Polygon and transect detectors in **secr**

Murray Efford

2014-11-17

Contents

Example data: flat-tailed horned lizards 1
Data input 2
Model fitting 2
Cue data 3
Transect search 4
More on polygons 6
Technical notes 6
References 7

The 'polygon' detector type is used for data from searches of one or more areas (polygons). Transect detectors are the linear equivalent of polygons; as the theory and implementation are very similar we mostly refer to polygon detectors and only briefly mention transects. [We do not consider here searches of linear habitats such as rivers]. Area and linear searches differ from other modes of detection in that each detection may have different coordinates, and the coordinates are random rather than fixed by the field design. The method may be used with individually identifiable cues (e.g., faeces) as well as for direct observations of individuals.

Polygons may be independent (detector type 'polygon') or exclusive (detector type 'polygonX'). Exclusivity is a particular type of dependence in which an animal may be detected at no more than one polygon on each occasion (i.e. polygons function more like multi-catch traps than 'count' detectors). Transect detectors also may be independent ('transect') or exclusive ('transectX').

Efford (2011) gives technical background on the fitting of polygon and transect models to spatially explicit capture–recapture data by maximum likelihood. This document illustrates the methods using the R package **secr**.

Example data: flat-tailed horned lizards

Royle and Young (2008) reported a Bayesian analysis of data from repeated searches for flat-tailed horned lizards (*Phrynosoma mcallii*) on a 9-ha square plot in Arizona, USA. Their dataset is included in **secr** as `hornedlizardCH` and will be used for demonstration. See '?hornedlizard' for more details.

The lizards were free to move across the boundary of the plot and often buried themselves when approached. Half of the 134 different lizards were seen only once in 14 searches over 17 days. Fig. 1 shows the distribution of detections within the quadrat; lines connect successive detections of the individuals that were recaptured.

```
library(secr)
```

```
## This is secr 2.9.1. For overview type ?secr
```

```
plot(hornedlizardCH, tracks = TRUE, varycol = FALSE, lab1cap = TRUE, laboffset = 8,
     border = 10, title ='')
```

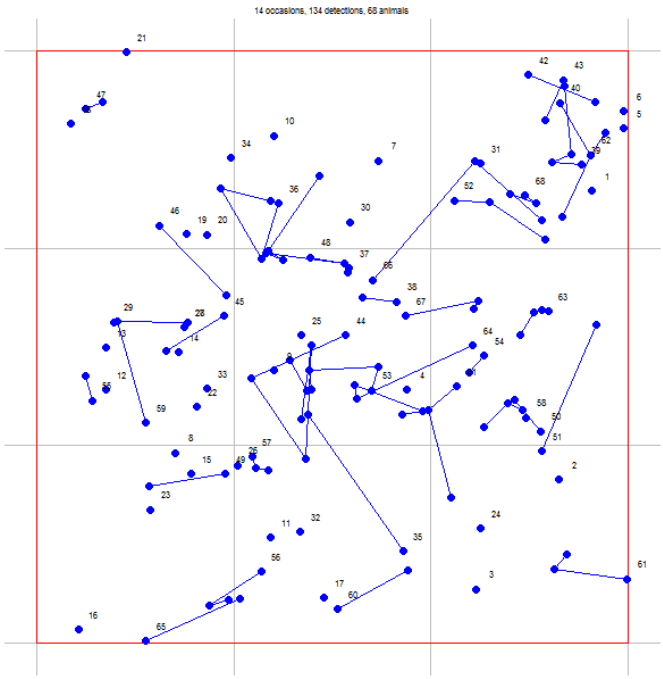

Fig. 1. Locations of horned lizards on a 9-ha plot in Arizona (Royle and Young 2008). Grid lines are 100 m apart.

Data input

Input of data for polygon and transect detectors is described in secr-datainput.pdf. It is little different to input of other data for secr. The key function is `read.capthist`, which reads text files containing the polygon or transect coordinates[1] and the capture records. Capture data should be in the 'XY' format of Density (one row per record with fields in the order Session, AnimalID, Occasion, X, Y). Capture records are automatically associated with polygons on the basis of X and Y (coordinates outside any polygon give an error). Transect data are also entered as X and Y coordinates and automatically associated with transect lines.

Model fitting

The function `secr.fit` is used to fit polygon or transect models by maximum likelihood, exactly as for other detectors. Any model fitting requires a habitat mask – a representation of the region around the detectors possibly occupied by the detected animals (aka the 'area of integration' or 'state space'). It's simplest to use a simple rectangular buffer around the detectors, specified via the 'buffer' argument of `secr.fit`. Alternatively, one can construct a mask with `make.mask` and provide that in the 'mask' argument of `secr.fit`. Pre-building the mask in this way can be more efficient as points can be dropped that are within the rectangle but far from detectors (see Transect search). For the horned lizard dataset it is safe to use the default buffer width (100 m) and the default detection function (circular bivariate normal). We use `trace = FALSE` to suppress intermediate output that would be untidy here.

```
FTHL.fit <- secr.fit(hornedlizardCH, detectfn = 'HN', trace = FALSE)
```

```
predict(FTHL.fit)
```

[1] For constraints on the shape of polygon detectors see Polygon shape

```
##         link estimate SE.estimate     lcl     ucl
## D        log   8.0578     1.06435  6.2268 10.4271
## g0     logit   0.1241     0.01332  0.1003  0.1526
## sigma    log  18.5054     1.19897 16.3008 21.0083
```

The estimated density is 8.06 / ha, somewhat less than the value given by Royle and Young (2008); see Efford (2011) for an explanation. The parameter labelled 'g0' is equivalent to p in Royle and Young (2008).

`FTHL.fit` is an object of class secr. Many methods are available for secr objects (`AIC`, `coef`, `deviance`, `print`, etc.) – see the **secr** help index or Appendix 4 of secr-overview.pdf. We would use the 'plot' method to graph the fitted detection function :

```
plot(FTHL.fit, xv = 0:70, ylab = 'p')
```

Cue data

By 'cue' in this context we mean a discrete sign identifiable to an individual animal by means such as microsatellite DNA. Faeces and passive hair samples may be cues. Animals may produce more than one cue per occasion. The number of cues in a specific polygon then has a discrete distribution such as Poisson, binomial or negative binomial.

A cue dataset is not readily available, so we simulate some cue data to demonstrate the analysis. The text file 'temppoly.txt' contains the boundary coordinates.

```
temppoly <- read.traps(file = 'temppoly.txt', detector = 'polygon')
tempcapt <- sim.capthist(temppoly, popn = list(D = 1, buffer = 200), detectpar =
                           list(g0 = 5, sigma = 50), noccasions = 1)
par(mar=c(1,2,3,2))
plot(tempcapt, tracks = TRUE, varycol = F, lab1cap = T, laboffset = 15, title =
       paste("Simulated 'polygon' data", "D = 1, g0 = 5, sigma = 50"))
```

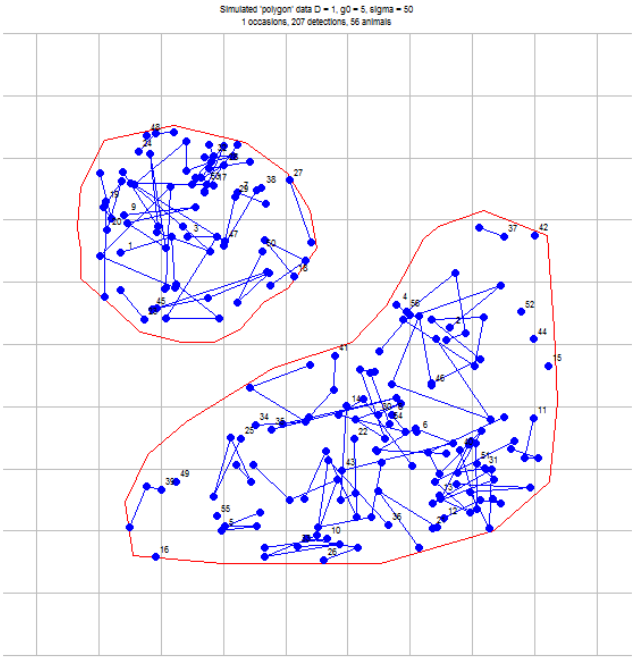

Fig. 2. Simulated cue data from a single search of two irregular polygons.

Our simulated sampling was a single search (noccasions = 1), and the intercept of the detection function (g0 = 5) is the expected number of cues that would be found per animal if the search was unbounded. The plot is slightly misleading because the cues are not ordered in time, but tracks = TRUE serves to link cues from the same animal.

To fit the model by maximum likelihood we use `secr.fit` as before:

```
cuesim.fit <- secr.fit(tempcapt, buffer = 200, trace = FALSE)
```

```
predict(cuesim.fit)
```

```
##       link estimate SE.estimate     lcl     ucl
## D      log   1.8661     0.24627  1.4424  2.4142
## g0     log   0.5247     0.06046  0.4189  0.6571
## sigma  log  54.4066     3.45907 48.0384 61.6190
```

Transect search

Transect data, as understood here, include the positions from which individuals are detected along a linear route through 2-dimensional habitat. They *do not* include distancesfrom the route to the location of the individual, at least, not yet. A route may be searched multiple times, and a dataset may include multiple routes, but neither of these is necessary. [Searches of linear habitat such as river banks require a different approach - see the forthcoming package **secrlinear**.]

We simulate some data for an imaginary wiggly transect.

```
x <- seq(0, 4*pi, length = 20)
temptrans <- make.transect(x = x*100, y = sin(x)*300, exclusive = FALSE)
summary(temptrans)
```

```
## Object class      traps
## Detector type     transect
## Number vertices   20
## Number transects  1
## Total length      2756 m
## x-range           0 1257 m
## y-range           -299 299 m
```

```
tempcapt <- sim.capthist(temptrans, popn = list(D = 2, buffer = 300), detectpar =
                           list(g0 = 1.0, sigma = 50), binomN = 0)
```

By setting exclusive = FALSE we signal that there may be more than one detection per animal per occasion on this single transect (i.e. this is a 'transect' detector rather than 'transectX').

Constructing a habitat mask explicitly with `make.mask` (rather than relying on 'buffer' in `secr.fit`) allows us to specify the point spacing and discard outlying points (Fig. 3).

```
tempmask <- make.mask(temptrans, type = 'trapbuffer', buffer = 300, spacing = 20)
```

```
par(mar=c(3,1,3,1))
plot(tempmask, border = 0)
plot(temptrans, add = TRUE, detpar = list(lwd = 2))
plot(tempcapt, tracks = TRUE, add = TRUE, title = '')
```

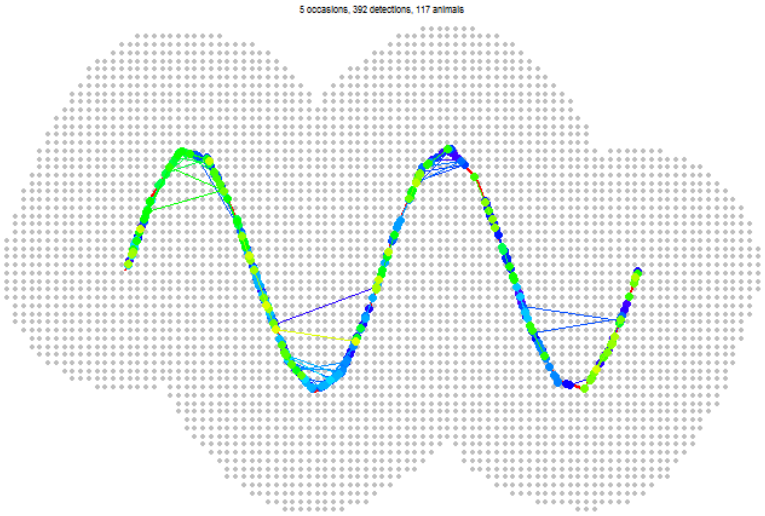

Fig. 3. Habitat mask (grey dots) and simulated transect data from five searches of a 2.8-km transect. Colours differ between individuals, but are not unique.

Model fitting uses `secr.fit` as before. We specify the distribution of the number of detections per individual per occasion as Poisson (binomN = 0), although this also happens to be the default. Setting method = 'BFGS' is slightly more likely to yield valid estimates of standard errors than using the default method (see Technical notes).

```
transim.fit <- secr.fit(tempcapt, mask = tempmask, binomN = 0, method = 'BFGS', trace = FALSE)
```

```
predict (transim.fit)
```

```
##       link estimate SE.estimate     lcl    ucl
## D      log    1.783     0.19487  1.4406  2.208
## g0     log    1.046     0.08539  0.8918  1.227
## sigma  log   50.822     1.99349 47.0629 54.882
```

Another way to analyse transect data is to discretize it. We divide the transect into 25-m segments and then change the detector type. In the resulting capthist object the transect has been replaced by a series of proximity detectors, each at the midpoint of a segment.

```
newCH <- snip(tempcapt, by = 25)
newCH <- reduce(newCH, outputdetector = 'proximity')
```

We can fit a model using the same mask as before. The result differs in the scaling of the g0 parameter, but in other respects is similar to that from the transect model.

```
snipped.fit <- secr.fit(newCH, mask = tempmask, trace = FALSE)
```

```
predict(snipped.fit)
```

```
##        link estimate SE.estimate     lcl     ucl
## D       log   1.7623     0.19647  1.4174  2.1912
## g0    logit   0.1853     0.01689  0.1545  0.2208
## sigma   log  52.0743     2.36679 47.6382 56.9234
```

More on polygons

The implementation in **secr** allows any number of disjunct polygons or non-intersecting transects.

Polygons may be irregularly shaped, but there are some limitations. Polygons may not be concave in an east-west direction, in the sense that there are more than two intersections with a vertical line. Sometimes east-west concavity may be fixed by rotating the polygon and its associated data points (see function `rotate`).

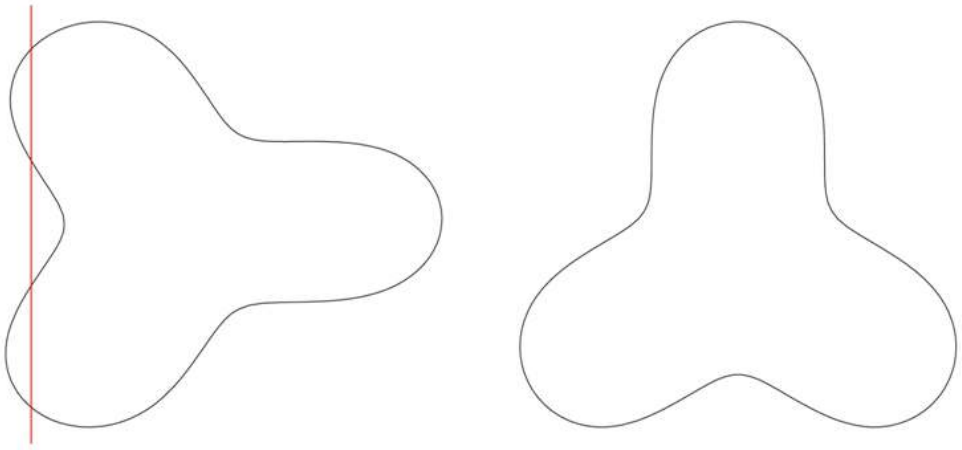

Fig. 4. The polygon on the left is not allowed because its boundary is intersected by a vertical line at more than two points.

Technical notes

Fitting models for polygon detectors with `secr.fit` requires the hazard function to be integrated in two-dimensions many times. This is done with repeated one-dimensional gaussian quadrature using the C function Rdqags provided by R (Rdqags is also used by R's own function `integrate`) (see R manual 'Writing R extensions'). Error messages including 'ier' may be traced in the code for Rdqags. A few such errors during maximisation may be ignored, as long as they do not occur at the end.

Polygon and transect SECR models seem to be prone to numerical problems in estimating the information matrix (negative Hessian), which flow on into poor variance estimates and missing values for the standard errors of 'real' parameters. At the time of writing these seem to be overcome by overriding the default maximisation method (Newton-Raphson in 'nlm') and using, for example, "method = 'BFGS' ". Another solution, perhaps more reliable, is to compute the information matrix independently by setting 'details = list(hessian = 'fdhess')' in the call to `secr.fit`. Yet another approach is to apply `secr.fit` with "method = 'none' " to a previously fitted model to compute the variances.

The algorithm for finding a starting point in parameter space for the numerical maximisation is not entirely reliable; it may be necessary to specify the 'start' argument of `secr.fit` manually, remembering that the values should be on the link scale (defaults – D: log, lambda0: log, g0: logit, sigma: log).

Data for polygons and transects are unlike those from detectors such as traps in several respects:

- The association between vertices in a 'traps' object and polygons or transects resides in an attribute 'polyID' that is out of sight, but may be retrieved with the `polyID` or `transectID` functions. If the attribute is NULL, all vertices are assumed to belong to one polygon or transect.

- The x-y coordinates for each detection are stored in the attribute 'detectedXY' of a capthist object. To retrieve these coordinates use the function `xy`. Detections are ordered by occasion, animal, and detector (i.e., polyID).
- `subset` or `split` applied to a polygon or transect 'traps' object operate at the level of whole polygons or transects, not vertices (rows).
- `usage` also applies to whole polygons or transects. The option of specifying varying usage by occasion is not fully tested for these detector types.
- The interpretation of detection functions and their parameters is subtly different; the detection function must be integrated over 1-D or 2-D rather than yielding a probability directly (see Efford 2011).

References

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

Efford, M. G. (2011) Estimation of population density by spatially explicit capture–recapture analysis of data from area searches. *Ecology* **92**, 2202–2207.

Marques, T. A., Thomas, L. and Royle, J. A. (2011) A hierarchical model for spatial capture–recapture data: Comment. *Ecology* **92**, 526–528.

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