS generate an initial value for lam.est. The additional likelihood component describes the binomial experiment. Convergence may be slower for this version of the code, so the number of MCMC iterations was increased to 10,000.

Estimates have similar precision to the original example if the reporting rate is high. Precision is reduced at lower reporting rates because of the reduced sample size of returned tags and uncertainty about lambda.

7.4 Telemetry estimates of F and M

In Section 6.2, we used fixed receiving stations and telemetry tags to estimate the survival rate. Here we extend that approach to partition total mortality into fishing and natural components by giving telemetered fish an external high-reward tag (Design C of **?**). Live fish "detect themselves" and provide information on survival by passing within range of a receiver. Harvest information comes from the reported high-reward tags. Thus we have observations on two of the three possible states, with natural mortality being the only unobserved true state. An important practical advantage of this approach is that field surveys are not needed after tagging is done (other than maintaining the receiver array and downloading detection data).

We begin with a detour to introduce the JAGS dcat distribution, used to estimate probabilities for observations from different categories.

```
rm(list=ls()) # Clear Environment

# Load necessary library
library(rjags)
```

```r
library(R2jags)

# JAGS code for estimating model parameters
sink("dcat_example.txt")
cat("
model {

# Priors
    for (j in 1:3) {
         a[j] ~ dgamma(1,1)
         } #j
     p[1] <- a[1]/sum(a[])
     p[2] <- a[2]/sum(a[])
     p[3] <- a[3]/sum(a[]) # Could also be obtained as 1-p[1]-p[2]

# Likelihood
  for (i in 1:N){
    y[i] ~ dcat(p[])
  } #i
}
    ",fill = TRUE)
sink()

# Bundle data
y <- c(1, 1, 3, 3, 2) # Each observation drawn from category 1, 2, or 3
N <- length(y)
jags.data <- list("N", "y")

model.file <- 'dcat_example.txt'

# Parameters monitored
jags.params <- c("p")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=NULL,
                n.chains = 3, n.thin=1, n.iter = 2000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

Our dcat example has a y vector with five observations in the range of 1 to 3. The categories can be of any sort; for example, age or species code for individual fish. The model parameters of interest are calculated variables representing the probabilities of the three categories. The probabilities sum to 1, so as in Section 7.3, we calculate them using an “a” vector with an uninformative gamma prior distribution. (A Dirichlet distribution can also be used as an uninformative prior distribution for probabilities that must sum to 1 (McCarthy, 2007).) Returning to the purpose of this Section, the p vector could represent probabilities of a live detection, harvest, or not being detected. The MCMC process adjusts the p estimates to achieve the best possible match between model predictions and the observations (y vector). For simplicity, we let JAGS provide the initial values for the a vector. The estimates have wide credible intervals because of the small sample size.

Returning to the telemetry example, we begin with simulation code, using arbitrary values for the number of occasions and sample size. Varying the patterns for rates of exploitation and natural death is helpful in judging the model’s ability to detect moderate time variation. This is a multistate model (**?**), with a matrix (PSI.STATE) describing the various true states and the probabilities for transitioning between states over time. The three rows represent true states alive, harvest, and natural death. A fish that is alive (row 1) can remain alive (probability S.true), be harvested (probability u.true), or become a natural mortality (probability v.true). A harvested fish (row 2) remains in that state with probability 1, as does a natural death (row 3). Note that the third matrix dimension for PSI.STATE is time, as the rates are time-dependent.

The observation matrix (PSI.OBS) has a similar structure. Rows are true states, columns are observed states, and the third dimension is time. There are three possible observed states: detected alive, harvested, and not detected. A live fish (row 1) can either be detected or not, with probabilities p[t] and 1-p[t]. A harvested fish (row 2) is assumed to be always detected (100% reporting), and a natural death (row 3) is never detected.

The function for generating the simulated capture-history matrix (simul.ms) is complex, but key features are as follows. All fish are assumed to be tagged (transmitters and external tags) at time 1. Next we loop over occasions and use a multinomial trial to determine the true state transition between time t-1 and t (i.e., a live fish at time t-1 can remain alive, be harvested, or transition to natural death). To better understand the multinomial trial, try a simpler example in the Console. For example, rmultinom(n=1, size=1, prob=c(0.7, 0.25, 0.05)) is a single trial (n=1) for one individual (size=1), with three states at the specified probabilities. The function returns a matrix with a 1 in a randomly determined row. The which() function locates the “1” and returns the row index (= fish’s state). This example could represent probabilities that a live fish at time t-1 survives (0.7), is harvested (0.25), or is a natural death (0.05). Returning to the code, a second multinomial trial based on the fish’s true state draws a random observed outcome (e.g., a live fish is either detected or missed).

```r
# Code modified from Kery and Schaub 2012, Section 9.5

rm(list=ls()) # Clear Environment

# Generation of simulated data
# Define probabilities as well as number of occasions, states, observations and released individua
n.occasions <- 5
n.tagged <- 100  # Simplify from Kery's version, only an initial release, and no spatial states

u.true <- c(0.2, 0.1, 0.3, 0.2)
v.true <- c(0.1, 0.15, 0.15, 0.1)
S.true <- 1-u.true-v.true
p <- c(1, 0.8, 0.8, 0.6, 0.7)
#  Detection probability fixed at 1 for first period, n.occasions-1 searches (e.g start of period 

n.states <- 3
n.obs <- 3

# Define matrices with survival, transition and detection probabilities

# 1. State process matrix. Three-dimensional matrix. 1=state, time t, 2=state, time t+1, 3=time
PSI.STATE <- array(NA, dim=c(n.states, n.states, n.occasions-1))
   for (t in 1:(n.occasions-1)){
      PSI.STATE[,,t] <- matrix(c(
      S.true[t], u.true[t], v.true[t],
      0, 1, 0,
      0, 0, 1), nrow = n.states, byrow = TRUE)
      } #t

# 2.Observation process matrix.  1=true state, 2=observed state, 3=time
PSI.OBS <- array(NA, dim=c(n.states, n.obs, n.occasions))
   for (t in 1:n.occasions){
      PSI.OBS[,,t] <- matrix(c(
      p[t], 0, 1-p[t],
      0,  1, 0,  # Caught w/ high-reward tag, assume 100% reporting
      0, 0, 1), nrow = n.states, byrow = TRUE)  # Natural deaths never detected
      } #t

# Define function to simulate multistate capture-recapture data
simul.ms <- function(PSI.STATE, PSI.OBS, n.tagged, n.occasions, unobservable = NA){
   # Unobservable: number of state that is unobservable
   CH <- CH.TRUE <- matrix(NA, ncol = n.occasions, nrow = n.tagged)
   CH[,1] <- CH.TRUE[,1] <- 1 # All releases at t=1
   for (i in 1:n.tagged){
     for (t in 2:n.occasions){
         # Multinomial trials for state transitions
         CH.TRUE[i,t] <- which(rmultinom(n=1, size=1,
                                         prob=PSI.STATE[CH.TRUE[i,t-1],,t-1])==1)
         # which vector element=1; i.e., which state gets the random draw
         # at time t given true state at t-1
         # True state determines which row of PSI.STATE provides the probabilities

         # Multinomial trials for observation process
         CH[i,t] <- which(rmultinom(1, 1, PSI.OBS[CH.TRUE[i,t],,t])==1)
         # which observation gets the 1 random draw, given true time t state.
         } #t
      } #i

    return(list(CH=CH, CH.TRUE=CH.TRUE)) # True (CH.TRUE) and observed (CH)
   } # simul.ms
```

The JAGS code is also complex but does generally mirror the simulation code.

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code for estimating model parameters
sink("Mort_Tel_HRTag.txt")
cat("
model {

# Priors
for (t in 1:(n.occasions-1)) {
     for (j in 1:3) { # Rows are return periods, columns are fates
         a[t,j] ~ dgamma(1,1)
         } #j
     S.est[t] <- a[t,1]/sum(a[t,]) # Probability of survival
     u.est[t] <- a[t,2]/sum(a[t,]) # Probability of harvest (exploitation rate)
     v.est[t] <- a[t,3]/sum(a[t,]) # Probability of natural death
     # v.est[t] could also be obtained as 1-S.est[t]-u.est[t]
    } #t

  p[1] ~ dbern(1)
  for (t in 2:n.occasions){
    p[t] ~ dunif(0,1)
    }#t

# Define state-transition and observation matrices
# Probabilities of true state at t+1 given state at time t
    for (t in 1:(n.occasions-1)){
    ps[1,t,1] <- S.est[t]
    ps[1,t,2] <- u.est[t]
    ps[1,t,3] <- v.est[t]
    ps[2,t,1] <- 0
    ps[2,t,2] <- 1
    ps[2,t,3] <- 0
    ps[3,t,1] <- 0
    ps[3,t,2] <- 0
    ps[3,t,3] <- 1
    } #t

# Probabilities of observed states given true state
    for (t in 2:n.occasions){
    po[1,t,1] <- p[t]  # Row 1 true state = alive
    po[1,t,2] <- 0
    po[1,t,3] <- 1-p[t]
    po[2,t,1] <- 0     # Row 2 true state = harvested
    po[2,t,2] <- 1
    po[2,t,3] <- 0
    po[3,t,1] <- 0     # Row 3 true state= natural death
    po[3,t,2] <- 0
    po[3,t,3] <- 1
    } #t

    # Likelihood
    for (i in 1:n.tagged){
    z[i,1] <- y[i,1] # Latent state known at time of tagging (t=1)
    for (t in 2:n.occasions){
    # State process: draw state at time t given state at time t-1
    z[i,t] ~ dcat(ps[z[i,(t-1)], (t-1),])
    # Observation process: draw observed state given true state at time t
    y[i,t] ~ dcat(po[z[i,t], t,])
    } #t
    } #i
}
    ",fill = TRUE)
sink()

# Generate initial values for z. Modified from Kery code for JAGS inits, age-specific example 9.5..
```

```
z.init <- function(ch) {
  # State 1=Obs 1 (alive) State 2=Obs 2 (fishing death). Start w/ known states from obs capture-hi
  # replace "3" with possible state(s)
  ch[ch==3] <- -1  # Not observed so temporarily replace w/ neg value
  ch[,1] <- NA  # Initial value not needed for release period
  for (i in 1:nrow(ch)){
    if(max(ch[i,], na.rm=TRUE)<2){
      ch[i, 2:ncol(ch)] <- 1 # Not detected dead so initialize as alive (after release period)
      } else {
      m <- min(which(ch[i,]==2))  # Period when fishing death first detected
      if(m>2) ch[i, 2:(m-1)] <- 1  # Initialize as alive up to period prior to harvest
      ch[i, m:ncol(ch)] <- 2  # Initialize as dead after detected fishing death
      } # if/else
} # i
  return(ch)
} # z.init

# Call function to get simulated true (CH.TRUE) and observed (CH) states
sim <- simul.ms(PSI.STATE, PSI.OBS, n.tagged, n.occasions)
y <- sim$CH

# Bundle data
jags.data <- list("n.occasions", "n.tagged", "y")

# Initial values.
jags.inits <- function(){ list(z = z.init(y))}

model.file <- 'Mort_Tel_HRTag.txt'

# Parameters monitored
jags.params <- c("u.est", "v.est", "p")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 5000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

The parameters for rates of survival, exploitation and natural death use the same approach as in the above dcat example.. The "a" matrix is estimated internally using a vague gamma prior distribution, then the three fractions estimate the probabilities of the three possible true states (survival, harvest, natural death), which sum to 1. The latent true states (z matrix) are estimated in the likelihood section of the code. The dcat (categorical distribution) is used to draw the true state at time t given state at time t-1, as well as the observation at each time given the true state.

One difficult part of fitting this model is obtaining initial values for the latent true states. JAGS will refuse to update model parameters if initial values are inconsistent with the observations (link)[1]. The function used here z.init() initializes all unknown true states to be “alive” unless a harvest occurs. This works because fish not detected could either be alive or a natural death, so “alive” is a valid initial value. Individuals that are ultimately harvested are known to be alive up to the period prior to harvest. It can be instructive to compare the matrices containing true and observed states with the initial values. For example, initial values can be saved by entering z.test <- z.init(y) in the Console. Entering sim$CH.TRUE[1:5,] and CH[1:5,] in the Console will display true and observed states for the first five individuals for comparison to z.test.

Results for the chosen simulation settings are moderately reliable. There are many parameters, as the model estimates true latent states as well as the probabilities for survival, harvest and natural death. It seems to consistently detect the increase in harvest rate in period 3. Uncertainty increases toward the end of the study, especially for probabilities of detection and natural death. This makes sense because fish not detected on the final occasion could either be alive or a natural death.

Perhaps the most obvious simulation settings to vary are the fairly robust sample size of tagged fish (100) and the detection probabilities (0.6 and above). Uncertainty increases markedly as detection probabilities decrease, so these simulations can be very helpful in planning the field study (e.g., number of receiver stations). The number of occasions can also be varied but it requires commensurate changes in the vectors for u, v, and p.

7.5 Exercises

1. For the exploitation rate study design (Section 7.1), modify the JAGS code to plot the posterior distribution for the exploitation rate. Include a vertical line showing the underlying true exploitation rate.
2. For the completed tagging study design (Section 7.2), modify the

[1] https://www.vogelwarte.ch/de/projekte/publikationen/bpa/code-for-running-bpa-using-jags

code to use a single F parameter (representing an average over periods).

3. For the multi-year tagging study (Section 7.3), run three replicate simulations of the version using planted tags, with tag releases of 1000 (current code), 100, and 30. How does uncertainty (e.g. credible interval width) vary? What sample size would you recommend if planning this study? Also, save credible intervals for v3 for the 1000 fish releases (for exercise 4 below).
4. Compare results for v3 (from exercise 3) with a modified design using five periods. Extend the u and v vectors by using the values from periods 1-2 for the final two periods. How do credible intervals for v3 change when the study is extended?
5. For the telemtry study (Section 7.4), compare results (particularly uncertainty) for parameter v4 when p5 takes on the following values: 0.2, 0.4, 0.7, 0.9. How might this affect planning for conducting the field work?

8

Growth

Growth is a basic aspect of a fish's life history, but it is also an important process for managing fish populations. For example, a minimum size for harvest might be used to increase the proportion of fish surviving to the size (and age) of sexual maturity. Another consideration in developing a harvest strategy is finding a balance between the growth rate (increasing size of each individual in a cohort) and the loss rate due to natural mortality (reducing the number of individuals in a cohort).

There are typically two aspects to growth modeling: age versus length and length versus weight. With those two equations, it is possible to predict the length or weight at any age, and to examine harvest strategies using minimum size or slot regulations. The focus in commercial marine fisheries is typically on weight because income from commercial fishing is a function of weight of the catch. Length gets more emphasis in freshwater fisheries management because anglers are not selling their catches (and trophy designations are generally length-based).

8.1 Growth in length

The relationship between age and length for adult fish usually is a curve of decreasing slope; that is, growth gradually slows with age as individuals approach some average maximum size. The equation most often used for age and length is the von Bertalanffy growth function: $L_t = L_\infty * (1 - e^{-k*(t-t_0)})$. This equation for size at time t has three parameters: the asymptotic size or average maximum length (L_∞), growth rate (k), and time (age) at length of 0 (t_0). The last parameter is a theoretical one rather than something that is observed. It can take on positive or negative values, and can be thought of as an extrapolation from sizes at older ages.

There are two usual sources of data for fitting a von Bertalanffy growth curve: paired age-length observations from a laboratory examination of otoliths or other hard parts that form annual marks (**?**); or a tagging study. We consider both approaches in the following sections.

8.1.1 Age-length

Age:length observations may come from a survey or from sampling a fishery. Some mis-ageing invariably occurs when interpreting otoliths or other hard parts (**?**), but for simplicity we assume that the ages are determined without error. We further assume that fish are not sampled randomly but are instead a systematic sample by age. This ensures that the data set includes less common ages (sizes), particularly older fish.

The first step in fitting the model is to generate a simulated age:length data set.

```r
# Fitting von Bertalanffy curve to simulated data
rm(list=ls()) # Clear Environment

# Choose von Bertalanffy parameter values and other simulation settings
L_inf <- 80  # e.g., 80-cm fish
k <- 0.3 # Slope
t_0 <- -0.5 # Age at length 0
MaxAge <- 12
SystSample <- 5  # Number per age group
N.AgeLen <- SystSample * MaxAge
Age <- rep(1:MaxAge, each=SystSample)
Var_L <- 10 # Variance about age-length relationship
Len <- rnorm(n=N.AgeLen,L_inf*(1-exp(-k*(Age-t_0))), sd=sqrt(Var_L))

plot(Age, Len, xlab="Age", ylab="Length (cm)")
```

The parameter values and other simulation settings provide a reasonable sample size and well-defined growth pattern and asymptote. The latter characteristic is critical in obtaining reliable results. For real data sets, it is helpful to look at the shape of the scatter plot of length versus age. Data sets with few young fish or lacking a well-defined asymptote may result in poor estimates. Note also that we are using integer ages. If there was good information about the timing of spawning, ages could be calculated as the integer age plus the fraction of the year since the spawning season. Using fractional ages would be more important for short-lived species. The model fitting process is the same whether using integer or fractional (real valued) ages.

We assume an additive error term; i.e., variability in length at age is constant. Whether an additive or multiplicative error term is appropriate can often be seen by examining a scatter plot of length versus age. A multiplicative error term would be called for when points have greater spread for older fish.

The JAGS code for fitting the model is generally similar to the simulation, except that the dnorm() function for the likelihood requires precision, defined as the inverse of the variance. Vague uniform priors are used for all four parameters. The prior distribution ranges and initial values may need to be adjusted if the range of simulated (or observed) lengths is varied. Note that max(Len) is used as a convenient upper bound for initial values of L_∞, but a fixed value greater than observed lengths could also be used.

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code
sink("GrowthCurve.txt")
cat("
model{

# Priors

 L_inf.est ~ dunif(0, 200)
 k.est ~ dunif(0, 2)
 t0.est ~ dunif(-5, 5)
 Var_L.est ~ dunif(0,100)   # Variability in length at age

# Calculated value
 tau.est <- 1/Var_L.est

# Likelihood
 for (i in 1:N.AgeLen) {
    Len_hat[i] <- L_inf.est*(1-exp(-k.est*(Age[i]-t0.est)))
    Len[i] ~ dnorm(Len_hat[i], tau.est)
 } #i
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("N.AgeLen", "Age", "Len")

# Initial values
jags.inits <- function(){ list(L_inf.est=runif(n=1, min=0, max=max(Len)),
                               k.est=runif(n=1, min=0, max=2),
                               t0.est=runif(n=1, min=-5, max=5),
                               Var_L.est=runif(n=1, min=0, max=100))}
```

```
model.file <- 'GrowthCurve.txt'

# Parameters monitored
jags.params <- c("L_inf.est", "k.est", "t0.est", "Var_L.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

Convergence is generally rapid and the estimates from simulated data are reliable. Try varying some of the simulation settings and parameter values to see how the model performs under different conditions. For example, note how the growth curve changes as you try different values for the von Bertalanffy parameters and the variance about the curve. (One caution is to ensure that the assumed additive error term does not produce negative length values. If that occurs, one solution would be to switch to a multiplicative error term, using a lognormal rather than a normal distribution (see Section 8.2.) Another setting to vary is the maximum age. A low value, as might occur if the population was heavily exploited, can make it difficult to estimate L_∞. One Bayesian approach for dealing with limited data is the use of informative prior distributions (**?**).

8.1.2 Using tagging data

Growth information can also be obtained from a tagging study (**???**). Fish seen on two or more occasions provide information on growth as a function of size at initial capture, time-at-large (interval between tagging and a subsequent recapture) and the growth increment. This approach differs from using age and length data because age at tagging is not known. However, it can be made compatible with a traditional age:length analysis by estimating relative age (true age - t_0) of all individuals as additional parameters (**???**). This allows growth to be modeled as a function of age rather than size. Flexible versions of these models can allow for separate growth parameters (L_∞, k) for each individual. Here we fit a simpler model (Model 4 of **?**) with all individuals sharing L_∞ and k. It is straightforward to modify the code to allow for either or both parameters to vary among individuals (**?**).

One important practical question is how to obtain length data when fish are encountered after tagging. Using lengths from fishers provides the greatest sample size but runs the risk of substantial measurement error (or outright bias if an angler "estimates" the length). Length data obtained by biologists limits the sample size but ensures data quality, and could be obtained through field survey sampling or fishery monitoring by creel clerks or at-sea observers. In order for our results to apply to the whole population, we also assume that growth is the same for tagged and untagged fish. The model allows for measurement error and other sources of variability (e.g., environmental, genetic) that cause individual fish lengths to vary about the fitted curve. A constant additive error is assumed here but the model could be modified to allow for a multiplicative error term. We begin with simulation code for generating the observations:

```r
rm(list=ls()) # Clear Environment

L_inf <- 80  # e.g., 80-cm fish
k <- 0.3 # Slope
Var_L <- 10 # Variance about age-length relationship
N.Tag <- 120 # Number of fish recaptured at least once
n <- rpois(N.Tag, lambda=0.1)+2 # Number of encounters for each recaptured individual, >2 uncommon
# Generate vector for relative age at initial capture (true age + t0)
Shape <- 5 # Parameters for gamma distribution generating relative ages
Rate <- 2
A <- rgamma(n=N.Tag, shape=Shape, rate=Rate) # Relative age vector
hist(A, xlab="Relative age", main="")

L <- array(data=NA, dim=c(N.Tag, max(n)))
dt <- array(data=NA, dim=c(N.Tag, max(n)-1))
for (i in 1:N.Tag){
  for (j in 1:(n[i]-1)){
    dt[i,j] <- rgamma(n=1, shape=6, rate=2) # Random dist of times at large
  }
}
hist(dt, xlab="Time at large", main="")

for (i in 1:N.Tag){
  L[i,1] <- rnorm(n=1, mean=L_inf *(1.0 - exp(-k*A[i])), sqrt(Var_L))
  for (j in 2:n[i]){
    L[i, j] <- rnorm(n=1, L_inf*(1.0 - exp(-k*(A[i]+dt[i, j-1]))), sqrt(Var_L))
  } #j
} #i

plot(A, L[,1], xlab="Relative age", ylab="Length at first capture")
plot(dt[,1], (L[,2]-L[,1]), xlab="Time at large", ylab="Growth increment")
plot(L[,1], (L[,2]-L[,1]), xlab="Length at first capture", ylab="Growth increment")
```

Choices for the growth parameters are arbitrary but should produce a well-defined asymptote when combined with the other settings (time-at-large, relative age). The sample size of fish recaptured at least once is relatively robust but can of course be varied to judge model performance given fewer or more individuals. To allow for random variation in the tagging data set, a Poisson distribution is used to generate the number of encounters for each individual. Each individual is seen at least twice but the lambda parameter can be adjusted to vary the typical number of encounters (use hist(n) to see distribution). As in previous studies (**???**), we use a gamma distribution for relative age. The two gamma parameters (shape and rate) can be varied to obtain relative age distributions with desired characteristics (as observed on the hist() plot). Here they are chosen to provide a data set of mostly younger fish. Times-at-large are also generated using a gamma distribution, with shape and rate settings chosen to obtain a realistic pattern (mostly shorter times). Three plots are used to examine the tagging data set. The first (relative age versus length) may not have a well-defined asymptote if mostly younger fish are tagged. The plot can be examined to ensure that the chosen variance in length at age is reasonable (and not generating negative lengths). The second and third plots are noisy and difficult to interpret because growth increment is a function of both time-at-large and size at tagging. They are nevertheless useful for ensuring that the data set encompasses the early faster-growing period and slowed growth near the asymptote.

The JAGS code for fitting the model uses uninformative uniform prior distributions for the two growth parameters, variance in length-at-age, and the two gamma hyperparameters that describe the distribution of relative ages.

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code
# Code modified from Scherrer et al. 2021: model 4
sink("vonBert_GrowthInc.txt")
cat("
model{
# Priors
  Var_L.est ~ dunif(0, 100) # Variance about expected length
  tau.est <- 1/Var_L.est # Precision
    k.est ~ dunif(0, 2)
    L_inf.est ~ dunif(0, 200)
    Shape.est ~ dunif(0, 100) # Gamma hyperparameters for relative age
    Rate.est ~ dunif(0, 100)
    # Likelihood
    for (i in 1:N.Tag)   {
        A.est[i] ~ dgamma(Shape.est, Rate.est) # Relative age (true age + t0)
        L_Exp[i, 1] <-   L_inf.est *(1.0 - exp(-k.est*A.est[i])) # Expected length at capture
        L[i, 1] ~ dnorm(L_Exp[i, 1], tau.est) # Length at initial capture
        for (j in 2:n[i])   { # Recapture lengths
            L_Exp[i, j] <-  L_inf.est*(1.0 - exp(-k.est*(A.est[i]+dt[i, j-1])))
            L[i, j] ~ dnorm(L_Exp[i, j], tau.est)
        } #j
    } #i

}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("N.Tag", "n", "L", "dt")

# Initial values
jags.inits <- function(){ list(L_inf.est=runif(n=1, min=0, max=200),
                              k.est=runif(n=1, min=0, max=2),
                              Var_L.est=runif(n=1, min=0, max=100),
                              Shape.est=runif(n=1, min=0, max=100),
                              Rate.est=runif(n=1, min=0, max=100)
                              )}

model.file <- 'vonBert_GrowthInc.txt'

# Parameters monitored
jags.params <- c("L_inf.est", "k.est", "Var_L.est" ,"Shape.est", "Rate.est"
)

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 20000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

The model requires a considerable number of updates to achieve convergence, which is understandable given the large number of parameters (due to estimating relative age for each individual). The estimates tend to be reliable, which can be attributed to the large sample size and good ranges for length-at-tagging and time-at-large. Varying the simulation settings for time-at-large and relative age are helpful for understanding this approach to growth analysis. It should not matter for this model whether observations are from more individuals or more captures per individual. Increasing the number of observations per individual (Poisson lambda) would be relevant if fitting a model with growth parameters that varied among individuals.

For a given sample size, precision is lower for tagging compared to age:length data. This seems reasonable given that relative age must be estimated in a tagging study, whereas absolute ages are (assumed to be) known in an age:length analysis. Nevertheless, there are some situations where tagging data are preferred to age:length, such as for tropical fishes that do not show annual marks on hard parts.

It is straightforward to carry out a combined analysis using age:length and tagging data. The age:length data provide information on all three growth parameters (L_∞, k, t_0) as well as the variance term. The tagging data provide information on all parameters except t_0. A combined analysis might be preferable if multiple methods provide better coverage over the entire age/size range. For example, **?** demonstrated the benefits of a combined analysis using age:length, tagging, and length frequency data.

8.2 Growth in weight

The length-weight relationship for fish is typically nonlinear: $W = a * L^b$. The ‘a’ parameter is a scaling coefficient and varies depending on the units chosen for measuring length and weight (e.g., cm vs mm or kg vs g). The ‘b’ parameter is close to 3 for most fishes, because weight is a three-dimensional value, increasing approximately as a cube of length.

The length-weight relationship is useful for several reasons. Length can be measured more quickly and accurately than weight, especially on the water. Fortunately, there tends to be a very strong relationship between length and weight, so it is often advantageous to measure length and predict weight. In addition, the length-weight relationship provides information about fish condition (shape). Fish in poor condition may be overcrowded (e.g., in an pond or aquaculture setting) or lacking in suitable forage.

We begin with simulation code, using an arbitrary range of 20 to 80 for length (e.g., measured in cm). For our example, the scaling parameter ‘a’ will be around 1E-6, so we instead work with the ln-scale value (-12). The model can be fitted using either ‘a’ or the ln-scale transformed parameter. Any assumed value around 3 is fine for the ‘b’ parameter. A multiplicative error term is assumed, as variability in weight typically increases as a function of length (**?**).

```
# Fitting length:weight relationship to simulated data
rm(list=ls()) # Clear Environment

# Choose parameter values and other simulation settings
ln_L_W_a <- -12 # e.g., L in cm, weight in kg
L_W_b <- 3.2
MinL <- 20
MaxL <- 80
SystSample <- 3  # Number per length group
Len <- rep(seq(from=MinL, to=MaxL, by=2), each=SystSample)
SampSize <- length(Len)
W_Var <- 0.1 # Ln-scale variance in weight at length
Wgt <- rlnorm(n=SampSize, mean=(ln_L_W_a+log(Len^(L_W_b))),
              sd=sqrt(W_Var))

par(mfcol=c(1,1)) # Ensure default of 1 panel (changes in trace plots)
par(mar=c(5.1, 4.1, 4.1, 2.1)) # Ensure default margins
plot(Len, Wgt, xlab="Length (cm)", ylab="Weight (kg)")
```

The simulation uses a systematic sample for length, using the seq() function nested within rep(). Use Help for the rep() function and try each function separately in the Console to understand how the two functions work together to generate the length vector. The plot par() function is used to undo formatting changes in the trace plots generated below. As the plot illustrates, variability around the underlying length-weight relationship increases as a function of length.

The JAGS code is relatively short and fitting appears straightforward, although appearances can be deceiving! This simple model is probably the most difficult one to fit in this book, because the two parameters are highly correlated. The high correlation means that offsetting changes in ‘a’ and ‘b’ can provide essentially the same fit, so the updating process is very slow. This model provides a good opportunity to learn about the updating process and using Rhat and trace plots to judge convergence.

The JAGS code for fitting the model begins with broad uniform ranges for the prior distributions. Note that we use same lognormal distribution for the likelihood as in the simulation code (dlnorm() in JAGS). The lognormal distribution assumes a constant, additive error term which is a multiplicative error in the back-transformed arithmetic scale. It is convenient to work with the ln-scale transformed value ($log(a) + log(L^b)$) but the model can also be fitted working directly with ‘a’ parameter ($log(aL^b)$).

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code
sink("LenWgt.txt")
cat("
model{

# Priors
 ln_L_W_a.est ~ dunif(-20, 20)
 L_W_b.est ~ dunif(1, 4)
 W_Var.est ~ dunif(0,0.5)   # Multiplicative error. Variability around ln-scale line

# Calculated value
 precision <- 1/W_Var.est

# Likelihood
 for (i in 1:SampSize) {
    lnWgt_hat[i] <- ln_L_W_a.est+log(Len[i]^L_W_b.est)
    Wgt[i] ~ dlnorm(lnWgt_hat[i], precision)
 } #i
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("SampSize", "Wgt", "Len")

# Initial values

jags.inits <- function(){ list(ln_L_W_a.est=runif(n=1, min=-20, 20),
                               L_W_b.est=runif(n=1, min=1, max=4),
                               W_Var.est=runif(n=1, min=0, max=0.5))}

model.file <- 'LenWgt.txt'

# Parameters monitored
jags.params <- c("ln_L_W_a.est", "L_W_b.est", "W_Var.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 5000,
                model.file)
```

```
print(jagsfit)
plot(jagsfit)

par(mar=c(5.1, 4.1, 4.1, 2.1)) # Set default margins
plot(jagsfit$BUGSoutput$sims.list$ln_L_W_a.est,
     jagsfit$BUGSoutput$sims.list$L_W_b.est)
cor(jagsfit$BUGSoutput$sims.list$ln_L_W_a.est,
    jagsfit$BUGSoutput$sims.list$L_W_b.est)

jagsfit.mcmc <- as.mcmc(jagsfit) # Creates an MCMC object for plotting
par(mar=c(1,1,1,1)) # Avoid error for "figure margins too large"
plot(jagsfit.mcmc) # Trace and density plots
autocorr.diag(jagsfit.mcmc)
```

With 5,000 MCMC iterations, the estimates are generally close to the true values but Rhat is inflated. Plotting the sequence of MCMC estimates for ln-scale 'a' versus 'b' makes clear the extremely high correlation between the two parameters. A numerical estimate of the correlation (~0.99) is obtained in the Console window by using the cor() function.

The trace plots show clearly that many additional updates are needed for the length-weight parameters. The high autocorrelations for 'a' and 'b' cause the trace plots to look more like sea serpents than grassy lawns. Compare this result to a longer run using 50,000 updates. The trace plots for 'a' and 'b' now have a much better appearance and we have good low estimates for Rhat. The autocorrelations can be reduced by thinning (e.g., try n.thin=10). This improves the appearance of trace plot, but the reduced sample size slightly reduces precision and is not generally recommended (**?**).

It can be helpful to add code for displaying the length-weight observations and the fitted curve. The as.numeric() function converts each array (of length 1) to a scalar for use in generating the fitted curve, which is added to the plot using the points function.

```
par(mar=c(5.1, 4.1, 4.1, 2.1)) # Restore default margins
plot(Len, Wgt, xlab="Length (cm)", ylab="Weight (kg)")
par_a <- exp(as.numeric(jagsfit$BUGSoutput$mean$ln_L_W_a.est))
par_b <- as.numeric(jagsfit$BUGSoutput$mean$L_W_b.est)
Wgt_hat <- par_a*(Len^par_b)
points(Len, Wgt_hat, type="l", col="red")
```

8.3 Exercises

1. Before carrying out simulation runs, describe the expected outcome in fitting the von Bertalanffy growth curve to age:length data if you changed the growth rate (k) to 0.05. Report the results for asymptotic size estimates from several runs. What other simulation setting would need to change in order to accommodate such a slow growth rate?
2. Modify the code from Section 8.1.1 to add the fitted curve to the age:length scatter plot. As a hint, you can either redo the scatter plot after the plot(jagsfit) line or add a # to turn off the jagsfit plot.
3. Before carrying out simulation runs, describe the expected outcome in fitting the von Bertalanffy growth curve to age:length data if you changed the maximum age to 4 (e.g., if the stock was heavily exploited). Report the results for asymptotic size estimates from several runs.
4. Modify the code for the von Bertalanffy age:length analysis to obtain the Age vector using the rgamma function that provided relative ages for the tagging analysis (shape=5, rate=2). How does that change affect estimates of asymptotic size? Find a pair of values for shape and rate that result in reliable estimates of the growth parameters.

9

Abundance trends

In Chapter 5, we examined three approaches for estimating absolute abundance. Those approaches are well suited for small populations that can be readily sampled. For larger populations, it is much more common to estimate relative abundance. The general approach for monitoring relative abundance is to conduct a survey using consistent methods over time, using a gear such as electrofishing, traps, or nets. It is assumed that trends in survey catch reflect trends in abundance. Said another way, we are assuming that catches (on average) are directly proportional to abundance. One practical issue to consider is whether gear saturation can occur. This refers to a decline in survey effectiveness as catches increase. For example, gill-net efficiency might decline as the net fills with fish because the gear is increasingly visible. A well-designed survey of relative abundance over time can be quite useful, whether trying to restore a fish at low abundance or determining whether management of an exploited population is effective.

There are several important aspects to planning a survey for monitoring relative abundance. Presumably the focus is on annual trends, so one decision would be how many years would be required before a trend could be detected. The survey catch rate for each year would need to be sufficiently precise to allow for trend detection. Related to these logistical considerations is the decision of what magnitude of change we seek to detect. More gradual changes require much higher precision than if we only seek to detect large increases or decreases. We begin with the simplest case of whether there is a linear trend in relative abundance. This is a Bayesian version of the simple linear regression analysis of traditional statistics courses. Then we consider a more advanced model intended to capture the underlying population dynamics.

9.1 Linear model

We begin with the simulation code for generating absolute abundance, which is unknown, and relative abundance as estimated by the survey. Variables that describe the data set and survey process include the length of the time series (n.years), mean population growth rate (mean.lambda), and the standard deviation for the annual growth rate (sigma.lambda). We use a 10-year survey data set, which seems relatively short for a trend analysis but would also be a daunting task if starting up a new survey. The starting absolute population size (10,000) is arbitrary and not estimable, but we can examine trends in relative abundance from the annual catches.

Following **?** (their Section 5.2), population change is modeled assuming exponential growth. A value just above 1 (1.02) for the mean population growth rate provides for a modest rate of population increase, on average. The standard deviation introduces stochasticity in the annual changes, as could occur due to variation in fishing effort or environmental factors. The expected survey catch is the product of population size and the catchability coefficient, or fraction of the population caught by our annual "unit" of survey effort. Catchability is calculated to provide an arbitrary annual catch of about 200 individuals. Observation error for the survey catches allows for normally-distributed random variation about the expected catch.

```
rm(list=ls()) # Clear Environment

# Generation of simulated data
n.years <- 10
N1 <- 10000  # Initial population size
mean.lambda <- 1.02 # Mean annual population growth rate
sigma.lambda <- 0.02 # Process (temporal) variation in the growth rate (SD)
p.survey <- 200/N1 # Catchability coefficient for survey
sigma.obs <- 40 # Observation error for survey catch (SD)
C <- N <- numeric(n.years)
N[1] <- N1
lambda <- rnorm(n.years-1, mean.lambda, sigma.lambda) # Draw vector of annual growth rates
for (y in 1:(n.years-1)){
  N[y+1] <- N[y] * lambda[y]
}
C <- rnorm(n=n.years, mean=p.survey*N, sd=sigma.obs)
plot(C, ylab="Survey Catch", xlab="Year")
```

Run the simulation code several times to gain insight into the level of variation possible in a ten-year pattern. Then vary survey length, the mean growth rate, temporal variation, and observation error to understand how each setting contributes to the survey pattern of relative abundance.

The JAGS code treats the annual survey catches as independent observations, and estimates an intercept (b_1) and slope (b_2) for the linear regression model relating year to the relative abundance data. A third parameter (sigma.est) accounts for variation in the survey catch around the line, which would be due not only to observation error but also annual variation in population growth. Uninformative prior distributions are used for the three parameters. The initial value for the slope is set at 0, and the initial value for the intercept varies between the minimum and maximum observed annual catch.

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code for estimating model parameters
sink("PopTrend_Linear.txt")
cat("
model {

# Priors
for (i in 1:2){
  b[i] ~ dnorm(0, 0.0001)
}
sigma.est ~ dunif(0, 100)
tau <- pow(sigma.est, -2)

  # Likelihood
  for (y in 1:n.years){
  ExpC[y] <- b[1] + b[2]*y
  C[y] ~ dnorm(ExpC[y], tau)
  } #y
  p.pos <- step(b[2])
}
    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("n.years", "C")

# Initial values.
jags.inits <- function(){ list(b=c(runif(n=1, min=min(C), max=max(C)), 0),
                               sigma.est=runif(n=1, min=0, max=100)
                               )}

model.file <- 'PopTrend_Linear.txt'

# Parameters monitored
jags.params <- c("b", "sigma.est", "p.pos")
```

```
# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 5000,
                model.file)
print(jagsfit)
#plot(jagsfit)

year.seq <- seq(1:n.years)
C.hat <- jagsfit$BUGSoutput$mean$b[1]+jagsfit$BUGSoutput$mean$b[2]*year.seq
points(year.seq, C.hat, type="l", col="red")
```

For convenience in viewing the results, we skip plotting the jagsfit object and simply add the regression line to the original plot of survey catches. We can examine the credible interval for b_2 to determine whether the survey provides evidence of a trend (credible interval not including 0). A more quantitative approach is to estimate the probability of a positive mean growth rate (**?**). We can use the step() function (which is 1 for an argument > 0; 0 otherwise) to calculate the fraction of retained slope estimates that are greater than 0 (mean for calculated variable p.pos).

Try running the simulation and analysis code multiple times and see whether your visual interpretation of the pattern (or lack thereof) matches with the regression results. Then vary the simulation settings to see the level of support for a positive trend. For example, the probability of a positive mean growth rate was 0.98, 0.97, 0.84, 0.71, and 0.51 for five trials using the default settings, compared to 0.99, 0.99, 0.69, 1.00, 0.71 if observation error was reduced by half.

9.2 Exponential growth model

An exponential growth model was used in Section 9.1 to simulate population change, but our analysis ignored the dependence between the annual survey catches. An improved approach would be to fit a model that attempts to reveal the underlying population dynamics. The simulation code is unchanged, but the JAGS code now specifies exponential growth. We model the catches as if they are population levels, because of the assumed direct proportionality between catch and absolute abundance. Following **?**, we fit the exponential growth model on the log scale.

Fitting an underlying population model makes it possible to distinguish between the two sources of variation in survey catch: variation in population growth rate and survey sampling. The former is referred to as process error (**?**), as it is variation in the biological process (population growth). The latter is observation error, or random variation in the survey catch (our proxy for annual abundance).

The JAGS code now has parameters for the starting "population size" (i.e., predicted ln-scale catch in year 1), the mean and standard deviation for the annual sequence of population growth rates, and the standard deviation for observation error. The parameters for growth rate are hyperparameters, describing a normal distribution for the annual growth rates. Population growth in ln-scale uses the model $lnN_{t+1} = lnN_t + r$, so 1+r provides a value comparable to mean.lambda. Uninformative prior distributions are used for all parameters.

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code for estimating model parameters
sink("PopTrend_ExpGr.txt")
cat("
model {
# Priors
logC.est[1] ~ dunif(1, 100)     # Prior for initial relative abundance
mean.r.est ~ dnorm(0, 0.001)    # Prior for mean growth rate
sigma.lambda.est ~ dunif(0, 1)  # Prior for process error (annual variation in r)
tau.lambda.est <- pow(sigma.lambda.est, -2)
sigma.obs.est ~ dunif(0, 100)     # Prior for observation error
tau.obs.est <- pow(sigma.obs.est, -2)

# Likelihood
for (y in 1:(n.years-1)){
   r[y] ~ dnorm(mean.r.est, tau.lambda.est)
   logC.est[y+1] <- logC.est[y] + r[y]
   } #y
for (y in 1:n.years) {
  C.est[y] <- exp(logC.est[y])
  C[y] ~ dnorm(C.est[y], tau.obs.est)
   } #y
     p.pos <- step(mean.r.est)
}
    ",fill = TRUE)
sink()
```

```
# Bundle data
jags.data <- list("n.years", "C")

# Initial values.
jags.inits <- function(){list(sigma.lambda.est = runif(n=1, min=0, max=1),
                              mean.r.est = rnorm(n=1, mean=0, sd=1),
                              sigma.obs.est = runif(n=1, min=0, max=100),
                              logC.est = c(runif(n=1, min=5, max=10), rep(NA, (n.years-1))))}

model.file <- 'PopTrend_ExpGr.txt'

# Parameters monitored
jags.params <- c("mean.r.est", "sigma.lambda.est", "sigma.obs.est", "C.est",
                 "p.pos")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 20000,
                model.file)
print(jagsfit)
#plot(jagsfit)

year.seq <- seq(1:n.years)
points(year.seq, jagsfit$BUGSoutput$mean$C.est, type="l", col="red")
```

This is a much more complex model than the linear regression, and we increase the number of iterations to achieve convergence (which should be confirmed by checking Rhat values). The predicted catches are now not a straight line because we are estimating annual values for population growth rate. The smoothed pattern for predicted catch is the model's compromise between the biological process (exponential growth with year-to-year variation in growth rate) and observation error (observed catches deviating from the predicted trajectory). Try several runs using a very small value for observation error to see that the model can closely mimic the population trajectory when observation error is negligible.

We use the step() function (calculated parameter p.pos) to estimate the probability that the mean population growth rate is positive. We obtained probabilities of 0.73, 0.73, 0.40, 0.44, and 0.61 for five trials using the default settings, compared to 0.74, 0.69, 0.92, 0.99, and 0.91 for a simulation mean growth rate of 1.05. These results show the challenge in detecting a population trend, which will depend on the magnitude of the mean annual change and the precision of the survey. Similar results can be obtained by varying the length of the time series or annual variability in population growth rate.

9.2.1 Forecasting

In Section 9.2, we fitted an exponential growth model to describe underlying population dynamics. One advantage of that approach is that we can use that underlying model to project future population levels. We add a new simulation setting (p.years) for the number of future years to predict abundance, and generate n.years+p.years values for population size (N). The catch vector has n.years of observed survey catches, augmented by p.years of NA values. The code for plotting catch is moved below the JAGS section, because now we need to choose a range for the y-axis that will accommodate the observed and predicted catches plus the (generally wider) credible intervals.

```r
rm(list=ls()) # Clear Environment

# Generation of simulated data
p.years <- 5 # Predicted future population levels
n.years <- 10 # Years of survey data
N1 <- 10000  # Initial population size
mean.lambda <- 1.02 # Mean annual population growth rate
sigma.lambda <- 0.02 # Process (temporal) variation in the growth rate (SD)
p.survey <- 200/N1 # Catchability coefficient for survey
sigma.obs <- 40 # Observation error for survey catch (SD)
C <- N <- numeric(n.years+p.years)
N[1] <- N1
lambda <- rnorm((n.years+p.years-1), mean.lambda, sigma.lambda) # Draw vector of annual growth rat
for (y in 1:(n.years+p.years-1)){
  N[y+1] <- N[y] * lambda[y]
}
C <- rnorm(n=n.years, mean=p.survey*N, sd=sigma.obs)
C <- c(C, rep(NA, p.years))  # Observed survey catches augmented by NA values
```

The likelihood in the JAGS code is modified slightly to accommodate the p.years of additional (missing) catches. We also add a step() function to estimate p.increase, the probability that predicted catch in the final year is higher than in year one. This is an arbitrary choice, but provides some insight into whether a detectable increase is expected over the modeled time horizon. Data passed to JAGS now includes p.years, and we modify the length of the vector of initial values for predicted catch.

```r
# Load necessary library
library(rjags)
library(R2jags)
```

```
# JAGS code for estimating model parameters
sink("PopTrend_ExpGr.txt")
cat("
model {
# Priors
logC.est[1] ~ dunif(1, 100)     # Prior for initial relative abundance
mean.r.est ~ dnorm(0, 0.001)    # Prior for mean growth rate
sigma.lambda.est ~ dunif(0, 1)  # Prior for annual variation in r
tau.lambda.est <- pow(sigma.lambda.est, -2)
sigma.obs.est ~ dunif(0, 100)     # Prior for observation error
tau.obs.est <- pow(sigma.obs.est, -2)

# Likelihood
for (y in 1:(n.years+p.years-1)){
   r[y] ~ dnorm(mean.r.est, tau.lambda.est)
   logC.est[y+1] <- logC.est[y] + r[y]
   } #y
for (y in 1:(n.years+p.years)) {
  C.est[y] <- exp(logC.est[y])
  C[y] ~ dnorm(C.est[y], tau.obs.est)
   } #y
     p.pos <- step(mean.r.est)
p.increase <- step(C.est[n.years+p.years]-C.est[1]) # Prob of increase from first to last
}
    ",fill = TRUE)
sink()

# Bundle data
jags.data <- list("n.years", "p.years", "C")

# Initial values.
jags.inits <- function(){list(sigma.lambda.est = runif(n=1, min=0, max=1),
                              mean.r.est = rnorm(n=1, mean=0, sd=1),
                              sigma.obs.est = runif(n=1, min=0, max=100),
                              logC.est = c(runif(n=1, min=5, max=10),
                                           rep(NA, (n.years+p.years-1))))}

model.file <- 'PopTrend_ExpGr.txt'

# Parameters monitored
jags.params <- c("mean.r.est", "sigma.lambda.est", "sigma.obs.est", "C.est",
                 "p.pos", "p.increase")
```

```r
# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 40000,
                model.file)
print(jagsfit)
plot(jagsfit)
```

Convergence was slower for this model compared to Section 9.2, so the number of iterations was increased to 40,000. Plotting the jagsfit object shows that uncertainty about predicted catch increases sharply over the forecast years. The future forecasts use annual population growth rates that are drawn randomly from the normal distribution. This makes clear how little can be said about population size or trends, even a couple of years into the future.

We can now plot the observed catches, augmented by the predicted catch vector and lines representing lower and upper 95% credible intervals (**?**). The quantile() function provides the 0.025 and 0.975 quantiles from the vector of retained MCMC updates (draws from the posterior distribution). The ylim plot option allows us to scale the y-axis to accommodate all four series.

```r
par(mfcol=c(1,1)) # Return to default of 1 panel
year.seq <- seq(1:length(C))
lower <- upper <- length(C)
for (y in 1:length(C)){
  lower[y] <- quantile(jagsfit$BUGSoutput$sims.list$C.est[,y], 0.025)
  upper[y] <- quantile(jagsfit$BUGSoutput$sims.list$C.est[,y], 0.975)
} #y
min.y <- min(C[1:n.years],jagsfit$BUGSoutput$mean$C.est, lower)
max.y <- max(C[1:n.years],jagsfit$BUGSoutput$mean$C.est, upper)
plot(year.seq, C, ylab="Survey Catch", xlab="Year",
     ylim = c(min.y, max.y))

points(year.seq, jagsfit$BUGSoutput$mean$C.est, type="l", col="red")
points(year.seq, lower, type="l", lty="dashed")
points(year.seq, upper, type="l", lty="dashed")
```

The observed catches should generally fall within the 95% credible intervals. As in Section 9.2, the red line for predicted catch has a smoother trajectory than the individual catches. How many years into the future would you be confident in predicting the population trend?

9.3 Exercises

1. For the linear model (Section 9.1), compare the probability of a positive mean growth rate (p.pos) for sigma.lambda=0.02 (default) and 0.01, using ten simulation trials at each level.
2. For the exponential growth model (Section 9.2), use simulation to determine the approximate level for observation error where p.pos would typically (e.g., 8 of 10) exceed 90%.
3. For the exponential growth model that includes forecasting (Section 9.2.1), compare a sequence of estimates for p.increase (probability that predicted catch in final year exceeds year one) for mean population growth rates of 1.02 and 1.05.

10

Recruitment

Growth, mortality, and recruitment are the three fundamental processes that regulate fish populations. Previous chapters have addressed the first two; here we consider methods for studying recruitment. Recruitment is usually considered to be the annual process of young fish entering the part of the population that is vulnerable to fishing (i.e., the fishable stock). Thus the timing of recruitment depends on fishery selectivity pattern(s), which tend to be gradual rather than knife-edged. One practical approach for defining age at recruitment is to choose the first age at which fishery catches are non-negligible. For simplicity in this chapter, we will simulate recruitment occurring at age one so there is a one-year lag from the time of spawning to recruitment.

What happens between spawning and recruitment is of interest to the fish ecologist and the fishery manager. Typical factors affecting recruitment include environmental conditions, prey availability, and predation. It may seem obvious to assume that spawning stock size affects recruitment, but it is not always easy to detect such a relationship (**?**). The relationship may not be evident if there is little contrast in spawning stock size, high variability in recruitment, too short a time series, or low precision for estimates of spawning stock and recruitment. Characterizing the strength of the relationship can provide valuable guidance to the fishery manager. For example, management will need to be more restrictive in cases where recruitment drops consistently as spawning stock declines. In other cases, a more aggressive harvest policy may be possible because recruitment appears to be relatively stable over a wide range of spawning stock sizes. The risk in any case is of fishing the stock down to a point where recruitment declines sharply. This is termed recruitment overfishing, and it can be very difficult to rebuild a population after that occurs.

The process of recruitment starts with eggs, but egg production is difficult to estimate directly so it is common to use spawning stock biomass (B_S) as a proxy. One of the most commonly used curves relating spawning stock size and recruitment is the Beverton-Holt equation: $R = 1/(\alpha + \beta/B_S)$. This curve approaches zero as B_S approaches zero and approaches $1/\alpha$ as B_S approaches infinity. It is often a good descriptor of observed stock-recruitment data in that recruitment is low when spawning stock size is close to zero but appears to be without trend at higher spawning stock sizes.

The best way to explore the behavior of the Beverton-Holt curve is to try different values for the two parameters:

```
SpBio <- seq(from=10, to=200, by=5)
a <- 0.001
b <- 0.1
R <- 1/(a+b/SpBio)
plot(SpBio, R)
```

In this example code, spawning stock biomass is generated with the seq() (sequence) function over a wide but arbitrary range. The Beverton-Holt parameters are also arbitrary but are chosen to allow for gradually increasing recruitment toward an asymptote of 1000. Experiment with different parameter values to find a set that would maintain an asymptote of 1000 but would produce a steeper curve, with about 90% of asymptotic recruitment at a spawning stock level of 100.

10.1 Fitting a stock-recruitment curve

We begin by fitting a Beverton-Holt model to simulated stock-recruitment data. Section 10.2 describes an approach for comparing full and reduced models, to explore the conditions under which an underlying relationship can be detected.

Our simulation uses a full age-structured model to generate the stock-recruitment data. First, we choose the structure of the population matrix (number of ages and years). Next, we choose arbitrary annual fishing mortality rates. By starting the population at a high level and fishing intensively, we ensure that there is sufficient contrast (**?**) in spawning stock size to be able to detect the underlying relationship. Age-specific fishing mortality rates are obtained by defining the fishery selectivity pattern. We use a logistic curve and arbitrarily choose parameters so that 50% selectivity occurs at age two. (It is useful to plot the selectivity curve (code commented out) while varying the two selectivity parameters.) Obtaining year- and age- specific fishing mortality rates as a product of a year-specific measure of fishing intensity and an age-specific vulnerability to fishing is sometimes referred to as a “separable” model.

```r
rm(list=ls()) # Clear Environment

A <- 10 # Arbitrary. If changed, redo maturity and meanwt vectors
Y <- 12 # Arbitrary. If changed, redo AnnualF vector

AnnualF <- c(0.59, 0.41, 0.74, 0.91, 0.86, 0.74, 1.07, 0.9, 0.87, 1.1, 0.93)
# Arbitrary, increasing trend to create contrast in SpBio. Redo if Y changes
AnnualM <- 0.2

# F by age, using separable model
Fishery_k <- 1 # Slope for logistic function
Fishery_a_50 <- 2 # Age at 0.5 selectivity
SelF <- array(data=NA, dim=A) # Fishery selectivity pattern
F_a <- array(data=NA, dim=c(A,(Y-1))) # Matrix for F by age and year
Z_a <- array(data=NA, dim=c(A,(Y-1))) # Matrix for Z by age and year
S_a <- array(data=NA, dim=c(A,(Y-1))) # Matrix for survival rate by age and year
for (a in 1:A){
  SelF[a] <- 1/(1+exp(-Fishery_k*(a-Fishery_a_50)))
  } #a
#plot(SelF)
for (y in 1:(Y-1)) {
  for (a in 1:A){
    F_a[a,y] <- AnnualF[y] * SelF[a]
    Z_a[a,y] <- F_a[a,y]+AnnualM
    S_a[a,y] <- exp(-Z_a[a,y])
  } #a
} #y

N <- array(data=NA, dim=c(A, Y))

BH_alpha <- 1/1E5 # Parameter defining Beverton-Holt asymptotic recruitment
BH_beta <- 0.1 # Arbitrary
SD_rec <- 0.25 # Low level of lognormal error in recruitment

# Set up year-1 vector based on asymptotic recruitment and year-1 survival rates
Mat <- c(0, 0, 0.2, 0.4, 0.8, 1, 1, 1, 1, 1) # Arbitrary maturity schedule. Redo if max age (A) ch
Wgt <- c(0.01, 0.05, 0.10, 0.20, 0.40, 0.62, 0.80, 1.01, 1.30, 1.56) # Mean wt. Redo if A changes
SpBio <- array(data=NA, dim=(Y-1)) # Spawning biomass vector
# Year-1 N (arbitrarily) from asymptotic recruitment, year-1 survival.
N[1,1] <- 1/BH_alpha*rlnorm(n=1, 0, SD_rec)
for (a in 2:A){
  N[a,1] <- rbinom(n=1, trunc(N[(a-1),1]), S_a[(a-1),1])
}#a
for (y in 2:Y){
  SpBio[y-1] <- sum(N[,y-1]*Wgt*Mat) # Spawning stock biomass
  N[1,y] <- 1/(BH_alpha+BH_beta/SpBio[y-1])*rlnorm(n=1, 0, SD_rec)
  for (a in 2:A){
    N[a,y] <- rbinom(1, trunc(N[(a-1), (y-1)]), S_a[(a-1),(y-1)])
    } #a
  } #y
#plot(SpBio,N[1,2:Y], ylab="Recruits", xlab="Spawning stock biomass")

# Generate survey data on recruitment and spawning stock
Survey_q <- 1E-3 # Arbitrary catchability coefficient for survey
SD_survey <- 0.2 # Arbitrary ln-scale SD
Exp_ln_B <- log(Survey_q*SpBio)
Survey_B <- rlnorm(n=Y, meanlog=Exp_ln_B, sdlog=SD_survey)
#plot(SpBio, Survey_B[1:Y-1])
Exp_ln_R <- log(Survey_q*N[1,2:Y])
Survey_R <- c(NA,rlnorm(n=(Y-1), meanlog=Exp_ln_R, sdlog=SD_survey))
#plot(N[1,], Survey_R)
```

The next section of code generates the population matrix. The Beverton-Holt parameters are arbitrarily chosen and can be varied to explore different shapes for the curve. Next, we generate the population vector for year one, using approximate equilibrium values. Expected recruitment is set equal to the curve's asymptote, with lognormal error term used to add random variation; for example, due to environmental conditions. The lognormal distribution is well suited to recruitment data because of the potential for occasional extreme values (**?**). The chosen standard deviation (0.25) is on the lower end of variation in recruitment (**?**). The rest of the year-1 vector is obtained from age-1 recruitment using year-1 mortality rates ($N_{a,1} = S_{a-1,1} * N_{a-1,1}$). A binomial function introduces stochastic variation in survival. Population size at the previous age is truncated because the binomial function requires a whole number for the size of the experiment.

The population vector for the remaining years can be filled in once the year-1 vector is determined. For example, recruitment in year 2 depends on spawning stock biomass in year 1. The age-specific vectors for maturity schedule and weight-at-age are arbitrary. Numbers of fish age 2 and older are obtained as survivors from the prior year's vector; i.e., $N_{a,y} = S_{a-1,y-1} * N_{a-1,y-1}$, with random variation in survival from a binomial distribution. The final step is to obtain survey observations for recruitment and spawning stock biomass. The random survey values are drawn from a lognormal distribution, using an arbitrary catchability coefficient and ln-scale standard deviation for error. The survey values are referred to as indices but could be estimates of the absolute magnitude of recruitment and spawning stock size. There is not a recruitment survey value for year 1 because the simulated recruitment was not based on spawning stock size.

The JAGS code uses uninformative prior distributions for the two Beverton-Holt parameters and the lognormal error term. Expected recruitment is predicted on ln-scale in order to match the assumed lognormal error. The recruitment index is predicted (R_hat) for use in plotting (see below).

```r
# Load necessary library
library(rjags)
library(R2jags)

sink("StockRecruit.txt")
cat("
model {
# Model parameters: Beverton-Holt alpha and beta, SD for fitted curve

# Priors
 BH_alpha.est ~ dunif(0, 10)
 BH_beta.est ~ dunif(0, 10)
 SD_rec.est ~ dunif(0, 10)
 tau <- pow(SD_rec.est, -2)

# Likelihood
    for(y in 2:Y) {
      lnR_hat[y] <- log(1/(BH_alpha.est+BH_beta.est/Survey_B[y-1]))
      Survey_R[y] ~ dlnorm(lnR_hat[y],tau)
      R_hat[y] <- exp(lnR_hat[y])
      } #y
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("Y", "Survey_R", "Survey_B")

# Initial values
y <- Survey_R[2:Y]
x <- Survey_B[1:(Y-1)]
nls_init <- nls(y ~ 1/(BH_a_start+BH_b_start/x),
                start = list(BH_a_start = 0.001, BH_b_start = 0.001),
                algorithm="port",
                lower=c(1E-6, 0), upper=c(1000,1000))
#summary(nls_init)
```

```r
nls_save <- coef(nls_init)
#y_hat <- 1/(nls_save[1]+nls_save[2]/x)
#plot(x,y)
#points(x,y_hat, col="red", type="l")

jags.inits <- function(){ list(BH_alpha.est=nls_save[1],
                               BH_beta.est=nls_save[2],
                               SD_rec.est=runif(n=1, min=0, max=0.75))}

model.file <- 'StockRecruit.txt'

# Parameters monitored
jags.params <- c("BH_alpha.est", "BH_beta.est", "SD_rec.est",
                 "R_hat")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
plot(jagsfit)

BH.DIC <- jagsfit$BUGSoutput$DIC # Used in next section

min.y <- min(Survey_R[2:Y],jagsfit$BUGSoutput$mean$R_hat)
max.y <- max(Survey_R[2:Y],jagsfit$BUGSoutput$mean$R_hat)

df.xy <- data.frame(Survey_B[1:(Y-1)], Survey_R[2:Y],jagsfit$BUGSoutput$mean$R_hat)
df.xy <- df.xy[order(df.xy[,1]),]  # Sort by survey index for plotting

plot(df.xy[,1],df.xy[,2], ylab="Recruitment index",
     xlab="Spawning Stock Index", ylim=c(min.y, max.y))
points(df.xy[,1], df.xy[,3], type="l", col="red")
```

Randomly chosen starting values for the two Beverton-Holt parameters sometimes caused JAGS to crash, so nls(), the R function for nonlinear least-squares, was used to generate good initial values. When unconstrained, nls() occasionally produced a negative estimate of the first parameter (1/asymptotic recruitment), so the "port" algorithm was used to allow bounds to be specified. The nls() code illustrates the compactness of using R functions, compared to a Bayesian analysis, but also the "black box" nature of the analysis.

The JAGS results tend to be reliable for the default settings. The plotted estimates of the recruitment indices capture the decrease due to declining spawning stock size. Estimates of the Beverton-Holt alpha parameter are typically close to 0.01 (BH_alpha/Survey_q), because we fit the curve to a recruitment index (true recruitment * Survey_q) rather than to absolute levels. Estimates of the Beverton-Holt beta parameter tend to be close to the true value. Estimates of the standard deviation for variation about the curve tend to be higher than the true value of 0.25, because the points vary due not only to true recruitment variation but also variation due to survey sampling.

As always, it is very helpful to view the points and fitted curve. We use a data frame to sort the spawning stock, recruitment, and predicted recruitment indicies (link[1]), in order to have a smooth fitted curve (i.e., continuously increasing spawning stock values). Try multiple runs to see how variation in recruitment affects the pattern of points and fitted curve.

10.2 Does spawning stock size affect recruitment?

Section 10.1 illustrated the process of model fitting but did not answer the question of whether there is a detectable relationship between the spawning stock and recruitment indices. To address that question, we return to the approach shown in Section 4.3.6; i.e., comparing DIC scores for candidate models. Here the two candidates are the full Beverton-Holt model and a reduced model with constant recruitment.

We append JAGS code for the constant recruitment case at the end of the previous code so that both models are applied to the same spawning stock and recruitment data. Convergence is extremely rapid for this simpler case. The difference in DIC scores between the full (Beverton-Holt) and reduced (constant recruitment) models is printed to the Console, but it is also useful to compare pD, deviance, and DIC scores.

```
# Constant recruitment fit
# JAGS code ##################################

sink("StockRecruit.txt")
cat("
model {
# Model parameters: Beverton-Holt alpha, SD for fitted curve
```

[1]https://stackoverflow.com/questions/2307925/most-efficient-way-to-sort-two-vectors-in-lockstep-in-r

```
# Priors
 BH_alpha.est ~ dunif(0, 10)
 SD_rec.est ~ dunif(0, 10)
 tau <- pow(SD_rec.est, -2)

# Likelihood
    for(y in 2:Y) {
      lnR_hat[y] <- log(1/BH_alpha.est)
      Survey_R[y] ~ dlnorm(lnR_hat[y],tau)
      R_hat[y] <- exp(lnR_hat[y])
      } #y
}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("Y", "Survey_R")

jags.inits <- function(){ list(BH_alpha.est=nls_save[1],
                               SD_rec.est=runif(n=1, min=0, max=0.75))}

model.file <- 'StockRecruit.txt'

# Parameters monitored
jags.params <- c("BH_alpha.est", "SD_rec.est", "R_hat")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 10000,
                model.file)
print(jagsfit)
plot(jagsfit)
ConstR.DIC <- jagsfit$BUGSoutput$DIC
ConstR.DIC - BH.DIC
```

With the default settings, the DIC score is typically higher (indicating a poorer fit) for the constant recruitment model. Next, try varying some of the simulation settings. The pattern for annual fishing mortality could be altered to produce less of a decline over time in spawning stock (less contrast). Higher levels of recruitment variation are not uncommon [e.g., 0.5-0.75; **?**], and will make the relationship harder to detect. For a lognormal variance of 0.5 or 0.75, the DIC score may actually be lower for the constant recruitment model because of a smaller penalty for model complexity (smaller estimate of pD). Other potential settings to vary include the length of the time series and the precision of the survey data. Be sure to look back at the scatter plot and fitted Beverton-Holt curve, to get a sense of how the constant recruitment model would compare.

10.3 Exercises

1. Revise the code for Section 10.1 to fit a Ricker model ($R = \alpha * B_S * exp(-\beta * B_S)$). Experiment with the Ricker parameters to produce maximum recruitment at an intermediate observed spawning stock size.
2. The expected value for α (0.01 for our default simulation settings) is the true underlying value divided by the survey catchability, because we are fitting the model using survey indices rather than estimates of absolute abundance. Compare estimates of α for the Beverton-Holt and constant recruitment models, using the approach from Section 10.2. Explain any pattern that you detect.

11

Integrated Population Models

The field and analytical methods discussed thus far have addressed population rates one at a time (e.g., estimating abundance or survival). An improved approach would be to design a complementary set of field studies that provides a complete picture of population dynamics. For example, **?** used harvest and tag recovery data to fit an integrated population model (IPM) for black bear (*Ursus americanus*). Their model provided estimates that were more precise and biologically realistic than point estimates using mark-recapture data alone. **?** developed an avian IPM using adult population counts, nest surveys, and capture-recapture data on survival. Here we illustrate a somewat simpler approach, using only a series of short-term (closed population) capture-recapture estimates and telemetry information on survival. Even this simpler approach has a higher cost (time and money) than a basic survey to monitor relative abundance, but could be justified for a subset of cases (e.g., a threatened population or one supporting an important fishery).

We begin with the simulation model:

```r
rm(list=ls()) # Clear Environment

# Simulation to create population size and field data

Y <- 10

LowS <- 0.5 # Adult survival, arbitrary lower bound
HighS <- 0.9 # Arbitrary upper bound
AnnualS <- array(data=NA, dim=(Y-1))
AnnualS <- runif(n=(Y-1), min=LowS, max=HighS)

MeanR <- 400 # Mean annual recruits
N.r <- rpois(n=Y, lambda=MeanR)
N.a <- array(data=NA, dim=Y)
N.a[1] <- N.r[1] + rpois(n=1, lambda=MeanR) # Arbitrary year-1 adult population size
for (y in 2:Y){
  N.a[y] <- N.r[y] + rbinom(n=1, size=N.a[y-1], prob=AnnualS[y-1])
  } #y
```

```
# Short-term (closed) two-sample population estimate
CapProb <- 0.1 # Fraction of population sampled
n1 <- rbinom(n=Y, size=N.a, prob=CapProb) # Number caught and marked in first sample
n2 <- rbinom(n=Y, size=N.a, prob=CapProb) # Number caught and examined for marks in second sample
m2 <- rbinom(n=Y, size=n2, prob=n1/N.a) # Number of marked fish in second sample
#n1*n2/m2 # Point estimates

# Telemetry information on annual survival
Telemetry <- array(data=NA, dim=Y)
Telemetry[1] <- 40 # Arbitrary number of telemetry tags to start study
for (y in 2:Y){
  Telemetry[y] <- rbinom(n=1, size=Telemetry[y-1], prob=AnnualS[y-1])
} #y
```

We model absolute population size, and assume that fish age 1 and older share population rates. Survival rate varies annually within user-specified lower and upper bounds. Annual recruitment (surviving age-0 fish from the prior year) is drawn from a Poisson distribution with an arbitrary mean of 400. We require a starting population in year 1 and draw the adult survivors from a Poisson distribution, using the same mean as recruitment. Population size in subsequent years is

$$N.a_y = N.r_y + Binomial(N.a_{y-1}, AnnualS_{y-1})$$

i.e., incoming recruitment plus surviving adults.

Model parameters are estimated using simulated data from two field studies that are assumed to be independent and would provide information on abundance and survival. The first data source is a two-sample (closed) capture-recapture estimate, assumed to be done at the start of each year. The arbitrary capture probability is used to generate the two binomally-distributed samples. The number of recaptures is also binomally distributed, using the observed marked fraction (on average equal to the capture probability). The second data source is a telemetry study, with an arbitrary initial number of fish given transmitters. The telemetry study is assumed to provide annual information on the number of surviving telemetered fish. For example, this could be done through fixed-receiver detections of migratory fish that return annually to a spawning area. For simplicity, the initial batch of transmitters is assumed to last for the duration of the overall population study. Other field designs could be tested with minor modifications in code; e.g., adding new transmitters each year.

Parameter estimates are obtained using uninformative prior distributions for recruitment and annual survival. Preliminary trials indicated that there was not sufficient information to estimate annual recruitment so only the overall mean is estimated. Starting population size in year 1 is Poisson distributed, uaing the capture-recapture point estimate as the expected value. Population size in subsequent years is also drawn from a Poisson distribution, using the population projection equation as the expected value. The likelihood for the capture-recapture data uses the approach from Section 4.3. The telemetry likelihood uses a binomail distribution for the annual observation of remaining live fish with transmitters.

Following the JAGS section is some R code to compare the true values with model estimates. The second plot uses the abline() function to add horizontal lines for the mean and empirical 0.025 and 0.975 bounds. The par(xpd=TRUE) turns off clipping (link[1]) so that a legend can be added outside the plot boundary (using inset to set the relative coordinates of the legend). For the third plot, the Chapman modification of the capture-recapture estimator **?** was used to avoid getting a plotting error due to an infinite point estimate in cases with $m_2 = 0$.

```
# Load necessary library
library(rjags)
library(R2jags)

# JAGS code ##################################
sink("IPM.txt")
cat("
model {

# Priors

 MeanR.est ~ dunif(0, 10000)

for (y in 1:(Y-1)){
  AnnualS.est[y] ~ dunif(0,1)  # Uninformative prior, adult survival
}#y

# Population projection
 N.a.est[1] ~ dpois((n1[1]+1)*(n2[1]+1)/(m2[1]+1)-1) # Year 1 total, Chapman mod in case m2=0
 # Point estimate needed to start model because no prior year information
 for(y in 2:Y) {
     N.a.est[y] ~ dpois(MeanR.est + (AnnualS.est[y-1]*N.a.est[y-1]))
     } #y

#Likelihoods
```

[1]https://stackoverflow.com/questions/3932038/plot-a-legend-outside-of-the-plotting-area-in-base-graphics

```
# Two-sample capture-recapture estimate for total pop size, done at start of year
  for (y in 2:Y){  # m2[1] used above to get starting pop size
    m2[y] ~ dbin((n1[y]/N.a.est[y]), n2[y])
    } #y

# Telemetry information on survival rate
     for (y in 1:(Y-1)){
    Telemetry[y+1] ~ dbin(AnnualS.est[y], Telemetry[y])
    } #y

}
    ",fill=TRUE)
sink()

# Bundle data
jags.data <- list("Y", "n1", "n2", "m2", "Telemetry")

# Initial values
jags.inits <- function(){ list(MeanR.est=runif(n=1, min=0, max=10000),
                               AnnualS.est=runif(n=(Y-1), min=0, max=1))}

model.file <- 'IPM.txt'

# Parameters monitored
jags.params <- c("AnnualS.est", "N.a.est", "MeanR.est")

# Call JAGS from R
jagsfit <- jags(data=jags.data, jags.params, inits=jags.inits,
                n.chains = 3, n.thin=1, n.iter = 20000,
                model.file)
print(jagsfit)
plot(jagsfit)

# Code for plots comparing true values and model estimates
par(mfrow = c(1, 3))
plot(seq(1:(Y-1)), jagsfit$BUGSoutput$mean$AnnualS.est, ylab="Annual S", xlab="Year",
     type="b", pch=19, ylim=c(min(AnnualS,jagsfit$BUGSoutput$mean$AnnualS.est),
                              max(AnnualS,jagsfit$BUGSoutput$mean$AnnualS.est)))
points(AnnualS, col="blue", type="b", pch=17)

MeanR.est.lower <- quantile(jagsfit$BUGSoutput$sims.list$MeanR.est, probs=c(0.025), na.rm=TRUE)
MeanR.est.upper <- quantile(jagsfit$BUGSoutput$sims.list$MeanR.est, probs=c(0.975), na.rm=TRUE)
plot(seq(1:Y), N.r, ylab="Recruits", xlab="Year",
     type="b", pch=19, col="blue", ylim=c(min(N.r,MeanR.est.lower),
                              max(N.r,MeanR.est.upper)))
par(xpd=TRUE) # Turn off clipping to put legend above plot
# https://stackoverflow.com/questions/3932038/plot-a-legend-outside-of-the-plotting-area-in-base-g
legend("top",
       legend = c("Est", "Obs"), col = c("black", "blue"),
       pch = c(19,17), text.col = "black",  horiz = T , inset = c(0, -0.2))
par(xpd=FALSE)
abline(h=MeanR.est.lower, lty=3) # Add line for lower bound estimated mean R
abline(h=jagsfit$BUGSoutput$mean$MeanR.est)
abline(h=MeanR.est.upper, lty=3) # Add line for upper bound estimated mean R

MR_N.hat <- (n1+1)*(n2+1)/(m2+1) -1 # Chapman modification in case m2=0
plot(seq(1:Y), jagsfit$BUGSoutput$mean$N.a.est, ylab="Pop size (includes new recruits)",
     xlab="Year",
     type="b", pch=19, ylim=c(min(N.a,jagsfit$BUGSoutput$mean$N.a.est,MR_N.hat),
                              max(N.a,jagsfit$BUGSoutput$mean$N.a.est,MR_N.hat)))
points(N.a, col="blue", type="b", pch=17) # True adult pop size
points(MR_N.hat, col='red', type="b") # Capture-recapture point estimates
```

Results for the default case show that population size and annual survival tend to be reliably estimated. Recruitment is more poorly estimated, which is not surprising given that there is no direct information on recruitment. The model makes indirect inferences about recruitment from the total year-y abundance (new age-1 recruits plus 2+ survivors) and telemetry information on survival. A field design that provided separate point estimates of age-1 and age-2+ abundance would improve the estimates of recruitment.

After doing multiple runs with the default settings, try varying the capture probability for the capture-recapture study and the initial release of fish with transmitters. A higher capture probability improves the estimates of total population size, and more transmitters improve the annual survival estimates.

11.1 Exercises

1. Compare true and estimated mean recruitment in ten trials using the default capture probability of 0.1 versus 0.2 and 0.3. How well does the model estimate total population size and annual survival for each sampling intensity?
2. How might the field sampling methods be modified or augmented to improve the model estimates?

A

Title Placeholder

If there is anything to put in an appendix, it would go here.

Bibliography

Front Matter. In Royle, J. A. and Dorazio, R. M., editors, *Hierarchical Modeling and Inference in Ecology*, page iii. Academic Press.

Allen, M. S. Effects of Variable Recruitment on Catch-Curve Analysis for Crappie Populations. 17(1):202–205.

Bolker, B. M. *Ecological Models and Data in R.* Princeton University Press, 1 edition.

Brooks, S. P. and Gelman, A. General Methods for Monitoring Convergence of Iterative Simulations. 7(4):434–455.

Brownie, C., Anderson, D., Burnham, K., and Robson, D. Statistical inference from band recovery data: A handbook.

Bryant, M. D. Estimating Fish Populations by Removal Methods with Minnow Traps in Southeast Alaska Streams. 20(4):923–930.

Burnham, K. P. and Anderson, D. R., editors. *Model Selection and Multimodel Inference.* Springer.

Campana, S. E. Accuracy, precision and quality control in age determination, including a review of the use and abuse of age validation methods. 59(2):197–242.

Conn, P. B., Diefenbach, D. R., Laake, J. L., Ternent, M. A., and White, G. C. Bayesian Analysis of Wildlife Age-at-Harvest Data. 64(4):1170–1177.

Conn, P. B., Johnson, D. S., Williams, P. J., Melin, S. R., and Hooten, M. B. A guide to Bayesian model checking for ecologists. 88(4):526–542.

Doll, J. C. and Jacquemin, S. J. Introduction to Bayesian Modeling and Inference for Fisheries Scientists. 43(3):152–161.

Dorazio, R. M. Bayesian data analysis in population ecology: Motivations, methods, and benefits. 58(1):31–44.

Efron, B. and Tibshirani, R. J. *An Introduction to the Bootstrap.* Chapman and Hall/CRC, 1st edition edition.

Ellison, A. M. Bayesian inference in ecology. 7(6):509–520.

family=Valpine, given=Perry, p.-u., Turek, D., Paciorek, C. J., Anderson-Bergman, C., Lang, D. T., and Bodik, R. Programming With Models: Writing Statistical Algorithms for General Model Structures With NIMBLE. 26(2):403–413.

Flowers, H. J. and Hightower, J. E. Estimating Sturgeon Abundance in the Carolinas Using Side-Scan Sonar. 7(1):1–9.

Flowers, H. J. and Hightower, J. E. A novel approach to surveying sturgeon using side-scan sonar and occupancy modeling. 5(1):211–223.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. *Bayesian Data Analysis*. Chapman and Hall/CRC, 3rd edition edition.

Gelman, A., Lee, D., and Guo, J. Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization. 40(5):530–543.

Gelman, A. and Rubin, D. B. Inference from Iterative Simulation Using Multiple Sequences. 7:457–472.

Hayes, D. B. and family=Brodziack, given=JKT, g.-i. Reply: Efficiency and bias of estimators and sampling designs for determining length weight relationships of fish. 54(3):744–745.

Hearn, W. S., Hoenig, J. M., Pollock, K. H., and Hepworth, D. A. Tag Reporting Rate Estimation: 3. Use of Planted Tags in One Component of a Multiple-Component Fishery. 23(1):66–77.

Hearn, W. S., Sandland, R. L., and Hampton, J. Robust estimation of the natural mortality rate in a completed tagging experiment with variable fishing intensity. 43(2):107–117.

Hewitt, D. A., Janney, E. C., Hayes, B. S., and Shively, R. S. Improving inferences from fisheries capture-recapture studies through remote detection of pit tags. 35(5):217–231.

Hightower, J. E. and Grossman, G. D. Comparison of Constant Effort Harvest Policies for Fish Stocks with Variable Recruitment. 42(5):982–988.

Hightower, J. E. and Harris, J. E. Estimating Fish Mortality Rates Using Telemetry and Multistate Models. 42(4):210–219.

Hightower, J. E., Loeffler, M., Post, W. C., and Peterson, D. L. Estimated Survival of Subadult and Adult Atlantic Sturgeon in Four River Basins in the Southeastern United States. 7(1):514–522.

Hilborn, R. and Walters, C. J. *Quantitative Fisheries Stock Assessment*. Springer US.

Hooten, M. B. and Hobbs, N. T. A guide to Bayesian model selection for ecologists. 85(1):3–28.

Jiang, H., Pollock, K. H., Brownie, C., Hoenig, J. M., Latour, R. J., Wells, B. K., and Hightower, J. E. Tag Return Models Allowing for Harvest and Catch and Release: Evidence of Environmental and Management Impacts on Striped Bass Fishing and Natural Mortality Rates. 27(2):387–396.

Kéry, M. *Introduction to WinBUGS for Ecologists: Bayesian Approach to Regression, ANOVA, Mixed Models and Related Analyses*. Academic Press, 1st edition edition.

Kéry, M. and Schaub, M. *Bayesian Population Analysis Using WinBUGS: A Hierarchical Perspective*. Academic Press, 1st edition edition.

Law, A. M. and Kelton, W. D. *Simulation Modeling and Analysis*. McGraw-Hill.

Link, W. A. and Barker, R. J. Bayesian inference : With ecological applications - NC State University Libraries Catalog.

Link, W. A. and Barker, R. J. *Bayesian Inference: With Ecological Applications*. Academic Press, 1st edition edition.

Link, W. A. and Eaton, M. J. On thinning of chains in MCMC. 3(1):112–115.

Lorenzen, K. The relationship between body weight and natural mortality in juvenile and adult fish: A comparison of natural ecosystems and aquaculture. 49(4):627–642.

Lunn, D., Jackson, C., Best, N., Thomas, A., and Spiegelhalter, D. *The BUGS Book: A Practical Introduction to Bayesian Analysis*. Chapman and Hall/CRC, 1st edition edition.

Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. The BUGS project: Evolution, critique and future directions. 28(25):3049–3067.

Lunn, D. J., Thomas, A., Best, N., and Spiegelhalter, D. WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility. 10(4):325–337.

McCarthy, M. A. *Bayesian Methods for Ecology*. Cambridge University Press, 1st edition edition.

Millar, R. B. and Meyer, R. Bayesian state-space modeling of age-structured data: Fitting a model is just the beginning. 57(1):43–50.

Mäntyniemi, S., Romakkaniemi, A., and Arjas, E. Bayesian removal estimation of a population size under unequal catchability. 62:291–300.

Ogle, b. D. H. *Introductory Fisheries Analyses with R*. Chapman & Hall/CRC The R Series. Boca Raton, FL : Chapman and Hall/CRC, [2018]., first edition. edition.

Plummer, M. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling.

Pollock, K. and Pine, W. The design and analysis of field studies to estimate catch-and-release mortality. 14:123–130.

Pollock, K. H., Hoenig, J. M., Hearn, W. S., and Calingaert, B. Tag Reporting Rate Estimation: 1. An Evaluation of the High-Reward Tagging Method. 21(3):521–532.

Pregler, K. C., Hanks, R. D., Childress, E. S., Hitt, N. P., Hocking, D. J., Letcher, B. H., Wagner, T., and Kanno, Y. State-space analysis of power to detect regional brook trout population trends over time. 76(11):2145–2155.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ricker, W. E. *Computation and Interpretation of Biological Statistics of Fish Populations*. Number 191 in Bulletin. Dept. of Fisheries and Oceans : Minister of Supply and Services Canada.

Robson, D. S. and Regier, H. A. Sample Size in Petersen Mark–Recapture Experiments. 93(3):215–226.

Scherrer, S. R., Kobayashi, D. R., Weng, K. C., Okamoto, H. Y., Oishi, F. G., and Franklin, E. C. Estimation of growth parameters integrating tag-recapture, length-frequency, and direct aging data using likelihood and Bayesian methods for the tropical deepwater snapper Pristipomoides filamentosus in Hawaii. 233:105753.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., and Van Der Linde, A. Bayesian measures of model complexity and fit. 64(4):583–639.

Vetter, E. Estimation of Natural Mortality in Fish Stocks - a Review. 86(1):25–43.

Wang, Y.-G., Thomas, M. R., and Somers, I. F. A maximum likelihood approach for estimating growth from tag–recapture data. 52(2):252–259.

Xie, Y. *Dynamic Documents with R and Knitr*. Chapman and Hall/CRC, 2 edition.

Xie, Y. (2024). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.38.

Zhang, Z., Lessard, J., and Campbell, A. Use of Bayesian hierarchical models to estimate northern abalone, Haliotis kamtschatkana, growth parameters from tag-recapture data. 95(2):289–295.