Simulating Realistic Acoustic Capture-Recapture Data

Melissa Bather

Department of Statistics

The University of Auckland

Supervisor: Dr. Ben Stevenson

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, The University of Auckland, 2023.

Contents

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Ben Stevenson for patiently guiding me throughout this entire project, answering my many questions, and encouraging me. I would also like to thank Professor James Russell for teaching me about, and sparking my interest in, estimating animal population densities for the first time in 2021. Of course, I also have to thank my amazing mum Noeline for being my Number 1 Supporter. Thank you also to all my friends, family, and work colleagues for their encouragement.

Chapter 1

Introduction

Estimating animal population densities is foundational to biology, ecology, and conservation within New Zealand and worldwide. Assessments of population densities underpin our understanding of biodiversity patterns and are fundamental to the formulation of effective conservation strategies. Numerous methods of density estimation have been employed by researchers since the 19th century, the earliest being *capture-recapture*, sometimes known as *mark-recapture*.

1.1 Capture-Recapture

Capture-recapture for ecological applications was first used by Danish marine biologist C.G. Johannes Petersen in 1896, to estimate plaise population sizes (Southwood & Henderson 2000). Traditionally, to conduct capture-recapture, researchers visit a study area in which they set traps, capture live animals, tag them so they may be uniquely identified, and then release them back into the study area. After enough time has passed so that the tagged individuals are likely to have been redistributed among the rest of the population, traps are set again, and a new sample of individuals is caught.

After these two trapping occasions, there are a number of different estimators available which may be used to estimate population size. One such method of note is the Lincoln-Petersen estimator. The estimator is as follows:

$$\hat{N} = \frac{nK}{k} \tag{1.1}$$

where $\hat{N}$ is the estimated number of animals in the population, n is the number of animals captured and tagged on the first occasion, K is the number of animals captured and tagged on the second occasion, and k is the number of animals captured on the second occasion that were also caught and tagged on the first occasion.

In its simplicity, this traditional form of capture-recapture is not without limitations. One of its key assumptions—that the probability of capture is uniform across all individuals within a study—is often not true. Various factors such as sex, natural variability, and learned attitudes towards the traps after trapping occasions can make some animals more or less likely to be caught than others. While probability of capture may occasionally be constant in some contexts, broadly assuming that detection probabilities are always constant without just cause can result in negatively biased estimators of N (Otis et al. 1978).

Moreover, it is common sense that the closer a trap is to an animal's home, the greater chance it has of being captured. This method also relies on populations being closed, and does not address the prospect that animals may enter or leave the population via birth, death, immigration, or emigration, during the study period. Most importantly, this method does not account for spatial effects.

1.1.1 The need to account for spatial factors

While capture-recapture can be used to derive an estimate of the total number of animals, N, for population estimates to be usefully applied to ecological or conservation efforts, population density is often a more helpful measure to have. However, to calculate density, D, a well-defined study area, A, is necessary in addition to N. In order to define the boundaries of A, animal movement within space must be considered. Therefore, density cannot be accurately estimated without accounting for space.

1.2 Spatial Capture-Recapture

A remedy to the issues arising from capture-recapture for density estimation is to use spatial capture-recapture (SCR) instead (Efford 2004, Borchers & Efford 2008, Royle & Young 2008). The SCR process involves detecting and then re-detecting animals in a well-defined region in both *space* and *time*. These detections don't necessarily have to be conducted with physical traps and tagging; other techniques such as camera traps (Green et al. 2020) (when individuals can be uniquely identified from patterns or markings), hair snares (Proctor et al. 2010), or even fecal DNA samples (Owen-Ramos et al. 2022) can be used as detection methods.

When working with SCR, each animal inhabiting A is assumed to have a subregion within A where the individual spends most of its time engaging in its usual activities such as foraging, sleeping, or mating. The central point within this region is referred to as its *activity centre* or *home-range centre*. Each individual i is associated with a pair of spatial coordinates $\boldsymbol{s}_i = (s_{i,x}, s_{i,y})$ which represent the location of its activity centre. By default, the distribution of activity centres is assumed to be from a homogeneous Poisson process, which is not always valid in reality. As one might expect, animals often cluster inhomogeneously in areas where the habitat is more suitable, which could depend on factors such as forest coverage, terrain, or resource availability. Figure 1.1 demonstrates the difference between such distributions of activity centres. Fortunately, spatial covariates can be incorporated into SCR models in order to use an inhomogeneous Poisson point process to describe the distribution of activity centres (Borchers & Efford 2008).

1.2.1 The detection function

Instead of assuming uniform detection probability, SCR allows a relationship between detection probability and the distance between an individual and the detector.

For K detectors, the locations of the detectors are denoted by $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_K)$, where $\boldsymbol{x}_k$ is the location of the kth detector. Let p_{ij} be the probability of individual i being detected at trap j. This can be modelled using a detection function $g(d_{ij})$ where d_{ij} is the distance between the activity centre of individual i and the location of detector j.

The most common detection function used to model p_{ij} is the *half-normal* function:

$$p_{ij} = g(d_{ij}) = g_0 \times \exp\left(\frac{-d_{ij}^2}{2\sigma^2}\right) \tag{1.2}$$

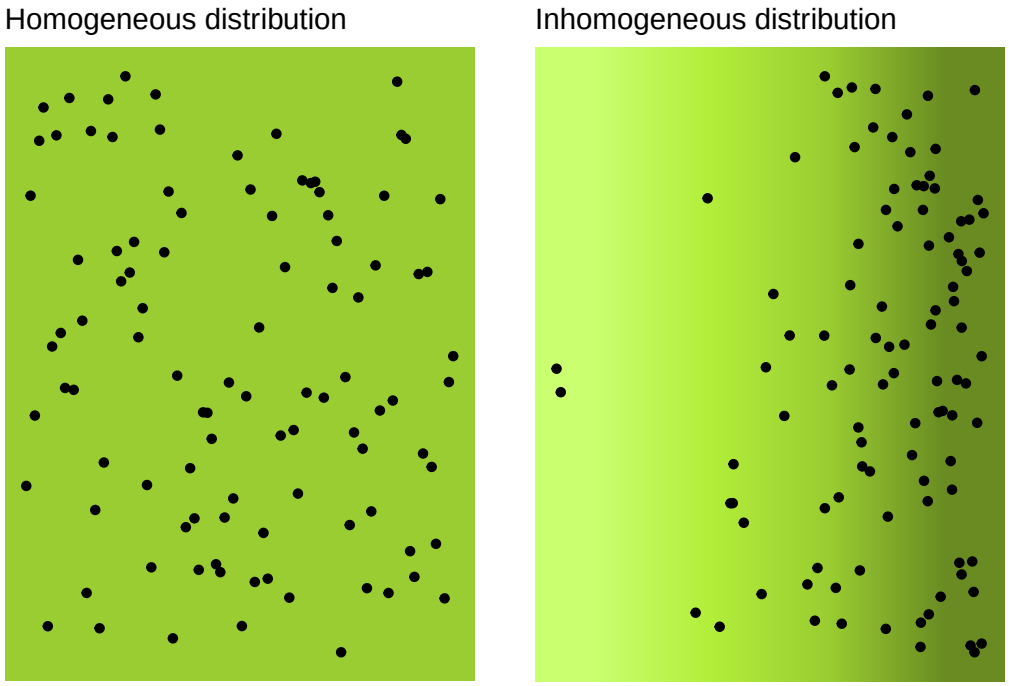


Figure 1.1: Comparison of homogeneously and inhomogeneously distributed activity centres. Here, the inhomogeneous distribution only depends on the x-coordinate of the area and is generated by the equation $\log(y) = 5x + 1$. In reality, the distribution may depend on a number of spatial or session covariates. For example, we may have a distribution given by the equation $\log(y) = \beta_0 + \beta_1 \times year + \beta_2 \times forest$, where 'year' and 'forest' are session and spatial covariates, and β_0, β_1, and β_2 are coefficients.

where,

- g_0 is the baseline detection probability when the distance between the detector and the animal's activity centre is 0
- σ is the parameter denoting the decline in detection probability as the distance between the detector and the animal's activity centre increases.

Figure 1.2 illustrates the shape of a half-normal detection function when $g_0 = 1$; Figures 1.3 and 1.4 demonstrate how changing g_0 and σ, respectively, changes this general shape of the detection function.

These parameters may be constant values or may depend on spatial covariates, such as proximity to a particular geographical feature, or session covariates, such as weather conditions or time. For example, hibernating animals such as bears may be less active during winter months and therefore less likely to be detected during a session that takes place during winter.

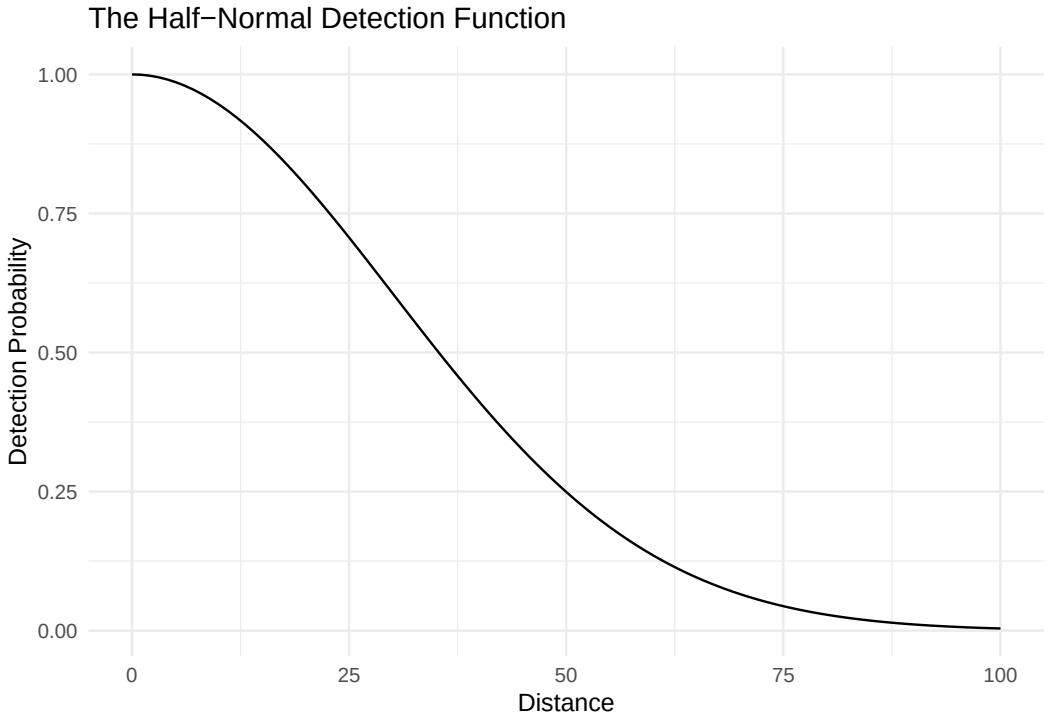


Figure 1.2: The half-normal detection function.

Other detection functions include hazard rate, exponential, or compound half-normal (Efford et al. 2023).

When collecting SCR data, capture histories are typically stored as binary matrices, where the rows represent unique individuals that have been captured, and the columns usually represent trapping occasions. A complete capture history for individual i is of the form $\boldsymbol{\omega}_i = (\omega_{i1}, ..., \omega_{iK})$, where $\omega_{ij} = 1$ when individual i is detected on the jth occasion, and $\omega_{ij} = 0$ otherwise. It follows that the sum of the elements in $\boldsymbol{\omega}_i$ returns the number of occasions on which individual i was detected, and the sum of the jth elements over capture histories of all individuals returns the number of unique individuals detected on occasion j. When in matrix form, this corresponds to row sums and column sums respectively.

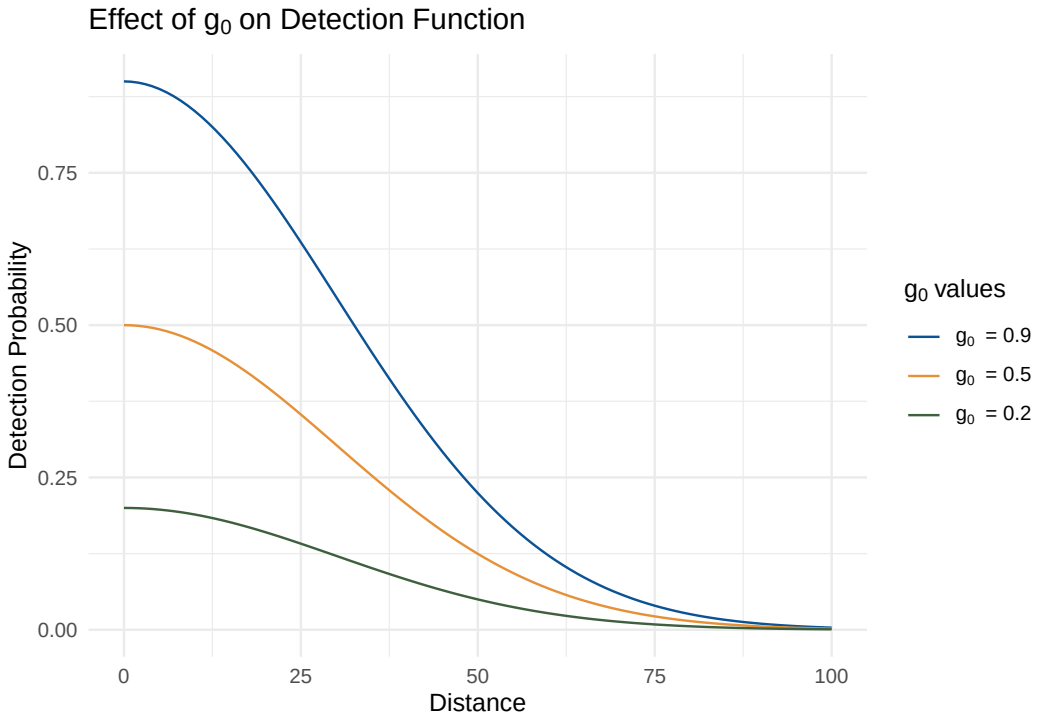


Figure 1.3: The half-normal detection function changes with different values of g_0

1.2.2 Likelihood function for parameter estimation

A common approach to parameter estimation for SCR is maximum likelihood (Borchers & Efford 2008). When n is the number of animals detected, and θ is the vector of parameters (D, g_0, σ), we derive the likelihood function by using the joint distribution of n and the capture histories of the n detected individuals:

$$\begin{aligned} L(\theta) &= f(n, \omega_1, ..., \omega_n) \\ &= f(n) f(\omega_1, ..., \omega_n | n) \\ &= f(n) \prod_{i=1}^{n} f(\boldsymbol{\omega}_i) \end{aligned}$$

The marginal distribution $f(\boldsymbol{\omega}_i)$ is derived through the integration of the joint distribution of $\boldsymbol{\omega}_i$ and $\boldsymbol{s}_i$ with respect to $\boldsymbol{s}_i$:

$$\begin{aligned} f(\boldsymbol{\omega_i}) &= \int_{R^2} f(\boldsymbol{\omega}_i, \boldsymbol{s}_i) d\boldsymbol{s}_i \\ &= \int_{R^2} f(\boldsymbol{\omega}_i | \boldsymbol{s}_i) f(\boldsymbol{s}_i) d\boldsymbol{s}_i \end{aligned} \tag{1.3}$$

By the nature of the SCR methodology, we have information about where and when individual animals were captured or detected, but we do not have direct information about them when they have not been captured, nor do we have information about the individuals that are never detected. This means

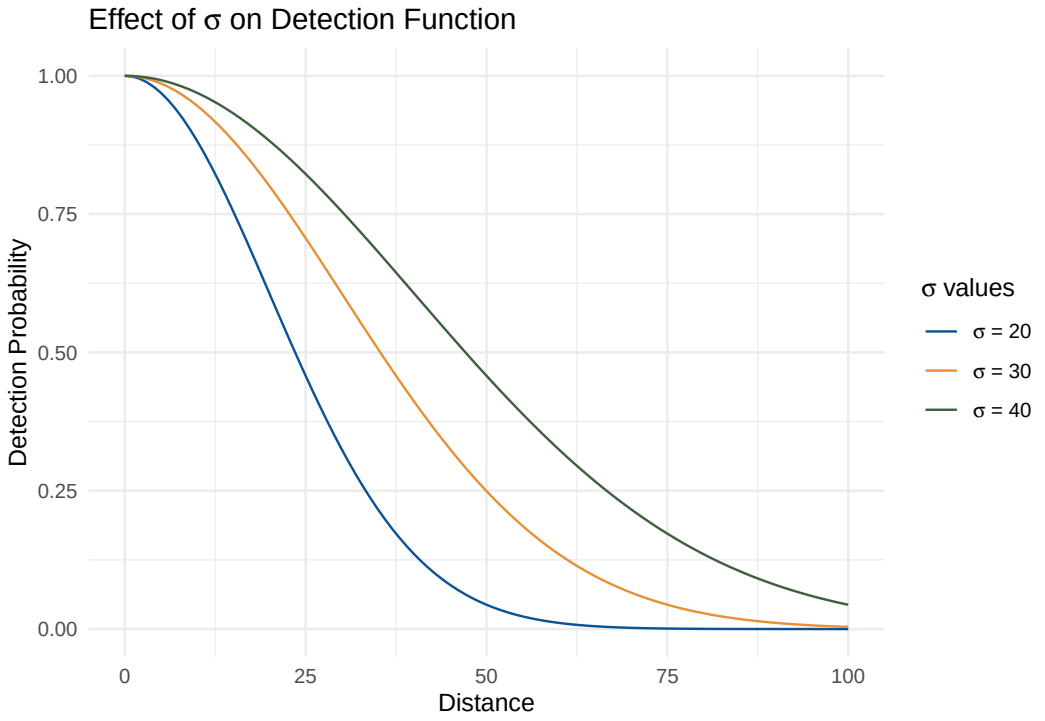


Figure 1.4: The half-normal detection function changes with different values of σ.

that the true activity centres, $\boldsymbol{s}_i$, are latent variables. For proper estimation of g_0 and σ, the locations of the activity centres must be inferred, and it is necessary to integrate over all possible locations where there could feasibly be an activity centre. This generally begs the need for a spatial boundary around the study area, defining the range of plausible locations for activity centres. In practice, we approximate these integrals by dividing the area contained within this boundary into a grid of cells, where each cell represents a potential location of an activity centre. This grid is known as a *mask*.

Finally, maximum likelihood estimation can be used to estimate g_0 and σ, along with parameters of the point process used to model the activity centre locations.

1.3 Limitations of SCR

While the visual or physical observations used in SCR can work well with some animals, they can be difficult to use effectively with other animals, particularly species that are very large, very small, or good at camouflaging. For example, the Cape Peninsula Moss Frog *Arthroleptella lightfooti* is a tiny frog that inhabits the Western Cape Province in South Africa and is extremely difficult to find (Measey et al. 2017). Monitoring animals of this size by direct observation can be time-consuming, expensive, and inaccurate. In cases like this, a more appropriate approach may be to detect animals' vocalisations to estimate population density. Models using this acoustic approach are known as acoustic spatial capture-recapture models (ASCR).

1.4 Acoustic Spatial Capture-Recapture

Acoustic spatial capture-recapture removes the need for researchers to visually observe or physically trap animals; animal vocalisations are recorded instead. The first acoustic capture-recapture methods were developed by Efford, Dawson, and Borchers (2009).

The following are some scenarios when acoustic surveys may be a more appropriate or preferable approach for estimating density:

1. Animals are good at camouflaging, e.g. the northern spotted owl (Appel et al. 2023).
2. It is not practical to physically trap animals due to their size, e.g. the aforementioned Cape Peninsula Moss Frog (Measey et al. 2017), or Cuvier's beaked whales (Barlow et al. 2021).
3. It is not practical to physically trap animals due to their habitat, e.g. gibbons that live in tall trees where it is difficult to trap or detect them (Kidney et al. 2016).
4. Visual or physical detections are possible, but more limited than if acoustic data were used, e.g. gibbons living in dense canopies where foliage can obscure them (Kidney et al. 2016).

As a consequence of the above scenarios, acoustic detections can sometimes be cheaper and/or easier than physical or visual methods.

A further benefit of acoustic data is that it can potentially be collected passively over a long period of time, which makes it possible to collect a very large amount of data cheaply and without too much effort. This is generally not possible to do when setting physical traps, as not only is it inhumane to keep animals in traps over long periods of time, it inherently does not work in a capture-recapture framework. Even for other kinds of detectors, such as hair snares (Proctor et al. 2010), physical evidence still must be collected, which does not allow for passive data collection.

After placing microphones or audio detectors in a study area, sound recording can be done continuously over time, and then software or machine learning techniques can be used to distinguish animal calls over background noise with good accuracy.

One such example of this is in *Using passive acoustic monitoring to estimate northern spotted owl landscape use and pair occupancy* (Appel et al. 2023), where six weeks of recordings were able to be made, and convolutional neural networks were used to identify spotted owl calls, proceeded by logistic regression which was able to distinguish male from female calls.

To be considered 'passive', an acoustic survey does not necessarily need to take place over a long time period—in fact it can occur over a very short time period, especially if animals call very frequently—but when long term data is needed then this can be beneficial.

Alternatively, active acoustic detection involves researchers emitting sounds in order to elicit animal responses. This approach has often been used to estimate fish population densities (Zwolinski et al. 2009). However, manufacturing animal calls within their habitat may have unknown consequences both within the population of interest and with other species occupying the same habitat. Emitting fake animal calls could, for example, attract more predators to the area which can disturb the natural balance of its ecosystem. Therefore, passive surveys are generally preferable.

Since individuals may vocalise or 'call' more than once, the detection function for acoustic methods estimates the probability of detecting a single vocalisation rather than an individual animal. Therefore, the estimate here is call density, $D\mu$, where μ is the average call rate of the animal, rather than population density. The latent location, in this case, is now a physical location of a *call* rather than the activity centre of an *animal*. This also means that it cannot be assumed that vocalisations come from a Poisson process; once an animal has vocalised in one place within A, if it vocalises again it is likely to be either in the same place or within very close proximity to its original place, therefore vocalisation locations are not independent.

Fortunately, using a Poisson process still results in an unbiased estimate of call density, where $D_c = D\mu$ (Stevenson et al. 2015). Call rate data can be collected separately from the density study to estimate μ. It then follows that $\hat{D} = \frac{\hat{D}_c}{\hat{\mu}}$.

Unlike physical traps, where a trap may only capture one animal per occasion, acoustic detectors are known as 'proximity' detectors (Marques et al. 2013) so individuals can be detected by multiple detectors on a single occasion. Additionally, acoustic data encompass more than simply whether or not an animal was heard by a particular acoustic detector. Upon an acoustic detection, researchers may be able to ascertain the signal strength of a call, an estimated bearing or distance of the caller from the microphone, and/or the received arrival time of the signal. It must be noted that measurement error parameters must be estimated for these auxiliary data types; for example, estimated bearings may not point exactly to the location of an individual responsible for a call. On account of measurement error, when multiple detectors identify the same call, the bearings estimated by each detector often do not converge to a single point, as demonstrated in Figure 1.5.

Since acoustic data are not restricted to a binary outcome ('detected' or 'not detected'), and multiple calls can be made by each animal, it makes more sense to represent detections in a long form table, as opposed to a binary matrix. ASCR data typically have columns for the session ID, animal ID, occasion, and trap ID. It may also include columns for observed bearings of the animals from the detectors or for received signal strength of the detections, depending on how much information researchers were interested in and able to acquire. Table 1.1 demonstrates what this data may look like in table form, and Figure 1.5 illustrates what Table 1.1 may look like when plotted over a study area.

Table 1.1: Example acoustic capture data

session	ID	occasion	trap	bearing
1	1	1	1	3.07
1	1	1	5	4.60
1	1	1	7	0.28
1	2	1	3	3.77
1	2	1	5	0.64
1	3	1	9	2.60
2	1	1	1	2.41
2	1	1	5	5.23
2	3	1	6	3.50
2	3	1	8	0.70
2	3	1	9	5.80
2	4	1	1	0.14
2	4	1	2	5.90

As with regular SCR, a detection function such as the half-normal detection function, with the same parameters g_0 and σ, may be used to model detection probability.

1.5 Software for Creating SCR and ASCR Models

There currently exist four notable R packages for the analyses described above: `openCR` (Efford 2022), `secr` (Efford et al. 2023), `ascr` (Stevenson 2022), and the currently in-development package `acre` (Stevenson 2023).

`openCR` and `secr` serve very similar purposes; indeed they were both authored by Murray Efford. `openCR` can generally be thought of as a more limited version of `secr`. Some examples of features of

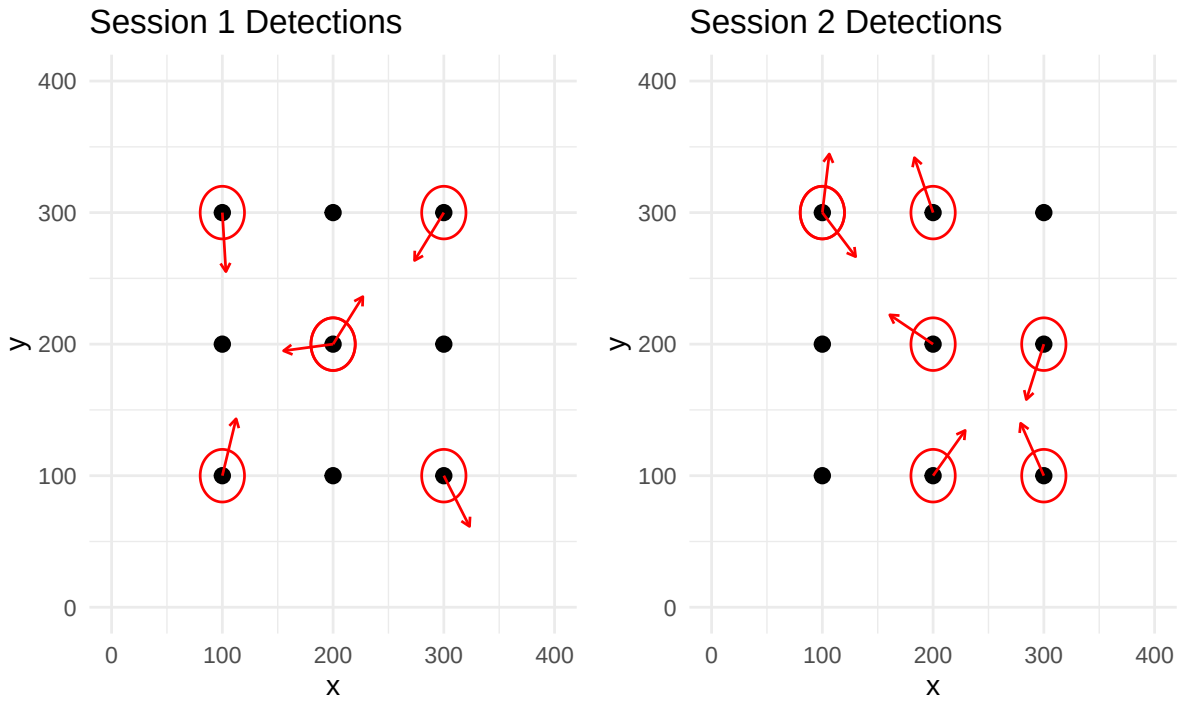


Figure 1.5: Sample acoustic detections and their observed bearings from the detectors.

`secr` that `openCR` lacks include overdispersion adjustment and bootstrap confidence intervals (Efford 2022).

Taken directly from the its vignette, the `secr` package can be described is as follows: *"Functions to estimate the density and size of a spatially distributed animal population sampled with an array of passive detectors, such as traps, or by searching polygons or transects. Models incorporating distance-dependent detection are fitted by maximizing the likelihood. Tools are included for data manipulation and model selection."* (Efford et al. 2023).

In addition to the R package itself, `secr` also has its own R Shiny web interface, where users can upload their capture history and trap data and have models of their choosing created for them. This addition allows users who are less confident programmers to use the package, while users who are comfortable with programming may still take advantage of the greater control and optimisation options that come with using the package directly with R. This flexibility is what sets the use of R packages apart from other types of capture-recapture software.

While `secr` provides functions necessary for regular spatial capture-recapture, it does not allow users to work with acoustic data. Here lies the specific value in the `ascr` package, authored by Ben Stevenson; users may input long form acoustic detection history data for analyses that require it, while still providing the option to perform regular SCR when required. Like `secr`, `ascr` also has an R Shiny web interface. The in-development package, `acre`, is a rewrite of `ascr` by the same author, with the addition of some important new features; these include:

- Model covariates are not restricted to the use of spatial covariates only; `acre` introduces the ability to include session-level and trap-level covariates.
- Information about spatial covariates does not need to be known for all mask cells; the user can

input spatial covariates at whichever locations they have been measured, which do not necessarily have to be within mask locations.

- All model parameters have the ability to vary with covariates—such as detection function parameters and auxiliary data measurement error parameters—as opposed to restricting this variation to density only.

These new additions allow different detectors or sessions to have different detection functions to one another.

When rewriting a package, assuming that the output of the original package is a 'source of truth', it is essential that the output of the new package match that of the original package when performing the same computations, regardless of how efficiently or effectively it may do so. Therefore, it is crucial that the outputs of the functions available in `acre` be validated against the outputs of the corresponding functions available in `ascr`.

Although it is possible to use existing data to compare the outputs of `acre` to the outputs of `ascr`, these data likely do not cover all scenarios researchers may encounter when it comes to study design and spatial and session covariates, especially when considering how the field is continuously evolving. Testing the package against a broad range of data that encompass a variety of study scenarios is important to ensure not only that the output is correct, but to identify potential edge cases where perhaps the usage of the package is unclear or not feasible. Using simulated data provides a practical and efficient way to test `acre` with many datasets in many different contexts, and appears the natural approach to test and validate `acre`'s output.

1.6 Dissertation Overview

There are multiple goals for this dissertation:

- Create a reusable simulation process to create realistic example acoustic capture-recapture data. This process must be able to change based on different parameter values and spatial and session covariates.
- Use the simulation to create a variety of example acoustic capture-recapture datasets, and then use these data with the functions available in `acre` and `ascr`.
- Identify issues that come about when using the simulated data with `acre` functions.
- Compare the output when using `acre` functions to the output when using `ascr` functions on the same data and flag any discrepancies.

Chapter 2

Simulating Acoustic Data with Realistic Spatial Effects

The simulation was written using the R programming language. Functions were written in individual scripts for readibility and cleanliness. All code can be found at the following GitHub repository: https://github.com/melbather/simulating-acr-data. The simulation was loosely based on acoustic surveys of gibbons (Kidney et al. 2016, McGrath et al. (in press)). To understand the simulation and its constituent functions, one must first understand the considerations made to accommodate the many covariates that can change between ASCR studies.

2.1 Considerations

When simulating acoustic capture-recapture data, there is no shortage of factors to consider. Spatial and session covariates can affect detection probabilities as well as animal densities at different points within the study area. Sessions can differ between one another in both or one of time and space.

2.1.1 Distance to the nearest village

The first consideration made within the simulation is spatial in nature: the distance of each mask cell to the centre of its nearest 'village'. In this context, a 'village' is synonymous with any geographical feature where there would typically be fewer animals living or dwelling, and for the purpose of this simulation any feature of such a nature will be referred to as a 'village'. By using 'distance to the nearest village' as a spatial covariate, density is allowed to vary with proximity to such features.

In regards to this covariate, `acre` holds a key advantage over `ascr`: village locations can be input into `acre` functions directly and distances are calculated on behalf of the user. In contrast, `ascr` requires the user to manually calculate the distance to the nearest village for every mask cell, a task which can be computationally expensive, especially if the number of mask cells used is large.

2.1.2 Other spatial covariates

Next, other spatial features within the study area that can affect densities are considered. For the purpose of this simulation, the chosen features included are:

- Areas of forest coverage
- Protected areas
- Altitude

'Forest coverage' and 'protected area' are the chosen terms for geographical features where animals are more likely to live, but in reality this could be any habitat feature that is desirable for the species of interest. Each mask cell is considered to be 'forest' or not, and 'protected area' or not. 'Altitude' is simply the altitude across the study area in metres, which must be accounted for as it can affect suitability of habitat for some species. Each mask cell is assigned a value for its altitude.

2.1.3 Spatial correlation in covariate simulation

Mask cells are not independent of one another, therefore it is not realistic to randomly assign spatial covariates to mask cells. For example, if a particular cell is considered to be 'forest', then it is likely that its adjacent cells are also 'forest'. Consequently, to produce realistic data that consider spatial covariates appropriately, spatial correlation is needed to simulate the variables above.

The method used to achieve this spatial correlation is random number generation from the multivariate normal distribution, with mean 1. The R package `mvtnorm` is used for this generation (Genz & Bretz 2009). The covariance matrix used in this distribution, Σ, is derived using the following equation:

$$\Sigma = \exp(-\mathbf{A}/\alpha) \tag{2.1}$$

where $\mathbf{A}$ is the Euclidean distance matrix of all the mask cells, and α is either α_{forest}, $\alpha_{protected}$, or $\alpha_{altitude}$, depending on the relevant covariate. These α values ultimately determine the extent of the spread of their corresponding covariates, and how 'smooth' the edges of these covariate regions will end up being. The greater the value of α, the 'smoother' the covariate region will be.

In the cases of the binary covariates 'forest' and 'protected area', after the random number generation produces a number for each mask cell, the cell is assigned the value '1' if the generated number is greater than 0, and '0' otherwise. Here, a '1' indicates that the cell is 'forest' or 'protected' depending on the covariate of interest, and '0' indicates that it is not these things.

Figures 2.1 and 2.2 demonstrate simulated forest cover and protected area cover, respectively, across the study region in the first session of a particular multi-session simulation. These figures were generated using `acre`.

In the case of the numeric covariate 'altitude', terrain must be considered, because, as one might expect, mountainous regions tend to have greater altitudes than non-mountainous regions. Therefore, if a session takes place on a mountainous terrain, its range of possible altitude values is restricted to be between 2000m and 4500m. Conversely, if a session takes place in a non-mountainous region, its range of possible altitude values is restricted to be between 0m and 2000m (below sea level altitudes are not currently considered here, and this may leave room for a potential expansion of the simulation in the future). Once the random number has been generated, it is subsequently multiplied by a quarter of the difference in the maximum and minimum altitude values and added to the mean of the range of possible altitudes in order to assign each mask cell an altitude. This results in a smoothly changing alititude.

Figure 2.3 demonstrates simulated altitude values from the same simulation noted above, and `acre` was again used to create this plot.

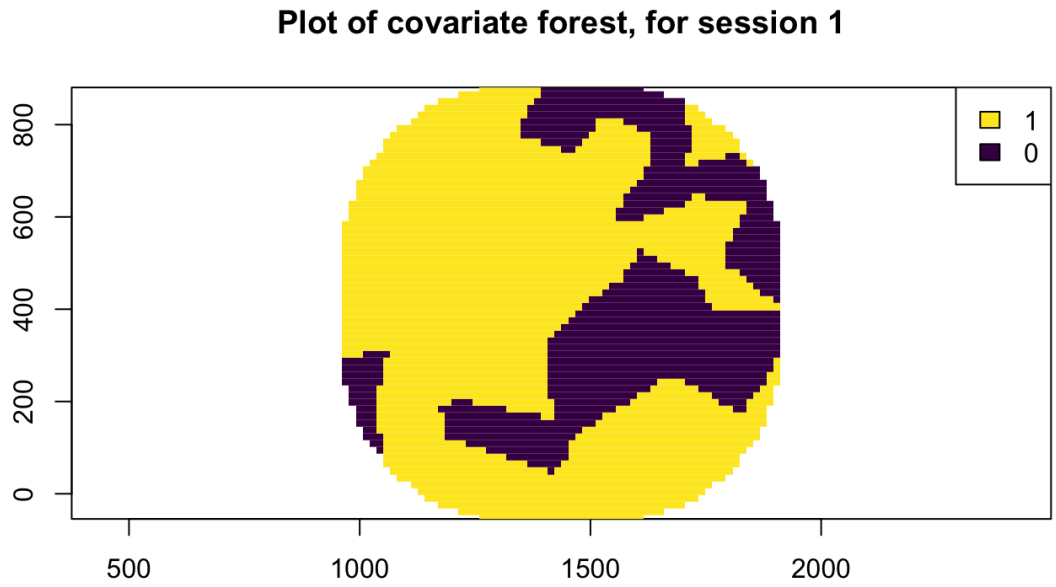


Figure 2.1: Simulated forest coverage created using spatial correlation.

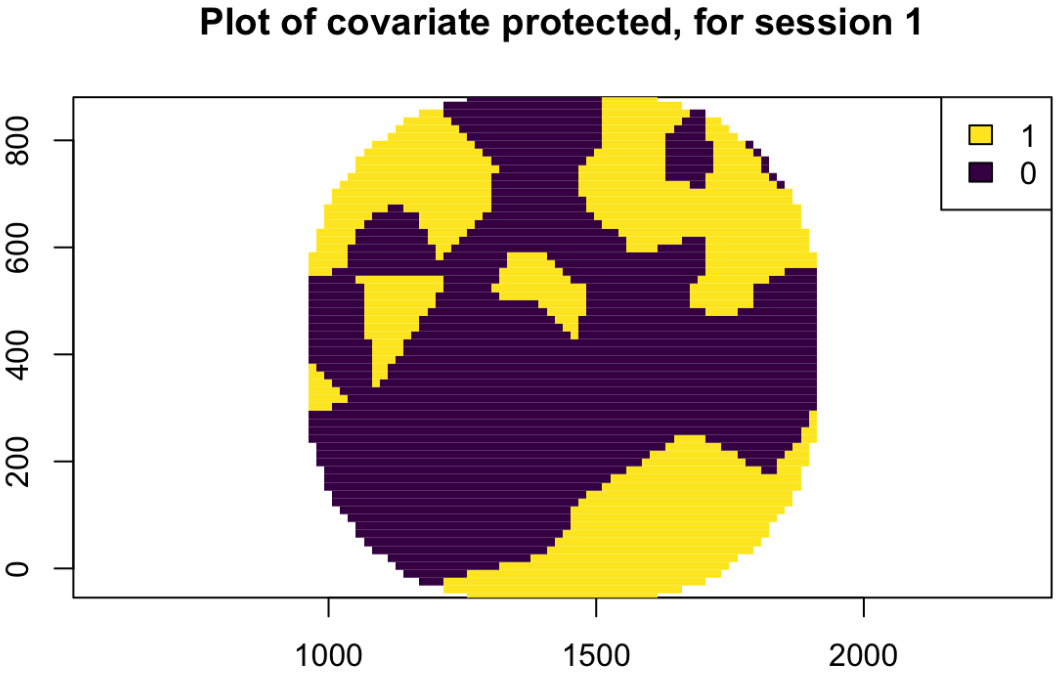


Figure 2.2: Simulated protected area coverage created using spatial correlation.

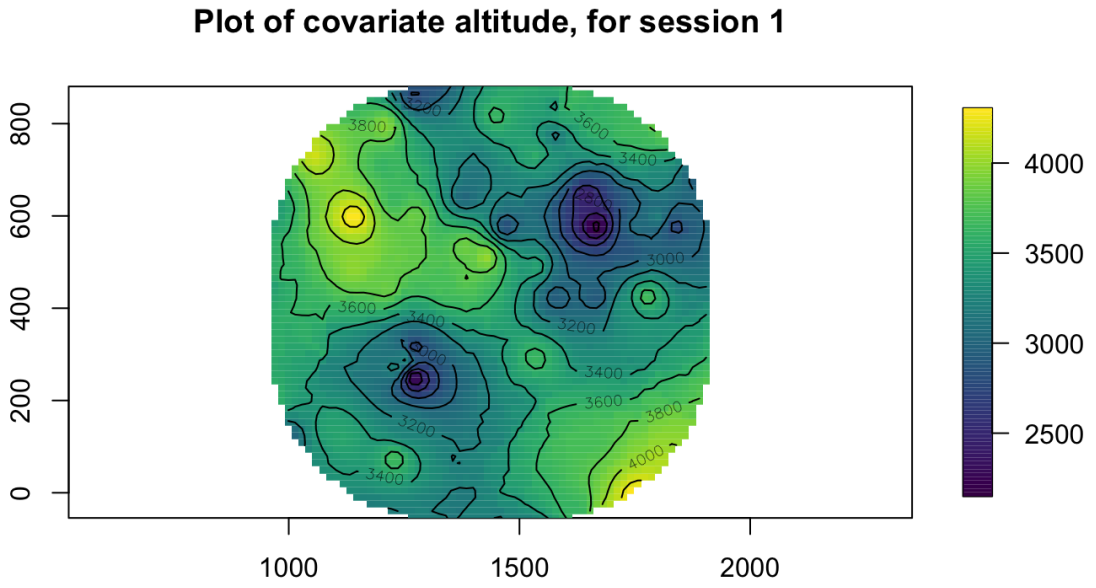


Figure 2.3: Simulated protected altitude values created using spatial correlation.

2.1.4 Session covariates

Sessions that reoccur in the same study area at different points in time do not necessarily share the same spatial covariates. For example, an area that was covered in forest in an initial session may be subject to deforestation, and therefore in subsequent sessions forest is no longer present to the degree it was in the first session. Sessions also do not need to occur in the same study area, therefore it is possible for spatial covariates to be entirely different between sessions. For this simulation, in addition to the spatial covariates described above, covariates that were subject to change between sessions are described below.

Location of the study area

The location of the study area usually dictates spatial covariates, and therefore affects population densities and detection probabilities. While sessions may occur in the same study area each time, it is possible for them to take place in completely independent areas.

Time

Time certainly has the potential to increase or decrease population densities, depending on the direction of the trend, if any, in population growth. For example, if a particular species is subject to hunting or poaching, densities may decrease with time.

Weather conditions

Weather conditions can negatively or positively impact detection probabilities, based on detection methods. For example, loud heavy rain could make audio detections more difficult. In this simulation, the weather conditions generated for each session are either 'snow', 'rain', 'overcast', or 'run', and, for simplicity, these conditions affect g_0 linearly, in the order given.

Terrain

The accuracy of observed bearings of acoustic detections is likely to change based on the terrain of the session in which it occurs. For example, mountainous areas can create echoes which distort animal calls, significantly reducing the accuracy of the detections. Therefore, when simulating measurement error, terrain should be accounted for.

Measurement error for observed bearings is simulated using the `CircStats` package (Agostinelli 2018) to generate random numbers from the von Mises distribution with mean of the true bearings and a value for κ chosen by the user. κ, also known as the *concentration parameter*, determines how close the observed bearings typically are to the true bearings. It is this parameter, then, that should vary with types of terrain.

Hence, the concentration parameter is adjusted within the simulation using the equation:

$$\log(\kappa) = \beta_{0cp} + \beta_{1cp} \times mountain \tag{2.2}$$

where *mountain* takes the value 1 when the session is mountainous and 0 otherwise, β_{0cp} is a user-chosen intercept, and β_{1cp} determines the extent to which the user wishes mountainous regions to impact observed bearing accuracy.

Terrain is also thought to affect g_0; for example, if a session occurs in a mountainous region, the ability of acoustic detectors to pick up sounds can be blocked or limited.

2.1.5 Density calculations based on covariates

To calculate the density of animals within each mask cell in the study area, the simulation uses the following equation:

$$D = \exp(\beta_0 + \beta_1 \times x + \beta_2 \times y + \beta_3 \times forest + \beta_4 \times village + \beta_7 \times protected + \beta_8 \times altitude + \beta_9 \times time) \tag{2.3}$$

where:

- x is the x coordinate of the mask cell
- y is the y cordinate of the mask cell
- $forest$ is the binary variable 1 if the mask cell is 'forest' and 0 otherwise
- *village* is the distance of the mask cell to the centre of its nearest village (if no villages exist, then this term is omitted)
- *protected* is the binary variable 1 if the mask cell is 'protected area' and 0 otherwise

- *altitude* is the altitude of the mask cell in metres
- *time* is the time (year) the session occurs
- β_0 is an intercept term
- β_1 is the coefficient determining how density varies horizontally across the study area
- β_2 is the coefficient determining how density varies vertically across the study area
- β_3 is the coefficient that determines the extent to which forest impacts density
- β_4 is the coefficient that determines how density changes with respect to distance to the nearest village
- β_7 is the coefficient that determines the extent to which protected area impacts density
- β_8 is the coefficient that determines how density changes with respect to altitude
- β_9 is the coefficient that determines how density changes over time

2.1.6 Adjusting g_0 with covariates

There are three terms involved in the adjustment of g_0:

- g_{0base} is the baseline value of g_0, not accounting for any impact of conditions
- *mountain* takes the user-chosen value β_6 if the terrain of the session is mountainous, and 0 otherwise
- *weather* takes the user-chosen value β_5 multiplied by 1, 2, 3, or 4, depending on whether the conditions are 'snow', 'rain', 'overcast', or 'sun' respectively.

While g_{0base} is chosen by the user, g_0 is adjusted within the simulation using the logistic distribution function with mean $logit(g_{0base}) + mountain + weather$. It is this value that is subsequently used in the detection function.

2.2 The Functions

With the above considerations in mind, the functions used within the simulation can be divided into three categories, depending on how they serve the overall purpose:

1. **Covariate creation**

- `create_villages()`
- `generate_session_information()`
- `generate_session_locations()`

2. **Helper functions**

- `create_mask()`
- `find_bearings()`

- `format_capture_histories()`

3. **The actual simulation**

- `simulate_capture_histories_with_sessions()`

These functions will be described in the order in which they are listed.

2.2.1 `create_villages()`

`create_villages()` is responsible for creating the locations of a desired number of hypothetical 'villages', as described in Section 2.1.1. The user of this function must input a chosen number of villages, as well as the range of x and y coordinates where these villages can occur. The function returns a data frame containing the x and y coordinates of the invented villages.

For example, the following function call can be used to create four villages that are located between -2000 and 2000 on the x-axis, and -500 and 1900 on the y-axis.

```
create_villages(number = 4, x_range = c(-2000, 2000),
                y_range = c(-500, 1900))
```

This can be expected to produce a data frame such as the following:

```
##            x          y
## 1 -1706.3424  -10.91383
## 2 -1453.9908  378.35233
## 3   738.4348 1530.86982
## 4 -1601.5123   68.49423
```

2.2.2 `generate_session_information()`

This function creates a range of conditions for a given number of sessions over a specified period of time. The user inputs a set number of sessions over which the hypothetical study is to be conducted, two vectors which contain specific conditions that are possible within the study, the range of years over which the sessions can occur, and the horizontal and vertical dimensions that the survey areas should take.

The vectors containing session conditions are intended primarily to store potential weather conditions and terrain possibilities, but could be used for any other similar covariates that the user wishes to generate. These must be character vectors that contain all possible states that the particular covariate can take. For example, a vector storing weather conditions could be `c("sun", "rain", "overcast", "snow")`. It should be noted that although this function can handle any kind of covariates, the simulation as a whole can currently only function with terrain and weather covariates.

`generate_session_information()` includes an optional argument `space_overlap`, which is set to `FALSE` by default. When set to `TRUE`, sessions are allowed to overlap in space, otherwise survey areas are created for sessions so that no two sessions may occur over any overlapping areas. In accordance with this is an additional optional argument, `min_space`—set to 50 by default—which sets the minimum horizontal and vertical distance between survey areas across sessions. The intention for this is to

achieve as much independence in space between sessions as possible, if that is desired for the simulated data. When `space_overlap` is set to `TRUE`, `min_space` is simply ignored.

The result of running `generate_session_information()` is a data frame with a column for the number of the session, columns for the weather and terrain (or other covariates) present at the session, the time (year) that the session occurred, and the range of x and y coordinates of the survey area, generated randomly within the function.

For a study with five sessions over varying weather conditions, where the terrain can either be mountainous or non-mountainous, the dimensions of the survey areas have to be 1000m horizontally and vertically, and sessions must occur at least 40m apart from one another in space, `generate_session_information()` can be used as below:

```
num_sessions <- 5
generate_session_information(num_of_sessions = num_sessions,
                             conditions1 = c("rain", "sun", "snow", "overcast"),
                             conditions2 = c("mountain", "non-mountain"),
                             time_range = 2012:2018, x_dimension = 1000,
                             y_dimension = 1000, min_space = 40)
```

and will produce a data frame as below:

```
##   session weather      terrain time x_start x_end y_start y_end
## 1       1    snow non-mountain 2013     513  1513    1239  2239
## 2       2    rain non-mountain 2018    3169  4169    6092  7092
## 3       3     sun     mountain 2015    8046  9046    8297  9297
## 4       4    snow non-mountain 2016   11636 12636   10605 11605
## 5       5    rain non-mountain 2012   13091 14091   14420 15420
```

2.2.3 `generate_session_locations()`

The locations of the sessions are generated in a separate function which is called within `generate_session_information()` for the sake of maintaining clean and readable modular code. The spatial dimensions and `min_space` arguments which are first passed to `generate_session_information()` are then passed to `generate_session_locations()`, along with the number of sessions.

If overlap of sessions in space is not allowed, then a range of potential x and y coordinates is calculated with consideration of the above arguments. This space is then partitioned into non-overlapping subregions, where each session is assigned a particular subregion. Within each session's subregion, a section the size of the predetermined spatial dimensions is randomly chosen, and the coordinates of this section are returned as the spatial coordinates of the study area for that session. If overlap of sessions in space *is* allowed, then a range of potential x and y coordinates is calculated *without* consideration of the above arguments. Sections the size of the predetermined spatial dimensions are then randomly chosen throughout this region, without the need for initial partitioning.

Given that `generate_session_locations()` is used only inside `generate_session_information()`, see the example given for `generate_session_information()` above for an example of how `generate_session_locations()` is used.

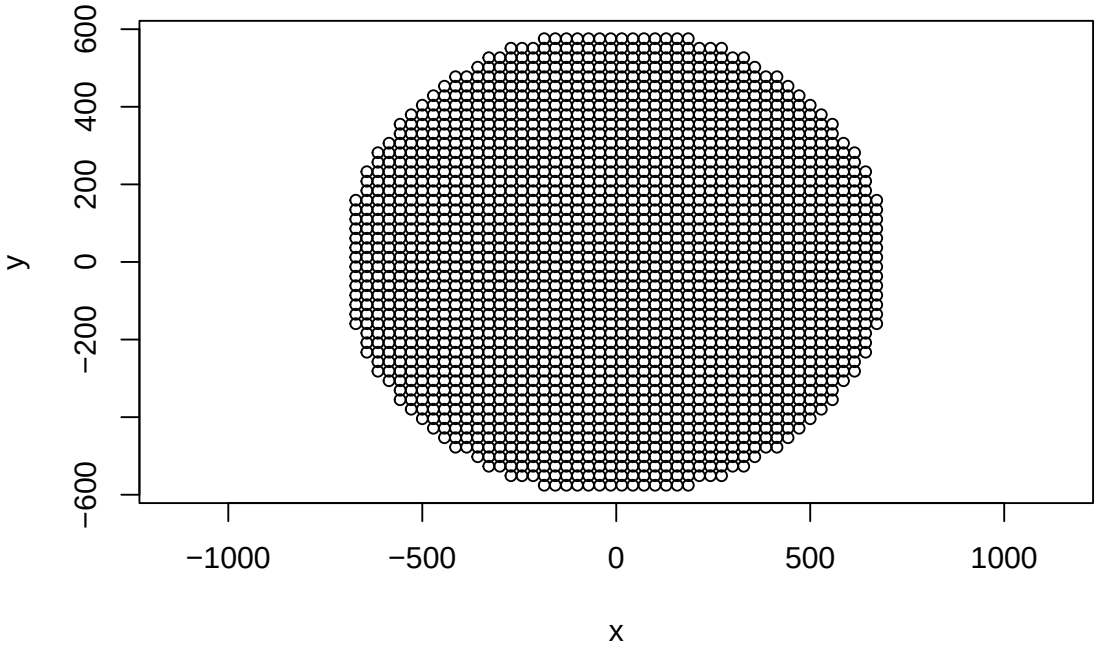
2.2.4 `create_mask()`

This is an optional function that provides an easy way to create an ellipse-shaped grid of points which act as mask cells within the simulation. Other methods of producing a mask are allowed and welcomed as substitutes for this function.

`create_mask()` takes the desired widths of the x and y axes of the ellipse shape, as well as the number of rows and columns of the grid of points, allowing the user flexibility in setting the granularity of the mask.

The function can be used, and the resulting grid plotted, as below:

```
mask <- create_mask(x_width = 700, y_width = 600, num_rows = 50, num_cols = 50)
plot(mask, cex = 0.8, asp = 1)
```



2.2.5 `find_bearings()`

The `find_bearings()` function is a simple helper function used inside `simulate_capture_histories_with_sessions()`. It finds the true bearing, in radians, from one set of coordinates in the study area to another. Some random variation proceeds the use of this function to account for measurement error in a 'real world' scenario.

The function's inputs are two vectors that contain the x and y coordinates of the points of interest, and its output is a single numerical value, which is the bearing. For simplicity, regular 2-dimensional geometry is used to calculate the bearings, rather than a true latitude-longitude approach.

As this is a trivial helper function, an example is not given.

2.2.6 format_capture_histories()

The second helper function used within simulate_capture_histories_with_sessions() is needed to transform a binary capture history matrix into a long form capture history data frame. This function operates on a single session at a time, so a session ID must be given as an argument in addition to the binary capture history matrix.

The resulting data frame has four columns: session (the session ID), ID (the animal ID), occasion, and trap. This structure is required by the ascr and acre packages.

The function may be used as below:

```
set.seed(42)
#Create example binary matrix
m <- matrix(0, 5, 5)
(example_binary_capthist <- cbind(apply(m, c(1,2),
                                        function(x) rbinom(1, 1, 0.5)), 1:nrow(m)))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    1    0    1    1    1
## [2,]    1    1    1    1    0    2
## [3,]    0    0    1    0    1    3
## [4,]    1    1    0    0    1    4
## [5,]    1    1    0    1    0    5
```

```
#Change to long form capture history
knitr::kable(do.call(rbind,
                     apply(example_binary_capthist, 1, format_capture_histories, 1)))
```

session	ID	occasion	trap
1	1	1	1
1	1	1	2
1	1	1	4
1	1	1	5
1	2	1	1
1	2	1	2
1	2	1	3
1	2	1	4
1	3	1	3
1	3	1	5
1	4	1	1
1	4	1	2
1	4	1	5
1	5	1	1
1	5	1	2
1	5	1	4

2.2.7 simulate_capture_histories_with_sessions()

The final function is where the simulation truly takes place. It operates on a single session at a time, so a for loop or similar method can be used with the function when multiple sessions are desired. The

output created is a named list containing the following components:

- `binary_capture_history` is a data frame containing the simulated binary capture history for the session.
- `long_form_capture_history` is a data frame containing the same information as `binary_capture_history` but in a long data format.
- `animal_locations` is a data frame containing the x and y coordinates of all the animals in the survey region, regardless of whether or not they were detected.
- `detected_animal_coords` is a data frame which contains the x and y coordinates of only the animals in the survey region that were detected.
- `forest` and `protected_areas` are binary character vectors containing values '1' and '0'. Each element in the vectors represents a mask cell. A '1' indicates that the mask cell is considered to be 'forest' or 'protected area' respectively, and a '0' indicates otherwise.
- `altitude` is a numeric vector, where, like `forest` and `protected_areas`, each element represents a mask cell. It contains the altitudes, in metres, for each mask cell.
- `traps` is a data frame containing the x and y coordinates of the traps in the survey region. It is necessary to return this, as the simulation adjusts the relative positions of the detectors based on where the particular session occurs in space.
- `mask` is a data frame containing the x and y coordinates of the centres of the mask cells in the survey region. This must be returned for the same reason as `traps`.

Although the output of `simulate_capture_histories_with_sessions()` is somewhat compendious, it is necessary for creating models with `ascr` and `acre`, validating their accuracy, and creating visualisations of the simulations.

`simulate_capture_histories_with_sessions()` accepts the following arguments:

- `session_info`: data frame, *required*
  - This is the data frame containing the session covariates that is created in `generate_session_information()`.
- `alpha_forest`, `alpha_protected`, `alpha_altitude`: numeric, *required*
  - The values of α_{forest}, $\alpha_{protected}$, and $\alpha_{altitude}$ discussed in Section 2.1.3.
- `beta0`: numeric, *required*:
  - The value of β_0 in Equation 2.3
- `beta1`, `beta2`, `beta3`, `beta4`: numeric, *optional (default = 0)*:
  - The values of β_{1-4} in Equation 2.3.
- `beta5`, `beta6`: numeric, *optional (default = 0)*:
  - The values of β_5 and β_6 discussed in Section 2.1.6.
- `beta7`, `beta8`, `beta9`: numeric, *optional (default = 0)*:
  - The values of β_{7-9} in Equation 2.3.
- `mask_locations`: numeric matrix, *required*:

  - A matrix containing the x and y coordinates of the centres of all the mask cells. This is to be the same for all sessions, regardless of whether the sessions overlap in space or not. The coordinates are adjusted relative to the study area within the simulation.

- `detector_locations`: numeric matrix, *required*:
  - A matrix containing the x and y coordinates of the the detectors. As with `mask_locations`, these are adjusted relative to the coordinates of the study area, so they may be the same across all sessions even if the sessions do not overlap in space.

- `g0_base`: numeric, *required*:
  - The baseline value for g_0 without interference from session covariates such as weather or terrain.

- `sigma`: numeric, *required*:
  - The value of σ used in the detection function.

- `x_range`, `y_range`: numeric vector, *required*:
  - The maximum and minimum x and y coordinates over which the survey area spans.

- `village_locations`: data frame, *optional (default = NULL)*:
  - A data frame created in `create_villages()` that stores the x and y locations of the centres of all the villages in the study area, if there are any.

- `cp_beta0`: numeric, *required*:
  - The value of β_{0cp} in Equaion 2.2.

- `cp_beta1`: numeric, *optional (default = 0)*:
  - The value of β_{1cp} in Equation 2.2.

While the number of arguments used in `simulate_capture_histories_with_sessions()` may seem exhaustive, it allows the function to simulate many different study scenarios with a large degree of flexibility and specificity, and is beneficial for testing how the functions within `acre` may handle particular edge cases. Ultimately, the ability to tailor so many parameters and covariates results in a simulation that better serves the goals of this dissertation.

2.2.7.1 Order of steps in `simulate_capture_histories_with_sessions()`

The simulation occurs in the following order of steps:

1. The locations of mask cells and detectors are adjusted relative to the x and y ranges of the study area.
2. The simulation method described in Section 2.1.3 is used to define forest areas, protected areas, and altitudes across all the mask cells.
3. As described in Section 2.1.6, variables `mountain` and `weather` are created.
4. As described in Section 2.1.6, using the logistic distribution, set a value for g_0 with the following code: `plogis(qlogis(g0_base) + mountain + weather)` where `mountain` and `weather` are the variables created in step 3, and `g0_base` is the user's chosen baseline value for g_0.
5. If villages exist, then for each mask cell calculate the distance to the centre of its nearest village.

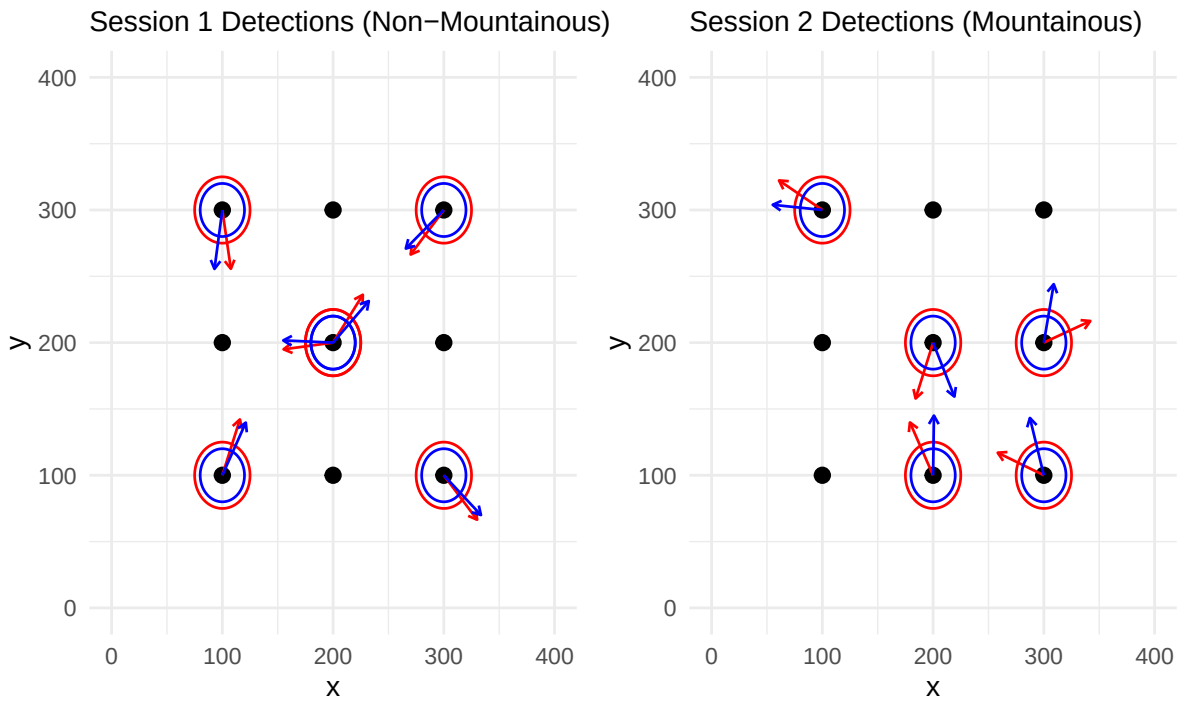


Figure 2.4: The effect of mountainous regions on observed bearing accuracy. Red represents the true bearing and blue represents the observed bearing.

6. For each mask cell, calculate the density of animals using Equation 2.3.

7. For each mask cell, generate a random number from a Poisson distribution where λ is the area of the mask cell multiplied by its density calculated in step 6. This is the number of animals that are truly present within the cell.

8. Sum the number of animals across all cells to get the total number of animals truly present across the entire study area.

9. Within each mask cell, consider each animal to be 'displaced' from its centre by some distance on the x axis and some other distance on the y axis, where these distances are less than or equal to half the width of the mask cell. Randomly generate these distances from a uniform distribution. Add these displacement distances to the coordindates of the centres of the mask cells to derive the true coordinates of all the animals in the study area.

10. Use the `fields` package (Nychka et al. 2021) to calculate the Euclidean distance matrix among all pairings of animals and detectors within the study area.

11. Use the distances calculated in step 10 with a half normal detection function (Equation 1.2) to calculate the detection probabilities for all animals by all detectors in the study area. For this detection function, g_0 is the value determined in step 4 and σ is the value from the function input. Store these probabilities in a matrix where each row represents an individual animal and each column represents a detector. Then the value in the ith row and jth column of this matrix is the probability of animal i being detected by detector j.

12. For each probability in the matrix created in step 11, use a binomial distribution with p equal to this probability to randomly generate either a 0 or a 1. If the result is 1, then the animal is considered to have been detected, and if the result is 0 then the animal has not been detected. Hence, the value in the ith row and the jth column of the matrix stores whether or not animal i has been detected by detector j. Essentially, this is now a binary capture history matrix. This matrix differs from the one described in Section 1.2.1, however, as its columns represent detectors rather than occasions.

13. In the matrix created in step 12, there are likely some or even many rows where all values are 0. This indicates that a particular animal has not been detected by any detectors within the study region. In a real world situation, it would not be possible to know that these animals were present at all, as evidently there would be no indication of their presence within the study area. Therefore, individuals with row sums equal to zero are removed from the binary capture history matrix, and the result of this is returned as a list item in the final output of `simulate_capture_histories_with_sessions()`.

14. The `format_capture_histories()` function is used to convert the binary capture history matrix into a long form capture history table.

15. Use Equation 2.2 with `cp_beta0` and `cp_beta1` (if provided) to adjust the concentration parameter (κ) used in bearing estimation.

16. For every detection recorded in the long form capture history, use the `find_bearings()` helper function to calculate the true bearing of the animal from the detector.

17. As described in Section 2.1.4, generate a random number from the von Mises distribution to create some measurement error around the true bearings calculated in step 16. The result of this random number generation is the observed bearings of the animals from each detector.

18. Join the observed bearings to the long form capture history table generated in step 14. This table is now ready to be returned as a list item in the final output of `simulate_capture_histories_with_sessions()`.

19. Return all necessary list items.

The following section contains a demonstration of the usage of `simulate_capture_histories_with_sessions()`.

2.3 Simulation Examples

In the simulation's GitHub repository, three files can be found which demonstrate the entire usage and workflow of the simulation: simulate_example1.R, simulate_example2.R, and simulate_example3.R.

These examples follow the same general workflow, and the code in simulate_example1.R will be used to illustrate the process. Please see the other two files to gauge how the function works with other parameter values.

1. Load the necessary packages, and load the functions described above.

```
#Load packages ----------------------------------------------------------------
library(mvtnorm)
library(fields)
library(gstat)
library(CircStats)
library(dplyr)

#Call functions ---------------------------------------------------------------
source("functions/create_villages.R")
source("functions/find_bearings.R")
source("functions/format_capture_histories.R")
source("functions/simulate_capture_histories_with_sessions.R")
source("functions/generate_session_information.R")
source("functions/generate_session_locations.R")
source("functions/create_mask.R")
```

2. If desired, created a set number of villages.

```
#Create two villages ----------------------------------------------------------
village_locations1 <- create_villages(2, c(0, 8000), c(0, 700))
```

3. Set the number of sessions, and generate session covariates.

```
#Set number of sessions -------------------------------------------------------
num_sessions1 <- 3

#Generate session information -------------------------------------------------
sessions1 <- generate_session_information(num_sessions1,
                                          c("rain", "sun", "snow", "overcast"),
                                          c("mountain", "non-mountain"),
                                          2012:2018, 1000, 1000, 40)
```

4. Set locations for the centres of the mask cells, as well as the detector locations.

```r
#Create a mask ------------------------------------------------------------------
mask1 <- create_mask(900, 600, 66, 66)

#Set detector locations ---------------------------------------------------------
detectors1 <- matrix(c(rep(-1:1, each = 3)*100, rep(-1:1, 3)*100), ncol=2)
```

It is suggested to plot these together to ensure they make sense spatially. Figure 2.5 shows the result of plotting these locations.

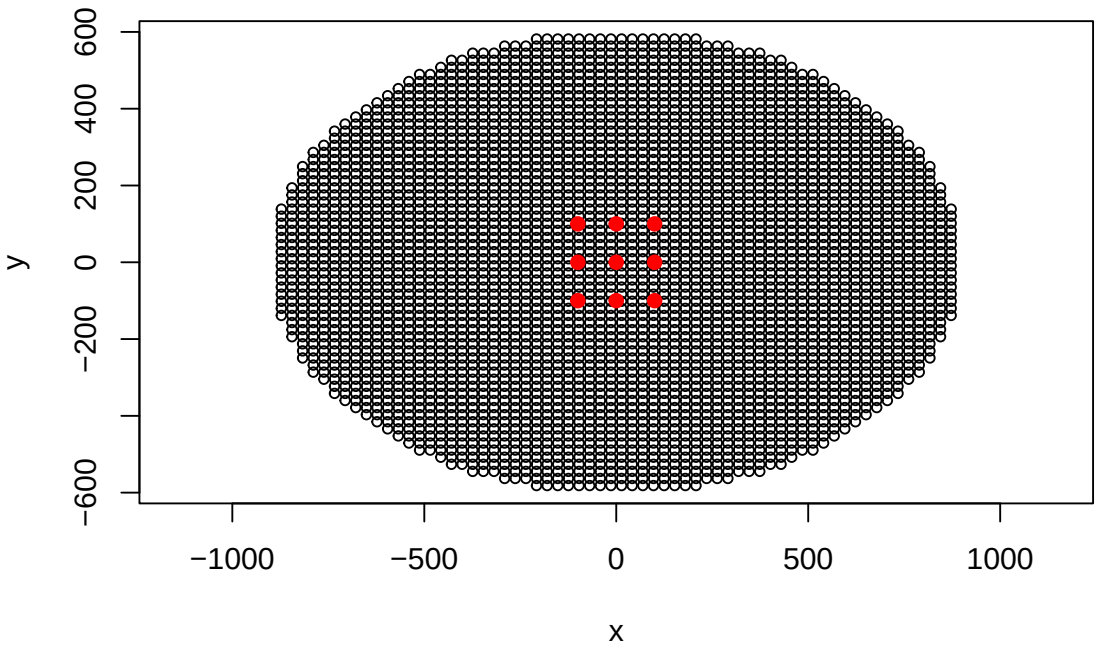


Figure 2.5: Generated mask cell points overlaid with generated detector locations.

5. An optional but recommended step within the workflow may be to run `simulate_capture_histories_with_sessions()` with the above variables and desired parameters using only a single session so as to test its output before committing to running the function with all sessions, which is more computationally expensive. This will be done here so that the output when running `simulate_capture_histories_with_sessions()` once can be shown.

```r
#Test with the first session only -----------------------------------------------
single_session_demo <- simulate_capture_histories_with_sessions(
  session_info = sessions1[1,],
  alpha_forest = 400,
  alpha_protected = 300,
```

```
  alpha_altitude = 400,
  beta0 = -1, beta1 = -0.0002,
  mask_locations = mask1,
  detector_locations = detectors1,
  g0_base = 0.85,
  sigma = 85,
  x_range = c(sessions1[1,5], sessions1[1,6]),
  y_range = c(sessions1[1,7], sessions1[1,8]),
  beta2 = -0.0002, beta3 = 2.5, beta4 = 0.00015,
  beta5 = 0.5, beta6 = -0.2, beta7 = 1.9,
  beta8 = -0.00001, beta9 = 0.000002,
  village_locations = village_locations1,
  cp_beta0 = log(50), cp_beta1 = -1)
```

Using the `head()` function, a preview of the various output list components from the result of running a single session simulation can be shown as below:

```
#Binary capture history
head(single_session_demo$binary_capture_history)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    0    0    0    0    0    0    1    0    0
## [2,]    1    0    0    0    0    0    0    0    0
## [3,]    1    0    0    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0    1    0    0
## [5,]    0    0    0    0    1    0    0    0    0
## [6,]    0    0    0    0    1    0    0    0    0
```

```
#Long form capture history
head(single_session_demo$long_form_capture_history)
```

```
##   session ID occasion trap  bearing
## 1       1  1        1    7 3.154098
## 2       1  2        1    1 3.530155
## 3       1  3        1    1 2.884174
## 4       1  4        1    7 4.270040
## 5       1  5        1    5 3.773105
## 6       1  6        1    5 3.511861
```

```
#Animal locations
head(single_session_demo$animal_locations)
```

```
##       animal_x  animal_y
## [1,] 378.29781 -245.2880
## [2,] 435.35161 -229.1750
## [3,] -45.14103 -219.0525
## [4,] 135.45544 -217.0646
## [5,] 468.03980 -225.9193
## [6,] -28.80107 -198.1939
```

```r
#Detected animal locations
head(single_session_demo$detected_animal_locations)
```

```
##    animal_x  animal_y ID
## 1 418.07025  35.96080  1
## 2  72.67465  93.86708  2
## 3 219.81108  89.55934  3
## 4 301.05585  88.72414  4
## 5 144.99168 104.08055  5
## 6 172.09568 121.41952  6
```

```r
#Forest area
head(single_session_demo$forest)
```

```
## [1] "1" "1" "1" "0" "0" "0"
```

```r
#Protected area
head(single_session_demo$protected_areas)
```

```
## [1] "0" "0" "0" "0" "0" "0"
```

```r
#Altitude
head(single_session_demo$altitude)
```

```
## [1] 4566.706 4490.872 4486.379 4347.740 4152.297 4198.350
```

```r
#Detector locations
head(single_session_demo$traps)
```

```
##     x   y
## 1 152 249
## 2 152 349
## 3 152 449
## 4 252 249
## 5 252 349
## 6 252 449
```

```r
#Mask cell locations
head(single_session_demo$mask)
```

```
##            x         y
## 92  44.30769 -232.5385
## 93  72.00000 -232.5385
## 94  99.69231 -232.5385
## 95 127.38462 -232.5385
## 96 155.07692 -232.5385
## 97 182.76923 -232.5385
```

6. If the results after creating data for a single session simulation look reasonable, then a `for` loop may be used to loop through all the sessions. Empty lists can be initiated before running the loop to provide places to store the components of the simulation output. The `do.call()` and `lapply` functions can be used subsequently to consolidate the output from the different sessions into single data frames.

```
#Run simulation with all sessions ----------------------------------------------
session_sim_capt_hist <- vector(mode = "list", length = num_sessions1)
session_sim_bin_capt_hist <- vector(mode = "list", length = num_sessions1)
session_animal_locations <- vector(mode = "list", length = num_sessions1)
session_sim_forest <- vector(mode = "list", length = num_sessions1)
session_sim_altitude <- vector(mode = "list", length = num_sessions1)
session_sim_protected <- vector(mode = "list", length = num_sessions1)
session_sim_mask <- vector(mode = "list", length = num_sessions1)
session_sim_traps <- vector(mode = "list", length = num_sessions1)

for(i in 1:num_sessions1) {
  session_sim <- simulate_capture_histories_with_sessions(
    session_info = sessions1[i,],
    alpha_forest = 400,
    alpha_protected = 300,
    alpha_altitude = 400,
    beta0 = -1, beta1 = 0.0002,
    mask_locations = mask1,
    detector_locations = detectors1,
    g0_base = 0.85,
    sigma = 85,
    x_range = c(sessions1[i,5], sessions1[i,6]),
    y_range = c(sessions1[i,7], sessions1[i,8]),
    beta2 = -0.0002, beta3 = 2.5, beta4 = 0.00015,
    beta5 = 0.5, beta6 = -0.2, beta7 = 1.9,
    beta8 = -0.00001, beta9 = 0.000002,
    village_locations = village_locations1,
    cp_beta0 = log(50), cp_beta1 = -1)
  session_sim_capt_hist[[i]] <- session_sim$long_form_capture_history
  session_sim_bin_capt_hist[[i]] <- session_sim$binary_capture_history
  session_animal_locations[[i]] <- session_sim$animal_locations
  session_sim_forest[[i]] <- session_sim$forest
  session_sim_altitude[[i]] <- session_sim$altitude
  session_sim_protected[[i]] <- session_sim$protected_areas
  session_sim_mask[[i]] <- session_sim$mask
  session_sim_traps[[i]] <- session_sim$traps
}

all_animal_locations1 <- do.call(rbind, session_animal_locations)
all_capt_hist1 <- do.call(rbind, session_sim_capt_hist)
all_bin_capt_hist1 <- do.call(rbind, session_sim_bin_capt_hist)
all_forest1 <- do.call(rbind, session_sim_forest)
all_altitude1 <- do.call(rbind, session_sim_altitude)
all_protected1 <- do.call(rbind, session_sim_protected)
all_mask1 <- do.call(rbind, session_sim_mask)
all_traps1 <- lapply(session_sim_traps,
```

```
                    function(dat) data.frame(x = dat[, 1], y = dat[, 2]))
```

The resulting data frames from this code are of the same form as those shown above, but include data over all sessions.

Once the full simulation has been run, it is helpful to contextualise the result by observing the location where each session occurs in space, along with locations of detectors and villages. This can be done using the following code, the result of which is shown in Figure 2.6.

```
plot(all_mask1, cex=0.01, xlim=c(-375, 7000))
points(all_traps1[[1]], col="red", pch=19, cex=0.2)
points(all_traps1[[2]], col="red", pch=19, cex=0.2)
points(all_traps1[[3]], col="red", pch=19, cex=0.2)
points(village_locations1$x, village_locations1$y, col="blue", pch=19, cex=0.8)
```

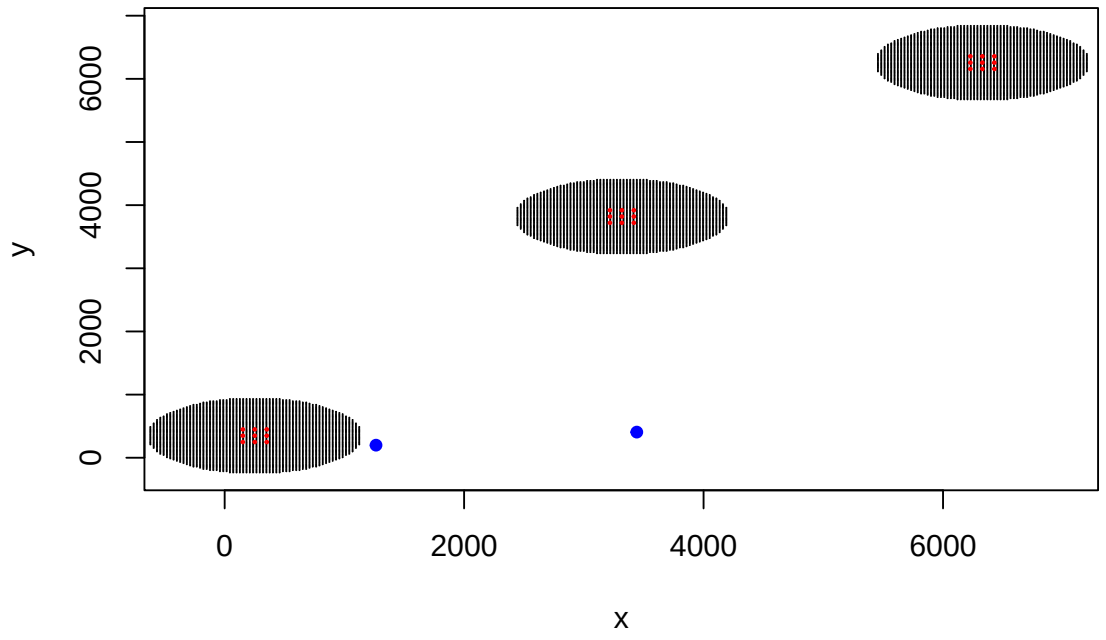


Figure 2.6: The locations of each session and their detectors (red) in the simulation exmaple, along with village locations (blue).

Chapter 3

Using Simulated Data with acre

The GitHub repository for this project contains a file acre_model_examples.R, which documents how the output from the three simulation examples saved in simulate_example1.R, simulate_example2.R, and simulate_example3.R can be used to create various ASCR models with `acre`.

To illustrate how `acre` can be used, this section will elaborate on the content of acre_model_examples.R that deals with the output of simulate_example1.R, the creation of which is shown in the previous chapter. The same variables created above will continue to be used henceforth.

`acre` can be installed from the GitHub repository https://github.com/b-steve/acre (Stevenson 2023). To use it with R, it must first be loaded using `library()`:

```
library(acre)
```

Before exploring models with `acre`, a data object must be generated. To create the data object, one must first initialise a data frame that contains information about the covariates over all or some of the mask cells. An exception to the covariates provided to this data frame is any covariate where a distance to a particular feature—in this case, the 'villages'—is measured, as these are handled separately during the creation of the `acre` object.

Unlike `ascr`, `acre` does not require the user to have information about all covariates for every single mask cell. This is hugely beneficial as researchers often do not have covariate information at this level of granularity. For example, altitude measurements may be taken at various points across the study area, but measuring the altitude within every individual mask cell is time-consuming and impractical. A demonstration of how using `acre` with limited covariate information compares to using it with full covariate information is shown in Section 3.7.

Additionally, a pre-specified mask is not actually required to use `acre`. The `read.acre()` function introduced below can create a grid of mask cells internally if a mask is not provided to it.

In this example, each row of this covariate data frame represents one particular mask cell. It may be initialised as below.

```
#Create covariates data frame ------------------------------------------------
cov_df1 <- data.frame(x = all_mask1[,1],
                      y = all_mask1[,2],
                      forest = as.vector(t(all_forest1)),
                      altitude = as.vector(t(all_altitude1)),
                      protected = as.vector(t(all_protected1)))
```

An `acre` object can now be produced by employing the `read.acre()` function, as shown below. The first two arguments are the long form capture history and the matrix containing locations of all the detectors over all the study sessions. The next argument, `control_create_mask()`, is a list containing a mask *buffer*. The goal of the buffer is to establish a region around the boundary of the mask cells where it is feasible that animals may be present, even if this region is not technically within the study area. This is necessary in order to account for animal movement outside of the boundary of the mask cells. In this example, the buffer has been set to the value that was used for σ in the simulation that created the data, multiplied by five. This is to allow for the distance that sound can travel from its source, multiplied to generously consider any potential cases where an animal may be heard from unusually far away. While it can increase computation times, using a buffer that is unnecessarily large does not reduce model accuracy, whereas using a buffer that is too small *can* reduce accuracy.

In this example, the object passed to the `loc_cov` argument is the data frame created immediately above, and the `session_cov()` argument is set to the data frame containing the session covariates, which was previously created using the `generate_session_information()` function. The locations of the simulated villages, `village_locations1`, is passed to the function via the `dist_cov` argument in the form of a list, as this is the data structure the argument requires. The list is, specifically, a named list, and therefore any subsequent mention of the village locations in the context of `acre` objects and models in this example will be referred to as `villages`, the name given to the object within the list.

```
#Create an acre object ------------------------------------------------------
acre_object1 <- read.acre(all_capt_hist1, all_traps1,
                          control_create_mask = list(buffer = 85*5),
                          loc_cov = cov_df1,
                          session_cov = sessions1,
                          dist_cov = list(villages = village_locations1)))
```

Finally, ACR models can be fitted using `fit.acre()` with the data object that was just produced. In this example, a null model is created initially, followed by models where density changes with respect to some covariates, and, ultimately, some variations of what can be considered 'full' models. The `summary()` function can be used to view parameter estimates, and `AIC()` may be used for model selection and comparison.

3.1 Null Model

```
#Create a null model --------------------------------------------------------
acre_model1.1 <- fit.acre(acre_object1)
```

```
#Summary output -------------------------------------------------------------
summary(acre_model1.1)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
##        Estimate Std. Error     2.5 %  97.5 %
```

```
## g0     0.958730   0.018595  0.902416  0.9832
## sigma 85.827157   1.232124 83.445895 88.2764
## kappa 21.820093   1.228559 19.540268 24.3659
## D     15.942275   0.581462 14.842411 17.1236
```

In this model a constant density estimate, depending on no variables, is about 15.9 animals per hectare (95% confidence interval: 14.8, 17.1). Using the estimated values of g_0 (0.96) and σ (85.8), a half-normal detection function can be plotted (Figure 3.1). `acre` makes this easy to do with its `show_detfn()` function, as demonstrated below:

```
show_detfn(acre_model1.1)
```

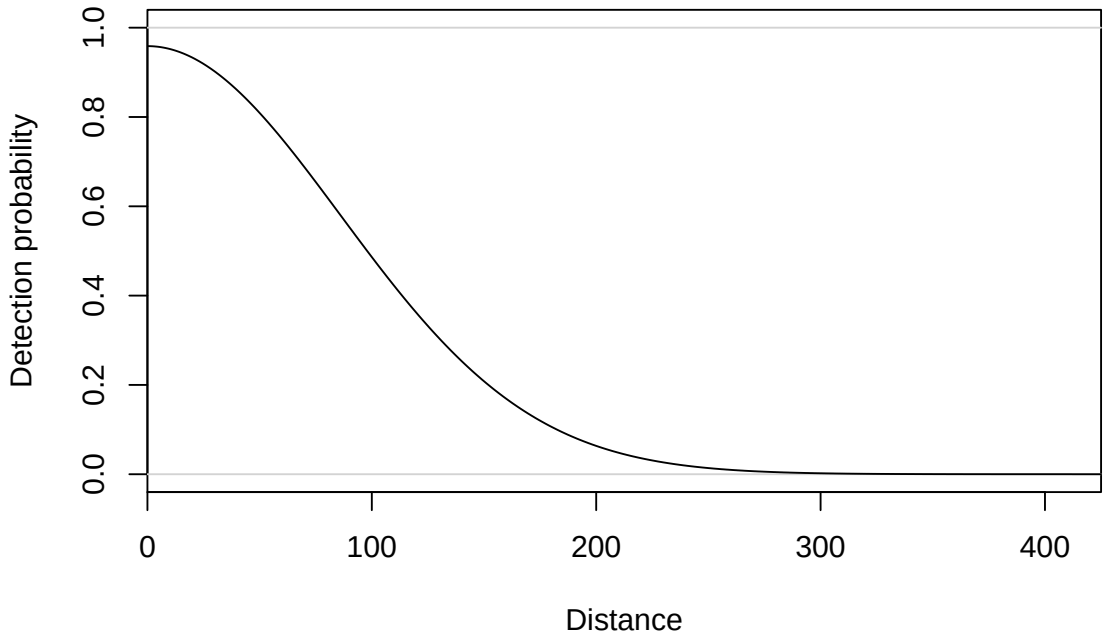


Figure 3.1: The estimated half-normal detection function of the null model.

```
#AIC ------------------------------------------------------------------------
AIC(acre_model1.1)
```

```
## [1] -4574.598
```

The AIC value produced, -4574.6, can be used to evaluate how well subsequent models are supported by the data in comparison.

3.2 Model where Density Varies with Forest Coverage

To add covariates into the model, the `par_extend_model` argument—which requires a list—can be used within `fit.acre()` as shown below:

```
#Create a model where density depends on forest coverage  ----------------------
acre_model1.2 <- fit.acre(acre_object1, par_extend_model = list(D =~forest))
```

As with the null model, the `summary()` and `AIC()` functions can be used if desired.

```
#Summary output ---------------------------------------------------------------
summary(acre_model1.2)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
##                 Estimate Std. Error     2.5 %  97.5 %
## g0              0.962475   0.018317  0.904693  0.9858
## sigma          84.719804   1.160984 82.474603 87.0261
## kappa          21.686067   1.205942 19.446716 24.1833
## D.(Intercept) -0.183487   0.293357 -0.758457  0.3915
## D.forest1      3.511265   0.298158  2.926886  4.0956
##
##
##  Extended parameters link functions:
## D : log
```

```
#AIC ---------------------------------------------------------------------------
AIC(acre_model1.2)
```

```
## [1] -5205.672
```

As expected, a lower value for AIC is produced for this model than that produced from the null model (-5205.7 compared to -4574.6), as a model incorporating an important variable should perform better than one without any important variables. In the summary output, the addition of a row for `D.forest1` can now be seen, indicating the model's estimate of the coefficient for 'forest' in the density equation. Given that this model is lacking a number of important covariates, however, this estimate may not be expected to be a true reflection of the role of forest coverage in dictating animal density, i.e. at this stage it is not an accurate estimate of β_3 in Equation 2.3.

To determine the model's estimate of the multiplicative effect that the presence of forest has on density, one can simply exponentiate the point estimate of `D.forest1`: $\exp(3.51) = 33.4$. Therefore, based on this model, it can be estimated that forested areas have 33.4 times the density of animals as non-forested areas.

For an intuitive way to gauge how well this model has performed, its density surface can be plotted using the following code.

```
#The session number can be replaced with any session of interest
show_Dsurf(acre_model1.2, session = 1)
```

This density surface can then be compared to a plot of the distribution of forest in the study area, which may be generated as follows:

```
#Observe the first plot
plot(acre_object1, types = "covariates", ask = FALSE)
```

These plots are shown in Figure 3.2, and demonstrate what appears to be a one-to-one correlation between animal density and forest coverage. Thus, the model has performed as expected in this regard.

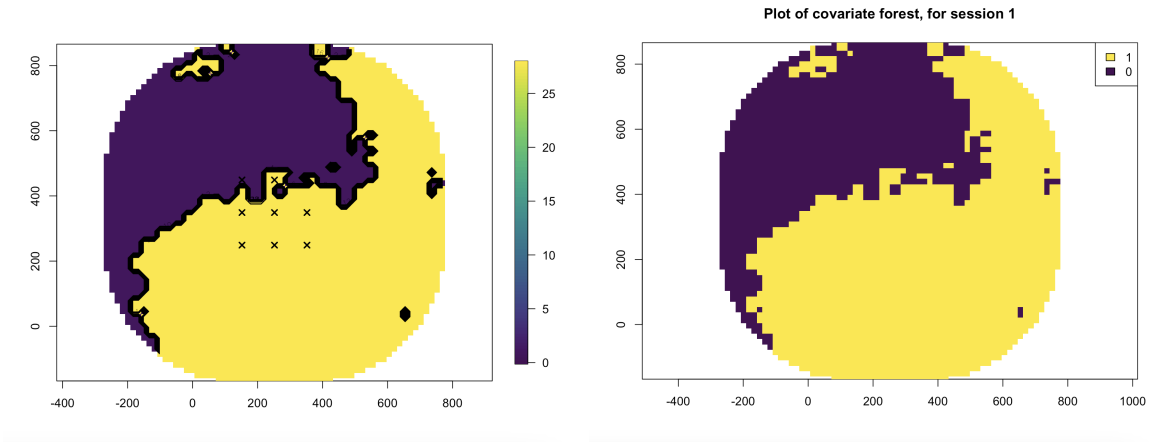


Figure 3.2: Plot of forest coverage and estimated density when density is modelled using forest as a covariate.

3.3 Model where Density Varies with Protected Area

In the same fashion as the preceding model, density may also be modelled with respect to protected areas.

```
#Create a model where density depends on protected area  ----------------------
acre_model1.3 <- fit.acre(acre_object1, par_extend_model = list(D =~protected))
```

```
#Summary output ---------------------------------------------------------------
summary(acre_model1.3)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
```

```
##                 Estimate Std. Error     2.5 %  97.5 %
## g0              0.955915   0.018206  0.902894  0.9806
## sigma          85.132708   1.162265 82.884917 87.4415
## kappa          21.468543   1.188807 19.260512 23.9297
## D.(Intercept)   1.668159   0.082318  1.506818  1.8295
## D.protected1    2.092878   0.097227  1.902317  2.2834
##
##
##  Extended parameters link functions:
## D : log
```

```
#AIC ------------------------------------------------------------------------
AIC(acre_model1.3)
```

```
## [1] -5136.98
```

The AIC here (-5136.98) is higher than that of `acre_model1.2` (-5205.672), which suggests that modelling density solely using protected area as a covariate performs worse than using only forest coverage; in essence, forest coverage is a more important covariate when determining density. This can be further verified by exponentating `D.protected` to calculate the estimated multiplicative effect of protected area on density: $\exp(2.09) = 8.08$. This demonstrates that, in this simulation, protected areas have about 8.08 times the density of non-protected areas, compared to forest areas which have about 33.4 times the density of non-forest areas.

Once again, a density plot can be compared to a plot of protected areas, using the same functions. The output is shown in Figure 3.3.

```
show_Dsurf(acre_model1.3, session = 1)
#Observe the third plot
plot(acre_object1, types = "covariates", ask = FALSE)
```

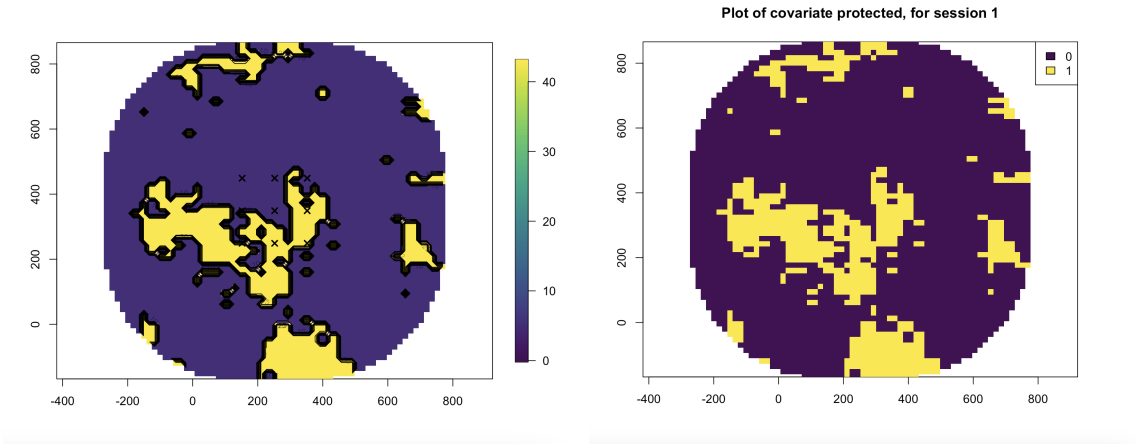


Figure 3.3: Plot of protected area and estimated density when density is modelled using protected area as a covariate.

3.4 Model where Density Varies with Altitude

Next, a model using altitude to explain density is demonstrated, Figure 3.4 demonstrates the comparison in estimated density and altitude.

```
#Create a model where density depends on altitude  ----------------------------
acre_model1.4 <- fit.acre(acre_object1, par_extend_model = list(D =~altitude))
```

```
#Summary output ---------------------------------------------------------------
summary(acre_model1.4)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
##                   Estimate Std. Error      2.5 %  97.5 %
## g0             9.5871e-01 1.8602e-02 9.0238e-01  0.9831
## sigma          8.5814e+01 1.2325e+00 8.3432e+01 88.2644
## kappa          2.1814e+01 1.2286e+00 1.9534e+01 24.3599
## D.(Intercept) 2.5602e+00 7.7424e-02 2.4085e+00  2.7120
## D.altitude     7.8884e-05 2.5157e-05 2.9578e-05  0.0001
##
##
##  Extended parameters link functions:
## D : log
```

```
#AIC ---------------------------------------------------------------------------
AIC(acre_model1.4)
```

```
## [1] -4582.651
```

In this case, the AIC value of -4582.651 is higher than those of both of the previous two models, indicating that using the altitude covariate alone doesn't perform quite as well.

The estimate of 0.0000789 for `D.altitude` can be exponentiated ($\exp(0.0000789) = 1.000079$) and the result interpreted as, for every 1m increase in altitude, the number of animals is multiplied by 1.000079. This means that, for every additional kilometer of altitude, there is about a 7.9% increase in number of animals. An additional kilometer is a significant increase in altitude, so with such a small effect size, it is unsurprising that models using the other covariates are preferable to this one.

Again, density surface can be plotted and compared to altitude distribution. Clear similarities in their general shapes can be seen (Figure 3.4).

```
show_Dsurf(acre_model1.4, session = 1)
plot(acre_object1, types = "covariates") #Hit ENTER on keyboard three times
```

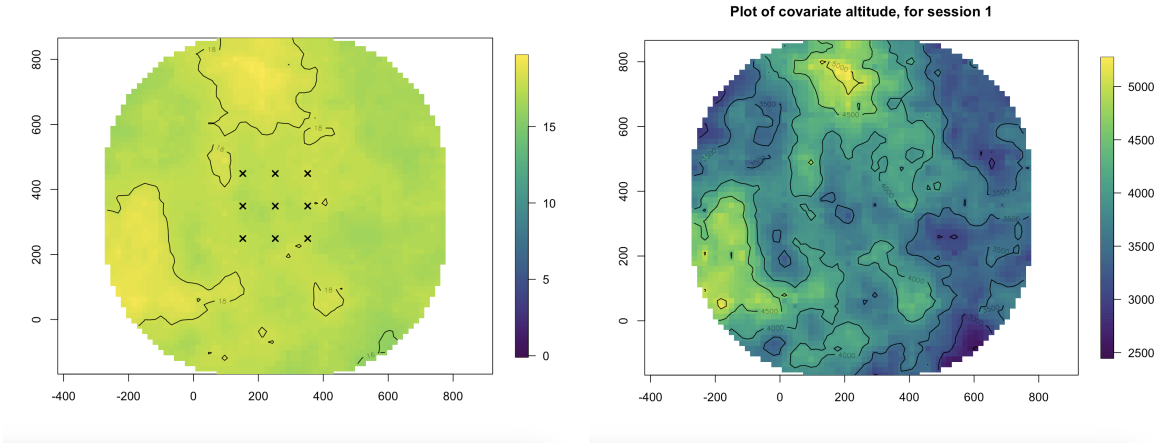


Figure 3.4: Plot of altitude and estimated density when density is modelled using altitude as a covariate.

3.5 Model where Density Varies with Distance to Nearest Village

The last single covariate used to model density alone in this demonstration is distance to nearest village.

```
#Create a model where density depends on village locations  --------------------
acre_model1.5 <- fit.acre(acre_object1, par_extend_model = list(D =~villages))
```

```
#Summary output ----------------------------------------------------------------
summary(acre_model1.5)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
##                    Estimate  Std. Error       2.5 %  97.5 %
## g0              9.5875e-01  1.8594e-02  9.0243e-01  0.9832
## sigma           8.5821e+01  1.2319e+00  8.3440e+01 88.2700
## kappa           2.1819e+01  1.2285e+00  1.9540e+01 24.3652
## D.(Intercept)   2.9001e+00  6.2056e-02  2.7785e+00  3.0218
## D.villages     -3.6749e-05  1.4442e-05 -6.5055e-05  0.0000
##
##
##  Extended parameters link functions:
## D : log
```

```
#AIC ------------------------------------------------------------------------
AIC(acre_model1.5)
```

```
## [1] -4579.119
```

The resulting AIC (-4579.119) is very similar to that produced by `acre_model1.4`.

While conducting this project, it was discovered that the `show_Dsurf()` function used to display density plots contains a bug which prevents plotting when objects that were given to the `d_cov` argument within `read.acre` are used as covariates in models. Therefore, any models using `villages` as a covariate do not currently have the ability to have their density estimates plotted. This has been raised as a GitHub issue and will be resolved by the package author.

3.6 Full Model

The next model in this demonstration is the full model; all covariates are used as explanatory variables for their suitable parameters, a distinguishing feature of `acre`.

```
#Create a full model  -------------------------------------------------------
acre_model1.6 <- fit.acre(acre_object1,
                          par_extend_model = list(D =~forest +
                                                    protected +
                                                    altitude +
                                                    villages,
                                                  g0 =~weather +
                                                    terrain,
                                                  kappa =~terrain))
```

```
#AIC ------------------------------------------------------------------------
AIC(acre_model1.6)
```

```
## [1] -5512.784
```

Unsurprisingly, the AIC value produced here (-5512.8) is the lowest of all models in this demonstration. This reveals that `acre` is correctly identifying the importance of added covariates; since all covariates affected parameters within the simulation, the importance of all covariates should be reflected in model accuracy.

Before examining the accuracy of the model's summary output, it is helpful to plot the estimated detection functions when different covariate conditions exist. `show_detfun()` accepts an argument `new_covariates` which takes in a data frame of particular covariates; in this case it may be used for particular weather conditions or terrain. For example, the following code produces the model's detection function for a sunny mountainous session compared to a rainy mountainous session. The output is displayed in Figure 3.5.

```
show_detfn(acre_model1.6, new_covariates = data.frame(weather = c("sun", "rain"),
                                                      terrain = rep("mountain", 2)),
           col= c("red", "blue"))
legend("topright", legend = c("Sunny, mountainous", "Rainy, mountainous"),
       lty = 1, col = c("red", "blue"))
```

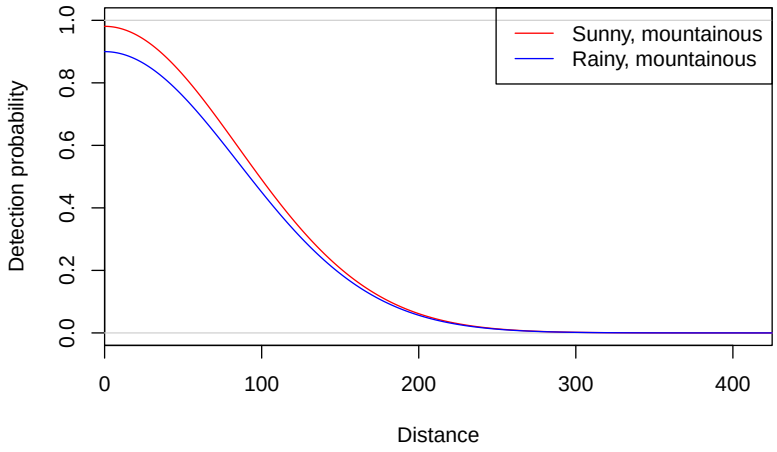


Figure 3.5: The detection functions of a sunny mountainous session compared to a rainy mountainous session.

This visually demonstrates that the model has identified that detection is generally better when a session is sunny compared to when it is rainy, which is true to the simulation. It should be noted that in this particular simulation, only two weather conditions were present in the sessions: sun and rain. Therefore, this model does not identify any other weather conditions.

Detection functions could also be used to compare the effects of mountainous and non-mountainous sessions on detection, as below. The output for this code is seen in Figure 3.6.

```
show_detfn(acre_model1.6, new_covariates = data.frame(weather = rep("sun", 2),
                                                      terrain = c("mountain",
                                                                  "non-mountain")),
           col= c("red", "blue"))
legend("topright", legend = c("Sunny, mountainous", "Sunny, non-mountainous"),
       lty = 1, col = c("red", "blue"))
```

Here it can be seen that mountains have a marginally negative impact on detection probabilities, which is also true to the simulation.

After visually observing the model's estimated effects that covariates have on detection probability, it is then a good idea to examine the full model's summary output.

```
#Summary output ---------------------------------------------------------------
summary(acre_model1.6)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
```

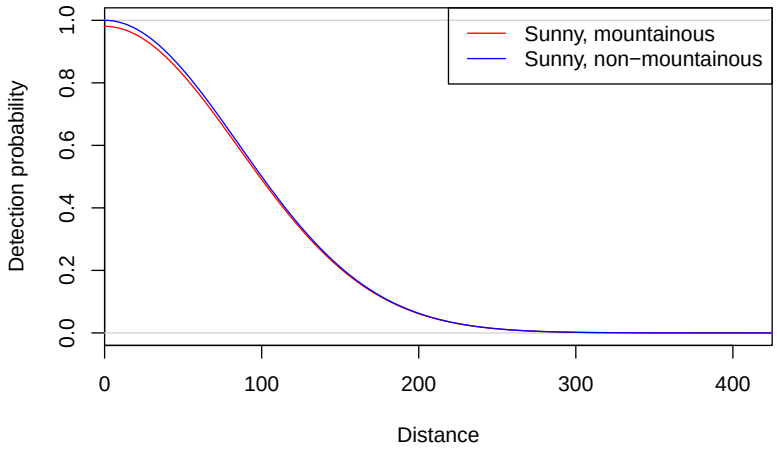


Figure 3.6: The detection functions of a sunny mountainous session compared to a sunny non-mountainous session.

```
##  Parameters:
##                                Estimate  Std. Error       2.5 %    97.5 %
## g0.(Intercept)              2.1961e+00  3.1716e-01  1.5745e+00    2.8177
## g0.weathersun               1.7379e+00  1.3438e+00 -8.9588e-01    4.3716
## g0.terrainnon-mountain      1.2815e+01  1.6229e+03 -3.1681e+03 3193.6895
## sigma                       8.4991e+01  1.1105e+00  8.2842e+01   87.1951
## kappa.(Intercept)           2.9305e+00  6.1791e-02  2.8093e+00    3.0516
## kappa.terrainnon-mountain   8.7206e-01  1.2643e-01  6.2426e-01    1.1199
## D.(Intercept)              -9.0922e-02  4.8952e-01 -1.0504e+00    0.8685
## D.forest1                   2.7079e+00  2.6243e-01  2.1936e+00    3.2223
## D.protected1                1.6686e+00  1.1062e-01  1.4518e+00    1.8854
## D.altitude                 -1.3934e-04  9.7546e-05 -3.3052e-04    0.0001
## D.villages                  3.0025e-05  5.2016e-05 -7.1924e-05    0.0001
##
##
##  Extended parameters link functions:
## g0 : logit
## kappa : log
## D : log
```

Here, `g0.(Intercept)` corresponds to the logit of the initial value of g_0 at baseline factor variables (`g0_base`, as described in Section 2.1.6). Therefore, to examine the accuracy of the model's estimate of the baseline value of g_0, the inverse logit can be used as follows:

```
plogis(2.1961)
```

```
## [1] 0.8998987
```

Pleasingly, this result is very close to the true value of `g0_base` (0.85) that was passed into the simulation.

Performing the same operation on the limits of the estimate's 95% confidence interval produces an interval that does contain the true value of `g0_base`:

```
cat(plogis(1.5745), plogis(2.8177))
```

```
## 0.8284242 0.9436248
```

As mentioned, this simulation only produced two of the four possible states of the weather covariate: sun and rain. Therefore, the summary output for this model only displays the effect of sun relative to the effect of rain on g_0, `g0.weathersun`. As discussed in Section 2.1.6, the 'weather' variable involved in adjusting g_0 takes the value β_5 multiplied by 2 if the session is rainy and β_5 multiplied by 4 if the session is sunny. Therefore, sun has twice the effect on g_0 as rain. The model's estimate of 1.7 for `g0.weathersun` is relatively close to this true value of 2.

In the simulation, the parameter responsible for determining the effect of mountains on g_0 is β_6. This value measures the relative effect of the presence of mountains, where absence of mountains is considered baseline. Since mountains are likely to have a negative impact on detection probability, β_6 was given a negative value in this simulation. However, this model has treated the presence of mountains as the baseline, and has generated `g0.terrainnon-mountain`, which is the relative effect of *not* having mountains in a session. It can therefore be expected that `g0.terrainnon-mountain` has a negative relationship to β_6. The value of β_6 in the simulation was -0.2, while the estimated value of `g0.terrainnon-mountain` is 12.8, and this directional difference is to be expected. However, it should be noted that the confidence interval for `g0.terrainnon-mountain` is very wide and contains 0; this is likely because there is not enough information provided from only three sessions when effect size of mountains on g_0 is small and weather is simultaneously affecting the same parameter. Of the three generated sessions, one is sunny and mountainous, one is rainy and mountainous, and one is sunny and non-mountainous; if a fourth session that happened to be rainy and non-mountainous existed, then a narrower confidence interval for `g0.terrainnon-mountain` could be expected. Narrower confidence intervals are observed for estimates using the other simulation examples in this project's GitHub repository, which have greater numbers of sessions.

`sigma`, which is quite obviously the estimate of σ, has been estimated to a very high degree of accuracy here: 84.99 compared to the true value of 85.

When it comes to the coefficient estimates involved in adjusting κ (the concentration parameter used in the von Mises distribution), this model also does a good job. The estimate for `kappa.(Intercept)`, which corresponds to `cp_beta0`, is 2.9, while the true value passed to the simulation was log(50), which is roughly 3.9. `kappa.terrainnon-mountain` is equivalent to the negative of `cp_beta1`, on account of the baseline being presence of mountains, rather than absence of them. The true value of `cp_beta1` was -1, while the estimate of `kappa.terrainnon-mountain` is 0.87, and this directional difference is correct. The negative of the true value is 1, which is contained in the confidence interval for `kappa.terrainnon-mountain` (0.6, 1.1).

The estimates `D.forest1`, `D.protected1`, `D.altitude`, and `D.villages` can be compared to their true values β_3, β_7, β_8, and β_4 respectively. Table 3.1 shows this comparison.

Table 3.1: Comparison of true parameters and model estimates.

Parameter	Name in summary output	True value	Model estimate
β_3	D.forest1	2.50	2.70
β_7	D.protected1	1.90	1.70
β_8	D.altitude (per km)	-0.01	-0.14

Parameter	Name in summary output	True value	Model estimate
β_4	D.villages (per km)	0.15	0.03

Estimates for β_3 and β_7 are very close to their true values. Given how minute the effects of altitude and distance to villages are, it is not concerning that β_4 and β_8 are not as proportionally close to their true values.

Overall, the accuracy of the full model's estimates can be visualised in Figure 3.7.

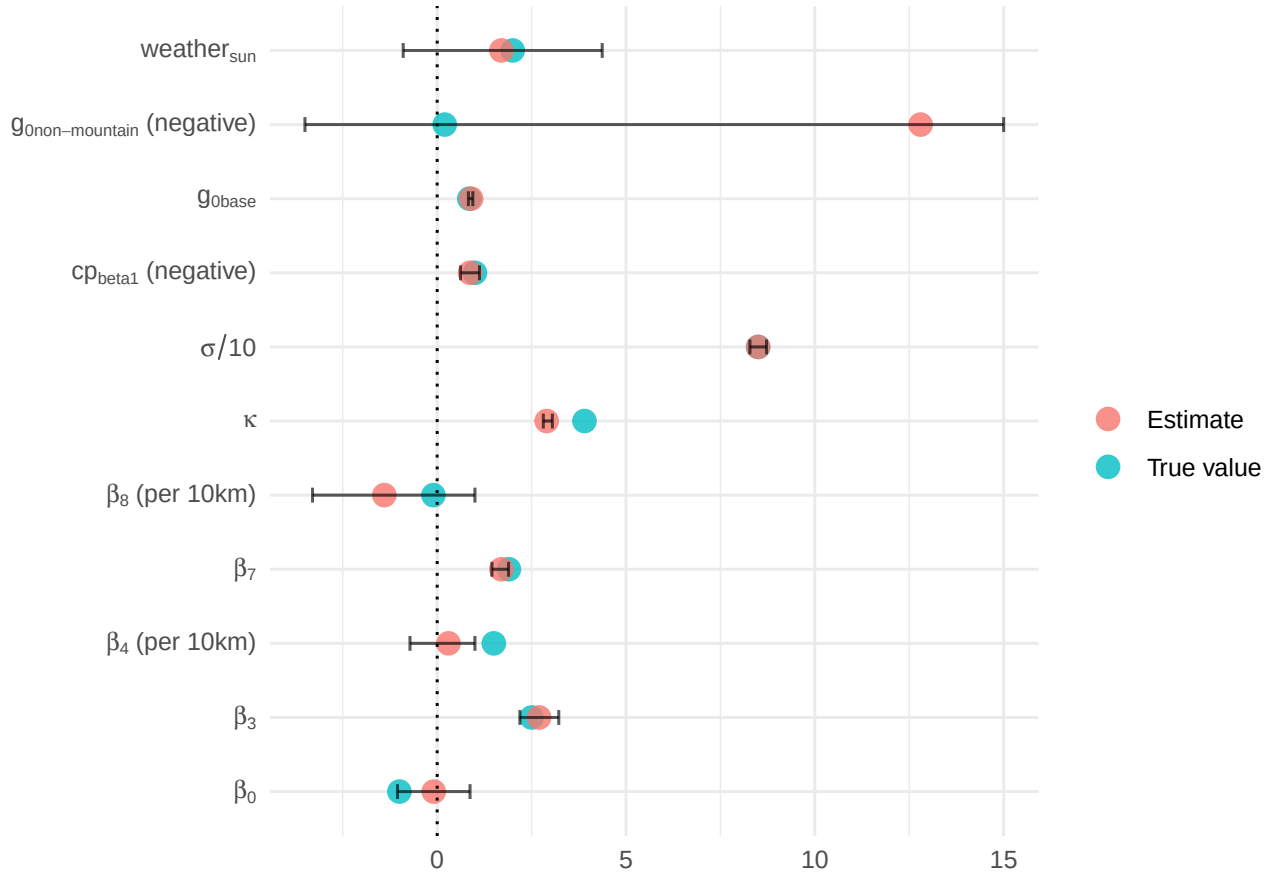


Figure 3.7: Model estimates (with 95% confidence intervals) and their true parameter values. *Note: the confidence interval limits for $g_{0non-mountain}$ have been truncated, and σ, β_4, and β_8 have been scaled in order to better fit on the plot.*

Since this model contains `villages` as a covariate, it is not possible to plot its density estimates with the current version of `acre`. So that some version of the plot may be illustrated, another 'full' model will be created, omitting `villages` only.

```
#Create a full model (minus villages) ----------------------------------------
acre_model1.6.1 <- fit.acre(acre_object1,
                            par_extend_model = list(D =~forest +
                                                    protected +
```

```
                                                    altitude,
                                                  g0 =~weather +
                                                    terrain,
                                                  kappa =~terrain))
```

With the consideration in mind that village locations were not accounted for in this model, a density plot can now be produced using this less complete model, so as to get at least some visual idea of density. This can be directly compared to the true locations of all the animals (detected and undetected) in the study area.

```
par(mfrow=c(1,2))
show_Dsurf(acre_model1.6.1, session = 1)
plot(session_animal_locations[[1]], pch = 19,
     ask = FALSE, xlab = "", ylab = "",
     col = rgb(0.5, 0.6, 0.1, 0.25),
     xlim = c(-250, 800),
     ylim = c(-400, 1100))
```

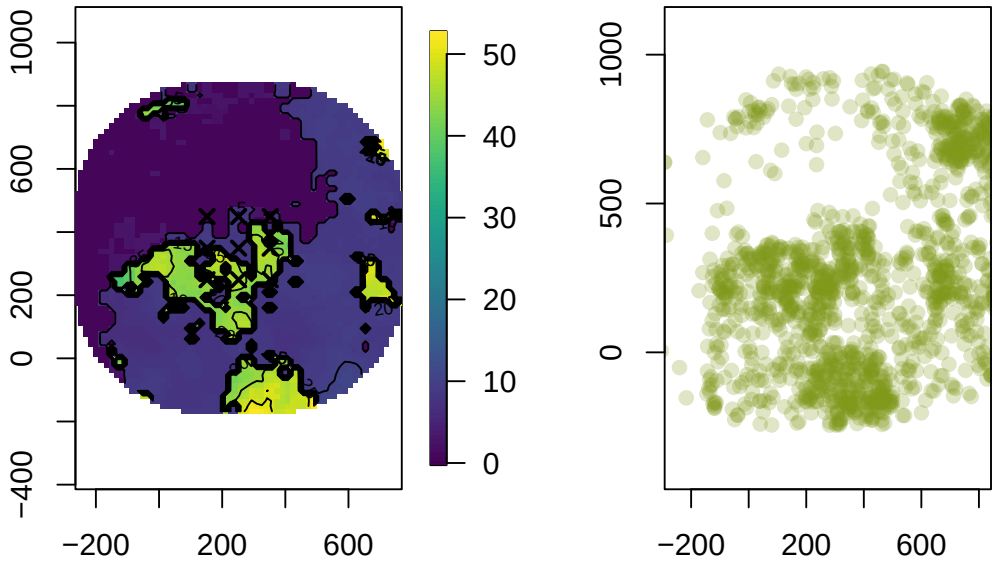


Figure 3.8: Plot of the 'full' model's estimated density surface (left) compared to the true locations of animals (right).

This plot reveals a few key areas of higher density, while much of the study area contains few to no animals. Interestingly, the shapes of forest areas, protected areas, and regions of high/low altitude can all be distinguished upon the density surface; the shapes of each of these appear layered upon one

another. This demonstrates that the model is correctly accounting for all spatial effects. Furthermore, comparing the estimated density surface to the true locations of all animals in the study area reveals that the model has correctly identified regions of high, medium, and low density.

3.7 Modelling Density Only

In the interest of comparing `acre` directly to `ascr`, a model using all spatial covariates to model density alone should be created, as this is what constitutes a 'full model' in `ascr`. Distance to nearest village has been deliberately left out in order to allow for plotting.

```
#Create a model where density depends on forest, protected areas and altitude
acre_model1.6.2 <- fit.acre(acre_object1,
                            par_extend_model = list(D =~forest +
                                                      protected +
                                                      altitude))
```

A discussion of the summary output of this model and an evaluation of its performance can be found in Chapter 4.

3.8 Full Model when Less Covariate Information is Known

As discussed earlier in this chapter, covariate information is often not known for all mask points. It is important to evaluate `acre`'s accuracy at extrapolating from the covariate information it does receive, as well as ensuring that limited covariate information doesn't drastically impact parameter estimates.

To simulate this, a random subset of covariate data can be taken and used to create a 'less complete' `acre` object. Here, the `sample()` function is employed to randomly select 500 covariate data points from the previously created covariate data frame, `cov_df`.

```
#Create an acre object with incomplete covariate information -------------------
acre_object1_incomplete <- read.acre(all_capt_hist1, all_traps1,
                            control_create_mask = list(buffer = 85*5),
                            loc_cov = cov_df1[sample(nrow(cov_df1), 500),],
                            session_cov = sessions1,
                            dist_cov = list(villages = village_locations1))
```

A full model is then built from this new `acre` object:

```
#Create a full model with incomplete covariate information ---------------------
acre_model1.7 <- fit.acre(acre_object1_incomplete,
                            par_extend_model = list(D =~forest +
                                                      protected +
                                                      altitude +
                                                      villages,
                                                    g0 =~weather +
                                                      terrain,
                                                    kappa =~terrain))
```

The summary output and AIC value may then be compared to the model where all covariate information was known. Table 3.2 demonstrates the differences in the parameter estimates between this model and `acre_model1.6`.

```
#Summary output ------------------------------------------------------------
summary(acre_model1.7)
```

Table 3.2: Comparison of model parameter estimates.

Parameter	Complete information	Incomplete information
g0.(Intercept)	2.196	2.410
g0.weathersun	1.738	1.187
g0.terrainnon-mountain	12.815	2.394
sigma	84.991	85.923
kappa.(Intercept)	2.930	2.914
kappa.terrainnon-mountain	0.872	0.848
D.(Intercept)	-0.091	-1.600
D.forest1	2.708	2.168
D.protected1	1.669	0.657
D.altitude (per km)	-0.139	0.529
D.villages (per km)	0.030	0.321

All parameter estimates have stayed very close to the estimates produced by the model based off full covariate information.

```
#AIC ------------------------------------------------------------------------
AIC(acre_model1.7)
```

```
## [1] -5160.166
```

As is to be expected, a model based on less complete information results in a higher AIC value than the model with information across all mask cells, but the parameter and coefficient estimates maintain a high degree of accuracy.

The reason for the conservation of accuracy is `acre`'s process of spatial interpolation. It takes the covariate information it receives for the mask cells for which it is available and extrapolates to estimate covariate information for the remaining cells. The precision with which it performs this process can be examined by plotting the distribution of the spatial covariates produced by the incomplete `acre` object and comparing it to the true distribution of spatial covariates which was produced by the complete `acre` object. These comparisons are demonstrated in Figure 3.9.

```
plot(acre_object1, types = "covariates")
plot(acre_object1_incomplete, types = "covariates")
```

It is obvious here that `acre` has done a good job at approximating the overall distributions of all the spatial covariates. The general patterns and shapes produced by plotting forest coverage, protected area, and altitude are all akin to their true distributions within the study area; it could be said that it has simply resulted in a 'smoothened' approximation of the truth.

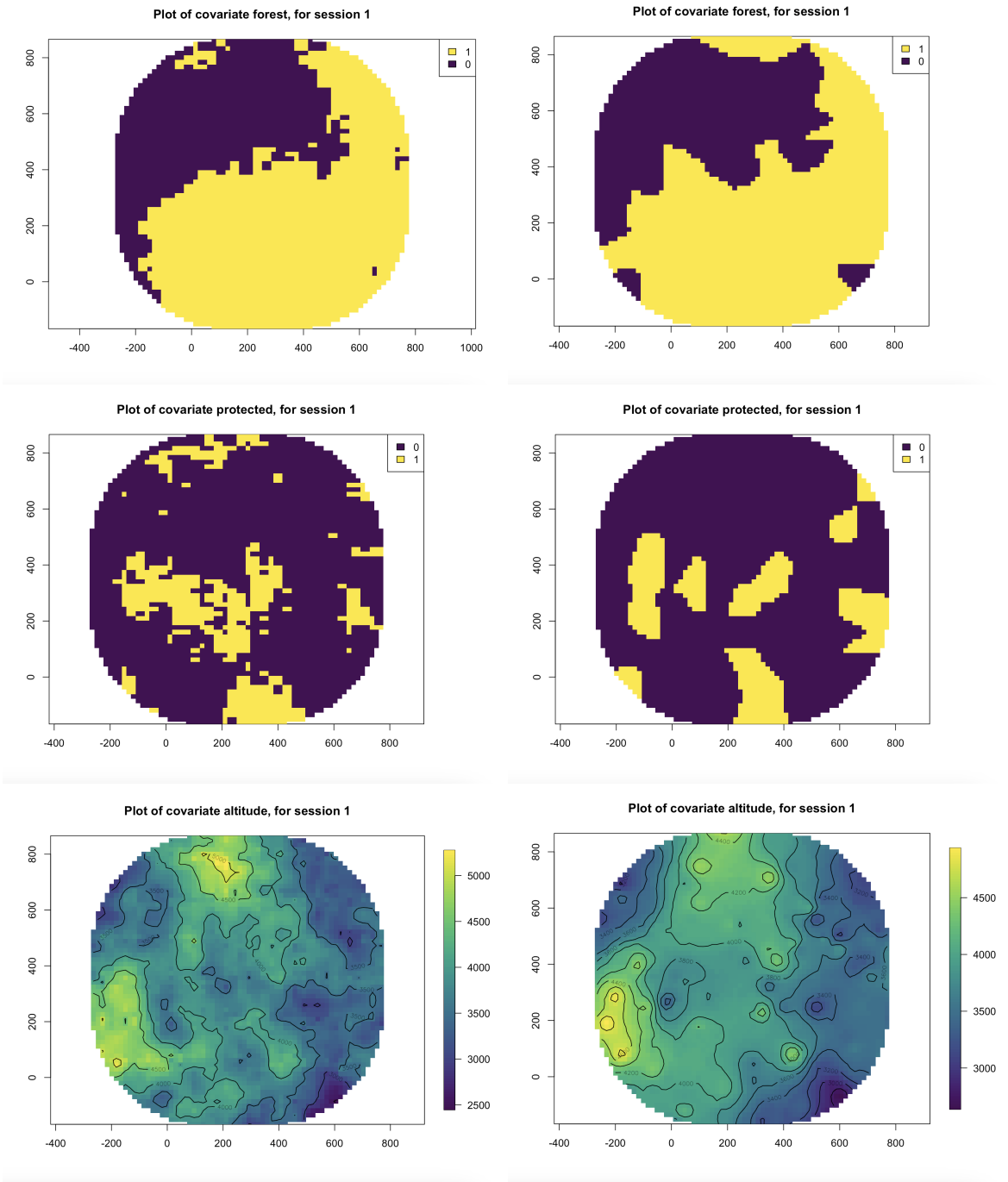


Figure 3.9: Plots of true spatial covariates (left) and estimated spatial covariates when information is only known for some mask cells (right).

Once again, it is not possible to plot the density estimates of this model, as it contains `villages` as a covariate. Therefore another model will replicate this model, without the village locations, in order to illustrate density estimates from this model visually. The density surface for this model, compared to that of the 'full information' model, is shown in Figure 3.10.

```
#Create a full model with incomplete covariate information, without villages ---
acre_model1.7.1 <- fit.acre(acre_object1_incomplete,
                            par_extend_model = list(D =~forest +
                                                      protected +
                                                      altitude,
                                                    g0 =~weather +
                                                      terrain,
                                                    kappa =~terrain))
```

```
par(mfrow=c(1,2))
show_Dsurf(acre_model1.7.1, session = 1)
show_Dsurf(acre_model1.6.1, session = 1)
```

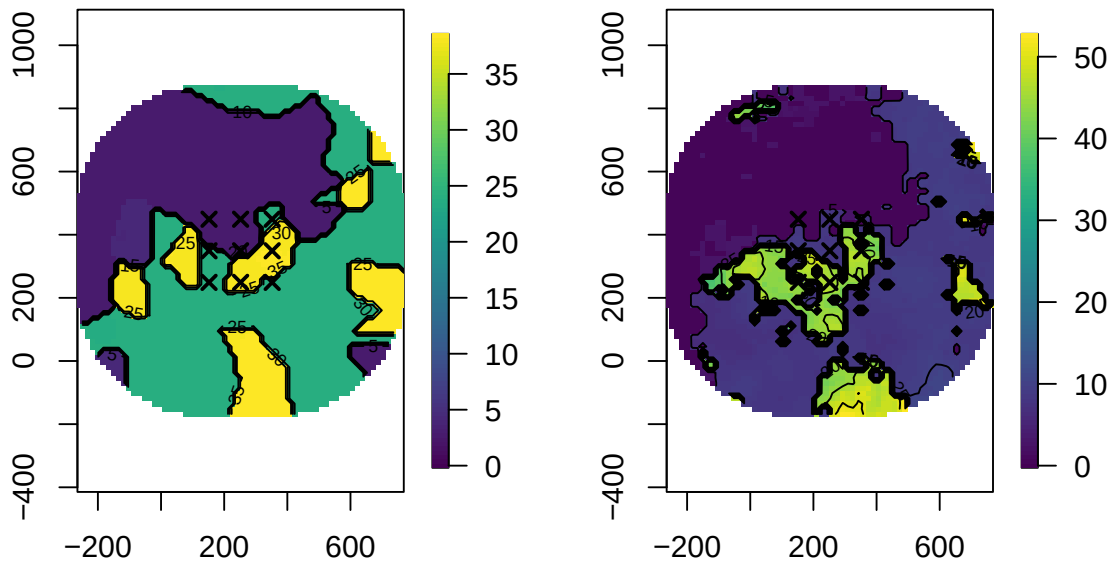


Figure 3.10: Plot of the 'full' model's estimated density surface when not all covariates are known for each mask cell (left), compared to the density surface produced by the same model when all covariates are known for each mask cell (right).

Not unlike the estimated spatial covariate distributions, the estimated density surface produced by this model can be considered a 'smoothened' version of the density surface constructed by

`acre_model1.6.1`. The overall density patterns are recognisable, and elements of the spatial covariates can be distinguished.

A key difference is that the model estimates more 'mid-range' density areas, rather than mostly areas that are either high or low density. This is an interesting result of `acre`'s spatial interpolation process. When covariate information isn't known across all mask cells, `acre` must make more general estimates of regions of covariates. For example, in this simulation, there are regions within the study area that could be considered 'mostly protected area', despite having smaller subregions within them that are not protected areas. During spatial interpolation, `acre` must determine the broader regions to be 'mostly protected area' and therefore the non-protected area subregions within them become generalised as part of the protected region too. This is clearly seen in Figure 3.9. Therefore, rather than comparing the effects of 'true protected areas' to 'true non-protected areas', `acre` must compare effects of 'mostly protected areas' to 'mostly non-protected areas'. A consequence of this is that the effect of protected areas is slightly dampened, and therefore a less extreme outcome is seen in the comparison of density estimates between protected and non-protected areas. Thus, density estimates can be expected to be lower for models fitted with less covariate information when those covariates have the effect of increasing density.

Chapter 4

Using Simulated Data with ascr vs. acre

Not all the models produced with `acre` in the preceding chapter can be reproduced with `ascr`, on account of `ascr`'s more limited features. However, where `acre` models can be reproduced and compared to their corresponding `ascr` models, they should. As discussed in the Introduction, when evaluating new software, the output of established software should be used as a premise of 'truth'. This project's GitHub repository contains a file 'acre_vs_ascr.R' which contains a more comprehensive version of the code that forms the basis of this final chapter.

`ascr` can be downloaded from the following GitHub repository and then loaded into R using `library(ascr)`.

4.1 Preparing to Use ascr

Firstly, `ascr` requires capture histories and detector locations to be stored in a named list, which can be done as shown here:

```
#ascr requires capture histories and traps to be in a named list format --------
ascr_capt <- create.capt(captures = all_capt_hist1, traps = all_traps1)
```

To allow for a fair comparison of the packages, the same mask used in the `acre` models will be used here. This can be extracted from `acre_object1` as shown below.

```
#Use the same mask used in acre to ensure a fair comparison --------------------
ascr_mask1 <- acre_object1$mask
```

The final preparation step is to create a list, where each list element contains a data frame of covariates from one session.

```
#Create a list of data frames (one for each session) with covariate information
cov_df_ascr <- list()
for(i in 1:nrow(sessions1)) {
  cov_df_ascr[[i]] <- subset(acre_object1$par.extend$data$mask, session == i)
}
```

4.2 Null Model

The first comparison available here is naturally the null model. The null model can be created simply with the following code:

```
#Create the null model ------------------------------------------------------
ascr_model1.1 <- fit.ascr(capt = ascr_capt, traps = all_traps1, mask = ascr_mask1)
```

The summary output can then be compared to the summary output of the null model created with `acre`:

```
#View summary and compare to acre's null model summary ------------------------
summary(ascr_model1.1)
```

```
## Detection function: Halfnormal
## Information types: Bearings
##
##  Parameters:
##       Estimate Std. Error
## D     15.94228     0.5815
## g0     0.95873     0.0186
## sigma 85.82716     1.2321
## kappa 21.82009     1.2286
## ---
##
## esa.1 20.11420     0.3430
## esa.2 20.11420     0.3430
## esa.3 20.11420     0.3430
```

```
summary(acre_model1.1)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
##        Estimate Std. Error     2.5 %  97.5 %
## g0     0.958730   0.018595  0.902416  0.9832
## sigma 85.827157   1.232124 83.445895 88.2764
## kappa 21.820093   1.228559 19.540268 24.3659
## D     15.942275   0.581462 14.842411 17.1236
```

Besides differences in rounding, estimates and standard errors are exactly the same for both models.

Density surfaces may also be plotted for a visual comparison of both models (Figure 4.1). Given that both models estimate a constant density of 15.9 animals per hectare across the entire study area, the density surfaces for the null models are identical.

```
#View density surface and compare to acre's null model density surface ---------
par(mfrow=c(1, 2))
show.Dsurf(ascr_model1.1)
show_Dsurf(acre_model1.1)
```

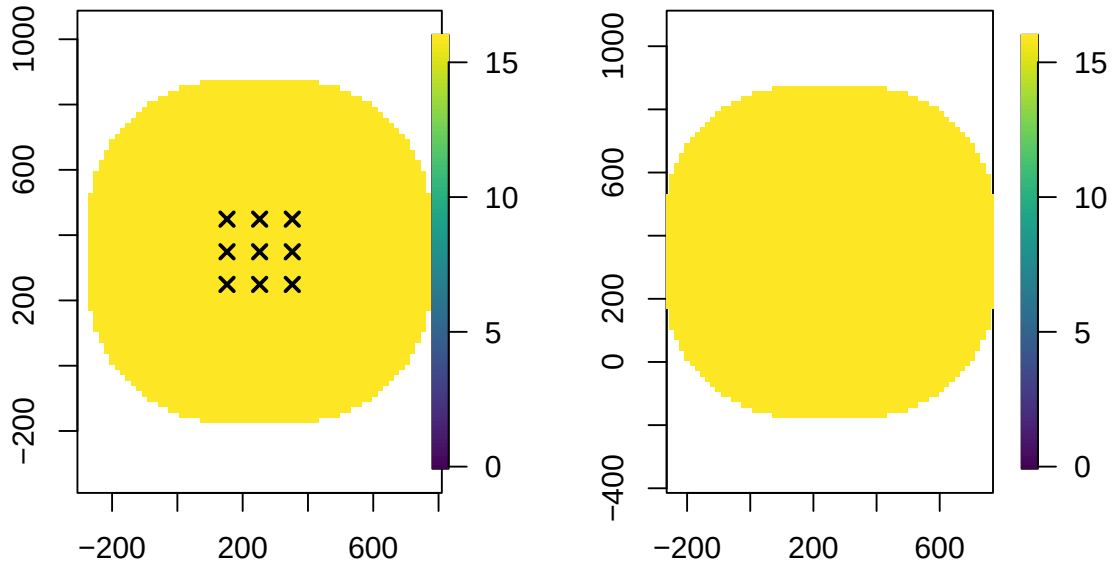


Figure 4.1: Null model density surfaces produced by ascr (left) and acre (right)

4.3 Models where Density Depends on One Covariate

Next, models where density depends on a single covariate can be created with `ascr` and compared to their `acre` counterparts. This is done via the argument `ihd.opts` which is passed to `fit.ascr()`. `ihd.opts` accepts a named list, where one component is the model formula and another is the data frame containing all the covariate information.

4.3.1 Density depends on forest coverage

Modelling density using forest coverage as the only covariate can be done with the following code. The summary outputs reveal that `acre` has been faithful to its 'source of truth'. A plot of the estimated density surface compared to its corresponding `acre` model (Figure 4.2) further highlights `acre`'s fidelity to `ascr`.

```r
#Create model where density depends on forest coverage ------------------------
ascr_model1.2 <- fit.ascr(capt = ascr_capt, traps = all_traps1, mask = ascr_mask1,
                          ihd.opts = list(model = ~forest,
                                          covariates = cov_df_ascr,
                                          scale = FALSE))
```

```r
#View summary and compare to summary of corresponding acre model ---------------
summary(ascr_model1.2)
```

```
## Detection function: Halfnormal
## Information types: Bearings
##
##  Parameters:
##               Estimate Std. Error
## D.(Intercept) -0.18349     0.2934
## D.forest1      3.51126     0.2982
## g0             0.96248     0.0183
## sigma         84.71980     1.1610
## kappa         21.68607     1.2059
## ---
## esa.1         19.79510     0.3156
## esa.2         19.79510     0.3156
## esa.3         19.79510     0.3156
```

```r
summary(acre_model1.2)
```

```
## Detection function: Halfnormal
## Number of sessions: 3
## Information types: Bearings
## Confidence interval method: Wald
##
##
##  Parameters:
##                Estimate Std. Error     2.5 %  97.5 %
## g0             0.962475   0.018317  0.904693  0.9858
## sigma         84.719804   1.160984 82.474603 87.0261
## kappa         21.686067   1.205942 19.446716 24.1833
## D.(Intercept) -0.183487   0.293357 -0.758457  0.3915
## D.forest1      3.511265   0.298158  2.926886  4.0956
##
##
##  Extended parameters link functions:
## D : log
```

```r
#View density surface and compare density surface of corresponding acre model --
par(mfrow=c(1, 2))
show.Dsurf(ascr_model1.2)
show_Dsurf(acre_model1.2)
```

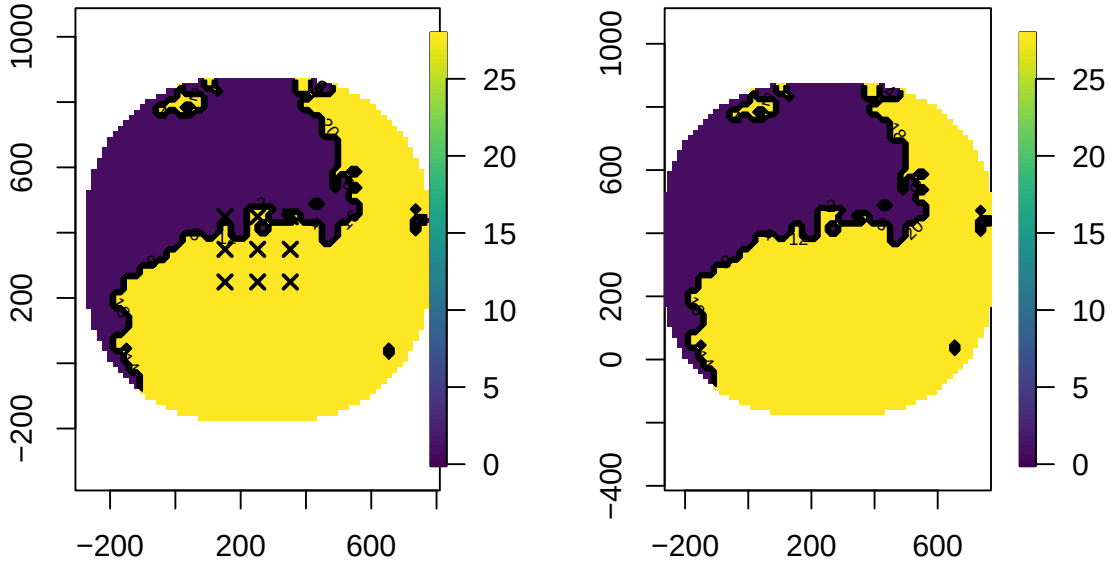


Figure 4.2: Forest coverage model density surfaces produced by ascr (left) and acre (right)

4.3.2 Density depends on protected areas

The same process can be used to model density with respect to protected areas. Summary outputs reveal that estimates produced by `acre` are virtually identical to those produced by `ascr` (there only exist differences in rounding), and Figure 4.3 displays the resulting density surfaces generated by each package.

```
#Create model where density depends on protected areas -------------------------
ascr_model1.3 <- fit.ascr(capt = ascr_capt, traps = all_traps1, mask = ascr_mask1,
                          ihd.opts = list(model = ~protected,
                                          covariates = cov_df_ascr,
                                          scale = FALSE))
```

```
#View summary and compare to summary of corresponding acre model ---------------
summary(ascr_model1.3)
summary(acre_model1.3)
```

```
#View density surface and compare density surface of corresponding acre model --
par(mfrow=c(1, 2))
show.Dsurf(ascr_model1.3)
show_Dsurf(acre_model1.3)
```

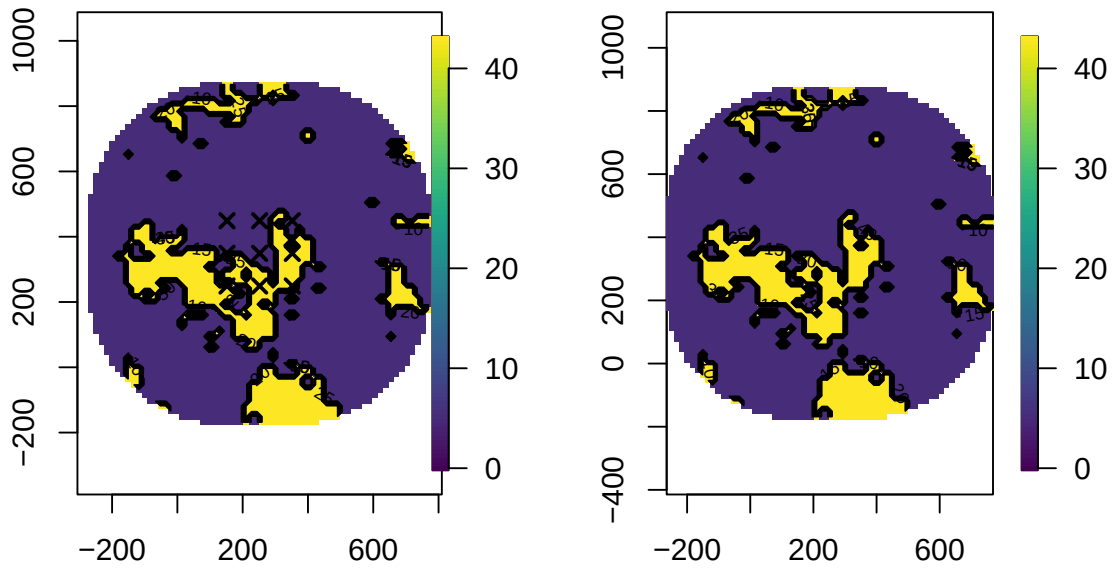


Figure 4.3: Protected areas model density surfaces produced by ascr (left) and acre (right)

4.3.3 Density depends on altitude

Estimates when modelling density using altitude alone, once again, produce the same estimates by both packages.

```
#Create model where density depends on altitude --------------------------------
ascr_model1.4 <- fit.ascr(capt = ascr_capt, traps = all_traps1, mask = ascr_mask1,
                          ihd.opts = list(model = ~altitude,
                                          covariates = cov_df_ascr,
                                          scale = FALSE))
```

```
#View summary and compare to summary of corresponding acre model ---------------
summary(ascr_model1.4)
summary(acre_model1.4)
```

```
#View density surface and compare density surface of corresponding acre model --
par(mfrow=c(1, 2))
show.Dsurf(ascr_model1.4)
show_Dsurf(acre_model1.4)
```

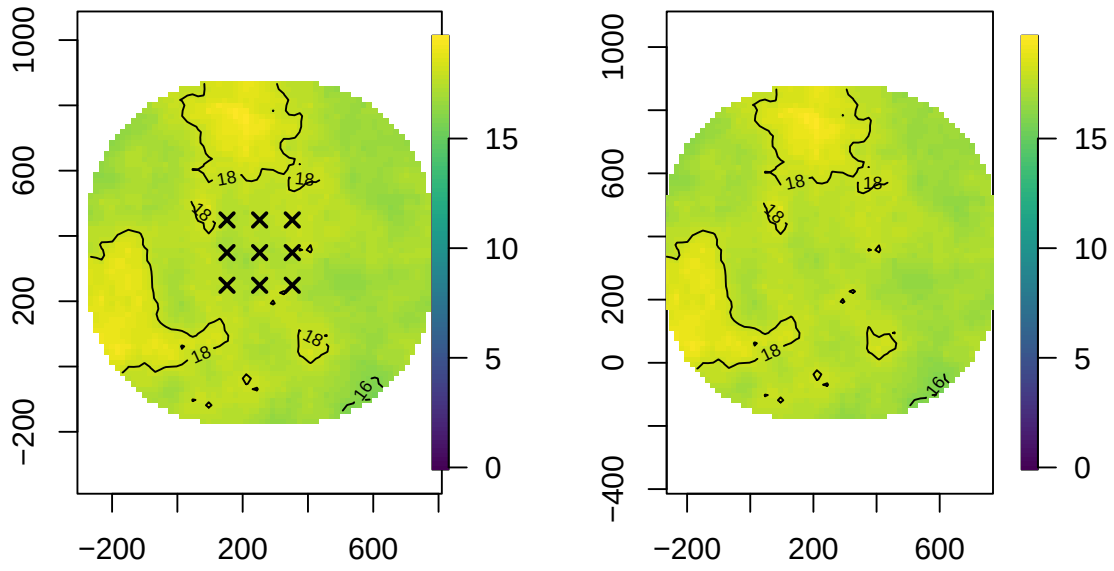


Figure 4.4: Altitude model density surfaces produced by ascr (left) and acre (right)

4.3.4 Density depends on distance to nearest village

The last single covariate used to model density is distance to nearest village. As with the previous models, estimates produced by `ascr` match those produced by `acre`.

```
#Create model where density depends on distance to nearest village -------------
ascr_model1.5 <- fit.ascr(capt = ascr_capt, traps = all_traps1, mask = ascr_mask1,
                          ihd.opts = list(model = ~villages,
                                          covariates = cov_df_ascr,
                                          scale = FALSE))
```

```
#View summary and compare to summary of corresponding acre model ---------------
summary(ascr_model1.5)
summary(acre_model1.5)
```

Unfortunately, due to the aforementioned `acre` bug, only the `ascr` model's density surface can be plotted in this case (Figure 4.5).

```
#View density surface (cannot compare this to acre, due to bug)-----------------
show.Dsurf(ascr_model1.5)
```

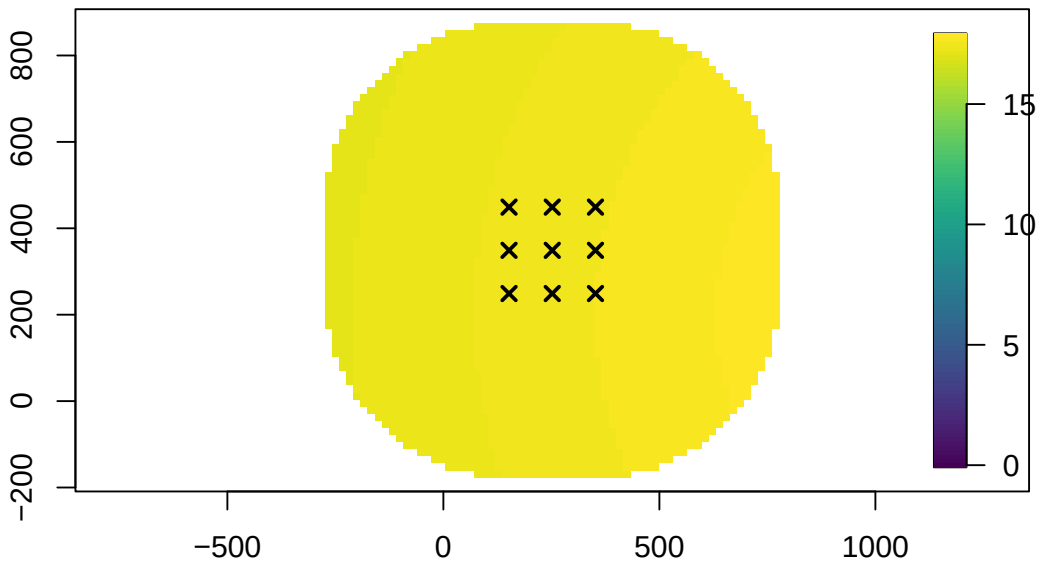


Figure 4.5: Distance to nearest village model density surface produced by ascr

4.4 Full Model

For the purpose of this comparison, a 'full model' is considered to use forest coverage, protected area, and altitude to explain density only, since it is not possible to model other parameters with `ascr`. Since the inclusion of distance to nearest village impedes the ability to plot the resulting density surface from an `acre` model, the village covariate has been omitted. Comparing the summary output to that of `acre_model1.6.2`, which was created in Section 3.7, reveals once again that the estimates are same as those produced by `acre`.

```
#Create model where density depends on forest coverage, protected areas, and
#altitude ------------------------------------------------------------------------
ascr_model1.6 <- fit.ascr(capt = ascr_capt, traps = all_traps1, mask = ascr_mask1,
                          ihd.opts = list(model = ~forest + protected + altitude,
                                          covariates = cov_df_ascr,
                                          scale = FALSE))
```

```
#View summary and compare to summary of corresponding acre model ---------------
summary(ascr_model1.6)
summary(acre_model1.6.2)
```

```
#View density surface and compare density surface of corresponding acre model --
par(mfrow=c(1, 2))
show.Dsurf(ascr_model1.6)
show_Dsurf(acre_model1.6.2)
```

Given that `ascr` requires covariate information to be known for all mask cells, it is not possible to directly compare `acre`'s models that were created with less complete covariate information to any `ascr` models.

On the whole, `acre` is aligned with `ascr` in all of its estimates; no discrepancies were found. Ultimately, there exist no concerning differences between the output of `acre` and `ascr` based on the explorations with the simulated data used in this project.

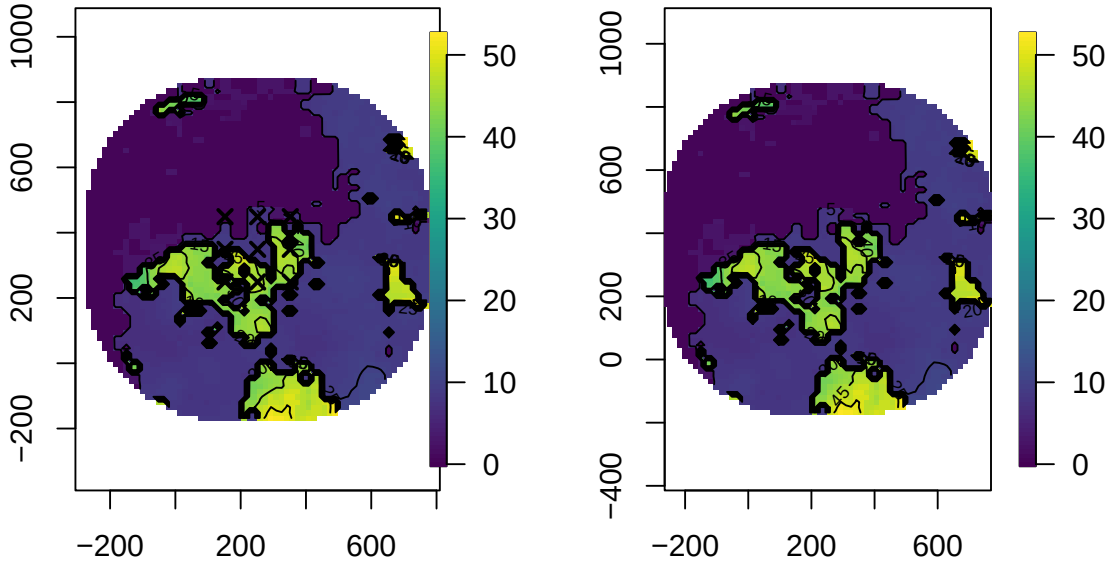


Figure 4.6: Full model density surfaces produced by ascr (left) and acre (right)

Chapter 5

Conclusion

5.1 Results

The first discovery of this project was that `acre` lacked the ability to account for spatial covariates that change over time. In order to realistically mimic the real world, many of the simulated data sets contained spatial covariates that changed over time, even when they occurred in the same study areas. In reality, this may occur when areas are subject to deforestation or new areas are ruled to be 'protected'. When this situation arose, `read.acre()` was only able to accept one version of spatial covariates and ignored any changes that might have occurred. As a result of this finding, the issue was raised to the package author and promptly resolved.

Next, a bug in the `show_Dsurf()` function was identified: when an `acre` model uses a distance covariate (such as distance to nearest village), `show_Dsurf()` is unable to produce a plot of its estimated density surface. This is an important finding, as `show_Dsurf()` is a key package function. An issue has been raised on GitHub to address this bug.

`fit.acre()` performs well, even when covariate information is not known for all mask cells within the study area. Given that this feature is not available in `ascr` and that it is common for researchers to be in this position, validation of this property is very valuable. It was also interesting to find that the spatial interpolation process used to fill in missing covariate information results in dampened covariate effects and therefore resulted in more 'mid range' density estimates.

Finally, and perhaps most importantly, `acre` produces accurate model estimates, both with respect to their true values and with respect to `ascr`'s estimates.

5.2 Improvements to the simulation

While the results of this project were, on the whole, valuable in the validation of `acre`, going forward, the simulation would benefit from some additional features:

- *Ability to use detection functions other than the half-normal*: The type of detection function could be chosen randomly by the simulation or decided by the user, in order to produce a wider variety of resulting data.
- *Ability for altitude to be below sea-level*: Having altitude that is below sea-level could introduce an additional type of terrain—water—which could engender its own effects on detection probability

or measurement error in observed bearings. This may lead to more interesting data, and, as a result, more findings about `acre` functions and how they tolerate various scenarios.

- *More flexible weather conditions*: Currently, the simulation only deals with 'sun', 'overcast', 'rain', and 'snow', each of which is made to linearly worsen baseline detection probabilities. Perhaps a user may wish to simulate a particular scenario which takes place in a specific climate; in a case like this, it would be beneficial to have the user select their own set of potential weather conditions and the order in which they might impact detection probabilities.

- *Parameter/coefficient checks*: Not all combinations of parameters and coefficients work in the simulation. For example, a coefficient being set too high may result in an infinite number of animals existing within the study area, or so many animals that it is not computationally realistic to use the resulting data. Testing the feasibility of the parameters and coefficients given to `simulate_capture_histories_with_sessions()` should be done before commencing with the simulation, so as to not waste a user's time or computing power.

- *Simulated parameters/coefficients*: It is not always straightforward which values, or combinations of values, for parameters and coefficients are practicable for the simulation process. Even with initial checks, a user may find themselves trying many different values before a simulation runs successfully. Rather than relying on the simulation's user to input these values, it would be helpful if the simulation could simulate them as well, while still allowing for the option of the user to input their own values if they wish. This would save time and effort, while still allowing for flexibility if particular scenarios are needed.

- *Create an R Shiny user interface*: A useful resource for users who are less familiar with R programming or who perhaps just have less time to spend, would be an R shiny application. A user could either input their own parameter values or opt to have values chosen for them, and the application could provide the resulting simulated data in a downloadable form. An additional feature of this application could include the automatic production of visualisations of various spatial covariates, and locations of sessions overlaid with their mask cells and detectors. An application like this could be a valuable resource for people who want to learn how to use the `acre` package but who do not have access to data to do so.

- *Run a large number of simulations*: While focusing on a handful of simulation examples is valuable for evaluating the usability of `acre` among various scenarios and identifying bugs, for a more rigorous test of its accuracy, the simulation should be run hundreds, perhaps thousands, of times with various parameter values, and the results examined overall. Deviation from true parameter values can then be analysed bootstrap-style. To do this, however, would require significant computing resources.

References

Agostinelli C. *Circular statistics, from "Topics in Circular Statistics" (2001)*. 2018.

Appel C, Lesmeister D, Duarte A, Davis R, Weldy M, Levi T. Using passive acoustic monitoring to estimate northern spotted owl landscape use and pair occupancy. *Ecosphere* 2023;14(2):e4421.

Barlow J, Fregosi S, Thomas L, Harris D, Griffiths E. Acoustic detection range and population density of Cuvier's beaked whales estimated from near-surface hydrophones. *The Journal of the Acoustical Society of America* 2021;149(1):111–125.

Borchers D, Efford M. Spatially explicit maximum likelihood methods for capture-recapture studies. *Biometrics* 2008;64(2):377–385.

Efford M. Density estimation in live-trapping studies. *Oikos* 2004;106(3):598–610.

Efford M. *Package "openCR"*. University of Otago; 2022.

Efford M, Jund P, Fletcher D. *Package "secr"*. University of Otago; 2023.

Genz A, Bretz F. *Computation of multivariate normal and t probabilities*. Heidelberg, Springer-Verlag; 2009.

Green A, Chynoweth M, Şekercioğlu Ç. Spatially explicit capture-recapture through camera trapping: A review of benchmark analyses for wildlife density estimation. *Frontiers in Ecology and Evolution* 2020;8.

Kidney D, Rawson B, Borchers D, Stevenson B, Marques T, Thomas L. An Efficient Acoustic Density Estimation Method with Human Detectors Applied to Gibbons in Cambodia. *PLoS ONE* 2016;11(9).

Marques T, Thomas L, Martin S, Mellinger D, Ward J, Moretti D, et al. Estimating animal population density using passive acoustics. *Biological Reviews* 2013;88(2):287–309.

McGrath S, Liu J, Stevenson B, Behie A and. Density and population size estimates of the endangered northern yellow-cheeked crested gibbon nomascus annamensis in selectively logged veun sai-siem pang national park in cambodia using acoustic spatial capture-recapture methods. *PLoS ONE* (in press).

Measey G, Stevenson B, Scott T, Altwegg R, Borchers D. Counting chirps: Acoustic monitoring of cryptic frogs. *Journal of Applied Ecology* 2017;54(3):894–902.

Nychka D, Furrer R, Paige J, Sain S. *Fields: Tools for spatial data*. Boulder, CO, USA, University Corporation for Atmospheric Research; 2021.

Otis D, Burnham K, White G, Anderson D. Statistical inference from capture data on closed animal populations. *Wildlife Monographs* 1978;62:3–135.

Owen-Ramos J, Sanchez C, Blair S, Holm S, Furnas B, Sacks B. Use of fecal DNA to estimate black bear density in an urban-wildland interface. *Wildlife Society Bulletin* 2022;46(4):e1347.

Proctor M, McLellan B, Boulanger J, Apps C, Stenhouse G, Paetkau B, et al. Ecological investigations of grizzly bears in canada using DNA from hair, 1995–2005: A review of methods and progress. *Ursus* 2010;21(2):169–188. International Association for Bear Research; Management.

Royle J, Young K. A hierarchical model for spatial capture-recapture data. *Ecology* 2008;89(8):2281–2289.

Southwood T, Henderson P. *Ecological methods, third edition*. Oxford, Blackwell Science Ltd; 2000.

Stevenson B. *Package "ascr"*. University of Auckland; 2022.

Stevenson B. *Package "acre"*. University of Auckland; 2023.

Stevenson B, Borchers D, Altwegg R, Swift R, Gillespie D, Measey G. A general framework for animal density estimation from acoustic detections across a fixed microphone array. *Methods in Ecology and Evolution* 2015;6:38–48.

Zwolinski J, Fernandes P, Marques V, Stratoudakis Y. Estimating fish abundance from acoustic surveys: Calculating variance due to acoustic backscatter and length distribution error. *Canadian Journal of Fisheries and Aquatic Sciences* 2009;66:2081–2095.