Data input for **secr**

Murray Efford

2026-11-25

Contents

Introduction **1**

Text formats - general **2**

Capture data format **2**

Detector layout format **3**

Using `read.capthist` **4**

When read.capthist is inadequate... **5**

Reading Excel files **5**

Other detector types **6**
Count detectors 6
Signal detectors 6
Polygon and transect detectors 6
Telemetry 8
Mark–resight 8

Troubleshooting **8**

Further notes **9**

References **9**

Appendix: Glossary **10**

Data for analysis in **secr** must be prepared as an R object of class 'capthist'. This object includes both the detector layout and the capture data. The structure of a capthist object is complex and depends on the detector type. Functions `make.capthist` or `read.capthist` are used to construct a capthist object from data already in R or from text files. This vignette describes data input directly from text files with `read.capthist`, which will be adequate for most analyses.

Introduction

The text file formats used by `read.capthist` are shared with program DENSITY (Efford 2012). Two types of file are needed, one for capture data and one for detector (trap) layouts. We use the jargon terms 'detector', 'identifier', 'covariate', 'session' and 'occasion'; if you are not familiar with these as used in **secr** then consult the Glossary.

Text formats - general

Input files should be prepared with a text editor, not a word processing program. Values are usually separated by blanks or tabs, but commas may also be used. Input files with extension '.csv' are recognised automatically as comma-delimited. Each line of header information should start with the comment character (default #).

Input files may also be prepared with spreadsheet software (see later section on reading Excel files).

Identifiers may be numeric values or alphanumeric values with no included spaces, tabs or commas. The underscore character should not be used in detector (trap) identifiers. Leading zeros in identifier fields will be taken literally ('01' is read as '01', not '1'), and it is essential to be consistent between the capture data file and the detector layout file when using 'trapID' format.

Capture data format

Capture data are read from a single text file with one detection record per line. Each detection record starts with a session identifier, an animal identifier, an occasion number, and the location of the detection. Location is usually given in 'trapID' format as a detector (trap) identifier that matches a detector in the trap layout file (below)[1]

Here is a simple example - the capture data for the 'stoatCH' dataset:

```
# Session ID Occasion Detector
MatakitakiStoats 2 1 A12
MatakitakiStoats 2 2 A12
MatakitakiStoats 9 2 A4
MatakitakiStoats 1 1 A9
... 22 lines omitted ...
MatakitakiStoats 7 6 G7
MatakitakiStoats 20 7 G8
MatakitakiStoats 17 4 G9
MatakitakiStoats 19 6 G9
```

The first line is ignored and is not needed. There is a single session 'MatakitakiStoats'. Individuals are numbered 1 to 20 (these identifiers could also have been alphanumeric). Detector identifiers 'A12', 'B5' etc. match the detector layout file as we shall see next. Animal 2 was detected at detector 'A12' on both day 1 and day 2. The order of records does not matter.

A study may include multiple sessions. All detections are placed in one file and sessions are distinguished by the identifier in the first column.

Animals sometimes die on capture or are removed during a session. Mark these detections with a minus sign before the occasion number.

Further columns may be added for individual covariates such as length or sex. Categorical covariates such as sex may use alphanumeric codes (e.g., 'F', 'M'; quotes not needed). Individual covariates are assumed to be permanent, at least within a session, and only the first non-missing value is used for each individual. Missing values on a particular occasion may be indicated with 'NA'. If a covariate is coded 'NA' on all occasions that an animal is detected then the overall status of the animal is 'NA'. Covariates with missing values like this may be used in hybrid mixture models (hcov), but not in other analyses.

For multi-session data it is necessary that the levels of a factor covariate are the same in each session, whether or not all levels are used. This may require the levels in the final capthist object to be set manually.

[1]The older, but still supported, 'XY' format uses the actual x- and y-coordinates of the detection, but this is risky as coordinates must exactly match those in the trap layout file.

Detector layout format

The basic format for a detector (trap) layout file simply gives the x- and y-coordinates for each detector, one per line. Coordinates must relate to a Cartesian (rectangular, projected) coordinate system. If your coordinates are geographic (latitude,longitude) then you must first project them (see secr-spatialdata.pdf).

```
# Detector X Y
 A1   -1500   -1500
 A2   -1500   -1250
 A3   -1500   -1000
 A4   -1500    -750
... 86 lines omitted ...
G10    1500     750
G11    1500    1000
G12    1500    1250
G13    1500    1500
```

This format may optionally be extended to identify occasions when particular detectors were not operated. A string of ones and zeros is added to each line, indicating the occasions when each detector was used or not used. The number of 'usage' codes should equal the number of occasions. Codes may be separated by white space (blanks, tabs, or commas). This is a fictitious example of a 7-day study in which detector A1 was not operated on day 1 or day 2 and detector A4 was not operated on day 6 or day 7:

```
# Detector X Y Usage
 A1   -1500   -1500   0011111
 A2   -1500   -1250   1111111
 A3   -1500   -1000   1111111
 A4   -1500    -750   1111100
 etc.
```

Usage is not restricted to binary values. Numeric values for detector-specific effort on each occasion are added to each line. The number of values should equal the number of occasions, as for binary usage. Values must be separated by white space; for input with `read.traps` or `read.capthist`, set `binary.usage = FALSE`. This is a fictitious example:

```
# Detector X Y Effort
 A1   -1500   -1500   0 0 3.2 5
 A2   -1500   -1250   2 2 2 2
 A3   -1500   -1000   2 2 4 4
 A4   -1500    -750   1 1 2 3
 etc.
```

Detector A2 was operated for the same duration on each occasion; usage of other detectors varied and A1 was not operated at all on the first two occasions. See secr-varyingeffort.pdf for more.

The format also allows one or more detector-level covariates to be coded at the end of each line, separated by one forward slash '/':

```
# Detector X Y Covariates
 A1   -1500   -1500   /0.5 2
 A2   -1500   -1250   /0.5 2
 A3   -1500   -1000   /2   2
 A4   -1500    -750   /2   3
 etc.
```

In this example the vectors of values (0.5, 0.5, 2, 2, ...) and (2, 2, 2, 3, ...) will be saved by default as a variables 'V1' and 'V2' in the covariates dataframe of the traps object. The names may be changed later. Alternatively, the argument 'trapcovnames' may be set in `read.capthist` (see below).

If your detector covariate varies over time (i.e., between occasions), after the slash (/) you should add at least one column for each occasion. Later use `timevaryingcov` in 'secr.fit' to identify a set of s covariate columns associated with occasions 1:s and to give the set of columns a name that may be used in model formulae.

Using `read.capthist`

Having described the file formats, we now demonstrate the use of `read.capthist` to import data to a 'capthist' object. The argument list of `read.capthist` is

```
read.capthist(captfile, trapfile, detector = "multi", fmt = c("trapID", "XY"),
    noccasions = NULL, covnames = NULL, trapcovnames = NULL, cutval = NULL,
    verify = TRUE, noncapt = "NONE", ...)
```

Our stoat example is very simple: apart from specifying the input file names we only need to alter the detector type (see `?detector`). The number of occasions (7) will be determined automatically from the input and there are no individual covariates to be named. The data are in the folder 'extdata' of the package installation.

```
library(secr)
captfile <- system.file("extdata", "stoatcapt.txt", package = "secr")
trapfile <- system.file("extdata", "stoattrap.txt", package = "secr")
stoatCH <- read.capthist(captfile, trapfile, detector = "proximity")
summary(stoatCH)
```

```
## Object class       capthist
## Detector type      proximity
## Detector number    94
## Average spacing    250 m
## x-range            -1500 1500 m
## y-range            -1500 1500 m
##
## Counts by occasion
##                     1  2  3  4  5  6  7 Total
## n                   3  4  8  3  3  7  2    30
## u                   3  3  8  2  0  3  1    20
## f                  12  6  2  0  0  0  0    20
## M(t+1)              3  6 14 16 16 19 20    20
## losses              0  0  0  0  0  0  0     0
## detections          3  4  8  3  3  7  2    30
## detectors visited   3  4  8  3  3  7  2    30
## detectors used     94 94 94 94 94 94 94   658
```

These results match those from loading the 'stoatCH' dataset provided with **secr** (not shown). The message 'No errors found' is from `verify` which can be switched off (`verify = FALSE` in the call to `read.capthist`). The labels 'n', 'u', 'f', and 'M(t+1)' refer to summary counts from Otis et al. (1978); for a legend see `?summary.capthist`.

Under the default settings of `read.capthist`:

- values should be separated by blanks or tabs.
- blank lines are ignored
- any text on a line after the comment character '#' is ignored

The defaults may be changed with settings that are passed by `read.capthist` to `read.table`, specifically

- `sep = ','` for comma-delimited data
- `comment.char = ';'` to change the comment character

If the study includes multiple sessions and the detector layout or usage varies between sessions then it is necessary to provide session-specific detector layout files. This is done by giving 'trapfile' as a vector of names, one per session (repetition allowed; all '.csv' or all not '.csv'). Sessions are sorted numerically if all session identifiers are numeric, otherwise alphanumerically. Care is needed to match the order of layout files to the session order: always confirm the result matches your intention by reviewing the summary.

When read.capthist is inadequate...

Some data do not neatly import with `read.capthist`. You may need to first construct `traps` objects with `read.traps` and then marry them to capture data by a custom call to `make.capthist` (`make.capthist` is called automatically by `read.capthist`). Please consult the help for `read.traps` and `make.capthist`.

Reading Excel files

Input from spreadsheets to R has been problematic. The package **readxl** (Wickham and Bryan 2017) appears now to provide a stable and general solution. From **secr** 3.0.2 onwards `read.traps` and `read.capthist` use **readxl** to read Excel workbooks (.xls or .xlsx files).

We demonstrate with an Excel workbook containing the stoat data. The trap locations and the detection data are in separate sheets. A capthist object is then formed in one call to `read.capthist`:

```
xlsname <- system.file("extdata", "stoat.xlsx", package = "secr")
CH <- read.capthist (xlsname, sheet = c("stoatcapt", "stoattrap"), skip = 1,
                     detector = "proximity")
```

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

```
summary(CH)
```

```
## Object class       capthist
## Detector type      proximity
## Detector number    94
## Average spacing    250 m
## x-range            -1500 1500 m
## y-range            -1500 1500 m
##
## Counts by occasion
##                     1  2  3  4  5  6  7 Total
## n                   3  4  8  3  3  7  2    30
## u                   3  3  8  2  0  3  1    20
## f                  12  6  2  0  0  0  0    20
## M(t+1)              3  6 14 16 16 19 20    20
## losses              0  0  0  0  0  0  0     0
## detections          3  4  8  3  3  7  2    30
## detectors visited   3  4  8  3  3  7  2    30
## detectors used     94 94 94 94 94 94 94   658
```

Note that in this case –

1. **readxl** must have been installed, but it does not need to be loaded with `library`.
2. If the 'trapfile' argument is omitted, as in the example, then 'captfile' is assumed to be a workbook with separate worksheets for the captures and the detector layout.
3. The 'sheet' and 'skip' arguments are passed to `read_excel`. They may be vectors of length 1 (same for captures and detector layout) or length 2 (first captures, then detector layout). Sheets may be specified by number or by name.

4. You should explicitly skip header lines except for the column headings (the comment character "#" is not recognised).
5. The columns of coordinates should be headed "x" and "y". If these names are not found then the second and third columns will be used, with a warning.
6. There is no provision yet for input from Excel of multi-session data with session-specific trap layouts.

Other detector types

Count detectors

The 'proximity' detector type allows at most one detection of each individual at a particular detector on any occasion. Detectors that allow repeat detections are called 'count' detectors in **secr**. Binary proximity detectors are a special case of count proximity detectors in which the count always has a Bernoulli distribution. Non-binary counts can result from devices such as automatic cameras, or from collapsing data collected over many occasions (Efford et al. 2009).

Count data are input by repeating each line in the capture data the required number of times. (Yes, it would have been more elegant to code the frequency, but this detector type was an afterthought.) See `?make.capthist` for an example that automatically replicates rows of a capture dataframe according to a frequency vector f (f could be a column in the capture dataframe).

Signal detectors

Signal strength detectors are described in the document secr-sound.pdf following Efford et al. (2009). Here we just note that signal strength data may be input with `read.capthist` using a minor extension of the DENSITY format: the signal strength for each detection is appended as the fifth ('fmt = trapID') or sixth ('fmt = XY') value in each row of the capture data file. There will usually be only one sampling 'occasion' as sounds are ephemeral. The threshold below which signals were classified as 'not detected' must be provided in the 'cutval' argument. Detections with signal strength coded as less than 'cutval' are discarded.

```
write.capthist(signalCH, "temp")  ## export data for demo
tempCH <- read.capthist("tempcapt.txt", "temptrap.txt", detector = "signal", cutval = 52.5)

## No errors found :-)
```

Polygon and transect detectors

'Detectors' are usually modelled as if they exist at a point, and each row of the 'trapfile' for `read.capthist` gives the x-y coordinates for one detector, as we have seen. However, sometimes detections are made across an area , as when an area is searched for faecal samples that are subsequently identified to individual by microsatellite DNA analysis. Then the observations comprise both detection or nondetection of each individual on each occasion, and the precise x-y coordinates at which each cue (e.g., faecal deposit) was found.

The 'polygon' detector type handles this sort of data. The area searched is assumed to comprise one or more polygons. To simplify the analysis some constraints are imposed on the shape of polygons: they should be convex, at least in an east-west direction (i.e. any transect parallel to the y-axis should cross the boundary at no more than 2 points) and cannot contain 'holes'. See secr-polygondetectors.pdf for more.

Despite the considerable differences between 'polygon' and other detectors, input is pretty much as we have already described. Use `read.capthist` with the 'XY' format:

```
read.capthist("captXY.txt", "perimeter.txt", fmt = "XY", detector = "polygon")
```

The detector file (in this case 'perimeter.txt') has three columns as usual, but rows correspond to vertices of the polygon(s) bounding the search area. The first column is used as a factor to distinguish the polygons ('polyID').

```
# polyID X Y
1 576407 13915205
1 576978 13915122
1 576866 13914572
1 576256 13914661
2 575500 13915038
2 575857 13915210
2 576093 13914833
2 575905 13914438
2 575509 13914588
```

If the input polygons are not closed (as here) then the first vertex of each is repeated in the resulting 'traps' object to ensure closure.

The fourth and fifth columns of the capture file (in this case 'captXY.txt') give the x- and y- coordinates of each detection. These are matched automatically to the polygons defined in the detector file. Detections with x-y coordinates outside any polygon are rejected.

Polygons may also be input with `read.traps`. Here is an example in which the preceding polygons are used to simulate some detections (we assume the polygon data have been copied to the clipboard):

```
temppoly <- read.traps(file = "clipboard", detector = "polygon")
tempcapt <- sim.capthist(temppoly, popn = list(D = 1, buffer = 1000), detectpar =
                           list(g0 = 0.5, sigma = 250))
plot(tempcapt, label = TRUE, tracks = TRUE, title = "Simulated detections within polygons")
```

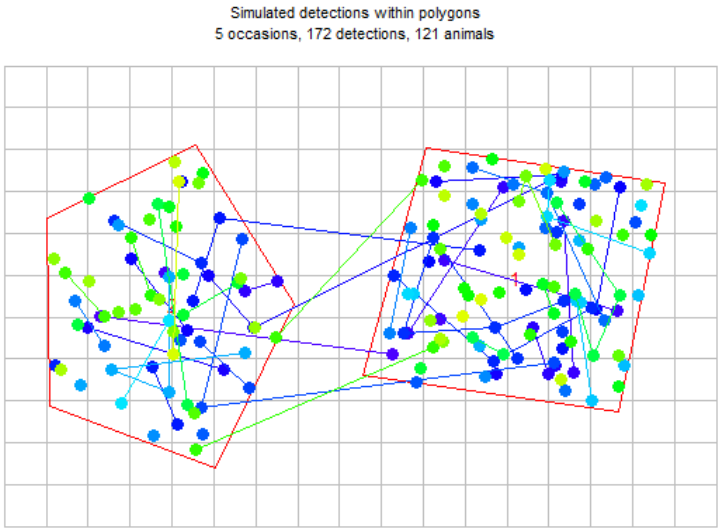

Figure 1: polygon simulations

See secr-polygondetectors.pdf for more on polygon and transect detectors.

Usage and covariates apply to the polygon or transect as a whole rather than to each vertex. Usage codes and covariates are appended to the end of the line, just as for point detectors (traps etc.). The usage and covariates for each polygon or transect are taken from its first vertex. Although the end-of-line strings of other vertices are not used, they cannot be blank and should use the same spacing as the first vertex.

Here is a polygon file that defines both usage and a categorical covariate. It would also work to repeat the usage and covariate for each vertex - this is just a little more readable.

```
# polyID X Y usage / habitat
1 576407 13915205 11000 / A
1 576978 13915122 - / -
1 576866 13914572 - / -
1 576256 13914661 - / -
2 575500 13915038 00111 / B
2 575857 13915210 - / -
2 576093 13914833 - / -
2 575905 13914438 - / -
2 575509 13914588 - / -
```

Detections along a transect have a similar structure to detections within a polygon, and input follows the same format. Locations are input as x-y coordinates for the position on the transect line from which each detection was made, not as distances along the transect.

Telemetry

NOTE: This section applies to secr 3.0 and later - previous versions handled telemetry differently.

The 'telemetry' detector type is used for animal locations from radiotelemetry. Even if the ultimate goal is to analyse telemetry data jointly with capture–recapture data, the first step is to create separate capthist objects for the telemetry and capture–recapture components. This section covers the input of standalone telemetry data; secr-telemetry.pdf shows how to combine telemetry and capture–recapture datasets with the function `addTelemetry`.

Telemetry observations ('fixes') are formatted in the standard 'XY' format. The input text file comprises at least five columns (session, animalID, occasion, x, y). 'Occasion' is largely redundant, and all fixes may be associated with occasion 1. It is desirable to keep fixes in the correct temporal order within each animal, but this is not used for modelling. The function `read.telemetry` is a version of `read.capthist` streamlined for telemetry data (detector locations are not needed).

See secr-telemetry.pdf for an example. For telemetry-calibrated cue counts (telemetry type 'marking'), in which the telemetered animals are the only ones identified at the detectors, the function `make.marking` assembles the capthist object in one step from the detections, fixes and sighting counts.

Mark–resight

Spatial mark–resight data combine detection histories of marked animals, input as described above, and counts of sighted but unmarked or unidentified animals. The latter are input separately as described in secr-markresight.pdf, using the `addSightings` function (for telemetry-calibrated cue counts `make.marking` takes the sighting counts directly). Sighting-only data may include all-zero detection histories of previously marked animals known to be alive but not seen; these are coded with occasion 0 (zero).

Troubleshooting

A message like

```
Error in scan(file, what, nmax, sep, dec, quote, skip, nlines,     na.strings,  :  line 1
did not have 6 elements
```

indicates unequal line lengths in one of the input files, possibly just one or two stray lines with an extra value. You can use `count.fields('filename.txt')` to track them down, replacing 'filename.txt' with your own filename.

Further notes

'Filters' are used in DENSITY to select and reconfigure data. Their function is taken over in **secr** by the methods `subset` and `reduce` for capthist objects.

`subset.capthist` allows the user to select a subset of individuals, occasions, detectors or sessions from a capthist object. For example:

```
summary(subset(stoatCH, traps = 1:47, occasions = 1:5))
```

```
## Warning in subset.capthist(stoatCH, traps = 1:47, occasions = 1:5): no detections on occasion(s) 5
```

```
## Object class       capthist
## Detector type      proximity
## Detector number    47
## Average spacing    250 m
## x-range            -1500 0 m
## y-range            -1500 1500 m
##
## Counts by occasion
##                    1  2  3  4  5 Total
## n                  2  4  1  1  0     8
## u                  2  3  1  0  0     6
## f                  4  2  0  0  0     6
## M(t+1)             2  5  6  6  6     6
## losses             0  0  0  0  0     0
## detections         2  4  1  1  0     8
## detectors visited  2  4  1  1  0     8
## detectors used    47 47 47 47 47   235
```

`reduce.capthist` allows occasions to be combined (or dropped), and certain changes of detector type. For example, 'count' data may be collapsed to binary 'proximity' data, or 'signal' data converted to 'proximity' data.

`addCovariates` is used to add spatial covariate information to a traps (detector) or mask object from another spatial data source.

The function `read.DA` is used to create a capthist object from polygon detection data in an R list, structured as input for the Bayesian analysis of Royle and Young (2008), using data augmentation.

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. (2012) *DENSITY 5.0: software for spatially explicit capture–recapture*. University of Otago, Dunedin, New Zealand https://www.otago.ac.nz/density/.

Efford, M. G., Dawson, D. K. and Borchers, D. L. (2009) Population density estimated from locations of individuals on a passive detector array. *Ecology* **90**, 2676–2682.

Miller, C. R., Joyce, P. and Waits, L. P. (2005) A new method for estimating the size of small populations from genetic mark–recapture data. *Molecular Ecology* **14**, 1991–2005.

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* **62**.

Pollock, K. H. (1982) A capture-recapture design robust to unequal probability of capture. *Journal of Wildlife Management* **46**, 752–757.

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

Wickham, H. and Bryan, J. (2017). **readxl**: Read Excel Files. R package version 1.0.0. https://CRAN.R-project.org/package=readxl

Appendix: Glossary

Covariate

Auxiliary data used in a model of detection probability. Covariates may be associated with detectors, individuals, sessions or occasions. Spatial covariates of density are a separate matter – see `?mask`. Individual covariates[2] are stored in a dataframe (one row per animal) that is an attribute of a capthist object. Detector covariates are stored in a dataframe (one row per detector) that is an attribute of a traps object (remembering that a capthist object always includes a traps object). Session and occasion (=time) covariates are not stored with the data; they are provided as the arguments 'sessioncov' and 'timecov' of function `secr.fit`.

Detector

A device used to detect the presence of an animal. Often used interchangeably with 'trap' but it is helpful to distinguish true traps, which always detain the animal until it is released, from other detectors such as hair snags and cameras that leave the animal free to roam. A detector in SECR has a known physical location, usually a point defined by its x-y coordinates.

Identifier

Label used to distinguish detectors, animals or sessions.

Occasion

In conventional capture–recapture, 'occasion' refers to a discrete sampling event (e.g., Otis et al. (1978) and program CAPTURE). A typical 'occasion' is a daily trap visit, but the time interval represented by an 'occasion' varies widely between studies. Although trapped samples accumulate over an interval (e.g., the preceding day), for analysis they are treated as instantaneous. Occasions are numbered 1, 2, 3, etc. Closed population analyses usually require two or more occasions (see Miller et al. 2005 for an exception).

SECR follows conventional capture–recapture in assuming discrete sampling events (occasions). However, SECR takes a closer interest in the sampling process, and each discrete sample is modelled as the outcome of processes operating through the interval between trap visits. In particular, a model of competing risks in continuous time is used for the probability of capture in multi-catch traps (Borchers & Efford 2008).

Proximity and count detectors allow multiple occurrences of an animal to be recorded in each sampling interval. Analysis is then possible with data from a single occasion. For consistency we retain the term 'occasion', although such a sample is clearly not instantaneous.

SECR

Spatially explicit capture–recapture, an inclusive term for capture–recapture methods that model detection probability as function of distance from unobserved 'home-range' centres (e.g., Borchers and Efford 2008). **secr** refers to the R package.

[2]Individual covariates may be used directly only when a model is fitted by maximizing the conditional likelihood, but they are used to define groups for the full likelihood case.

Session

A session is a set of occasions over which a population is considered closed to gains and losses. Each 'primary session' in the 'robust' design of Pollock (1982) is treated as a session in **secr**. **secr** also uses 'session' for independent subsets of the capture data distinguished by characteristics other than sampling time. For example, two grids trapped simultaneously could be analysed as distinct 'sessions' if they were far enough apart that there was little chance of the same animal being caught on both grids. Equally, males and females could be treated as 'sessions'. For many purposes, 'sessions' are functionally equivalent to 'groups'; sessions are (almost) set in concrete when the data are entered whereas groups may be defined on the fly (see `?secr.fit`).