Modelling density surfaces in secr

Murray Efford

November 16, 2011

The formulation of spatially explicit capture–recapture (SECR) by Borchers and Efford (2008) allows for population density to vary over space. Fitting models that include spatial covariates (e.g., habitat class), or spatial trend, has been possible in **secr** from the start, but use of this feature is now made easier by the addition of functions for prediction and plotting. This document provides an overview and worked examples.

Contents

Density surfaces - some background

SECR fits a state model and an observation model. The observation model is a detection function; this is described in secr-overview.pdf[1] and we will not consider it further here. The state model is a spatial Poisson process for animal range centres that may be homogeneous (constant over space) or inhomogeneous (varying over space). The notation $D(\mathbf{x}; \phi)$ covers both possibilities: density (D) is a function of location (the vector $\mathbf{x}$ representing a pair of x- and y-coordinates). If density is homogeneous (i.e. flat expected value) then the parameter ϕ is one number, the density. In other cases, ϕ is a vector of several parameters.

To model variation in a density surface we need to maximise the full likelihood. Maximising the conditional likelihood (conditional on n, the number of observed individuals) is a way to estimate the observation model; to go from there to a Horvitz-Thompson estimate of density we assume that density is homogeneous.

Although $D(\mathbf{x}; \phi)$ may be a smooth function, in **secr** we evaluate it only at the fixed points of a habitat 'mask' (you can think of these as the cell centres of a pixellated or raster representation). A mask defines the region of habitat relevant to a particular study: most simply it is a buffered zone around the detector locations, but it may exclude interior areas of non-habitat.

A density model $D(\mathbf{x}; \phi)$ is specified either in the 'model' argument of `secr.fit` or in a user-written function supplied to `secr.fit` (examples later). Spatial covariates, if any, are needed for each mask point; they are stored in the 'covariates' attribute of the mask. Results from fitting the model (including the estimated ϕ) are saved in an object of class 'secr'. To visualise a fitted density model we first evaluate it at each point on a mask with the function predictDsurface to create an object of class 'Dsurface'. A Dsurface is a mask with added density data, and plotting a Dsurface is like plotting a mask covariate.

Using the 'model' argument in secr.fit

The model argument of secr.fit is a list of formulae, one for each 'real' parameter[2] in both the state model (just 'D' for density) and the observation model (typically 'g0' and 'sigma'). A model formula defines variation in each parameter as a function of covariates (broad sense) that is linear on the 'link' scale, as in a generalized linear model.

The options differ between the state and observation models. D may vary

[1] see the 'directory' link in the 'secr' package help index

[2] null formulae e.g. `D ~ 1` may be omitted, and if a single formula is used, it may be presented on its own rather than in list() form

with respect to group, session or point in space; g0 and sigma may vary by group, session, occasion or latent class (finite mixture), but not with respect to continuous space. This is partly a design choice, to tame the complexity that would result if g0 and sigma varied continuously.

Link function

The default link for D is 'log'. It is equally feasible in most cases to choose 'identity' as the link (see the `secr.fit` argument of that name), and for the null model D ~ 1 the estimate will be nearly the same, as will estimates involving only categorical variables (e.g., session). However, with an 'identity' link the routine (asymptotic) confidence limits will be symmetrical (unless truncated at zero) rather than asymmetrical. In all models with continuous predictors or trend surfaces the link function will affect the result, although the difference may be small when the amplitude of variation on the surface is small. Otherwise, serious thought is needed regarding which model is biologically more appropriate: logarithmic or linear.

You may wonder why `secr.fit` is ambivalent: link functions have seemed a necessary part of the machinery for capture–recapture modelling since Lebreton et al. (1992). Their key role is to keep the 'real' parameter within feasible bounds (e.g., 0-1 for probabilities). In `secr.fit` any modelled value of D that falls below zero is truncated at zero (of course this condition will not arise with a log link).

Built-in variables

`secr.fit` automatically recognises the spatial variables x,y,x2,y2 and xy if they appear in the formula for D. These refer to the x-coordinate, y-coordinate, x-coordinate^2 etc. for each mask point, and will be constructed automatically as needed.

The formula for D may also include the non-spatial variables g (group), session (categorical), and Session (continuous), defined as for modelling g0 and sigma (see Overview).

The built-in variables offer limited model possibilities:

D ~ 1	flat surface (default)
D ~ x + y	linear trend surface (planar)
D ~ x + x2	quadratic trend in east-west direction only
D ~ x + y + x2 + y2 + xy	quadratic trend surface
etc.	

User-provided variables

More interesting models can be made with variables provided by the user. These are stored in a data frame as the 'covariates' attribute of a mask object. Covariates must be defined for every point on a mask.

Variables may be categorical (a factor or character value that can be coerced to a factor) or continuous (a numeric vector). The habitat variable 'habclass' constructed in the Examples section of the `skink` help is an example of a two-class categorical covariate. Remember that categorical variables entail one additional parameter for each extra level.

There are several ways to create or input mask covariates.

- read columns of covariates along with the x- and y-coordinates when creating a mask from a dataframe or external file (`read.mask`)
- read the covariates dataframe separately from an external file (`read.table`)
- infer values for points on an existing mask from the nearest points in another dataset, such as detector covariates (e.g., 'habclass' in `skink` Examples)
- infer values for points on an existing mask from GIS data, such as a polygon shapefile or SpatialPolygonsDataFrame (see 'sp' (Pebesma and Bivand 2005), 'maptools' (Lewin-Koh, Bivand et al. 2011), and related R packages)
- compute from coordinates (e.g., distance to shore in possum example below)

Use the function `addCovariates` for the third and fourth options.

User-provided model functions

Some density models cannot be coded in the generalized linear model form of the 'model' argument. To alleviate this problem, a model may be specified as an R function that is passed to `secr.fit`, specifically as the component 'userDfn' of the list argument 'details'. We document this feature here, although you may never use it.

The userDfn function must follow some rules.

- It should accept four arguments, the first a vector of parameter values or a character value (below), and the second a 'mask' object, a data frame of x and y coordinates for points at which density must be predicted.

`Dbeta`	coefficients of density model, or one of c("name", "parameters")
`mask`	secr habitat mask object
`ngroup`	number of groups
`nsession`	number of sessions

- When called with `Dbeta = 'name'`, the function should return a character string to identify the density model in output. (This should not depend on the values of other arguments).
- When called with `Dbeta = 'parameters'`, the function should return a character vector naming each parameter. (When used this way, the call always includes the `mask` argument, so information regarding the model may be retrieved from any attributes of `mask` that have been set by the user).
- Otherwise, the function should return a numeric array with `dim = c(nmask, ngroup, nsession)` where `nmask` is the number of points (rows in mask). Each element in the array is the predicted density (natural scale, in animals / hectare) for each point, group and session. This is simpler than it sounds, as usually there will be a single session and single group.

The coefficients form the density part of the full vector of 'beta' coefficients used by the likelihood maximization function (`nlm` or `optim`). Ideally, the first one should correspond to an intercept or overall density, as this is appears in the output of predict.secr. If transformation of density to the 'link' scale is required it should be hard-coded in userDfn.

Covariates are available to user-provided functions, but they must be extracted 'manually' (e.g., 'covariates(mask)$habclass' rather than just 'habclass'). To pass other arguments (e.g., a basis for splines), add attribute(s) to the mask.

It will usually be necessary to specify starting values for optimisation manually with the 'start' argument of `secr.fit`.

If the parameter values in `Dbeta` are invalid the function should return an array of all zero values.

Here is a 'null' userDfn that emulates D $\sim$ 1 with log link

```
> userDfn0 <- function (Dbeta, mask, ngroup, nsession) {
    if (Dbeta[1] == 'name') return ('0')
    if (Dbeta[1] == 'parameters') return ('intercept')
    D <- exp(Dbeta[1])   ## constant for all points
    tempD <- array(D, dim = c(nrow(mask), ngroup, nsession))
    return(tempD)
 }
```

We can compare the result using userDfn0 to a fit of the same model using the 'model' argument. We drop two columns of the AIC table to save space. Note how the model description combines 'user.' and the name '0'.

```
> library(secr)
> model.0 <- secr.fit(captdata, model = D ~ 1, trace = FALSE)
> userDfn.0 <- secr.fit(captdata, details = list(userDfn = userDfn0),
    trace = FALSE)
> AIC(model.0, userDfn.0)[,-c(2,5)]

                          model npar    logLik     AICc dAICc AICwt
model.0         D~1 g0~1 sigma~1    3 -759.0198 1524.373     0   0.5
userDfn.0 D~userD.0 g0~1 sigma~1    3 -759.0198 1524.373     0   0.5

> predict(model.0)

       link   estimate SE.estimate        lcl        ucl
D       log  5.4788238  0.64671181  4.3507106  6.8994500
g0    logit  0.2731604  0.02705172  0.2234466  0.3292453
sigma   log 29.3695713  1.30602252 26.9193515 32.0428119

> predict(userDfn.0)

       link   estimate SE.estimate        lcl        ucl
D       log  5.4788224  0.64671168  4.3507094  6.8994483
g0    logit  0.2731605  0.02705177  0.2234466  0.3292456
sigma   log 29.3695700  1.30602306 26.9193492 32.0428118

> coef(userDfn.0)

                beta    SE.beta       lcl        ucl
D.intercept  1.700890 0.11763036  1.470339  1.9314415
g0          -0.978646 0.13625084 -1.245693 -0.7115992
sigma        3.379959 0.04444662  3.292845  3.4670729
```

Not very exciting, maybe, but reassuring!

Prediction and plotting

We return to the main thread, predicting and plotting a fitted density surface.

Fitting a model provides estimates of its coefficients. In order to plot a fitted model we first predict the height of the density surface at each point on a mask. This is done with `predictDsurface`, which has arguments `(object, mask = NULL, se.D = FALSE, cl.D = FALSE, alpha = 0.05)`. By default, prediction is at the mask points used when fitting the model (i.e. object$mask); specify the 'mask' argument to extrapolate the model to a different area. The output from `predictDsurface` is a specialised mask object called a Dsurface (class `c('Dsurface', 'mask', 'data.frame')`).

Use the arguments `se.D` and `cl.D` to request computation of the estimated standard error and/or upper and lower confidence limits for each mask point[3]. If requested, values are saved as additional covariates of the output Dsurface (SE.1, lcl.1, ucl.1 for one group).

A Dsurface is a mask whose covariate dataframe has additional column(s) for the predicted density of each group. Usually when you print a mask you see only the x- and y-coordinates. The `print` method for Dsurface objects displays both the coordinates and the density values as one dataframe, as also do the `head` and `tail` methods.

The plot method for a Dsurface object has arguments `(x, covariate = 'D', group = 1, plottype = 'shaded', ...)`. Note that `covariate` may either be a prefix (one of 'D', 'SE', 'lcl', 'ucl') or any full covariate name. `plottype` may be one of 'shaded', 'dots', 'persp', or 'contour'. For details on how to specify colours, and many other options, read the help pages for `plot.mask`, `contour` and `persp`.

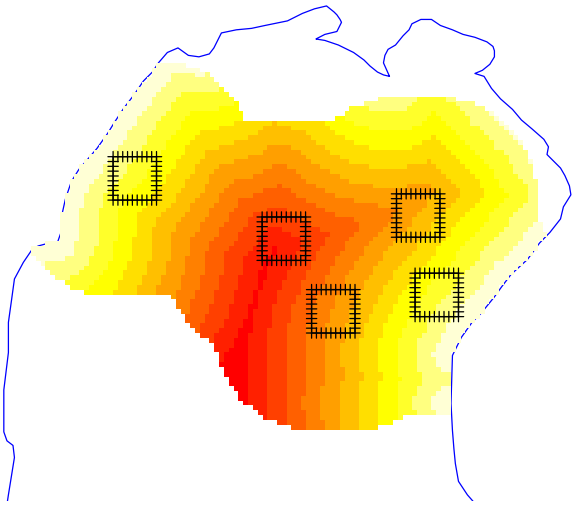
Applying this to the pre-fitted model `possum.model.Dsh2` we get

```
> shorePossums <- predictDsurface(possum.model.Dsh2, cl.D = TRUE)
> head(shorePossums)

         x       y      D.0    lcl.0    ucl.0
58 2698543 6077080 1.534592 1.136332 1.932852
59 2698563 6077080 1.558118 1.161337 1.954900
60 2698583 6077080 1.581645 1.185896 1.977393
61 2698603 6077080 1.605170 1.210006 2.000335
62 2698623 6077080 1.628696 1.233665 2.023727
63 2698643 6077080 1.652221 1.256872 2.047569

> plot(shorePossums, plottype = 'shaded', polycol = 'blue', border = 100)
> plot(traps(possumCH), detpar = list(col = 'black'), add = TRUE)
```

[3]option available only for models specified in generalized linear model form with the 'model' argument of secr.fit, not for user-defined functions



A more extensive treatment of this example follows.

Brushtail possum example

Brushtail possums (*Trichosurus vulpecula*) were live-trapped by Efford et al. (2005) in pine (*Pinus radiata*) forest and scrub on a ∼300-ha coastal peninsula in New Zealand. The site was surrounded by water on three sides, leaving only one 'open' boundary. Cage traps were set in groups of 36 at 20-m spacing around the perimeter of five squares, each 180 m on a side. The squares ('hollow grids') were centered at random points. Animals were trapped, tagged and released daily for 5 days. Subsequently, strenuous efforts were made to remove all possums by cyanide poisoning and leghold trapping across the entire area. The capture data are provided as 'possumCH' in **secr**, along with a dataframe 'possumarea' containing the coordinates of a boundary that follows the shoreline in the west, north and east. See ?possum in **secr** for more details.

There is evidence for non-spatial heterogeneity in capture probabilities (unpubl. results) so we fit a finite-mixture observation model that allows for individual differences (g0 ∼ h2, sigma ∼ h2). We use the default detection function (halfnormal). For model fitting the mask needs to include all habitat near the traps: we use a 400-m buffer around the traps, clipped to the shoreline (this is larger than the 300-m buffer used by Efford et al. (2005) to allow for one class

in the finite mixture model having large sigma).

```
> possummask <- make.mask (traps(possumCH), buffer = 400, spacing = 20,
     poly = possumarea, type = 'trapbuffer')
```

Results in Table 2 of Efford et al. (2005) suggest some variation between the hollow grids (fewer possums were caught on the central grids, 1 and 2) so we are interested to consider models of the density surface. Ideally sampling would have been conducted at more sites with this in mind, but we'll do what we can with the data as they stand.

Using 'possummask' we can fit both a flat density surface (D $\sim$ 1) and a quadratic surface, suppressing output of each iteration during likelihood maximization. This takes a few minutes (11 min in R 2.14.0 on my machine).

```
> fit1 <- secr.fit (possumCH, model = list(D ~ 1, g0 ~ h2,
     sigma ~ h2), mask = possummask, trace = FALSE)
> fit2 <- secr.fit (possumCH, model = list(D ~ x + y + x2 + y2 + xy,
     g0 ~ h2, sigma ~ h2), mask = possummask, trace = FALSE)
```

Did we gain anything by fitting the density surface?

```
> AIC(fit1, fit2)
```

```
                                          model   detectfn npar
fit1                    D~1 g0~h2 sigma~h2 pmix~h2 halfnormal    6
fit2 D~x + y + x2 + y2 + xy g0~h2 sigma~h2 pmix~h2 halfnormal   11
        logLik      AIC     AICc dAICc  AICwt
fit1 -1081.268 2174.537 2175.337 0.000 0.9557
fit2 -1078.419 2178.839 2181.479 6.142 0.0443
```

Probably not. Nevertheless, it is interesting to look at the quadratic model surface. First, review the fitted model

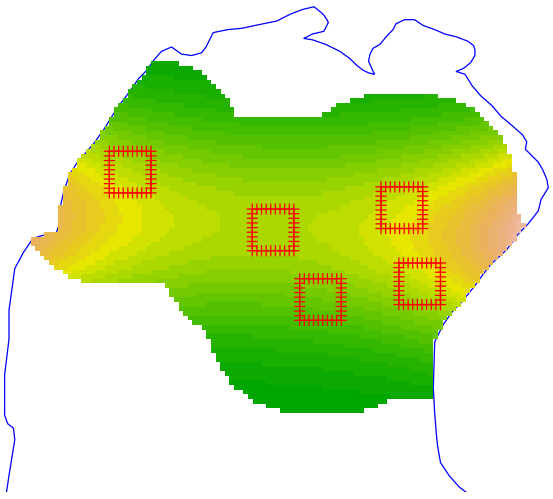
```
> predict(fit2)
```

```
$`session = WaitarerePossums, h2 = 1, x = 0, y = 0, x2 = 0, y2 = 0, xy = 0`
       link   estimate SE.estimate       lcl        ucl
D       log  1.8749799  0.37352554  1.273712  2.7600830
g0    logit  0.3554018  0.06868268  0.234516  0.4980571
sigma   log 36.0251319  3.04893140 30.527691 42.5125546
pmix  logit  0.7451412          NA        NA         NA
```

```
$`session = WaitarerePossums, h2 = 2, x = 0, y = 0, x2 = 0, y2 = 0, xy = 0`
       link    estimate SE.estimate         lcl         ucl
D       log  1.87497987   0.3735255  1.27371151   2.7600830
g0    logit  0.07062142   0.0196774  0.04051248   0.1203017
sigma   log 87.85032404   7.4350765 74.44435033 103.6704518
pmix  logit  0.25485885          NA          NA          NA
```

Note how by default `predict.secr` evaluates the surface at (x = 0, y = 0). This is not the origin of the old New Zealand metric grid somewhere in the sub-antarctic, but the centroid of the mask points (remembering that coordinates were scaled before the model was fitted). Now plot the surface; `predictDsurface` uses the mask from the fitted object if no other is provided. The `plot.Dsurface` argument `plottype = 'shaded'` plots pixels instead of points, and `meshcol = NA` suppresses lines between pixels.

```
> surface2 <- predictDsurface(fit2)
> plot(surface2, border = 150, polyc = 'blue', plottype = 'shaded',
     col = terrain.colors(30), breaks = 25, meshcol = NA)
> plot(traps(possumCH), add = TRUE)
```



Two more tricks add value to this plot. Firstly, we can extract and (by

default) display the height of the fitted surface by clicking on chosen points. (If you try this, click within each hollow grid and then exit by clicking outside the map).

```
> spotHeight(surface2, dec = 1)
```

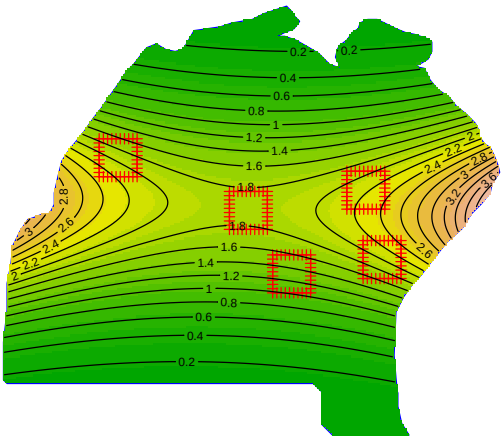
It can also help to add contours as a convenient alternative to a colour legend. This is a challenge, because the 'contour' function in R requires a rectangular matrix of values, and our mask is not rectangular. We could make it so with the secr function `rectangularmask`, which makes a rectangular Dsurface with missing (NA) values of density at all the external points. `plot.Dsurface` recognises an irregular mask and attempts to fix this with an internal call to `rectangularMask`:

```
> plot(surface2, border=150, polyc='blue', plottype = 'shaded',
     col = terrain.colors(30), breaks = 25, mesh = NA)
> plot(traps(possumCH), add = TRUE)
> plot(surface2, plottype = 'contour',levels = seq(0.2,4,0.2), add = TRUE)
```

Contours are labelled in animals / hectare. You can control the contour levels and labelling with the usual arguments of `contour`.

What if we extrapolate the fitted surface to the entire peninsula? For this we create a new mask ('regionmask') using the boundary polygon in the data object 'possumremovalarea'. The density model is evaluated at each point in 'regionmask'. We plot this and add contours as before.

```
> regionmask <- make.mask(traps(possumCH), type = 'polygon',
     spacing = 10, poly = possumremovalarea)
> surface2a <- predictDsurface(fit2, regionmask)
> plot(surface2a, border=150, polycol = 'blue', plottype = 'shaded',
     col = terrain.colors(30), breaks = 25, mesh = NA)
> plot(traps(possumCH), add = TRUE)
> plot(surface2a, plottype = 'contour', levels = seq(0.2,4,0.2),
     add = TRUE)
```



From this plot the quadratic surface is clearly implausible when extrapolated well beyond the traps: there is no reason to expect very low density in the northern and southern sectors, nor the coastal peaks in the east and the west.

The artificiality of the polynomial surface leads us to consider a more plausible spatial model. If indeed there is variation among grids, perhaps it reflects a habitat gradient away from the shore? This is not entirely realistic (the western shore is marine, the eastern one an estuary), but it has potential.

First we need to define a covariate for distance to shore. This is easily computed with the secr 'distancetotrap' function as we have access to points along the shoreline in the object `possumarea` (from version 2.2.0 it no longer matters that this is not a traps object). Then we repeat the steps of fitting the model, comparing by AIC and plotting the surface. We choose the identity link so that the function relating density to distance from the shore is linear rather than exponential. The usual maximization algorithm failed, so we resort to the more robust Nelder-Mead method.

```
> dts <- distancetotrap(possummask, possumarea)
> covariates(possummask) <- data.frame(d.to.shore = dts)
> fit3 <- secr.fit (possumCH, model = list(D ~ d.to.shore,
     g0 ~ h2, sigma ~ h2), mask = possummask, trace = FALSE,
     link = list (D = 'identity'), method = 'Nelder-Mead')
```

```
> AIC (fit1, fit2, fit3)
```

To evaluate the new surface over the entire peninsula we would use:

```
> dts <- distancetotrap(regionmask, possumarea)
> covariates(regionmask) <- data.frame(d.to.shore = dts)
> surface3a <- predictDsurface(fit3, regionmask)
> plot(surface3a, border=150, polycol = 'blue', plottype = 'shaded',
     col = terrain.colors(30), breaks = 25, mesh = NA)
> plot(traps(possumCH), add = TRUE)
> plot(surface3a, plottype = 'contour', levels = seq(0.2,4,0.2), add = TRUE)
```

Although this model is not strong in terms of overall fit, its AICc is lower than that of the quadratic model, and it has the major advantage of not predicting implausible values immediately outside the state space used in fitting. Even this model breaks down if pushed a little further: it implies brushtail possum populations decline to zero about 2.1 km inland!

Potential problems

Modelling density surfaces can be tricky. Recognise when model fitting has failed. If there is no asymptotic variance-covariance matrix, the estimates cannot be trusted. Some forensic work may be needed. If in doubt, try repeating the fit, perhaps starting from the previously fitted values (you can use `secr.fit(..., start=last.model)` where `last.model` is a previously fitted secr object) or from new arbitrary values. Problems may result when the discretization is too coarse, so try with smaller mask cells.

It pays to adjust your model so the absolute expected values are roughly similar for all parameters on their respective 'link' scales, rather than varying by orders of magnitude. This can be achieved by setting the `typsize` argument of `nlm` or the `parscale` control argument of `optim`. Another solution that has sometimes worked for us is to set the 'details' option `hessian = 'fdhess'`.

You can try another optimization method; method = 'Nelder-Mead' is generally more robust than the default gradient-based method. Any method may fail to find the true maximum from a given starting point. We have no experience with simulated annealing (SANN in `optim`); it is reputedly effective, but slow. In the `optim` help it is stated ominously that "the 'SANN' method depends critically on the settings of the control parameters. It is not a general-purpose method".

'secr' scales x- and y-coordinates to mean = 0, SD = 1 before using the coordinates in a model. Remember this when you come to use the coefficients.

Functions such as `predictDsurface` should take care of scaling automatically (if not, please report a bug). The mean and SD used in scaling are saved as the 'meanSD' attribute of a mask (dataframe with columns x,y, rows mean SD). Scaling of covariates other than x and y is up to the user.

Scaling is not performed routinely by `secr.fit` for other operations. Sometimes, large numeric values in coordinates can cause loss of precision in distance calculations (there are a lot of them at each likelihood evaluation). The problem is serious in datasets that combine large coordinates with small detector spacing, such as the Lake Station skink dataset. Set details$centred = TRUE to force scaling; this may become the default setting in a future version of secr.

Avoid using [to extract subsets from mask, capthist and other secr objects objects. Use the provided `subset` methods. (With care, it is possible to replace selected elements in situ, but note that changing coordinates will invalidate the meanSD attribute).

Do you really want the default log link?

References

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

Lewin-Koh, N. J., Bivand, R., et al. (2011). maptools: Tools for reading and handling spatial objects. R package version 0.8-10. http://CRAN.R-project.org/package=maptools

Pebesma, E.J. and Bivand, R. S. (2005) Classes and methods for spatial data in R. R News 5(2). http://cran.r-project.org/doc/Rnews/

Pledger, S. (2000) Unified maximum likelihood estimates for closed capture–recapture models using mixtures. *Biometrics* **56**, 434–442.