Polygon and transect detectors in **secr** 5.2

Murray Efford

2025-01-26

Contents

Theory **2**

Parameterisation **3**

Example data: flat-tailed horned lizards **3**

Data input **4**

Model fitting **4**

Cue data **5**

Discretizing polygon data **6**

Transect search **7**

More on polygons **9**
Solutions for non-conforming polygons 10

Technical notes **11**

References **12**

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 transect searches differ from other modes of detection in that each detection may have different coordinates, and the coordinates are continuously distributed rather than constrained to fixed points 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**. The theory is briefly sketched before moving on to an example.

Theory

This description has been modified slightly from Efford (2011), emphasising the case of independent Poisson-distributed counts for a single search area on a single occasion, and omitting the dependence of each term ($\Pr(n), \lambda(\mathbf{x}), h(\mathbf{u})$) on the various parameters (D, λ_c and σ).

The basic model relates the expected number of detections of an individual within an irregular searched area to the quantitative overlap between its home range[1] (assumed radially symmetrical) and the area (Fig. 1).

```
## [1] 18
```

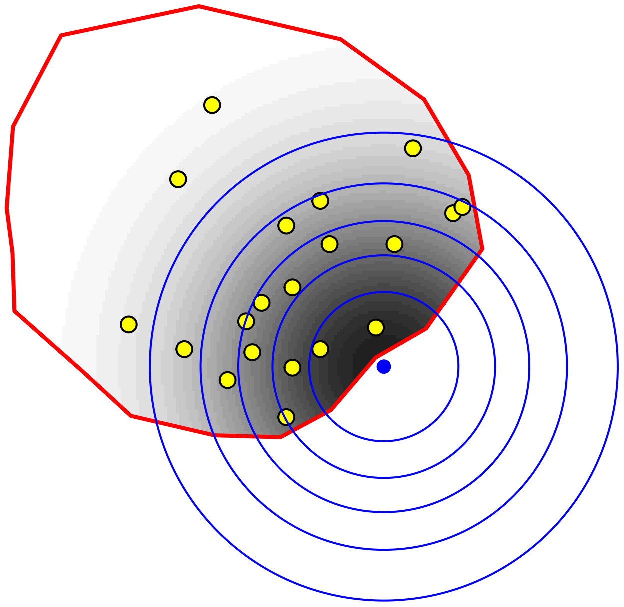


Fig. 1. Detections of one animal, centered at the blue dot, on an irregular searched polygon. The expected number of detections is modelled by the quantitative overlap (grey shading) between a radially symmetrical probability density (circular contours) and the searched area (red outline). Yellow dots indicate a random sample of cues from this animal.

If λ_c is the expected number of detections (detected cues) of an animal whose home range lies completely within the area κ, and $h(u|\mathbf{x})$ is proportional to the activity at point $\mathbf{u}$ for an animal centred at $\mathbf{x}$, then the expected number of detections of an individual on one occasion is

$$\lambda(\mathbf{x}) = \frac{\lambda_c}{H(\mathbf{x})} \int_\kappa h(\mathbf{u}|\mathbf{x})\, du, \tag{1}$$

where $H(\mathbf{x}) = \int_{R^2} h(\mathbf{u}|\mathbf{x})\, d\mathbf{u}$ is a normalising constant. If $h(\mathbf{u}|\mathbf{x})$ is a pdf then normalisation is unnecessary. Further, $H(\mathbf{x})$ is constant for all $\mathbf{x}$ in **secr** 5.2[2].

The likelihood component associated with J detections of animal i is then a product of the probability of observing J detections, and the pdf associated with each detection:

$$L_i = \frac{\lambda(\mathbf{x})^J e^{-\lambda(\mathbf{x})}}{J!} . \prod_{j=1}^{J} \frac{h(y_{ij}|\mathbf{x})}{\int_\kappa h(\mathbf{u}|\mathbf{x})\, du}.$$

[1]The term is used here very loosely - a more nuanced explanation would distinguish between the stationary distribution of activity (the home-range utilisation distribution) and the spatial distribution of cues (opportunities for detection) generated by an individual.

[2]This is because detection parameters in **secr** are not allowed to vary over space except as driven by a non-Euclidean distance metric, and non-Euclidean distances are only allowed with point detectors ("multi", "proximity", "count" etc.).

The likelihood components L_i are combined across all n individuals:

$$L = \Pr(n) \prod_{i=1}^{n} L_i.$$

Parameterisation

The detection model is fundamentally different for polygon detectors and detectors at a point ("single", "multi", "proximity", "capped", "count"):

- For point detectors, the detection function directly models the probability of detection or the expected number of detections. All that matters is the distance between the animal's centre and the detector.
- For polygon detectors, these quantities (probability or expected number) depend also on the geometrical relationships (Fig. 1) and the integration in equation 1. The detection function serves only to define the *potential* detections if the search area was unbounded (blue contours in Fig. 1).

We use a parameterisation that separates two aspects of detection – the expected number of cues from an individual (λ_c) and their spatial distribution given the animal's location ($h(\mathbf{u}|\mathbf{x})$ normalised by dividing by $H(\mathbf{x})$ (1). The parameters of $h()$ are those of a typical detection function in **secr** (e.g., λ_0, σ), except that the factor λ_0 cancels out of the normalised expression. The expected number of cues, given an unbounded search area, is determined by a different parameter here labelled λ_c.

There are complications:

1. Rather than designate a new 'real' parameter lambdac, **secr** grabs the redundant intercept of the detection function (lambda0) and uses that for λ_c. Bear this in mind when reading output from polygon or transect models.

2. If each animal can be detected at most once per detector per occasion (as with exclusive detector types 'polygonX' and 'transectX') then instead of $\lambda(\mathbf{x})$ we require a probability of detection between 0 and 1, say $g(\mathbf{x})$. In **secr 5.2** the probability of detection is derived from the cumulative hazard using $g(\mathbf{x}) = 1 - \exp(-\lambda(\mathbf{x}))$. The horned lizard dataset of Royle and Young (2008) has detector type 'polygonX' and their parameter 'p' was equivalent to $1 - \exp(-\lambda_c)$ $(0 < p \leq 1)$. For the same scenario and parameter Efford (2011) used p_∞.

3. Unrelated to (2), detection functions in **secr** may model either the probability of detection (HN, HR, EX etc.) or the cumulative hazard of detection (HHN, HHR, HEX etc.) (see `?detectfn` for a list). Although probability and cumulative hazard are mostly interchangeable for point detectors, it's not so simple for polygon and transect detectors. The integration in equation 1 always uses the hazard form for $h(\mathbf{u}|\mathbf{x})$ (**secr** 3.0.0 and later)[3], and only hazard-based detection functions are allowed (HHN, HHR, HEX, HAN, HCG, HVP). The default function is HHN.

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. 2 shows the distribution of detections within the quadrat; lines connect successive detections of the individuals that were recaptured.

```r
par(mar=c(1,1,2,1))
plot(hornedlizardCH, tracks = TRUE, varycol = FALSE, lab1cap = TRUE, laboffset = 8,
     border = 10, title ='')
```

[3]The logic here is that hazards are additive whereas probabilities are not.

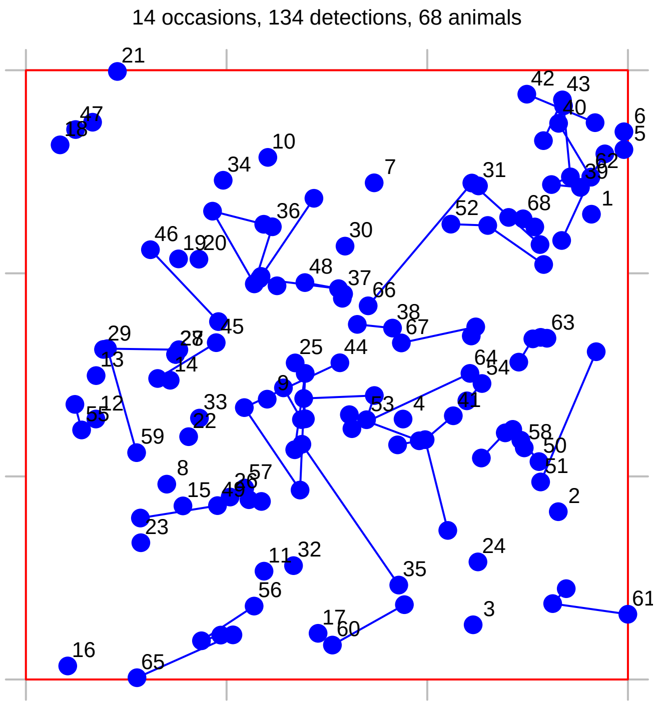


Fig. 2. Locations of horned lizards on a 9-ha plot in Arizona (Royle and Young 2008). Grid lines are 100 m apart.

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[4] 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 buffer around the detectors, specified via the 'buffer' argument of `secr.fit`[5]. 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, buffer = 80, trace = FALSE)
```

```
## Warning: using default starting values
```

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

```
##         link   estimate SE.estimate        lcl        ucl
```

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

[5]Alternatively, one can construct a mask with `make.mask` and provide that in the 'mask' argument of `secr.fit`. Note that `make.mask` defaults to `type = 'rectangular'`; see Transect search for an example in which points are dropped if they are within the rectangle but far from detectors (the default in `secr.fit`)

```
## D        log  8.0130680  1.06170100  6.1873999 10.3774218
## lambda0  log  0.1317132  0.01512831  0.1052403  0.1648453
## sigma    log 18.5049025  1.19938839 16.2995006 21.0087060
```

The estimated density is 8.01 / ha, somewhat less than the value given by Royle and Young (2008); see Efford (2011) for an explanation, also Dorazio (2013). The parameter labelled 'lambda0' (i.e. λ_c) is equivalent to p in Royle and Young (2008) (using $\hat{p} \approx 1 - \exp(-\hat{\lambda}_c)$).

`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 'polygonexample1.txt' contains the boundary coordinates.

```
datadir <- system.file("extdata", package = "secr")
examplefile1 <- paste0(datadir, '/polygonexample1.txt')
polyexample1 <- read.traps(file = examplefile1, detector = 'polygon')
polygonCH <- sim.capthist(polyexample1, popn = list(D = 1, buffer = 200),
    detectfn = 'HHN', detectpar = list(lambda0 = 5, sigma = 50),
    noccasions = 1, seed = 123)
```

```
par(mar = c(1,2,3,2))
plot(polygonCH, tracks = TRUE, varycol = FALSE, lab1cap = TRUE, laboffset = 15,
     title = paste("Simulated 'polygon' data", "D = 1, lambda0 = 5, sigma = 50"))
```

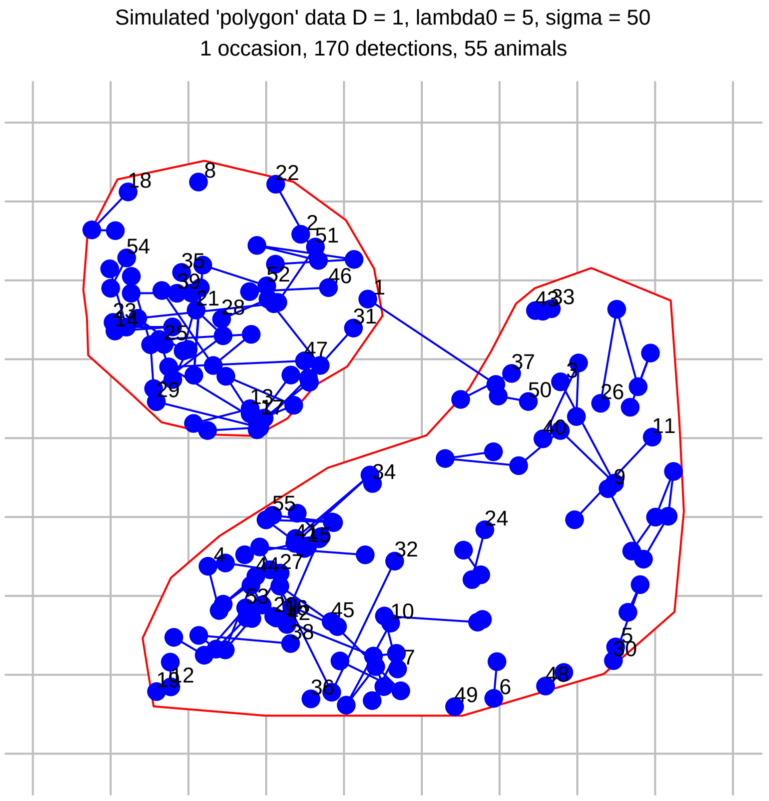


Fig. 3. Simulated cue data from a single search of two irregular polygons.

Our simulated sampling was a single search (noccasions = 1), and the intercept of the detection function (lambda0 = 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(polygonCH, buffer = 200, trace = FALSE)
```

```
## Warning: using default starting values
```

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

```
##          link  estimate SE.estimate        lcl       ucl
## D         log  1.103460   0.1536607  0.8409933  1.447841
## lambda0   log  4.376359   0.4188488  3.6293803  5.277076
## sigma     log 49.446414   2.4313307 44.9061287 54.445750
```

Discretizing polygon data

An alternative way to handle polygon capthist data is to convert it to a raster representation i.e. to replace each polygon with a set of point detectors, each located at the centre of a square pixel. Point detectors (‘multi’, ‘proximity’, ‘count’ etc.) tend to be more robust and models often fit faster. They also allow habitat attributes to be associated with detectors at the scale of a pixel rather than the whole polygon. The **secr** function `discretize` performs the necessary conversion in a single step. Selection of an appropriate pixel size (`spacing`) is up to the user. There is a tradeoff between faster execution (larger pixels are better) and controlling artefacts from the discretization, which can be checked by comparing estimates with different levels of `spacing`.

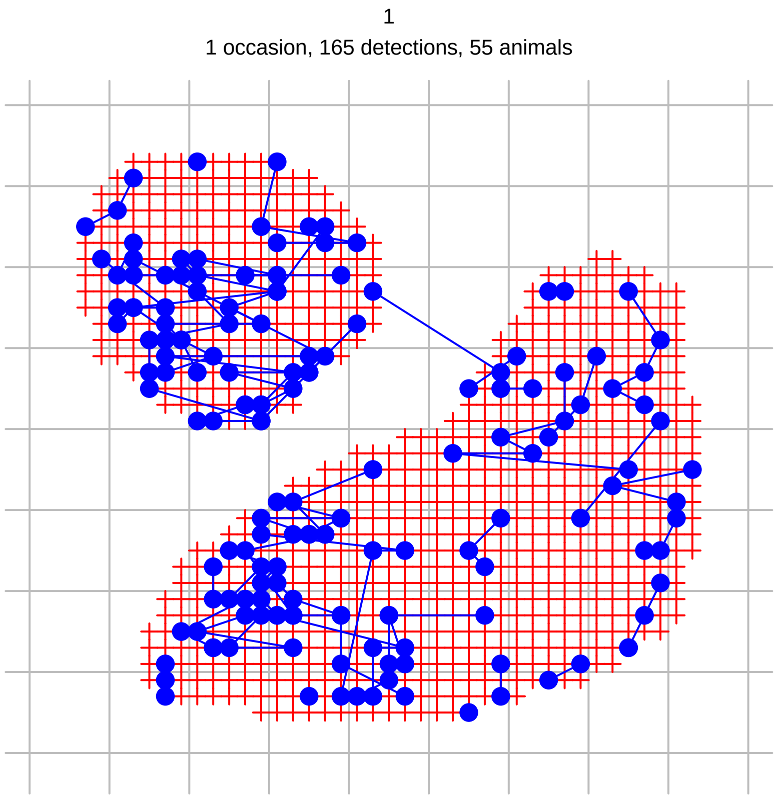
Taking our example from before,

```
discreteCH <- discretize (polygonCH, spacing = 20)
```

```
## Warning: count data converted to binary; information may be lost
```

```
par(mar = c(1,2,3,2))
plot(discreteCH, varycol = FALSE, tracks = TRUE)
```

```
## Warning: track for repeat detections on same occasion joins points in arbitrary
## sequence
```



```
discrete.fit <- secr.fit(discreteCH, buffer = 200, detectfn = 'HHN', trace = FALSE)
```

```
## Warning in dpois(data$nc, N * meanpdot, log = TRUE): NaNs produced
```

```
predict(discrete.fit)
```

```
##         link   estimate SE.estimate         lcl        ucl
## D        log  1.0953502   0.1527677  0.83446446  1.4377990
## lambda0  log  0.1120455   0.0144916  0.08704808  0.1442215
## sigma    log 49.8384122   2.6739977 44.86701744 55.3606518
```

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 distances from 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 package secrlinear.

We simulate some data for an imaginary wiggly transect.

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

```
## Object class       traps
## Detector type      transect
## Number vertices    20
## Number transects   1
## Total length       2756.105 m
## x-range            0 1256.637 m
## y-range            -298.9753 298.9753 m
```

```
transectCH <- sim.capthist(transect, popn = list(D = 2, buffer = 300),
    detectfn = 'HHN', detectpar = list(lambda0 = 1.0, sigma = 50),
    binomN = 0, seed = 123)
```

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. 4).

```
transectmask <- make.mask(transect, type = 'trapbuffer', buffer = 300, spacing = 20)
par(mar = c(3,1,3,1))
plot(transectmask, border = 0)
plot(transect, add = TRUE, detpar = list(lwd = 2))
plot(transectCH, tracks = TRUE, add = TRUE, title = '')
```

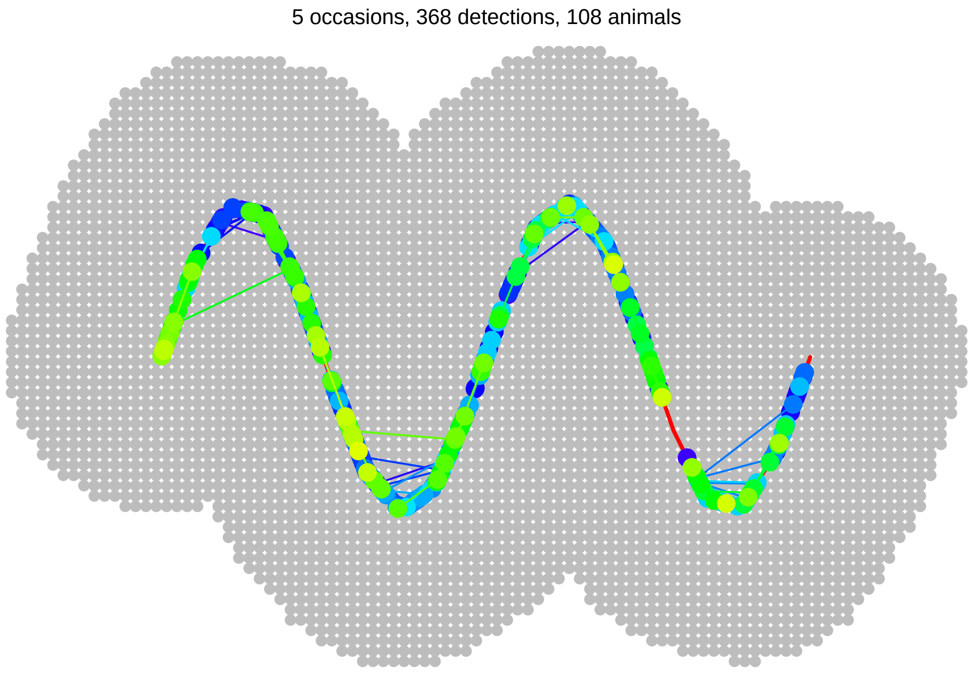


Fig. 4. Habitat mask (grey dots) and simulated transect data from five searches of a 2.8-km transect. Colours differ between individuals, but are not unique.

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 = 'Nelder-Mead' is slightly more likely to yield valid estimates of standard errors than using the default method (see Technical notes).

```
transect.fit <- secr.fit(transectCH, mask = transectmask, binomN = 0,
                         method = 'Nelder-Mead', trace = FALSE)
```

```
## Warning: using default starting values
```

Occasional 'ier' error codes may be ignored (see Technical notes). The estimates are close to the true values except for sigma, which may be positively biased.

```
predict (transect.fit)
```

```
##          link  estimate SE.estimate        lcl       ucl
## D         log  1.852659  0.19667422  1.5055215  2.279839
## lambda0   log  1.041484  0.08043325  0.8953909  1.211415
## sigma     log 53.629004  2.31338666 49.2831701 58.358057
```

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.

```
snippedCH <- snip(transectCH, by = 25)
snippedCH <- reduce(snippedCH, outputdetector = 'proximity')
```

The same may be achieved with `newCH <- discretize(transectCH, spacing = 25)`. We can fit a model using the same mask as before. The result differs in the scaling of the lambda0 parameter, but in other respects is similar to that from the transect model.

```
snipped.fit <- secr.fit(snippedCH, mask = transectmask, detectfn = 'HHN', trace = FALSE)
predict(snipped.fit)
```

```
##          link   estimate SE.estimate        lcl        ucl
## D         log  1.8421677  0.19562536  1.4968932  2.2670837
## lambda0   log  0.1931212  0.01803396  0.1608851  0.2318163
## sigma     log 53.6790082  2.32112866 49.3190781 58.4243672
```

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 in the default implementation. 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`). Polygons should not contain holes, and the polygons used on any one occasion should not overlap.

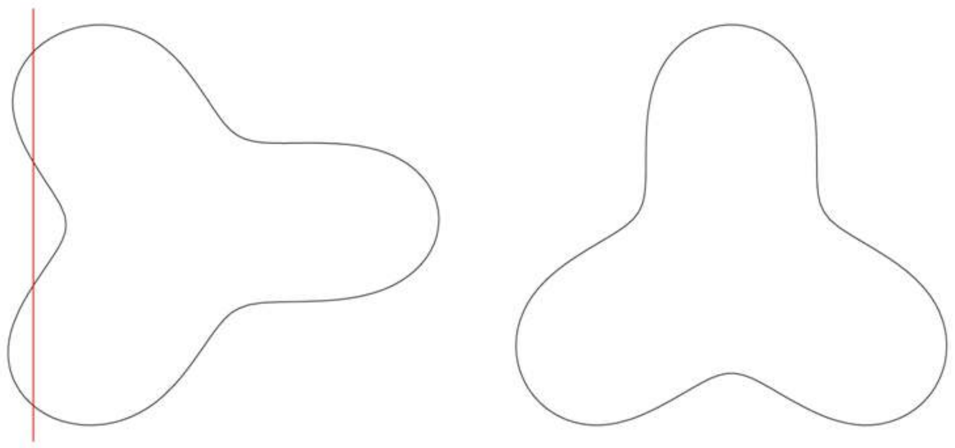


Fig. 5. The polygon on the left is not allowed because its boundary is intersected by a vertical line at more than two points.

Solutions for non-conforming polygons

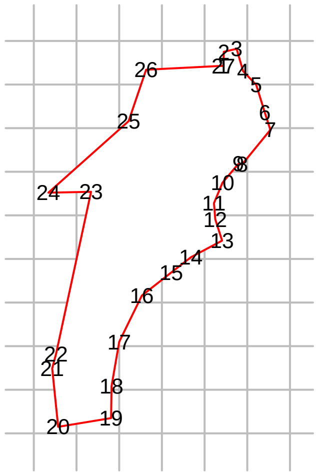
1. Break into parts

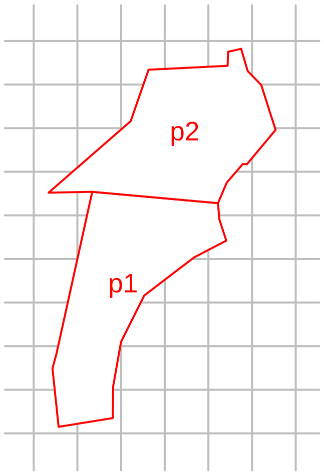
One solution to 'east-west concavity' is to break the offending polygon into two or more parts. For this you need to know which vertices belong in which part, but that is (usually) easily determined from a plot. In this real example we recognise vertices 11 and 23 as critical, and split the polygon there. Note the need to include the clip vertices in both polygons, and to maintain the order of vertices. Both `polyexample2` and `newpoly` are traps objects.

```r
examplefile2 <- paste0(datadir, '/polygonexample2.txt')
polyexample2 <- read.traps(file = examplefile2, detector = 'polygon')
par(mfrow = c(1,2), mar = c(2,1,1,1))
plot(polyexample2)
text(polyexample2$x, polyexample2$y, 1:nrow(polyexample2), cex = 0.7)
newpoly <- make.poly (list(p1 = polyexample2[11:23,],
  p2 = polyexample2[c(1:11, 23:27),]))
```

```
## No errors found :-)
```

```r
plot(newpoly, label = TRUE)
```





Attributes such as covariates and usage must be rebuilt by hand.

2. Pointwise testing

Another solution is to evaluate whether each point chosen dynamically by the integration code lies inside the polygon of interest. This is inevitably slower than the default algorithm that assumes all points between the lower and upper intersections lie within the polygon. Select the slower, more general option by setting `details = list(convexpolygon = FALSE)`.

Technical notes

Fitting models for polygon detectors with `secr.fit` requires the hazard function to be integrated in two-dimensions many times. In **secr** >= 4.4 this is done with repeated one-dimensional Gaussian quadrature using the C++ function 'integrate' in RcppNumerical (Qiu et al. 2019).

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 (default log for D, lambda0 and sigma).

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.

Dorazio, R. M. (2013) Bayes and empirical Bayes estimators of abundance and density from spatial capture–recapture data. *PLoS ONE* **8**, e84017.

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.

Qiu, Y., Balan, S., Beall, M., Sauder, M., Okazaki, N. and Hahn, T. (2019) RcppNumerical: 'Rcpp' Integration for Numerical Computing Libraries. R package version 0.3-3. https://CRAN.R-project.org/package=Rcpp Numerical

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