What could possibly go wrong? Troubleshooting spatially explicit capture–recapture models in **secr 4.4**

Murray Efford

2021-07-26

Contents

Introduction

A lot can go wrong when fitting spatially explicit capture–recapture (SECR) models. This vignette assembles a list of known difficulties with the R package **secr** 4.4, with examples, and suggests some solutions. It largely supercedes the Troubleshooting help page in **secr** 4.4. Potential problems with density surface models are also discussed in secr-densitysurfaces.pdf.

Speed issues are also addressed.

When `secr.fit` fails

Bias limit exceeded

Suppose we omitted to specify the buffer argument for the first snowshoe hare model in secr-tutorial.pdf:

```
fit <- secr.fit (hareCH6, trace = FALSE)
```

```
Warning messages:
1:  In secr.fit(hareCH6, trace = FALSE) : using default buffer width 100 m
2:  In bufferbiascheck(output, buffer, biasLimit) :
predicted relative bias exceeds 0.01 with buffer = 100
```

The second warning message is clearly a consequence of the first: relying on the 100-m default buffer for an animal as mobile as the snowshoe hare is likely to cause mask truncation error. This is easily fixed by following advice on choosing the buffer width in secr-tutorial.pdf or secr-habitatmasks.pdf.

A check for mask truncation bias[1] is performed routinely by `secr.fit` after fitting a model that relies on the 'buffer' argument. It may be avoided by setting `biasLimit = NA` or providing a pre-computed habitat mask in the 'mask' argument.

Initial log likelihood NA

Maximization will fail completely if the likelihood cannot be evaluated at the starting values. This will be obvious with `trace = TRUE`. Otherwise, the first indication will be a premature end to fitting and a lot of NAs in the estimates.

For an example, this section previously used the default starting values for the dataset `infraCH` (*Oligosoma infrapunctatum* skinks sampled with pitfall traps over two 3-occasion sessions labelled '6' and '7'). Unfortunately, those now seem to work, so we have to contrive an example by specifying a bad starting value for sigma:

```
fit <- secr.fit(infraCH, buffer = 25, start = list(sigma = 2), trace = TRUE)
```

```
## Checking data
## Preparing detection design matrices
## Preparing density design matrix
## Finding initial parameter values...
## Initial values  D = 271.84353, g0 = 0.12719, sigma = 2.87777
## Maximizing likelihood...
## Eval     Loglik        D       g0    sigma
##    1         NA   5.6052  -1.9260   0.6931
##    2         NA   5.6052  -1.9260   0.6931
##    3         NA   5.6052  -1.9260   0.6931
##    4         NA   5.6052  -1.9260   0.6931
##    5         NA   5.6052  -1.9260   0.6931
##    6         NA   5.6058  -1.9260   0.6931
##    7         NA   5.6052  -1.9259   0.6931
##    8         NA   5.6052  -1.9260   0.6932
##    9         NA   5.6063  -1.9260   0.6931
##   10         NA   5.6058  -1.9259   0.6931
##   11         NA   5.6058  -1.9260   0.6932
##   12         NA   5.6052  -1.9258   0.6931
##   13         NA   5.6052  -1.9259   0.6932
##   14         NA   5.6052  -1.9260   0.6933
```

[1] The positive bias that results when the buffer width for a detector array in continuous habitat is too small to encompass the centres of all animals potentially detected.

```
## Completed in 9.37 seconds at 18:13:37 19 May 2021
```

Unsurprisingly, the problem can be addressed by manually providing a better starting value for sigma:

```
fit1 <- secr.fit(infraCH, buffer = 25, start = list(sigma = 5), trace = FALSE)
```

The original problem seems to have been due to a discrepancy between the two sessions (try `RPSV(infraCH, CC = TRUE)`). The default sigma was suitable for the first session and not the second, whereas a larger sigma suits both. Rather than manually providing the starting value we could have directed `secr.fit` to use the second session for determining starting values:

```
fit2 <- secr.fit(infraCH, buffer = 25, details = list(autoini = 2), trace = FALSE)
```

Variance calculation failed

If warnings had not been suppressed in the preceding example we would have seen

```
Warning message:
In secr.fit(infraCH, buffer = 25, details = list(autoini = 2), trace = FALSE) :
at least one variance calculation failed
```

Examination of the output would reveal missing values for SE, lcl and ucl in both the coefficients and predicted values for g0.

Failure to compute the variance can be a sign that the model is inherently non-identifiable or that the particular dataset is inadequate (e.g., Gimenez et al. 2004). However, here it is due to a weakness in the default algorithm. Call `secr.fit` with `method = "none"` to recompute the variance-covariance matrix using a more sophisticated approach:

```
fit2r <- secr.fit(infraCH, buffer = 25, start = fit2, method = "none")
```

```
## Checking data
## Preparing detection design matrices
## Preparing density design matrix
## Computing Hessian with fdHess in nlme
##    1  -2260.348   5.5118  -2.2335   1.4912
##    2  -2260.354   5.5173  -2.2335   1.4912
##    3  -2260.348   5.5118  -2.2313   1.4912
##    4  -2260.350   5.5118  -2.2335   1.4927
##    5  -2260.355   5.5063  -2.2335   1.4912
##    6  -2260.349   5.5118  -2.2358   1.4912
##    7  -2260.350   5.5118  -2.2335   1.4897
##    8  -2260.357   5.5173  -2.2313   1.4912
##    9  -2260.360   5.5173  -2.2335   1.4927
##   10  -2260.352   5.5118  -2.2313   1.4927
##   11  -2260.348   5.5118  -2.2335   1.4912
## Completed in 9.36 seconds at 17:38:05 19 May 2021
```

```
predict(fit2r)
```

```
## $`session = 6`
##        link     estimate SE.estimate          lcl         ucl
## D       log 247.59173982 13.96700058 221.69546160 276.5129660
## g0    logit   0.09677928  0.00840723   0.08150741   0.1145557
## sigma   log   4.44253548  0.16206272   4.13608621   4.7716901
##
## $`session = 7`
##        link     estimate SE.estimate          lcl         ucl
```

```
## D       log 247.59173982 13.96700058 221.69546160 276.5129660
## g0    logit   0.09677928  0.00840723   0.08150741   0.1145557
## sigma   log   4.44253548  0.16206272   4.13608621   4.7716901
```

The trapping sessions were only 4 weeks apart in spring 1995. We can further investigate session differences by fitting a session-specific model. The fastest way to fit fully session-specific models is to fit each session separately; `lapply` here applies `secr.fit` separately to each component of `infraCH`:

```
fits3 <- lapply(infraCH, secr.fit, buffer = 25, trace = FALSE)
```

```
## Warning in FUN(X[[i]], ...): possible maximization error: nlm returned code 3.
## See ?nlm
```

```
class(fits3) <- "secrlist"  # ensure secr will recognise the fitted models
predict(fits3)
```

```
## $`6`
##        link    estimate SE.estimate         lcl         ucl
## D       log 255.7577859 28.39818527 205.8755083 317.7262102
## g0    logit   0.1712718  0.03003002   0.1201179   0.2383099
## sigma   log   2.5190284  0.18168045   2.1873650   2.9009808
##
## $`7`
##        link     estimate  SE.estimate          lcl         ucl
## D       log 278.02271367 18.662192836 243.78547641 317.0682292
## g0    logit   0.09894276  0.009247182   0.08223807   0.1186021
## sigma   log   4.95110168  0.205389892   4.56463499   5.3702887
```

Notice that there is no issue with starting values when the sessions are treated separately. The skinks appeared to enlarge their home ranges as the weather warmed; they may also have become more active overall[2]. It is plausible that density did not change: the estimate increased slightly, but there is substantial overlap of confidence intervals.

Log likelihood becomes NA or improbably large after a few evaluations

The default maximization method (Newton-Raphson in function `nlm`) takes a large step away from the initial values at evaluation `np + 3` where `np` is the number of estimated coefficients. This often results in a very negative or NA log likelihood, from which the algorithm quickly returns to a reasonable part of the parameter space. However, for some problems it does not return and estimation fails[3]. Two solutions are suggested:

- change to the more robust Nelder-Mead maximization algorithm

```
secr.fit(CH, method = "Nelder-Mead", ...)
```

- vary the scaling of each parameter in `nlm` by passing the `typsize` (typical size) argument. The default is `typsize = rep(1, np)`. Suppose your model has four coefficients and it is the second one that appears to be behaving wildly:

```
secr.fit(CH, typsize = c(1, 0.1, 1, 1), ...)
```

In these examples `CH` is your capthist object and `...` indicates other arguments of `secr.fit`.

Possible maximization error: nlm code 3

The default algorithm for numerical maximization of the likelihood is `nlm` in the **stats** package. That uses a Newton-Raphson algorithm and numerical derivatives. It was chosen because it is significantly faster than

[2]Home-range area increased about 4-fold; g0 showed some compensatory decrease, but compensation was incomplete, implying increased total activity (treating g_0 as an estimate of λ_0; see Efford and Mowat 2014).

[3]You may also get weird messages about infinite densities; these messages are due to be removed in the next release.

the alternatives. However, it sometimes returns estimates with the ambiguous result code 3, which means that the optimization process terminated because "[the] last global step failed to locate a point lower than estimate. Either estimate is an approximate local minimum of the function or steptol is too small."

Here is an example:

```
fit3 <- secr.fit(infraCH, buffer = 25, model = list(g0~session, sigma~session),
                 details = list(autoini = 2), trace = FALSE)
```

```
Warning message:
In secr.fit(infraCH, buffer = 25, model = list(g0   session, sigma   :
possible maximization error:  nlm returned code 3.  See ?nlm
```

The results seem usually to be reliable even when this warning is given. If you are nervous, you can try a different algorithm in `secr.fit` – "Nelder-Mead" is recommended. We can derive starting values from the earlier fit:

```
fit3nm <- secr.fit(infraCH, buffer = 25, model = list(g0~session, sigma~session),
                   method = "Nelder-Mead", start = fit3, trace = FALSE)
```

There is no warning. Comparing the density estimates we see a trivial difference in the SE and confidence limits and none at all in the estimates themselves:

```
collate(fit3 = fit3, fit3nm = fit3nm)[1,,,'D']
```

```
##         estimate SE.estimate      lcl      ucl
## fit3   271.9508    15.78539 242.7302 304.6890
## fit3nm 271.9508    15.80973 242.6878 304.7423
```

This suggests that only the variance-covariance estimates were in doubt, and it would have been much quicker merely to check them with `method = "none"` as in the previous section.

Possible maximization error: nlm code 4

The `nlm` Newton-Raphson algorithm may also finish with the result code 4, which means that the optimization process terminated when the maximum number of iterations was reached ("iteration limit exceeded"). The maximum is set by the argument `iterlim` which defaults to 100 (each 'iteration' uses several likelihood evaluations to numerically determine the gradient for the Newton-Raphson algorithm).

The number of iterations can be checked retrospectively by examining the nlm output saved in the 'fit' component of the fitted model. Ordinarily nlm uses less than 50 iterations (for example `fit3$fit$iterations` = 27).

A 'brute force' solution is to increase `iterlim` (`secr.fit()` passes `iterlim` directly to `nlm()`) or to re-fit the model starting at the previous solution (`start = oldfit`). This is not guaranteed to work. Alternative algorithms such as `method = 'Nelder-Mead'` are worth trying, but they may struggle also.

There does not appear to be a universal solution. Slow or poor fitting seems more common when there are many beta parameters, and when one or more parameters is very imprecise, at a boundary, or simply unidentifiable. It is suggested that you examine the coefficients of the provisional result with `coef(fit)` and seek to eliminate those that are not identifiable.

Tricks include:

- combining levels of poorly supported factor covariates
- fixing the value of non-identifiable beta parameters with details argument `fixedbeta`
- ensuring that all levels of a factor x factor interaction are represented in the data (possibly by defining a single factor with valid levels)
- changing the coding of factor covariates with details argument `contrasts`.

The following code demonstrates fixing a beta parameter, although it is neither needed nor recommended in this case.

```
# review the fitted values
coef(fit3)
```

```
##                       beta    SE.beta        lcl        ucl
## D                5.6056212 0.05799623  5.4919507  5.7192917
## g0              -1.6268200 0.19768514 -2.0142757 -1.2393642
## g0.session7     -0.5757405 0.21736905 -1.0017760 -0.1497049
## sigma            0.9197120 0.07151489  0.7795454  1.0598786
## sigma.session7   0.6821825 0.08116608  0.5231000  0.8412651
```

```
# extract the coefficients
betafix <- coef(fit3)$beta
# set first 4 values to NA as we want to estimate these
betafix[1:4] <- NA
betafix
```

```
## [1]        NA        NA        NA        NA 0.6821825
```

```
# refit, holding last coefficient constant
fit3a <- secr.fit(infraCH, buffer = 25, model = list(g0~session, sigma~session),
         details = list(autoini = 2, fixedbeta = betafix), trace = FALSE)
coef(fit3a)
```

```
##                    beta    SE.beta        lcl        ucl
## D             5.6055911 0.05836630  5.4911953  5.7199870
## g0           -1.6268602 0.14119448 -1.9035963 -1.3501241
## g0.session7 -0.5756668 0.13135596 -0.8331198 -0.3182139
## sigma         0.9197314 0.03682716  0.8475515  0.9919113
```

Note that the estimated coefficients ('beta') have not changed, but the estimated 'SE.beta' of each detection parameter has dropped - a result of our spurious claim to know the true value of 'sigma.session7'.

There is no direct mechanism for holding the beta parameters for different levels of a factor (e.g., `session`) at a single value. The effect can be achieved by defining a new factor covariate with combined levels.

`secr.fit` requests more memory than is available

In **secr** 4.4 the memory required by the external C code is at least $32 \times C \times M \times K$ bytes, where C is the number of distinct sets of values for the detection parameters (across all individuals, occasions, detectors and finite mixture classes), M is the number of points in the habitat mask and K is the number of detectors. Each distinct set of values appears as a row in a lookup table[4] whose columns correspond to real parameters; a separate parameter index array (PIA) has entries that point to rows in the lookup table. Four arrays with dimension $C \times M \times K$ are pre-filled with, for example, the double-precision (8-byte) probability an animal in mask cell m is caught in detector k under parameter values c.

The number of distinct parameter sets C can become large when any real parameter (g0, lambda0, sigma) is modelled as a function of continuous covariates, because each unit (individual, detector, occasion) potentially has a unique level of the parameter. A rough calculation may be made of the maximum size of C for a given amount of available RAM. Given say 6GB of available RAM, $K = 200$ traps, and $M = 4000$ mask cells, C should be substantially less than 6e9 / 200 / 4000 / 32 $\approx$ 234. Allowance must be made for other memory allocations; this is simply the largest.

There is a different lookup table for each session; the limiting C is for the most complex session. The memory constraint concerns detection parameters only.

[4] If you are really keen you can see this table by running `secr.fit` with `details = list(debug = 3)` and typing `Xrealparval` at the browser prompt (type Q to exit).

Most analyses can be re-configured to bring the memory request down to a reasonable number.

1. C may be reduced by replacing each continuous covariate with one using a small number of discrete levels (e.g. the mid-points of weight classes). For example, `weightclass <- 10 * trunc(weight/10) + 5` for midpoints of 10-g classes.
2. M can be reduced by building a habitat mask with an appropriate spacing (see secr-habitatmasks.pdf).
3. K might seem to be fixed by the design, but in extreme cases it may be appropriate to combine data from adjacent detectors (see Collapsing detectors).

The `mash` function (see Mashing) may be used to reduce the number of detectors when the design uses many identical and independent clusters. Otherwise, apply your ingenuity to simplify your model, e.g., by casting 'groups' as 'sessions'. Memory is less often an issue on 64-bit systems (see also `?"Memory-limits"`).

Covariate factor levels differ between sessions

This is fairly explicit; `secr.fit` will stop if you include in a model any covariate whose factor levels vary among sessions, and `verify` will warn if it finds any covariate like this. This commonly occurs in multi-session datasets with 'sex' as an individual covariate when only males or only females are detected in one session. Use the function `shareFactorLevels` (new in secr 3.0) to force covariates to use the same superset of levels in all sessions.

Estimates from finite mixture models appear unstable

These models have known problems due to multimodality of the likelihood. See secr-finitemixtures.pdf.

Faster is better

There is nothing virtuous about waiting days for a model to fit if there is a faster alternative. Here are some things you can do.

Fast proximity models

In **secr** 4.4 there are several tricks that make fitting of many models much faster. These tricks are implemented by default in `secr.fit`, but may be turned off by setting the details argument 'fastproximity = FALSE'. See secr-version4.pdf for more.

Mask tuning

Consider carefully the necessary extent of your habitat mask and the acceptable cell size (secr-habitatmasks.pdf has detailed advice). If your detectors are clustered then your mask may have gaps between the clusters. Masks with more than 2000 points are generally excessive (and the default is about 4000!).

Individual mask subsets

secr 4.4 allows the user to customise the mask for each detected animal by considering only a subset of points. The subset is defined by a radius in metres around the centroid of detections; set this using the details argument 'maxdistance'. Speed gains vary with the layout, but can exceed 2-fold. Bias results when the radius is too small (try 5σ).

Conditional likelihood

The default in `secr.fit` is to maximize the full likelihood (i.e., to jointly fit both the state process and the observation process). If you do not need to model spatial, temporal or group-specific variation in density (the

sole real parameter of the state model) then you can save time by first fitting only the observation process[5]. This is achieved by maximizing only the likelihood conditional on n, the number of detected individuals (Borchers and Efford 2008). Conditional likelihood maximization is selected in `secr.fit` by setting `CL = TRUE`.

```
fit <- secr.fit(hareCH6, buffer = 250, trace = FALSE)     # default CL = FALSE
fitCL <- secr.fit(hareCH6, buffer = 250, CL = TRUE, trace = FALSE)
```

Fitting time is reduced by 45% because maximization is over two parameters (g0, sigma) instead of three. The relative reduction will be less for more complex detection models, but still worthwhile.

Having selected a suitable observation model with `CL = TRUE` you can then either resort to a full-likelihood fit to estimate density, or compute a Horvitz-Thompson-like (HT) estimate in function `derived`. In models without individual covariates the HT estimate is $n/a(\hat{\theta})$ where n is the number of detected individuals, θ represents the parameters of the observation model, and a is the effective sampling area as a function of the estimated detection parameters.

Compare

```
predict(fit)      # CL = FALSE
```

```
##         link    estimate SE.estimate         lcl         ucl
## D        log  1.46598705 0.191312466  1.13636093  1.89122837
## g0     logit  0.06158395 0.009300448  0.04568551  0.08253649
## sigma    log 68.34577713 4.469311417 60.13242671 77.68097027
```

```
derived(fitCL)    # CL = TRUE
```

```
##      estimate SE.estimate      lcl      ucl       CVn        CVa       CVD
## esa 46.385111          NA       NA       NA        NA         NA        NA
## D    1.465988   0.1905122 1.137563 1.889232 0.1212678 0.04671589 0.1299548
```

Estimated density is exactly the same, to 6 significant figures (1.46599). This is expected when n is Poisson; slight differences arise when n is binomial (because the number of animals N in the masked area is considered fixed rather than random). The estimated variance differs slightly - that from `derived` follows an unpublished and slightly *ad hoc* procedure (Borchers and Efford 2007).

Parallel fitting of multiple models

Your computer almost certainly has multiple cores, allowing computations to be run in parallel. The function `par.secr.fit` is no longer the preferred way to use multiple cores. Multi-threading in `secr.fit` uses multiple cores by default (i.e. unless you specify `ncores = 1`). The default number of threads (cores) is one less than the maximum available. See `?Parallel` for more.

Mashing

Mashing is a very effective way of speeding up estimation when the design uses many replicate clusters of detectors, each with the same geometry, and far enough apart that animals are not detected on more than one. The approach for M clusters is to overlay all data as if from a single cluster; the estimated density will be M times the per-cluster estimate, and SE etc. will be inflated by the same factor. This relies on individuals being detected independently of each other, which is a standard assumption in any case. The present implementation assumes density is uniform.

We describe in general terms an actual example in which 18 separated clusters of 12 traps were operated on 6 occasions. Each cluster had the same geometry (two parallel rows of traps separated by 200 m along

[5]Selecting an observation model with `CL = TRUE` (and first focussing on detection parameters) is a good strategy even if you intend to model density later. It may occasionally be desirable to re-visit the selection if covariates can affect both density and detection parameters.

and between rows). Trap numbering was consistently up one row and down the other. The capthist object CH included detections of 150 animals in the 216 traps. To mash these data we first assign attributes for the cluster number (clusterID) and the sequence number of each trap within its cluster (clustertrap). The function `mash` then collapses the data as if all detections were made on one cluster. A mask based on this single notional cluster has many fewer points than a mask with the same spacing spanning all clusters.

```
clustertrap(traps(CH)) <- rep(1:12,18)
clusterID(traps(CH)) <- rep(1:18, each = 12)
mashedCH <- mash(CH)
mashedmask <- make.mask(traps(mashedCH), buffer = 900, spacing = 100, type = "trapbuffer")
fitmash <- secr.fit(mashedCH, mask = mashedmask)
```

The model for the mashed data fitted in 4% of the time required for the original. The mashed estimate of density shrank by 2% in this case, which is probably due to slight variation among clusters in the actual spacing of traps (one cluster was arbitrarily chosen to represent all clusters). Mashing prevents the inclusion of cluster-specific detail in the model (such as discontinuous habitat near the traps). For further details see `?mash`.

Reducing complexity (session- or group-specific models)

Simultaneous estimation of many parameters can be painfully slow, but it can be completely avoided. If your model is fully session- or group-specific then it is much faster to analyse each group separately. For sessions this is can be done simply with `lapply` above and in secr-multisession.pdf). For groups you may need to construct new capthist objects using `subset` to extract groups corresponding to the levels of one or more individual covariates.

Collapsing occasions

If there is no temporal aspect to the model you want to fit (such as a behavioural response) then it is attractive to collapse all sampling occasions to one with `reduce`. This can usually be done without loss of data by choosing 'outputdetector' carefully. For example,

```
FTHL.fit <- secr.fit (hornedlizardCH, buffer = 80, trace = FALSE)
```

```
## Warning in secr.fit(hornedlizardCH, buffer = 80, trace = FALSE): using default
## starting values
```

```
collapsedCH <- reduce(hornedlizardCH, by = "ALL", outputdetector = "polygon")
usage(traps(collapsedCH))
```

```
##      [,1]
## [1,]   14
```

```
FTHL.fit14 <- secr.fit (collapsedCH, buffer = 80, trace = FALSE, binomN = 1)
```

```
## Warning in secr.fit(collapsedCH, buffer = 80, trace = FALSE, binomN = 1): using
## default starting values
```

The collapsed capthist object receives a usage attribute equal to the number of collapsed occasions (14). Unfortunately, in this instance the collapsed model takes longer to fit, on account of having to compute a Binomial probability with size = 14 rather than a Bernoulli probability. A Poisson model (not shown) is slightly faster than the original, but the estimates differ by about 5%.

```
collate(FTHL.fit = FTHL.fit, FTHL.fit14 = FTHL.fit14)[1,,,'D']
```

```
##             estimate SE.estimate    lcl      ucl
## FTHL.fit    8.013068    1.061701 6.1874 10.37742
## FTHL.fit14        NA          NA     NA       NA
```

```
FTHL.fit$proctime
```

```
## elapsed
##   23.79
```

```
FTHL.fit14$proctime
```

```
## elapsed
##    7.86
```

Collapsing detectors

The theoretical (but often unrealised) benefit from collapsing occasions has a spatial analogue: if there are many detectors and they are closely spaced relative to animal movements σ then nearby detectors may be aggregated into new notional detectors located at the centroid. The reduce.traps method has an argument 'span' explained as follows in the help –

> If `span` is specified a clustering of detector sites will be performed with `hclust` and detectors will be assigned to groups with `cutree`. The default algorithm in `hclust` is complete linkage, which tends to yield compact, circular clusters; each will have diameter less than or equal to `span`.

"multi" detector type instead of "proximity"

The type of the detectors is usually determined by the sampling reality. For example, if individuals can physically be detected at several sites on one occasion then the "proximity" detector type is preferred over "multi". However, if data are very sparse, so that individuals in practice are almost never observed at multiple sites on one occasion, then the detectors may as well by of type "multi", in the sense that there is no observable difference between the two types of detection process. "multi" models used to fit much more quickly than "proximity" models, and this is still true for elaborate or time-dependent models that cannot use the 'fastproximity' option.

Some models are just slower than others

Detector covariates are a particular problem. Models with learned responses take slightly longer to fit.

References

Borchers, D. L. and Efford, M. G. (2007) Supplements to Biometrics paper. Available online at https://www.otago.ac.nz/density.

Borchers, D. L. and Efford, M. G. (2008) Spatially explicit maximum likelihood methods for capture–recapture studies. *Biometrics* **64**, 377–385.

Gimenez, O., Viallefont, A., Catchpole, E. A., Choquet, R. and Morgan, B. J. T. (2004) Methods for investigating parameter redundancy. *Animal Biodiversity and Conservation* **27**, 561–572.

Otis, D. L., Burnham, K. P., White, G. C. and Anderson, D. R. (1978) Statistical inference from capture data on closed animal populations. *Wildlife Monographs* No. **62**.

Royle, J. A. and Young, K. V. (2008) A hierarchical model for spatial capture–recapture data. *Ecology* **89**, 2281–2289.