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 WinBUGS. Despite our focus on Bayesian analysis, it is instructive to develop the basic conceptual and methodological ideas behind classical analysis based on likelihood methods. In fact, simple SCR models can be analyzed fairly easily using classical likelihood methods. This has been the approach taken by Borchers and Efford (2008), Dawson and Efford (2009) and related papers.

In this chapter we provide some conceptual and technical footing for likelihood-based analysis of spatial capture-recapture models. We recognized earlier (Section xxxx) that SCR models are versions of binomial (or other) GLMs with random effects i.e., GLMMs, which 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, 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 standalone 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 book XYZ) 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. XYZ) 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 maximize the integrated likelihood numerically. It is worth analyzing this really simple case in order to understand the underlying mechanics and basic concepts which are directly relevant to the manner in which many capture-recapture models are classically analyzed, such as model Mh, and individual covariate models (see Chapter 6 from Royle and Dorazio 2008).

The observation model specified conditional on $\mathbf{s}_i$ is

$$y[i,j]|s[i] \; Bin(K, p[i,j])$$

where p_{ij} is a function of the individual-specific random effect, $\mathbf{s}_i$, which has a Unif($\mathcal{S}$) prior distribution. The conditional-on-$\mathbf{s}$ likelihood for the encounter history data for individual i is the product of J such terms:

$$L(\mathbf{y}_i|\mathbf{s}_i) = \prod_j \text{Bin}(K, p_{ij})$$

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

$$f(y[i,j]) = \int_{\mathcal{S}} f(\mathbf{y}_i|s[i])g(s[i])ds[i]$$

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 various multi-dimensional integration routines including adapt (REF) and Rcuba (REF XYZ). In practice, we wont 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 S

and explicitly evaluate the integrand at each point. Let u=1,2,,nG index a grid of points where the area of grid cell u is a[u] = a for a regular grid where each cell has the same area. In this case, the trapezoidal rule for approximating the integral yields

$$f(y[i,j]) = sum_{u=1}^{nG} f(\mathbf{y}_i|u).g(u) * area$$

Where a is the constant area of the grid cells used to approximate the continuous region. This is a general expression that could be used for approximating the integral for any arbitrary bivariate distribution g(u). In the present context note that g(u) = (1/area(S)) = 1/[nG*a] and thus the grid-cell area a cancels in the above expression and we have

$$f(y[i,j]) = sum_{u=1}^{nG} f(\mathbf{y}_i|u)(1/nG)$$

Which not surprisingly is the same answer we get if S were inherently discrete, having nG unique values with equal probabilities 1/nG. Then the marginal probability of $y[I,j]$, i.e., by the Law of Total probability, is precisely that last expression.

1.1.1 Implementation (simulated data)

Here we will illustrate how to carry-out this integration and optimization based on the integrated likelihood using simulated data set (i.e., following that from Chapter 4, section XYZ).

From before, we simulated data for 100 individuals and a 25 trap array layed out in a 5 x 5 grid of unit spacing. The specific encounter model is the logit model given above and we used this code to simulate data used in subsequent analyses. The 100 activity centers were simulated on a state-space defined by a 8 x 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.

```
# create 5 x 5 grid of trap locations with unit spacing
traplocs<- cbind(sort(rep(1:5,5)),rep(1:5,5))
Dmat<-e2dist(traplocs,traplocs)
ntraps<-nrow(traplocs)

# define state-space of point process. (i.e., where animals live).
# "delta" just adds a fixed buffer to the outer extent of the traps.

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

N<-100   # population size
```

```
K<- 20    # number nights of effort

sx<-runif(N,Xl,Xu)    # simulate activity centers
sy<-runif(N,Yl,Yu)
S<-cbind(sx,sy)
D<- e2dist(S,traplocs)  # distance of each individual from each trap

alpha0<- -2.5      # define parameters of encounter probability
sigma<- 0.5        #
alpha1<- 1/(2*sigma*sigma)
probcap<- plogis(alpha0 - alpha1*D)    # probability of encounter

# now generate the encounters of every individual in every trap
y<-matrix(NA,nrow=N,ncol=ntraps)
for(i in 1:nrow(y)){
   y[i,]<-rbinom(ntraps,K,probcap[i,])
}
```

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

```
delta<- .2
xg<-seq(Xl+delta/2,Xu-delta/2,by=delta)
yg<-seq(Yl+delta/2,Yu-delta/2,by=delta)
npix<-length(xg)  # assumes xg and yg same dimension here
area<- (Xu-Xl)*(Yu-Yl)/((npix)*(npix)) # dont need area here
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 0.2 which produces a 20 x 20 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., as done in the previous chapter for the analysis using WinBUGS. In addition to these data objects, we need to have defined the various quantities associated with the integration grid created just previously – G, nG, and area. 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 delta 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:

```

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

```
142
143 Xl<-min(X[,1])-ssbuffer
144 Xu<-max(X[,1])+ ssbuffer
145 Yu<-max(X[,2])+ ssbuffer
146 Yl<-min(X[,2])-ssbuffer
147
148 xg<-seq(Xl+delta/2,Xu-delta/2,,length=npix)
149 yg<-seq(Yl+delta/2,Yu-delta/2,,length=npix)
150 npix<-length(xg)
151 area<- (Xu-Xl)*(Yu-Yl)/((npix)*(npix))
152 G<-cbind(rep(xg,npix),sort(rep(yg,npix)))
153 nG<-nrow(G)
154 D<- e2dist(X,G)
155
156 alpha0<-parm[1]
157 alpha1<-parm[2]
158 probcap<- plogis(alpha0-alpha1*D)
159 Pm<-matrix(NA,nrow=nrow(probcap),ncol=ncol(probcap))
160 n0<-sum(apply(y,1,sum)==0) # all zero encounter histories
161 ymat<-y[apply(y,1,sum)>0,] # encounter histories with at least 1 detection
162 ymat<-rbind(ymat,rep(0,ncol(ymat)))
163 lik.marg<-rep(NA,nrow(ymat))
164 for(i in 1:nrow(ymat)){
165 Pm[1:length(Pm)]<- (dbinom(rep(ymat[i,],nG),K,probcap[1:length(Pm)],log=TRUE))
166 lik.cond<- exp(colSums(Pm))
167 lik.marg[i]<- sum( lik.cond*(1/nG))
168 }
169 nv<-c(rep(1,length(lik.marg)-1),n0)
170 -1*( sum(nv*log(lik.marg)) )
171 }
172
```

173 Remarks about the likelihood function: 1. The function accepts as input the
174 encounter history matrix, y, the trap locations, X, and the state-space buffer.
175 This allows us to vary the state-space buffer and easily evaluate the sensitivity
176 of the MLE to the size of the state-space. 2. We have a peculiar handling of
177 the encounter history matrix y. In particular, we remove the all-zero encounter
178 histories from the matrix and tack-on a single all-zero encounter history as the
179 last row which then gets weighted by the number of such encounter histories
180 (n0). This is a bit long-winded and strictly unnecessary when N is known, but we
181 did it this way because the extension to the unknown-N case is now transparent
182 (as we demonstrate in the following section). 3. The matrix Pm holds the
183 log-likelihood contributions of each encounter frequency for each possible state-
184 space location of the individual. 4. The log contributions are summed up and
185 the result exponentiated on the next line, producing lik.cond, the conditional-
186 on-s likelihood. (formula XXX above). 5. The marginal likelihood (lik.marg)

sums up the conditional elements weighted by Pr(s) (formula XXX above). 6. 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 data set provided in the Online Supplement):

```
# should take 15-30 seconds
> starting.values <- c(a0=-2, a1=1) # just to make things clear
> frog<-nlm(intlik1,starting.values,y=y,delta=.1,X=traplocs,ssbuffer=2,hessian=TRUE)

Warning message:
In nlm(intlik1, c(-2, 1), y = y, delta = 0.1, X = traplocs, ssbuffer = 2,  :
  NA/Inf replaced by maximum positive value

> frog
$minimum
[1] 182.6882

$estimate
[1] -2.623456  1.658335

$gradient
[1] -1.867945e-06  1.587044e-06

$hessian
          [,1]      [,2]
[1,]  69.16351 -72.13976
[2,] -72.13976 108.22528

$code
[1] 1

$iterations
[1] 12
```

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

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

It is worth drawing attention to the fact that the estimates are different than the Bayesian estimates reported in the previous chapter (section XYZ)!!! How can that be?! There are many 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 mode of the posterior is the MLE but note that Bayesians almost always report posterior means and so there will typically be a discrepancy there. Secondly, we have implemented an approximation to the integral here and there might be a slight bit of error induced by that. We should evaluate that now..

To compute the integrated likelihood we used a discrete representation of 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 s is uniform on the state-space 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 wont 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, lets 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 dont 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 n0. 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 N choose n ways to realize a sample of size n. The combinatorial term involves the unknown n0 and thus it must be included in the likelihood (we have already done that in the previous likelihood construction even though we didnt need it).

To summarize, when N is unknown, the n observed encounter histories have a multinomial distribution with probabilities pi(i) and sample size N[1]. The last cell the zero cell is computed by carrying out the integral in expression XYZ above for the all-zero encounter history and we have to account for the fact that there are n0 = N-n such encounter histories.

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

```
 simSCR0.fn( blah blah blah)

> y<-read.csv("ind_by_trap.csv")[,-1]
> y<-as.matrix(y)
>
> X<-read.csv("traplocs.csv")[,-1]
> X<-as.matrix(X)
>
```

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

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

Xl<-min(X[,1]) -ssbuffer
```

[1]Maybe you could show an alternative simulation script to generate data using the rmultinom function. This would make it a little more clear for people

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

#delta<- (Xu-Xl)/npix
xg<-seq(Xl+delta/2,Xu-delta/2,delta)
yg<-seq(Yl+delta/2,Yu-delta/2,delta)
npix.x<-length(xg)
npix.y<-length(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)

alpha0<-parm[1]
alpha1<-parm[2]
n0<-exp(parm[3])
probcap<- plogis(alpha0-alpha1*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(t(Pm)%*%rep(1,nrow(probcap)))
lik.marg[i]<- sum( lik.cond*(1/nG) )
}
nv<-c(rep(1,length(lik.marg)-1),n0)
part1<- lgamma(nrow(y)+n0+1) - lgamma(n0+1)
part2<- sum(nv*log(lik.marg))
 -1*(part1+ part2)
}
```

To execute this function for the data that we just read into the workspace, we execute the following command (saving the result in our friend frog).

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

This results in the usual output, including the parameter estimates, the gradient, and the numerical Hessian which is useful for obtaining asymptotic standard errors (omitted here).

```
> frog
$minimum
[1] 181.4657

$estimate
```

```
[1] -2.629342  1.790849  3.997268

[. Additional output deleted .]
```

The estimate of population size for the state-space (using the default state-space buffer) is

```
> nrow(y)+exp(3.997)
[1] 113.4346
```

Which differs from the data-generating truth (N=100) as we might expect.

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 probcap there, and add the relevant parameters to the argument that gets passed to the intlik 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 n0) 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 psi. See Royle et al. (2007); Royle and Dorazio (2008, distance sampling example), etc.. Briefly, we note that the likelihood under data augmentation looks like a zero-inflated binomial mixture precisely as an occupancy type model (see Royle 2006). We think 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. Despite that we can carry-out likelihood analysis of models under data augmentation, we primarily advocate data augmentation for Bayesian analysis.

1.2.3 Extensions and Classical model selection and assessment

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 XYZ; see chapter XYZ). .

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 dont 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 knkown. In general, however, applied statisticians tend to adopt bootstraping 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 discussion 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.

1.3 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 (see Online Supplement). 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 online supplement (shall we provide it here?).

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 section XYZ.XYZ.

```
> 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
>
> nlm(lik,c(-1.5,1.2,log(4)),hessian=TRUE,y=y.wolv,K=K.wolv,X=traplocs.wolv,
delta=.1,ssbuffer=2)$estimate

[1] -1.646692  3.128712  3.761974
```

We optained the MLEs for a state-space buffer of 2 (standardized units) and for integration grid with spacing delta = .3, .2, .1, .05. The MLES for these 4 cases are as follows:

```
> nlm(intlik3,c(-1.5,1.2,log(4)),hessian=TRUE,y=y.wolv,K=K.wolv, X=traplocs.wolv,delta=

[1] -1.654535  3.108126  3.723244

> nlm(lik,c(-1.5,1.2,log(4)),hessian=TRUE,,delta=.2,ssbuffer=2)$estimate

 [1] -1.643023  3.133985  3.749195

> nlm(lik,c(-1.5,1.2,log(4)),hessian=TRUE,,delta=.1,ssbuffer=2)$estimate

[1] -1.646692  3.128712  3.761974

> nlm(lik,c(-1.5,1.2,log(4)),hessian=TRUE,,delta=.05,ssbuffer=2)$estimate

[1] -1.647191  3.128308  3.769455
```

We see the results change only slightly as the fineness of the integration grid increases. Conversely, the runtime on the platform of the day for the 4 cases was approximately 19s, 40s, 169s, and 723s which as we have suggested before could probably be regarded as relative to each other, across platforms, for gaging the decrease in speed as the fineness of the integration grid increases.

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

```
### REDO ANALYSES

     ssbuff       Ass      Nhat      Dhat
[1,]    1.0  66.98212  42.18630 0.6298144
[2,]    1.5  84.36242  52.12473 0.6178667
[3,]    2.0 103.74272  64.03329 0.6172317
[4,]    2.5 125.12302  77.36792 0.6183348
[5,]    3.0 148.50332  91.99139 0.6194568
[6,]    3.5 173.88362 107.87487 0.6203855
[7,]    4.0 201.26392 125.01359 0.6211426
```

The estimates of D stabilize rapidly[2] and the results suggest that wolverine density is around 0.62 individuals per 100 square km (recall that a unit is 10 x 10 km). This is about 6.2 individuals per thousand square km which compares with XYZ reported in Royle et al. (2011) based on a clipped state-space as described in section XYZ.

In a later chapter analysis by MLE is done with an irregular state-space. Therefore we will have to extend the R function again to accept as possible input a pre-defined state-space grid.

1.3.1 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. [should we do that here?].

1.4 Program DENSITY and the R package secr

DENSITY is a software program developed by Efford et al. (2004) for fitting spatial capture-recapture models based mostly on classical maximum likelihood

[2] not very convincing

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 itin 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 covariate. Possible predictors for detection probability include both pre-defined variables (e.g., t and b corresponding to time and behavior), and user-defined covariates of several kinds. The discussion of covariates is developed in chapter XX(8).

Before we can fit the models, the data must first be entered into SECR. Two input files are required: trap layout (location and identification information for each trap) and capture data (e.g., sampling session, animal identification, trap day, and trap location). SECR requires that you specify the trap type, the two most common for camera trapping/hair snares are proximity detectors and count detectors. The 'proximity' detector type allows, at most, one detection of each individual at a particular detector on any occasion. 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, e.g., scat surveys, and 'signal' detector which allows for traps that have a strength indicator, e.g., acoustic arrays. The detector types single and multi can be confusing as multi seems like it would appropriate for something like a camera trap, but instead these two designations 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, while multi indicates traps that may catch more than one animal at a time. For a full review of the detector types, one should look at the help manual, which can be accessed in R after installing the SECR package by using the command:

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

As with all of the scr models, SECR fits a detection function relating the probability of detection 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, hazard rate, and exponential. There are 12 different functions, but some are only available for simulating data, and one should take caution when using different detection functions as the interpretation of the parameters, such as sigma, may not be consistent across formulations. The different detection functions are defined in the secr manual and can be found by calling the help function for the detection function:

```
> ?detectfn
```

It is useful 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 other models, we will often have to convert the density to the same units. Also, note that sigma is returned in units of meters. [3]

1.4.1 Analysis using 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.

```
>traps= read.csv(wtraps.csv)
>colnames(traps)[1:3]<- c("trapID","x", "y")  #name the first 3 columns
# to match the secr nomenclature

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

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.capthist 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.capthist 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

[3] One 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.

knows how to match the encounters to the trap, and finally, we provide the number of occasions.

```
>wolv.dat <- wcaps[,c(2, 3, 1)]   #NEED TO UPDATE THIS WHEN I GET THE FILES, I JUST GU
>wolv.dat <- cbind(rep(1, dim(wolv.dat)[1], wolv.dat)
>colnames(wolv.dat) <- c("Session", "ID", "Occasion", "trapID")

>wolvcapt=make.capthist(wolv.dat, trapfile, fmt = "trapID", noccasions = 165)
```

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.

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

>wolv.secr

secr.fit( capthist = wolvcapt, buffer = 20000, binomN = 1 )
secr 2.0.0, 18:26:39 05 Jul 2011

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       :  1037069 ha

Model           :  D~1 g0~1 sigma~1
Fixed (real)    :  none
Detection fn    :  halfnormal
Distribution    :  poisson
N parameters    :  3
Log likelihood  :  -746.754
AIC             :  1499.508
AICc            :  1500.920

Beta parameters (coefficients)
           beta    SE.beta       lcl       ucl
D     -9.749576 0.23027860 -10.200913 -9.298238
g0    -4.275736 0.15846104  -4.586313 -3.965158
sigma  8.699202 0.07868944   8.544973  8.853430
```

```
Variance-covariance matrix of beta parameters
                  D           g0        sigma
D      0.053028233  0.000546922 -0.005226926
g0     0.000546922  0.025109900 -0.005885213
sigma -0.005226926 -0.005885213  0.006192027

Fitted (real) parameters evaluated at base levels of covariates
       link     estimate  SE.estimate          lcl          ucl
D       log 5.831941e-05 1.360973e-05 3.713638e-05 9.158548e-05
g0    logit 1.371121e-02 2.142902e-03 1.008756e-02 1.861207e-02
sigma   log 5.998124e+03 4.727205e+02 5.140849e+03 6.998355e+03
```

Under the fitted (real) parameters, we find D, the density, given in units of individuals/hectare (1 hectare = 100 m2). To convert this into individuals/1000km2, we multiply by 100000, thus our density estimate is 5.83 individuals/1000 km2. Sigma is given in units of meters, to convert to kilometers, we 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 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.

1.5 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-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 (and other packages such as ADMB) 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 M0 or model Mh 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. Basic benefits of Bayesian analysis have been discussed elsewhere (Chapter 2? BPA book? Link and Barker?) and we believe these are compelling. On the other hand, likelihood analysis makes it easy to do model-selection by AIC. Goodness-of-fit is probably no more difficult or easy under either paradigm (see next chapter?).

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