

Chapter 1

Likelihood Analysis of SCR Models

In this book we mainly focus on Bayesian analysis of spatial capture-recapture models. And, in the previous chapters we learned how to fit some basic spatial capture-recapture models using a Bayesian formulation of the models analyzed in BUGS engines including **WinBUGS** and **JAGS**. Despite our focus on Bayesian analysis, it is instructive to develop the basic conceptual and methodological ideas behind classical analysis based on likelihood methods and frequentist inference. In fact, simple SCR models can be analyzed fairly easily using such methods. This has been the approach taken by Borchers and Efford (2008); Dawson and Efford (2009) and related papers.

This chapter provides some conceptual and technical footing for likelihood-based analysis of spatial capture-recapture models. We recognized earlier (chapt. 4) that SCR models are versions of binomial (or other) GLMs, but with random effects i.e., GLMMs. These models are routinely analyzed by likelihood methods. In particular, likelihood analysis is based on the integrated likelihood in which the random effects are removed by integration from the likelihood. In SCR models, the random effect, \mathbf{s} , i.e., the 2-dimensional coordinate, is a bivariate random effect. In this chapter, we show 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 an **R** package **secr** (Efford, 2011) which is based on the stand-alone package **DENSITY** (Efford et al., 2004). To set the context we analyze the SCR model here when N is known because, in that case, it is precisely a GLMM and does not pose any difficulty at all. We generalize the model to allow for unknown N using both conventional ideas based on the “joint likelihood” (e.g., Borchers et al., 2002) and also using a formulation based on data augmentation. We consider likelihood analysis of SCR models in the context

of the wolverine camera trapping study (Magoun et al., 2011) we analyzed in previous chapters to compare/contrast the results.

1.1 Likelihood analysis

We noted in chapter 4 that, with N known, the basic SCR model is a type of binomial regression with a random effect. For such models we can easily 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. Conceptually, our model is a specification of the conditional-on- s model $[y|s, \theta]$ and we have a “prior distribution” for s , say $[s]$, and the marginal distribution of the data y is

$$[y|\theta] = \int_{\mathbf{s}} [y|\mathbf{s}, \theta][\mathbf{s}]d\mathbf{s}.$$

When viewed as a function of θ for purposes of estimation, the marginal distribution $[y|\theta]$ 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 Mh, and individual covariate models (see chapt. 6 from Royle and Dorazio (2008)). To develop integrated likelihood for SCR models, we first identify the conditional likelihood.

The observation model for each encounter observation y_{ij} , specified conditional on \mathbf{s}_i , is

$$y_{ij}|\mathbf{s}_i \sim \text{Bin}(K, p_{\theta}(\mathbf{x}_j, \mathbf{s}_i)) \quad (1.1)$$

where we have indicated the dependence of p_{ij} on \mathbf{s} and parameters θ explicitly. For the random effect we have $\mathbf{s}_i \sim \text{Unif}(\mathcal{S})$. The joint distribution of the data for individual i is the product of J such terms (i.e., contributions from each of J traps).

$$[\mathbf{y}_i|\mathbf{s}_i, \theta] = \prod_j \text{Bin}(K, p_{\theta}(\mathbf{x}_j, \mathbf{s}_i))$$

We note that 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 SCR0 or “multi-catch” traps.

The so-called “marginal likelihood” is computed by removing \mathbf{s}_i , by integration, from the conditional-on- s likelihood. That is, we compute:

$$[y|\theta] = \int_{\mathcal{S}} \mathcal{L}(\theta|\mathbf{y}_i|\mathbf{s}_i)g(\mathbf{s}_i)d\mathbf{s}_i$$

here $g(s) = 1/|\mathcal{S}|$.

The joint likelihood for all N individuals, assuming independence of encounters among individuals, is the product of N such terms:

$$\prod_i f(y[i, j])$$

We emphasize that two independence assumptions are explicit in this development: independence of trap-specific encounters within individuals and also independence among individuals. In particular, this would only be 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 2-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., 2011). In practice, we won’t rely on these extraneous **R** packages 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 here¹ 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 $u = 1, 2, \dots, nG$ index a grid of nG points, \mathbf{s}_u , where the area of grid cell u is A . In this case, the integrand, i.e., the marginal probability of y_{ij} , is approximated by

$$f(y_{ij}|\theta) = \frac{1}{nG} \sum_{u=1}^{nG} f(\mathbf{y}_i|\mathbf{s}_u) \quad (1.2)$$

This is a specific case of the general expression that could be used for approximating the integral for any arbitrary bivariate distribution $g(u)$. The general case is

$$[y] = \frac{A}{nG} \sum_u [y|\mathbf{s}_u][\mathbf{s}_u]$$

In the present context note that $[\mathbf{s}] = (1/A)$ and thus the grid-cell area cancels in the above expression to yield eq. 1.2.

Not surprisingly this the same answer we get if \mathcal{S} were inherently discrete, having nG unique values with equal probabilities $1/nG$, and we apply the Law of Total Probability directly to compute the marginal probability $[y|\theta]$.

1.1.1 Implementation (simulated data)

Here we will illustrate how to carryout this integration and optimization based on the integrated likelihood using simulated data (i.e., following that from Chapter 4). Using **simSCR0.fn** we simulate data for 100 individuals and a 25 trap array layed out in a 5×5 grid of unit spacing. The specific encounter model is the half-normal model. The 100 activity centers were simulated on a state-space defined by a 8×8 square within which the trap array was centered (thus the trap array is buffered by 2 units). Therefore, the density of individuals in this system is fixed at $100/64$.

In the following set of R commands we generate the data and then harvest the required data objects:

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

```

98 data<-simSCRO.fn(discard0=FALSE,sd=2013)
99 y<-data$Y
100 traplocs<-data$traplocs
101 nind<-nrow(y)
102 X<-data$traplocs
103 J<-nrow(X)
104 K<-data$K
105 Xl<-data$xlim[1]
106 Yl<-data$ylim[1]
107 Xu<-data$xlim[2]
108 Yu<-data$ylim[2]

```

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):

```

111 delta<- .2
112 xg<-seq(Xl+delta/2,Xu-delta/2,by=delta)
113 yg<-seq(Yl+delta/2,Yu-delta/2,by=delta)
114 npix<-length(xg) # assumes xg and yg same dimension here
115 area<- (Xu-Xl)*(Yu-Yl)/((npix)*(npix)) # dont need area for anything
116 G<-cbind(rep(xg,npix),sort(rep(yg,npix)))
117 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 if the state-space buffer is set at 2.

We next create an **R** function that defines the likelihood as a function of the data objects *y* and *X* which were created above but, in general, you would read these files into **R**, e.g., from a .csv file. In addition to these data objects, we need to have defined the various quantities associated with the integration grid **G** and *nG*. However, instead of worrying about making all of these objects and keeping track of them we just put that code above into the likelihood function and pass δ as an additional (optional) argument and a few other things that we need such as the boundary of the state-space over which the integration (summation) is being done. Here is one reasonably useful variation of a function for estimation based on the integrated likelihood:

```

131
132 intlik1<-function(parm,y=y,delta=.2,X=traplocs,ssbuffer=2){
133
134   Xl<-min(X[,1]) - ssbuffer
135   Xu<-max(X[,1]) + ssbuffer
136   Yl<-min(X[,2]) - ssbuffer
137   Yu<-max(X[,2]) + ssbuffer
138
139   xg<-seq(Xl+delta/2,Xu-delta/2,,length=npix)
140   yg<-seq(Yl+delta/2,Yu-delta/2,,length=npix)
141   npix<-length(xg)

```

```

142
143 G<-cbind(rep(xg,npix),sort(rep(yg,npix)))
144 nG<-nrow(G)
145 D<- e2dist(X,G)
146
147 alpha0<-parm[1]
148 alpha1<-parm[2]
149 probcap<- plogis(alpha0)*exp(-alpha1*D*D)
150 Pm<-matrix(NA,nrow=nrow(probcap),ncol=ncol(probcap))
151           # all zero encounter histories
152 n0<-sum(apply(y,1,sum)==0)
153           # encounter histories with at least 1 detection
154 ymat<-y[apply(y,1,sum)>0,]
155 ymat<-rbind(ymat,rep(0,ncol(ymat)))
156 lik.marg<-rep(NA,nrow(ymat))
157 for(i in 1:nrow(ymat)){
158   Pm[1:length(Pm)]<- (dbinom(rep(ymat[i,],nG),K,probcap[1:length(Pm)],log=TRUE))
159   lik.cond<- exp(colSums(Pm))
160   lik.marg[i]<- sum( lik.cond*(1/nG))
161 }
162 nv<-c(rep(1,length(lik.marg)-1),n0)
163 -1*( sum(nv*log(lik.marg)) )
164 }

```

The function 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- s likelihood (Eq. 1.1 above). The marginal likelihood (`lik.marg`) sums up the conditional elements weighted by $\Pr(s)$ (formula XXX above). Finally, this function assumes that K , the number of replicates, is constant for each trap. Further, it assumes that the state-space is a square. As an exercise, consider resolving these two issues by generalizing the code.

Here is the **R** command for maximizing the likelihood and saving the results into an object called `frog`. The output is a list of the following structure and these specific estimates are produced using the simulated data set:

```

186 # should take 15-30 seconds
187

```

```

188 > starting.values <- c(-2, 2)
189 > frog<-nlm(intlik1,starting.values,y=y,delta=.1,X=traplocs,ssbuffer=2,hessian=TRUE)
190 > frog
191
192 $minimum
193 [1] 297.1896
194
195 $estimate
196 [1] -2.504824  2.373343
197
198 $gradient
199 [1] -2.069654e-05  1.968754e-05
200
201 $hessian
202           [,1]      [,2]
203 [1,]  48.67898 -19.25750
204 [2,] -19.25750  13.34114
205
206 $code
207 [1] 1
208
209 $iterations
210 [1] 11

```

211 Details about this output can be found on the help page for `nlm`. We note
 212 briefly that `frog$minimum` is the negative log-likelihood value at the MLEs,
 213 which are stored in the `frog$estimate` component of the list. The hessian is
 214 the observed Fisher information matrix, which can be inverted to obtain the
 215 variance-covariance matrix using the commands:

```

216 > solve(frog$hessian)

```

217 It is worth drawing attention to the fact that the estimates are different
 218 than the Bayesian estimates reported in the previous chapter (section XYZ)!!!
 219 How can that be?! There are several reasons for this. First Bayesian inference
 220 is based on the posterior distribution and it is not generally the case that the
 221 MLE should correspond to any particular value of the posterior distribution. If
 222 the prior distributions in a Bayesian analysis are uniform, then the mode of the
 223 posterior is the MLE, but note that Bayesians almost always report posterior
 224 means and so there will typically be a discrepancy there. Secondly, we have
 225 implemented an approximation to the integral here and there might be a slight
 226 bit of error induced by that. We will evaluate that shortly. Third, the Bayesian
 227 analysis by MCMC is subject to some amount of Monte Carlo error which the
 228 analyst should always be aware of in practical situations. All of these different
 229 explanations are likely responsible for some of the discrepancy. Accounting
 230 for these, as a practical matter, we see general consistency between the two
 231 estimates.

232 To compute the integrated likelihood we used a discrete representation of
 233 the state-space so that the integral could be approximated as a summation

over possible values of \mathbf{s} with each value being weighted by its probability of occurring, which is $1/nG$ under the assumption that \mathbf{s} is uniform on the state-space \mathcal{S} . In chapter 4 we used a discrete state-space in developing a Bayesian analysis of the model in order to be able to modify the state-space in a flexible manner. Bayesian analysis requires simulation of the point process conditional on the observations, and this can be a difficult task when the state-space is continuous but has irregular geometry. Conversely, if the state-space is a regular polygon then Bayesian analysis by MCMC is possibly more efficient with a continuous state-space. We emphasize that the state-space is a part of the model. In some cases there won't be a natural choice of state space beyond "some large rectangle containing the trap grid" and, in such cases, for regular detection functions the estimate of density is invariant to the size of the state-space (i.e., the buffer) as long as it is sufficiently large. However if there are good reasons to restrict the state-space, it will tend to have an influence on the likelihood and hence AIC and so forth. As an illustration, let's do that by changing the state space here.....Use my polygon clipping stuff

In summary, we note that, for the basic SCR model, integrated likelihood is a really easy calculation when N is known. Even for N unknown it is not too difficult, and we will do that shortly. However, if you can solve the known- N problem then you should be able to do a real analysis, for example by considering different values of N and computing the results for each value and then making a plot of the log-likelihood or AIC and choosing the value of N that produces the best log likelihood or AIC. As a homework problem we suggest that the reader take the code given above and try to estimate N without modifying the code by just repeatedly calling that code for different values of N and trying to deduce the best value. Nevertheless, we will formalize the unknown- N problem shortly. We note that the software package **DENSITY** (Efford et al., 2004) implements certain types of SCR models using integrated likelihood methods. **DENSITY** has been made into an **R** package called **secr** (Efford, 2011) and we provide an analysis of some data using **secr** shortly along with a discussion of its capabilities.

1.2 MLE when N is Unknown

Here we build on the previous introduction to integrated likelihood but we consider now the case in which N is unknown. We will see that adapting the analysis based on the N -known model is really 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 which we do by tacking a row of zeros onto the encounter history matrix. In addition, we include the number of such all-zero encounter histories as an unknown parameter of the model. Call that unknown quantity n_0 . In addition, we have to be sure to include a combinatorial term 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

277 must be included in the likelihood.

278 **DETAILS NEEDED HERE**

279 To summarize, when N is unknown, the n observed encounter histories have
 280 a multinomial distribution with probabilities $\pi(i)$ and sample size N^2 . The last
 281 cell the “zero cell” is computed by carrying out the integral in expression XYZ
 282 above for the all-zero encounter history and we have to account for the fact that
 283 there are $n_0 = N - n$ such encounter histories.

284 To analyze a specific case, we read in our fake data set (simulated using the
 285 parameters given above). To set some things up in our workspace we do this:

```
286 data<-simSCR0.fn(discard0=TRUE,sd=2013)
287 y<-data$Y
288 nind<-nrow(y)
289 X<-data$traplocs
290 J<-nrow(X)
291 K<-data$K
292 Xl<-data$xlim[1]
293 Yl<-data$ylim[1]
294 Xu<-data$xlim[2]
295 Yu<-data$ylim[2]
```

296 Recall that these data were generated with $N = 100$, on an 8×8 unit
 297 state-space representing the trap locations (\mathbf{X}) buffered by 2 units. As before,
 298 the likelihood is defined in the **R** workspace as an **R** function which takes an
 299 argument being the unknown parameters of the model and additional arguments
 300 as prescribed. In particular, as before, we provide the encounter history matrix
 301 \mathbf{y} , the trap locations traplocs , the spacing of the integration grid (δ) and the
 302 state-space buffer. Here is the new likelihood function:

```
303 intlik2<-function(parm,y=y,delta=.3,X=traplocs,ssbuffer=2){
304
305   Xl<-min(X[,1]) -ssbuffer
306   Xu<-max(X[,1]) + ssbuffer
307   Yu<-max(X[,2]) + ssbuffer
308   Yl<-min(X[,2]) - ssbuffer
309
310   #delta<- (Xu-Xl)/npix
311   xg<-seq(Xl+delta/2,Xu-delta/2,delta)
312   yg<-seq(Yl+delta/2,Yu-delta/2,delta)
313   npix.x<-length(xg)
314   npix.y<-length(yg)
315   area<- (Xu-Xl)*(Yu-Yl)/((npix.x)*(npix.y))
316   G<-cbind(rep(xg,npix.y),sort(rep(yg,npix.x)))
317   nG<-nrow(G)
318   D<- e2dist(X,G)
```

² Maybe you could show an alternative simulation script to generate data using the `rmulti-`
`nom` function. This would make it a little more clear for people


```

319 alpha0<-parm[1]
320 alpha1<-parm[2]
321 n0<-exp(parm[3])
322 probcap<- plogis(alpha0)*exp(-alpha1*D*D)
323 Pm<-matrix(NA,nrow=nrow(probcap),ncol=ncol(probcap))
324 ymat<-rbind(y,rep(0,ncol(y)))
325
326 lik.marg<-rep(NA,nrow(ymat))
327 for(i in 1:nrow(ymat)){
328   Pm[1:length(Pm)]<- (dbinom(rep(ymat[i,],nG),K,probcap[1:length(Pm)],log=TRUE))
329   lik.cond<- exp(colSums(Pm))
330   lik.marg[i]<- sum( lik.cond*(1/nG) )
331 }
332 nv<-c(rep(1,length(lik.marg)-1),n0)
333 part1<- lgamma(nrow(y)+n0+1) - lgamma(n0+1)
334 part2<- sum(nv*log(lik.marg))
335 -1*(part1+ part2)
336 }
337 }

```

338 To execute this function for the data that we created with `simSCRO.fn`, we
 339 execute the following command (saving the result in our friend `frog`). This
 340 results in the usual output, including the parameter estimates, the gradient,
 341 and the numerical Hessian which is useful for obtaining asymptotic standard
 342 errors (see below):

```

343 > frog<-nlm(intlik2,c(-2.5,2,log(4)),hessian=TRUE,y=y,X=X,delta=.2,ssbuffer=2)
344 There were 50 or more warnings (use warnings() to see the first 50)
345 >
346 >
347 > frog
348 $minimum
349 [1] 113.5004
350
351 $estimate
352 [1] -2.538334  2.466515  4.232810
353
354 [. Additional output deleted .]

```

355 While this produces some **R** warnings, these happen to be harmless in this case,
 356 and we will see from the `nlm` output that the algorithm performed satisfactory
 357 in minimizing the objective function. The estimate of population size for the
 358 state-space (using the default state-space buffer) is

```

359 > nrow(y)+exp(4.2328)
360 [1] 110.9099

```

361 Which differs from the data-generating value ($N = 100$) as we might expect. We
 362 usually will present an estimate of uncertainty associated with this MLE which

we can obtain by inverting the Hessian. Note that $Var(\hat{N}) = n + Var(\hat{n}_0)$. Since we have parameterized the model in terms of $\log(n_0)$ we use a delta approximation 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
```

1.2.1 Exercises

1. Run the analysis with different state-space buffers and comment on the result.

2. Conduct a brief simulation study using this code by simulating 100 data sets and obtain the MLEs for each data set. Do things seem to be working as you expect?

3. Further extensions: It should be straightforward to generalize the integrated likelihood function to accommodate many different situations. For examples, if we want to include more covariates in the model we can just add stuff to the object `probcap`, and add the relevant parameters to the argument that gets passed to the main function. For the simulated data, make up a covariate by generating a Bernoulli covariate (“trap type” perhaps baited or not baited) randomly and try to modify the likelihood to accommodate that.

4. We would probably be interested in devising the integrated likelihood for the full 3-d encounter history array so that we could include temporally varying covariates. This is not difficult but naturally will slow down the execution substantially. The interested reader should try to expand the capabilities of this basic **R** function.

1.2.2 Integrated Likelihood using the model under data augmentation

Note that this likelihood analysis is based on the standard likelihood in which N (or n_0) is an explicit parameter. This is usually called the “joint likelihood” or “unconditional likelihood”. We could also express the joint likelihood using data augmentation, replacing the parameter N with ψ (e.g., Royle and Dorazio, 2008, sec. xyz). We don’t go into detail here, but we do note that the likelihood under data augmentation is a zero-inflated binomial mixture precisely as an occupancy type model (Royle, 2006). The interested reader could adapt the material from Royle and Dorazio (2008) with the **R** code given above for the likelihood and implement the likelihood analysis based on the model under data augmentation. While we can carryout likelihood analysis of models under data augmentation, we primarily advocate data augmentation for Bayesian analysis.

1.2.3 Extensions

There are other types of covariates of interest: behavioral response, sex-specificity of parameters and all of these things. Some of these can be added directly to the likelihood if the covariate is fixed and known for all individuals captured or not. This excludes most covariates but it does include behavioral response. Sex-specificity is more difficult since sex is not known for uncaptured individuals. Trap-specific covariates such as trap type or status, or time-specific covariates such as date, are relatively easy to deal with (we leave these as exercises). We apply these various models in Chapter XXXX. To analyze such models, we do Bayesian analysis of the joint likelihood facilitated by the use of data augmentation. For covariates that are not fixed and known for all individuals, it is hard to do MLE for these based on the joint likelihood as we have developed above. Instead what people normally do is 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, 2011, see sec. 1.5).

1.3 Classical model selection and assessment

In most analyses, one is interested in choosing from among various potential models. A good thing about classical analysis based on likelihood is we can do rote application of AIC without thinking about anything. With distance as a covariate (e.g., distance sampling) this is usually applied to some arbitrary selection of distance functions. We don't recommend this. Given there is hardly ever (if at all) a rational science-based reason for choosing some particular distance function we believe that this standard approach will invariably lead to over-fitting. The fact that AIC is easy to compute does not mean that it should be abused in such fashions. Further discussion is made in chapters XYZ.

Goodness-of-fit In many analyses based on likelihood methods it is possible to cook-up fit statistics for which asymptotic distributions are known. In general, however, applied statisticians tend to adopt bootstrapping based on heuristically appealing fit statistics. An omnibus global GoF statistic is not so obvious but we can apply bootstrapping principles to SCR models directly which we discuss in chapter XYZ. Bayesian goodness-of-fit is almost always addressed with Bayesian p-values or some other posterior predictive check (REF XXX). Thus the approach whether Bayesian or classical is the same. We identify a fit statistic, we do a bootstrap (classical) or a Bayesian p-value. Royle et al. (2011) decomposed the fit problem into separate evaluations of the CSR hypothesis and the encounter process model. We discuss all of this in Chapter XYZ.

1.4 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 K), which requires also a modification to the construction of the likelihood. In addition, we had to accommodate that the state-space is a general rectangle, and we included a line in the code to compute the state-space area which we apply below for computing density. The more general function (`intlik3`) is given in the **R** package. It has a general purpose wrapper named `scr` which has other capabilities too.

The data were read into our R session and manipulated using the following commands. Note that we use the utility **R** function `SCR23darray.fn` which we defined in chapt. 4.

```
> wcaps<-source("wcaps.R")$value
> wtraps<-source("wtraps.R")$value
> K.wolv<-apply(wtraps[,4:ncol(wtraps)],1,sum)
>
> xx<-SCR23darray.fn(wcaps,ntraps=37,nperiods=165)
> y.wolv<- apply(xx,c(1,3),sum)
> traplocs.wolv<-wtraps[,2:3]
> traplocs.wolv<-traplocs.wolv/10000
>
> frog<-nlm(intlik3,c(-1.5,1.2,log(4)),hessian=TRUE,y=y.wolv,K=K.wolv,X=traplocs.wolv,
There were 23 warnings (use warnings() to see them)
> frog

$minimum
[1] 220.4355

$estimate
[1] -2.817570  1.255112  3.599040

$gradient
[1] -6.274309e-06  2.146722e-05 -1.045566e-05

$hessian
      [,1]      [,2]      [,3]
[1,] 37.687931 -11.852236  4.688911
[2,] -11.852236 30.846144 -9.199113
[3,]  4.688911 -9.199113 13.050428

$code
[1] 1
```

```

482 $iterations
483 [1] 12
484
485 > exp(3.599)*sqrt(solve(frog$hessian)[3,3])
486 [1] 11.41059
487 >
488
489

```

490 We obtained the MLEs for a state-space buffer of 2 (standardized units) and
491 for integration grid with spacing $\delta = .3, .2, .1, .05$. The MLEs for these 4 cases
492 including the relative runtime are given in Table 1.1.

Table 1.1: Run time and MLEs for different integration grid resolutions.

δ	runtime	Estimates		
		α_0	θ	$\log(n_0)$
0.30	8.4	-2.819786	1.258468	3.569731
0.20	22.6	-2.817610	1.254757	3.583690
0.10	99.0	-2.817570	1.255112	3.599040
0.05	403.0	-2.817559	1.255281	3.607158

493 We see the results change only slightly as the fineness of the integration
494 grid increases. Conversely, the runtime on the platform of the day for the 4
495 cases increases rapidly which, as we have suggested before, could probably be
496 regarded in relative terms, across platforms, for gaging the decrease in speed
497 as the fineness of the integration grid increases. The effect of this is that we
498 anticipate some numerical error in approximating the integral on a mesh of
499 points, and that error increases as the coarseness of the mesh increases.

500 In section ?? back in chapt. 4 we used a discrete representation of the state-
501 space in order to have control over its extent and shape, for example so that we
502 could clip out “non-habitat”. Clearly that formulation of the model is relevant
503 to the use of integrated likelihood in the sense that such a representation of
504 the state-space underlies the computation of the integral. Thus, for example,
505 we could easily compute the MLE of parameters under some model with a
506 restricted state-space merely by creating the required state-space at whatever
507 grid resolution is desired, and then feed that state-space into the likelihood
508 evaluation above. The **R** function `scr` which comes with the **R** package for this
509 book accommodates an arbitrary state-space fashioned in this manner, as well
510 as state-spaces created by polygons or GIS shapefiles³.

511 Next we studied the effect of the state-space buffer on the MLEs, using a
512 fixed $\delta = .2$ for all analyses. We used state-space buffers of 1 to 4 units stepped
513 by .5. This produced the following results, given here are the state-space buffer,
514 area of the state-space, the MLE of N for the prescribed state-space and the
515 corresponding MLE of density:

³to be completed!

	ssbuff	Ass	Nhat	Dhat
[1,]	1.0	66.98212	37.73338	0.5633352
[2,]	1.5	84.36242	46.21008	0.5477567
[3,]	2.0	103.74272	57.00617	0.5494956
[4,]	2.5	125.12302	69.03616	0.5517463
[5,]	3.0	148.50332	82.17550	0.5533580
[6,]	3.5	173.88362	96.44018	0.5546249
[7,]	4.0	201.26392	111.83524	0.5556646

The estimates of D stabilize rapidly and the incremental difference is within the numerical error associated with approximating the integral. The results suggest that wolverine density is around 0.56 individuals per 100 km^2 (recall that a state-space unit is $10 \times 10 km$). This is about 5.6 individuals per thousand km^2 which compares with XYZ-lookup-XYZ reported in Royle et al. (2011) based on a clipped state-space as described in section XYZ (XYZ chapter 4 XYZ).

1.4.1 Exercises

1. Compute the 95% confidence interval for wolverine density, somehow.
2. Compute the AIC of this model and modify `intlik3` to consider alternative link functions (at least one additional) and compare the AIC of the different models and the estimates. Comment.

1.5 Program 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 (2011) has also released an **R** package named **secr**, that contains many of the functions within **DENSITY** but also incorporates new models and features. Here, we will focus on **secr** as it will continue to be developed, contains more functionality and is based in **R**. 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. **secr** uses the standard **R** model specification framework using tildes. E.g., the model command is `secr.fit` and is generally written as

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

where we have `g0~1` indicating the intercept model. To include covariates, this would be written as `g0~b` where b is a behavioral response covariate. Possible

553 predictors for detection probability include both pre-defined variables (e.g., `t`
 554 and `b` corresponding to “time” and “behavior”), and user-defined covariates of
 555 several kinds. The discussion of covariates is developed in chapter XX(8)⁴

556 Before we can fit the models, the data must first be entered into `secr`. Two
 557 input files are required: trap layout (location and identification information for
 558 each trap) and capture data (e.g., sampling session, animal identification, trap
 559 day, and trap location). SECR requires that you specify the trap type, the
 560 two most common for camera trapping/hair snares are proximity detectors and
 561 count detectors. The ‘proximity’ detector type allows, at most, one detection
 562 of each individual at a particular detector on any occasion. The count detector
 563 designation allows repeat encounters of each individual at a particular detector
 564 on any occasion. There are other detector types that one can select such as:
 565 ‘polygon’ detector type which allows for a trap to be a sampled polygon, e.g.,
 566 scat surveys, and ‘signal’ detector which allows for traps that have a strength
 567 indicator, e.g., acoustic arrays. The detector types single and multi can be
 568 confusing as multi seems like it would appropriate for something like a camera
 569 trap, but instead these two designations refer to traps that retain individuals,
 570 thus precluding the ability for animals to be captured in other traps during
 571 the sampling occasion. The single type indicates trap that can only catch one
 572 animal at a time, while multi indicates traps that may catch more than one
 573 animal at a time. For a full review of the detector types, one should look at the
 574 help manual, which can be accessed in R after installing the SECR package by
 575 using the command:

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

577 As with all of the `scr` models, `secr` fits a detection function relating the probabil-
 578 ity of detection to the distance of a detector from an individual activity center.
 579 `secr` allows the user to specify one of a variety of detection functions including
 580 the commonly used half-normal, hazard rate, and exponential. There are 12 dif-
 581 ferent functions, but some are only available for simulating data, and one should
 582 take caution when using different detection functions as the interpretation of
 583 the parameters, such as sigma, may not be consistent across formulations. The
 584 different detection functions are defined in the `secr` manual and can be found
 585 by calling the help function for the detection function:

```
586 > ?detectfn
```

587 It is useful to note that `secr` requires the buffer distance to be defined in meters
 588 and density will be returned as number of animals per hectare. Thus to make
 589 comparisons between `secr` and other models, we will often have to convert the
 590 density to the same units. Also, note that sigma is returned in units of meters.

591 5

⁴Beth: does `secr` fit a local trap-specific response or just a global behavioral response?

⁵One question: SECR only ever reports sigma. What exactly is sigma? It is a scale parameter of a detection function and all detection functions have a scale parameter. But in what sense is this sigma parameter related to home range diameter? Efford doesnt explain this, does he? In some sections in chapter 4 or possibly 6 we get into this issue.

1.5.1 Analysis using the secr package

To demonstrate the use of the secr package, we will show how to do the same analysis on the wolverine study as shown in section 4.6. To use the secr package, the data need to be formatted in a similar but slightly different manner than we use in WinBUGS. After installing the secr package, we first have to read in the trap locations and other related information, such as if the trap is operational during a sampling occasion. The secr package reads in the trap data through a command called `read.traps`, which requires the detector type as input. The detector type is important because it will determine the likelihood that secr will use to fit the model. Here, we have selected proximity since individuals are captured at most once in each trap during each sampling occasion.

```

593 > traps= read.csv(wtraps.csv)
594 > colnames(traps)[1:3]<- c("trapID","x", "y") #name the first 3 columns
595 # to match the secr nomenclature
596
597
598 > trapfile <- read.traps(data = traps, detector = "proximity")
599
600
601
602

```

After reading in the data, we now need to create the encounter matrix or array. The secr package does this through the use of the `make.caphist` command, where we provide the capture histories in raw data format (each line contains the session, identification number, occasion, and trap id for only 1 individual). This is the format that was shown in the data input file `wcaps`, and we only need a line or two to organize the data into the order that the `make.caphist` command wants. In creating the capture history, we provide also the trapfile with the trap information, and the format (e.g., here `fmt= trapID`) so that secr knows how to match the encounters to the trap, and finally, we provide the number of occasions.

```

608 > wolv.dat <- wcaps[,c(2, 3, 1)]
609 #NEED TO UPDATE THIS WHEN I GET THE FILES,
610 ### I JUST GUESSED AT THE CODE, BUT WOULD LIKE TO TRY IT.
611 > wolv.dat <- cbind(rep(1, dim(wolv.dat)[1]), wolv.dat)
612 > colnames(wolv.dat) <- c("Session", "ID", "Occasion", "trapID")
613
614
615 > wolvcapt=make.caphist(wolv.dat, trapfile, fmt = "trapID", noccasions = 165)
616
617
618
619
620
621
622
623
624

```

Calling the `secr.fit` command, will run the model. We are using the basic model (SCR0), so we do not need to make any specifications in the command line except for the providing the buffer size (in m). To specify different models, you can change the default `D~1`, `g0~1`, `sigma~1`, which the interested reader can do with very little difficulty.

```

625 > wolv.secr=secr.fit(wolvcapt, model = list(D~1, g0~1, sigma~1), buffer = 20000)
626
627
628
629
630
631 > wolv.secr
632

```



```

633
634 secr.fit( capthist = wolvcapt, buffer = 20000, binomN = 1 )
635 secr 2.0.0, 18:26:39 05 Jul 2011
636
637 Detector type      proximity
638 Detector number   37
639 Average spacing   4415.693 m
640 x-range           593498 652294 m
641 y-range           6296796 6361803 m
642 N animals        : 21
643 N detections      : 115
644 N occasions       : 165
645 Mask area        : 1037069 ha
646
647 Model             : D~1 g0~1 sigma~1
648 Fixed (real)      : none
649 Detection fn      : halfnormal
650 Distribution       : poisson
651 N parameters      : 3
652 Log likelihood    : -746.754
653 AIC               : 1499.508
654 AICc              : 1500.920
655
656 Beta parameters (coefficients)
657      beta      SE.beta      lcl      ucl
658 D      -9.749576 0.23027860 -10.200913 -9.298238
659 g0     -4.275736 0.15846104 -4.586313 -3.965158
660 sigma  8.699202 0.07868944  8.544973  8.853430
661
662 Variance-covariance matrix of beta parameters
663      D      g0      sigma
664 D      0.053028233 0.000546922 -0.005226926
665 g0     0.000546922 0.025109900 -0.005885213
666 sigma -0.005226926 -0.005885213 0.006192027
667
668 Fitted (real) parameters evaluated at base levels of covariates
669      link      estimate SE.estimate      lcl      ucl
670 D      log 5.831941e-05 1.360973e-05 3.713638e-05 9.158548e-05
671 g0     logit 1.371121e-02 2.142902e-03 1.008756e-02 1.861207e-02
672 sigma  log 5.998124e+03 4.727205e+02 5.140849e+03 6.998355e+03

```

673 Under the fitted (real) parameters, we find D , the density, given in units
674 of individuals/hectare (1 hectare = 100 m²). To convert this into individu-
675 als/1000km², we multiply by 100000, thus our density estimate is 5.83 individ-
676 uals/1000 km². Sigma is given in units of meters, to convert to kilometers, we
677 divide by 1000, which puts sigma at 5.99 km. Both of these estimates are very

similar to those provided in section 4.6 for the buffer size equal to 20 km. As an exercise, run this analysis for 30 and 40 km buffers and compare those found in section 4.6 under **WinBUGS**. NOTE: The function `secr.fit` will return a warning when the buffer size appears to be too small. This is useful particularly with the different units being used between programs and packages.

1.6 Summary and Outlook

In this chapter, we showed that classical analysis of SCR models based on likelihood methods is a relatively simple proposition. Analysis is based on the so-called integrated likelihood in which the individual activity centers (random effects) are removed from the conditional-on- \mathbf{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 in the software package **DENSITY** or in the equivalent **R** library `secr` which we provided an illustration of here. In later chapters we provide more detailed analyses of SCR data using the `secr` package.

To compute the integrated likelihood we have to precisely describe the state-space of the underlying point process. In practice, this leads to a buffer around the trap array. We note that this is not really a buffer strip in the sense of Wilson et al. (XYZ) which is a feature of the analysis but it is somewhat more general here. In particular, it establishes the support of the integrand which we generally require to compute any integral. It might be that the integrand itself is finite even if the support is infinity but that may or may not be the case depending on the choice of detection function. As a practical matter then, it will typically be the case that, while estimates of N increase with the size of the buffer, estimates of density stabilize. This is not a feature of the classical methods based on using model M_0 or model M_h and buffering the trap array.

Why or why not use likelihood inference exclusively? For certain specific models, it is probably more computationally efficient to produce MLEs. However, **WinBUGS** is extremely flexible in terms of describing models, although it sometimes can be quite slow. We can devise models in **WinBUGS** easily that we cannot fit in `secr`. E.g., random individual effects of various types (see next chapter), we can handle missing covariates in complete generality and seamlessly, and impose arbitrary distributions on random variables. Moreover, models can easily be adapted to include auxiliary data types. For example, we might have camera trapping and genetic data and we can describe the models directly in **WinBUGS** and fit a joint model. For the MLE we have to write a custom new piece of code for each model or hope someone has done it for us. Later we consider open population models which are straightforward to develop in **WinBUGS** but, so far, there is no available platform for doing MLE although we imagine one could develop this. Another thing that is more conceptual here is non-CSR point processes (see chapter XYZ) and generating predictions of how many individuals have home range centers in any particular polygon.

721 Basic benefits of Bayesian analysis have been discussed elsewhere (Chapter 2?
722 BPA book? Link and Barker?) and we believe these are compelling. On the
723 other hand, likelihood analysis makes it easy to do model-selection by AIC.
724 Goodness-of-fit is probably no more difficult or easy under either paradigm (see
725 next chapter?).

726 In summary, basic SCR models are easy to implement by either likelihood
727 or Bayesian methods but we feel that the typical user will realize much more
728 flexibility in model development using existing platforms for Bayesian analysis.
729 While these tend to be slow (sometimes excruciatingly slow), this will probably
730 not be an impediment in most problems, especially at some near point in the
731 future. Since we spent a lot of time here talking about specific technical details
732 on how to implement likelihood analysis of SCR models, we provided a corre-
733 sponding treatment in the next chapter on how to devise MCMC algorithms
734 for SCR models. This is a bit more tedious and requires more coding, but is
735 not technically challenging (except perhaps to develop highly efficient algorithms
736 which we don't excel at).

Bibliography

- Borchers, D. and Efford, M. (2008), “Spatially explicit maximum likelihood methods for capture–recapture studies,” *Biometrics*, 64, 377–385.
- Borchers, D. L., Buckland, S. T., and Zucchini, W. (2002), *Estimating animal abundance: closed populations*, vol. 13, Springer Verlag.
- Dawson, D. and Efford, M. (2009), “Bird population density estimated from acoustic signals,” *Journal of Applied Ecology*, 46, 1201–1209.
- Efford, M. (2004), “Density estimation in live-trapping studies,” *Oikos*, 106, 598–610.
- (2011), “secr-spatially explicit capture–recapture in R,” .
- Efford, M., Dawson, D., and Robbins, C. (2004), “DENSITY: software for analysing capture–recapture data from passive detector arrays,” *Animal Biodiversity and Conservation*, 217–228.
- Genz, A. S., Meyer, M. R., Lumley, T., and Maechler, M. (2007), “The adapt Package. R package version 1.0-4,” .
- Hahn, T., Bouvier, A., and Kieu, K. (2011), “Package ‘R2Cuba’ R package version 1.0-6,” .
- Magoun, A. J., Long, C. D., Schwartz, M. K., Pilgrim, K. L., Lowell, R. E., and Valkenburg, P. (2011), “Integrating motion-detection cameras and hair snags for wolverine identification,” *The Journal of Wildlife Management*, 75, 731–739.
- Royle, J. (2006), “Site occupancy models with heterogeneous detection probabilities,” *Biometrics*, 62, 97–102.
- Royle, J. and Dorazio, R. (2008), *Hierarchical modeling and inference in ecology: the analysis of data from populations, metapopulations and communities*, Academic Press.
- Royle, J. A., Magoun, A. J., Gardner, B., Valkenburg, P., and Lowell, R. E. (2011), “Density estimation in a wolverine population using spatial capture–recapture models,” *The Journal of Wildlife Management*, 75, 604–611.