CHAPTER

Likelihood Analysis of Spatial Capture-Recapture Models 6

So far, we have mainly focused on Bayesian analysis of spatial capture-recapture models. And, in the previous chapter we learned how to fit some basic spatial capture-recapture models in **BUGS** engines including **WinBUGS** and **JAGS**. Despite our focus on Bayesian analysis, it is instructive to develop the basic concepts and ideas behind likelihood methods and frequentist inference for SCR models. We recognized earlier (Chapter 5) that SCR models are versions of binomial (or other) GLMs, but with random effects. Throughout statistics, such models are routinely analyzed by likelihood, methods. In particular, likelihood analysis is based on the integrated or marginal likelihood, in which the random effects are removed, by integration, from the conditional-on-**s** likelihood (**s** being the individual activity center). This has been the approach taken by Borchers and Efford (2008), Dawson and Efford (2009), and related papers. Therefore, in this chapter, we provide some conceptual and technical foundation for likelihood-based analysis of spatial capture-recapture models.

We will show here that it is straightforward to compute the maximum likelihood estimates (MLE) for SCR models by integrated likelihood. We develop the MLE framework using **R**, and we also provide a basic introduction to the **R** package `secr` (Efford, 2011a) which does likelihood analysis of SCR models (see also the standalone program **DENSITY** (Efford et al., 2004)). To set the context for likelihood analysis of SCR models, we first analyze the SCR model when N is known because, in that case, analysis proceeds as in standard GLMMs. We generalize the model to allow for unknown N using both conventional ideas based on the "full likelihood" (e.g., Borchers et al., 2002) and also using a formulation based on data augmentation. We obtain the MLEs for the SCR model from the wolverine camera trapping study (Magoun et al., 2011) analyzed in previous chapters to compare/contrast the results.

6.1 MLE for SCR with known *N*

We noted in Chapter 5 that, with N known, the basic SCR model is a type of binomial model with a random effect. For such models we can obtain maximum likelihood estimators of model parameters based on integrated likelihood. The integrated likelihood is based on the marginal distribution of the data, y, in which the random effects are removed by integration from the conditional-on-**s** distribution of the observations.

Spatial Capture-Recapture. http://dx.doi.org/10.1016/B978-0-12-405939-9.00006-2

See Chapter 2 for a review of marginal, conditional, and joint distributions. Conceptually, any SCR model begins with a specification of the conditional-on-**s** model $[y|\mathbf{s}, \boldsymbol{\alpha}]$ and a "prior distribution" for **s**, say $[\mathbf{s}]$. Then, the marginal distribution of the data y is

$$[y|\boldsymbol{\alpha}] = \int_{\mathcal{S}} [y|\mathbf{s}, \boldsymbol{\alpha}][\mathbf{s}]d\mathbf{s}.$$

When viewed as a function of $\boldsymbol{\alpha}$ for purposes of estimation, the marginal distribution $[y|\boldsymbol{\alpha}]$ is often referred to as the *integrated likelihood*.

It is worth analyzing the simplest SCR model with known N in order to understand the underlying mechanics and basic concepts. These are directly relevant to the manner in which many capture-recapture models are classically analyzed, such as model M_h, and individual covariate models (see Chapter 4).

To develop the integrated likelihood for SCR models, we first identify the conditional-on-**s** likelihood. We work with the Bernoulli encounter model here (SCR0) although the basic approach is the same for all other SCR models. The observation model for each encounter observation y_{ij}, for individual i and trap j, specified conditional-on-$\mathbf{s}_i$, is

$$y_{ij}|\mathbf{s}_i \sim \text{Binomial}(K, p_\alpha(\mathbf{x}_j, \mathbf{s}_i)), \tag{6.1.1}$$

where we have indicated the dependence of encounter probability, p_{ij}, on **s** and parameters $\boldsymbol{\alpha}$ explicitly. For example, p_{ij} might be the Gaussian model given by

$$p_{ij} = \text{logit}^{-1}(\alpha_0)\exp(-\alpha_1\|\mathbf{x}_j - \mathbf{s}_i\|^2),$$

where $\alpha_1 = 1/(2\sigma^2)$. The joint distribution of the data for individual i is the product of J such terms (i.e., contributions from each of J traps).

$$[\mathbf{y}_i|\mathbf{s}_i, \boldsymbol{\alpha}] = \prod_{j=1}^{J} \text{Binomial}(K, p_\alpha(\mathbf{x}_j, \mathbf{s}_i)).$$

We note this assumes that encounter of individual i in each trap is independent of encounter in every other trap, conditional on $\mathbf{s}_i$. This is the fundamental property of the basic model SCR0. The marginal likelihood is computed by removing $\mathbf{s}_i$, by integration from the conditional-on-**s** likelihood, so we compute:

$$[\mathbf{y}_i|\boldsymbol{\alpha}] = \int_{\mathcal{S}} [\mathbf{y}_i|\mathbf{s}_i, \boldsymbol{\alpha}][\mathbf{s}_i]d\mathbf{s}_i.$$

In most SCR models, $[\mathbf{s}] = 1/A(\mathcal{S})$ where $A(\mathcal{S})$ is the area of the prescribed state-space $\mathcal{S}$ (but see Chapter 11 for alternative specifications of $[\mathbf{s}]$). The joint likelihood for all N individuals, assuming independence of encounters among individuals, is the product of N such terms:

$$\mathcal{L}(\boldsymbol{\alpha}|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N) = \prod_{i=1}^{N}[\mathbf{y}_i|\boldsymbol{\alpha}].$$

We emphasize that two independence assumptions are explicit in this development: independence of trap-specific encounters within individuals and independence among individuals. In particular, these assumptions are only valid when individuals are not physically restrained or removed upon capture, and when traps do not "fill up."

The key operation for computing the likelihood is solving a two-dimensional integration problem. There are some general-purpose **R** packages that implement a number of multi-dimensional integration routines including `adapt` (Genz et al., 2007) and `R2cuba` (Hahn et al., 2010). In practice, we won't rely on these extraneous **R** packages (except see Chapter 11 for an application of `R2cuba`), but instead will use perhaps less efficient methods in which we replace the integral with a summation over an equal-area mesh of points on the state-space $\mathcal{S}$ and explicitly evaluate the integrand at each point. We invoke the rectangular rule for integration here,[1] in which we evaluate the integrand on a regular grid of points of equal area and compute the average of the integrand over that grid of points. Let $u = 1, 2, \ldots, nG$ index a grid of nG points, $\mathbf{s}_u$, where the area of grid cells is constant. In this case, the integrand, i.e., the marginal pmf of $\mathbf{y}_i$, is approximated by

$$[\mathbf{y}_i|\boldsymbol{\alpha}] = \frac{1}{nG}\sum_{u=1}^{nG}[\mathbf{y}_i|\mathbf{s}_u, \boldsymbol{\alpha}]. \tag{6.1.2}$$

This is a specific case of the general expression that could be used for approximating the integral for any arbitrary distribution $[\mathbf{s}]$. The general case is

$$[\mathbf{y}_i|\boldsymbol{\alpha}] = \frac{A(\mathcal{S})}{nG}\sum_{u=1}^{nG}[\mathbf{y}_i|\mathbf{s}_u, \boldsymbol{\alpha}][\mathbf{s}_u].$$

Under the uniformity assumption, $[\mathbf{s}] = 1/A(\mathcal{S})$ and thus the grid-cell area cancels in the above expression to yield Eq. (6.1.2). The rectangular rule for integration can be seen as an application of the Law of Total Probability for a discrete random variable $\mathbf{s}$, having nG unique values with equal probabilities $1/nG$.

6.1.1 Implementation (simulated data)

Here, we will illustrate how to carry out this integration and optimization based on the integrated likelihood using simulated data (see Section 5.5). Using `simSCR0`, we simulate data for 100 individuals and an array of 25 traps laid out in a 5×5 grid of traps having unit spacing. The specific encounter model is the Gaussian model. The 100 activity centers were simulated on a state-space defined by an 8×8 square within which the trap array was centered (thus the trap array is buffered by 2 units). In the following **R** commands we generate the data and harvest the required data objects:

[1] e.g., http://en.wikipedia.org/wiki/Rectangle_method.

```
  ## simulate a complete data set
> data <- simSCR0(discard0=FALSE,rnd=2013)
  ## extract the objects that we need for analysis
> y <- data$Y
> traplocs <- data$traplocs
> nind <- nrow(y)   ## in this case nind=N
> J <- nrow(traplocs)
> K <- data$K
> xlim <- data$xlim
> ylim <- data$ylim
```

Now, we need to define the integration grid, say `G`, which we do with the following set of **R** commands (here, `delta` is the grid spacing):

```
> delta <- .2
> xg <- seq(xlim[1]+delta/2,xlim[2]-delta/2,by=delta)
> yg <- seq(ylim[1]+delta/2,ylim[2]-delta/2,by=delta)
> npix <- length(xg)          # valid for square state-space only
> G <- cbind(rep(xg,npix),sort(rep(yg,npix)))
> nG <- nrow(G)
```

In this case, the integration grid is set up as a grid with spacing $\delta = 0.2$, which produces a 40×40 grid of points for evaluating the integrand with the state-space buffer set at 2. We note that the integration grid is set up here to correspond exactly to the state-space used in simulating the data. However, in practice, we wouldn't know this, and our estimate of N (for the unknown case, see below) would be sensitive to choice of the extent of the integration grid. As we've discussed previously, density, which is N standardized by the area of the state-space, will not be so sensitive in most cases.

We are now ready to compute the conditional-on-**s** likelihood and carry out the marginalization described by Eq. (6.1.2). We need to do this by defining an **R** function that computes the likelihood for the integration grid, as a function of the data objects `y` and `traplocs`, which were created above. However, it is a bit untidy to store the grid information in your workspace, and define the likelihood function in a way that depends on these things that exist in your workspace. Therefore, we build the **R** function so that it computes the integration grid *within* the function, thereby avoiding potential problems if our trapping grid locations change, or if we want to modify the state-space buffer. We therefore define the function, called `intlik1`, to which we pass the data objects and other information necessary to compute the marginal likelihood. This function is available in the `scrbook` package (use `?intlik1` at the **R** prompt). The code is reproduced here:

```
intlik1 <- function(parm,y=y,X=traplocs, delta=.2, ssbuffer=2){

   Xl <- min(X[,1]) - ssbuffer   ## These lines of code are setting up the
   Xu <- max(X[,1]) + ssbuffer   ## support for the integration which is
   Yu <- max(X[,2]) + ssbuffer   ## the same as the state-space of "s"
   Yl <- min(X[,2]) - ssbuffer
   xg <- seq(Xl+delta/2,Xu-delta/2„length=npix)
   yg <- seq(Yl+delta/2,Yu-delta/2„length=npix)
   npix<- length(xg)
   G <- cbind(rep(xg,npix),sort(rep(yg,npix)))
   nG <- nrow(G)
   D <- e2dist(X,G)

   alpha0 <- parm[1]
   alpha1 <- exp(parm[2])  # alpha1 restricted to be positive here

   probcap <- plogis(alpha0)*exp(-alpha1*D*D)
   Pm <- matrix(NA,nrow=nrow(probcap),ncol=ncol(probcap))
                    # Frequency of all-zero encounter histories
   n0 <- sum(apply(y,1,sum)==0)
                    # Encounter histories with at least 1 detection
   ymat <- y[apply(y,1,sum)>0,]
   ymat <- rbind(ymat,rep(0,ncol(ymat)))
   lik.marg <- rep(NA,nrow(ymat))

   for(i in 1:nrow(ymat)){
       ## Nextline: log conditional likelihood for ALL possible values of s
       Pm[1:length(Pm)] <- dbinom(rep(ymat[i,],nG),K,probcap[1:length(Pm)],
                            log=TRUE)
       ## Next line: sum the log conditional likelihoods, exp() result
       ##   same as taking the product
       lik.cond <- exp(colSums(Pm))
       ## Take the average value == computing marginal
       lik.marg[i] <- sum( lik.cond*(1/nG))
   }
   ## n0 = number of all-0 encounter histories
   nv <- c(rep(1,length(lik.marg)-1),n0)
   return( -1*( sum (nv*log(lik.marg)) ) )
}
```

We emphasize that this and subsequent functions are not meant to be general-purpose routines for solving all of your SCR problems but, rather, they are meant for illustrative purposes—so you can see how the integrated likelihood is constructed and how we connect it to data and other information that is needed.

The function `intlik1` accepts as input the encounter history matrix, `y`, the trap locations, `X`, and the state-space buffer. This allows us to vary the state-space buffer and easily evaluate the sensitivity of the MLE to the size of the state-space. Note that we have a peculiar handling of the encounter history matrix `y`. In particular, we remove the all-zero encounter histories from the matrix and tack on a single all-zero encounter history as the last row, which then gets weighted by the number of such encounter histories (`n0`). This is a bit long-winded and strictly unnecessary when N is known, but we did it this way because the extension to the unknown-N case is now transparent (as we demonstrate in the following section). The matrix `Pm` holds the log-likelihood contributions of each encounter frequency for each possible state-space location of the individual. The log contributions are summed up and the result exponentiated on the next line, producing `lik.cond`, the conditional-on-$\mathbf{s}$ likelihood (Eq. (6.1.1)). The marginal likelihood (`lik.marg`) sums up the conditional elements weighted by the probabilities $[\mathbf{s}]$ (Eq. (6.1.2)).

This is a fairly primitive function that doesn't allow much flexibility in the data structure. For example, it assumes that K, the number of replicates, is constant for each trap. Further, it assumes that the state-space is a square. We generalize this to some extent later in this chapter.

Here is the **R** command for maximizing the likelihood using `nlm` (the function `optim` could also be used) and saving the results into an object called `frog`. The output is a list shown below for the simulated data set

```
> starts <- c(-2,2)
> frog <- nlm(intlik1,starts,y=y,X=traplocs,delta=.1,ssbuffer=2,hessian=TRUE)
> frog

$minimum
[1] 297.1896

$estimate
[1] −2.504824  2.373343

$gradient
[1] −2.069654e-05  1.968754e-05

$hessian
            [,1]        [,2]
[1,]    48.67898   -19.25750
[2,]   -19.25750    13.34114

$code
[1] 1

$iterations
[1] 11
```

Details about this output can be found on the help page for `nlm`. We note briefly that `frog$minimum` is the negative log-likelihood value at the MLEs, which are stored in the `frog$estimate` component of the list. The order of the parameters is as they are defined in the likelihood function so, in this case, the first element (value = −2.504824) is the logit transform of p_0 and the second element (value = 2.373343) is the value of α_1, the "coefficient" on distance-squared. The Hessian is the observed Fisher information matrix, which can be inverted to obtain the variance-covariance matrix using the command:

```
> solve(frog$hessian)
```

It is worth drawing attention to the fact that the estimates are slightly different than the Bayesian estimates reported in Section 5.6. There are several reasons for this. First, Bayesian inference is based on the posterior distribution and it is not generally the case that the MLE should correspond to any particular value of the posterior distribution. If the prior distributions in a Bayesian analysis are uniform, then the (multivariate) mode of the posterior is the MLE, but note Bayesians almost always report posterior *means* so that there will typically be some discrepancy. Secondly, we have implemented an approximation to the integral here and there might be a slight bit of error induced by that. We will evaluate that shortly. Third, the Bayesian analysis by MCMC is itself subject to some amount of Monte Carlo error, which the analyst should always be aware of in practical situations. All of these different explanations are likely responsible for some of the discrepancy. Despite these, we see general consistency between the two estimates.

In summary, for the basic SCR model, computing the integrated likelihood is a simple task when N is known. Even for N unknown it is not too difficult, and we will do that shortly.

6.2 MLE when N is unknown

Here, we build on the previous introduction to integrated likelihood and consider now the case in which N is unknown. We will see that adapting the analysis based on the known-N model is straightforward for the more general problem. The main distinction is that we don't observe the all-zero encounter history so we have to make sure we compute the probability for that encounter history. We do that by including a row of zeros in the encounter history matrix and treating the number of such all-zero encounter histories (that is, the number of individuals *not* encountered) as an unknown parameter of the model. Call that unknown quantity n_0, so that $N = n_0 + n$, where n is the number of unique individuals encountered. We will usually parameterize the likelihood in terms of n_0, because optimization over a parameter space in which $\log(n_0)$ is unconstrained is preferred to a parameter space in which N must be constrained so that $N \geq n$. With n_0 unknown, we have to be sure to include a combinatorial term in the likelihood to account for the fact that, of the n observed individuals, there are $\binom{N}{n}$ ways to realize a sample of size n. The combinatorial term involves the unknown n_0 and thus it must be included in the likelihood. In evaluating the log-likelihood, we have to compute terms such as the log-factorial, $\log(N!) = \log((n_0 + n)!)$. We do this in **R** by making use of the log-gamma function (`lgamma`) and the identity

$$\log(N!) = \text{lgamma}(N + 1).$$

Therefore, to compute the likelihood, we require the following three components: (1) The marginal probability of each $\mathbf{y}_i$ as before:

$$[\mathbf{y}_i|\boldsymbol{\alpha}] = \int_{\mathcal{S}} [\mathbf{y}_i|\mathbf{s}_i, \boldsymbol{\alpha}][\mathbf{s}_i]d\mathbf{s}_i.$$

(2) The probability of an all-0 encounter history:

$$\pi_0 = [\mathbf{y} = \mathbf{0}|\boldsymbol{\alpha}] = \int_{\mathcal{S}} \text{Binomial}(\mathbf{0}|\mathbf{s}_i, \boldsymbol{\alpha})[\mathbf{s}_i]d\mathbf{s}_i.$$

(3) The combinatorial term: $\binom{N}{n}$.

Then, the marginal likelihood has this form:

$$\mathcal{L}(\boldsymbol{\alpha}, n_0|\mathbf{y}) = \frac{N!}{n!n_0!}\left\{\prod_{i=1}^{n}[\mathbf{y}_i|\boldsymbol{\alpha}]\right\}\pi_0^{n_0}. \tag{6.2.1}$$

This is discussed in Borchers and Efford (2008, p. 379) as the conditional-on-N form of the likelihood—we also call it the "binomial form" of the likelihood because of its appearance.

Operationally, things proceed much as before: We compute the marginal probability of each observed $\mathbf{y}_i$, i.e., by removing the latent $\mathbf{s}_i$ by integration. In addition, we compute the marginal probability of the "all-zero" encounter history, and make sure to weight it n_0 times. We accomplish this by "padding" the data set with a single encounter history having $y_{n+1,j} = 0$ for all traps $j = 1, 2, \ldots, J$. Then we make sure to include the combinatorial term in the likelihood or log-likelihood computation. We demonstrate this by analyzing a simulated data set. To set some things up in our workspace we do this:

```
## Obtain a simulated data set
> data <- simSCR0(discard0=TRUE, rnd=2013)

## Extract the items we need for analysis
> y <- data$Y
> nind <- nrow(y)
> traplocs <- data$traplocs
> J <- nrow(traplocs)
> K <- data$K
```

Recall that these data are simulated by default with $N = 100$, on an 8×8 unit state-space representing the trap locations buffered by two units, although you can modify the simulation script easily.

As before, the likelihood is defined as an **R** function, `intlik2`, which takes as an argument the unknown parameters of the model and additional arguments as prescribed: the encounter history matrix `y`, the trap locations `traplocs`, the spacing of the integration grid (argument `delta`), and the state-space buffer. Here is the new likelihood function:

```
intlik2 <- function(parm,y=y,X=traplocs,delta=.3,ssbuffer=2){

  Xl <- min(X[,1]) - ssbuffer
  Xu <- max(X[,1]) + ssbuffer
  Yu <- max(X[,2]) + ssbuffer
  Yl <- min(X[,2]) - ssbuffer

  xg <- seq(Xl+delta/2,Xu-delta/2,delta)
  yg <- seq(Yl+delta/2,Yu-delta/2,delta)
  npix.x <- length(xg)
  npix.y <- plength(yg)
  area <- (Xu-Xl)*(Yu-Yl)/((npix.x)*(npix.y))
  G <- cbind(rep(xg,npix.y),sort(rep(yg,npix.x)))
  nG <- nrow(G)
```

```
   D <- e2dist(X,G)
   # extract the parameters from the input vector
   alpha0 <- parm[1]
   alpha1 <- exp(parm[2])
   n0 <- exp(parm[3])  # note parm[3] lives on the real line
   probcap <- plogis(alpha0)*exp(-alpha1*D*D)
   Pm <- matrix(NA,nrow=nrow(probcap),ncol=ncol(probcap))
   ymat <- rbind(y,rep(0,ncol(y)))

   lik.marg <- rep(NA,nrow(ymat))
   for(i in 1:nrow(ymat)){
      Pm[1:length(Pm)] <- (dbinom(rep(ymat[i,],nG),K,probcap[1:length(Pm)],
                          log=TRUE))
      lik.cond <- exp(colSums(Pm))
      lik.marg[i] <- sum(lik.cond*(1/nG) )
   }
   nv <- c(rep(1,length(lik.marg)-1),n0)
   ## part1 here is the combinatorial term.
   ## math: log(factorial(N)) = lgamma(N+1)
   part1 <- lgamma(nrow(y)+n0+1) - lgamma(n0+1)
   part2 <- sum(nv*log(lik.marg))
   return( -1*(part1+part2))
}
```

To execute this function for the data we created with `simSCR0`, we execute the following commands (saving the results in our friend `frog`). This results in the usual output, including the parameter estimates, the gradient, and the numerical Hessian, which is useful for obtaining asymptotic standard errors (see below):

```
> starts <- c(-2.5,0,4)
> frog <- nlm(intlik2,starts,hessian=TRUE,y=y,X=traplocs,delta=.2,ssbuffer=2)

Warning message:
In nlm(intlik2, starts, hessian = TRUE, y = y, X = traplocs, delta = 0.2,  :
  NA/Inf replaced by maximum positive value

> frog
$minimum
[1] 113.5004

$estimate
[1] -2.538333  0.902807  4.232810

[... additional output deleted ...]
```

Executing `nlm` here usually produces one or more **R** warnings due to numerical calculations happening on extremely small or large numbers (calculation of p near the edge of the state-space), and they also happen if a poor parameterization is used, which produces evaluations of the objective function beyond the boundary of the parameter space (e.g., $n_0 < 0$). Such numerical warnings can often be minimized or avoided altogether by picking judicious starting values of parameters or properly transforming or scaling the parameters but, in general, they can be ignored. You will see from the `nlm` output that the algorithm performed satisfactorily in minimizing the objective function. The estimate of population size, $\hat{N}$, for the state-space (using the default state-space buffer) is

```
> Nhat <- nrow(y) + exp(4.2328)   ### This is n + MLE of n0
> Nhat
[1] 110.9099
```

This estimate differs from the data-generating value ($N = 100$), as we might expect for a single realization. We usually will present an estimate of uncertainty associated with this MLE, which we can obtain by inverting the Hessian. Note that $\text{Var}(\hat{N}) = n + \text{Var}(\hat{n}_0)$. Since we have parameterized the model in terms of $\log(n_0)$, we use the delta method[2] described in Williams et al. (2002, Appendix F4) (see also Ver Hoef, 2012) to obtain the variance on the scale of n_0 as follows:

```
> (exp(4.2328)^2)*solve(frog$hessian)[3,3]
[1] 260.2033
```

```
> sqrt(260)
[1] 16.12452
```

Therefore, the asymptotic "Wald-type" confidence interval for N is $110.91 \pm 1.96 \times 16.125 = (79.305, 142.515)$. To report this in terms of density, we scale appropriately by the area of the prescribed state-space, which is 64 square units in area. Our MLE of D is $\hat{D} = 110.91/64 = 1.733$ individuals per square unit. To get the standard error for $\hat{D}$ we need to divide the SE for $\hat{N}$ by the area of the state-space, and so $\text{SE}(\hat{D}) = (1/64) * 16.12452 = 0.252$.

6.2.1 Integrated likelihood under data augmentation

The analysis developed in the previous sections is based on the likelihood in which N (or n_0) is an explicit parameter. This is usually called the "full likelihood" or sometimes "unconditional likelihood" (Borchers et al., 2002), because it is the likelihood for all individuals in the population, not just those that have been captured, i.e., it is not the likelihood that is *conditional on capture*. It is also possible to express an alternative unconditional likelihood using data augmentation, replacing the parameter N with ψ as we described in Section 4.2 (also see Royle and Dorazio, 2008, Section 7.1.6). While it is possible to carry out likelihood analysis of models under data augmentation, we primarily advocate data augmentation for Bayesian analysis.

6.2.2 Extensions

So far, we have only considered basic SCR models with no additional covariates. However, in practice, we are interested in covariate effects including "behavioral response," sex-specificity of parameters, and potentially others. Some of these can be added directly to the likelihood if the covariate is fixed and known for all individuals, captured or not. An example is a behavioral response, which amounts to having a

[2]We found a good set of notes on the delta approximation on Dr. David Patterson's ST549 notes: http://www.math.umt.edu/patterson/549/Delta.pdf.

covariate $x_{ik} = 1$ if individual i was captured prior to occasion k and $x_{ik} = 0$ otherwise. For uncaptured individuals, $x_{ik} = 0$ for all k. Royle et al. (2011b) called this a global behavioral response, because the covariate is defined for all traps, no matter the trap in which an individual was captured. We also define a *local* behavioral response, that occurs at the level of the trap, i.e., $x_{ijk} = 1$ if individual i was captured in trap j prior to occasion k, etc. Trap-specific covariates such as trap type or status, or time-specific covariates such as date, are easily accommodated as well. As an example, Kéry et al. (2010) develop a model for the European wildcat (*Felis silvestris*) in which traps are either baited or not (a trap-specific covariate with only two values), and encounter probability varies over time in the form of a quadratic seasonal response. We consider models with behavioral response or fixed covariates in Chapter 7. The integrated likelihood routines we provided above can be modified directly for such cases, and we leave that to the interested reader to investigate.

Sex specificity is more difficult to deal with, since sex is not known for uncaptured individuals (and sometimes not even for all captured individuals). To analyze such models, we do Bayesian analysis of the joint likelihood using data augmentation (Gardner et al., 2010b; Russell et al., 2012), discussed further in Chapter 7. For such covariates (i.e., that are not fixed and known for all individuals), it is somewhat more challenging to do MLE based on the joint or full likelihood as we have developed it above. Instead, it is more conventional to use what is colloquially referred to as the "Huggins-Alho"-type model, which is one of the approaches taken in the software package `secr` (Efford, 2011a). We introduce the `secr` package in Section 6.5.

6.3 Classical model selection and assessment

In most analyses, one is interested in choosing from among various potential models, or ranking models, or otherwise assessing the relative merits of a set of models. A good thing about classical analysis based on likelihood is that we can apply Akaike Information Criterion (AIC) methods (Burnham and Anderson, 2002; Borchers and Efford, 2008, Section 6.5) without difficulty. AIC is convenient for assessing the relative merits of these different models, although if there are only a few models, it is not objectionable to use hypothesis tests or confidence intervals to determine importance of effects. A second model selection context has to do with choosing among various detection models (e.g., Gaussian, hazard rate, exponential, etc.), although, as a general rule, we don't recommend this application of model selection. This is because there is hardly ever (if at all) a rational subject-matter based reason motivating specific distance functions. As a result, we believe that doing too much model selection can lead to overfitting and thus potentially overstatement of precision. This is the main reason that we haven't loaded you down with a basket of models for detection probability so far, although we discuss many possibilities in Chapter 7.

Goodness-of-fit or model-checking—For many standard capture-recapture models, it is possible to identify goodness-of-fit statistics based on the multinomial likelihood (Cooch and White, 2006, Chapter 5), and evaluate model adequacy using formal statistical tests. Similar strategies can be applied to SCR models using expected cell frequencies based on the marginal distribution of the observations. Also, because computing MLEs is somewhat more efficient in many cases compared to Bayesian analysis, it is sometimes feasible to use bootstrap methods. At the present time, there are few applications of goodness-of-fit testing for SCR models based on likelihood inference (but see Borchers and Efford, 2008, Section 6.5). We discuss the use of Bayesian p-values for assessing model fit in Chapter 8. An important practical problem in trying to evaluate goodness-of-fit is that, in realistic sample sizes, fit tests often lack the power to detect departures from the model under consideration and so they may not be generally useful in practice.

6.4 Likelihood analysis of the wolverine camera trapping data

Here, we compute the MLEs for the wolverine data using an expanded version of the function we developed in the previous section. To accommodate that each trap might be operational a variable number of nights, we provided an additional argument to the likelihood function (allowing for a vector $\mathbf{K} = (K_1, \ldots, K_J)$), which requires also a modification to the construction of the likelihood. The more general function (`intlik3`) is given in the **R** package `scrbook`. In addition, this modification accommodates the state-space as a rectangle, and we included a line of code to compute the state-space area, which we apply below for computing density. To use this function to obtain the MLEs for the wolverine camera trap study, we execute the following commands (note: these commands will execute if you type `example(intlik3)`):

```
> library(scrbook)
> data(wolverine)

> traps <- wolverine$wtraps
> traplocs <- traps[,2:3]/10000
> K.wolv <- apply(traps[,4:ncol (traps)],1,sum)

> y3d <- SCR23darray(wolverine$wcaps,traps)
> y2d <- apply(y3d,c(1,2),sum)

> starts <- c(-1.5,0,3)

> wolv <- nlm(intlik3,starts,hessian=TRUE,y=y2d,K=K.wolv,X=traplocs,
              delta=.2,ssbuffer=2)

> wolv
$minimum
[1] 220.4313

$estimate
[1] -2.8176120 0.2269395 3.5836875

[.... output deleted ....]
```

Of course we're interested in obtaining an estimate of population size for the prescribed state-space, or density, and associated measures of uncertainty which we do using the delta method. To do all of that we need to manipulate the output of `nlm`, since we have our estimate in terms of $\log(n_0)$. We execute the following commands:

```
> wolv <- nlm(intlik3,starts,hessian=TRUE,y=y2d,K=K.wolv,X=traplocs,delta=.2,
            ssbuffer=2)
> Nhat <- nrow(y2d)+exp(wolv$estimate[3])
> area <- attr(intlik3(starts,y=y2d,K=K.wolv,X=traplocs,delta=.2,ssbuffer=2),
            "SSarea")
> Dhat <- Nhat/area

> Dhat
[1] 0.5494947

> SE <- (1/area)*exp(wolv$estimate[3])*sqrt(solve (wolv$hessian)[3,3])

> SE
[1] 0.1087073
```

Our estimate of density is 0.55 individuals per "standardized unit" which is 100 km^2, because we divided UTM coordinates by 10,000. So this is about 5.5 individuals per 1,000 km^2, with a SE of around 1.09 individuals. This compares closely with 5.77 reported in Section 5.9 based on Bayesian analysis of the model.

6.4.1 Sensitivity to integration grid and state-space buffer

The effect of approximating the integral by a discrete mesh of points is that it induces some numerical error that increases as the coarseness of the mesh increases. To evaluate the effect (or sensitivity) of the integration spacing, we obtained the MLEs for a state-space buffer of 2 (standardized units) and for integration grids with spacing $\delta = .3, .2, .1, .05$. The MLEs for these four cases including the relative runtime are given in Table 6.1. We see that the results change only slightly as the integration grid changes. Conversely, the runtime on the platform of the day for the four cases increases rapidly. These runtimes could be regarded in relative terms, across platforms, for gaging the decrease in speed as the fineness of the integration grid increases.

Table 6.1 Runtime and MLEs for different integration grid resolutions for the wolverine camera trapping data.

		Estimates		
δ	Runtime(s)	$\hat{\alpha}_0$	$\hat{\alpha}_1$	$\widehat{\log(n_0)}$
0.30	9.9	−2.820	1.258	3.570
0.20	32.3	−2.818	1.255	3.584
0.10	115.1	−2.817	1.255	3.599
0.05	407.3	−2.817	1.255	3.607

Table 6.2 Results of the effect of the state-space buffer on the MLE. Given are the state-space buffer, area of the state-space (area), the MLE of N ($\hat{N}$) for the prescribed state-space, and the corresponding MLE of density ($\hat{D}$).

Buffer	Area	$\hat{N}$	$\hat{D}$
1.0	66.982	37.733	0.563
1.5	84.362	46.210	0.548
2.0	103.743	57.006	0.549
2.5	125.123	69.036	0.552
3.0	148.503	82.175	0.553
3.5	173.884	96.440	0.555
4.0	201.264	111.835	0.556

We studied the effect of the state-space buffer on the MLEs, using a fixed $\delta = .2$ for all analyses. We used state-space buffers of 1–4 units stepped by .5. As we can see (Table 6.2), the estimates of D stabilize rapidly and the incremental difference is small.

6.4.2 Using a habitat mask (restricted state-space)

In Section 5.10 we used a discrete representation of the state-space in order to have control over its extent and shape. This makes it easy to do things like clip out non-habitat, or create a *habitat mask* that defines suitable habitat. Clearly, that formulation of the model is relevant to the calculation of the marginal likelihood in the sense that the discrete state-space is equivalent to the integration grid. Thus, for example, we could easily compute the MLE of parameters under some model with a restricted state-space merely by creating the required state-space at whatever grid resolution is desired, and then inputting that state-space into the likelihood functions above, instead of computing it within the function. We can easily create an explicit state-space grid for integration from arbitrary polygons or GIS shapefiles, which we demonstrate here. Our approach is to create the integration grid (or state-space grid) outside of the likelihood evaluation, and then determine which points of the grid lie in the polygon defined by the shapefile using functions in the **R** packages `sp` and `maptools`. For each point in the state-space grid (object `G`, in the code below which is assumed to exist in your workspace), we determine whether it is inside the polygon,[3] identifying such points with a value of `mask=1`, and `mask=0` for points that are *not* in the polygon. We load the shapefile by applying the `readShapeSpatial` function.

[3]We perform this check using the `over` function. This function takes as its second argument an object of the class “SpatialPolygons” or “SpatialPolygonsDataFrame,” which can hold additional information for each polygon, and the output value of the function differs slightly for these two classes: if using a “SpatialPolygons” object, the function returns a vector of length equal to the number of points (e.g., in the example above), but if using a “SpatialPolygonsDataFrame” it returns a data frame (e.g., see Section 17.7 in Chapter 17). If you use the `over` function, make sure you know the class of your second argument so that when processing the function output you index it correctly.

We have saved the result into an **R** data object called SSp, which is in the scrbook package. Here are the **R** commands for doing this (see also the helpfile ?intlik4):

```
> library(maptools)
> library(sp)
> library(scrbook)

#### If we have the .shp file in place, we would use this command:
#### SSp <- readShapeSpatial('Sim_Polygon.shp')
#### The object SSp is in data(fakeshapefile)
> data(fakeshapefile)
> Pcoord <- SpatialPoints(G)
> PinPoly <- over(Pcoord,SSp)   ### determine if each point is in polygon
> mask <- as.numeric(!is.na(PinPoly[,1]))   ## convert to binary 0/1
> G <- G[mask==1,]
```

We created the function intlik4 to accept the integration grid as an explicit argument; this function is also available in the package scrbook.

We apply this modified function to the wolverine camera trapping study. Royle et al. (2011b) created 2, 4, and 8-km state-space grids so as to remove "non-habitat" (mostly ocean, bays, and large lakes). To set up the wolverine data and fit the model with the habitat mask using maximum likelihood we execute the following commands (data loaded as in Section 6.4:

```
> starts <- c(-1.5,0,3)
> wolv <- nlm(intlik4, starts, y=y2d, K=K.wolv, X=traplocs, G=G)

> wolv

$minimum
[1] 225.8355

$estimate
[1] -2.9955424  0.2350885  4.1104757

[... some output deleted ...]
```

As before, we convert the parameter estimates to estimates of total population size for the prescribed state-space, and then obtain an estimate of density (per 1,000 km^2) using the area computed as the number of pixels in the state-space grid, G, multiplied by the area per grid cell. In the present case (the calculation above) we used a state-space grid with 2 km $\times$ 2 km pixels. Finally, we compute a standard error using the delta approximation:

```
> area <- nrow(G)*4
# Nhat = n (observed) + MLE of n0 (not observed)
> Nhat <- 21 + exp(wolv$estimate[3])
> SE <- exp(wolv$estimate[3])*sqrt(solve(wolv$hessian)[3,3])
> D <- (Nhat/(nrow(G)*area))*1000
> SE.D <- (SE/(nrow(G)*area))*1000
```

We did this for each 2 km, 4 km, and 8 km state-space grid, which produced the estimates summarized in Table 6.3. These estimates compare with the 8.6 (2-km grid) and 8.2 (8-km grid) reported in Royle et al. (2011b) based on a clipped state-space as described in Section 5.10.

Table 6.3 Maximum likelihood estimates (MLEs) and asymptotic standard errors (SE) for the wolverine camera trapping data using 2, 4, and 8 km state-space grids and a habitat mask.

Grid	α_0	α_1	$\log(n_0)$	N	SE	D(1000)	SE
2	−3.00	1.27	4.11	81.98	16.31	8.31	1.65
4	−2.99	1.34	4.16	84.88	16.76	8.57	1.69
8	−3.05	1.08	4.06	78.89	15.31	7.85	1.52

6.5 DENSITY and the R package `secr`

DENSITY is a software program developed by Efford (2004) for fitting spatial capture-recapture models based mostly on classical maximum likelihood estimation and related inference methods. Efford (2011a) has also released an **R** package called `secr`, that contains much of the functionality of **DENSITY**, but also incorporates new models and features. Here, we briefly introduce the `secr` package, which we prefer to use over **DENSITY**, because it allows us to remain in the **R** environment for data processing and summarization. We provide a brief introduction to `secr` and some of its capabilities here, and we also use it for doing some analysis in other parts of this book (e.g., Chapter 7). We believe that `secr` will be sufficient for many (if not most) of the SCR problems that one might encounter. It provides a flexible analysis platform, with a large number of summary features, and "publication ready" output. Its user interface is clean and intuitive to **R** users, and it has been stable, efficient, and reliable in the (fairly extensive) evaluations that we have done.

To install and run models in `secr`, you must download the package and load it in **R**.

```
> install.packages("secr")
> library(secr)
```

`secr` allows the user to simulate data and fit a suite of models with various detection functions and covariate responses. It also contains a number of helpful constructor functions for creating objects of the proper class that are recognized by other `secr` functions. The `secr` help manual can be accessed with the command:

```
> RShowDoc("secr-manual", package = "secr")
```

We note that `secr` has many capabilities that we will not cover or do so only sparingly. We encourage you to read through the manual, the extensive documentation, and the vignettes, in order to get a better understanding of what the package is capable of.

The main model-fitting function in `secr` is called `secr.fit`, which makes use of the standard **R** model specification framework with tildes. As an example, the equivalent of the basic model SCR0 is fitted as follows:

```
> secr.fit(capturedata, model = list(D ~ 1, g0 ~ 1, sigma ~ 1),
           buffer = 20000)
```

where `capturedata` is the object created by `secr` containing the encounter history data and the trap information, and the model expression `g0~1` indicates the intercept-only (i.e., constant) model. Note that we use p_0 for the baseline encounter probability parameter, which is g_0 in `secr` notation. A number of possible models for encounter probability can be fitted including both pre-defined variables (e.g., `t` and `b` corresponding to "time" and "behavior"), and user-defined covariates of several kinds. For example, to include a global behavioral response, this would be written as `g0~b`. The discussion of this (global versus local trap-specific behavioral response) and other covariates is developed more in Chapter 7. We can also model covariates on density in `secr`, which we discuss in Chapter 11. It is important to note that `secr` requires the buffer distance to be defined in meters and density will be returned as number of animals per hectare. Thus to make comparisons between `secr` and output from other programs, we will often have to convert the density to the same units.

Before we can fit a model, the data must first be packaged properly for `secr`. We require data files that contain two types of information: trap layout (location and identification information for each trap), which is equivalent to the trap deployment file (TDF) described in Section 5.9, and the capture data file containing sampling *session*, animal identification, trap occasion, and trap location, equivalent in information content to the encounter data file (EDF). Sample session can be thought of as primary period identifier in a robust design-like framework—it could represent a yearly sample or multiple sample periods within a year, each of them producing data on a closed population. We discuss "multi-session" models in more detail in Section 6.5.4, and Chapter 14.

There are three important constructor functions that help bundle your data for use in `secr`: `read.traps`, `make.capthist`, and `read.mask`. We provide a brief description of each here, and apply them to our wolverine camera trapping data in Section 6.5.2:

1. `read.traps`: This function points to an external file *or* **R** data object containing the trap coordinates, and other information, and requires specification of the type of encounter devices (described in Section 6.5.1). A typical application of this function looks like the following, invoking the `data=` option when there is an existing **R** object containing the trap information:

```
> trapfile <- read.traps(data=traps, detector="proximity")
```

2. `make.capthist`: This function takes the EDF and combines it with trap information, and the number of sampling occasions. A typical application looks like this:

```
> capturedata <- make.capthist(enc.data, trapfile, fmt="trapID",
                  noccasions=165)
```

   See `?make.capthist` for definition of distinct file formats. Specifying `fmt = trapID` is equivalent to our EDF format.
3. `read.mask`: If there is a habitat mask available (as described in Section 6.4.2), then this function will organize it so that `secr.fit` knows what to do with it. The

function accepts either an external file name (see `?read.mask` for details of the structure) or a $nG \times 2$ **R** object, say `mask.coords`, containing the coordinates of the mask. A typical application looks like the following:

```
> grid <- read.mask(data=mask.coords)
```

These constructor functions produce output that can then be used in the fitting of models using `secr.fit`.

6.5.1 Encounter device types and detection models

The `secr` package requires that you specify the type of encounter device. Instead of describing models by their statistical distribution (Bernoulli, Poisson, etc.), `secr` uses certain operational classifications of detector types including "proximity," "multi," "single," "polygon," and "signal." For camera trapping/hair snares we might consider "proximity" detectors or "count" detectors. The "proximity" detector type allows, at most, one detection of each individual at a particular detector on any occasion (i.e., it is equivalent to what we call the Bernoulli or binomial encounter process model, or model SCR0). The "count" detector designation allows repeat encounters of each individual at a particular detector on any occasion. There are other detector types that one can select such as: "polygon" detector type, which allows for a trap to be a sampled polygon (Royle and Young, 2008), which we discussed further in Chapter 15, and "signal" detector, which allows for traps that have a strength indicator, e.g., acoustic arrays (Dawson and Efford, 2009). The detector types "single" and "multi" refer to traps that retain individuals, thus precluding the ability for animals to be captured in other traps during the sampling occasion. The "single" type indicates trap that can only catch one animal at a time (single-catch traps), while "multi" indicates traps that may catch more than one animal at a time (multi-catch). These are both variations of the multinomial encounter models described in Chapter 9.

As with all SCR models, `secr` fits an encounter probability model ("detection function" in `secr` terminology) relating the probability of encounter to the distance of a detector from an individual activity center. `secr` allows the user to specify one of a variety of detection functions including the commonly used half-normal ("Gaussian"), hazard rate ("Gaussian hazard"), and (negative) exponential models. There are 12 different functions as of version 2.3.1 (see Table 7.1). The different detection functions are defined in the `secr` manual and can be found by calling the help function for the detection function:

```
> ?detectfn
```

Most of the detection functions available in `secr` contain some kind of a scale parameter that is usually labeled σ. The units of this parameter default to meters in the `secr` output. We caution that the meaning of this parameter depends on the specific detection model being used, and it should not be directly compared as a measure of home range size across models. Instead, as we noted in Section 5.4, most encounter probability models imply a model of space usage and fitted encounter models should be converted to a common currency such as implied home range area.

6.5.2 Analysis using the secr package

To demonstrate the use of the secr package, we will show how to do the same analysis of the wolverine data as in Section 5.9. To use the secr package, the data need to be formatted in a similar manner to that analysis. For example, in Section 5.9 we introduced a standard data format for the encounter data file (EDF) and trap deployment file (TDF). The EDF shares the same format as that used by the secr package, with 1 row for every encounter observation and 4 columns representing trap session ("Session"), individual identity ("ID"), sample occasion ("Occasion"), and trap identity ("trapID"). For a standard closed population study that takes place during a single season, the "Session" column is all 1's, to indicate a single primary sampling occasion. In addition to providing the encounter data file (EDF), we must tell secr information about the traps, which is formatted as a data frame with column labels "trapID," "x," and "y," the last two being the coordinates of each trap, with additional columns representing the operational state of each trap during each occasion ($1 =$ operational, $0 =$ not).

We now walk through an application of secr to the wolverine camera trapping data. To read in the trap locations and other related information, we make use of the constructor function read.traps, which also requires that we specify the detector type. The detector type is important because it determines the likelihood that secr will use to fit the model. Here, we have selected "proximity," which corresponds to the Bernoulli encounter model in which individuals are captured at most once in each trap during each sampling occasion:

```
> library(secr)
> library(scrbook)
> data(wolverine)

> traps <- as.matrix(wolverine$wtraps)
> dimnames(traps) <- list(NULL,c("trapID","x","y",paste("day",1:165,sep="")))
> traps1 <- as.data.frame(traps[,1:3])
> trapfile1 <- read.traps(data=traps1,detector="proximity")
```

Here, we note that trap coordinates are extracted from the wolverine data but we do *not* scale them. This is because secr defaults to coordinate scaling of meters, which is the extant scaling of the wolverine trap coordinates. Note that we add appropriate column labels to the "traps" data frame. An important aspect of the wolverine study is that while the camera traps were operated over a 165-day period, each trap was operational during only a portion of that period. We need to provide the trap operation information, which is contained in the columns to the right of the trap coordinates in our standard trap deployment file (TDF). Unfortunately, this is less easy to do in secr,[4] which requires an external file with a single long string of 1's and 0's, indicating the days in which each trap was operational (1) or not (0). The read.traps function will not allow for this information on trap operation if the data exists as an **R** object—instead, we can create this external file and then read it back in with read.traps using these commands:

[4]as of v. 2.3.1.

```
> hold <- rep(NA,nrow(traps))
> for(i in 1:nrow(traps)){
+     hold[i] <- paste(traps[i,4:ncol(traps)],collapse="")
+ }
> traps1 <- cbind(traps[,1:3],"usage"=hold)

> write.table(traps1, "traps.txt", row.names=FALSE, col.names=FALSE)
> trapfile2 <- read.traps("traps.txt",detector="proximity")
```

These operations can be accomplished using the function `scr2secr`, which is provided in the **R** package `scrbook`.

After reading in the trap data, we now need to create the encounter matrix using the `make.capthist` command, where we provide the capture histories in EDF format, which is the existing format of the data input file `wcaps`. In creating the capture history, we also provide the trapfile created previously, the format (e.g., here EDF format is `fmt= "trapID"`), and finally, the number of occasions.

```
# Grab the encounter data file and format it:
#
wolv.dat <- wolverine$wcaps
dimnames(wolv.dat) <- list(NULL,c("Session","ID","Occasion","trapID"))
wolv.dat <- as.data.frame(wolv.dat)
wolvcapt2 <- make.capthist(wolv.dat,trapfile2,fmt="trapID",noccasions=165)
```

We next set up a habitat mask using the 2×2 km grid we used previously in the analysis of the wolverine data and then pass the relevant objects to `secr.fit` as follows:

```
# Grab the habitat mask (2 × 2 km) and format it:
#
> gr2 <- (as.matrix(wolverine$grid2))
> dimnames(gr2) <- list(NULL,c("x","y"))
> gr2 <- read.mask(data=gr2)
#
# To fit the model we use secr.fit:
#
wolv.secr2 <- secr.fit(wolvcapt2,model=list(D ~ 1, g0 ~ 1, sigma ~ 1),
                     buffer=20000,mask=gr2)
```

We are using the "proximity detector" (model SCR0), which is the default model, so we do not need to make any specifications in the command line, except to provide the buffer size (in meters). To specify different models, you can change the default model `D~1, g0~1, sigma~1`. We provide all of these commands and additional analyses in the `scrbook` package with the function called `secr_wolverine`. Printing the output object produces the following (slightly edited):

```
> wolv.secr2

secr 2.3.1, 15:52:45 29 Aug 2012

Detector type     proximity
Detector number   37
Average spacing   4415.693 m
x-range           593498 652294 m
y-range           6296796 6361803 m
```

```
N animals       :  21
N detections    :  115
N occasions     :  165
Mask area       :  987828.1 ha

Model           :  D ~ 1 g0 ~ 1 sigma ~ 1
Fixed (real)    :  none
Detection fn    :  halfnormal
Distribution    :  poisson
N parameters    :  3
Log likelihood  :  -602.9207
AIC             :  1211.841
AICc            :  1213.253

Beta parameters (coefficients)

            beta     SE.beta        lcl        ucl
D      -9.390124  0.22636698  -9.833795  -8.946452
g0     -2.995611  0.16891982  -3.326688  -2.664535
sigma   8.745547  0.07664648   8.595323   8.895772

Variance-covariance matrix of beta parameters
                   D            g0         sigma
D       0.0512420110  -0.0004113326  -0.003945371
g0     -0.0004113326   0.0285339045  -0.006269477
sigma  -0.0039453711  -0.0062694767   0.005874683

Fitted (real) parameters evaluated at base levels of covariates
        link      estimate   SE.estimate           lcl           ucl
D        log  8.354513e-05  1.915674e-05  5.360894e-05  1.301982e-04
g0     logit  4.762453e-02  7.661601e-03  3.466689e-02  6.509881e-02
sigma    log  6.282651e+03  4.822512e+02  5.406315e+03  7.301037e+03
```

The object returned by `secr.fit` provides extensive default output when printed. Much of this is basic descriptive information about the model, the traps, or the encounter data. We focus here on the parameter estimates. Under the fitted (real) parameters, we find D, the density, given in units of individuals/hectare (1 hectare $=$ 10,000 m^2). To convert this into individuals/1,000 km^2, we multiply by 100,000, thus our density estimate is 8.35 individuals/1,000 km^2. The parameter σ is given in units of meters, and so this corresponds to 6.283 km. Both of these estimates are very similar to those obtained in our likelihood analysis summarized in Table 6.3 which, for the 2×2 km grid, returned $\hat{D} = 8.31$ with a SE of $(100{,}000 \times 1.915674e - 05) = 1.9156$ and, accounting for the scale difference (1 unit = 10,000 m in the previous analysis), $\hat{\sigma} = \sqrt{1/(2\hat{\alpha}_1)} \times 10{,}000 = 6.289$ km. The difference in the MLE between Table 6.3 and those produced by `secr` could be due to subtle differences in internal tuning of optimization algorithms, starting values, or other numerical settings. In addition, the likelihood used by `secr` is based on a Poisson prior for N (see Section 6.5.3).

6.5.3 Likelihood analysis in the `secr` package

The `secr` package does likelihood analysis of SCR models for most classes of models as developed by Borchers and Efford (2008). Their formulation deviates slightly from the binomial form we presented in Section 6.2 above (though Borchers and Efford (2008) also mention the binomial form). Specifically, the likelihood that `secr` implements is that based on removing N from the likelihood by integrating the binomial likelihood (Eq. (6.2.1)) over a Poisson prior for N—what we will call the *Poisson-integrated likelihood* as opposed to the conditional-on-N (*binomial-form*) considered previously.

To develop the Poisson-integrated likelihood we compute the marginal probability of each $\mathbf{y}_i$ and the probability of an all-zero encounter history, π_0, as before, to arrive at the marginal likelihood in the binomial-form:

$$\mathcal{L}(\boldsymbol{\alpha}, n_0|\mathbf{y}) = \frac{N!}{n!n_0!}\left\{\prod_i^n [\mathbf{y}_i|\boldsymbol{\alpha}]\right\}\pi_0^{n_0}.$$

Now, Borchers and Efford (2008) assume that $N \sim \text{Poisson}(\Lambda)$ and they do a further level of marginalization over this prior distribution:

$$\sum_{n_0=0}^{\infty}\frac{N!}{n!n_0!}\left\{\prod_i^n [\mathbf{y}_i|\boldsymbol{\alpha}]\right\}\pi_0^{n_0}\frac{\exp(-\Lambda)\Lambda^N}{N!}.$$

In Chapter 11 we write $\Lambda = \mu\|\mathcal{S}\|$ where $\|\mathcal{S}\|$ is the area of the state-space and μ is the density ("intensity") of the point process. Carrying out the summation above produces exactly this marginal likelihood:

$$\mathcal{L}_2(\boldsymbol{\alpha}, \Lambda|\mathbf{y}) = \left\{\prod_i^n [\mathbf{y}_i|\boldsymbol{\alpha}]\right\}\Lambda^n \exp(-\Lambda(1-\pi_0)),$$

which is Eq. (6.1.2) of Borchers and Efford (2008), except for notational differences. It also resembles the binomial form of the likelihood in Eq. (6.2.1) with $\Lambda^n\exp(-\Lambda\pi_0)$ replacing the combinatorial term and the $\pi_0^{n_0}$ term. We emphasize there are two marginalizations going on here: (1) the integration to remove the latent variables $\mathbf{s}$; and (2) summation to remove the parameter N. We provide a function for computing this in the `scrbook` package, called `intlik3Poisson`. The help file for that function shows how to conduct a small simulation study to compare the MLE under the Poisson-integrated likelihood with that from the binomial form.

The essential distinction between our MLE and that of Borchers and Efford (2008) as implemented in `secr` is whether you keep N in the model or remove it by integration over a Poisson prior. If you have prescribed a state-space explicitly with a sufficiently large buffer, then we imagine there should be hardly any difference at all between the MLEs obtained by either the Poisson-integrated likelihood or the binomial form of the likelihood, which retains N as an explicit parameter. There is a subtle distinction in the sense that under the binomial form, we estimate the realized population size N for the state-space whereas, for the Poisson-integrated form, we estimate the *prior* expected value which would apply to a hypothetical new study of a similar population (see Section 5.7.3).

Both models (likelihoods) assume **s** is uniformly distributed over space, but for the binomial model we make no additional assumption about N whereas we assume N is Poisson using the formulation in secr from Borchers and Efford (2008). Using data augmentation we could do a similar kind of integration but integrate N over a binomial (M, ψ) prior—which we referred to as the binomial-integrated likelihood in Section 4.2.4.

6.5.4 Multi-session models in secr

In practice we will often deal with SCR data that have some meaningful stratification or group structure. For example, we might conduct mist netting of birds on K consecutive days, repeated, say, T times during a year, or perhaps over T years. Or we might collect data from R distinct trapping grids. In these cases, we have T or R groups, which we might reasonably regard as being samples of independent populations. While the groups might be distinct sites, years, or periods within years, they could also be other biological groups such as sex or age. Conveniently, secr fits a specific model for stratified populations—referred to as *multi-session* models. These models build on the Poisson assumption which underlies the integrated likelihood used in secr (as described in Section 6.5.3). To understand the technical framework underlying multi-session models, let N_g be the population size of group g and *assume*

$$N_g \sim \text{Poisson}(\Lambda_g).$$

Naturally, we model group-specific covariates on Λ_g:

$$\log(\Lambda_g) = \beta_0 + \beta_1 C_g,$$

where C_g is some group-specific covariate such as a categorical index to the group, or a trend variable, or a spatial covariate, such as treatment effect or habitat structure, if the groups represent spatial units. Under this model, we can marginalize *all* N_g parameters out of the likelihood to concentrate the likelihood on the parameters β_0 and β_1 precisely as discussed in Section 6.5.3. This Poisson hierarchical model is the basis of the multi-session models in secr.

To implement a multi-session model (or stratified population model) in secr, we provide the relevant stratification information in the "Session" variable of the input encounter data file (EDF). If "Session" has multiple values, then a "multi-session" object is created by default and session-specific variables can be described in the model. For example, if the session has two values for males and females then we can estimate sex-specific densities, and baseline encounter probability p_0 (g_0 in secr) by just doing this:

```
> out <- secr.fit(capdata, model=list(D ~ session, g0 ~ session, sigma ~ 1),
                buffer=20000)
```

See Chapter 8 for the **R** code to set this up. More detailed analysis is given in Section 8.1 where we fit a number of different models to the wolverine camera trapping data and apply methods of model selection to obtain model-averaged estimates of density.

We can also easily implement stratified population models in the various **BUGS** engines using data augmentation (Converse and Royle, 2012; Royle and Converse, In review), which we discuss in Chapter 14.

6.5.5 Some additional capabilities of `secr`

The `secr` package has capabilities to do a complete analysis of SCR data, including model fitting, selection, and many summary analyses. In the previous sections, we've given a basic overview, and we do more in later chapters of this book. Here we mention a few of these other capabilities that you should know about as you use `secr`. Of course, you should skim through the associated documentation (`?secr`) to see more of what is available.

6.5.5.1 *Alternative observation models*

`secr` fits a wide range of alternative observation models besides the Bernoulli encounter model, including multinomial encounter models for "multi-catch" and "single catch" traps, models for sound attenuation from acoustic detection devices, and many others. We discuss many of these other methods in Chapter 9 and elsewhere in the book.

6.5.5.2 *Summary statistics*

`secr` provides a useful default summary of the data, but it also has summary statistics about animal movement including mean-maximum distance moved (the function `MMDM`). For example, see the help page `?MMDM`, which lists a number of other summary functions for `capthist` objects:

```
> moves(capthist)
> dbar(capthist)
> RPSV(capthist)
> MMDM(capthist, min.recapt = 1, full = FALSE)
> ARL(capthist, min.recapt = 1, plt = FALSE, full = FALSE)
```

The function `moves` returns the observed distances moved, `dbar` returns the average distance moved, `RPSV` produces a measure of dispersion about the home range center, and `ARL` gives the *Asymptotic Range Length* which is the asymptote of an exponential model fit to the observed range length vs. the number of detections of each individual (Jett and Nichols, 1987).

6.5.5.3 *State-space buffer*

`secr` will produce a warning if the state-space buffer is chosen too small. For example, in fitting the wolverine data as in Section 6.5.2 but with a 1,000 m buffer, we see the following warning message:

```
Warning message:
In secr.fit(wolvcapt2, model=list(D ~ 1, g0 ~ 1, sigma ~ 1), buffer=1000):
  predicted relative bias exceeds 0.01 with buffer = 1000
```

This should cause you to contemplate increasing the state-space buffer, if that is a reasonable thing to do in the specific application.

6.5.5.4 *Model selection and averaging*

`secr` does likelihood ratio tests to compare nested models using the function `LR.test`. You can create model selection tables based on AIC or AICc, using the function `AIC`, and obtain model-averaged parameter estimates using the function `model.average` (See Chapter 8 for examples).

6.5.5.5 *Population closure test*

`secr` has a population closure test with the function `closure.test`, which implements the tests of Stanley and Burnham (1999) or Otis et al. (1978). The function is used like this: `closure.test(object, SB = FALSE)`. Here, `object` is a capthist object and `SB` is a logical variable that, if `TRUE`, produces the Stanley and Burnham (1999) test.

6.5.5.6 *Density mapping and effective sample area*

`secr` produces likelihood versions of the various summaries of posterior density and effective sample area that we discussed in Chapter 5. For example, while `secr` reports estimates of the expected value of N or density directly in the summary output from fitting a model, you can use the function `region.N` to produce estimates of N for any given region. In addition, `secr` has functions for creating maps of detection contours for individuals traps, or for the entire trap array. See the functions `pdot.contour`, and `fxi.contour` for computing the two-dimensional pdf of the locations of one or more individual activity centers (as in Section 5.11.3). In the context of likelihood analysis, estimation of a random effect **s** is based on a plug-in application of Bayes' Rule. When **s** has a uniform distribution, and we use a discrete evaluation of the integral, it can be computed simply by renormalizing the likelihood:

$$[\mathbf{s}|\mathbf{y},\theta] = \frac{[\mathbf{y}|\mathbf{s},\theta]}{\sum_s [\mathbf{y}|\mathbf{s},\theta]}.$$

Any of the `intlik` functions given previously in this chapter can be easily modified to return the posterior distribution of **s** for any, or all, individuals, or an individual that is not encountered.

Effective sample area (see Section 5.12) can be calculated in `secr` using the functions `esa` and `esa.plot`).

6.5.5.7 *Covariate models*

`secr` has many capabilities for modeling covariates. It has a number of built-in models that allow certain covariates on encounter probability, which we cover to a large extent in Chapters 7 and 8. `secr` also allows covariates to be built into the density model (see Chapter 11). It has some built-in response surface models, allowing for the fitting of linear or quadratic response surfaces. This is done by modifying the density model in `secr.fit`. For example, `D` $\sim$ 1 is a constant density surface, and `D` $\sim$ `x` + `y` fits a linear response surface, etc. See the manual `secr-densitysurfaces.pdf` for details.

There are a number of ways to incorporate your own "custom" covariates into a model (as opposed to pre-specified models). One way is to use the `addCovariates` function and supply it a `mask` or `traps` object along with some "spatialdata." Or, if you have covariates at each trap location then it will extrapolate to all points on the habitat mask. There's also a method by which the user can create a function of geographic coordinates, `userDfn`, which seems to provide additional flexibility, although we haven't used this method. There is a handy function `predictDsurface` for producing density maps under the specified model for density.

6.6 Summary and outlook

In this chapter, we discussed basic concepts related to classical analysis of SCR models based on likelihood methods. Analysis is based on the so-called integrated or marginal likelihood, in which the individual activity centers (random effects) are removed from the conditional-on-**s** likelihood by integration. We showed how to construct the integrated likelihood and fit some simple models in the **R** programming language. In addition, likelihood analysis for some broad classes of SCR models can be accomplished using the **R** package `secr` (Efford, 2011a), which we provided a brief introduction to. In later chapters we provide more detailed analyses of SCR data using likelihood methods and the `secr` package.

Why or why not use likelihood inference exclusively? For certain specific models, it is more computationally efficient to produce MLEs (for an example see Chapter 12). And, likelihood analysis makes it easy to do model selection by AIC and compute standard errors or confidence intervals. However, **BUGS** is extremely flexible in terms of describing models and we can devise models in the **BUGS** language easily that we cannot fit in `secr`. For example, in Chapter 16 we consider open population models, which are straightforward to develop in **BUGS** but, so far, there is no available platform for doing MLE of such models. We can also fit models in **BUGS** that accommodate missing covariates in complete generality (e.g., unobserved sex of individuals), and we can adopt SCR models to include auxiliary data types. For example, we might have camera trapping and genetic data and we can describe the models directly in **BUGS** and fit a joint model (Gopalaswamy et al., 2012b). To do maximum likelihood estimation, we have to write a custom new piece of code for each model[5] or hope someone has done it for us. You should have some capability to develop your own MLE routines with the tools we provided in this chapter.

[5]Although we may be able to handle multiple survey methods together in `secr` using the multi-session models.

Non Print Items

Abstract: In this chapter we show how to compute maximum likelihood estimates of SCR model parameters both directly in R and using the R package secr (Efford, 2011a,b). We walk through a simulation example in which the number of animals, *N*, is known in order to provide a clear development of concepts and implementation, and show that this approach is readily extended to the more general case where *N* is unknown. Using a discrete state space gives us the flexibility to clip out areas of non-habitat (for example, bodies of water) and carry out our likelihood analysis only across habitat deemed suitable to the species. Likelihood functions for all these scenarios are provided in the accompanying R package scrbook and we apply these models to a wolverine camera trapping data set. Alternatively, we can use the R package secr, which provides a range of functionality for preparing data, fitting different models (e.g., with behavioral, time of site-specific effects) and summarizing and visualizing the results. We guide the reader through all steps associated with an analysis in secr and re-analyze the wolverine data set. While the Bayesian implementation of SCR models in the BUGS language gives us the flexibility to fit models we cannot readily fit in secr, for certain models a likelihood-based analysis is straightforward and can be more practical and efficient. We cover some of these cases in later chapters of the book.

Keywords: Akaike Information Criterion (AIC), Binomial integrated likelihood, Conditional likelihood, Effective sample area, Habitat mask, Integrated likelihood, Marginal likelihood, Numerical integration, Numerical optimization, Poisson integrated likelihood, Program DENSITY, secr, Unconditional likelihood