Ch. 7

Introduction to mark-recapture models

"Man is the only kind of varmint who sets his own trap, baits it, and then steps in it."
-- John Steinbeck

Questions to ponder:

- *What is the difference between closed and open populations?*
- *What is meant by 'encounter probability'?*
- *How do catch-per-unit-effort methods compare to mark-recapture methods?*

Mark-recapture data and analyses

The use of mark-recapture data has a long history in wildlife management and ecological studies. The placement of marks on animals will continue to be a staple method in the toolkit of a wildlife biologist. Mark-recapture data allows us to answer questions such as: "how many animals are there in this population?" Or, "Are some animals at a higher risk of mortality than other animals?" Or, "what is the probability that animals in the population will remain in the population?"

As technology has changed, biologists have added more type of marks to their toolkit—for example, the use of Passive Integrated Transponders (PIT) and satellite telemetry. But, the basic approach to analysis of mark-recapture data has not changed in decades. Here, we will explore some newer methodologies and analysis models, but we begin with the standard: a "closed" mark-recapture model (Chapter 8) to consider animal abundance questions. Chapters 9-13 will explore "open" mark-recapture models to address questions of mortality and movement.

The common denominator of all mark-recapture methodology is that an animal is captured in some manner and marked. The animal is then released and captured at a subsequent time period. Some marking methods provide unique identifications (e.g. "pheasant #133"), but other marking methods only show than an animal is marked (e.g., a splotch of white paint on the scales of an iguana).

Samples and encounter probability

We know that when we capture animals in most populations, we are not able to capture the entire population at one time. Thus, by definition, we are sampling the population. And, we will try to make inferences about the population from our sample.

What if you were told that a biologist went to three study sites, placed some sort of net or trap, and recorded the capture of 100 unique individuals at each site? Could you make an inference about the size of the population without knowing any other information? The inference, certainly, is limited—the best inference is that there are at least 100 individuals in each population.

But, how many animals are really in the population? What additional information about our trapping effort do we need to know?

We must know something about the capture, or encounter, probability. That is, what is the probability that an individual living in the population is captured? In mathematical language, we have our sample, *n*, but we want to estimate the population size, *N*. We know that if capture probability is *p*:

$$\hat{N} = \frac{n}{\hat{p}}$$

So, if we told you that in the first population, $p = 0.30$, you could tell me that the population size is 333. We know that because we captured 30 percent of the population (another way of saying that an animal had a 30% chance of being captured in our trapping program). So, 100 animals represented 30 percent of the population. And, by math we can see that $\widehat{N} = 100/0.30 = 333$.

Types of encounters

Specific types of research often have different terms to indicate the type of encounter that is involved:

"Capture probability" typically refers to the probability that an individual is captured during a given time period in an actual trap or net.

"Recovery probability" is defined as the probability that a hunter will, during a given time period, shoot a marked animal—shot and recovered animals cannot be released to the wild because they are dead. Thus, "recoveries" are a special type of mark/recapture data.

"Re-sighting probabilities" are defined as the probability that an animal marked with colored or numbered tags will be seen by a person and recorded (i.e., not in a trap) during a given time period.

Can you estimate the population size if $\hat{p} = 0.50$? What if $\hat{p} = 0.70$?

You should find that $\widehat{N} = 100/0.50 = 200$ and $\widehat{N} = 100/0.70 = 142$.

So, capturing 100 animals in each population is only our sample, and we cannot infer much about the population size until we know the encounter probability, *p*.

NOTE: You may see, especially in older literature, the term 'encounter rate' or 'capture rate'. Most scientists (and editors of journals!) now prefer the term "encounter probability" as a true "rate" has a time-specific element attached, such as a flow rate of a stream (meters/second). Although it is true that 'survival rates' also have a time element that should be specified (annual, monthly, daily, etc.), the stochastic nature of binomial-type events is emphasized with the use of the term "probability".

Catch-per-unit-effort

There are implications to standard methodologies that assume equal catchabilities—fisheries biologists and small mammal biologists, for example, historically calculated "Catch-per-unit-effort" (CPUE) statistics. CPUE might be expressed as 350 fish/net or 1300 small mammals per trap night.

But, if a net captures 350 fish of size >=700mm, 350 fish of size between 200 and 500mm, and 350 fish of size <200mm, we are in the same situation as we were with our 100 animals captured at each of three study sites. The size of the holes in the net, perhaps, might favor the capture of fish in the 200-500 mm size—larger fish bounce off and small fish swim through it. So, if capture probability is highest for the 200-500mm fish and lowest for the other two size classes, our CPUE cannot be used to infer anything about population size. In fact, there would be many more of the smallest- and largest-sized fish, in our example, than the middle size class. Why? Because capture probability of the smallest and largest size classes was very low. For example, if p=0.10 for those size classes, $\widehat{N}$ = 350/0.10 = 3500 fish. And, if p=0.9 for the middle size class, $\widehat{N}$ = 350/0.9=389 fish.

Despite the original inference from CPUE of similar numbers of each size category, we now see that there are many more small and large fish than mid-sized fish. For this reason, most fisheries biologists use CPUE results with caution because they know that their nets' capture efficiencies are different for individuals of different sizes.

Some logic

Before we launch into the mathematics of the simple estimator for population size, let's take a look at the logic behind the estimator. In Figure 7.1, you see two populations (one at top, the other at the bottom). A sample of 100 animals is taken during the first capture occasion in both populations. Things are looking fairly even at the moment. But, we should be aware that we can't infer much about population size, yet. We only know that each population has at least 100 animals!

Next, a second capture occasion is held and another 100 animals are captured (perhaps our traps or nets only hold 100 animals?). But, some interesting differences arise, now. In the first population only 20 of the 100 animals captured in time 2 are marked. And, in the second population, 80 of the 100 animals captured in time 2 are marked (Figure 7.1).

Can you suggest which population is larger than the other?

You should have answered "Population #1" (Figure 7.1). Why? Because we marked 100 animals in both populations in time 1. And, in time 2, we appear to have marked almost all (80% in fact) of Population #2, while we appear to have only marked 20% of Population #1. If 100 animals are only 20% of the population in Population #1, it would seem to be very big—much bigger than Population #2. In a few pages, we will put some equations to this logic. For now, you should be able to look at this example and see we can make a much better inference about population size with a very simple mark-recapture experiment.

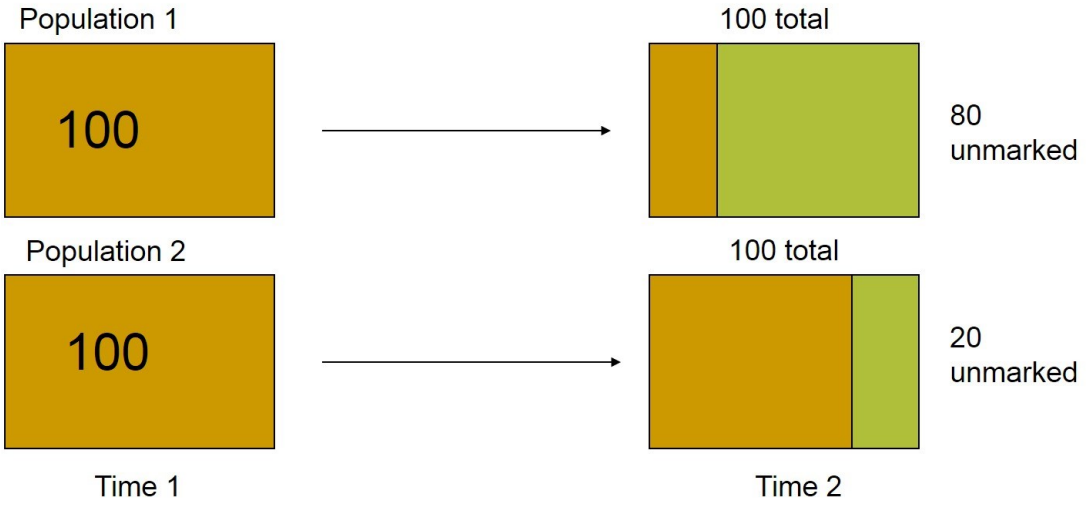

Figure 7.1: *Two populations with very different sizes (Population 1 and 2) are sampled with 100 animals marked in each population during Time 1. During Time 2 another 100 animals are sampled from each population. The number of marked animals captured varies between populations in Time 2. The presence of many unmarked individuals in Population 1, relative to the number of unmarked individuals in Population 2 suggests the Population 1 is bigger than Population 2.*

The cohort

An important concept in the use of mark-recapture analyses is the "cohort". A cohort is the number of marked animals released during time period *t*. We use cohorts to establish capture histories, which are the basis for all analyses of mark-recapture data. Capture histories are records that show when an individual has been captured during a study. For example, if a cohort of individuals is captured during time period 1 and then released immediately, those animals can have four different capture histories during a study with three capture occasions: "111", "110", "101", and "100". For simple capture histories, we use a "1" to indicate that an animal has been captured (a 'success'). And, we use a "0" to indicate that an animal was not captured (a 'failure').

This type of data is called **"binomial"**—two forms (1/0, or success/failure).

But, the released cohort is critical to our analyses—all subsequent work is conditioned (or statistically based) on that initial sample of animals that are captured and released. Mark-recapture analyses, at their core, are an attempt to estimate something you don't know (e.g., population size). To accomplish that goal, you must have something you do know—and when

we release a cohort of animals, we know how many we released. And, that will be very important, as you will see.

For example, if you were to decide to study ducks in the state of New York, USA, you could not start by collecting bands, or rings, or ducks recovered by hunters in New York. Those bands, or rings, would provide useless data unless you knew how many birds had been banded, or ringed, initially. That is, without knowledge of the cohort, all you have is a shiny pile of bands in your hand!

Closed or open?

Mark-recapture analyses can be divided into two main types of analyses: analyses of 'closed' populations and analyses of 'open' populations.

Remember the basic equation of population growth (Chapter 2) is a function of births (B), deaths (D), immigration (I), and emigration (E): $\Delta N = B + I - D - E$.

We refer to 'closed' populations as having $\Delta N = 0$. That is, no births, deaths, or movements of animals in or out. Although we know that no species exists in closed populations, populations may be considered 'closed' if we study them over very short time periods. This is necessary to estimate population size. And, it is important to note that 'very short time period' is relative to each species. The scale over which animals are born and die is short for small rodents, relative to elephants. So, a month-long study of small rodents might be considered to be too long for our closure assumption (many small rodents might die or be born during a month). But, a 4-month study of elephants might be considered very acceptable (very few deaths and births are likely during such a period).

We will come back to questions of closure of populations later, but this is a key consideration when you start to design a study for your species. When are the birth pulses? How often do individuals die? After a given time period (e.g., day, week, month), will animals have left your study site?

Alternatively, models of "open" populations are often more realistic. When we study 'open' populations, we assume that deaths are occurring. Mark-recapture methods treat movements in unique ways, but many types of analyses of open populations assume movements are occurring as well.

Therefore, studies of open populations are usually longer in duration than studies of closed populations (however, remember that a closed study of elephants might be longer than an open study of mice!). In fact, we want deaths to occur during studies of open populations. If animals do not die (or, for more complex models, leave or enter the study site), our study will not be able to estimate a useful survival (or movement) probability.

Models matter

As we start to envision our analyses as models of systems, it may be useful to compare simple open and closed model structures. As you begin working with mark-recapture data, you must start to think in terms of probability statements. So, let's start with a simple example:

Let's consider the situation where you captured a sample of individuals and released them as a cohort during time 1. Then, during time 2 you recapture some of that cohort. Given that you captured the animals during time 1, they will either have a "11" or "10" capture history (obviously, other animals not captured during time 1 could have a "01" or a "00" capture history as well, correct?).

So, what is the probability of having a capture history of "11"? It depends completely on the 'model' under which we are working. Is it a closed system? If so, the only thing that had to happen for our animal released in time 1 to have a "11" capture history is…to be captured again. So, we can represent that as a probability statement: p_2—the probability of capture during time 2.

What is the probability of having a 10 capture probability? It is $1-p_2$, or the probability of NOT being captured during time 2. It is as simple as that.

Type of model	Capture history	Probability of animal having capture history
Closed population	11 10	p_2 $1-p_2$
Open population, CJS	11 10	S_1p_2 $S_1(1-p_2)+(1-S_1)$

Note: probabilities for capture histories are conditioned on capture and release in time 1

Now, let's change our model. Let's say, instead, that we are doing an analysis of animals in an open population (specifically, a "Cormack-Jolly-Seber," or CJS-type, open population—more on that in Chapter 8). We have the same animals released in time period 1. Now…what has to happen for them to have a capture history of "11"? It's going to be more complicated than the probability in our closed model—because there is something else happening: *some animals are dying.*

For an animal to have a capture history of "11" in an open population, the animal must first survive the interval in which they are released, and then they must be captured in the next interval. So, the probability would be: S_1p_2. We multiply the two probabilities, S_1 and p_2, together, as the animal must do both—survive and be captured. We will cover this again when we discuss open populations, but we assume in analyses of open populations that the cohort of animals is released simultaneously at the beginning of time 1. So, they must survive through time period 1, and hence we used the S_1 designation to refer to survival during time period 1. Capture happens at the beginning of the next time period, so we use the p_2 to designate capture probability in time 2.

What is the probability of having a capture history of "10"? You might simply look at the closed model and suggest: $S_1(1-p_2)$. And, you would be partially correct. That is, an animal *can* have a capture history of "10" by surviving time 1 and then avoiding capture in time 2.

But, there is *another alternative* that also results in a capture history of "10"—the animal might die during time 1. Yes, as the famous Alfred, Lord Tennyson wrote, Nature is *"red in tooth and claw."* And, that affects our statement of capture probability—we must account for both possibilities. The total probability for having capture history of "10" is the *sum* of the two alternatives: capture avoidance and death. So, we write it as: $S_1(1-p_2) + (1-S_1)$. For a further discussion of CJS-type capture histories with more than two capture periods see Chapter 10.

Conclusion

Biologists use mark-recapture models to estimate parameters for populations, and the models are based on probability statements that require certain assumptions about the population in question. To estimate population size, we use models that assume a 'closed population' (no deaths or movement in/out). To estimate survival, we use models that are based on 'open populations'. Capture histories (e.g., "1001100") are the typical type of input data for these analyses, and our analyses are conditioned on the released cohort: animals released at the same time.

For more information on topics in this chapter

Amstrup, S. C., T. L. McDonald, and B. F. J. Manly. 2005. Handbook of capture-recapture analysis. Princeton Univ. Press: Princeton, NJ.

Conroy, M. J., and J. P. Carroll. 2009. Quantitative Conservation of Vertebrates. Wiley-Blackwell: Sussex, UK.

Cooch, E., and G. White. 2014. Chapter 1: First Steps. *In* Program MARK: a gentle introduction, 12th edition, Cooch, E. and G. White, eds. Online: http://www.phidot.org/software/mark/docs/book/pdf/chap1.pdf

Nichols, J. D. 1992. Capture-recapture models. BioScience 42: 94-102.

White, G. C., D. R. Anderson, K. P. Burnham, and D. L. Otis. 1982. Capture-recapture and removal methods for sampling closed populations. Los Alamos National Laboratory. LA-8787-NERP. 235 pp.

Williams, B. K., J. D. Nichols, and M. J. Conroy. 2002. Analysis and management of animal populations. Academic Press, San Diego.

A female white-tailed deer (Odocoileus virginianus) is marked with a numbered ear tag and a radio-telemetry collar in western Iowa, USA. Photo by Greg Clements, University of Nebraska-Lincoln.

Ch. 8

Estimation of population size: closed populations

"My parents removed to Missouri in the early 'thirties; I do not remember just when, for I was not born then and cared nothing for such things…[our] home was made in the wee village of Florida, in Monroe County, and I was born there in 1835. The village contained a hundred people and I increased the population by 1 per cent. It is more than many of the best men in history could have done for a town. It may not be modest in me to refer to this, but it is true. There is no record of a person doing as much--not even Shakespeare. But I did it for Florida, and it shows that I could have done it for any place--even London, I suppose."
-- Mark Twain

Questions to ponder:

- *Is there a fixed period of time that should be considered 'closed' for all species?*
- *What kind of information is needed for a simple Lincoln-Petersen estimate of N?*
- *How can I assess the potential for bias with a simple Lincoln-Petersen estimate when I have small samples?*

A bit of history (*histoire*)

The Lincoln-Petersen estimator was first used to estimate the population size of a **closed** wildlife population in 1896 when **C. G. Johannes Petersen**, a marine biologist in Denmark, estimated the size of a population of plaice (*Pleuronectes platessa*)—a species of flatfish, similar to flounder (Petersen 1896).

Frederick C. Lincoln, an American ornithologist, described the method in 1930 in USDA Circular 118: "*Calculating Waterfowl Abundance on the Basis of Banding Returns*." Lincoln, who also developed the 'flyway' concept for migratory birds, devised the method to estimate the continental population size of waterfowl (Lincoln 1930).

$$\hat{N} = \frac{n_1 n_2}{m_2}$$

Although we give Lincoln and Petersen credit for this method, the general idea of using a 'known ratio' to estimate components of an 'unknown ratio' is much older. Ken Pollock, of North Carolina State University, suggests that the first use of the Lincoln-Petersen-type estimator was by a chap named **Laplace** in 1783. Laplace wanted to estimate something useful—the population of France. There were a lot of people in France, and census-type data were not common at this time. But, one thing was known—the number of births in the whole of France. So, Laplace knew how many babies (think of babies as 'marked animals' in time 1).

In a small subset of the parishes (a small, local geographic region similar to 'counties' in the US) in France, Laplace could obtain a relatively accurate count for the total number of people in the parishes (think of this as n_2), and he was also able to obtain the number of babies born in those same parishes (again, babies are 'marked individuals', so think of this as m_2).

So, he was ready to apply the estimator. If he assumed that the birth rate (babies/population) was the same in those parishes as it was in the whole of France, he could estimate the population of France (N) as equal to $n_1 * n_2/m_2$ (Laplace 1783).

Perhaps it is useful for you to see that this estimator has been used for a very long time?!

Closed population models

In Chapter 7, we compared open and closed population models. Here, we will continue with the topic of closed populations. Thus, we will assume no births and no deaths during the sampling of our population. The simplest form of a closed mark-recapture analysis is called the **Lincoln-Petersen** method. Although it is simple, the Lincoln-Petersen method provides an unbiased maximum likelihood estimate of N for a two-occasion sample. The underlying assumption is that the proportion of marked animals remains the same during the two sample periods.

The L-P method begins with the basic idea that an unknown portion of the population is captured and marked during the first time period. And, although you don't know what that proportion is, we can denote it as the capture probability, p. That is, the unknown proportion of the population that is marked during the first time period is equal to the capture probability (if $p = 0.35$ during time 1, then you should capture 35 percent of the population).

But, we obviously don't know anything except how many animals we captured. This sounds like the example earlier in this chapter of 100 animals captured in a study area, doesn't it? Well, never fear, Lincoln and Petersen both came up with a brilliant plan to estimate p, and therefore estimate N.

The brilliantly simple idea is that if you have a second capture occasion and you look around at all of your captured animals, the proportion of the animals with marks in your sample SHOULD be the same proportion of the population that you captured (and marked) during time 1. So simple! We can see this graphically in Figure 8.1.

If those proportions are equal, then we can use an equation to state:

$$\frac{m_2}{n_2} = \frac{n_1}{N} \quad \Longrightarrow \quad \hat{N} = \frac{n_1 n_2}{m_2}$$

That is, the portion of animals caught (n_1) from the population (N) during the first time period is equal to the portion of the sample during time 2 (n_2) that have marked on them (m_2). By rearranging with algebra, we can solve for the only unknown in the equation, which is the parameter whose value we want to estimate: N.

Lincoln and Petersen's method is simple, and has a few **assumptions**. First, we assume that capture probability is the same for all animals in the population. Interestingly, p_1 and p_2 do not have to be the same, nor do the capture methods need to be the same. In fact, Lincoln used traps to band and mark ducks and then he used hunter kills as the second capture method.

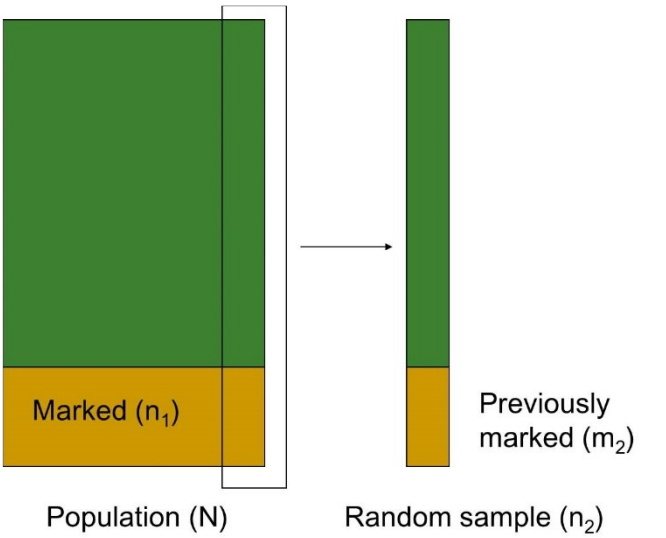

Figure 8.1: *The logic behind the 2-sample Lincoln-Petersen approach to estimation of population size. First (left), a population of unknown size (N) is sampled and n_1 individuals are marked (the unmarked portion of the population is represented by dark color). A second random sample (n_2) of the population is taken (right). The proportion of marked animals can be determined in the second sample, and it should be equal to the proportion of marked animals in the population immediately after the first sampling/marking.*

Other assumptions:

- the marking of an animal does not affect p_2.
- marks are not lost between capture occasions.
- all marks at time 2 are reported.
- each sample of the population (during time 1 and 2) is a random sample of the population.

You try it!

Ready to try your hand at a problem?

Pollock et al. (1990) described a study in Florida in which 148 northern bobwhite (quail [*Colinus virginianus*]) were marked with leg bands after being trapped in cornfields during January of 1982. During a 3-day controlled hunt in February 1982, 39 of 82 shot quail were determined to have leg bands.

What is the estimate of the population size ($\hat{N}$) by the Lincoln-Petersen estimator? To provide the estimate, you need to figure out the values for n_1, n_2, and m_2. You'll find the answer at the end of this chapter.

Generalizing the Lincoln-Petersen

Thus far, we have used summaries of n_1, n_2, and m_2 to estimate population size. But, we need to be working towards an understanding of the generalization of the method—how does it work when you have a set of capture histories that will be the input for a more complicated analysis in a software package such as Program MARK?

Four possible capture histories	Probability of having capture history	Number of animals with this capture history
10	p_1q_2	$n_1 - m_2$
01	q_1p_2	$n_2 - m_2$
11	p_1p_2	m_2
00	q_1q_2	*N-r (NOTE: can't observe)*

The table above shows the four possible capture histories of animals in a population under the 2-sample Lincoln-Petersen research design. Note that the first three will be the animals in your data set, and the 4th capture history ("00") is not observable—these are the animals that are never captured.

We can also define the capture probability as p and the probability of not being captured as q (which is also, by definition, $1-p$). You have seen n_1, n_2, and m_2 before. The last definition we need is for r, which is the total number of captured animals. Although r is helpful, we are really interested in N, the population size.

If we allow capture probability to be different in each time interval (thus, the use of p_1 and p_2 to indicate that capture probability may be different in time 1 and time 2), we can see that the probability of having a capture history of 11 is: $p_1 * p_2$. That is, the probability of being captured during time 1 is p_1 and the probability of being captured during time 2 is p_2. To have a capture history of 11, *both* of these events have to occur (i.e., captured in each period), so we multiply the two probabilities together.

Similarly, to have a history of 01, an animal must avoid capture in time 1 and be captured in time 2. Thus, the probability is: $q_1 * p_2$. And, the opposite is true for a history of 10: $p_1 * q_2$.

These are the probability statements that a maximum likelihood estimator will use to find the best estimate for p. And, when we estimate capture probability, we can estimate N. Remember that, in general form, $N = n/p$? That is, if we can estimate the proportion of the population that we didn't capture, we can estimate population size—because we know the number of animals we captured. In the table above, if you add the first three cells together, we can calculate how many

animals we captured as the sum of animals with 11, 10, and 01 capture histories. Or, mathematically, $r = (n_1-m_2)+(n_2-m_2)+m_2$.

Chapman wasn't satisfied

Like all maximum likelihood estimators, the L-P estimator is unbiased for very large samples. But, it is biased for small samples. This bias can be shown by putting 10 white balls in a hat (a very small population, which ensures small samples). If you randomly draw a sample of 4 balls (a small sample) and mark them, we'll now have 6 unmarked and 4 marked balls in our 'population' of balls. We know (because we set up this experiment) that the marked proportion of the population is 0.4. And, $n_1 = 4$.

Now, if we draw a sample of four more balls, we could get a variety of results. Using simple logic, if 40% of our population is marked, we might predict that we'll most likely get two marked balls in our sample of four balls that we randomly draw. You'll notice that it is impossible for us to get exactly 40% of our sample (0.4 * 4 = 1.6, and we are not splitting balls in half…just as you cannot split animals in half during sampling!). And, that is the reason that the L-P is biased for small samples.

To carry the example forward, if m_2=2 and n_2=4, then our population estimate would be: $\widehat{N}$=4*4/2 = 8.

Now, let's allow a random occurrence to happen…and let's say we got, instead, 1 marked ball in our sample. Such an event might occur with such a small sample. Now, $m_2 = 1$. And, $\widehat{N}$ = 4*4/1 = 16. A change in one marked ball in our sample doubles our population estimate.

We can pause and state that most scientists would never attempt to do a mark-recapture exercise if the size of the population was thought to be near 10! We have used such a small population and small samples to illustrate the problem. But, it might be common for your samples to be less than 30 in some situations. For example, what happens to the L-P method if $m_2 = 0$ (no marked animals captured)? $\widehat{N}$ is undefined, as you cannot divide n_1*n_2 by 0.

The Lincoln-Petersen estimator was modified by Chapman, and the resulting estimator is known as the "**Chapman modification**" (yes, we biologists are often very literal in the names we give to methods!). The modification has less bias for small samples than the L-P estimator. Chapman based his modification on the hypergeometric probability distribution, which is used when sampling without replacement. We mention this simply to note that there is statistical rigor and theory behind the modification. However, it is not a maximum likelihood estimator:

$$\hat{N} = \frac{(n_1 + 1) \times (n_2 + 1)}{(m_2 + 1)} - 1$$

The modified method does solve, mathematically, the problem of situations when m_2 is 0. However, we would remind the reader that if a study does not result in capturing a marked animal, it is more likely that the best course of action is to reconsider the study design. How might you mark more animals during time 1? Or, how might one might work with a species that is either so rare or so trap-timid that no marked animals are caught.

You try it!

Estimate $\hat{N}$ using the Pollock et al. (1990) quail data from above. Were the samples small enough to warrant use of the Chapman method? Compare your answers to those found at the end of the chapter.

Schnabel had more data…

The **Schnabel method** expanded the basic philosophy of Lincoln-Petersen to be used when the number of capture occasions is larger than 2. When Zoe Emily Schnabel, a mathematician at the University of Wisconsin, developed this method in the 1930's, only 2-occasion studies could be used. So, her work redefined possibilities for wildlife science. The Schnabel method is a closed form method, which means it can be condensed to an equation. But, it is an approximation to a maximum likelihood estimation method that can be found in modern software tools. White et al. (1982) noted that many biologists have disregarded a note that Schnabel wrote about her method: *"It should be emphasized, however, that none of the solutions can be expected to provide more than a general estimate of the order of magnitude of the total population."*

For this method, C_t is defined as the number of captures during time t. R_t is the number of recaptures (marked animals, captured again) captured during time t. And, M_t is the number of animals marked in the population, at time t.

$$\hat{N} = \frac{\sum_t (C_t \times M_t)}{\sum_t R_t}$$

Note: To properly calculate M_t*, do NOT count animals that are marked for the first time during time t. That is,* M_t *is the number of animals marked in the population just as you begin to conduct your observations of the traps or nets. Or, you can also think of* M_t *as the number of marked animals in the population that were at 'risk' of being trapped during time t.*

You try it!

During 2006, students from the University of Nebraska-Lincoln traveled to a 7-ha island off the coast of Puerto Rico. The island was named Isla Magueyes, and it had a population of Cuban rock iguanas (*Cyclura nubila*). These iguanas are a species of conservation concern elsewhere in their range, but their population was causing damage on Isla Magueyes. A small population of iguanas had been left on the island when a zoo was removed. No one knew how many iguanas were on the island.

So, the students marked animals each morning for three days. On the first day, the students found 155 iguanas, and they squirted them with small marks of latex paint to mark them. The marks were not individual ID's, but the marks were easy to see on day 2 and 3.

On day 2, the students saw 109 marked animals and 66 unmarked animals. They marked any animals that were not marked. On day 3, the students observed 116 marked animals, and only 15 unmarked animals. Use the Schnabel method to estimate the population size.

	Animals Observed	**Animals with Marks When Seen**
Day 1:	155	0
Day 2:	175	109
Day 3:	131	116

The first step is to complete a summary table. M_t can be a tricky statistic to calculate, so we have added a 'newly marked column' here. The numbers of newly marked animals have been added to help you stay on track. In addition, we have noted that $M_1 = 0$ (no marked animals in the population before the first period).

Day	**C_t** (Captures during time t)	**R_t** (Number of recaptures captured during time t)	**Number newly marked**	**M_t** (Number of animals marked in the population, just before you conduct trapping during time *t*)
1			155	0
2			66	
3			15	

Population size estimate: _____

See the end of this chapter for answers after you have given this a try.

Modern closed-population methods

The analytical methods described above can be accomplished by **batch marking** individuals, rather than going to the effort of applying an individual identifying letter and/or number combination to each animal. For example, the Lincoln-Petersen and the Schnabel data can be gathered by applying a basic mark—like the splotch of paint used on the iguanas in our example. In the third time period, there is no need to know whether a marked animal was previously marked in the first or second time period—it is enough to know that it was marked at some point in the past.

The assumption of the Lincoln-Petersen and the Schnabel is that all animals in the population at a given time period have the same probability of capture. What if that is not true? For example,

what if the iguanas that encountered students were less likely to be in the accessible areas of the island on the second day, while their unmarked friends were content to sunbathe on sidewalks that would eventually be used by students who marked them? If we break that assumption, the L-P estimator does not work as it should.

The alternative is to use **individual marks**, which allow the application of more modern closed-population estimation methods (Figure 8.2).

Figure 8.2: *A prairie rattlesnake (Crotalus viridis) is marked with a Passive Integrated Transponder (PIT) tag to provide individual identification upon recapture. Photo provided by Dennis Ferraro and used with permission.*

Otis et al. (1978) proposed several models for use in estimation of population size that are still in general use today. The most important contribution was the notion that animals could become "trap happy" (recapture rates are higher than initial capture rates) or "trap shy" (recapture rates are lower than initial capture rates). Today, we define **capture rate**, p, as the probability of initial capture and **recapture rate**, c, as the probability of recapture after being captured at least once.

And, although Otis et al. (1978) did not use AIC to compare alternative models that described variation in capture probability, their suggestion to compare multiple models led biologists toward the idea that capture estimates from "poor" models should not be used. Rather, estimates from models judged to be better should be used. The application of AIC for model selection (see Chapter 4) in ecology developed from this work.

By the use of individual marks, we can develop individual capture histories, which are not possible with batch marking (e.g., data often used for Schnabel method). Capture histories are important, because we can tie specific probability statements to each capture history—that is, each capture history has a specific probability of occurrence. For example, the probability of an animal having specific capture histories in a 10-occasion, closed-population, mark-recapture study is:

Capture history	Probability
0100111000	$(1-p_1)p_2(1-c_3)(1-c_4)c_5c_6c_7(1-c_8)(1-c_9)(1-c_{10})$
1000110101	$p_1(1-c_2)(1-c_3)(1-c_4)c_5c_6(1-c_7)c_8(1-c_9)c_{10}$
0000001101	$(1-p_1)(1-p_2)(1-p_3)(1-p_4)(1-p_5)(1-p_6)c_7c_8(1-c_9)c_{10}$

With an adequate sample of marked animals, you may use software that uses maximum likelihood estimators to estimate the capture and recapture rates, as well as population size for the species in your study. In the above example, we show variation caused by behavior (capture and recapture rates not equal), as well as time (different capture and recapture rates for each time period). However, you might explore alternative models. For example, we might hypothesize that behavior has no impact on recapture rates, and hence $c = p$. Therefore, the probability statement for the first capture history above could be represented using only p and no c (the recapture probability):

0100111000 $(1-p_1)p_2(1-p_3)(1-p_4)p_5p_6p_7(1-p_8)(1-p_9)(1-p_{10})$

A third model that we might explore could state that behavior causes differences in capture and recapture, but time is not important. So, the probability statement for the first capture history in the table above would not have any time-specific references (no subscript numbers):

0100111000 $(1-p)p(1-c)(1-c)ccc(1-c)(1-c)(1-c)$

And, last (at least for our simple example), we might hypothesize one last model—a null model—that neither behavior nor time cause capture rate to vary. Thus, the likelihood statement can be represented using only p and not c, the recapture probability. It also does not include time-specific p's (no subscripts):

0100111000 $(1-p)p(1-p)(1-p)ppp(1-p)(1-p)(1-p)$

Which model is best? We could use **AIC** to tell us which model is the most likely to represent reality, given our data (Chapter 4). And, we would report the population size and capture probabilities estimated by the best model.

Lukacs (2014) provides a description of many modifications of the closed-population mark-recapture models. We suggest the beginning user start with **full likelihood** (p and c) or **Huggins'** (p and c) models. Each allows the user to explore the basic models we have outlined above.

Conclusion

Simple models to estimate population size require the assumption of closure during the sampling period: no births, deaths, or movements in or out. The history of estimating the size of a population started with a simple 2-occasion mark-recapture experiment: the Lincoln-Petersen scenario. The Schnabel method allows an estimate based on more than one period, which paved the way for modern methods. All of these methods are based on the same general concept: for a set of captured and marked animals, if you have a second capture occasion and you look at the second set of captured animals, the proportion of the animals with marks in your sample is assumed to be the same proportion of the population that you marked during the first capture period. Most modern methods require individual marks on animals, and the methods allow the exploration of multiple hypotheses regarding the variation of capture probability in the population during the study.

Answers: You try it!

Pollock et al. (1990) quail problem: 148 northern bobwhite (quail) were marked with leg bands; of 82 shot quail in the next month, 39 were marked with leg bands.

$n_1 = 148$
$n_2 = 82$
$m_2 = 39$

$\hat{N} = 311$ (*rounded*)

Chapman modification for the Pollock et al. (1990) quail problem:

$\hat{N} = 308$ (*rounded*).

Were the samples small enough to warrant use of the Chapman method? The difference between 308 and 311 (a measure of the bias of the Lincoln-Petersen estimator) is pretty small (just a difference of 3 animals).

Schnabel method (iguana problem):

	Animals Observed	**Animals with Marks When Seen**
Day 1:	155	0
Day 2:	175	109
Day 3:	131	116

Day	$\mathbf{C}_t$	$\mathbf{R}_t$	**Number newly marked**	$\mathbf{M}_t$
1	155	0	155	0
2	175	109	66	155
3	131	116	15	155+66=221

$$\hat{N} = \frac{(155 \times 0) + (175 \times 155) + (131 \times 221)}{0 + 109 + 116} = \frac{0 + 27125 + 28951}{225} = 249.2$$

References

Laplace, P. S. 1783. Sur les naissances, les mariages et les morts. Mémoires de l'Académie Royale des Sciences de Paris 693–702.

Lincoln, F. C. 1930. Calculating waterfowl abundance on the basis of banding returns. United States Department of Agriculture Circular 118: 1–4.

Lukacs, P. 2014. Chapter 14: Closed population capture-recapture models. *In* Program MARK: a gentle introduction, 12th edition, Cooch, E. and G. White, eds. Online: http://www.phidot.org/software/mark/docs/book/pdf/chap14.pdf

Petersen, C. G. J. 1896. The yearly immigration of young plaice into the Limfjord from the German Sea. Report of the Danish Biological Station (1895) 6: 5–84.

Pollock, K. H., J. D. Nichols, C. Brownie, and J. E. Hines. 1990. Statistical inference for capture-recapture experiments. Wildlife Monographs 107: 3-97.

White, G. C., D. R. Anderson, K. P. Burnham, and D. L. Otis. 1982. Capture-recapture and removal methods for sampling closed populations. Los Alamos National Laboratory. LA-8787-NERP. 235 pp.

For more information on topics in this chapter

Amstrup, S. C., T. L. McDonald, and B. F. J. Manly. 2005. Handbook of capture-recapture analysis. Princeton Univ. Press: Princeton, NJ.

Conroy, M. J., and J. P. Carroll. 2009. Quantitative conservation of vertebrates. Wiley-Blackwell: Sussex, UK.

Williams, B. K., J. D. Nichols, and M. J. Conroy. 2002. Analysis and management of animal populations. Academic Press, San Diego.

Citing this primer

Powell, L. A., and G. A. Gale. 2015. Estimation of parameters for animal populations: a primer for the rest of us. Caught Napping Publications: Lincoln, NE.

A maned wolf (Chrysocyon brachyurus) is measured by a research team prior to marking with a radio-telemetry transmitter in Brazil. Photo taken by Adriano Gambarini. Lucia Corral: left, Nucharin Songsasen: right) and used with permission of the photographer.

have already been asked. So, first search the archive and if you can't find an answer, post your question or problem about RMark to the Program MARK - RMark section. If you have a question/problem, then others may also, so please don't be afraid to post. If I think the answer may take some back-and-forth, I may answer you directly off-list for an email exchange and then we can post a summary of our exchange on the forum. BUT, before you post, please search the forum as your question may have already been answered! Also, provide enough detail including the code and any errors interleaved with the code so we can see where the problem occurred. You can imagine that it is not very useful to get a post like, "I was analyzing my nest survival data and I got an error. Can someone help me!"

3 A Simple CJS Example

So let's get started. We'll consider a capture-recapture experiment in which animals were individually marked and released on 4 occasions. In our experiment the population is open and animals may die between occasions but we are not considering births or immigration because the model is only for events after the initial release. So these could be tagged fish being released at dams from a hatchery and we want to estimate their survival over time. This type of model is called Cormack-Jolly-Seber (CJS) which is the surnames of the 3 folks that developed this type of model.

On the 3 occasions following the initial capture, marked animals are recaptured and re-released. The data are represented by a capture history (ch) which is also called an encounter history. The values in the history are either 1 meaning the animal was caught (seen)/initially released or 0 meaning the animal was not caught(seen). Each position in the capture history represents the 4 occasions in order. For example, the history 1001 represents an animal that was marked on the first occasion and was released. It was not seen on occasions 2 and 3 but was seen on occasion 4. Likewise, the history 0110 was an animal first marked and released on occasion 2, seen on occasion 3 and then not seen on occasion 4. Note that the capture history 0001 is animals marked and released on occasion 4 but this provides no information for the CJS model and can be excluded. In closed capture or Jolly-Seber models those data are informative and are included in the data.

There are two types of parameters in this model: 1) apparent survival which is named Phi and represented by the Greek letter ϕ and 2) capture/recapture probability named and represented by the letter p. Now to complicate our example somewhat I'll assume that we have both males and females and they can be easily distinguished. One goal is to decide if survival varies by sex. Also we could consider the possibility that capture probability is sex-specific. We also want to know if survival is a function of the animal's weight at release.

We can represent the data in tabular format with rows being the release cohorts (1-3) and columns being the recapture occasions (2-4). I'm excluding release cohort 4 because it does not provide any information.

	Recapture Occasion		
Release Occasion	2	3	4
1	1	2	3
2		4	5
3			6

The first release cohort can potentially be recaptured on occasions 2,3 and 4. The second can only be recaptured on occasions 3 and 4 and the third has only one opportunity to be recaptured on occasion 4. I'll get to the meaning of the numbers in the table later. We can also represent the data in tabular format with respect to the survival process:

	Survival Interval		
Release Occasion	1 to 2	2 to 3	3 to 4
1	1	2	3
2		4	5
3			6

To be recaptured on occasion 2, 3 or 4, the animal has to survive until that occasion. The survival period will be denoted by the time at the beginning of the period (e.g. time=1 for survival in interval from time 1 to time 2); whereas the time for capture is the occasion time.

An animal released on occasion 1, has to survive until occasion 2 to be captured on occasion 2. If we use ϕ for survival and p for capture probability, then the probability an animal released at occasion 1 is captured at occasion 2 is $\phi_1 p_1$. The subscript index for ϕ and p is the number in the table cell. Likewise, the probability that an animal is first captured on occasion 3 after release on occasion 1 (ch=101.) is $\phi_1\phi_2(1-p_1)p_2$ because it has to survive until occasion 3, be missed on occasion 2 and then captured on occasion 3. The probability of observing each capture history (ch) is constructed in this manner. For example, the Pr(ch=0101) = $\phi_4\phi_5(1-p_4)p_5$. A slightly more difficult one is Pr(ch=0110) = $\phi_4 p_4(\phi_5(1-p_5)+(1-\phi_5))$ because there are two possible outcomes for the last position. It is a 0 because either the animal survived and was not caught or the animal died. While it is useful to understand how these probabilities are constructed, you will not need to construct them because MARK handles all of that for you.

All of the parameters (e.g., Phi and p) are combined into a single parameter vector θ and each has its own index (not necessarily unique). Below are the survival and capture indices where each are unique. These are called PIMs (parameter index matrices) in MARK terminology.

	Survival Interval		
Release Occasion	1 to 2	2 to 3	3 to 4
1	1	2	3
2		4	5
3			6

	Recapture Occasion		
Release Occasion	2	3	4
1	7	8	9
2		10	11
3			12

Using the single vector θ, now Pr(ch=0101) = $\theta_4\theta_5(1-\theta_{10})\theta_{11}$. Even though the indices are unique, the values they represent may not be unique, as I show below with design matrices.

Now in this CJS model, the 12 parameters cannot all be uniquely estimated so the parameters must be restricted (e.g., set equal) in some fashion. One way to do that is to use a different set of indices for the PIMs. For example, if you wanted a model with constant survival and constant capture probability you could create the following PIMs:

	Survival Interval		
Release Occasion	1 to 2	2 to 3	3 to 4
1	1	1	1
2		1	1
3			1

	Recapture Occasion		
Release Occasion	2	3	4
1	2	2	2
2		2	2
3			2

and θ would be a vector with 2 values.

Another way to construct the same model would be with a design matrix (DM). Consider the following DM which has a row for each index and two columns for the estimated parameters. The indices (rows) are for the real parameters and the columns are the estimated beta parameters. The real parameters are computed from the beta parameters.

Index	β_1	β_2
1	1	0
2	1	0
3	1	0
4	1	0
5	1	0
6	1	0
7	0	1
8	0	1
9	0	1
10	0	1
11	0	1
12	0	1

There are 2 estimated beta parameters: one for survival and one for capture probability. They are labeled β_1 and β_2 respectively and are called beta parameters in the MARK/RMark output. If X is the design matrix and β is the column vector of parameters then multiplying the matrix X and β represented by $X\beta$ will give the beta values for each real parameter index. In the above example, the multiplication simply assigns β_1 to indices 1-6 and β_2 to indices 7-12. Now let's consider an example in which ϕ is constant but all of the p's differ. A DM for such a model might look like:

Index	β_1	β_2	β_3	β_4	β_5	β_6	β_7
1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0
5	1	0	0	0	0	0	0
6	1	0	0	0	0	0	0
7	0	1	0	0	0	0	0
8	0	1	1	0	0	0	0
9	0	1	0	1	0	0	0
10	0	1	0	0	1	0	0
11	0	1	0	0	0	1	0
12	0	1	0	0	0	0	1

Now the vector β has 7 values. The product $X\beta$ assigns β_1 to indices 1-6, β_2 to index 7, $\beta_2 + \beta_3$ to index 8, $\beta_2 + \beta_4$ to index 9, ..., $\beta_2 + \beta_7$ to index 12.

While you may not have seen a DM before you used them when you learned simple and multiple linear regression. For simple linear regression, you were probably presented with an equation like $y = a + bx + \epsilon$ where y is the dependent variable, x is the independent variable, a is the intercept, b is the slope and ϵ is the error term. The equation for the mean value (excluding error term) can be written as $Y = X\beta$ where Y is a column vector of the n dependent variable values, β is the column vector $\begin{bmatrix} a \\ b \end{bmatrix}$ and the design matrix X is:

Intercept	x
1	x_1
1	x_2
1	x_3
1	...
1	x_n

where x_i is the value of x for the i$^{\text{th}}$ observation. In regression the $X\beta$ values are the mean for the dependent variable values ($y_i i$=1,n). For the example above the $X\beta$ values are $a + bx_1$, $a + bx_2$,..., $a + bx_n$ which are

the predicted mean for the dependent values ($y_i i$=1,n). However, with capture-recapture models the $X\beta$ values are used to compute real parameter values which are used in the model for the "dependent variable" which is the capture history.

Most real parameters θ in capture-recapture models are bounded like a survival probability which can have any real value between 0 and 1. $X\beta$ values are unbounded (i.e., can take any value from -infinity to +infinity) and to relate these to the θ parameters we use a link function which bounds the real parameters. A common link function is the logit link which bounds the real parameters between 0 and 1 for probabilities. It is used in logistic regression for the probability of a binomial/Bernoulli response variable. The logit link is defined such that $log(\theta/(1-\theta)) = \beta$ and the inverse logit link is $\theta = (1+exp(-\beta))^{-1}$. You may have seen the following alternative form for the inverse logit link $\theta = exp(\beta)/(1+exp(\beta))$. By dividing the numerator and denominator by $exp(\beta)$ you will get $(1+exp(-\beta))^{-1}$. This latter formulation is more numerically stable when x gets large and is used in the `plogis` function in R.

The general formula relating real parameters to beta parameters is $\theta = link^{-1}(X\beta)$ where $link^{-1}$ represents the inverse link function. Another useful link is the log link which restricts a parameter like abundance to be greater than 0. The log link is used in Poisson regression. Another useful link is the mlogit link which restricts a set of probabilities to sum to 1 which is used for multinomial probabilities. The mlogit link is used for transition probabilities in multistate models and in the entry probability parameter for the POPAN model. In MARK, the default link for probabilities is the sin link; however it cannot be used with general DMs such as those with numeric covariates or an intercept formulation. Thus, RMark uses the logit link by default for probabilities. I give an example showing the usefulness of the sin link in section 6.6.

Notice that the different types of parameters (ϕ and p) do not share columns of the design matrix. In general, the complete DM is constructed by creating a DM for each parameter type and then pasting together the sub-matrices and adding zeros such that rows corresponding to one type of parameter only contain 0 in the columns for another type of parameter as shown in the example below with 3 different types of parameters:

Sex	β_1	β_2	β_3	β_4	β_5
F	1	0	0	0	0
M	1	1	0	0	0
F	0	0	1	0	0
M	0	0	1	1	0
F	0	0	0	0	1
M	0	0	0	0	1

This does not mean that you cannot use the same covariate for different types of parameters. For example, you may want to use sex as a predictor of both survival and capture probability. The above example shows that each type of parameter can be sex-specific. The restriction does mean that the sex-effect variable cannot be restricted to have the same value for each type of parameter. When the parameter types are similar like p, capture probability and c, recapture probability in closed models it can be useful to share the column in the design matrix which is demonstrated in section 6.9.

As an aside, the negative log-likelihood that is minimized by MARK is $-lnL = \sum_{\omega} Pr(\omega|\beta)$ where ω is the set of observed capture histories and β is the vector of parameters. The structure of $Pr(\omega|\beta)$ depends on the type of capture-recapture model but you need not worry about that because MARK does all of that calculation for you and RMark will create the PIMs and DM for you from formulas. What you do need to understand is what the parameters represent, what models are relevant for each parameter for your data and how you construct formulas to represent those models.

We will get to that in due time but before we venture into an example, we need to consider 2 further complications to the PIMs and DMs. The first is factor variables which are referred to as groups in MARK terminology. You can think of these variables as something that splits your data into different strata or groups. We will use a single factor variable "sex" with values "Female" and "Male" but you can define groups based on more than one factor variable (e.g., sex and region). When you define a group variable the set of PIMs expands. Below is the set of PIMs for our example with Females and Males.

Female	Survival Interval		
Release Occasion	1 to 2	2 to 3	3 to 4
1	1	2	3
2		4	5
3			6

Male	Survival Interval		
Release Occasion	1 to 2	2 to 3	3 to 4
1	7	8	9
2		10	11
3			12

Female	Recapture Occasion		
Release Occasion	2	3	4
1	13	14	15
2		16	17
3			18

Male	Recapture Occasion		
Release Occasion	2	3	4
1	19	20	21
2		22	23
3			24

Now we have twice as many possible parameters with 2 groups. If we used group variables with 10 values then we would have 120 possible parameters. If you imagine a scenario with many more occasions and many groups, you can see how managing the parameters with a GUI could become unmanageable. With 24 real parameters we will have 24 rows in the DM but we would still only have 2 columns if we chose to have a model with constant survival and constant capture probability.

One final complication with the DM is an individual covariate. This is a numeric covariate (**not** a factor variable) which can be different for each animal. For the time being we will only consider an individual covariate that is treated as static throughout the course of the experiment. Time-varying individual covariates are discussed in section 6.8. We will use the weight of the animal at the time of its initial release. With the 2 groups, a DM for a model with constant capture probability and survival that is dependent on `weight` would look as follows with only the beginning and end of the DM shown:

Index	β_1	β_2	β_3
1	1	weight	0
2	1	weight	0
3	1	weight	0
4	1	weight	0
5	1	weight	0
...	...	...	...
19	0	0	1
20	0	0	1
21	0	0	1
22	0	0	1
23	0	0	1
24	0	0	1

The important thing to note here is that the DM contains the name of the individual covariate; whereas factor (group) variables are handled by expanding the set of PIMs. That affects what you get and what you can do with these covariates in RMark. With the above DM and the logit link, the formula for survival probability is $(1 + exp(-\beta_1 - \beta_2 \text{Weight}))^{-1}$ and the formula for capture probability is $(1 + exp(-\beta_3))^{-1}$.

4 Analyzing the Simple CJS Example with MARK

The simulated data for our simple example is in 2 files. The first is `example.inp` which is in the format required for MARK and we will use it with the MARK GUI. The second is `example.txt` which is in the format for RMark. The MARK format can be converted to the RMark format with the function `convert.inp`. Our example data contains 600 capture histories with 300 females and 300 males. The experiment has 4 occasions each separated by a unit time interval (e.g., one year). On each of the first 3 occasions 100 males and 100 females were released and then possibly captured on subsequent occasions. The `ch` field in both files is the capture history which is a character string of length 4 with values 0 and 1 (e.g., 1001). Each file also has the weight value for the animal. The column is named "weight" in `example.txt` but is unnamed and is the last field in `example.inp`. The file formats differ in two ways:

- For specification of groups (factor variables), the RMark format specifies the value(s) of the variables used for groups; whereas, with the MARK format, the groups are denoted by a value of the frequency field (number of animals with that capture history) in different columns with one column per group
- Fields in `example.txt` are tab-delimited whereas in `example.inp` the fields are space delimited and each line ends with a ;.

```
MARKData=read.table(file="example.inp",header=FALSE,
  colClasses=c("character",rep("numeric",2),"character"),
  col.names=c("ch","Females","Males","Weight"))
head(MARKData)

##     ch Females Males Weight
## 1 0011       1     0  2.96;
## 2 0010       1     0  3.09;
## 3 0010       1     0  3.14;
## 4 0010       1     0  3.23;
## 5 0011       1     0  3.47;
## 6 0011       1     0   3.5;

RMarkData=read.table(file="example.txt",header=TRUE,
  colClasses=c("character","factor","numeric"))
head(RMarkData)

##     ch    sex weight
## 1 0011 Female   2.96
## 2 0010 Female   3.09
## 3 0010 Female   3.14
## 4 0010 Female   3.23
## 5 0011 Female   3.47
## 6 0011 Female   3.50
```

We will start by using MARK so you can understand what RMark must do to create models for MARK and also to compare the processes for analyzing data. The steps you'll take with MARK are as follows:

1. Start MARK interface program and select File/New.
2. Complete the form by:
   (a) selecting the Data Type which for this example is Live Recaptures (CJS)
   (b) Provide a title for the analysis (optional)
   (c) Select the input file (`example.inp`); a results file name is automatically provided
   (d) Specify the number of encounter occasions which is 4 for this example. We do not need to set the time interval lengths between occasions because they are all 1 for our example.

example data sets that accompany the MARK book are provided either as examples with RMark (`?dipper`) or just the data sets (`?convert.inp`). You can learn and become more comfortable with RMark by comparing results in the book with your attempts to construct the same model in RMark.

5 Analyzing the Simple CJS Example with RMark

Everything we did with MARK needs to be accomplished with RMark but I think you'll find it much easier once you learn the R syntax. We need to provide the data, assign the data and model attributes, describe and run models and examine the output.

5.1 Providing the data

You can use any of the R functions like `read.table` or `read.delim` to read in the data. Alternatively you can use the functions `convert.inp` if you have an existing file in MARK format or `import.chdata` (a function in the RMark package) if the data are in RMark format. First I'll describe the RMark data format which is fairly simple for almost all of the models. I will not address nest survival data which is completely different but you can see `?killdeer` or `?mallard` for examples. For all of the remainder of the models, the only required field in the data is the capture history which must be named `ch` and it must be a character variable. It is important to realize that the default behavior of R is to read character data into dataframes as factor variables which are numeric variables with a character label. It is essential that the capture history be read in as a character variable. The function `import.chdata` forces the first field (`ch`) to be character. However, you can do the same by using the `colClasses` argument in `read.table` or `read.delim`. The only other field which has a fixed name is `freq` which is the frequency of each capture history. The `freq` field is not needed if each capture history represents a single individual (`freq`=1). Losses on capture (caught but not subsequently released) are represented by a negative freq value. The remainder of any fields in the data can have any name or type. If they are factor variables they can be used to define groups in the data (e.g., `sex`) or if they are numeric variables they can be used as individual covariates (e.g. `weight`). Below is code that shows how our simple example data can either be read in with `read.table` or `import.chdata`. Note that before we try to use any functions in RMark we must load the package from the library with `library(RMark)`.

```
library(RMark)

## Loading required package:  snowfall
## Loading required package:  snow
## This is RMark 2.1.13

# read in the data; header=TRUE means first row is column names;
# colClasses specify that the fields are of type character (ch),
# factor (sex) and numeric (weight)
RMarkData=read.table(file="example.txt",header=TRUE,
             colClasses=c("character","factor","numeric"))
head(RMarkData)

##     ch    sex weight
## 1 0011 Female   2.96
## 2 0010 Female   3.09
## 3 0010 Female   3.14
## 4 0010 Female   3.23
## 5 0011 Female   3.47
## 6 0011 Female   3.50

# use import.chdata to read in the data; ch is first field and is character
# field.types f and n are for sex and weight fields
```

```
# Here I'll use the name df so typing will be easier later
df=import.chdata("example.txt",field.types=c("f","n"))
head(df)

##     ch    sex weight
## 1 0011 Female   2.96
## 2 0010 Female   3.09
## 3 0010 Female   3.14
## 4 0010 Female   3.23
## 5 0011 Female   3.47
## 6 0011 Female   3.50
```

You can use the `str` and `summary` R functions to make sure your data have been read in properly.

```
str(df)

## 'data.frame': 600 obs. of  3 variables:
##  $ ch    : chr  "0011" "0010" "0010" "0010" ...
##  $ sex   : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ weight: num  2.96 3.09 3.14 3.23 3.47 3.5 3.6 3.61 3.65 3.68 ...

summary(df)

##       ch                sex          weight
##  Length:600         Female:300   Min.   :1.700
##  Class :character   Male  :300   1st Qu.:4.317
##  Mode  :character                Median :5.020
##                                  Mean   :5.008
##                                  3rd Qu.:5.652
##                                  Max.   :8.340
```

5.2 Describing data and model attributes

As with the first screen in the MARK interface, we need to specify the type of model for the data and some attributes for the data and model. This is accomplished with the RMark function `process.data` which uses the `data` and `model` arguments along with other optional arguments to create a list with the data and its attributes. To see all of the possible arguments, use ?`process.data` to see the help file. For our example we only need to specify three arguments as shown below:

```
dp=process.data(df,model="CJS",groups="sex")
str(dp)

## List of 15
##  $ data           :'data.frame': 600 obs. of  4 variables:
##   ..$ ch    : chr [1:600] "0011" "0010" "0010" "0010" ...
##   ..$ sex   : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ weight: num [1:600] 2.96 3.09 3.14 3.23 3.47 3.5 3.6 3.61 3.65 3.68 ...
##   ..$ group : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ model          : chr "CJS"
##  $ mixtures       : num 1
##  $ freq           :'data.frame': 600 obs. of  2 variables:
##   ..$ sexFemale: num [1:600] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ sexMale  : num [1:600] 0 0 0 0 0 0 0 0 0 0 ...
##  $ nocc           : num 4
```

```
##  $ nocc.secondary  : NULL
##  $ time.intervals  : num [1:3] 1 1 1
##  $ begin.time      : num 1
##  $ age.unit        : num 1
##  $ initial.ages    : num [1:2] 0 0
##  $ group.covariates:'data.frame': 2 obs. of  1 variable:
##   ..$ sex: Factor w/ 2 levels "Female","Male": 1 2
##  $ nstrata         : num 1
##  $ strata.labels   : NULL
##  $ counts          : NULL
##  $ reverse         : logi FALSE
```

The list of possible values for model are given in MarkModels.pdf. Notice that we didn't need to specify the number of capture occasions, number of groups or group labels or the individual covariates and their names. That is done automatically by the code. If you had different time intervals or animals of different ages or a multistate model then there are other arguments that would be used to define those attributes.

The next step is to create design data which doesn't have an equivalent step in the MARK interface but it is essential to the RMark approach to defining a model from formulas. Design data are data that are attached to the parameters in the model and thus are specific to the type of model (e.g., CJS, Closed etc) and its attributes. The following creates the design data for our example:

```
ddl=make.design.data(dp)
str(ddl)
```

```
## List of 3
##  $ Phi     :'data.frame': 12 obs. of  11 variables:
##   ..$ par.index  : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ model.index: num [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ group      : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ cohort     : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 3 1 1 1 2 ...
##   ..$ age        : Factor w/ 3 levels "0","1","2": 1 2 3 1 2 1 1 2 3 1 ...
##   ..$ time       : Factor w/ 3 levels "1","2","3": 1 2 3 2 3 3 1 2 3 2 ...
##   ..$ occ.cohort : num [1:12] 1 1 1 2 2 3 1 1 1 2 ...
##   ..$ Cohort     : num [1:12] 0 0 0 1 1 2 0 0 0 1 ...
##   ..$ Age        : num [1:12] 0 1 2 0 1 0 0 1 2 0 ...
##   ..$ Time       : num [1:12] 0 1 2 1 2 2 0 1 2 1 ...
##   ..$ sex        : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##  $ p       :'data.frame': 12 obs. of  11 variables:
##   ..$ par.index  : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ model.index: num [1:12] 13 14 15 16 17 18 19 20 21 22 ...
##   ..$ group      : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ cohort     : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 3 1 1 1 2 ...
##   ..$ age        : Factor w/ 3 levels "1","2","3": 1 2 3 1 2 1 1 2 3 1 ...
##   ..$ time       : Factor w/ 3 levels "2","3","4": 1 2 3 2 3 3 1 2 3 2 ...
##   ..$ occ.cohort : num [1:12] 1 1 1 2 2 3 1 1 1 2 ...
##   ..$ Cohort     : num [1:12] 0 0 0 1 1 2 0 0 0 1 ...
##   ..$ Age        : num [1:12] 1 2 3 1 2 1 1 2 3 1 ...
##   ..$ Time       : num [1:12] 0 1 2 1 2 2 0 1 2 1 ...
##   ..$ sex        : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##  $ pimtypes:List of 2
##   ..$ Phi:List of 1
##   .. ..$ pim.type: chr "all"
##   ..$ p  :List of 1
##   .. ..$ pim.type: chr "all"
```

You can use any name but I tend to use `ddl` which stands for design data list because as shown above, the function returns a list with a dataframe for each parameter and then a list of `pimtypes`. The parameters for the CJS model are `Phi` and `p`, so the `ddl` contains a design dataframe `ddl$Phi` and `ddl$p`.

So what are these dataframes? The dataframes assign "data" to each real parameter in the model. The "data" that is assigned depends on the type of model (e.g., CJS vs Closed) as well as the parameter in the model. For the CJS model, we can define the following "data" for each parameter index:

1. the group which is either Female or Male
2. the release cohort occasion which is constant in a row of the PIM
3. the occasion (time) which is constant in a column of the PIM
4. the age or time since released which is constant along a diagonal of the PIM.

Each one of those fields is represented as a factor variable (`group, cohort, age, time`) and some as a numeric variable (`Cohort, Age, Time`) which enable fitting trend models. Using a capital letter is a convention from MARK where p(t) means capture probability varies separately for each time and p(T) is a trend with only an intercept and slope. The values of `cohort`, `age`, and `time` and their numeric counterparts depend on the value of `begin.time` and `time.intervals` specified for the model. However the variable `occ.cohort` is always the row number in the PIM. Note that the value of `time` and `age` differ for Phi and p because survival (Phi) refers to the survival in the interval between occasions and its value is for the `time` and `age` at the beginning of the interval; whereas, capture probability refers to a point in time. However, the values of the numeric equivalents like `Time` are the same for both Phi and p because `Time` is always given an origin of 0 which typically makes the intercept more useful. For example, if `begin.time=1990` and if `Time` started at 1990, the intercept for a trend model would be the value extrapolated back to the year 0 (1990 years previous) which would not be very useful. By using the origin 0 for `Time`, in any trend model the intercept is for `begin.time` for an interval parameter like Phi and `begin.time+time.intervals[1]` for p. Additionally it also avoids numerical issues when the value of `time` is large. For the same reasons, `Cohort` and `Age` are treated the same.

In this example, `group` and `sex` have the same values but if groups were defined based on more than one factor variable, then the `group` field would be the concatenated value of each combination (e.g., `MaleArea1`) and each factor variable would be included separately (e.g., `sex=MALE,Area=1`). These are the default data that are created specifically for the type of model specified in `process.data`. However, you can add any additional design data fields that you want and I'll show examples of this later.

The `pimtype` is the default value of `all` which means all-different indices. You can set the `pimtype` to either `time` or `constant` but that restricts the type of models that can be constructed and those values are only used rarely to restrict the number of parameters.

Each row in the design dataframes has 2 indices. The `par.index` is the index for each parameter within that parameter type; whereas `model.index` is the unique index across all of the parameters in the model. The `model.index` values are the unique indices across all parameters in the model. They match the values in the PIMs that we defined earlier in MARK. For the first parameter the value of `par.index` and `model.index` are the same because the `model.index` values are assigned sequentially starting with the first parameter. If you see code in examples (particularly fixing real parameters), help files and older documentation for RMark that looks something like `as.numeric(row.names(ddl$...))`, that code is no longer needed because it was constructing the `par.index` value on the fly and it is now contained in the design data. In other functions within RMark you will see the label `all.diff.index` which is the same as `model.index`. Unfortunately I have not been consistent with the use of `par.index` so be aware that its meaning as a label can change. You will see another index labeled `vcv.index` which is the index in the variance-covariance matrix if one is provided from the function.

5.3 Describing and running a model

Running a simple model with default formulas is quite simple. We call the `mark` function with the arguments `dp` and `ddl` and assign it to an object with name `model` (or any name). A brief summary of the model is printed by default.

```
model=mark(dp,ddl)

##
## Output summary for CJS model
## Name : Phi(~1)p(~1)
##
## Npar :  2
## -2lnL:  1525.035
## AICc :  1529.047
##
## Beta
##                  estimate        se       lcl      ucl
## Phi:(Intercept) 2.9193645 0.2651802 2.3996112 3.439118
## p:(Intercept)   0.9221261 0.0898523 0.7460156 1.098237
##
##
## Real Parameter Phi
## Group:sexFemale
##           1         2         3
## 1 0.9487954 0.9487954 0.9487954
## 2           0.9487954 0.9487954
## 3                     0.9487954
##
## Group:sexMale
##           1         2         3
## 1 0.9487954 0.9487954 0.9487954
## 2           0.9487954 0.9487954
## 3                     0.9487954
##
##
## Real Parameter p
## Group:sexFemale
##           2         3         4
## 1 0.7154751 0.7154751 0.7154751
## 2           0.7154751 0.7154751
## 3                     0.7154751
##
## Group:sexMale
##           2         3         4
## 1 0.7154751 0.7154751 0.7154751
## 2           0.7154751 0.7154751
## 3                     0.7154751
```

The default formulas for this model are constant survival and capture probability. The output shows the formulas used Phi(~1)p(~1) where ~1 is the intercept (constant) model. Then it shows the number of parameters (2), the -2*log likelihood and AICc for model selection. That is followed by the estimates, standard errors and confidence intervals for the beta parameters (values on the logit scale) and finally by the real parameter values in triangular PIM format for Females and Males for Phi and p. This is a constant model so the values for Phi are the same for each sex and for all indices and likewise for p. You can compute the real parameter value for Phi directly from plogis(2.9194)=0.9488.

So what did the `mark` function do? It constructed an input file for MARK using the same structure we saw earlier with the Save Structure in the MARK interface. The `mark` function then ran mark.exe and extracted the output from the files with extensions `.out` and `.vcv` and the result was stored in a list which was stored in the object `model`. The structure of the `model` object is shown below.

```
str(model)
```

```
## List of 25
##  $ data            :List of 15
##   ..$ data           :'data.frame': 600 obs. of  4 variables:
##   .. ..$ ch    : chr [1:600] "0011" "0010" "0010" "0010" ...
##   .. ..$ sex   : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ weight: num [1:600] 2.96 3.09 3.14 3.23 3.47 3.5 3.6 3.61 3.65 3.68 ...
##   .. ..$ group : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ model          : chr "CJS"
##   ..$ mixtures       : num 1
##   ..$ freq           :'data.frame': 600 obs. of  2 variables:
##   .. ..$ sexFemale: num [1:600] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ sexMale  : num [1:600] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ nocc           : num 4
##   ..$ nocc.secondary  : NULL
##   ..$ time.intervals  : num [1:3] 1 1 1
##   ..$ begin.time     : num 1
##   ..$ age.unit       : num 1
##   ..$ initial.ages    : num [1:2] 0 0
##   ..$ group.covariates:'data.frame': 2 obs. of  1 variable:
##   .. ..$ sex: Factor w/ 2 levels "Female","Male": 1 2
##   ..$ nstrata        : num 1
##   ..$ strata.labels   : NULL
##   ..$ counts         : NULL
##   ..$ reverse        : logi FALSE
##  $ model           : chr "CJS"
##  $ title           : chr ""
##  $ model.name      : chr "Phi(~1)p(~1)"
##  $ links           : chr "logit"
##  $ mixtures        : NULL
##  $ call            : language mark(data = dp, ddl = ddl)
##  $ parameters      :List of 2
##   ..$ Phi:List of 7
##   .. ..$ begin   : num 0
##   .. ..$ num     : num -1
##   .. ..$ default : num 1
##   .. ..$ type    : chr "Triang"
##   .. ..$ pim.type: chr "all"
##   .. ..$ link    : chr "logit"
##   .. ..$ formula :Class 'formula' length 2 ~1
##   .. .. .. .. ..- attr(*, ".Environment")=<environment: 0x07dbb5bc>
##   ..$ p  :List of 7
##   .. ..$ begin   : num 1
##   .. ..$ num     : num -1
##   .. ..$ default : num 0
##   .. ..$ type    : chr "Triang"
##   .. ..$ pim.type: chr "all"
##   .. ..$ link    : chr "logit"
##   .. ..$ formula :Class 'formula' length 2 ~1
##   .. .. .. .. ..- attr(*, ".Environment")=<environment: 0x07dbb5bc>
##  $ time.intervals  : num [1:3] 1 1 1
##  $ number.of.groups: int 2
##  $ group.labels    : chr [1:2] "sexFemale" "sexMale"
```

```
##  $ nocc            : num 4
##  $ begin.time      : num 1
##  $ covariates      : NULL
##  $ fixed           : NULL
##  $ design.matrix   : chr [1:2, 1:2] "1" "0" "0" "1"
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "Phi gFemale c1 a0 t1" "p gFemale c1 a1 t2"
##   .. ..$ : chr [1:2] "Phi:(Intercept)" "p:(Intercept)"
##  $ pims            :List of 2
##   ..$ Phi:List of 2
##   .. ..$ :List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 1 0 0 2 4 0 3 5 6
##   .. .. ..$ group: int 1
##   .. ..$ :List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 7 0 0 8 10 0 9 11 12
##   .. .. ..$ group: int 2
##   ..$ p  :List of 2
##   .. ..$ :List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 13 0 0 14 16 0 15 17 18
##   .. .. ..$ group: int 1
##   .. ..$ :List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 19 0 0 20 22 0 21 23 24
##   .. .. ..$ group: int 2
##  $ design.data     :List of 3
##   ..$ Phi     :'data.frame': 12 obs. of  11 variables:
##   .. ..$ par.index  : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ model.index: num [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ group      : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##   .. ..$ cohort     : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 3 1 1 1 2 ...
##   .. ..$ age        : Factor w/ 3 levels "0","1","2": 1 2 3 1 2 1 1 2 3 1 ...
##   .. ..$ time       : Factor w/ 3 levels "1","2","3": 1 2 3 2 3 3 1 2 3 2 ...
##   .. ..$ occ.cohort : num [1:12] 1 1 1 2 2 3 1 1 1 2 ...
##   .. ..$ Cohort     : num [1:12] 0 0 0 1 1 2 0 0 0 1 ...
##   .. ..$ Age        : num [1:12] 0 1 2 0 1 0 0 1 2 0 ...
##   .. ..$ Time       : num [1:12] 0 1 2 1 2 2 0 1 2 1 ...
##   .. ..$ sex        : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ p       :'data.frame': 12 obs. of  11 variables:
##   .. ..$ par.index  : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ model.index: num [1:12] 13 14 15 16 17 18 19 20 21 22 ...
##   .. ..$ group      : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##   .. ..$ cohort     : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 3 1 1 1 2 ...
##   .. ..$ age        : Factor w/ 3 levels "1","2","3": 1 2 3 1 2 1 1 2 3 1 ...
##   .. ..$ time       : Factor w/ 3 levels "2","3","4": 1 2 3 2 3 3 1 2 3 2 ...
##   .. ..$ occ.cohort : num [1:12] 1 1 1 2 2 3 1 1 1 2 ...
##   .. ..$ Cohort     : num [1:12] 0 0 0 1 1 2 0 0 0 1 ...
##   .. ..$ Age        : num [1:12] 1 2 3 1 2 1 1 2 3 1 ...
##   .. ..$ Time       : num [1:12] 0 1 2 1 2 2 0 1 2 1 ...
##   .. ..$ sex        : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ pimtypes:List of 2
##   .. ..$ Phi:List of 1
##   .. .. ..$ pim.type: chr "all"
##   .. ..$ p  :List of 1
##   .. .. ..$ pim.type: chr "all"
```

```
##  $ strata.labels   : NULL
##  $ mlogit.list     :List of 2
##   ..$ structure: NULL
##   ..$ ncol     : num 1
##  $ profile.int     : logi FALSE
##  $ simplify        :List of 2
##   ..$ pim.translation: int [1:24] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ real.labels    : chr [1:24] "Phi gFemale c1 a0 t1" "Phi gFemale c1 a1 t2" "Phi gFemale c1 a2 t3
##  $ model.parameters: list()
##  $ results         :List of 14
##   ..$ lnl             : num 1525
##   ..$ deviance        : num 153
##   ..$ deviance.df     : num 20
##   ..$ npar            : num 2
##   ..$ n               : int 1012
##   ..$ AICc            : num 1529
##   ..$ beta            :'data.frame': 2 obs. of  4 variables:
##   .. ..$ estimate: num [1:2] 2.919 0.922
##   .. ..$ se      : num [1:2] 0.2652 0.0899
##   .. ..$ lcl     : num [1:2] 2.4 0.746
##   .. ..$ ucl     : num [1:2] 3.44 1.1
##   ..$ real            :'data.frame': 2 obs. of  6 variables:
##   .. ..$ estimate: num [1:2] 0.949 0.715
##   .. ..$ se      : num [1:2] 0.0129 0.0183
##   .. ..$ lcl     : num [1:2] 0.917 0.678
##   .. ..$ ucl     : num [1:2] 0.969 0.75
##   .. ..$ fixed   : Factor w/ 1 level "     ": 1 1
##   .. ..$ note    : Factor w/ 1 level "       ": 1 1
##   ..$ beta.vcv        : num [1:2, 1:2] 0.07032 -0.0134 -0.0134 0.00807
##   ..$ derived         : NULL
##   ..$ derived.vcv     : NULL
##   ..$ covariate.values: NULL
##   ..$ singular        : NULL
##   ..$ real.vcv        : NULL
##  $ output          : chr "mark004"
##  - attr(*, "class")= chr [1:2] "mark" "CJS"
```

The model object is self-contained with all of the data and attributes including:

- `data`: the processed data and its attributes
- `parameters`: which is the structure of the parameters used in the model
- `design.matrix`: the design matrix for the model
- `design.data`: the design data used to construct the model
- `pims`: the model PIMs
- `results`: a list containing all of the results from MARK including the beta and real estimates including standard errors and confidence intervals and the variance-covariance matrix.
- `output`: the base name of the input and output files (in this case it is mark004 with extensions `.inp, .out, .vcv, .res`); these files are kept in the working directory but are typically only needed if there was an error in constructing the model. Viewing the output file can actually be confusing which I'll explain in section 6.10. However, you can easily view the output by typing `model` (or whatever you

named the result) and the output will be opened in a notepad window. If an error occurred then model would not have been constructed and you'll have to open the *.out file directly.

5.4 Examining the model output

You can extract the estimates directly from the results, but it is best to use the RMark functions `coef` and `summary` to extract the parameter estimates. The reason is described in section 10. The `coef` function extracts the beta estimates:

```
coef(model)
```

```
##                  estimate        se       lcl      ucl
## Phi:(Intercept) 2.9193645 0.2651802 2.3996112 3.439118
## p:(Intercept)   0.9221261 0.0898523 0.7460156 1.098237
```

The `summary` function produces a list with various results. By default the summary results are printed as with any R object:

```
summary(model)
```

```
## Output summary for CJS model
## Name : Phi(~1)p(~1)
##
## Npar :  2
## -2lnL:  1525.035
## AICc :  1529.047
##
## Beta
##                  estimate        se       lcl      ucl
## Phi:(Intercept) 2.9193645 0.2651802 2.3996112 3.439118
## p:(Intercept)   0.9221261 0.0898523 0.7460156 1.098237
##
##
## Real Parameter Phi
## Group:sexFemale
##           1         2         3
## 1 0.9487954 0.9487954 0.9487954
## 2           0.9487954 0.9487954
## 3                     0.9487954
##
## Group:sexMale
##           1         2         3
## 1 0.9487954 0.9487954 0.9487954
## 2           0.9487954 0.9487954
## 3                     0.9487954
##
##
## Real Parameter p
## Group:sexFemale
##           2         3         4
## 1 0.7154751 0.7154751 0.7154751
## 2           0.7154751 0.7154751
## 3                     0.7154751
##
```

```
## Group:sexMale
##           2         3         4
## 1 0.7154751 0.7154751 0.7154751
## 2           0.7154751 0.7154751
## 3                     0.7154751
```

However, you can extract any of the `summary` function components which are shown with the `str` function:

```
str(summary(model))
```

```
## List of 10
##  $ model     : chr "CJS"
##  $ title     : chr ""
##  $ model.name: chr "Phi(~1)p(~1)"
##  $ model.call: language mark(data = dp, ddl = ddl)
##  $ npar      : num 2
##  $ lnl       : num 1525
##  $ AICc      : num 1529
##  $ beta      :'data.frame': 2 obs. of  4 variables:
##   ..$ estimate: num [1:2] 2.919 0.922
##   ..$ se      : num [1:2] 0.2652 0.0899
##   ..$ lcl     : num [1:2] 2.4 0.746
##   ..$ ucl     : num [1:2] 3.44 1.1
##  $ reals     :List of 2
##   ..$ Phi:List of 2
##   .. ..$ Group:sexFemale:List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 0.949 NA NA 0.949 0.949 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:3] "1" "2" "3"
##   .. .. .. .. ..$ : chr [1:3] "1" "2" "3"
##   .. .. ..$ group: int 1
##   .. ..$ Group:sexMale  :List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 0.949 NA NA 0.949 0.949 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:3] "1" "2" "3"
##   .. .. .. .. ..$ : chr [1:3] "1" "2" "3"
##   .. .. ..$ group: int 2
##   ..$ p  :List of 2
##   .. ..$ Group:sexFemale:List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 0.715 NA NA 0.715 0.715 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:3] "1" "2" "3"
##   .. .. .. .. ..$ : chr [1:3] "2" "3" "4"
##   .. .. ..$ group: int 1
##   .. ..$ Group:sexMale  :List of 2
##   .. .. ..$ pim  : num [1:3, 1:3] 0.715 NA NA 0.715 0.715 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:3] "1" "2" "3"
##   .. .. .. .. ..$ : chr [1:3] "2" "3" "4"
##   .. .. ..$ group: int 2
##  $ brief     : logi FALSE
##  - attr(*, "class")= chr "summary.mark"
```

```
summary(model)$AICc
```

```
## [1] 1529.047
```

6 Design Data and Formulas

So far we have only fit a very simple default model. Now we will discuss how you can fit just about any model that you can imagine. You can certainly do the same with the MARK interface but here we introduce the use of formulas so you can avoid creating DMs manually. By this point you should understand the relationship between PIMs and DMs. If not, go re-read the previous sections. I'll start by describing the R function `model.matrix` which is an integral component of RMark.

6.1 What is `model.matrix`?

The R function `model.matrix` creates a design matrix from a formula applied to data. You do not need to use `model.matrix` to create a model with RMark because it does that for you. However, unless you are using individual covariates, you can try out your formula with `model.matrix` to help you understand the DM that your formula will create. I will use `model.matrix` here to explain the link between formulas and design matrices, so you have a better understanding of what RMark is doing. Many other modeling packages also use `model.matrix` so it will help your understanding in general with specifying models and interpreting the coefficients. I have found that folks with only one or two statistics courses generally lack this understanding and MARK is often their first encounter with DMs. Even though RMark creates the DM for you, you need to understand what you are doing.

Before jumping into use of `model.matrix` with RMark, I'll give a simple example to describe the important difference between factor and numeric variables. The focus here is the independent predictor variables and not the dependent variable. For more details see LinearModels.pdf in the RMark documentation archive. Consider the following simple set of data:

```
data=expand.grid(sex=c("Female","Male"),age=c("0","1","2","3"))
set.seed(98215)
data$height=rnorm(8,10,3)
data
```

```
##      sex age    height
## 1 Female   0 12.273859
## 2   Male   0  9.150697
## 3 Female   1  8.179441
## 4   Male   1  8.141125
## 5 Female   2  8.645582
## 6   Male   2 16.343100
## 7 Female   3 10.784280
## 8   Male   3 10.101928
```

```
str(data)
```

```
## 'data.frame': 8 obs. of  3 variables:
##  $ sex   : Factor w/ 2 levels "Female","Male": 1 2 1 2 1 2 1 2
##  $ age   : Factor w/ 4 levels "0","1","2","3": 1 1 2 2 3 3 4 4
##  $ height: num  12.27 9.15 8.18 8.14 8.65 ...
##  - attr(*, "out.attrs")=List of 2
##   ..$ dim     : Named int  2 4
##   .. ..- attr(*, "names")= chr  "sex" "age"
##   ..$ dimnames:List of 2
##   .. ..$ sex: chr  "sex=Female" "sex=Male"
##   .. ..$ age: chr  "age=0" "age=1" "age=2" "age=3"
```

Notice that `sex` and `age` are factor variables and `height` is a numeric variable. The variable `sex` has 2 levels and the variable `age` has 4 levels. Factor variables define a subset of the data and when a factor with k levels is used in a formula with `model.matrix` and treatment contrast (see `?options` and `?contrasts` in R) $k-1$

dummy variables are created in addition to the intercept. Thus if you use the formulas `~sex` and `~age` with `model.matrix` with data they create design matrices with 2 (intercept + 1) and 4 (intercept +3) columns respectively:

```
model.matrix(~sex,data)

##   (Intercept) sexMale
## 1           1       0
## 2           1       1
## 3           1       0
## 4           1       1
## 5           1       0
## 6           1       1
## 7           1       0
## 8           1       1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$sex
## [1] "contr.treatment"

model.matrix(~age,data)

##   (Intercept) age1 age2 age3
## 1           1    0    0    0
## 2           1    0    0    0
## 3           1    1    0    0
## 4           1    1    0    0
## 5           1    0    1    0
## 6           1    0    1    0
## 7           1    0    0    1
## 8           1    0    0    1
## attr(,"assign")
## [1] 0 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$age
## [1] "contr.treatment"
```

Notice that the intercept represents the first level of the factor variable which is Female for `sex` and 0 for `age`. If you remove the intercept with a factor variable `model.matrix` still creates 2 columns in the DM but there is now a separate column for each factor level rather than specifying k-1 levels relative to an intercept:

```
model.matrix(~-1+sex,data)

##   sexFemale sexMale
## 1         1       0
## 2         0       1
## 3         1       0
## 4         0       1
## 5         1       0
## 6         0       1
## 7         1       0
## 8         0       1
## attr(,"assign")
## [1] 1 1
```

```
## attr(,"contrasts")
## attr(,"contrasts")$sex
## [1] "contr.treatment"

model.matrix(~-1+age,data)

##   age0 age1 age2 age3
## 1    1    0    0    0
## 2    1    0    0    0
## 3    0    1    0    0
## 4    0    1    0    0
## 5    0    0    1    0
## 6    0    0    1    0
## 7    0    0    0    1
## 8    0    0    0    1
## attr(,"assign")
## [1] 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$age
## [1] "contr.treatment"
```

Now when you specify an additive formula with 2 factor variables, each factor provides $k-1$ dummy variables plus the intercept, so `~sex+age` creates 5 (1+1+3) columns:

```
model.matrix(~sex+age,data)

##   (Intercept) sexMale age1 age2 age3
## 1           1       0    0    0    0
## 2           1       1    0    0    0
## 3           1       0    1    0    0
## 4           1       1    1    0    0
## 5           1       0    0    1    0
## 6           1       1    0    1    0
## 7           1       0    0    0    1
## 8           1       1    0    0    1
## attr(,"assign")
## [1] 0 1 2 2 2
## attr(,"contrasts")
## attr(,"contrasts")$sex
## [1] "contr.treatment"
##
## attr(,"contrasts")$age
## [1] "contr.treatment"
```

If you were to remove the intercept in a model with additive factor variables, the first factor variable has separate columns but they represent the first level of the second factor variable as shown below:

```
model.matrix(~-1+sex+age,data)

##   sexFemale sexMale age1 age2 age3
## 1         1       0    0    0    0
## 2         0       1    0    0    0
## 3         1       0    1    0    0
## 4         0       1    1    0    0
## 5         1       0    0    1    0
```

We could also create a sex-specific model for survival and create a complete DM for the 24 real parameters in the model with:

```
pdm=model.matrix(~sex,ddl$p)
Phidm=model.matrix(~sex,ddl$Phi)
dm=cbind(Phidm,matrix(0,nrow=12,ncol=2))
dm=rbind(dm,cbind(matrix(0,nrow=12,ncol=2),pdm))
colnames(dm)=c(paste("Phi:",colnames(Phidm),sep=""),paste("p:",colnames(pdm),sep=""))
rownames(dm)=1:nrow(dm)
dm
```

```
##    Phi:(Intercept) Phi:sexMale p:(Intercept) p:sexMale
## 1                1           0             0         0
## 2                1           0             0         0
## 3                1           0             0         0
## 4                1           0             0         0
## 5                1           0             0         0
## 6                1           0             0         0
## 7                1           1             0         0
## 8                1           1             0         0
## 9                1           1             0         0
## 10               1           1             0         0
## 11               1           1             0         0
## 12               1           1             0         0
## 13               0           0             1         0
## 14               0           0             1         0
## 15               0           0             1         0
## 16               0           0             1         0
## 17               0           0             1         0
## 18               0           0             1         0
## 19               0           0             1         1
## 20               0           0             1         1
## 21               0           0             1         1
## 22               0           0             1         1
## 23               0           0             1         1
## 24               0           0             1         1
```

What is done in the code above is similar to what RMark does in in the function `make.mark.model` which is called from the `mark` function to create the model input for MARK. See section C.4 in Appendix C of the Cooch and White book for a more detailed discussion of what the `mark` function does.

6.2 Specifying RMark models with formulas

So, let's learn how to specify models with formulas for RMark. I'll use a specific naming convention that will be used later on in section 8. To create the above model with sex-specific survival and capture probability we create 2 lists with each containing a single element named `formula` and in this case the lists are identical but are stored in separate objects:

```
Phi.sex=list(formula=~sex)
p.sex=list(formula=~sex)
```

Now we will use those values to create and run the model with our example data using the `mark` function by specifying the argument `model.parameters` which is a list containing each parameter list (a list of lists):

```
sex.model=mark(dp,ddl,model.parameters=list(Phi=Phi.sex,p=p.sex))

##
## Output summary for CJS model
## Name : Phi(~sex)p(~sex)
##
## Npar :  4
## -2lnL:  1515.939
## AICc :  1523.979
##
## Beta
##                  estimate        se        lcl       ucl
## Phi:(Intercept) 2.4135006 0.2895129  1.8460553 2.9809460
## Phi:sexMale     1.2704264 0.6632554 -0.0295542 2.5704069
## p:(Intercept)   0.8914318 0.1361638  0.6245508 1.1583129
## p:sexMale       0.0721623 0.1809107 -0.2824227 0.4267472
##
##
## Real Parameter Phi
## Group:sexFemale
##          1        2        3
## 1 0.917851 0.917851 0.917851
## 2          0.917851 0.917851
## 3                   0.917851
##
## Group:sexMale
##           1         2         3
## 1 0.9754916 0.9754916 0.9754916
## 2           0.9754916 0.9754916
## 3                     0.9754916
##
##
## Real Parameter p
## Group:sexFemale
##           2         3         4
## 1 0.7091856 0.7091856 0.7091856
## 2           0.7091856 0.7091856
## 3                     0.7091856
##
## Group:sexMale
##           2         3         4
## 1 0.7238408 0.7238408 0.7238408
## 2           0.7238408 0.7238408
## 3                     0.7238408

# Note in this case we could have done the same thing with:
# sex=list(formula=~sex)
# sex.model=mark(dp,ddl,model.parameters=list(Phi=sex,p=sex))
# But usually the models will differ and the reason for different
# specifications will become clear later.
```

Now we have 4 beta parameters and the real parameters in PIM format show the differences for males and females.

```
##           1         2         3
## 1 0.9096382 0.9096382 0.9096382
## 2           0.9096382 0.9096382
## 3                     0.9096382
##
## Group:sexMale
##           1         2         3
## 1 0.9096382 0.9096382 0.9096382
## 2           0.9096382 0.9096382
## 3                     0.9096382
##
##
## Real Parameter p
## Group:sexFemale
##           2 3       4
## 1 0.4505767 1 0.732892
## 2           1 0.732892
## 3             0.732892
##
## Group:sexMale
##           2 3       4
## 1 0.4505767 1 0.732892
## 2           1 0.732892
## 3             0.732892
```

While this is still a valid approach to fix the parameters for specific models, if you are going to fix the real parameter values for all models it is cleaner and easier to add the `fix` field to the design data as described above.

8 Scripting and Model Selection

So far I have only shown examples of running a single model. Next I'll describe how to set up a function to analyze a set of models for your analysis. But before I get there it is important to keep in mind the following points:

- Too many models can be a bad thing! Beware of paralysis of analysis!
- Too much data dredging can lead you to a nonsensical "best" model.
- A nonsensical "best" model is still nonsensical and not useful!

Because RMark makes it fairly easy to construct models, it is easy to get carried away. With models containing many parameters, it is fairly easy to specify several thousand models if you don't do some critical thinking.

You may have already discovered that it is quite easy to make mistakes while you are typing in the R console by forgetting a comma or parenthesis here and there. Also, it may be clear that if you created many models you would have to devise a bunch of different names for the model object and after awhile the formulas would clutter your workspace to the point of not being able to keep it all organized. As I developed RMark this became obvious to me so I produced code to help organize the workflow of building models. The first step in organizing any workflow in R is to recognize the value of scripts and functions and to use them even with trivial projects. First of all what is a script? A script is simply a text file of R code that you can use in R with either the R script editor (File/New script) or with a development environment like Rstudio. With an external environment like Rstudio you get:

- syntax checking and auto-completion; matching braces and parentheses

- built-in help for R
- easily running all or part of the script in the R console from the script editor window

With scripts you can add comments. Comments can be a big time saver if you have to pick up an analysis 6 months after you did it when some reviewer asks a question about the specifics of the analysis you did for your paper. Also, if you include the graphics for your figures in your script, you can easily change the font or title when the editor or publisher wants you to make a change. But even more importantly, the script provides the mechanism to replicate fully the analysis you did on a particular data set. If there is any question about what was done or how you can always go back to the script and the data that it used. If you find a mistake, it can be corrected and the analysis can be run again with the correction.

If you want to take the next step beyond scripts investigate reproducible research tools. Those tools allow you to interleave the text for your paper with your R code and the results from the R code into tables, figures and computed values embedded directly into the text. A reproducible research document:

- Reduces the chances of making mistakes transcribing results into a paper
- Makes it easy to reproduce the entire paper once you receive review comments or if you happened to find errors in your data
- Provides documentation for all of the calculations in your document when you get questions from reviewers or interested researchers
- Provides a learning tool for others who can follow how you did your analysis.

The easiest reproducible research tool to learn is rmarkdown with the knitr package which can output HTML and Word files and a PDF if you have a LaTeX system. Both rmarkdown and knitr are integrated into the Rstudio development environment. If you do have LaTeX then you should consider using knitr or Sweave to construct your reproducible documents. LaTeX is far more capable than rmarkdown but it is harder to learn and you cannot create a Word document. I use the LyX gui interface for LaTeX which helps reduce the burden of learning LaTeX. This document was written with LyX for LaTeX with knitr. The code file accompanying this document includes the various code chunks contained in the LyX document.

I also need to introduce functions which can be used in scripts. You are already using base R functions and functions from R packages; however, you may never have written a function. To define a function from code in a script all you need to do is:

- wrap the script code in braces {},
- add somename= function() on the first line before the left brace {, where somename will be the name of your function,
- add a line before the right } with a `return` function to return any object or list of objects that you want to return to the calling environment, and
- you can also optionally add arguments in the () after function which allows you to pass values to the function; however, for our functions we will not use arguments.

Once your function is defined you need to call your function to execute the code in the function. It is similar to copying and pasting your script into the R console. However, functions are different in a couple of ways which make them very useful. First, the objects in the function "workspace" that would be returned with the `ls()` function are only those that you define in your function. That will be useful below when we create sets of models. If you want to see this for yourself type `ls()` in the R console with a non-empty workspace and see the names of the objects in your workspace. Then define the following very short function by typing the following into your R console: `myf=function() { x=1; return(ls()) }`. If you were then to type `myf` into the console, R will show you the contents of the function `myf` but doesn't call it. To call the function you need to type `myf()` and it will show you a vector containing the name "`x`" which is all that is contained in the function workspace. If you were to refer to `x` in the function it would use the `x` in that function and not any `x` that was defined outside of `myf`. But if the function used an object `y` that did not exist in `myf`

but existed in the parent environment (typically the workspace) of `myf`, it would find `y` and use it. In other words, if you create an object in a script (e.g., processed data list) outside of the function, you can use that object in the function that you also create in the script without passing it as an argument. If you don't fully understand some of this, it is no big loss. The main point is that you'll want to use both scripts and functions to organize and document your workflow and I'll demonstrate a template you can follow.

So why create a function? You create a function so that you can create a set of formulas for each parameter in the model and then use functions `create.model.list` and `mark.wrapper` to create the set of models from all combinations of the parameter specifications, run each and return a list with each of the models and a model selection table called `model.table`. Because the call to `create.model.list` is embedded in the function, it will only retrieve parameter specifications with names like parameter.xxx (e.g., Phi.time) defined within the function that are a list containing an object named `formula`. That was done to make sure it was not confused other objects with that naming structure (e.g., Phi.0=0). If for some reason you do not see all of the model combinations attempted, check to make sure each is a list with a formula. In my first attempt I spelled formula as formula and I didn't get all combinations. Below is an example which could be stored in a script.

```
# read in data
df=import.chdata("example.txt",field.types=c("f","n"))
# create fake age data and make it a factor variable
set.seed(9481)
df$ageclass=factor(ifelse(runif(600)<0.5,0,1),labels=c("Hatch-Year","Adult"))
# process data for CJS model with sex and age groups
dp=process.data(df,model="CJS",groups=c("sex","ageclass"),age.var=2,initial.age=c(0,1))
# create design data with age bins for Phi
ddl=make.design.data(dp,parameters=list(Phi=list(age.bins=c(0,1,4))),right=FALSE)
levels(ddl$Phi$age)=c("0","1Plus")
# create analysis function
do_analysis=function()
{
  # create formulas for Phi
  Phi.dot=list(formula=~1)
  Phi.sex=list(formula=~sex)
  Phi.sex.age=list(formula=~sex+age)
  Phi.sex.weight=list(formula=~sex+weight)
  #create formulas for p
  p.dot=list(formula=~1)
  p.time=list(formula=~time)
  # create all combinations (8 models)
  cml=create.model.list("CJS")
  # run all all 8 models and return as a list with class marklist
  results=mark.wrapper(cml,data=dp,ddl=ddl,output=FALSE,silent=TRUE)
  return(results)
}
# Now call the function to run the models and return the results
# stored into example.results
example.results=do_analysis()
# Show the model selection table
example.results
```

```
##                         model npar     AICc DeltaAICc       weight   Deviance
## 8 Phi(~sex + weight)p(~time)    6 1393.198   0.00000 0.9648913852 1381.11450
## 6          Phi(~sex)p(~time)    5 1400.498   7.29986 0.0250803711   51.89973
## 4    Phi(~sex + age)p(~time)    6 1402.521   9.32290 0.0091208732   51.89883
## 2            Phi(~1)p(~time)    4 1407.137  13.93844 0.0009073705   60.55824
```

```
## 7    Phi(~sex + weight)p(~1)    4 1515.238 122.03974 0.0000000000 1507.19810
## 5              Phi(~sex)p(~1)    3 1522.122 128.92373 0.0000000000  177.55939
## 3       Phi(~sex + age)p(~1)    4 1523.536 130.33764 0.0000000000  176.95740
## 1                Phi(~1)p(~1)    2 1529.047 135.84891 0.0000000000  186.49653
```

If you look at this simple script broadly there are 4 tasks and the first 2 you have already encountered with processing the data and creating the design data. The third task is to write a function with the set of parameter specifications that you want to examine and to call `create.model.list` and `mark.wrapper` and return the results from the latter. The final task is to call the function you created to fit the models and store the results which we will describe next. Scripts can become more complex once you begin to manipulate design data, produce plots or model average values, etc. However, the basic structure outlined above can always be used as the initial template for an analysis script. Just make sure to document, document, document! Be aware that many of the examples in the RMark help files do not use this approach because they were written before I developed `mark.wrapper`.

Now let's look at the results. What is shown is a model selection table that is similar to what the MARK interface provides. For each fitted model, listed in order of ascending AICc value, the following is given:

- model number: element number in the marklist that contains that model
- model name using the formulas for each parameter
- npar: number of beta parameters
- AICc: small sample corrected Akaike's Information Criterion
- DeltaAICc: difference in AICc from best model
- weight: model weight based on DeltaAICc value
- Deviance: residual deviance (null deviance - explained deviance from model)

One obvious aspect of this table and a point of confusion and questions is the deviance value that is wildly different for models with and without individual covariates. The reason is that null deviance is 0 for models with individual covariates and it has a non-zero value for models without covariates. You will never see this occur with models created in the MARK interface because when you define the project to the MARK interface with an individual covariate, all models will be treated as if they have an individual covariate even if one is not used in the DM. With RMark each input file is created based on the information provided in the formulas. If there is no individual covariate in the formulas, then none are described in the inp file passed to mark.exe so it is treated as if there were no individual covariates. This has the effect shown above and the positive effect that models without covariates will run more quickly in mark.exe. It runs more quickly because mark.exe accumulates similar capture histories and only computes the likelihood with the unique capture histories and uses the accumulated freq value in computing the likelihood (freq*lnl(ch)); whereas, with individual covariates it will not accumulate like capture histories if the inp file is defined with individual covariates. Note that RMark does this accumulation with or without covariates prior to creating the inp file and thus gets the gain in speed without relying on mark.exe. Now, if it bothers you to see the deviance value, you can use the following code to switch the model selection table to show -2LnL (2*negative log-likelihood) which is used to compute AICc which are not wildly different:

```
example.results$model.table=model.table(example.results,use.lnl=TRUE)
example.results
```

```
##                        model npar     AICc DeltaAICc       weight  Neg2LnL
## 8 Phi(~sex + weight)p(~time)    6 1393.198   0.00000 0.9648913852 1381.114
## 6          Phi(~sex)p(~time)    5 1400.498   7.29986 0.0250803711 1390.438
## 4    Phi(~sex + age)p(~time)    6 1402.521   9.32290 0.0091208732 1390.437
## 2            Phi(~1)p(~time)    4 1407.137  13.93844 0.0009073705 1399.097
```

```
## 7    Phi(~sex + weight)p(~1)    4 1515.238 122.03974 0.0000000000 1507.198
## 5              Phi(~sex)p(~1)    3 1522.122 128.92373 0.0000000000 1516.098
## 3        Phi(~sex + age)p(~1)    4 1523.536 130.33764 0.0000000000 1515.496
## 1                Phi(~1)p(~1)    2 1529.047 135.84891 0.0000000000 1525.035
```

Another argument to `model.table` called `model.name` is by default `TRUE` but if you set it to `FALSE` it will show the name of the R objects used to specify parameters rather than the formulas for each parameter. I find this useful when the formulas become complex and when there are several parameters in the model. Instead of using extensions like `.sex` or `.age` I use `.1,.2`, etc and set `model.name = FALSE` to get the results shown below.

```
do_analysis=function()
{
  # create formulas for Phi
  Phi.1=list(formula=~1)
  Phi.2=list(formula=~sex)
  Phi.3=list(formula=~sex+age)
  Phi.4=list(formula=~sex+weight)
  #create formulas for p
  p.1=list(formula=~1)
  p.2=list(formula=~time)
  # create all combinations (8 models)
  cml=create.model.list("CJS")
  # run all all 8 models and return as a list with class marklist
  results=mark.wrapper(cml,data=dp,ddl=ddl,output=FALSE,silent=TRUE)
  return(results)
}
# Now call the function to run the models and return the results
# stored into example.results
example.results=do_analysis()
# reset model names and use -2lnL
example.results$model.table=model.table(example.results,use.lnl=TRUE,model.name=FALSE)
# Show the model selection table
example.results

##       model npar     AICc DeltaAICc       weight  Neg2LnL
## 8 Phi.4.p.2    6 1393.198   0.00000 0.9648913852 1381.114
## 4 Phi.2.p.2    5 1400.498   7.29986 0.0250803711 1390.438
## 6 Phi.3.p.2    6 1402.521   9.32290 0.0091208732 1390.437
## 2 Phi.1.p.2    4 1407.137  13.93844 0.0009073705 1399.097
## 7 Phi.4.p.1    4 1515.238 122.03974 0.0000000000 1507.198
## 3 Phi.2.p.1    3 1522.122 128.92373 0.0000000000 1516.098
## 5 Phi.3.p.1    4 1523.536 130.33764 0.0000000000 1515.496
## 1 Phi.1.p.1    2 1529.047 135.84891 0.0000000000 1525.035
```

This is all very tidy having all of the models and the `model.table` in a single object called `example.results` here; but, you may be wondering how you retrieve results from a particular model. You use the model number listed in the `model.table` to retrieve that model. Below I use code to retrieve the model number of the best model (the first one in the table) and then show a summary of the results. Anywhere a model object can be used you can use `example.results[[#]]` where # is a model number.

```
# summarize the best model
best.model.number=as.numeric(row.names(example.results$model.table)[1])
summary(example.results[[best.model.number]])
```