Package: secr (via r-universe)

October 19, 2026

Type Package

Title Spatially Explicit Capture-Recapture

Version 5.5.2

Date 2026-10-19

Description Functions to estimate the density and size of a spatially distributed animal population sampled with an array of passive detectors, such as traps, or by searching polygons or transects. Models incorporating distance-dependent detection are fitted by maximizing the likelihood. Tools are included for data manipulation and model selection.

Depends R (>= 3.5.0), methods

Imports secrfunc (>= 1.1.2), abind, graphics, grDevices, MASS, mgcv, mvtnorm, nlme, parallel, raster (>= 3.5-15), Rcpp (>= 0.12.14), RcppParallel (>= 5.1.1), sf, stats, stringr, terra (>= 1.5-12), tools, utils

Suggests gdistance, igraph, knitr, readxl, rmarkdown, sp, spatstat (>= 3.3-3), spatstat.geom, spatstat.random, spcosa, spsurvey (>= 5.3.0), testthat

LinkingTo BH, Rcpp, RcppParallel

VignetteBuilder knitr

License GPL (>= 2)

LazyData yes

KeepSource yes

LazyDataCompression xz

SystemRequirements GNU make

URL https://www.otago.ac.nz/density/, https://github.com/MurrayEfford/secr/

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://murrayefford.r-universe.dev

Date/Publication 2026-10-19 09:08:28 UTC

RemoteUrl https://github.com/MurrayEfford/secr

RemoteRef HEAD

RemoteSha b9cc2bc0f473e6ecc9470c7aa6f4fc12b082ab61

Contents

secr-package . . . 6
addCovariates . . . 8
addSightings . . . 10
addTelemetry . . . 12
AIC.secr . . . 16
AICcompatible . . . 20
as.data.frame . . . 21
as.mask . . . 22
as.popn . . . 23
autoini . . . 24
binCovariate . . . 26
blackbear . . . 27
BUGS . . . 29
capthist . . . 31
capthist.parts . . . 33
chat . . . 34
circular . . . 36
clone . . . 39
closedN . . . 40
closure.test . . . 43
cluster . . . 44
coef.secr . . . 45
collate . . . 46
confint.secr . . . 48
contour . . . 49
covariates . . . 52
CV . . . 53
D.designdata . . . 55
deermouse . . . 56
deleteMaskPoints . . . 57
derived . . . 59
derivedDsurface . . . 62
details . . . 64
detectfn . . . 69
detector . . . 71
deviance . . . 72
discretize . . . 74
distancetotrap . . . 76
Dsurface . . . 77
ellipse.secr . . . 78

empirical.varD . . . 79
esaPlot . . . 84
expected.n . . . 87
extractMoves . . . 89
FAQ . . . 90
fillNA . . . 92
Fletcher.chat . . . 93
fxi . . . 95
fxTotal . . . 98
gridCells . . . 100
hcov . . . 101
head . . . 104
homerange . . . 105
hornedlizard . . . 109
housemouse . . . 111
Internal . . . 113
intervals . . . 114
join . . . 115
kfn . . . 117
list.secr.fit . . . 119
LLsurface . . . 120
logit . . . 122
logmultinom . . . 123
LR.test . . . 124
make.capthist . . . 125
make.lacework . . . 128
make.mask . . . 129
make.spcosa . . . 132
make.systematic . . . 134
make.traps . . . 136
make.tri . . . 139
makeStart . . . 141
mask . . . 142
mask.check . . . 143
MCgof . . . 146
modelAverage . . . 150
ms . . . 152
newdata . . . 153
nk . . . 154
nontarget . . . 157
occasionKey . . . 158
ovenbird . . . 159
ovensong . . . 162
OVpossum . . . 164
Parallel . . . 166
pdot . . . 169
PG . . . 171
plot.capthist . . . 173

plot.mask . . . 177
plot.MCgof . . . 180
plot.popn . . . 182
plot.secr . . . 183
plot.traps . . . 185
plotMaskEdge . . . 187
pmixProfileLL . . . 188
pointsInPolygon . . . 189
polyarea . . . 190
popn . . . 191
possum . . . 192
predict.secr . . . 194
predictDsurface . . . 196
print.capthist . . . 199
print.secr . . . 200
print.traps . . . 202
randomHabitat . . . 202
raster . . . 205
rbind.capthist . . . 207
rbind.popn . . . 210
rbind.traps . . . 211
read.capthist . . . 212
read.mask . . . 215
read.telemetry . . . 216
read.traps . . . 217
rectangularMask . . . 219
reduce . . . 220
reduce.capthist . . . 221
region.N . . . 225
RMarkInput . . . 228
RSE . . . 230
Rsurface . . . 231
score.test . . . 233
secr-defunct . . . 235
secr-deprecated . . . 237
secr-version5 . . . 238
secr.design.MS . . . 240
secr.fit . . . 242
secr.refit . . . 248
secr.test . . . 250
secrdemo . . . 252
secrRNG . . . 254
secrtest . . . 256
session . . . 258
setNumThreads . . . 259
shareFactorLevels . . . 260
sighting . . . 261
signal . . . 262

signalmatrix . . . 263
sim.capthist . . . 264
sim.popn . . . 269
sim.secr . . . 275
simOU.capthist . . . 278
skink . . . 280
smooths . . . 282
snip . . . 284
sort.capthist . . . 286
spacing . . . 287
speed . . . 288
stoatDNA . . . 291
strip.legend . . . 293
subset.capthist . . . 295
subset.mask . . . 297
subset.popn . . . 299
subset.traps . . . 300
suggest.buffer . . . 301
summary.capthist . . . 304
summary.mask . . . 306
summary.popn . . . 307
summary.traps . . . 308
timevaryingcov . . . 309
transformations . . . 311
trap.builder . . . 313
traps . . . 318
traps.info . . . 319
Trend . . . 321
trim . . . 322
Troubleshooting . . . 323
turnover . . . 324
updateCH . . . 327
usage . . . 328
usagePlot . . . 330
userdist . . . 332
utility . . . 334
vcov.secr . . . 336
verify . . . 337
write.captures . . . 339
writeGPS . . . 340

Index **343**

secr-package *Spatially Explicit Capture–Recapture Models*

Description

Functions to estimate the density and size of a spatially distributed animal population sampled with an array of passive detectors, such as traps, or by searching polygons or transects.

Details

Package:	secr
Type:	Package
Version:	5.5.2
Date:	2026-10-19
License:	GNU General Public License Version 2 or later

Spatially explicit capture–recapture is a set of methods for studying marked animals distributed in space. Data comprise the locations of detectors (traps, searched areas, etc. described in an object of class 'traps'), and the detection histories of individually marked animals. Individual histories are stored in an object of class 'capthist' that includes the relevant 'traps' object.

Models for population density (animals per hectare) and detection are defined in **secr** using symbolic formula notation. Density models may include spatial or temporal trend. Possible predictors for detection probability include both pre-defined variables (t, b, etc.) corresponding to 'time', 'behaviour' and other effects), and user-defined covariates of several kinds. Habitat is distinguished from nonhabitat with an object of class 'mask'.

Models are fitted in **secr** by maximizing either the full likelihood or the likelihood conditional on the number of individuals observed (n). Conditional likelihood models allow continuous individual covariates for detection. A model fitted with `secr.fit` is an object of class `secr`. Generic methods (plot, print, summary, etc.) are provided for each object class.

A link at the bottom of each help page takes you to the help index. Some vignettes complement the help pages:

secr-overview.pdf	general introduction
secr-datainput.pdf	data formats and input functions
secr-version4.pdf	changes in secr 4.0
secr-manual.pdf	consolidated help pages
secr-sound.pdf	analysing data from microphone arrays

This reference includes material previously covered in separate vignettes:

Efford, M. G. (2025) The SECR book. A handbook of spatially explicit capture–recapture methods. Version 1.0. https://murrayefford.github.io/SECRbook/ and Zenodo doi:10.5281/zenodo.15109937.

The datasets `captdata` and `ovenbird` include examples of fitted models. For models fitted to other datasets see secr-version4.pdf Appendix 2.

Add-on packages extend the capability of **secr** and are documented separately. **secrlinear** enables the estimation of linear density (e.g., animals per km) for populations in linear habitats such as stream networks (secrlinear-vignette.pdf). **secrdesign** enables the assessment of alternative study designs by Monte Carlo simulation; scenarios may differ in detector (trap) layout, sampling intensity, and other characteristics (secrdesign-vignette.pdf). **ipsecr** fits some awkward models (e.g., for single-catch traps) by simulation and inverse prediction (ipsecr-vignette.pdf). **openCR** fits open population models, both non-spatial and spatial (openCR-vignette.pdf).

The analyses in **secr** extend those available in the software Density (see www.otago.ac.nz/density/ for the most recent version of Density). Help is available on the 'DENSITY | secr' forum at www.phidot.org and the Google group secrgroup. Feedback on the software is also welcome, including suggestions for additional documentation or new features consistent with the overall design.

Author(s)

Murray Efford <murray.efford@otago.ac.nz>

References

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

Borchers, D. L. and Fewster, R. M. (2016) Spatial capture–recapture models. *Statistical Science* **31**, 219–232.

Efford, M. G. (2004) Density estimation in live-trapping studies. *Oikos* **106**, 598–610.

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

Efford, M. G., Borchers D. L. and Mowat, G. (2013) Varying effort in capture–recapture studies. *Methods in Ecology and Evolution* **4**, 629–636.

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.

Efford, M. G., Dawson, D. K. and Robbins C. S. (2004) DENSITY: software for analysing capture-recapture data from passive detector arrays. *Animal Biodiversity and Conservation* **27**, 217–228.

Efford, M. G. and Fewster, R. M. (2013) Estimating population size by spatially explicit capture–recapture. *Oikos* **122**, 918–928.

Efford, M. G. and Hunter, C. M. (2017) Spatial capture–mark–resight estimation of animal population density. *Biometrics* **74**, 411–420.

Efford, M. G. and Mowat, G. (2014) Compensatory heterogeneity in capture–recapture data.*Ecology* **95**, 1341–1348.

Royle, J. A., Chandler, R. B., Sollmann, R. and Gardner, B. (2014) *Spatial capture–recapture.* Academic Press.

Royle, J. A. and Gardner, B. (2011) Hierarchical spatial capture–recapture models for estimating density from trapping arrays. In: A.F. O'Connell, J.D. Nichols and K.U. Karanth (eds) *Camera Traps in Animal Ecology: Methods and Analyses*. Springer, Tokyo. Pp. 163–190.

See Also

read.capthist, secr.fit, traps, capthist, mask

Examples

```
## Not run:

## generate some data & plot
detectors  <- make.grid (nx = 10, ny = 10, spacing = 20,
    detector = "multi")
plot(detectors, label = TRUE, border = 0, gridspace = 20)
detections <- sim.capthist (detectors, noccasions = 5,
    popn = list(D = 5, buffer = 100),
    detectpar = list(g0 = 0.2, sigma = 25))
session(detections) <- "Simulated data"
plot(detections, border = 20, tracks = TRUE, varycol = TRUE)

## generate habitat mask
mask <- make.mask (detectors, buffer = 100, nx = 48)

## fit model and display results
secr.model <- secr.fit (detections, model = g0~b, mask = mask)
secr.model


## End(Not run)
```

---

addCovariates *Add Covariates to Mask or Traps*

Description

Tools to construct spatial covariates for existing mask or traps objects from a spatial data source.

Usage

```
addCovariates(object, spatialdata, columns = NULL, strict = FALSE, replace = FALSE)
```

Arguments

`object`	mask, traps or popn object
`spatialdata`	spatial data source (see Details)
`columns`	character vector naming columns to include (all by default)
`strict`	logical; if TRUE a check is performed for points in `object` that lie outside `spatialdata` (mask data sources only)
`replace`	logical; if TRUE then covariates with duplicate names are replaced; otherwise a new column is added

Details

The goal is to obtain the value(s) of one or more spatial covariates for each point (i.e. row) in `object`. The procedure depends on the data source `spatialdata`, which may be either a spatial coverage (raster or polygon) or an object with covariate values at points (another mask or traps object). In the first case, an overlay operation is performed to find the pixel or polygon matching each point. In the second case, a search is conducted for the closest point in `spatialdata`.

If `spatialdata` is a character value then it is interpreted as the name of a polygon shape file (excluding '.shp').

If `spatialdata` is a SpatialPolygonsDataFrame, SpatialGridDataFrame or 'sf' object from **sf** then it will be used in an overlay operation as described.

If package **terra** has been installed then `spatialdata` may also be a RasterLayer from package **raster** or SpatRaster from **terra**. If provided `counts` should be a single name that will be used for the values (otherwise 'raster' will be used).

If `spatialdata` is a `mask` or `traps` object then it is searched for the closest point to each point in `object`, and covariates are drawn from the corresponding rows in `covariates(spatialdata)`. By default (`strict = FALSE`), values are returned even when the points lie outside any cell of the mask.

If a covariate has a few missing values these may be filled cheaply using `fillNA`.

Value

An object of the same class as `object` with new or augmented `covariates` attribute. Column names and types are derived from the input.

Warning

Use of a SpatialGridDataFrame for `spatialdata` is untested.

See Also

`fillNA`, `make.mask`, `read.mask`, `read.traps`

Examples

```
## In the Lake Station skink study (see ?skink), habitat covariates were
## measured only at trap sites. Here we extrapolate to a mask, taking
## values for each mask point from the nearest trap.
```

```
LSmask <- make.mask(LStraps, buffer = 30, type = "trapbuffer")
tempmask <- addCovariates(LSmask, LStraps)
## show first few lines
head(covariates(tempmask))

## Missing covariates at a fraction of mask locations may also be filled with fillNA()
```

---

addSightings *Mark-resight Data*

Description

Add sighting data on unmarked individuals and/or unidentified marked individuals to an existing capthist object.

Usage

```
addSightings(capthist, unmarked = NULL, nonID = NULL, uncertain = NULL, verify = TRUE,
    ...)
```

Arguments

capthist	secr capthist object
unmarked	matrix or list of matrices of sightings of unmarked animals, Tu, or file name (see Details)
nonID	matrix or list of matrices of unidentified sightings of marked animals, Tm, or file name (see Details)
uncertain	matrix or list of matrices of uncertain sightings, Tn, or file name (see Details)
verify	logical; if TRUE then the resulting capthist object is checked with verify
...	other arguments passed to read.table

Details

The capthist object for mark-resight analysis comprises distinct marking and sighting occasions, defined in the markocc attribute of traps(capthist). Add this attribute to traps(capthist) with markocc before using 'addSightings'. See also read.traps and read.capthist.

Mark-resight data may be binary (detector type 'proximity') or counts (detector types 'count', 'polygon' or 'transect'). The detector type is an attribute of traps(capthist). Values in unmarked and nonID should be whole numbers, and may be greater than 1 even for binary proximity detectors because multiple animals may be detected simultaneously at one place.

Arguments unmarked, nonID, uncertain provide data for attributes 'Tu', 'Tm', 'Tn' respectively. They may take several forms

- a single integer, the sum of all counts*
- a matrix of the count on each occasion at each detector (dimensions K x S, where K is the number of detectors and S is the total number of occasions). Columns corresponding to marking occasions should be all-zero.
- for multi-session data, a list with components as above
- a character value with the name of a text file containing the data; the file will be read with `read.table`. The ... argument allows some control over how the file is read. The data format comprises at least S+1 columns. The first is a session identifier used to split the file when the data span multiple sessions; it should be constant for a single-session capthist. The remaining S columns contain the counts for occasions 1:S, one row per detector. Further columns may be present; they are ignored at present.

* although this is convenient, the full matrix of counts provides more flexibility (e.g., when you wish to subset by occasion), and enables modelling of variation across detectors and occasions.

Value

A capthist object with the same structure as the input, but with new sighting-related attributes Tu (sightings of unmarked animals) and/or Tm (unidentified sightings of marked animals). Input values, including NULL, overwrite existing values.

Warning

** Mark-resight data formats and models are experimental and subject to change **

See Also

`markocc`, `read.capthist`, `read.traps`, `sim.resight`, `Tm`, `Tu`, `Tn`, secr-markresight.pdf

Examples

```
## construct capthist object MRCH from text files provided in
## 'extdata' folder, assigning attribute 'markocc' and add unmarked
## and marked sightings from respective textfiles

datadir <- system.file("extdata", package = "secr")
captfile <- paste0(datadir, '/MRCHcapt.txt')
trapfile <- paste0(datadir, '/MRCHtrap.txt')
Tufile <- paste0(datadir, '/Tu.txt')
Tmfile <- paste0(datadir, '/Tm.txt')

MRCH <- read.capthist(captfile, trapfile, detector = c("multi",
    rep("proximity",4)), markocc = c(1,0,0,0,0))
MRCH1 <- addSightings(MRCH, Tufile, Tmfile)

## alternatively (ignoring marked, not ID sightings)

MRCH <- read.capthist(captfile, trapfile, detector = c("multi",
    rep("proximity",4)), markocc = c(1,0,0,0,0))
Tu <- read.table(Tufile)[,-1]  # drop session column
```

```
MRCH2 <- addSightings(MRCH, unmarked = Tu)
summary(MRCH2)
```

---

addTelemetry *Combine Telemetry and Detection Data*

Description

Animal locations determined by radiotelemetry can be used to augment capture–recapture data. The procedure in **secr** is first to form a capthist object containing the telemetry data and then to combine this with true capture–recapture data (e.g. detections from hair-snag DNA) in another capthist object. `secr.fit` automatically detects the telemetry data in the new object.

Data from an arbitrary sample of telemetered animals combined with cue counts for unmarked animals are a special case that is discussed below ('Telemetry-calibrated cue counts').

Usage

```
addTelemetry (detectionCH, telemetryCH, type = c('concurrent','dependent',
   'independent','marking'), collapsetelemetry = TRUE, verify = TRUE,
   prefix = 'T')

xy2CH (CH, inflation = 1e-08)

telemetrytype (object) <- value

telemetrytype (object, ...)
```

Arguments

`detectionCH`	single-session capthist object, detector type 'single', 'multi', 'proximity' or 'count'
`telemetryCH`	single-session capthist object, detector type 'telemetryonly'
`type`	character (see Details)
`collapsetelemetry`	logical; if TRUE then telemetry occasions are collapsed to one
`verify`	logical; if TRUE then `verify.capthist` is called on the output
`prefix`	character; pre-pended to ID of telemetry-only animals
`CH`	capthist object with telemetryxy attribute
`inflation`	numeric tolerance for polygon
`object`	secr traps object
`value`	character telemetry type replacement value
`...`	other arguments

Details

It is assumed that a number of animals have been radiotagged, and their telemetry data (xy-coordinates) have been input to `telemetryCH`, perhaps using `read.capthist` with `detector = "telemetryonly"` and `fmt = "XY"`, or with `read.telemetry`.

A new capthist object is built comprising all the detection histories in `detectionCH`, plus empty (all-zero) histories for every telemetered animal not in `detectionCH`. Telemetry is associated with new sampling occasions and a new detector (nominally at the same point as the first in `detectionCH`). The number of telemetry fixes of each animal is recorded in the relevant cell of the new capthist object (CH[i, s, K+1] for animal i and occasion s if there were K detectors in detectionCH).

The new sampling occasion(s) are assigned the detector type 'telemetry' in the traps attribute of the output capthist object, and the traps attribute `telemetrytype` is set to the value provided. The telemetry type may be "independent" (no matching of individuals in captured and telemetered samples), "dependent" (telemetered animals are a subset of captured animals), "concurrent" (histories may be capture-only, telemetry-only or both capture and telemetry), or "marking" (one detection history for each telemetered animal, and no others).

The telemetry locations are carried over from telemetryCH as attribute 'xylist' (each component of xylist holds the coordinates of one animal; use `telemetryxy` to extract).

The default behaviour of 'addTelemetry' is to automatically collapse all telemetry occasions into one. This is computationally more efficient than the alternative, but closes off some possible models.

`xy2CH` partly reverses `addTelemetry`: the location information in the telemetryxy attribute is converted back to a capthist with detector type 'telemetry'.

Value

A single-session capthist object with the same detector type as `detectionCH`, but possibly with empty rows and an 'telemetryxy' attribute.

Telemetry-calibrated cue counts

Telemetry type 'marking' is used for telemetry-calibrated cue counts: cameras (or other 'count' detectors) detect *all* animals, but identify only animals wearing radio-collars (the 'marked' animals). The collared animals also provide telemetry fixes. The data are assembled as follows.

- `detectionCH` holds the identified sightings of collared animals on the sighting occasions, one history for each collared animal. Animals never sighted have all-zero histories; they are added by `addTelemetry` if they are present in `telemetryCH`. Detector type should be 'count'.
- `telemetryCH` holds the telemetry fixes. They are collapsed to a single telemetry occasion (the last), recorded at a notional extra detector.
- The traps attribute `markocc` (see `markocc`) is set by `addTelemetry` to 0 for each sighting occasion and 1 for the telemetry occasion, e.g. `c(0,0,0,0,0,1)`, unless `markocc` has already been set for `detectionCH`.
- The counts of sightings of animals that were *not* identified as collared are added with `addSightings`: argument `unmarked` (attribute Tu, a K x S matrix of counts) for animals without collars and, optionally, `nonID` (attribute Tm) for collared animals that were seen but not identified.

Cues of collared animals recorded in Tm or as identified sightings are not counted again as unmarked cues; this is the principal difference from a model that treats the sightings of unmarked animals independently of those of marked animals. The probability that a sighting of a collared animal is identified (parameter pID, 'q') is estimated only if Tm is present.

A collar may not function throughout the survey. Use marked(capthist) <- (see marked) to specify when each animal could be recognised as marked (default is all occasions). Collared animals that were never detected contribute information only through their fixes and exposure.

The model is fitted by secr.fit with detectfn = "HHN" (or 'HEX'). The activity centres of collared animals are integrated by summing over the mask or, usually faster, by Gauss-Hermite quadrature around the mean of the fixes (details = list(telemetryint = "GH"); see details). Density is estimated from the total sightings of unmarked animals after allowing for the collared animals' own sightings, so collars need not be deployed in proportion to density, but the collared animals should be representative of the population in their detectability (absolute density is biased low if collars favour animals that are individually more detectable). A density model such as D ~ x is possible if the mask has covariate x; coefficients are then estimated from the spatial pattern of unmarked sightings, and derived reports a cue-weighted density 'Dcw' (the mean of D(x) over the mask weighted by the expected number of cues from an animal centred at x) that is better determined than the mean density over the mask. A mask covariate named 'marking' gives a non-uniform prior for the activity centres of collared animals (default uniform).

secr.fit stops with an error for telemetry type 'marking' if any non-telemetry detector is not of type 'count'.

With CL = TRUE the likelihood is conditional on the collared animals: it uses their identified sightings, their telemetry and, if present, Tm. The unmarked sightings Tu contribute only through their spatial pattern across detectors, and only when there is a (relative) density model such as D ~ x; the density level is not estimated. The detection parameters (including pID) and the density coefficients are those of the full likelihood, and the CL fit is a quick way to obtain them. Density requires the full likelihood (CL = FALSE); derived, derivedDcoef, derivedDsurface and region.N are not available for a CL fit of this type.

Sightings are overdispersed relative to the Poisson (the unmarked total is a sum over animals of correlated cues). The scale factor 'chat' is computed analytically from the fitted model: fit once, then secr.refit(fit, details = list(chatmethod = "analytic")). Simulation of chat (details$nsim) is not available for this telemetry type. See details.

Finite mixture models (e.g. model = list(sigma ~ h2, lambda0 ~ h2)) are supported when the class of every collared animal is known: the classes are given by a categorical individual covariate named in hcov, and an error is reported if any collared animal has an unknown class. The classes of the unmarked animals are unknown and are modelled as a latent mixture with population proportions pmix. By default pmix is estimated from the class composition of the collared animals, which assumes that collaring was independent of class. If that is doubtful (for example, sex-biased collaring) fix pmix at an external value with fixed = list(pmix = p), where for two classes p is the proportion in the second class.

Note

Telemetry provides independent data on the location and presence of a sample of animals. These animals may be missed in the main sampling that gives rise to detectionCH i.e., they may have all-zero detection histories.

The 'telemetry' detector type is used for telemetry occasions in a combined dataset.

See Also

capthist, make.telemetry, read.telemetry, telemetryxy, telemetered, addSightings, marked, markocc

Examples

```
# Telemetry type 'marking': simulated cameras count all animals, but only
# collared animals are identified

set.seed(1)
tr  <- make.grid(nx = 9, ny = 9, spacing = 11, detector = "count")
msk <- make.mask(tr, buffer = 30, spacing = 5, type = "traprect")
pop <- sim.popn(D = 50, core = msk, buffer = 0)
collared <- 1:nrow(pop) %in% sample(nrow(pop), 15)
dp <- list(lambda0 = 1, sigma = 4)

# identified sightings of collared animals, and telemetry fixes
sightCH <- sim.capthist(tr, popn = subset(pop, collared), detectfn = "HHN",
    detectpar = dp, noccasions = 5, renumber = FALSE)
teCH <- sim.capthist(make.telemetry(), popn = subset(pop, collared),
    detectfn = "HHN", detectpar = dp, noccasions = 50, renumber = FALSE)

# counts at each detector and occasion of animals without collars (K x S matrix)
uCH <- sim.capthist(tr, popn = subset(pop, !collared), detectfn = "HHN",
    detectpar = dp, noccasions = 5, renumber = FALSE)
Tu <- t(apply(uCH, 2:3, sum))

ch <- addTelemetry(sightCH, teCH, type = "marking")
markocc(traps(ch))     # 5 sighting occasions + telemetry
ch <- addSightings(ch, unmarked = Tu)


fit <- secr.fit(ch, mask = msk, detectfn = "HHN", trace = FALSE,
    details = list(telemetryint = "GH"))
predict(fit)

# model-based overdispersion (chat) and cue-weighted density
fit2 <- secr.refit(fit, details = list(chatmethod = "analytic"), trace = FALSE)
predict(fit2)
derived(fit2)


## Not run:

# Generate some detection and telemetry data, combine them using
# addTelemetry, and perform analyses

# detectors
te <- make.telemetry()
tr <- make.grid(detector = "proximity")

# simulated population and 50% telemetry sample
```

```
totalpop <- sim.popn(tr, D = 20, buffer = 100)
tepop <- subset(totalpop, runif(nrow(totalpop)) < 0.5)

# simulated detection histories and telemetry
# the original animalID (renumber = FALSE) are needed for matching
trCH <- sim.capthist(tr,  popn = totalpop, renumber = FALSE, detectfn = "HHN")
teCH <- sim.capthist(te, popn = tepop, renumber=FALSE, detectfn = "HHN",
    detectpar = list(lambda0 = 3, sigma = 25))

combinedCH <- addTelemetry(trCH, teCH)

# summarise and display
summary(combinedCH)
plot(combinedCH, border = 150)
ncapt <- apply(combinedCH,1,sum)
points(totalpop[row.names(combinedCH)[ncapt==0],], pch = 1)
points(totalpop[row.names(combinedCH)[ncapt>0],], pch = 16)

# for later comparison of precision we must fix the habitat mask
mask <- make.mask(tr, buffer = 100)
fit.tr <- secr.fit(trCH, mask = mask, CL = TRUE, detectfn = "HHN")  ## trapping alone
fit.te <- secr.fit(teCH, mask = mask, CL = TRUE, start = log(20),   ## telemetry alone
    detectfn = "HHN")
fit2   <- secr.fit(combinedCH, mask = mask, CL = TRUE,              ## combined
    detectfn = "HHN")

# improved precision when focus on realised population
# (compare CVD)
derived(fit.tr, distribution = "binomial")
derived(fit2, distribution = "binomial")


# may also use CL = FALSE
secr.fit(combinedCH, CL = FALSE, detectfn = "HHN", trace = FALSE)

## End(Not run)
```

AIC.secr *Compare SECR Models*

Description

Terse report on the fit of one or more spatially explicit capture–recapture models. Models with smaller values of AIC (Akaike's Information Criterion) are preferred. Extraction ([) and logLik methods are included.

Usage

```
## S3 method for class 'secr'
```

```
AIC(object, ..., sort = TRUE, k = 2, dmax = 10, criterion = c("AIC","AICc"), chat = NULL)
## S3 method for class 'secrlist'
AIC(object, ..., sort = TRUE, k = 2, dmax = 10, criterion = c("AIC","AICc"), chat = NULL)
## S3 method for class 'secr'
logLik(object, ...)
secrlist(..., names = NULL)
## S3 method for class 'secrlist'
x[i]
```

Arguments

object	secr object output from the function secr.fit, or a list of such objects with class c("secrlist", "list")
...	other secr objects
sort	logical for whether rows should be sorted by ascending AIC
k	numeric, penalty per parameter to be used; always k = 2 in this method
dmax	numeric, maximum AIC difference for inclusion in confidence set
criterion	character, criterion to use for model comparison and weights
chat	numeric optional variance inflation factor for quasi-AIC
names	character vector of names (optional)
x	secrlist
i	indices

Details

Models to be compared must have been fitted to the same data and use the same likelihood method (full vs conditional). From version 4.1 a warning is issued if AICcompatible reveals a problem.

AIC is given by

$$\mathrm{AIC} = -2\log(L(\hat{\theta})) + 2K$$

where K is the number of "beta" parameters estimated.

AIC with small sample adjustment is given by

$$\mathrm{AIC}_c = -2\log(L(\hat{\theta})) + 2K + \frac{2K(K+1)}{n-K-1}.$$

The sample size n is the number of individuals observed at least once (i.e. the number of rows in capthist).

Model weights are calculated as

$$w_i = \frac{\exp(-\Delta_i/2),}{\sum \exp(-\Delta_i/2)}$$

where Δ refers to differences in AIC or AICc depending on the argument 'criterion' (see Notes).

Models for which delta > dmax are given a weight of zero and are excluded from the summation. Model weights may be used to form model-averaged estimates of real or beta parameters with modelAverage (see also Buckland et al. 1997, Burnham and Anderson 2002).

The argument k is included for consistency with the generic method AIC.

secrlist forms a list of fitted models (an object of class 'secrlist') from the fitted models in .... Arguments may include secrlists. If secr components are named the model names will be retained unless 'names' is specified. (see Examples).

If chat ($\hat{c}$) is provided then quasi-AIC values are computed (**secr** >= 4.6.0):

$$\mathrm{QAIC} = -2\log(L(\hat{\theta}))/\hat{c} + 2K.$$

Value

A data frame with one row per model. By default, rows are sorted by ascending 'criterion' (default AIC see Notes).

model	character string describing the fitted model
detectfn	shape of detection function fitted (halfnormal vs hazard-rate)
npar	number of parameters estimated
logLik	maximized log likelihood
AIC	Akaike's Information Criterion
AICc	AIC with small-sample adjustment of Hurvich & Tsai (1989)

And depending on criterion:

dAIC	difference between AIC of this model and the one with smallest AIC
AICwt	AIC model weight

or

dAICc	difference between AICc of this model and the one with smallest AICc
AICcwt	AICc model weight

logLik.secr returns an object of class 'logLik' that has attribute df (degrees of freedom = number of estimated parameters).

If the variance inflation factor 'chat' is provided then outputs AIC, AICc etc. are replaced by the corresponding quasi-AIC values labelled QAIC, QAICc etc.

Note

It is not be meaningful to compare models by AIC if they relate to different data (see AICcompatible).

Specifically:

- an 'secrlist' generated and saved to file by mask.check may be supplied as the object argument of AIC.secrlist, but the results are not informative
- models fitted by the conditional likelihood (CL = TRUE) and full likelihood (CL = FALSE) methods cannot be compared

- hybrid mixture models (using hcov argument of secr.fit) should not be compared with other models
- grouped models (using groups argument of secr.fit) should not be compared with other models
- multi-session models should not be compared with single-session models based on the same data.

A likelihood-ratio test (LR.test) is a more direct way to compare two models.

The issue of goodness-of-fit and possible adjustment of AIC for overdispersion has yet to be addressed (cf QAIC in MARK).

The user may select between AIC and AICc for comparing models. AICc is widely used, but AIC may be better for model averaging even when samples are small (Turek and Fletcher 2012; Fletcher 2019, p. 60). The default was changed to AIC in version 5.0.0).

References

Buckland S. T., Burnham K. P. and Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.

Burnham, K. P. and Anderson, D. R. (2002) *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Second edition. New York: Springer-Verlag.

Fletcher, D. (2019) Model averaging. SpringerBriefs in Statistics. Berlin: Springer-Verlag.

Hurvich, C. M. and Tsai, C. L. (1989) Regression and time series model selection in small samples. *Biometrika* **76**, 297–307.

Turek, D. and Fletcher, D. (2012) Model-averaged Wald confidence intervals. *Computational statistics and data analysis* **56**, 2809–2815.

See Also

AICcompatible, modelAverage, AIC, secr.fit, print.secr, score.test, LR.test, deviance.secr

Examples

```
## Compare two models fitted previously
## secrdemo.0 is a null model
## secrdemo.b has a learned trap response

AIC(secrdemo.0, secrdemo.b)

## Form secrlist and pass to AIC.secr
temp <- secrlist(null = secrdemo.0, learnedresponse = secrdemo.b)
AIC(temp)
```

AICcompatible *Model Compatibility*

Description

Determine whether models can be compared by AIC. Incompatibility may be due to difference in the data or the specifications of the groups, hcov or binomN arguments to secr.fit,

Usage

```
## S3 method for class 'secr'
AICcompatible(object, ...)
## S3 method for class 'secrlist'
AICcompatible(object, ...)
```

Arguments

object	secr object output from the function secr.fit, or a list of such objects with class c("secrlist", "list")
...	other secr objects

Details

The capthist objects are checked for strict identity with the function identical.

If the capthist objects are found to be different (AICcompatible data component FALSE), the difference may be in an attribute (traps, covariates) rather than the detection histories themselves. Investigate further with all.equal.

All elements in the output must be TRUE for valid AIC comparison or model averaging using AIC or AICc.

Value

Named logical vector with elements ‘data’, ‘CL’, ‘groups’, ‘hcov’ and ‘binomN’.

See Also

AIC.secr, modelAverage

Examples

```
AICcompatible(secrdemo.0, secrdemo.CL)

## Not run:

## A common application of AICcompatible() is to determine
## the compatibility of models fitted with and without the
```

```
## fastproximity option.

ovenCHp1 <- reduce(ovenCHp, by = 'all', outputdetector = 'count')
msk <- make.mask(traps(ovenCHp)[[1]], buffer = 300, nx = 25)
ob1 <- secr.fit(ovenCHp, mask = msk, details = list(fastproximity = TRUE))
ob2 <- secr.fit(ovenCHp1, mask = msk, details = list(fastproximity = FALSE))
ob3 <- secr.fit(ovenCHp1, mask = msk, details = list(fastproximity = FALSE), binomN = 1)
AICcompatible(ob1,ob2)
AICcompatible(ob1,ob3)


## End(Not run)
```

`as.data.frame` *Coerce capthist to Data Frame or Array*

Description

Method for generic `as.data.frame` function that partially reverses `make.capthist`.

Usage

```
## S3 method for class 'capthist'
as.data.frame(x, row.names = NULL, optional = FALSE, covariates = FALSE,
                                    fmt = c("trapID", "XY"), ...)
## S3 method for class 'traps'
as.data.frame(x, row.names = NULL, optional = FALSE, usage = FALSE,
                               covariates = FALSE, ...)
## S3 method for class 'capthist'
as.array(x, ...)
```

Arguments

`x`	`capthist` object
`row.names`	unused argument of generic function
`optional`	unused argument of generic function
`covariates`	logical or a character vector of covariates to export
`fmt`	character string for capture format
`usage`	logical; if TRUE then usage columns are appended if present
`...`	other arguments (not used)

Details

By default individual covariates are not exported. When exported they are repeated for each detection of an individual.

Value

A data frame or list of data frames (in the case of a multisession input).

For capthist objects –

The core columns are (Session, ID, Occasion, TrapID) or (Session, ID, Occasion, x, y), depending on the value of `fmt`. Additional columns for covariates and signal strength (detector 'signal') are appended to the right.

For traps objects –

The core columns are (x, y). Usage columns are named u1, u2, ..., uS where S is the number of occasions.

The `as.array` method for capthist objects returns an object with the same dimensions and dimnames but different class, or a list of such objects in the case of multisession input.

Examples

```
as.data.frame (captdata)
as.data.frame (traps(captdata))
```

`as.mask` *Coerce traps object to mask*

Description

This function is used primarily for plotting covariates, for which the plot.mask function has greater functionality than `plot.traps`. It also generates pretty maps of grid cells.

Usage

```
as.mask(x)
```

Arguments

`x`	an object of class 'traps'

Details

A mask derived by coercion with `as.mask` may behave unpredictably e.g., in `secr.fit`.

Value

If `x` is a single-session traps object –

an object of class c("mask", "data.frame")

If `x` is a multi-session traps object –

an object of class c("mask", "list"), for which each component is a single-session mask.

See Also

make.mask, plot.mask, mask, traps

Examples

```
plot(as.mask(traps(captdata)), dots = FALSE, meshcol = "black")
plot(traps(captdata), add = TRUE)
```

---

as.popn *Coerce ppp object to popn*

Description

This function converts a **spatstat** "ppp" object (Baddeley et al. 2015), making it easier to use the simulation capability of **spatstat** in **secr**.

Usage

```
as.popn(x)
```

Arguments

x	an object of class 'ppp'

Details

Not all attributes are carried over.

Value

An object of class c("popn", "data.frame") with attribute "boundingbox". The attribute "Lambda" (**spatstat** class "im") is also carried over if present (used for the intensity surface of LGCP simulations).

References

Baddeley, A., Rubak, E., and Turner, R. 2015. Spatial Point Patterns: Methodology and Applications with R. Chapman and Hall/CRC Press, London. ISBN 9781482210200, https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/p/book/9781482210200/.

See Also

sim.popn, popn

autoini *Initial Parameter Values for SECR*

Description

Find plausible initial parameter values for secr.fit. A simple SECR model is fitted by a fast ad hoc method.

Usage

```
autoini(capthist, mask, detectfn = 0, thin = 0.2, tol = 0.001,
    binomN = 1, adjustg0 = TRUE, adjustsigma = 1.2, ignoreusage = FALSE,
    ncores = NULL)
```

Arguments

capthist	capthist object
mask	mask object compatible with the detector layout in capthist
detectfn	integer code or character string for shape of detection function 0 = halfnormal
thin	proportion of points to retain in mask
tol	numeric absolute tolerance for numerical root finding
binomN	integer code for distribution of counts (see secr.fit)
adjustg0	logical for whether to adjust g0 for usage (effort) and binomN
adjustsigma	numeric scalar applied to RPSV(capthist, CC = TRUE)
ignoreusage	logical for whether to discard usage information from traps(capthist)
ncores	integer number of threads to be used for parallel processing

Details

Plausible starting values are needed to avoid numerical problems when fitting SECR models. Actual models to be fitted will usually have more than the three basic parameters output by autoini; other initial values can usually be set to zero for secr.fit. If the algorithm encounters problems obtaining a value for g0, the default value of 0.1 is returned.

Only the halfnormal detection function is currently available in autoini (cf other options in e.g. detectfn and sim.capthist).

autoini implements a modified version of the algorithm proposed by Efford et al. (2004). In outline, the algorithm is

1. Find value of sigma that predicts the 2-D dispersion of individual locations (see RPSV).
2. Find value of g0 that, with sigma, predicts the observed mean number of captures per individual (by algorithm of Efford et al. (2009, Appendix 2))
3. Compute the effective sampling area from g0, sigma, using thinned mask (see esa)
4. Compute D = n/esa(g0, sigma), where n is the number of individuals detected

Here 'find' means solve numerically for zero difference between the observed and predicted values, using uniroot.

Halfnormal sigma is estimated with RPSV(capthist, CC = TRUE). The factor adjustsigma is applied as a crude correction for truncation of movements at the edge of the detector array.

If RPSV cannot be computed the algorithm tries to use observed mean recapture distance $\bar{d}$. Computation of $\bar{d}$ fails if there no recaptures, and all returned values are NA.

If the mask has more than 100 points then a proportion 1–thin of points are discarded at random to speed execution.

The argument tol is passed to uniroot. It may be a vector of two values, the first for g0 and the second for sigma.

If traps(capthist) has a usage attribute (defining effort on each occasion at each detector) then the value of g0 is divided by the mean of the non-zero elements of usage. This adjustment is not precise.

If adjustg0 is TRUE then an adjustment is made to g0 depending on the value of binomN. For Poisson counts (binomN = 0) the adjustment is linear on effort (adjusted.g0 = g0 / usage). Otherwise, the adjustment is on the hazard scale (adjusted.g0 = 1 – (1 – g0) ^ (1 / (usage x binomN))). An arithmetic average is taken over all non-zero usage values (i.e. over used detectors and times). If usage is not specified it is taken to be 1.0.

Setting ncores = NULL uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see setNumThreads).

Value

A list of parameter values :

D	Density (animals per hectare)
g0	Magnitude (intercept) of detection function
sigma	Spatial scale of detection function (m)

Note

autoini always uses the Euclidean distance between detectors and mask points.

You may get this message from secr.fit: "'autoini' failed to find g0; setting initial g0 = 0.1". If the fitted model looks OK (reasonable estimates, non-missing SE) there is no reason to worry about the starting values. If you get this message and model fitting fails then supply your own values in the start argument of secr.fit.

References

Efford, M. G., Dawson, D. K. and Robbins C. S. (2004) DENSITY: software for analysing capture–recapture data from passive detector arrays. *Animal Biodiversity and Conservation* **27**, 217–228.

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.

See Also

capthist, mask, secr.fit, dbar

Examples

```
## Not run:

demotraps <- make.grid()
demomask <- make.mask(demotraps)
demoCH <- sim.capthist (demotraps, popn = list(D = 5, buffer = 100), seed = 321)
autoini (demoCH, demomask)


## End(Not run)
```

---

binCovariate *Add Binned Covariate*

Description

Forms a new covariate, replacing values of an old numeric covariate either by the central value of equal-width bins (a new numeric covariate) or factor levels from `cut`.

Usage

```
binCovariate(object, covname, width, breaks = NULL, replace = FALSE, ...)
```

Arguments

`object`	secr object with covariates attribute (capthist, traps, mask)
`covname`	character name of covariate
`width`	numeric bin width
`breaks`	numeric (see `cut`)
`replace`	logical; if TRUE any existing covariate of the new name is replaced
`...`	arguments passed to cut when breaks is provided

Details

Argument 'width' is ignored if 'breaks' is specified.

By default (breaks not specified) the name of the new covariate is `paste0(covname, width)`. Otherwise, the new covariate name is `paste0(covname, 'cut')`.

Fails if covariate not found or is not numeric or there is already a covariate with the new name and replace = FALSE.

Multi-session objects are handled appropriately.

Value

Object of the same class as the input with new covariate.

See Also

covariates, skink

Examples

```
# bin values of skink snout-vent length (mm)
infraCH <- binCovariate (infraCH, 'SVL', 5)
table(covariates(infraCH[[1]])$SVL5)

# bin values of trap covariate 'HtBrack' (height of bracken, cm)
traps(infraCH) <- binCovariate(traps(infraCH), "HtBrack", 20)
table(covariates(traps(infraCH)[[1]])$HtBrack20)
```

---

blackbear *Black Bear Hair Snag Dataset*

Description

American black bears *Ursus americanus* were surveyed with baited hair snags in the Great Smoky Mountains National Park, Tennessee, in the summer of 2003.

Usage

```
blackbearCH
GSM
blackbear.0
blackbear.h2bk
```

Details

American black bears *Ursus americanus* were surveyed in the Tennessee sector of Great Smoky Mountains National Park over 9 June–15 August 2003. Baited hair snags (barbed wire enclosures) were operated for 10 weeks at 65 sites, about 1 km apart and mostly close to trails. Bait consisted of bakery products in a small waxed-paper bag. Raspberry extract was used as a scent lure.

Genotyping and non-spatial capture-recapture analysis of a data subset were described by Settlage et al. (2008). The sex of each genotyped bear was determined subsequently and some additional samples were included (J. Laufenberg pers. comm. 2012-05-09; Laufenberg et al. 2013).

The dataset is a single-session capthist object with binary proximity detector type. Snags were visited weekly, so there were 10 occasions in the raw data. A single covariate 'sex' was recorded for each individual.

The dataset comprises 282 detections of 81 females and 58 males. Female 15 apparently made a long movement (17 km) between occasions 1 and 3.

The hair snag array sampled less than 20% of the area of the park. The unforested area outside the park on the northwestern boundary of the study area was not considered to be black bear habitat (F.

van Manen pers. comm. 2012-05-18) and should be excluded in analyses. The approximate boundary of the park is included as a shapefile 'GSMboundary.shp' in the 'extdata' folder of the package and as the **sf** sfc_POLYGON object `GSM`. The latter may be used in `make.mask` (see Examples).

Two models (blackbear.0 and blackbear.h2bk) were fitted as shown in the Examples.

Source

The data were provided by Jared Laufenberg, Frank van Manen and Joe Clark.

References

Laufenberg, J. S., van Manen, F. T., and Clark, J. D. (2013) Effects of sampling conditions on DNA-based estimates of American black bear abundance. *Journal of Wildlife Management* **77**, 1010–1020.

Settlage, K. E., Van Manen, F. T., Clark, J. D., and King, T. L. (2008) Challenges of DNA-based mark–recapture studies of American black bears. *Journal of Wildlife Management* **72**, 1035–1042.

Examples

```
summary(blackbearCH)

# Fitted models are included as data
AIC(blackbear.0, blackbear.h2bk)
summary(blackbear.h2bk)

# How many if we extrapolate to GSM NP?
region.N(blackbear.h2bk, region = GSM)



# GSM is the approximate boundary of Great Smoky Mountains National Park
# Make a habitat mask restricted to the park

tr <- traps(blackbearCH)
msk <- make.mask(tr, buffer = 6000, type = 'trapbuffer', poly = GSM)

# Plot

plot(GSM)
plot(msk, add = TRUE)
plot(blackbearCH, tracks = TRUE, add = TRUE)
plot(tr, add = TRUE)



## Not run:

# The models were fitted as follows (several minutes)
# suppress fastproximity to allow learned response

setNumThreads()   # as appropriate
```

```
# null model
blackbear.0 <- secr.fit(blackbearCH,  detectfn = 'EX', hcov = 'sex',
    mask = msk, details = list(fastproximity = FALSE), trace = FALSE)

# sex differences and site-specific behavioural response
blackbear.h2bk <- secr.fit(blackbearCH,  detectfn = 'EX', hcov = 'sex',
    model = list(g0~bk+h2, sigma~h2), mask = msk,
    details = list(fastproximity = FALSE), trace = FALSE)


## End(Not run)
```

BUGS *Convert Data To Or From BUGS Format*

Description

Convert data between 'capthist' and BUGS input format.

Usage

```
read.DA(DAlist, detector = "polygonX", units = 1, session = 1,
    Y = "Y", xcoord = "U1", ycoord = "U2", xmin = "Xl",
    xmax = "Xu", ymin = "Yl", ymax = "Yu", buffer = "delta",
    verify = TRUE)

write.DA(capthist, buffer, nzeros = 200, units = 1)
```

Arguments

DAlist	list containing data in BUGS format
detector	character value for detector type: 'polygon' or 'polygonX'
units	numeric for scaling output coordinates
session	numeric or character label used in output
Y	character, name of binary detection history matrix (animals x occasions)
xcoord	character, name of matrix of x-coordinates for each detection in Y
ycoord	character, name of matrix of y-coordinates for each detection in Y
xmin	character, name of coordinate of state space boundary
xmax	character, name of coordinate of state space boundary
ymin	character, name of coordinate of state space boundary
ymax	character, name of coordinate of state space boundary
buffer	see Details

verify	logical if TRUE then the resulting capthist object is checked with verify
capthist	capthist object
nzeros	level of data augmentation (all-zero detection histories)

Details

Data for OpenBUGS or WinBUGS called from R using the package **R2WinBUGS** (Sturtz et al. 2005) take the form of an R list.

These functions are limited at present to binary data from a square quadrat such as used by Royle and Young (2008). Marques et al. (2011) provide an R function `create.data()` for generating simulated datasets of this sort (see `sim.capthist` for equivalent functionality).

When reading BUGS data –

The character values `Y`, `xcoord`, `ycoord`, `xmin` etc. are used to locate the data within `DAlist`, allowing for variation in the input names.

The number of sampling occasions is taken from the number of columns in `Y`. Each value in `Y` should be 0 or 1. Coordinates may be missing

A numeric value for `buffer` is the distance (in the original units) by which the limits Xl, Xu etc. should be shrunk to give the actual plot limits. If `buffer` is character then a component of `DAlist` contains the required numeric value.

Coordinates in the output will be *multiplied by* the scalar `units`.

Augmentation rows corresponding to 'all-zero' detection histories in `Y`, `xcoord`, and `ycoord` are discarded.

When writing BUGS data –

Null (all-zero) detection histories are added to the matrix of detection histories `Y`, and missing (NA) rows are added to the coordinate matrices `xcoord` and `ycoord`.

Coordinates in the output will be *divided by* the scalar `units`.

Value

For `read.DA`, an object of class 'capthist'.

For `write.DA`, a list with the components

Xl	left edge of state space
Xu	right edge of state space
Yl	bottom edge of state space
Yu	top edge of state space
delta	buffer between edge of state space and quadrat
nind	number of animals observed
nzeros	number of added all-zero detection histories
T	number of sampling occasions
Y	binary matrix of detection histories (dim = c(nind+nzeros, T))
U1	matrix of x-coordinates, dimensioned as Y
U2	matrix of y-coordinates, dimensioned as Y

U1 and U2 are 'NA' where animal was not detected.

References

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.

Sturtz, S., Ligges, U. and Gelman, A. (2005) R2WinBUGS: a package for running WinBUGS from R. *Journal of Statistical Software* **12**, 1–16.

See Also

hornedlizardCH, verify, capthist

Examples

```
write.DA (hornedlizardCH, buffer = 100, units = 100)

## In this example, the input uses Xl, Xu etc.
## for the limits of the plot itself, so buffer = 0.
## Input is in hundreds of metres.
## First, obtain the list lzdata
olddir <- setwd (system.file("extdata", package="secr"))
source ("lizarddata.R")
setwd(olddir)
str(lzdata)
## Now convert to capthist
tempcapt <- read.DA(lzdata, Y = "H", xcoord = "X",
    ycoord = "Y", buffer = 0, units = 100)

## Not run:

plot(tempcapt)
secr.fit(tempcapt, trace = FALSE)
## etc.


## End(Not run)
```

---

capthist *Spatial Capture History Object*

Description

A capthist object encapsulates all data needed by secr.fit, except for the optional habitat mask.

Details

An object of class `capthist` holds spatial capture histories, detector (trap) locations, individual covariates and other data needed for a spatially explicit capture-recapture analysis with `secr.fit`.

A `capthist` is primarily an array of values with dim(capthist) = c(nc, noccasions, ntraps) where nc is the number of detected individuals. Values maybe binary ({−1, 0, 1}) or integer depending on the detector type.

Deaths during the experiment are represented as negative values.

Ancillary data are retained as attributes of a `capthist` object as follows:

- traps — object of class `traps` (required)
- session — session identifier (required)
- covariates — dataframe of individual covariates (optional)
- cutval — threshold of signal strength for detection ('signal' only)
- signalframe — signal strength values etc., one row per detection ('signal' only)
- detectedXY — dataframe of coordinates for location within polygon ('polygon'-like detectors only)
- xylist — coordinates of telemetered animals
- Tu — detectors x occasions matrix of sightings of unmarked animals
- Tm — detectors x occasions matrix of sightings of marked but unidentified animals
- Tn — detectors x occasions matrix of sightings with unknown mark status

`read.capthist` is adequate for most data input. Alternatively, the parts of a capthist object can be assembled with the function `make.capthist`. Use `sim.capthist` for Monte Carlo simulation (simple models only). Methods are provided to display and manipulate `capthist` objects (print, summary, plot, rbind, subset, reduce) and to extract and replace attributes (covariates, traps, xy).

A multi-session `capthist` object is a list in which each component is a `capthist` for a single session. The list maybe derived directly from multi-session input in Density format, or by combining existing `capthist` objects with `MS.capthist`.

Note

Early versions of **secr** (before 3.0) used an individual x occasion matrix for data from single-catch and multi-catch traps, instead of a 3-D array. Entries in the matrix corresponded to trap numbers. The function `updateCH` converts the old format.

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., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

See Also

traps, secr.fit, read.capthist, make.capthist, sim.capthist, subset.capthist, rbind.capthist, MS.capthist, reduce.capthist, mask

---

capthist.parts *Dissect Spatial Capture History Object*

Description

Extract parts of an object of class ‘capthist’.

Usage

```
animalID(object, names = TRUE, sortorder = c("snk", "ksn"))
occasion(object, sortorder = c("snk", "ksn"))
trap(object, names = TRUE, sortorder = c("snk", "ksn"))
alive(object, sortorder = c("snk", "ksn"))
alongtransect(object, tol = 0.01)
xy(object)
xy(object) <- value
telemetryxy(object, includeNULL = FALSE)
telemetryxy(object) <- value
telemetered(object)
```

Arguments

object	a ‘capthist’ object
names	if FALSE the values returned are numeric indices rather than names
sortorder	character code for sort order (see Details)
tol	tolerance for snapping to transect line (m)
value	replacement value (see Details)
includeNULL	logical; if TRUE a NULL component is included for untelemetered animals

Details

These functions extract data on detections, ignoring occasions when an animal was not detected. By default, detections are ordered by occasion, animalID and trap (sortorder = "snk"). The alternative is to order by trap, occasion and animalID (sortorder = "ksn"). (‘n’, ‘s’ and ‘k’ are the indices used internally for animals, occasions and traps respectively).

For historical reasons, "ksn" is used for locations within polygons and similar (xy).

trap returns polygon or transect numbers if traps(object) has detector type ‘polygon’ or ‘transect’.

alongtransect returns the distance of each detection from the start of the transect with which it is associated.

Replacement values must precisely match `object` in number of detections and in their order. `xy<-` expects a dataframe of x and y coordinates for points of detection within a 'polygon' or 'transect' detector. `telemetryxy<-` expects a list of dataframes, one per telemetered animal.

Value

For `animalID` and `trap` a vector of numeric or character values, one per detection.

For `alive` a vector of logical values, one per detection.

For `occasion`, a vector of numeric values, one per detection.

For `xy`, a dataframe with one row per detection and columns 'x' and 'y'.

If `object` has multiple sessions, the result is a list with one component per session.

See Also

`capthist`, `polyID`, `signalmatrix`

Examples

```
## `captdata' is a demonstration dataset
animalID(captdata)

temp <- sim.capthist(popn = list(D = 1), make.grid(detector
    = "count"))
cbind(ID = as.numeric(animalID(temp)), occ = occasion(temp),
    trap = trap(temp))
```

`chat` *Overdispersion of Activity Centres*

Description

Stochastic variation in the intensity surface may cause the number of detected individuals n to be overdispersed relative to a Poisson distribution (Efford and Fletcher 2025). This can cause the sampling variance of density estimates to be understated.

Use `chat.nj` to compute Fletcher's $\hat{c}$ estimate of overdispersion for use as a variance inflation factor. The method requires replicate samples of the intensity surface, such as from multiple independent subgrids. Replicates may be provided as sessions or clusters of detectors within a session (see `cluster` and `trap.builder`).

`adjustVarD` adjusts the SE and confidence limits of density estimates using the given $\hat{c}$. The implementation is limited to simple detection models (see Warnings).

See Cooch and White (2022) for an introduction to measurement of overdispersion in capture–recapture due to non-independence in the detection process. The focus here is on overdispersion of the number detected n relative to the Poisson (or binomial) distribution of n expected when the distribution of activity centres is an inhomogeneous or homogeneous Poisson (or binomial) point process (IHPP).

Usage

```
chat.nj(object, bycluster = FALSE, ...)

adjustVarD(object, chatmin = 1, alpha = 0.05, chat = NULL)
```

Arguments

object	fitted secr model or dataframe (see Warnings for restrictions)
bycluster	logical; if TRUE then j refers to clusters within a single-session capthist rather than sessions
...	other arguments passed to Fletcher.chat, e.g., verbose = FALSE
chatmin	minimum value of Fletcher's $\hat{c}$
alpha	alpha level for confidence intervals
chat	numeric chat (optional)

Details

chat.nj uses expected.n to compute the expected values.

No adjustment is made by adjustVarD when $\hat{c}$ is less than the minimum. Set 'chatmin' to zero to override.

adjustVarD also accepts a single dataframe as the argument 'object'; the dataframe should have row 'D' and columns 'link', 'estimate', 'SE.estimate' as in the output from predict.secr.

Value

For chat.nj, usually a list comprising –

expected	expected number at each cluster or session
observed	observed number at each cluster or session
stats	vector of summary statistics: mean(expected), var(expected), mean(observed), var(observed), sbar, nu (=df), cX2 = X2/nu
chat	$\hat{c}$ (Fletcher or Wedderburn depending on 'type')

If 'verbose = FALSE' then only the numeric value of $\hat{c}$ is returned (a vector of 2 values if 'type = "both"').

For adjustVarD, a dataframe with one row for each session, based on predict.secr or derived.secr, with extra column 'c-hat'.

Warning

The variance inflation factor given by chat.nk was shown by Efford and Fletcher (2025) to be inadequate and should not be used. For replicate spatial samples, chat.nj is a better alternative.

adjustVarD previously computed Fletcher's 'chat' using chat.nk; that is no longer recommended.

References

Bischof, R., P. Dupont, C. Milleret, J. Chipperfield, and J. A. Royle. 2020. Consequences of ignoring group association in spatial capture–recapture analysis. *Wildlife Biology* wlb.00649. doi:10.2981/wlb.00649

Cooch, E. and White, G. (eds) (2022) *Program MARK: A Gentle Introduction*. 22nd edition. Most recent edition available online at www.phidot.org/software/mark/docs/book/.

Efford, M. G. and D. Fletcher. 2025. Effect of spatial overdispersion on confidence intervals for population density estimated by spatial capture-recapture. *bioRxiv* https://doi.org/10.1101/2024.03.12.584742

Fletcher, D. (2012) Estimating overdispersion when fitting a generalized linear model to sparse data. *Biometrika* **99**, 230–237.

Wedderburn, R. W. M. (1974) Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika* **61**, 439–47.

See Also

secr, make.mask, Detection functions, Fletcher.chat, cluster

Examples

```
## Not run:

## Clustered design
mini <- make.grid(nx = 3, ny = 3, spacing = 50, detector = "proximity")

tempgrids2 <- trap.builder (
    cluster = mini , method = "all",
    frame = expand.grid(x = seq(200, 1800, 400), y = seq(200, 1800, 400)),
    plt = TRUE)
tempmask2 <- make.mask(tempgrids2, buffer = 100, spacing = 20)
pop1 <- sim.popn(D = 2, core = tempmask2, model2D = 'rLGCP',
                 details = list(var = 0.5, scale = 100))
capt <- sim.capthist(tempgrids2, popn = pop1, noccasions = 5,
    detectpar = list(g0 = 0.2, sigma = 25))

fit <- secr.fit(capt, mask = tempmask2, trace = FALSE)

chat.nj(fit, bycluster = TRUE)


## End(Not run)
```

---

circular *Circular Probability*

Description

Functions to answer the question "what radius is expected to include proportion p of points from a circular bivariate distribution corresponding to a given detection function", and the reverse. These functions may be used to relate the scale parameter(s) of a detection function (e.g., σ) to home-range area (specifically, the area within an activity contour for the corresponding simple home-range model) (see Note).

WARNING: the default behaviour of these functions changed in version 2.6.0. Integration is now performed on the cumulative hazard (exposure) scale for all functions unless `hazard = FALSE`. Results will differ.

Usage

```
circular.r (p = 0.95, detectfn = 0, sigma = 1, detectpar = NULL, hazard
= TRUE, upper = Inf, ...)

circular.p (r = 1, detectfn = 0, sigma = 1, detectpar = NULL, hazard
= TRUE, upper = Inf, ...)
```

Arguments

`p`	vector of probability levels for which radius is required
`r`	vector of radii for which probability level is required
`detectfn`	integer code or character string for shape of detection function 0 = halfnormal, 2 = exponential etc. – see detectfn for other codes
`sigma`	spatial scale parameter of detection function
`detectpar`	named list of detection function parameters
`hazard`	logical; if TRUE the transformation $-log(1-g(d))$ is applied before integration
`upper`	numeric upper limit of integration
`...`	other arguments passed to `integrate`

Details

`circular.r` is the quantile function of the specified circular bivariate distribution (analogous to `qnorm`, for example). The quantity calculated by `circular.r` is sometimes called 'circular error probable' (see Note).

For detection functions with two parameters (intercept and scale) it is enough to provide `sigma`. Otherwise, `detectpar` should be a named list including parameter values for the requested detection function (g0 may be omitted, and order does not matter).

Detection functions in **secr** are expressed in terms of the decline in probability of detection with distance $g(d)$, and both `circular.r` and `circular.p` integrate this function by default. Rather than integrating $g(d)$ itself, it may be more appropriate to integrate $g(d)$ transformed to a hazard i.e. $1 - log(-g(d))$. This is selected with `hazard = TRUE`.

Integration may also fail with the message "roundoff error is detected in the extrapolation table". Setting `upper` to a large number less than infinity sometimes corrects this.

Value

Vector of values for the required radii or probabilities.

Note

The term 'circular error probable' has a military origin. It is commonly used for GPS accuracy with the default probability level set to 0.5 (i.e. half of locations are further than CEP from the true location). A circular bivariate normal distriubution is commonly assumed for the circular error probable; this is equivalent to setting `detectfn = "halfnormal"`.

Closed-form expressions are used for the normal and uniform cases; in the circular bivariate normal case, the relationship is $r = \sigma\sqrt{-2\ln(1-p)}$. Otherwise, the probability is computed numerically by integrating the radial distribution. Numerical integration is not foolproof, so check suspicious or extreme values.

When `circular.r` is used with the default `sigma = 1`, the result may be interpreted as the factor by which sigma needs to be inflated to include the desired proportion of activity (e.g., 2.45 sigma for 95% of points from a circular bivariate normal distribution fitted on the hazard scale (detectfn = 14) OR 2.24 sigma on the probability scale (detectfn = 0)).

References

Calhoun, J. B. and Casby, J. U. (1958) Calculation of home range and density of small mammals. Public Health Monograph No. 55. United States Government Printing Office.

Johnson, R. A. and Wichern, D. W. (1982) Applied multivariate statistical analysis. Prentice-Hall, Englewood Cliffs, New Jersey, USA.

See Also

`detectfn`, `detectfnplot`

Examples

```
## Calhoun and Casby (1958) p 3.
## give p = 0.3940, 0.8645, 0.9888
circular.p(1:3, hazard = FALSE)

## halfnormal, hazard-rate and exponential
circular.r ()
circular.r (detectfn = "HR", detectpar = list(sigma = 1, z = 4))
circular.r (detectfn = "EX")
circular.r (detectfn = "HHN")
circular.r (detectfn = "HHR", detectpar = list(sigma = 1, z = 4))
circular.r (detectfn = "HEX")

plot(seq(0, 5, 0.05), circular.p(r = seq(0, 5, 0.05)),
    type = "l", xlab = "Radius (multiples of sigma)", ylab = "Probability")
lines(seq(0, 5, 0.05), circular.p(r = seq(0, 5, 0.05), detectfn = 2),
    type = "l", col = "red")
lines(seq(0, 5, 0.05), circular.p(r = seq(0, 5, 0.05), detectfn = 1,
    detectpar = list(sigma = 1,z = 4)), type = "l", col = "blue")
```

```
abline (h = 0.95, lty = 2)

legend (2.8, 0.3, legend = c("halfnormal","hazard-rate, z = 4", "exponential"),
    col = c("black","blue","red"), lty = rep(1,3))

## in this example, a more interesting comparison would use
## sigma = 0.58 for the exponential curve.
```

---

clone *Replicate Rows*

Description

Clone rows of an object a constant or random number of times

Usage

```
  ## Default S3 method:
clone(object, type, ...)
  ## S3 method for class 'popn'
clone(object, type, ...)
  ## S3 method for class 'capthist'
clone(object, type, ...)
```

Arguments

`object`	any object
`type`	character ‘constant’, ‘poisson’, ‘truncatedpoisson’ or ‘nbinom’
`...`	other arguments for distribution function

Details

The ... argument specifies the number of times each row should be repeated. For random distributions (Poisson or negative binomial) ... provides the required parameter values: `lambda` for Poisson, `size, prob` or `size, mu` for negative binomial.

One application is to derive a population of cues from a popn object, where each animal in the original popn generates a number of cues from the same point.

Cloning a capthist object replicates whole detection histories. Individual covariates and detection-specific attributes (e.g., signal strength or xy location in polygon) are also replicated. Cloned data from single-catch traps will cause verify() to fail, but a model may still be fitted in `secr.fit` by overriding the check with `verify = FALSE`.

Value

Object of same class as `object` but with varying number of rows. For `clone.popn` and `capthist` an attribute 'freq' is set, a vector of length equal to the original number of rows giving the number of repeats (including zeros).

If popn or `capthist` is a multi-session object the returned value will be a multi-session object of the same length.

See Also

`sim.popn`

Examples

```
## population of animals at 1 / hectare generates random
## Poisson number of cues, lambda = 5
mics4 <- make.grid( nx = 2, ny = 2, spacing = 44, detector = "signal")
pop <- sim.popn (D = 1, core = mics4, buffer = 300, nsessions = 6)
pop <- clone (pop, "poisson", 5)
attr(pop[[1]],"freq")

clone(captdata, "poisson", 3)

# To avoid losing any individuals use zero-truncated Poisson
# First find lambda of truncated Poisson with given mean
getlambda <- function (target) {
    fn <- function(x) x / (1-exp(-x)) - target
    uniroot(interval = c(1e-8, target), f = fn)$root
}
clone(captdata, "truncatedpoisson", getlambda(3))
```

---

closedN *Closed population estimates*

Description

Estimate N, the size of a closed population, by several conventional non-spatial capture–recapture methods.

Usage

```
closedN(object, estimator = NULL, level = 0.95, maxN = 1e+07,
    dmax = 10 )
```

Arguments

`object`	`capthist` object
`estimator`	character; name of estimator (see Details)
`level`	confidence level (1 – alpha)
`maxN`	upper bound for population size
`dmax`	numeric, the maximum AIC difference for inclusion in confidence set

Details

Data are provided as spatial capture histories, but the spatial information (trapping locations) is ignored.

AIC-based model selection is available for the maximum-likelihood estimators `null`, `zippin`, `darroch`, `h2`, and `betabinomial`.

Model weights are calculated as

$$w_i = \frac{\exp(-\Delta_i/2)}{\sum \exp(-\Delta_i/2)}$$

Models for which dAICc > `dmax` are given a weight of zero and are excluded from the summation, as are non-likelihood models.

Computation of `null`, `zippin` and `darroch` estimates differs slightly from Otis et al. (1978) in that the likelihood is maximized over real values of N between `Mt1` and `maxN`, whereas Otis et al. considered only integer values.

Asymmetric confidence intervals are obtained in the same way for all estimators, using a log transformation of $\hat{N} - Mt1$ following Burnham et al. (1987), Chao (1987) and Rexstad and Burnham (1991).

The available estimators are

Name	Model	Description	Reference
`null`	M0	null	Otis et al. 1978 p.105
`zippin`	Mb	removal	Otis et al. 1978 p.108
`darroch`	Mt	Darroch	Otis et al. 1978 p.106-7
`h2`	Mh	2-part finite mixture	Pledger 2000
`betabinomial`	Mh	Beta-binomial continuous mixture	Dorazio and Royle 2003
`jackknife`	Mh	jackknife	Burnham and Overton 1978
`chao`	Mh	Chao's Mh estimator	Chao 1987
`chaomod`	Mh	Chao's modified Mh estimator	Chao 1987
`chao.th1`	Mth	sample coverage estimator 1	Lee and Chao 1994
`chao.th2`	Mth	sample coverage estimator 2	Lee and Chao 1994

Value

A dataframe with one row per estimator and columns

`model`	model in the sense of Otis et al. 1978
`npar`	number of parameters estimated

`loglik`	maximized log likelihood
`AIC`	Akaike's information criterion
`AICc`	AIC with small-sample adjustment of Hurvich & Tsai (1989)
`dAICc`	difference between AICc of this model and the one with smallest AICc
`Mt1`	number of distinct individuals caught
`Nhat`	estimate of population size
`seNhat`	estimated standard error of Nhat
`lclNhat`	lower 100 x level % confidence limit
`uclNhat`	upper 100 x level % confidence limit

Warning

If your data are from spatial sampling (e.g. grid trapping) it is recommended that you do *not* use these methods to estimate population size (see Efford and Fewster 2013). Instead, fit a spatial model and estimate population size with `region.N`.

Note

Prof. Anne Chao generously allowed me to adapt her code for the variance of the 'chao.th1' and 'chao.th2' estimators.

Chao's estimators have been subject to various improvements not included here (e.g., Chao et al. 2016).

References

Burnham, K. P. and Overton, W. S. (1978) Estimating the size of a closed population when capture probabilities vary among animals. *Biometrika* **65**, 625–633.

Chao, A. (1987) Estimating the population size for capture–recapture data with unequal catchability. *Biometrics* **43**, 783–791.

Chao, A., Ma, K. H., Hsieh, T. C. and Chiu, Chun-Huo (2016) SpadeR: Species-Richness Prediction and Diversity Estimation with R. R package version 0.1.1. https://CRAN.R-project.org/package=SpadeR

Dorazio, R. M. and Royle, J. A. (2003) Mixture models for estimating the size of a closed population when capture rates vary among individuals. *Biometrics* **59**, 351–364.

Efford, M. G. and Fewster, R. M. (2013) Estimating population size by spatially explicit capture–recapture. *Oikos* **122**, 918–928.

Hurvich, C. M. and Tsai, C. L. (1989) Regression and time series model selection in small samples. *Biometrika* **76**, 297–307.

Lee, S.-M. and Chao, A. (1994) Estimating population size via sample coverage for closed capture-recapture models. *Biometrics* **50**, 88–97.

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**, 1–135.

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

Rexstad, E. and Burnham, K. (1991) User's guide for interactive program CAPTURE. Colorado Cooperative Fish and Wildlife Research Unit, Fort Collins, Colorado, USA.

See Also

capthist, closure.test, region.N

Examples

```
closedN(deermouse.ESG)
```

---

closure.test *Closure tests*

Description

Perform tests to determine whether a population sampled by capture-recapture is closed to gains and losses over the period of sampling.

Usage

```
closure.test(object, SB = FALSE, min.expected = 2)
```

Arguments

object	capthist object
SB	logical, if TRUE then test of Stanley and Burnham 1999 is calculated in addition to that of Otis et al. 1978
min.expected	integer for the minimum expected count in any cell of a component 2x2 table

Details

The test of Stanley and Burnham in part uses a sum over 2x2 contingency tables; any table with a cell whose expected count is less than min.expected is dropped from the sum. The default value of 2 is that used by CloseTest (Stanley and Richards 2005, T. Stanley pers. comm.; see also Stanley and Burnham 1999 p. 203).

Value

In the case of a single-session capthist object, either a vector with the statistic (z-value) and p-value for the test of Otis et al. (1978 p. 120) or a list whose components are data frames with the statistics and p-values for various tests and test components as follows –

Otis	Test of Otis et al. 1978
Xc	Overall test of Stanley and Burnham 1999
NRvsJS	Stanley and Burnham 1999
NMvsJS	Stanley and Burnham 1999
MtvsNR	Stanley and Burnham 1999
MtvsNM	Stanley and Burnham 1999

compNRvsJS	Occasion-specific components of NRvsJS
compNMvsJS	Occasion-specific components of NMvsJS

Check the original papers for an explanation of the components of the Stanley and Burnham test.

In the case of a multi-session object, a list with one component (as above) for each session.

Note

No omnibus test exists for closure: the existing tests may indicate nonclosure even when a population is closed if other effects such as trap response are present (see White et al. 1982 pp 96–97). The test of Stanley and Burnham is sensitive to individual heterogeneity which is inevitable in most spatial sampling, and it should not in general be used for this sort of data.

References

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**, 1–135.

Stanley, T. R. and Burnham, K. P. (1999) A closure test for time-specific capture–recapture data. *Environmental and Ecological Statistics* **6**, 197–209.

Stanley, T. R. and Richards, J. D. (2005) A program for testing capture–recapture data for closure. *Wildlife Society Bulletin* **33**, 782–785.

White, G. C., Anderson, D. R., Burnham, K. P. and Otis, D. L. (1982) *Capture-recapture and removal methods for sampling closed populations*. Los Alamos National Laboratory, Los Alamos, New Mexico.

See Also

capthist

Examples

```
closure.test(captdata)
```

---

cluster *Detector Clustering*

Description

Clusters are uniform groups of detectors. Use these functions to extract or replace cluster information of a traps object, or extract cluster information for each detection in a capthist object.

Usage

```
clusterID(object)
clusterID(object) <- value
clustertrap(object)
clustertrap(object) <- value
```

Arguments

object	traps or capthist object
value	factor (clusterID) or integer-valued vector (clustertrap)

Details

Easy access to attributes used to define compound designs, those in which a detector array comprises several similar subunits ('clusters'). 'clusterID' identifies the detectors belonging to each cluster, and 'clustertrap' is a numeric index used to relate matching detectors in different clusters.

For replacement ('traps' only), the number of rows of value must match exactly the number of detectors in object.

'clusterID' and 'clustertrap' are assigned automatically by trap.builder. They may also be assigned to the traps object post hoc.

Value

Factor (clusterID) or integer-valued vector (clustertrap).

clusterID(object) may be NULL.

See Also

traps, trap.builder, mash, derivedCluster, cluster.counts, cluster.centres

Examples

```
## 25 4-detector clusters
mini <- make.grid(nx = 2, ny = 2)
tempgrid <- trap.builder (cluster = mini , method = "all",
    frame = expand.grid(x = seq(100, 500, 100), y = seq(100,
    500, 100)))
clusterID(tempgrid)
clustertrap(tempgrid)

tempCH <- sim.capthist(tempgrid)
table(clusterID(tempCH)) ## detections per cluster
cluster.counts(tempCH)   ## distinct individuals
```

---

coef.secr *Coefficients of secr Object*

Description

Extract coefficients (estimated beta parameters) from a spatially explicit capture–recapture model.

Usage

```
## S3 method for class 'secr'
 coef(object, alpha = 0.05, ...)
```

Arguments

object	secr object output from secr.fit
alpha	alpha level
...	other arguments (not used currently)

Value

A data frame with one row per beta parameter and columns for the coefficient, SE(coefficient), asymptotic lower and upper 100(1–alpha) confidence limits.

See Also

secr.fit, esaPlot

Examples

```
## load & extract coefficients of previously fitted null model
coef(secrdemo.0)
```

---

collate *Array of Parameter Estimates*

Description

Estimates from one or more openCR models are formed into an array.

Usage

```
## S3 method for class 'secr'
collate(object, ..., realnames = NULL, betanames = NULL, newdata = NULL,
    alpha = 0.05, perm = 1:4, fields = 1:4)

## S3 method for class 'ipsecr'
collate(object, ..., realnames = NULL, betanames = NULL, newdata = NULL,
    alpha = 0.05, perm = 1:4, fields = 1:4)

## S3 method for class 'secrlist'
collate(object, ..., realnames = NULL, betanames = NULL, newdata = NULL,
    alpha = 0.05, perm = 1:4, fields = 1:4)
```

Arguments

`object`	secr or secrlist object
`...`	other secr objects
`realnames`	character vector of real parameter names
`betanames`	character vector of beta parameter names
`newdata`	optional dataframe of values at which to evaluate models
`alpha`	alpha level for confidence intervals
`perm`	permutation of dimensions in output from `collate`
`fields`	vector to restrict summary fields in output

Details

`collate` extracts parameter estimates from a set of fitted secr model objects.

`fields` may be used to select a subset of summary fields ("estimate","SE.estimate","lcl","ucl") by name or number.

Value

A 4-dimensional array of model-specific parameter estimates. By default, the dimensions correspond respectively to

- rows in `newdata` (usually sessions),
- models,
- statistic fields (estimate, SE.estimate, lcl, ucl), and
- parameters ("phi", "sigma" etc.).

It often helps to reorder the dimensions with the `perm` argument.

See Also

`modelAverage`, `secr.fit`

Examples

```
collate (secrdemo.0, secrdemo.b, perm = c(4,2,3,1))[,,1,]
```

confint.secr *Profile Likelihood Confidence Intervals*

Description

Compute profile likelihood confidence intervals for 'beta' or 'real' parameters of a spatially explicit capture-recapture model,

Usage

```
## S3 method for class 'secr'
 confint(object, parm, level = 0.95, newdata = NULL,
tracelevel = 1, tol = 0.0001, bounds = NULL, ncores = NULL, ...)
```

Arguments

object	secr model object
parm	numeric or character vector of parameters
level	confidence level (1 – alpha)
newdata	optional dataframe of values at which to evaluate model
tracelevel	integer for level of detail in reporting (0,1,2)
tol	absolute tolerance (passed to uniroot)
bounds	numeric vector of outer starting values – optional
ncores	number of threads used for parallel processing
...	other arguments (not used)

Details

If parm is numeric its elements are interpreted as the indices of 'beta' parameters; character values are interpreted as 'real' parameters. Different methods are used for beta parameters and real parameters. Limits for the j-th beta parameter are found by a numerical search for the value satisfying $-2(l_j(\beta_j) - l) = q$, where l is the maximized log likelihood, $l_j(\beta_j)$ is the maximized profile log likelihood with β_j fixed, and q is the $100(1-\alpha)$ quantile of the χ^2 distribution with one degree of freedom. Limits for real parameters use the method of Lagrange multipliers (Fletcher and Faddy 2007), except that limits for constant real parameters are backtransformed from the limits for the relevant beta parameter.

If bounds is provided it should be a 2-vector or matrix of 2 columns and length(parm) rows.

Setting ncores = NULL uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see setNumThreads).

Value

A matrix with one row for each parameter in parm, and columns giving the lower (lcl) and upper (ucl) 100*level

Note

Calculation may take a long time, so probably you will do it only after selecting a final model.

The R function uniroot is used to search for the roots of $-2(l_j(\beta_j) - l) = q$ within a suitable interval. The interval is anchored at one end by the MLE, and at the other end by the MLE inflated by a small multiple of the asymptotic standard error (1, 2, 4 or 8 SE are tried in turn, using the smallest for which the interval includes a valid solution).

A more efficient algorithm was proposed by Venzon and Moolgavkar (1988); it has yet to be implemented in **secr**, but see plkhci in the package **Bhat** for another R implementation.

References

Evans, M. A., Kim, H.-M. and O'Brien, T. E. (1996) An application of profile-likelihood based confidence interval to capture–recapture estimators. *Journal of Agricultural, Biological and Experimental Statistics* **1**, 131–140.

Fletcher, D. and Faddy, M. (2007) Confidence intervals for expected abundance of rare species. *Journal of Agricultural, Biological and Experimental Statistics* **12**, 315–324.

Venzon, D. J. and Moolgavkar, S. H. (1988) A method for computing profile-likelihood-based confidence intervals. *Applied Statistics* **37**, 87–94.

Examples

```
## Not run:

## Limits for the constant real parameter "D"
confint(secrdemo.0, "D")


## End(Not run)
```

---

contour *Contour Detection Probability*

Description

Display contours of the net probability of detection p.(X), or the area within a specified distance of detectors. bufferContour adds a conventional 'boundary strip' to a detector (trap) array, where buffer equals the strip width.

Usage

```
pdotContour(traps, border = NULL, nx = 64, detectfn = 0,
    detectpar = list(g0 = 0.2, sigma = 25, z = 1), noccasions = NULL,
    binomN = NULL, levels = seq(0.1, 0.9, 0.1), poly =
    NULL, poly.habitat = TRUE, plt = TRUE, add = FALSE, fill = NULL, ...)
```

```
bufferContour(traps, buffer, nx = 64, convex = FALSE, ntheta = 100,
     plt = TRUE, add = FALSE, poly = NULL, poly.habitat = TRUE,
     fill = NULL, ...)
```

Arguments

traps	traps object (or mask for bufferContour)
border	width of blank margin around the outermost detectors
nx	dimension of interpolation grid in x-direction
detectfn	integer code or character string for shape of detection function 0 = halfnormal etc. – see detectfn
detectpar	list of values for named parameters of detection function
noccasions	number of sampling occasions
binomN	integer code for discrete distribution (see secr.fit)
levels	vector of levels for p.(X)
poly	matrix of two columns, the x and y coordinates of a bounding polygon (optional)
poly.habitat	logical as in make.mask
plt	logical to plot contours
add	logical to add contour(s) to an existing plot
fill	vector of colours to fill contours (optional)
...	other arguments to pass to contour
buffer	vector of buffer widths
convex	logical, if TRUE the plotted contour(s) will be convex
ntheta	integer value for smoothness of convex contours

Details

pdotContour constructs a rectangular mask and applies pdot to compute the p.(X) at each mask point.

If convex = FALSE, bufferContour constructs a mask and contours the points on the basis of distance to the nearest detector at the levels given in buffer.

If convex = TRUE, bufferContour constructs a set of potential vertices by adding points on a circle of radius = buffer to each detector location; the desired contour is the convex hull of these points (this algorithm derives from Efford, 2012).

If traps has a usage attribute then noccasions is set accordingly; otherwise it must be provided.

If traps is for multiple sessions then detectpar should be a list of the same length, one component per session, and noccasions may be a numeric vector of the same length.

Increase nx for smoother lines, at the expense of speed.

Value

Coordinates of the plotted contours are returned as a list with one component per polygon. The list is returned invisibly if `plt = TRUE`.

For multi-session input (`traps`) the value is a list of such lists, one per session.

Note

The precision (smoothness) of the fitted line in `bufferContour` is controlled by `ntheta` rather than `nx` when `convex = TRUE`.

To suppress contour labels, include the argument `drawlabels = FALSE` (this will be passed via ... to `contour`). Other useful arguments of `contour` are `col` (colour of contour lines) and `lwd` (line width).

You may wish to consider function st_buffer in package **sf** as an alternative to `bufferContour`.

`bufferContour` failed with multi-session `traps` before **secr** 2.8.0.

References

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

See Also

pdot, make.mask

Examples

```
possumtraps <- traps(possumCH)

## convex and concave buffers
plot(possumtraps, border = 270)
bufferContour(possumtraps, buffer = 100, add = TRUE, col = "blue")
bufferContour(possumtraps, buffer = 100, convex = TRUE, add = TRUE)

## areas
buff.concave <- bufferContour(possumtraps, buffer = 100,
    plt = FALSE)
buff.convex  <- bufferContour(possumtraps, buffer = 100,
    plt = FALSE, convex = TRUE)
sum (sapply(buff.concave, polyarea)) ## sum over parts
sapply(buff.convex, polyarea)

## effect of nx on area
buff.concave2 <- bufferContour(possumtraps, buffer = 100,
    nx = 128, plt = FALSE)
sum (sapply(buff.concave2, polyarea))

## Not run:
```

```
plot(possumtraps, border = 270)
pdotContour(possumtraps, detectfn = 0, nx = 128, detectpar =
    detectpar(possum.model.0), levels = c(0.1, 0.01, 0.001),
    noccasions = 5, add = TRUE)

## clipping to polygon
olddir <- setwd(system.file("extdata", package = "secr"))
possumtraps <- traps(possumCH)
possumarea <- read.table("possumarea.txt", header = TRUE)
par(xpd = TRUE, mar = c(1,6,6,6))
plot(possumtraps, border = 400, gridlines = FALSE)
pdotContour(possumtraps, detectfn = 0, nx = 256, detectpar =
    detectpar(possum.model.0), levels = c(0.1, 0.01, 0.001),
    noccasions = 5, add = TRUE, poly = possumarea, col = "blue")
lines(possumarea)
setwd(olddir)
par(xpd = FALSE, mar = c(5,4,4,2) + 0.1)    ## reset to default


## End(Not run)
```

`covariates` *Covariates Attribute*

Description

Extract or replace covariates

Usage

```
covariates(object, ...)
covariates(object) <- value
```

Arguments

`object`	an object of class `traps`, `popn`, `capthist`, or `mask`
`value`	a dataframe of covariates
`...`	other arguments (not used)

Details

For replacement, the number of rows of `value` must match exactly the number of rows in `object`.

Value

covariates(object) returns the dataframe of covariates associated with `object`. `covariates(object)` may be NULL.

Individual covariates are stored in the 'covariates' attribute of a `capthist` object.

Covariates used for modelling density are stored in the 'covariates' attribute of a `mask` object.

Detector covariates may vary between sampling occasions. In this case, columns in the detector covariates data.frame are associated with particular times; the matching is controlled by the `timevaryingcov` attribute.

See Also

`timevaryingcov`

Examples

```
## detector covariates
temptrap <- make.grid(nx = 6, ny = 8)
covariates (temptrap) <- data.frame(halfnhalf =
    factor(rep(c("left","right"),c(24,24))) )
summary(covariates(temptrap))
```

CV — *Coefficient of Variation*

Description

The coefficient of variation of effective sampling area predicts the bias in estimated density (Efford and Mowat 2014). These functions assist its calculation from fitted finite mixture models.

Usage

```
CV(x, p, na.rm = FALSE)
CVa0(object, ...)
CVa(object, sessnum = 1, ...)
```

Arguments

`x`	vector of numeric values
`p`	vector of class probabilities
`na.rm`	logical; if TRUE missing values are dropped from x
`object`	fitted secr finite mixture model
`sessnum`	integer sequence number of session to analyse
`...`	other arguments passed to predict.secr (e.g., `newdata`)

Details

CV computes the coefficient of variation of x. If p is provided then the distribution is assumed to be discrete, with support x and class membership probabilities p (scaled automatically to sum to 1.0).

CVa computes CV(a) where a is the effective sampling area of Borchers and Efford (2008).

CVa0 computes CV(a0) where a0 is the single-detector sampling area defined as $a_0 = 2\pi\lambda_0\sigma^2$ (Efford and Mowat 2014); a0 is a convenient surrogate for a, the effective sampling area. CV(a0) uses either the fitted MLE of a0 (if the a0 parameterization has been used), or a0 computed from the estimates of lambda0 and sigma.

CVa and CVa0 do not work for models with individual covariates.

Value

Numeric

Note

Do not confuse the function CVa with the estimated relative standard error of the estimate of a from derived, also labelled CVa in the output. The relative standard error RSE is often labelled CV in the literature on capture–recapture, but this can cause unnecessary confusion. See also RSE.

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. and Mowat, G. (2014) Compensatory heterogeneity in capture–recapture data. *Ecology* **95**, 1341–1348.

See Also

CVpdot, derived, details, RSE

Examples

```
## Not run:

## housemouse model
morning <- subset(housemouse, occ = c(1,3,5,7,9))
msk <- make.mask((traps(morning)), nx = 32)
morning.h2   <- secr.fit(morning, buffer = 20, model = list(g0~h2), mask = msk,
    trace = FALSE)
CVa0(morning.h2 )


## End(Not run)
```

D.designdata *Construct Density Design Data*

Description

Internal function used by secr.fit, confint.secr, and score.test.

Usage

```
D.designdata (mask, Dmodel, grouplevels, sessionlevels, sessioncov =
NULL, meanSD = NULL)
```

Arguments

mask	mask object.
Dmodel	formula for density model
grouplevels	vector of group names
sessionlevels	vector of character values for session names
sessioncov	optional dataframe of values of session-specific covariate(s).
meanSD	optional external values for scaling x- and y- coordinates

Details

This is an internal **secr** function that you are unlikely ever to use. Unlike secr.design.MS, this function does *not* call model.matrix.

Value

Dataframe with one row for each combination of mask point, group and session. Conceptually, we use a 3-D rectangular array with enough rows to accommodate the largest mask, so some rows in the output may merely hold space to enable easy indexing. The dataframe has an attribute 'dimD' that gives the relevant dimensions: attr(dframe, "dimD") = c(nmask, ngrp, R), where nmask is the number of mask points, ngrp is the number of groups, and R is the number of sessions. Columns correspond to predictor variables in Dmodel.

The number of valid rows (points in each session-specific mask) is stored in the attribute 'validMaskRows'.

For a single-session mask, meanSD is a 2 x 2 matrix of mean and SD (rows) for x- and y-coordinates. For a multi-session mask, a list of such objects. Ordinarily these values are from the meanSD attribute of the mask, but they must be specified when applying a new mask to an existing model.

See Also

secr.design.MS

`deermouse` *Deermouse Live-trapping Datasets*

Description

Data of V. H. Reid from live trapping of deermice (*Peromyscus maniculatus*) at two sites in Colorado, USA.

Usage

```
deermouse.ESG
deermouse.WSG
```

Details

Two datasets of V. H. Reid were described by Otis et al. (1978) and distributed with their CAPTURE software. They have been used in several other papers on closed population methods (e.g., Huggins 1991, Stanley and Richards 2005). This description is based on pages 32 and 87–93 of Otis et al. (1978).

Both datasets are from studies in Rio Blanco County, Colorado, in the summer of 1975. Trapping was for 6 consecutive nights. Traps were arranged in a 9 x 11 grid and spaced 50 feet (15.2 m) apart.

The first dataset was described by Otis et al. (1978: 32) as from 'a drainage bottom of sagebrush, gambel oak, and serviceberry with pinyon pine and juniper on the uplands'. By matching with the 'examples' file of CAPTURE this was from East Stuart Gulch (ESG).

The second dataset (Otis et al. 1978: 87) was from Wet Swizer Creek or Gulch (WSG) in August 1975. No specific vegetation description is given for this site, but it is stated that Sherman traps were used and trapping was done twice daily.

Two minor inconsistencies should be noted. Although Otis et al. (1978) said they used data from morning trap clearances, the capture histories in 'examples' from CAPTURE include a 'pm' tag on each record. We assume the error was in the text description, as their numerical results can be reproduced from the data file. Huggins (1991) reproduced the East Stuart Gulch dataset (omitting spatial data that were not relevant to his method), but omitted two capture histories.

The data are provided as two single-session `capthist` objects 'deermouse.ESG' and 'deermouse.WSG'. Each has a dataframe of individual covariates, but the fields differ between the two study areas. The individual covariates of deermouse.ESG are sex (factor levels 'f', 'm'), age class (factor levels 'y', 'sa', 'a') and body weight in grams. The individual covariates of deermouse.WSG are sex (factor levels 'f','m') and age class (factor levels 'j', 'y', 'sa', 'a') (no data on body weight). The aging criteria used by Reid are not recorded.

The datasets were originally in the CAPTURE 'xy complete' format which for each detection gives the 'column' and 'row' numbers of the trap (e.g. ' 9 5' for a capture in the trap at position (x=9, y=5) on the grid). Trap identifiers have been recoded as strings with no spaces by inserting zeros (e.g. '905' in this example).

Sherman traps are designed to capture one animal at a time, but the data include double captures (1 at ESG and 8 at WSG – see Examples). The true detector type therefore falls between 'single' and 'multi'. Detector type is set to 'multi' in the distributed data objects.

Object	Description
deermouse.ESG	capthist object, East Stuart Gulch
deermouse.WSG	capthist object, Wet Swizer Gulch

Source

File 'examples' distributed with program CAPTURE.

References

Huggins, R. M. (1991) Some practical aspects of a conditional likelihood approach to capture experiments. *Biometrics* **47**, 725–732.

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**, 1–135.

Stanley, T. R. and Richards, J. D. (2005) A program for testing capture–recapture data for closure. *Wildlife Society Bulletin* **33**, 782–785.

See Also

closure.test

Examples

```
par(mfrow = c(1,2), mar = c(1,1,4,1))
plot(deermouse.ESG, title = "Peromyscus data from East Stuart Gulch",
    border = 10, gridlines = FALSE, tracks = TRUE)
plot(deermouse.WSG, title = "Peromyscus data from Wet Swizer Gulch",
    border = 10, gridlines = FALSE, tracks = TRUE)

closure.test(deermouse.ESG, SB = TRUE)

## reveal multiple captures
table(trap(deermouse.ESG), occasion(deermouse.ESG))
table(trap(deermouse.WSG), occasion(deermouse.WSG))
```

---

deleteMaskPoints *Edit Mask Points*

Description

Mask points may be removed by one of three methods: clicking on points, clicking on vertices to define a polygon from which points will be removed, or specifying a polygon to which the mask will be clipped.

Usage

```
deleteMaskPoints(mask, onebyone = TRUE, add = FALSE, poly = NULL,
 poly.habitat = FALSE, ...)
```

Arguments

mask	secr mask object
onebyone	logical; see Details
add	logical; if true then the initial mask plot will be added to an existing plot
poly	polygon defining habitat or non-habitat as described in make.mask
poly.habitat	logical; if TRUE polygon represents habitat
...	other arguments to plot.mask

Details

The default method (onebyone = TRUE, poly = NULL) is to click on each point to be removed. The nearest mask point will be selected.

Setting onebyone = FALSE allows the user to click on the vertices of a polygon within which all points are to be removed (the default) or retained (poly.habitat = TRUE). Vertices need not coincide with mask points.

Defining poly here is equivalent to calling make.mask with poly defined. poly one of the several types described in boundarytoSF. Whether poly represents habitat or non-habitat is toggled with poly.habitat – the default here differs from make.mask.

Value

A mask object, usually with fewer points than the input mask.

See Also

make.mask, subset.mask

Examples

```
if (interactive()) {
    mask0 <- make.mask (traps(captdata))
    ## Method 1 - click on each point to remove
    mask1 <- deleteMaskPoints (mask0)
    ## Method 2 - click on vertices of removal polygon
    mask2 <- deleteMaskPoints (mask0, onebyone = FALSE)
    ## Method 3 - predefined removal polygon
    plot(captdata)
    poly1 <- locator(5)
    mask3 <- deleteMaskPoints (mask0, poly = poly1)
}
```

derived *Derived Parameters of Fitted SECR Model*

Description

Compute derived parameters of spatially explicit capture-recapture model. Density is a derived parameter when a model is fitted by maximizing the conditional likelihood. So also is the effective sampling area (in the sense of Borchers and Efford 2008).

Usage

```
derived(object, ...)

## S3 method for class 'secr'
derived(object, sessnum = NULL, groups = NULL, alpha = 0.05,
    se.esa = FALSE, se.D = TRUE, loginterval = TRUE, distribution = NULL,
    ncores = NULL, bycluster = FALSE, Dweight = FALSE, ...)

## S3 method for class 'secrlist'
derived(object, sessnum = NULL, groups = NULL, alpha = 0.05,
    se.esa = FALSE, se.D = TRUE, loginterval = TRUE, distribution = NULL,
    ncores = NULL, bycluster = FALSE, Dweight = FALSE, ...)

## S3 method for class 'secr'
esa(object, sessnum = 1, beta = NULL, real = NULL, noccasions = NULL,
    ncores = NULL, Dweight = FALSE, ...)
```

Arguments

`object`	`secr` object output from `secr.fit`, or an object of class `c("secrlist", "list")`
`sessnum`	index of session in object$capthist for which output required
`groups`	vector of covariate names to define group(s) (see Details)
`alpha`	alpha level for confidence intervals
`se.esa`	logical for whether to calculate SE(mean(esa))
`se.D`	logical for whether to calculate SE(D-hat)
`loginterval`	logical for whether to base interval on log(D)
`distribution`	character string for distribution of the number of individuals detected
`ncores`	integer number of threads used for parallel processing
`bycluster`	logical; if TRUE results are reported separately for each cluster of detectors
`Dweight`	logical; if TRUE and a (relative) density model has been fitted then the effective sampling area uses density weights
`beta`	vector of fitted parameters on transformed (link) scale

real	vector of 'real' parameters
noccasions	integer number of sampling occasions (see Details)
...	other arguments passed to fdHess or ignored (esa)

Details

The derived estimate of density is a Horvitz-Thompson-like estimate:

$$\hat{D} = \sum_{i=1}^{n} a_i(\hat{\theta})^{-1}$$

where $a_i(\hat{\theta})$ is the estimate of effective sampling area for animal i with detection parameter vector θ.

A non-null value of the argument distribution overrides the value in object$details. The sampling variance of $\hat{D}$ from secr.fit by default is spatially unconditional (distribution = "Poisson"). For sampling variance conditional on the population of the habitat mask (and therefore dependent on the mask area), specify distribution = "binomial". The equation for the conditional variance includes a factor $(1 - a/A)$ that disappears in the unconditional (Poisson) variance (Borchers and Efford 2007). Thus the conditional variance is always less than the unconditional variance. The unconditional variance may in turn be an overestimate or (more likely) an underestimate if the true spatial variance is non-Poisson.

Derived parameters may be estimated for population subclasses (groups) defined by the user with the groups argument. Each named factor in groups should appear in the covariates dataframe of object$capthist (or each of its components, in the case of a multi-session dataset).

esa is used by derived to compute individual-specific effective sampling areas:

$$a_i(\hat{\theta}) = \int_A p.(\mathbf{X}; \mathbf{z}_i, \hat{\theta})\, \mathrm{d}\mathbf{X}$$

where $p.(\mathbf{X})$ is the probability an individual at X is detected at least once and the $\mathbf{z}_i$ are optional individual covariates. Integration is over the area A of the habitat mask.

The argument noccasions may be used to vary the number of sampling occasions; it works only when detection parameters are constant across individuals and across time.

Setting ncores = NULL uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see setNumThreads).

The effective sampling area 'esa' ($a(\hat{\theta})$) reported by derived is equal to the harmonic mean of the $a_i(\hat{\theta})$ (arithmetic mean prior to version 1.5). The sampling variance of $a(\hat{\theta})$ is estimated by

$$\widehat{\mathrm{var}}(a(\hat{\theta})) = \hat{G}_\theta^T \hat{V}_\theta \hat{G}_\theta,$$

where $\hat{V}$ is the asymptotic estimate of the variance-covariance matrix of the beta detection parameters (θ) and $\hat{G}$ is a numerical estimate of the gradient of $a(\theta)$ with respect to θ, evaluated at $\hat{\theta}$.

A 100(1–alpha)% asymptotic confidence interval is reported for density. By default, this is asymmetric about the estimate because the variance is computed by backtransforming from the log scale. You may also choose a symmetric interval (variance obtained on natural scale).

The vector of detection parameters for esa may be specified via beta or real, with the former taking precedence. If neither is provided then the fitted values in objectfitpar are used. Specifying real parameter values bypasses the various linear predictors. Strictly, the 'real' parameters are for a naive capture (animal not detected previously).

The computation of sampling variances is relatively slow and may be suppressed with se.esa and se.D as desired.

For computing derived across multiple models in parallel see par.derived.

From **secr** 5.0.0 the ... argument may be used to control the step size (.relStep) used by fdHess when estimating gradients for SE(D) and SE(esa).

Value

Dataframe with one row for each derived parameter ('esa', 'D') and columns as below

estimate	estimate of derived parameter
SE.estimate	standard error of the estimate
lcl	lower 100(1–alpha)% confidence limit
ucl	upper 100(1–alpha)% confidence limit
CVn	relative SE of number observed (Poisson or binomial assumption)
CVa	relative SE of effective sampling area
CVD	relative SE of density estimate

For a multi-session or multi-group analysis the value is a list with one component for each session and group.

The result will also be a list if object is an 'secrlist'.

For models of telemetry type 'marking' (see addTelemetry) the Horvitz-Thompson estimators above do not apply. Instead derived returns a matrix with a single row 'Dcw' and columns estimate, SE.estimate, lcl and ucl. Dcw is the cue-weighted mean density: the mean of the fitted density surface over the mask, weighted by the expected number of cues from an animal centred at each point ($D_{cw} = \sum_x D(x)L(x)/\sum_x L(x)$). It equals D when density is uniform. When density varies it is the estimate to report, because the unweighted mean of $D(x)$ over the mask depends on extrapolation to places the cameras hardly sample (region.N gives that). The standard error uses the delta method with the variance-covariance matrix of the coefficients.

Warning

derived() may be applied to detection models fitted by maximizing the full likelihood (CL = FALSE). However, models using g (groups) will not be handled correctly.

Note

Before version 2.1, the output table had columns for 'varcomp1' (the variance in $\hat{D}$ due to variation in n, i.e., Huggins' s^2), and 'varcomp2' (the variance in $\hat{D}$ due to uncertainty in estimates of detection parameters).

These quantities are related to CVn and CVa as follows:

$\mathrm{CVn} = \sqrt{\mathrm{varcomp1}}/\hat{D}$

$\mathrm{CVa} = \sqrt{\mathrm{varcomp2}}/\hat{D}$

References

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

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

Huggins, R. M. (1989) On the statistical analysis of capture experiments. *Biometrika* **76**, 133–140.

See Also

predict.secr, print.secr, secr.fit, empirical.varD

Examples

```
## Not run:
## extract derived parameters from a model fitted previously
## by maximizing the conditional likelihood
derived (secrdemo.CL)

## what happens when sampling variance is conditional on mask N?
derived(secrdemo.CL, distribution = "binomial")
## fitted g0, sigma
esa(secrdemo.CL)
## force different g0, sigma
esa(secrdemo.CL, real = c(0.2, 25))

## End(Not run)
```

---

derivedDsurface *Derived Absolute Density*

Description

A relative density model (one fitted by maximising the conditional likelihood) has a zero intercept for density on the default log scale (the intercept is 1.0 for the identity link).

The constant *k* that relates absolute density to relative density $D' = k^{-1}D$ is obtained as described in Efford (2025).

These functions infer the value of *k* and use this to construct the predicted density surface (predictDsurface is the equivalent function for full-likelihood models).

The derived method for 'secr' objects performs the same calculation as derivedDcoef and also back-transforms the intercept and computes a delta-method estimate of its variance and confidence limits. However that variance estimate is unreliable.

derivedDfit constructs an secr fitted model object resembling that from an unconditional model fit.

Usage

```
## S3 method for class 'secr'
derivedDcoef(object, se = FALSE, ...)
## S3 method for class 'secrlist'
derivedDcoef(object, se = FALSE, ...)

derivedDsurface(object, mask = NULL, sessnum = NULL, groups = NULL)
derivedDfit(object, vcv = TRUE)
```

Arguments

`object`	fitted secr relative density model
`se`	logical; if TRUE the variance of beta0 is estimated by the delta method
`mask`	new mask for which to compute densities
`sessnum`	integer session number
`groups`	character vector of covariate names to define groups (optional)
`vcv`	logical; if TRUE output includes the extended variance-covariance matrix
`...`	other arguments (not used)

Details

The theory is provided by Efford in prep.

`derivedDcoef(object, se = TRUE)` is equivalent to `derived(object, Dweight = TRUE)`, although `derivedDcoef` returns estimates on the link scale.

Value

For derivedDcoef –

A dataframe mimicking `coef(object)` with an initial row for the derived density intercept β_0. For an identity link, subsequent density coefficients are scaled by the derived β_0.

For derivedDsurface –

Dsurface object with class c("Dsurface", "mask", "data.frame"). Multi-session if object is multi-session and sessnum = NULL.

If groups are defined the result is a list of Dsurfaces.

References

Efford, M. G. In prep. SECR models for relative density.

See Also

`derived`

`predictDsurface`

details *Detail Specification for secr.fit*

Description

The function secr.fit allows many options. Some of these are used infrequently and have been bundled as a single argument details to simplify the documentation. They are described here. Some components (param, relativeD) are primarily used to record the type of model fitted that is an indirect result of other settings.

Detail components

details$autoini specifies the session number from which to compute starting values (multi-session data only; default 1). From 4.0.0, the character value 'all' first forms a single-session capthist using join(); this may be slow or not work at all (especially with telemetry data).

details$centred = TRUE causes coordinates of both traps and mask to be centred on the centroid of the traps, computed separately for each session in the case of multi-session data. This may be necessary to overcome numerical problems when x- or y-coordinates are large numbers. The default is not to centre coordinates.

details$chat optionally specifies the overdispersion of unmarked sightings Tu and unidentified marked sightings Tm. It is used only for mark-resight models, and is usually computed within secr.fit (details$nsim > 0), but may be provided by the user. For a single session 'chat' is a vector of length 2; for multiple sessions it is a 2-column matrix. For telemetry type 'marking' (see addTelemetry) 'chat' has three elements (a scalar is repeated): the overdispersion of the total count of unmarked sightings, of unidentified marked sightings Tm, and, for models with density covariates, of the spread of unmarked sightings across detectors (usually the Pearson chi-square statistic divided by its degrees of freedom, at least 1). Simulation of 'chat' (details$nsim) is not available for this telemetry type; use details$chatmethod = "analytic".

details$chatmethod is "simulate" (default) or "analytic". The default estimates overdispersion by simulation when details$nsim > 0. With "analytic" the overdispersion is computed from the variance of the total count of unmarked sightings implied by the model (compound Poisson: the cues from one animal fall at several detectors), at the parameter values in start, without simulation; details$nsim is ignored. It reproduces in expectation the quantity estimated by simulation (the variance of the pooled count divided by its naive Poisson or binomial variance), for count, proximity and multi detectors, a Poisson or fixed (binomial) number of animals (details$distribution), and is currently available for (i) sighting-only data (all markocc 0, i.e. all animals marked beforehand), and (ii) telemetry type 'marking', for which the posterior variance of the marked animals' contribution is included and, for models with density covariates, the Pearson dispersion of the spread of unmarked sightings across detectors is also returned. It is not available when there are marking occasions or unresolved sightings. For telemetry type 'marking' with CL = TRUE the overdispersion of the total count is not used and is returned as 1; the dispersion across detectors applies as usual when there is a density model. The overdispersion is much larger for a Poisson than a fixed number of animals. As with simulation, fit the model first (chat 1), then refit with start set to the first fit (or use secr.refit); method = "none" recomputes only the variance.

The analytic value for Tm treats the marked animals as fixed at their expected spatial distribution, so for count detectors it is 1. The simulation for sighting-only models (sightmodel 5 and 6) is

not quite the same: it places the marked animals at random but adds their expected cues (from the expected rather than the realised number in each mask cell) only in cells that contain a marked animal. The simulated overdispersion of Tm is then about 10 percent greater than the analytic value (for example 1.10–1.12 in a test with 25 marked animals; Tu is unaffected). Treating the realised locations of the marked animals as random would give a much larger value (about 2.2–2.6 in the same test). The simulation has been left unchanged for compatibility with earlier results.

`details$chatonly` = TRUE used with `details$nsim > 0` (or `details$chatmethod = "analytic"`) causes the overdispersion statistics for sighting counts Tu and Tm to be estimated and returned as a vector or 2-column matrix (multi-session models), with no further model fitting.

`details$contrasts` may be used to specify the coding of factor predictors. The value should be suitable for the 'contrasts.arg' argument of `model.matrix`. See 'Trend across sessions' in secr-multisession.pdf for an example.

`details$convexpolygon` may be set to FALSE for searches of non-convex polygons. This is slower than the default which requires poygons to be convex east-west (secr-polygondetectors.pdf).

`details$debug` is an integer code used to control the printing of intermediate values (1,2) and to switch on the R code browser (3). In ordinary use it should not be changed from the default (0).

`details$Dfn` is a function for reparameterizing density models; this is set internally when Dlambda = TRUE. Exotic variations may be specified directly by the user when Dlambda = FALSE. The defaults (Dfn = NULL, Dlambda = FALSE) leave the original density model unchanged. Note there is no connection to userDfn (except that the two are incompatible).

`Dlambda` if TRUE causes reparameterization of density as the session-on-session finite rate of increase $lambda$. Details at (secr-trend.pdf).

`details$distribution` specifies the distribution of the number of individuals detected n; this may be conditional on the number in the masked area ("binomial") or unconditional ("poisson"). `distribution` affects the sampling variance of the estimated density. The default is "poisson". The component 'distribution' may also take a numeric value larger than nrow(capthist), rather than "binomial" or "poisson". The likelihood then treats n as a binomial draw from a superpopulation of this size, with consequences for the variance of density estimates. This can help to reconcile MLE with Bayesian estimates using data augmentation.

`details$externalqx` names a mask covariate that is used as an offset for relative density. This can be useful in a two-phase study when animals are tagged in phase one and sampled in phase two, with no further tagging.

`details$fastproximity` controls special handling of data from binary proximity and count detectors. If TRUE and other conditions are met (no temporal variation or groups) then a multi-occasion capthist is automatically reduced to a count for a single occasion and further compressed by storing only non-zero counts, which can greatly speed up computation of the likelihood (default TRUE).

`details$fixedbeta` may be used to fix values of beta parameters. It should be a numeric vector of length equal to the total number of beta parameters (coefficients) in the model. Parameters to be estimated are indicated by NA. Other elements should be valid values on the link scale and will be substituted during likelihood maximisation. Check the order of beta parameters in a previously fitted model.

`details$grain` sets the grain argument for multithreading in RcppParallel parallelFor (default 1). `details$grain = 0` suppresses multithreading (equivalent to `ncores = 1`).

`details$hessian` is a character string controlling the computation of the Hessian matrix from which variances and covariances are obtained. Options are "none" (no variances), "auto" (the de-

fault) or "fdhess" (use the function fdHess in **nlme**). If "auto" then the Hessian from the optimisation function is used.

`details$ignoreusage` = TRUE causes the function to ignore usage (varying effort) information in the traps component. The default (`details$ignoreusage` = FALSE) is to include usage in the model.

`details$intwidth2` controls the half-width of the interval searched by optimise() for the maximum likelihood when there is a single parameter. Default 0.8 sets the search interval to $(0.2s, 1.8s)$ where s is the 'start' value.

`details$knownmarks` = FALSE causes secr.fit to fit a zero-truncated sightings-only model that implicitly estimates the number of marked individuals, rather than inferring it from the number of rows in the capthist object.

`details$LLonly` = TRUE causes the function to returns a single evaluation of the log likelihood at the 'start' values.

`details$maxdistance` sets a limit to the centroid-to-mask distances considered. The centroid is the geometric mean of detection locations for each individual. If no limit is specified then summation is over all mask points. Specifying `maxdistance` can speed up computation; it is up to the user to select a limit that is large enough not to affect the likelihood (5σ?).

`details$miscparm` (default NULL) is an optional numeric vector of starting values for additional parameters used in a user-supplied distance function (see 'userdist' below). If the vector has a names attribute then the names will be used for the corresponding coefficients ('beta' parameters) which will otherwise be named 'miscparm1', miscparm2' etc. These parameters are constant across each model and do not appear in the model formula, but are estimated along with other coefficients when the likelihood is maximised. Any transformation (link function) etc. is handled by the user in the userdist function. The coefficients appear in the output from `coef.secr` and `vcov.secr`, but not `predict.secr`.

`details$newdetector` specifies a detector type to use for this fit, replacing the previous `detector(traps(capthist))`. The value may be a vector (one value per occasion) or for multi-session data, a list of vectors. A scalar value (e.g. "proximity") is otherwise used for all occasions and sessions. The true detector type is usually known and will be specified in the 'traps' attribute; `newdetector` is useful in simulation studies that examine the effect of misspecification. The capthist component of the output from secr.fit has the new type.

`details$nsim` specifies the number of replicate simulations to perform to estimate the overdispersion statistics for the sighting counts Tu and Tm. See also `details$chat` and `details$chatonly`.

`details$param` chooses between various parameterisations of the SECR model. The default `details$param = 0` is the formulation in Borchers and Efford (2008) and later papers.

`details$param = 1` was once used to select the Gardner & Royle parameterisation of the detection model (*p0*, σ; Gardner et al. 2009) when the detector type is 'multi'. This parameterisation was discontinued in 2.10.0.

`details$param = 2` selects parameterisation in terms of $(esa(g_0, \sigma), \sigma)$ (Efford and Mowat 2014).

`details$param = 3` selects parameterisation in terms of $(a_0(\lambda_0, \sigma), \sigma)$ (Efford and Mowat 2014). This parameterization is used automatically if a0 appears in the model (e.g., a0 ~ 1).

`details$param = 4` selects parameterisation of sigma in terms of the coefficient sigmak and constant c (sigma = sigmak / D^0.5 + c) (Efford et al. 2016). If c is not included explicitly in the model (e.g., c ~ 1) then it is set to zero. This parameterization is used automatically if sigmak appears in the model (e.g., sigmak ~ 1)

`details$param = 5` combines parameterisations (3) and (4) (first compute sigma from D, then compute lambda0 from sigma).

`details$relativeD` indicates that a density model has been fitted conditional on n and hence describing relative density instead of absolute density. The intercept of the submodel for D as a function of covariates is fixed at zero for a log link or 1 for the identity link. Not set by the user in secr >= 5.3.0.

`details$savecall` determines whether the full call to `secr.fit` is saved in the output object. The default is TRUE except when called by `list.secr.fit` as names in the call are then evaluated, causing the output to become unwieldy.

`details$saveprogress` is a non-negative integer indicating the frequency with which progress is saved to the RDS file named in details$progressfilename (overwritten each time). Logical values FALSE and TRUE correspond to 0 (never) and 1 (after each likelihood evaluation). Less frequent saving is advised, as the operation is slow. If fitting is terminated early it may be resumed from the saved file with function `secr.refit`. The value saved to the RDS file is a list of inputs for 'secr.fit'. The list has attribute 'log' that is a dataframe in which each row is a vector with the iteration number, log likelihood, time, and current values of the coefficients.

`details$progressfilename` is the name of an RDS file for 'saveprogress' (see previous). Ignored when saveprogress = 0. Default "progress.RDS".

`details$safeLL` uses the 'log-sum-exp' algorithm to avoid numeric underflow in C++ code for the likelihood. Computation with safeLL = TRUE is slightly slower so the default is safeLL = FALSE, except that it is TRUE when `capthist` includes telemetry. Used together with 'uselog' for telemetry data (with many fixes, as a rule of thumb more than a few tens, the likelihood otherwise underflows and the fit fails).

`details$splitmarked` determines whether the home range centre of marked animals is allowed to move between the marking (markocc = 1) and sighting (markocc = 0) phases of a spatial capture–mark–resight study. The default is to assume a common home-range centre (`splitmarked = FALSE`).

`details$telemetryint` selects the method for integrating over the activity centres of telemetered animals. The default `"mask"` sums over mask cells, which requires a mask spacing small relative to $\sigma/\sqrt{n}$ for an animal with n fixes. The alternative `"GH"` uses Gauss-Hermite quadrature centred on the mean of the animal's fixes; it is faster and independent of mask spacing, but places the quadrature nodes without regard to the mask, so there is no restriction to habitat in the mask. Currently available only for telemetry type 'marking' (for which the prior for the activity centre of a marked animal is uniform, whatever the method).

`details$telemetryscale` defunct option, superceded by 'safeLL' and 'uselog'.

`details$telemetrytype` defunct from 3.0.0 (2017)

`details$usecov` selects the mask covariate to be used for normalization. NULL limits denominator for normalization to distinguishing habitat from non-habitat.

`details$uselog` selects robust (log sum) likelihood that avoids numeric underflow especially with telemetry data, where it makes 'telemetryscale' obsolete. Computation of other likelihoods is slower so the default is FALSE, except that it is TRUE when `capthist` includes telemetry (as for safeLL).

`details$userDfn` is a user-provided function for modelling a density surface (see Appendix I of The SECR book). The function has arguments `Dbeta, mask, ngroup, nsession` and returns an array with dimensions `c(nrow(mask), ngroup, nsession)`. When the session-specific masks are identical (same cells and covariates) it is called once with `nsession` equal to the number of sessions. Otherwise it is called once per session with that session's mask and `nsession = 1`. A function may

also declare the optional argument `sessioncov`, which is then supplied. This is a copy of the `sessioncov` argument of `secr.fit` restricted to the sessions in the call (one row per session), with the automatic covariates `session` (factor) and `Session` (0, 1, ..., nsessions-1) added if not already present. A function that returns a single session (an array with last dimension 1) is used for every session, as in earlier versions.

`details$userdist` is either a function to compute non-Euclidean distances between detectors and mask points, or a pre-computed matrix of such distances. The first two arguments of the function should be 2-column matrices of x-y coordinates (respectively k detectors and m mask points). The third argument is a habitat mask that defines a non-Euclidean habitat geometry (a linear geometry is described in documentation for the package 'secrlinear'). The matrix returned by the function must have exactly k rows and m columns. When called with no arguments the function should return a character vector of names for the required covariates of 'mask', possibly including the dynamically computed density 'D' and a parameter 'noneuc' that will be fitted. A slightly expanded account is at userdist, and full documentation is in the separate document secr-noneuclidean.pdf.

Do not use 'userdist' for polygon or transect detectors

References

Efford, M. G., Dawson, D. K., Jhala, Y. V. and Qureshi, Q. (2016) Density-dependent home-range size revealed by spatially explicit capture–recapture. *Ecography* **39**, 676–688.

Efford, M. G. and Mowat, G. (2014) Compensatory heterogeneity in capture–recapture data.*Ecology* **95**, 1341–1348.

Gardner, B., Royle, J. A. and Wegan, M. T. (2009) Hierarchical models for estimating density from DNA mark-recapture studies. *Ecology* **90**, 1106–1115.

Royle, J. A., Chandler, R. B., Sun, C. C. and Fuller, A. K. (2013) Integrating resource selection information with spatial capture–recapture. *Methods in Ecology and Evolution* **4**, 520–530.

See Also

`secr.fit` , userdist

Examples

```
## Not run:

## Demo of miscparm and userdist
## We fix the usual 'sigma' parameter and estimate the same
## quantity as miscparm[1]. Differences in CI reflect the implied use
## of the identity link for miscparm[1].

mydistfn3 <- function (xy1,xy2, mask) {
    if (missing(xy1)) return(character(0))
    xy1 <- as.matrix(xy1)
    xy2 <- as.matrix(xy2)
    miscparm <- attr(mask, 'miscparm')
    distmat <- edist(xy1,xy2) / miscparm[1]
    distmat
}
```

```
fit0 <- secr.fit (captdata)
fit <- secr.fit (captdata, fixed = list(sigma=1), details =
    list(miscparm = c(sig = 20), userdist = mydistfn3))
predict(fit0)
coef(fit)


## End(Not run)
```

`detectfn` *Detection Functions*

Description

A detection function relates the probability of detection g or the expected number of detections λ for an animal to the distance of a detector from a point usually thought of as its home-range centre. In **secr** only simple 2- or 3-parameter functions are used. Each type of function is identified by its number or by a 2–3 letter code (version $\geq$ 2.6.0; see below). In most cases the name may also be used (as a quoted string).

Choice of detection function is usually not critical, and the default 'HN' is usually adequate.

Functions (14)–(19) are parameterised in terms of the expected number of detections λ, or cumulative hazard, rather than probability. 'Exposure' (e.g. Royle and Gardner 2011) is another term for cumulative hazard. This parameterisation is natural for the 'count' detector type or if the function is to be interpreted as a distribution of activity (home range). When one of the functions (14)–(19) is used to describe detection probability (i.e., for the binary detectors 'single', 'multi','proximity','polygonX' or 'transectX'), the expected number of detections is internally transformed to a binomial probability using $g(d) = 1 - \exp(-\lambda(d))$.

The hazard halfnormal (14) is similar to the halfnormal exposure function used by Royle and Gardner (2011) except they omit the factor of 2 on σ^2, which leads to estimates of σ that are larger by a factor of sqrt(2). The hazard exponential (16) is identical to their exponential function.

Code	Name	Parameters	Function
0 HN	halfnormal	g0, sigma	$g(d) = g_0 \exp\left(\frac{-d^2}{2\sigma^2}\right)$
1 HR	hazard rate	g0, sigma, z	$g(d) = g_0[1 - \exp\{-({}^{d}/_{\sigma})^{-z}\}]$
2 EX	exponential	g0, sigma	$g(d) = g_0 \exp\{-({}^{d}/_{\sigma})\}$
3 CHN	compound halfnormal	g0, sigma, z	$g(d) = g_0[1 - \{1 - \exp\left(\frac{-d^2}{2\sigma^2}\right)\}^z]$
4 UN	uniform	g0, sigma	$g(d) = g_0, d <= \sigma; g(d) = 0, \text{otherwise}$
5 WEX	w exponential	g0, sigma, w	$g(d) = g_0, d < w; g(d) = g_0 \exp\left(-\frac{d-w}{\sigma}\right), \text{otherwise}$
6 ANN	annular normal	g0, sigma, w	$g(d) = g_0 \exp\{\frac{-(d-w)^2}{2\sigma^2}\}$
7 CLN	cumulative lognormal	g0, sigma, z	$g(d) = g_0[1 - F\{(d-\mu)/s\}]$
8 CG	cumulative gamma	g0, sigma, z	$g(d) = g_0\{1 - G(d; k, \theta)\}$
9 BSS	binary signal strength	b0, b1	$g(d) = 1 - F\{-(b_0 + b_1 d)\}$
10 SS	signal strength	beta0, beta1, sdS	$g(d) = 1 - F[\{c - (\beta_0 + \beta_1 d)\}/s]$
11 SSS	signal strength spherical	beta0, beta1, sdS	$g(d) = 1 - F[\{c - (\beta_0 + \beta_1(d-1) - 10\log_{10} d^2)\}/s]$

14 HHN	hazard halfnormal	lambda0, sigma	$\lambda(d) = \lambda_0 \exp\left(\frac{-d^2}{2\sigma^2}\right)$; $g(d) = 1 - \exp(-\lambda(d))$
15 HHR	hazard hazard rate	lambda0, sigma, z	$\lambda(d) = \lambda_0(1 - \exp\{-({}^d/_\sigma)^{-z}\})$; $g(d) = 1 - \exp(-\lambda(d)$
16 HEX	hazard exponential	lambda0, sigma	$\lambda(d) = \lambda_0 \exp\{-({}^d/_\sigma)\}$; $g(d) = 1 - \exp(-\lambda(d))$
17 HAN	hazard annular normal	lambda0, sigma, w	$\lambda(d) = \lambda_0 \exp\{\frac{-(d-w)^2}{2\sigma^2}\}$; $g(d) = 1 - \exp(-\lambda(d))$
18 HCG	hazard cumulative gamma	lambda0, sigma, z	$\lambda(d) = \lambda_0\{1 - G(d; k, \theta)\}$; $g(d) = 1 - \exp(-\lambda(d))$
19 HVP	hazard variable power	lambda0, sigma, z	$\lambda(d) = \lambda_0 \exp\{-({}^d/_\sigma)^z\}$; $g(d) = 1 - \exp(-\lambda(d))$
20 OU	Ornstein-Uhlenbeck movement	epsilon, sigma, tau	See `simOU`

Functions (1) and (15), the "hazard-rate" detection functions described by Hayes and Buckland (1983), are not recommended for SECR because of their long tail, and care is also needed with (2) and (16).

Function (3), the compound halfnormal, follows Efford and Dawson (2009).

Function (4) uniform is defined only for simulation as it poses problems for likelihood maximisation by gradient methods. Uniform probability implies uniform hazard, so there is no separate function 'HUN'.

For function (7), 'F' is the standard normal distribution function and μ and s are the mean and standard deviation on the log scale of a latent variable representing a threshold of detection distance. See Note for the relationship to the fitted parameters sigma and z.

For functions (8) and (18), 'G' is the cumulative distribution function of the gamma distribution with shape parameter k (= `z`) and scale parameter θ (= `sigma/z`). See R's `pgamma`.

For functions (9), (10) and (11), 'F' is the standard normal distribution function and c is an arbitrary signal threshold. The two parameters of (9) are functions of the parameters of (10) and (11): $b_0 = (\beta_0 - c)/sdS$ and $b_1 = \beta_1/s$ (see Efford et al. 2009). Note that (9) does not require signal-strength data or c.

Function (11) includes an additional 'hard-wired' term for sound attenuation due to spherical spreading. Detection probability at distances less than 1 m is given by $g(d) = 1 - F\{(c - \beta_0)/sdS\}$

Functions (12) and (13) are undocumented methods for sound attenuation.

Function (19) has been used in some published papers and is included for comparison (see e.g. Ergon and Gardner 2014).

Function (20) defines a detection model based on a bivariate Ornstein-Uhlenbeck movement model (mean-reverting random walk). Like the uniform (4), it is used in **secr** only for simulation. The three parameters are epsilon (radius of a detector within which detection occurs), sigma (asymptotic bivariate normal scale, as in HHN) and tau (serial correlation parameter $\tau = 1/\beta$). See `simOU`.

Note

The parameters of function (7) are potentially confusing. The fitted parameters describe a latent threshold variable on the natural scale: sigma (mean) = $\exp(\mu + s^2/2)$ and z (standard deviation) = $\sqrt{\exp(s^2 + 2\mu)(\exp(s^2) - 1)}$. As with other detection functions, sigma is a spatial scale parameter, although in this case it corresponds to the mean of the threshold variable; the standard deviation of the threshold variable (z) determines the shape (roughly 1/max(slope)) of the detection function.

References

Efford, M. G. and Dawson, D. K. (2009) Effect of distance-related heterogeneity on population size estimates from point counts. *Auk* **126**, 100–111.

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.

Ergon, T. and Gardner, B. (2014) Separating mortality and emigration: modelling space use, dispersal and survival with robust-design spatial capture–recapture data. *Methods in Ecology and Evolution* **5**, 1327–1336.

Hayes, R. J. and Buckland, S. T. (1983) Radial-distance models for the line-transect method. *Biometrics* **39**, 29–42.

Royle, J. A. and Gardner, B. (2011) Hierarchical spatial capture–recapture models for estimating density from trapping arrays. In: A.F. O'Connell, J.D. Nichols & K.U. Karanth (eds) *Camera Traps in Animal Ecology: Methods and Analyses*. Springer, Tokyo. Pp. 163–190.

See Also

`detectfnplot`

`detector` *Detector Type*

Description

Extract or replace the detector type.

Usage

```
detector(object, ...)
detector(object) <- value
```

Arguments

`object`	object with 'detector' attribute e.g. `traps`
`value`	character string for detector type
`...`	other arguments (not used)

Details

Valid detector types are 'single', 'multi', 'proximity', 'count', 'capped', 'signal', 'polygon', 'transect', 'polygonX', and 'transectX'. The detector type is stored as an attribute of a `traps` object. Detector types are mostly described by Efford et al. (2009a,b; see also secr-overview.pdf). Polygon and transect detector types are for area and linear searches as described in secr-polygondetectors.pdf and Efford (2011). The 'signal' detector type is used for acoustic data as described in secr-sound.pdf.

The 'capped' detector type refers to binary proximity data in which no more than one individual may be detected at a detector on any occasion. The type is partially implemented in **secr** 3.1.1: data may be simulated and manipulated, but for model fitting these are treated as proximity data by `secr.fit()`.

Value

character string for detector type

References

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009a) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

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

See Also

traps, RShowDoc

Examples

```
## Default detector type is "multi"
temptrap <- make.grid(nx = 6, ny = 8)
detector(temptrap) <- "proximity"
summary(temptrap)
```

---

deviance — *Deviance of fitted secr model and residual degrees of freedom*

Description

Compute the deviance or residual degrees of freedom of a fitted secr model, treating multiple sessions and groups as independent. The likelihood of the saturated model depends on whether the 'conditional' or 'full' form was used, and on the distribution chosen for the number of individuals observed (Poisson or binomial).

Usage

```
## S3 method for class 'secr'
deviance(object, ...)
## S3 method for class 'secr'
df.residual(object, ...)
```

Arguments

object	secr object from secr.fit
...	other arguments (not used)

Details

The deviance is $-2log(\hat{L}) + 2log(L_{sat})$, where $\hat{L}$ is the value of the log-likelihood evaluated at its maximum, and L_{sat} is the log-likelihood of the saturated model, calculated thus:

Likelihood conditional on n -

$L_{sat} = \log(n!) + \sum_{\omega}[n_\omega \log(\frac{n_\omega}{n}) - \log(n_\omega!)]$

Full likelihood, Poisson n -

$L_{sat} = n\log(n) - n + \sum_{\omega}[n_\omega \log(\frac{n_\omega}{n}) - \log(n_\omega!)]$

Full likelihood, binomial n -

$L_{sat} = n\log(\frac{n}{N}) + (N-n)\log(\frac{N-n}{N}) + \log(\frac{N!}{(N-n)!}) + \sum_{\omega}[n_\omega \log(\frac{n_\omega}{n}) - \log(n_\omega!)]$

n is the number of individuals observed at least once, n_ω is the number of distinct histories, and N is the number in a chosen area A that we estimate by $\hat{N} = \hat{D}A$.

The residual degrees of freedom is the number of distinct detection histories minus the number of parameters estimated. The detection histories of two animals are always considered distinct if they belong to different groups.

When samples are (very) large the deviance is expected to be distributed as χ^2 with $n_\omega - p$ degrees of freedom when p parameters are estimated. In reality, simulation is needed to assess whether a given value of the deviance indicates a satisfactory fit, or to estimate the overdispersion parameter c. `sim.secr` is a convenient tool.

Value

The scalar numeric value of the deviance or the residual degress of freedom extracted from the fitted model.

References

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

See Also

`secr.fit`, `sim.secr`

Examples

```
deviance(secrdemo.0)
df.residual(secrdemo.0)
```

`discretize` *Rasterize Area Search or Transect Data*

Description

It is sometimes useful to re-cast area-search (polygon or polygonX) data as if it was from a set of closely spaced point detectors, i.e. to rasterize the detection locations. This function makes that conversion. Each polygon detector in the input is replaced by a number of point detectors, each representing a square pixel. Detections are mapped to the new detectors on the basis of their x-y coordinates.

If `object` contains transect data the problem is passed to `snip` and `reduce.capthist`.

Usage

```
discretize(object, spacing = 5, outputdetector = c("proximity", "count", "multi"),
    tol = 0.001, cell.overlap = FALSE, type = c("centre","any", "all"), ...)
```

Arguments

`object`	secr capthist or traps object
`spacing`	numeric spacing between point detectors in metres
`outputdetector`	character output detector type
`tol`	numeric fractional inflation of perimeter (see Details)
`cell.overlap`	logical; if TRUE the area of overlap is stored in usage attribute
`type`	character; see Details
`...`	other arguments passed to `snip` if object is transect

Details

The input should have detector type 'polygon' or 'polygonX'.

A new array of equally spaced detectors is generated within each polygon of the input, inflated radially by 1 + tol to avoid some inclusion problems. The origin of the superimposed grid is fixed automatically. If `type = "centre"` detectors are included if they lie within the (inflated) polygon. Otherwise, the decision on whether to include a candidate new detector is based on the corner vertices of the cell around the detector (side = `spacing`); `type = "any"` and `type = "all"` have the obvious meanings.

`tol` may be negative, in which case the array(s) will be shrunk relative to the polygon(s).

For irregular polygons the edge cells in the output may be only partially contained within the polygon they represent. Set `cell.overlap = TRUE` to retain the proportion of overlap as the 'usage' of the new traps object. This can take a few seconds to compute. If 'usage' is already defined then the new 'usage' is the old multiplied by the proportion of overlap.

Combining `type = "any"` and `cell.overlap = TRUE` with `tol` > 0 can have the odd effect of including some marginal detectors that are assigned zero usage.

With `type = "any"`, the sum of the overlap proportions times cell area is equal to the area of the polygons.

Value

A capthist or traps object of the requested detector type, but otherwise carrying forward all attributes of the input. The embedded traps object has a factor covariate 'polyID' recording the polygon to which each point detector relates.

Note

Consider the likely number of detectors in the output before you start.

See Also

`reduce.capthist`, `snip`

Examples

```
## Not run:

## generate some polygon data
pol <- make.poly()
CH <- sim.capthist(pol, popn = list(D = 30), detectfn = 'HHN',
   detectpar = list(lambda0 = 0.3))
plot(CH, border = 10, gridl = FALSE, varycol = FALSE)

## discretize and plot
CH1 <- discretize(CH, spacing = 10, output = 'count')
plot(CH1, add = TRUE, cappar = list(col = 'orange'), varycol =
    FALSE, rad = 0)
plot(traps(CH1), add = TRUE)
# overlay cell boundaries
plot(as.mask(traps(CH1)), dots = FALSE, col = NA, meshcol = 'green',
    add = TRUE)

## show how detections are snapped to new detectors
newxy <- traps(CH1)[nearesttrap(xy(CH),traps(CH1)),]
segments(xy(CH)[,1], xy(CH)[,2], newxy[,1], newxy[,2])

plot(traps(CH), add = TRUE)  # original polygon

## Incomplete overlap

pol <- rotate(make.poly(), 45)
CH2 <- sim.capthist(pol, popn = list(D = 30), detectfn = 'HHN',
   detectpar = list(lambda0 = 0.3))
plot(CH2, border = 10, gridl = FALSE, varycol = FALSE)
CH3 <- discretize(CH2, spacing = 10, output = 'count', type = 'any',
    cell.overlap = TRUE, tol=0.05)

plot(CH3, add = TRUE, cappar = list(col = 'orange'), varycol =
```

```
    FALSE, rad = 0)
plot(traps(CH3), add = TRUE)

# overlay cell boundaries and usage
msk <- as.mask(traps(CH3))
covariates(msk) <- data.frame(usage = usage(traps(CH3))[,1])
plot(msk, dots = FALSE, cov='usage', meshcol = 'green',
    add = TRUE)


## End(Not run)
```

---

distancetotrap *Distance To Nearest Detector*

Description

Compute Euclidean distance from each of a set of points to the nearest detector in an array, or return the sequence number of the detector nearest each point.

Usage

```
distancetotrap(X, traps)

nearesttrap(X, traps)
```

Arguments

X	coordinates
traps	traps object or 2-column matrix of coordinates

Details

distancetotrap returns the distance from each point in X to the nearest detector in traps. It may be used to restrict the points on a habitat mask.

For traps objects with polygon detector type (polygon, polygonX), and for SpatialPolygons, the function sf::st_distance is used internally(from **secr** 4.5.2).

Value

distancetotrap returns a vector of distances (assumed to be in metres).

nearesttrap returns the index of the nearest trap.

See Also

make.mask

Examples

```
## restrict a habitat mask to points within 70 m of traps
## this is nearly equivalent to using make.mask with the
## `trapbuffer' option
temptrap <- make.grid()
tempmask <- make.mask(temptrap)
d <- distancetotrap(tempmask, temptrap)
tempmask <- subset(tempmask, d < 70)
```

`Dsurface` *Density Surfaces*

Description

S3 class for rasterized fitted density surfaces. A Dsurface is a type of 'mask' with covariate(s) for the predicted density at each point.

Usage

```
## S3 method for class 'Dsurface'
print(x, scale = 1, ...)
## S3 method for class 'Dsurface'
summary(object, scale = 1, ...)
```

Arguments

`x, object`	Dsurface object to display
`scale`	numeric multiplier for density
`...`	other arguments passed to print method for data frames or summary method for masks

Details

A Dsurface will usually have been constructed with `predictDsurface`.

The 'scale' argument may be used to change the units of density from the default (animals / hectare) to animals / km^2 (scale = 100) or animals / 100km^2 (scale = 10000).

A virtual S4 class 'Dsurface' is defined to allow the definition of a method for the generic function `raster` from the **raster** package.

See Also

`predictDsurface`, `plot.Dsurface`

`ellipse.secr` *Confidence Ellipses*

Description

Plot joint confidence ellipse for two parameters of secr model, or for a distribution of points.

Usage

```
ellipse.secr(object, par = c("g0", "sigma"), alpha = 0.05,
    npts = 100, plot = TRUE, linkscale = TRUE, add = FALSE,
    col = palette(), ...)

ellipse.bvn(xy, alpha = 0.05, npts = 100, centroid = TRUE, add = FALSE, ...)
```

Arguments

`object`	`secr` object output from `secr.fit`
`par`	character vector of length two, the names of two 'beta' parameters
`alpha`	alpha level for confidence intervals
`npts`	number of points on perimeter of ellipse
`plot`	logical for whether ellipse should be plotted
`linkscale`	logical; if FALSE then coordinates will be backtransformed from the link scale
`add`	logical to add ellipse to an existing plot
`col`	vector of one or more plotting colours
`...`	arguments to pass to plot functions (or polygon() in the case of ellipse.bvn)
`xy`	2-column matrix of coordinates
`centroid`	logical; if TRUE the plotted ellipse is a confidence region for the centroid of points in `xy`

Details

`ellipse.secr` calculates coordinates of a confidence ellipse from the asymptotic variance-covariance matrix of the beta parameters (coefficients), and optionally plots it.

If `linkscale` == FALSE, the inverse of the appropriate link transformation is applied to the coordinates of the ellipse, causing it to deform.

If `object` is a list of secr models then one ellipse is constructed for each model. Colours are recycled as needed.

`ellipse.bvn` plots a bivariate normal confidence ellipse for the centroid of a 2-dimensional distribution of points (default centroid = TRUE), or a Jennrich and Turner (1969) elliptical home-range model.

Value

A list containing the x and y coordinates is returned invisibly from either function.

References

Jennrich, R. I. and Turner, F. B. (1969) Measurement of non-circular home range. *Journal of Theoretical Biology*, **22**, 227–237.

Examples

```
ellipse.secr(secrdemo.0)
```

---

empirical.varD *Empirical Variance of H-T Density Estimate*

Description

Compute Horvitz-Thompson-like estimate of population density from a previously fitted spatial detection model, and estimate its sampling variance using the empirical spatial variance of the number observed in replicate sampling units. Wrapper functions are provided for several different scenarios, but all ultimately call `derivednj`. The function `derived` also computes Horvitz-Thompson-like estimates, but it assumes a Poisson or binomial distribution of total number when computing the sampling variance.

An alternative and probably more general solution is got by combining `chat.nj` and `adjustVarD`.

Usage

```
derivednj ( nj, esa, se.esa = NULL, method = c("SRS", "R2", "R3", "local",
    "poisson", "binomial"), xy = NULL, alpha = 0.05, loginterval = TRUE,
    area = NULL, independent.esa = FALSE )

derivedMash ( object, sessnum = NULL,  method = c("SRS", "local"),
    alpha = 0.05, loginterval = TRUE)

derivedCluster ( object, method = c("SRS", "R2", "R3", "local", "poisson", "binomial"),
    alpha = 0.05, loginterval = TRUE)

derivedSession ( object,  method = c("SRS", "R2", "R3", "local", "poisson", "binomial"),
   xy = NULL, alpha = 0.05, loginterval = TRUE, area = NULL, independent.esa = FALSE )

derivedExternal ( object, sessnum = NULL, nj, cluster, buffer = 100,
    mask = NULL, noccasions = NULL,  method = c("SRS", "local"), xy = NULL,
    alpha = 0.05, loginterval = TRUE)
```

Arguments

`object`	fitted secr model
`nj`	vector of number observed in each sampling unit (cluster)
`esa`	estimate of effective sampling area ($\hat{a}$)
`se.esa`	estimated standard error of effective sampling area ($\widehat{SE}(\hat{a})$)
`method`	character string 'SRS' or 'local'
`xy`	dataframe of x- and y- coordinates (`method = "local"` only)
`alpha`	alpha level for confidence intervals
`loginterval`	logical for whether to base interval on log(N)
`area`	area of region for method = "binomial" (hectares)
`independent.esa`	logical; controls variance contribution from esa (see Details)
`sessnum`	index of session in object$capthist for which output required
`cluster`	'traps' object for a single cluster
`buffer`	width of buffer in metres (ignored if `mask` provided)
`mask`	mask object for a single cluster of detectors
`noccasions`	number of occasions (for `nj`)

Details

`derivednj` accepts a vector of counts (`nj`), along with $\hat{a}$ and $\widehat{SE}(\hat{a})$. The argument `esa` may be a scalar or (if se.esa is NULL) a 2-column matrix with $\hat{a_j}$ and $\widehat{SE}(\hat{a_j})$ for each replicate j (row). In the special case that `nj` is of length 1, or `method` takes the values 'poisson' or 'binomial', the variance is computed using a theoretical variance rather than an empirical estimate. The value of `method` corresponds to 'distribution' in `derived`, and defaults to 'poisson'. For `method = 'binomial'` you must specify `area` (see Examples).

If `independent.esa` is TRUE then independence is assumed among cluster-specific estimates of esa, and esa variances are summed. The default is a weighted sum leading to higher overall variance.

`derivedCluster` accepts a model fitted to data from clustered detectors; each cluster is interpreted as a replicate sample. It is assumed that the sets of individuals sampled by different clusters do not intersect, and that all clusters have the same geometry (spacing, detector number etc.).

`derivedMash` accepts a model fitted to clustered data that have been 'mashed' for fast processing (see `mash`); each cluster is a replicate sample: the function uses the vector of cluster frequencies (n_j) stored as an attribute of the mashed `capthist` by `mash`.

`derivedExternal` combines detection parameter estimates from a fitted model with a vector of frequencies `nj` from replicate sampling units configured as in `cluster`. Detectors in `cluster` are assumed to match those in the fitted model with respect to type and efficiency, but sampling duration (`noccasions`), spacing etc. may differ. The `mask` should match `cluster`; if `mask` is missing, one will be constructed using the `buffer` argument and defaults from `make.mask`.

`derivedSession` accepts a single fitted model that must span multiple sessions; each session is interpreted as a replicate sample.

Spatial variance is calculated by one of these methods

Method	Description
"SRS"	simple random sampling with identical clusters
"R2"	variable cluster size cf Thompson (2002:70) estimator for line transects
"R3"	variable cluster size cf Buckland et al. (2001)
"local"	neighbourhood variance estimator (Stevens and Olsen 2003) SUSPENDED in 4.4.7
"poisson"	theoretical (model-based) variance
"binomial"	theoretical (model-based) variance in given area

The weighted options R2 and R3 substitute $\hat{a}_j$ for line length l_k in the corresponding formulae of Fewster et al. (2009, Eq 3,5). Density is estimated by $D = n/A$ where $A = \sum a_j$. The variance of A is estimated as the sum of the cluster-specific variances, assuming independence among clusters. Fewster et al. (2009) found that an alternative estimator for line transects derived by Thompson (2002) performed better when there were strong density gradients correlated with line length (R2 in Fewster et al. 2009, Eq 3).

The neighborhood variance estimator is implemented in package **spsurvey** and was originally proposed for generalized random tessellation stratified (GRTS) samples. For 'local' variance estimates, the centre of each replicate must be provided in xy, except where centres may be inferred from the data. It is unclear whether 'local' can or should be used when clusters vary in size.

derivedSystematic, now defunct, was an experimental function in earlier versions of **secr**.

Value

Dataframe with one row for each derived parameter ('esa', 'D') and columns as below

estimate	estimate of derived parameter
SE.estimate	standard error of the estimate
lcl	lower 100(1–alpha)% confidence limit
ucl	upper 100(1–alpha)% confidence limit
CVn	relative SE of number observed (across sampling units)
CVa	relative SE of effective sampling area
CVD	relative SE of density estimate

Note

The variance of a Horvitz-Thompson-like estimate of density may be estimated as the sum of two components, one due to uncertainty in the estimate of effective sampling area ($\hat{a}$) and the other due to spatial variance in the total number of animals n observed on J replicate sampling units ($n = \sum_{j=1}^{J} n_j$). We use a delta-method approximation that assumes independence of the components:

$$\widehat{\mathrm{var}}(\hat{D}) = \hat{D}^2\{\frac{\widehat{\mathrm{var}}(n)}{n^2} + \frac{\widehat{\mathrm{var}}(\hat{a})}{\hat{a}^2}\}$$

where $\widehat{\mathrm{var}}(n) = \frac{J}{J-1}\sum_{j=1}^{J}(n_j - n/J)^2$. The estimate of $\mathrm{var}(\hat{a})$ is model-based while that of $\mathrm{var}(n)$ is design-based. This formulation follows that of Buckland et al. (2001, p. 78) for conventional distance sampling. Given sufficient independent replicates, it is a robust way to allow for unmodelled spatial overdispersion.

There is a complication in SECR owing to the fact that $\hat{a}$ is a derived quantity (actually an integral) rather than a model parameter. Its sampling variance $\mathrm{var}(\hat{a})$ is estimated indirectly in **secr** by combining the asymptotic estimate of the covariance matrix of the fitted detection parameters θ with a numerical estimate of the gradient of $a(\theta)$ with respect to θ. This calculation is performed in derived.

References

Buckland, S. T., Anderson, D. R., Burnham, K. P., Laake, J. L., Borchers, D. L. and Thomas, L. (2001) *Introduction to Distance Sampling: Estimating Abundance of Biological Populations*. Oxford University Press, Oxford.

Fewster, R. M. (2011) Variance estimation for systematic designs in spatial surveys. *Biometrics* **67**, 1518–1531.

Fewster, R. M., Buckland, S. T., Burnham, K. P., Borchers, D. L., Jupp, P. E., Laake, J. L. and Thomas, L. (2009) Estimating the encounter rate variance in distance sampling. *Biometrics* **65**, 225–236.

Stevens, D. L. Jr and Olsen, A. R. (2003) Variance estimation for spatially balanced samples of environmental resources. *Environmetrics* **14**, 593–610.

Thompson, S. K. (2002) *Sampling*. 2nd edition. Wiley, New York.

See Also

derived, esa, chat.nj, adjustVarD

Examples

```
## The `ovensong' data are pooled from 75 replicate positions of a
## 4-microphone array. The array positions are coded as the first 4
## digits of each sound identifier. The sound data are initially in the
## object `signalCH'. We first impose a 52.5 dB signal threshold as in
## Dawson & Efford (2009, J. Appl. Ecol. 46:1201--1209). The vector nj
## includes 33 positions at which no ovenbird was heard. The first and
## second columns of `temp' hold the estimated effective sampling area
## and its standard error.

## Not run:

signalCH.525 <- subset(signalCH, cutval = 52.5)
nonzero.counts <- table(substring(rownames(signalCH.525),1,4))
nj <- c(nonzero.counts, rep(0, 75 - length(nonzero.counts)))
temp <- derived(ovensong.model.1, se.esa = TRUE)
derivednj(nj, temp["esa",1:2])

## The result is very close to that reported by Dawson & Efford
## from a 2-D Poisson model fitted by maximizing the full likelihood.

## If nj vector has length 1, a theoretical variance is used...
msk <- ovensong.model.1$mask
A <- nrow(msk) * attr(msk, "area")
derivednj (sum(nj), temp["esa",1:2], method = "poisson")
```

```
derivednj (sum(nj), temp["esa",1:2], method = "binomial", area = A)

## Set up an array of small (4 x 4) grids,
## simulate a Poisson-distributed population,
## sample from it, plot, and fit a model.
## mash() condenses clusters to a single cluster

testregion <- data.frame(x = c(0,2000,2000,0),
    y = c(0,0,2000,2000))
t4 <- make.grid(nx = 4, ny = 4, spacing = 40)
t4.16 <- make.systematic (n = 16, cluster = t4,
    region = testregion)
popn1 <- sim.popn (D = 5, core = testregion,
    buffer = 0)
capt1 <- sim.capthist(t4.16, popn = popn1)
fit1 <- secr.fit(mash(capt1), CL = TRUE, trace = FALSE)

## Visualize sampling
tempmask <- make.mask(t4.16, spacing = 10, type =
    "clusterbuffer")
plot(tempmask)
plot(t4.16, add = TRUE)
plot(capt1, add = TRUE)

## Compare model-based and empirical variances.
## Here the answers are similar because the data
## were simulated from a Poisson distribution,
## as assumed by \code{derived}

derived(fit1)
derivedMash(fit1)

## Now simulate a patchy distribution; note the
## larger (and more credible) SE from derivedMash().

popn2 <- sim.popn (D = 5, core = testregion, buffer = 0,
    model2D = "hills", details = list(hills = c(-2,3)))
capt2 <- sim.capthist(t4.16, popn = popn2)
fit2 <- secr.fit(mash(capt2), CL = TRUE, trace = FALSE)
derived(fit2)
derivedMash(fit2)

## The detection model we have fitted may be extrapolated to
## a more fine-grained systematic sample of points, with
## detectors operated on a single occasion at each...
## Total effort 400 x 1 = 400 detector-occasions, compared
## to 256 x 5 = 1280 detector-occasions for initial survey.

t1 <- make.grid(nx = 1, ny = 1)
t1.100 <- make.systematic (cluster = t1, spacing = 100,
    region = testregion)
capt2a <- sim.capthist(t1.100, popn = popn2, noccasions = 1)
## one way to get number of animals per point
```

```
nj <- attr(mash(capt2a), "n.mash")
derivedExternal (fit2, nj = nj, cluster = t1, buffer = 100,
    noccasions = 1)

## Review plots
base.plot <- function() {
    MASS::eqscplot( testregion, axes = FALSE, xlab = "",
        ylab = "", type = "n")
    polygon(testregion)
}
par(mfrow = c(1,3), xpd = TRUE, xaxs = "i", yaxs = "i")
base.plot()
plot(popn2, add = TRUE, col = "blue")
mtext(side=3, line=0.5, "Population", cex=0.8, col="black")
base.plot()
plot (capt2a, add = TRUE,title = "Extensive survey")
base.plot()
plot(capt2, add = TRUE, title = "Intensive survey")
par(mfrow = c(1,1), xpd = FALSE, xaxs = "r", yaxs = "r")  ## defaults


## Weighted variance

derivedSession(ovenbird.model.1, method = "R2")


## End(Not run)
```

---

esaPlot *Mask Buffer Diagnostic Plot*

Description

Plot effective sampling area (Borchers and Efford 2008) as a function of increasing buffer width.

`esaPlot` was previously called `esa.plot`.

Usage

```
esaPlot (object, max.buffer = NULL, spacing = NULL, max.mask = NULL,
    detectfn, detectpar, noccasions, binomN = NULL, thin = 0.1,
    poly = NULL, poly.habitat = TRUE, session = 1, plt = TRUE,
    type = c('density', 'esa', 'meanpdot', 'CVpdot'), n = 1, add = FALSE,
    overlay = TRUE, conditional = FALSE, ...)
```

Arguments

object	traps object or secr object output from secr.fit
max.buffer	maximum width of buffer in metres
spacing	distance between mask points
max.mask	mask object
detectfn	integer code or character string for shape of detection function 0 = halfnormal etc. – see detectfn
detectpar	list of values for named parameters of detection function
noccasions	number of sampling occasions
binomN	integer code for discrete distribution (see secr.fit)
thin	proportion of mask points to retain in plot and output
poly	matrix of two columns interpreted as the x and y coordinates of a bounding polygon (optional)
poly.habitat	logical as in make.mask
session	vector of session indices (used if object spans multiple sessions)
plt	logical to plot results
type	character, what to plot
n	integer number of distinct individuals detected
add	logical to add line to an existing plot
overlay	logical; if TRUE then automatically add = TRUE for plots after the first
conditional	logical; if TRUE the reported mean and CV are conditional on detection (see CVpdot)
...	graphical arguments passed to plot() and lines()

Details

Effective sampling area (esa) is defined as the integral of net capture probability ($p.(\mathbf{X})$) over a region. esaPlot shows the effect of increasing region size on the value of esa for fixed values of the detection parameters. The max.buffer or max.mask arguments establish the maximum extent of the region; points (cells) within this mask are sorted by their distance d_k from the nearest detector. esa(buffer) is defined as the cumulative sum of $cp.(\mathbf{X})$ for $d_k(\mathbf{X}) <= \text{buffer}$, where c is the area associated with each cell.

The default (type = 'density') is to plot the reciprocal of esa multiplied by n; this is on a more familiar scale (the density scale) and hence is easier to interpret.

Because esaPlot uses the criterion 'distance to nearest detector', max.mask should be constructed to include all habitable cells within the desired maximum buffer and no others. This is achieved with type = "trapbuffer" in make.mask. It is a good idea to set the spacing argument of make.mask rather than relying on the default based on nx. Spacing may be small (e.g. sigma/10) and the buffer of max.mask may be quite large (e.g. 10 sigma), as computation is fast.

Thinning serves to reduce redundancy in the plotted points, and (if the result is saved and printed) to generate more legible numerical output. Use thin=1 to include all points.

esaPlot calls the internal function esaPlotsecr when object is a fitted model. In this case detectfn, detectpar and noccasions are inferred from object.

Value

A dataframe with columns

buffer	buffer width
esa	computed effective sampling area
density	n/esa
pdot	$p.(\mathbf{X})$
pdotmin	cumulative minimum ($p.(\mathbf{X})$)
meanpdot	expected pdot across mask (see CVpdot)
CVpdot	CV of pdot across mask (see CVpdot)

If plt = TRUE the dataframe is returned invisibly.

Note

The response of effective sampling area to buffer width is just one possible mask diagnostic; it's fast, graphic, and often sufficient. mask.check performs more intensive checks, usually for a smaller number of buffer widths.

The old argument 'as.density' was superceded by 'type' in 3.1.7.

References

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

See Also

mask, pdot, CVpdot, make.mask, mask.check, Detection functions

Examples

```
## Not run:

## with previously fitted model
esaPlot(secrdemo.0)

## from scratch
trps <- make.grid()
msk <- make.mask(trps, buffer = 200, spacing = 5, type = "trapbuffer")
detectpar <- list(g0 = 0.2, sigma = 25)
esaPlot(trps,,, msk, 0, detectpar, nocc = 10, col = "blue")
esaPlot(trps,,, msk, 0, detectpar, nocc = 5, col = "green",
    add = TRUE)

esaPlot(trps,,, msk, 0, detectpar, nocc = 5, thin = 0.002, plt = FALSE)


## End(Not run)
```

expected.n *Expected Number of Individuals*

Description

Computes the expected number of individuals detected across a detector layout or at each cluster of detectors.

Usage

```
expected.n(object, session = NULL, group = NULL, bycluster
    = FALSE, splitmask = FALSE, ncores = NULL, markingonly = FALSE)
```

Arguments

object	secr object output from secr.fit
session	character session vector
group	group – for future use
bycluster	logical to output the expected number for clusters of detectors rather than whole array
splitmask	logical for computation method (see Details)
ncores	integer number of threads to be used for parallel processing
markingonly	logical; if TRUE then sighting and telemetry occasions are excluded

Details

The expected number of individuals detected is $E(n) = \int p.(X)D(X)dX$ where the integration is a summation over object$mask. $p.(X)$ is the probability an individual at X will be detected at least once either on the whole detector layout (bycluster = FALSE) or on the detectors in a single cluster (see pdot for more on $p.$). $D(X)$ is the expected density at X, given the model. $D(X)$ is constant (i.e. density surface flat) if object$CL == TRUE or object$model$D == ~1, and for some other possible models.

If the bycluster option is selected and detectors are not, in fact, assigned to clusters then each detector will be treated as a cluster, with a warning.

Setting ncores = NULL uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see setNumThreads).

By default, a full habitat mask is used for each cluster. This is the more robust option. Alternatively, the mask may be split into subregions defined by the cells closest to each cluster.

The calculation takes account of any fitted continuous model for spatial variation in density (note Warning).

When sampling includes sighting and telemetry occasions it is probably more useful to exclude these, as allowed from version 5.5.0.

Value

The expected count (bycluster = FALSE) or a vector of expected counts, one per cluster. For multi-session data, a list of such vectors.

Warning

This function changed slightly between 2.1.0 and 2.1.1, and now performs as indicated here when bycluster = TRUE and clusters are not specified.

Clusters of detectors are assumed to be independent (always true with detector types 'proximity', 'count' etc.). The computed E(n) does not apply when there is competition among clusters of detectors.

The prediction of density at present considers only the base level of density covariates, such as cell-specific habitat variables.

See Also

region.N

Examples

```
## Not run:

expected.n(secrdemo.0)
expected.n(secrdemo.0, bycluster = TRUE)
expected.n(ovenbird.model.D)

## Clustered design
mini <- make.grid(nx = 3, ny = 3, spacing = 50, detector =
    "proximity")
tempgrids <- trap.builder (cluster = mini , method = "all",
    frame = expand.grid(x = seq(1000, 9000, 2000),
    y = seq(1000, 9000, 2000)), plt = TRUE)
capt <- sim.capthist(tempgrids, popn = list(D = 2))
tempmask <- make.mask(tempgrids, buffer = 100,
    type = "clusterbuffer")
fit <- secr.fit(capt, mask = tempmask, trace = FALSE)
En <- expected.n(fit, bycluster = TRUE)

## GoF or overdispersion statistic
p <- length(fit$fit$par)
y <- cluster.counts(capt)
## scaled by n-p
sum((y - En)^2 / En) / (length(En)-p)
sum((y - En)^2 / En) / sum(y/En)


## End(Not run)
```

`extractMoves` *Simulated Movements*

Description

Extract movements from a previously simulated multi-session population.

Usage

```
extractMoves(pop, plotn = NULL, add = FALSE, collapse = TRUE, maxradius = Inf, ...)
```

Arguments

`pop`	popn object from `sim.popn`
`plotn`	integer maximum number of instances to plot at each session
`add`	logical for whether to add to existing plot
`collapse`	logical; if TRUE plots for sessions 2, 3,... are added to the first
`maxradius`	numeric radius for selecting subset of initial locations
`...`	arguments passed to `arrows`

Details

This function is mostly used to check the movement simulations.

Moves are constrained by the edge (argument 'edgemethod' of `sim.popn`). 'maxradius' may be set to restrict the extraction to the subset of animals initially near the centroid of the arena in each session.

Plotting uses the graphics function `arrows` that has some quirks, such as difficult-to-suppress warnings for zero-length moves. Set `code = 0` to suppress arrowheads; `length = 0.1` to shorten to 0.1 inches, etc.

Value

List of data frames, one for each session but the last (columns 'x1','y1','x2','y2','d').

See Also

turnover, `sim.popn`

Examples

```
set.seed(12345)
pop3 <- sim.popn(D = 2, core = make.grid(), buffer = 200, nsessions = 3,
    details = list(lambda = 1.0, movemodel = 'BVE', move.a = 50,
    edgemethod = 'stop'))
m <- extractMoves(pop3, plotn = 10, length = 0.1)
mean(unlist(sapply(m, '[', 'd')))    # less than nominal 2 x move.a

# For distances closer to nominal for BVE (2 x move.a = 100),
# increase size of arena (e.g., buffer = 500) and consider only
# central animals (e.g., maxradius = 300).
```

FAQ *Frequently Asked Questions, And Others*

Description

A place for hints and miscellaneous advice.

How do I install and start secr?

Follow the usual procedure for installing from CRAN archive (see menu item Packages | Install package(s)... in Windows). You also need to get the package **abind** from CRAN.

Like other contributed packages, **secr** needs to be loaded before each use e.g.,`library(secr)`.

You can learn about changes in the current version with `news(package = "secr")`.

How can I get help?

There are three general ways of displaying documentation from within R. Firstly, you can bring up help pages for particular functions from the command prompt. For example:

`?secr` or `?secr.fit`

Secondly, help.search() lets you ask for a list of the help pages on a vague topic (or just use ?? at the prompt). For example:

`?? "linear models"`

Thirdly, you can display various **secr** documents listed in `secr-package`.

Tip: to search all secr help pages open the pdf version of the manual in Acrobat Reader (secr-manual.pdf; see also ?secr) and use <ctrl> F.

There is a support forum at www.phidot.org/forum under 'DENSITY|secr' and another at secrgroup. See below for more R tips. Some specific problems with `secr.fit` are covered in Troubleshooting.

How should I report a problem?

If you get really stuck or find something you think is a bug then please report the problem to one of the online lists.

You may be asked to send an actual dataset - ideally, the simplest one that exhibits the problem. Use `save` to wrap several R objects together in one .RData file, e.g., `save("captdata", "secrdemo.0", "secrdemo.b", file = "mydata.RData")`. Also, paste into the text of your message the output from `packageDescription( "secr" )`.

Why do I get different answers from secr and Density?

Strictly speaking, this should not happen if you have specified the same model and likelihood, although you may see a little variation due to the different maximization algorithms. Likelihoods (and estimates) may differ if you use different integration meshes (habitat masks), which can easily happen because the programs differ in how they set up the mesh. If you want to make a precise comparison, save the Density mesh to a file and read it into **secr**, or vice versa.

Extreme data, especially rare long-distance movements, may be handled differently by the two programs. The 'minprob' component of the 'details' argument of `secr.fit` sets a lower threshold of probability for capture histories (smaller values are all set to minprob), whereas Density has no explicit limit.

How can I speed up model fitting and model selection?

There are many ways - see Speed tips and secr-troubleshooting.pdf.

Keep the number of mask points to a minimum and avoid detection covariates with many levels.

Does secr use multiple cores?

Some computations can be run in parallel on multiple processors (most desktops these days have multiple cores). Likelihood calculations in `secr.fit` assign capture histories to multiple parallel threads whenever possible.

The number of threads (cores) is controlled by an environment variable set by `setNumThreads` or the 'ncores' argument of some functions.

Can a model use detector-level covariates that vary over time?

Yes. See ?timevaryingcov. However, a more direct way to control for varying effort is provided - see the 'usage' atribute, which now allows a continuous measure of effort (secr-varyingeffort.pdf).

A tip: covariate models for detection fit more quickly when the covariate takes only a few different values. Use `binCovariate` to bin values.

Things You Might Need To Know About R

The function `findFn` in package **sos** lets you search CRAN for R functions by matching text in their documentation.

There is now a vast amount of R advice available on the web. For the terminally frustrated, 'R inferno' by Patrick Burns is recommended (`https://www.burns-stat.com/pages/Tutor/R_inferno.pdf`). "If you are using R and you think you're in hell, this is a map for you".

Method functions for S3 classes cannot be listed in the usual way by typing the function name at the R prompt because they are 'hidden' in a namespace. Get around this with getAnywhere(). For example:

```
getAnywhere(print.secr)
```

R objects have 'attributes' that usually are kept out of sight. Important attributes are 'class' (all objects), 'dim' (matrices and arrays) and 'names' (lists). **secr** hides quite a lot of useful data as named 'attributes'. Usually you will use summary and extraction methods (`traps`, `covariates`, `usage` etc.) to view and change the attributes of the various classes of object in **secr**. If you're curious, you can reveal the lot with 'attributes'. For example, with the demonstration capture history data 'captdata':

```
traps(captdata) ## extraction method for `traps'
```

```
attributes(captdata) ## all attributes
```

Also, the function `str` provides a compact summary of any object:

```
str(captdata)
```

References

Claeskens, G. and Hjort N. L. (2008) *Model Selection and Model Averaging*. Cambridge: Cambridge University Press.

`fillNA` *Fill Missing Covariate Values*

Description

Missing covariate values are not allowed in a habitat mask. This function replaces each missing value with a given value or the closest non-missing value. This is not recommended if more than a few cells have missing values.

Usage

```
fillNA(mask, cov = NULL, method = c("nearest", "value"), value = NULL, verbose = TRUE)
```

Arguments

`mask`	habitat mask or similar
`cov`	character vector of covariate names
`method`	character
`value`	character, numeric, or factor value to replace all NA
`verbose`	logical; if TRUE the replacements are reported

Details

If there is more than one covariate then each is handled separately. If no covariate is named then all covariates are filled.

When method = "value" all NA values are replaced with the given value.

The method argument switches automatically to 'value' if value is specified.

Value

An object with the same class as the input and the requested covariate(s) replaced with non-missing values.

See Also

skink, addCovariates

Examples

```
# artificial example
msk <- make.mask(traps(captdata), spacing = 20)
covariates(msk) <- data.frame(rand = sample(factor(c(NA,1,2)), nrow(msk),
    replace = TRUE, prob = c(0.2,0.4,0.4)))

msk2 <- fillNA(msk)

par(mfrow = c(1,2))
plot(msk, cov='rand', dots = FALSE)
plot(msk2, cov='rand', dots = FALSE)

msk3 <- fillNA(msk, value = factor(2, levels=1:2))

# a call to fillNA shows whether any covariate has missing values
# assign result to tmp to avoid unwanted output
tmp <- fillNA(msk2)
```

Fletcher.chat *Estimate overdispersion*

Description

General function for estimating a variance inflation factor ($\hat{c}$) from observed counts.

Usage

```
Fletcher.chat (observed, expected, np, verbose = TRUE,
    type = c('Fletcher', 'Wedderburn', 'both'), multinomial = FALSE)
```

Arguments

`observed`	integer vector of observed counts, or a list of such vectors
`expected`	numeric vector of expected counts
`np`	integer number of parameters estimated
`verbose`	logical; if TRUE returns extended output
`type`	character
`multinomial`	logical; if TRUE, one df is subtracted for the constraint

Details

`Fletcher.chat` applies the overdispersion formula of Fletcher (2012) or computes the conventional (Wedderburn 1974) variance inflation factor X^2/df. It is used by `chat.nj`.

A conventional variance inflation factor due to Wedderburn (1974) is $\hat{c}_X = X^2/(K-p)$ where K is the number of detectors, p is the number of estimated parameters, and

$$X^2 = \sum_k (n_k - E(n_k))^2/E(n_k).$$

Fletcher's $\hat{c}$ is an improvement on $\hat{c}_X$ that is less affected by small expected counts. It is defined by

$$\hat{c} = c_X/(1+\bar{s}),$$

where $\bar{s} = \sum_k s_k/K$ and $s_k = (n_k - E(n_k))/E(n_k)$.

The inputs 'observed' and 'expected' are vectors of counts (e.g., number of distinct individuals per detector); 'observed' may also be a list of such vectors, possibly simulated.

Value

Output depends on 'verbose', 'observed' and 'type':

– if 'observed' is a list of nk vectors (usually generated by simulation) then the output is a vector of (Fletcher or Wedderburn) $\hat{c}$ values, one element for each component of 'observed', unless type = "both" when the output is a list of two such vectors. Argument 'verbose' is ignored.

– if 'observed' is a simple vector then 'verbose' output is a list comprising input values, various summary statistics, and the computed Fletcher overdispersion ('chat'). The statistic 'cX2' is the conventional variance inflation factor of Wedderburn (1974) – X^2/df. For `verbose = FALSE`, a single estimate of $\hat{c}$ is returned when `type = "Fletcher"` or `type = "Wedderburn"`, otherwise a vector of the two estimates.

References

Fletcher, D. (2012) Estimating overdispersion when fitting a generalized linear model to sparse data. *Biometrika* **99**, 230–237.

Wedderburn, R. W. M. (1974) Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika* **61**, 439–47.

See Also

`chat.nj`, `adjustVarD`

fxi *Probability Density of Activity Centre*

Description

Display contours of the probability density function for the estimated location of one or more activity centres (AC), compute values for particular points X, or compute mode of pdf. The pdf is given by $f(X_j|\omega_i) = \Pr(\omega_i|X_j)\pi(X_j)$, where $\pi(X)$ is the probability density of range centres across the mask (Borchers and Efford 2008).

These functions were previously named `fxi.secr`, `fxi.contour` and `fxi.mode`.

Usage

```
## S3 method for class 'secr'
fxi(object, i = NULL, sessnum = 1, X = NULL, ncores = NULL, ...)

fxiContour (object, i = 1, sessnum = 1, border = 100, nx = 64,
    levels = NULL, p = seq(0.1,0.9,0.1), plt = TRUE, add = FALSE,
    fitmode = FALSE, plotmode = FALSE, fill = NULL,
    output = c('list','sf','SPDF'), ncores = NULL, ...)

fxiMode(object, i = 1, sessnum = 1, start = NULL, ncores = NULL, ...)
```

Arguments

`object`	a fitted secr model
`i`	integer or character vector of individuals (defaults to all in `fxi`), or a single individual as input to fxiMode
`sessnum`	session number if `object$capthist` spans multiple sessions
`border`	width of blank margin around the outermost detectors
`nx`	dimension of interpolation grid in x-direction
`levels`	numeric vector of confidence levels for Pr(X|wi)
`p`	numeric vector of contour levels as probabilities
`plt`	logical to plot contours
`add`	logical to add contour(s) to an existing plot
`fitmode`	logical to refine estimate of mode of each pdf
`plotmode`	logical to plot mode of each pdf
`X`	2-column matrix of x- and y- coordinates (defaults to mask)
`fill`	vector of colours to fill contours (optional)
`output`	character; format of output (list, sf or SpatialPolygonsDataFrame)
`ncores`	integer number of threadss to be used for parallel processing
`start`	vector of x-y coordinates for maximization
`...`	additional arguments passed to `contour` or `nlm`

Details

`fxiContour` computes contours of AC probability density for one or more detection histories. Increase `nx` for smoother contours. If `levels` is not set, contour levels are set to approximate the confidence levels in `p`.

`fxi` computes the AC probability density for one or more detection histories; `X` may contain coordinates for one or several points; a dataframe or vector (x then y) will be coerced to a matrix.

`fxiMode` attempts to find the x- and y-coordinates corresponding to the maximum of the AC pdf for a single detection history (i.e. `i` is of length 1). `fxiMode` calls `nlm`.

`fxiContour` with `fitmode = TRUE` calls `fxiMode` for each individual. Otherwise, the reported mode is an approximation (mean of coordinates of highest contour).

If `i` is character it will be matched to row names of object$capthist (restricted to the relevant session in the case of a multi-session fit); otherwise it will be interpreted as a row number.

Values of the pdf are normalised by dividing by the integral of $\Pr(\omega_i|X)\pi(X)$ over the habitat mask in `object`. (In secr >= 4.0 may differ from previous versions).

Setting `ncores = NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

If `start` is not provided to `fit.mode` then (from 2.9.4) the weighted mean of all detector sites is used (see Warning below).

The ... argument gives additional control over a contour plot; for example, set `drawlabels = FALSE` to suppress contour labels.

Value

`fxi` –

Vector of probability densities

`fxiContour` (output = 'list') –

Coordinates of the plotted contours are returned as a list with one component per polygon. The list is returned invisibly if plt = TRUE.

An additional component 'mode' reports the x-y coordinates of the highest point of each pdf (see Details).

`fxiContour` (output = 'SPDF') –

Contours are returned as a SpatialPolygonsDataFrame (see package **sp**) with one component per animal. The attributes dataframe has two columns, the x- and y-coordinates of the mode. The SpatialPolygonsDataFrame is returned invisibly if plt = TRUE.

`fxiContour` (output = 'sf') – simple features 'sf' object, as for SPDF.

`fxiMode` –

List with components 'x' and 'y'

Warnings

`fxiMode` may fail to find the true mode unless a good starting point is provided. Note that the distribution may have multiple modes and only one is reported. The default value of `start` before **secr** 2.9.4 was the first detected location of the animal.

Note

From **secr** 2.8.3, these functions work with both homogeneous and inhomogeneous Poisson density models, and `fxi` accepts vector-valued `i`.

See `fxTotal` for a surface summed across individuals.

References

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

See Also

`pdotContour`, `contour`, `fxTotal`

Examples

```
## Not run:

fxi(secrdemo.0, i = 1, X = c(365,605))

## contour first 5 detection histories
plot(secrdemo.0$capthist)
fxiContour (secrdemo.0, i = 1:5, add = TRUE,
    plotmode = TRUE, drawlabels = FALSE)

## extract modes only
## these are more reliable than those from fit.mode called directly as
## they use a contour-based approximation for the starting point
fxiout <- fxiContour (secrdemo.0, i = 1:5, plt = FALSE, fitmode = TRUE)
t(sapply(fxiout, "[[", "mode"))

## using fill colours
## lty = 0 suppresses contour lines
## nx = 256 ensures smooth outline
plot(traps(captdata))
fxiContour(secrdemo.0, i = 1:5, add = TRUE, p = c(0.5,0.95), drawlabels
    = FALSE, nx = 256, fill = topo.colors(4), lty = 0)

## output as simple features
sf <- fxiContour(secrdemo.0, i = 1:3, plt = FALSE, p = c(0.5,0.95),
    nx = 256, output = 'sf', fitmode = TRUE)

## save as ESRI shapefile testsf.shp etc.
library(sf)
st_write(sf, 'testsf.shp')
## plot contours and modes
plot(st_as_sfc(sf))    # outline only
points(sf$modex, sf$modey)

## output as SpatialPolygonsDataFrame
spdf <- fxiContour(secrdemo.0, i = 1:3, plt = FALSE, p = c(0.5,0.95),
```

```
    nx = 256, output = 'SPDF', fitmode = TRUE)
sp::plot(spdf)
points(data.frame(spdf))


## End(Not run)
```

---

fxTotal *Activity Centres of Detected and Undetected Animals*

Description

The summed probability densities of both observed and unobserved individuals are computed for a fitted model and dataset.

Function `fx.total` was replaced by method `fxTotal` in **secr** 5.0.0.

Usage

```
## S3 method for class 'secr'
fxTotal(object, sessnum = 1, mask = NULL, ncores = NULL, ...)
```

Arguments

`object`	a fitted secr model
`sessnum`	session number if `object$capthist` spans multiple sessions
`mask`	x- and y- coordinates of points at which density will be computed
`ncores`	integer number of threads to be used for parallel processing
`...`	other arguments passed to `detectpar` and thence to `predict.secr`

Details

This function calls `fxi` for each detected animal and overlays the results to obtain a summed probability density surface D.fx for the locations of the home-range centres of detected individuals.

A separate calculation using `pdot` provides the expected spatial distribution of undetected animals, as another density surface: crudely, D.nc(X) = D(X) * (1 – pdot(X)).

The pointwise sum of the two surfaces is sometimes used to represent the spatial distrbution of the population, but see Notes.

Setting `ncores = NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

Value

An object of class 'Dsurface' (a variety of mask) with a 'covariates' attribute that is a dataframe with columns –

D.fx	sum of fxi over all detected individuals
D.nc	expected density of undetected ('not caught') individuals
D.sum	sum of D.fx and D.nc

All densities are in animals per hectare (the 'scale' argument of plot.Dsurface allows the units to be varied later).

Note

The surface D.sum represents what is known from the data about a specific realisation of the spatial point process for home range centres: varying the intensity of sampling will change its shape. It is not an unbiased estimate of a biologically meaningful density surface. The surface will always tend to lack relief towards the edge of a habitat mask where the main or only contribution is from D.nc.

References

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

See Also

fxi, fxiContour, pdot

Examples

```
## Not run:

tmp <- fxTotal(secrdemo.0)

## to plot we must name one of the covariates:
## the Dsurface default 'D.0' causes an error

plot(tmp, covariate = 'D.sum', col = terrain.colors(16),
   plottype = 'shaded')
plot(tmp, covariate = 'D.sum', col = 'white', add = TRUE,
   plottype = 'contour')
if (interactive()) {
    spotHeight(tmp, prefix = 'D.sum')
}

fxsurface <- fxTotal(ovenbird.model.D, sessnum = 3)
plot(fxsurface, covariate = 'D.sum')


## End(Not run)
```

gridCells *Construct Grid Cells*

Description

Forms grid cells centred on input points.

Usage

```
gridCells(x, cellsize = spacing(x), crs = NA)
```

Arguments

x	matrix or dataframe with x- and y-coordinates
cellsize	length of gridcell side
crs	crs description suitable for st_crs

Details

The argument x will often be a traps or mask object with spacing attribute. Otherwise cellsize must be provided.

See make.grid for jittered locations within grid cells.

Value

A simple features (sf) object of class 'sfc_MULTIPOLYGON'.

crs may be the integer EPSG code (e.g. 3578 Yukon Albers).

See Also

plotMaskEdge, spacing

Examples

```
plot(gridCells(traps(captdata)))
plot(traps(captdata), add = TRUE)
```

hcov *Hybrid Mixture Model*

Description

The argument hcov in secr.fit is used to fit a hybrid mixture model. 'Hybrid' refers to a flexible combination of latent classes (as in a finite mixture) and known classes (cf groups or sessions). A hybrid mixture model includes a parameter 'pmix' for the mixing proportion and optionally allows detection parameters to be modelled as class-specific (~ h2). This is particularly useful for modelling sex ratio and sex differences in detection, and matches the Bayesian sex-specific model of Gardner et al. (2010).

For observed animals all of unknown class the model is identical to a finite mixture (i.e. latent-class) model. For observed animals all of known class, the classes are no longer 'latent' and the model is equivalent to a grouped model with an additional binomial factor for class membership.

Assumptions

hcov identifies a single individual covariate (the class covariate) that should be a factor with two levels, or contain character values that will be coerced to a factor (e.g., 'f', 'm'). Missing values (NA) are used for individuals of unknown class. If hcov has more than two levels, all but the first two levels are converted to NA (but see exception for h3 models below).

It is assumed that the probability of recording a missing value for the class covariate is independent of the true class membership (e.g., sex equally likely to be recorded for males and females).

Operational details

A hybrid mixture model is fitted whenever hcov is not NULL. Mixture models include a parameter 'pmix', the mixing proportion. If the covariate identified by hcov is missing (‘' or NA) for all individuals *and* a mixture term (h2 or h3) appears in the detection model (e.g., g0 ~ h2) then a conventional finite mixture model is fitted (cf Pledger 2000, Borchers & Efford 2008).

As with finite mixture models, any detection parameter (g0, sigma etc.) may be modelled as depending on mixture class by model specifications such as (g0 ~ h2, sigma ~ h2). See Examples.

In general hcov has been designed for two classes and two classes are assumed if neither 'h2' nor 'h3' appears in the model formulae. However, there is a small exception: hcov may have three non-missing levels if 'h3' appears in a model formula. Note that h2 cannot be combined with h3; h3 is for advanced use only and has not been fully tested.

The number of fitted parameters is the same as the corresponding finite mixture model if mixture terms ('h2', 'h3') appear in the model formulae. Otherwise (no mixture terms) estimating pmix requires a single extra parameter. The estimate of pmix then depends solely on the observed class proportions in the covariate, and the beta variance-covariance matrix will show zero covariance of pmix with other detection parameters.

Models for pmix

Variation in the parameter pmix may be modelled across sessions i.e., models such as pmix ~ session or pmix ~ Session are valid, as are formulae involving session covariates defined in the sessioncov argument of secr.fit.

If no mixture term appears in the formula for pmix then one is added automatically (usually 'h2'). This serves mostly to keep track of values in the output.

Attempting to model pmix as a function of individual covariates or other within-session terms (t, b etc.) will cause an error.

Interpreting output

When you display a fitted secr model the parameter estimates are in a final section headed 'Fitted (real) parameters evaluated at base levels of covariates'. The same output may be obtained by calling the `predict` method directly. Calling `predict` has the advantage that you can obtain estimates for levels of the covariates other than the base levels, by specifying `newdata`. An example below shows how to specify h2 in `newdata`. [Note: `predict` is generic, and you must consult ?predict.secr to see the help for the specific implementation of this method for fitted secr objects].

The output from `predict.secr` for a mixture model is a list with one component for each (possibly latent) class. Each row corresponds to a fitted real parameter: ordinarily these include the detection parameters (e.g., g0, sigma) and the mixing proportion (pmix).

In the case of a model fitted by maximizing the full likelihood (`CL = FALSE`), density D will also appear in the output. Note that only one parameter for density is estimated, the total density across classes. This total density figure appears twice in the output, once for each class.

The standard error (SE.estimate) is shown for each parameter. These are asymptotic estimates back-transformed from the link scale. The confidence limits are also back-transformed from the link scale (95% CI by default; vary `alpha` in `predict.secr` if you want e.g. 90% CI).

The mixing proportion pmix depends on the composition of the sample with respect to `hcov` and the detection model. For a null detection model the mixing proportion is exactly the proportion in the sample, with appropriate binomial confidence limits. Otherwise, the mixing proportion adjusts for class differences in the probability and scale of detection (see Examples).

The preceding refers to the default behaviour when pmix ~ h2. It is possible also to fix the mixing proportion at any arbitrary value (e.g., fixed = list(pmix = 0.5) for 1:1 sex ratio).

On output the classes are tagged with the factor levels of `hcov`, regardless of how few or how many individuals were actually of known class. If only a small fraction were of known class, and there is cryptic variation unrelated to `hcov`, then the association between the fitted classes and the nominal classes (i.e. levels of `hcov`) may be weak, and should not be trusted.

Limitations

Hybrid mixture models are incompatible with groups as presently implemented.

The hcov likelihood conditions on the number of known-class individuals. A model fitted with `hcov = NULL` or with a different hcov covariate has in effect a different data set, and likelihoods, deviances or AICs cannot be compared. AIC can be used to compare models provided they all have the same hcov covariate in the call to `secr.fit`, whether or not h2 appears in the model definition.

Likelihood

The likelihood of the hybrid mixture model is detailed in an appendix of the vignette secr-finitemixtures.pdf.

References

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

Gardner, B., Royle, J.A., Wegan, M.T., Rainbolt, R. and Curtis, P. (2010) Estimating black bear density using DNA data from hair snares. *Journal of Wildlife Management* **74**, 318–325.

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

See Also

secr.fit

Examples

```
## Not run:

## house mouse dataset, morning trap clearances
## 81 female, 78 male, 1 unknown
morning <- subset(housemouse, occ = c(1,3,5,7,9))
summary(covariates(morning))

## speedy model fitting with coarse mask
mmask <- make.mask(traps(morning), buffer = 20, nx = 32)

## assuming equal detection of males and females
## fitted sex ratio p(female) = 0.509434 = 81 / (81 + 78)
fit.0 <- secr.fit(morning, hcov = "sex", mask = mmask, trace = FALSE)
predict(fit.0)

## allowing sex-specific detection parameters
## this leads to new estimate of sex ratio
fit.h2 <- secr.fit(morning, hcov = "sex", mask = mmask, trace = FALSE,
    model = list(g0 ~ h2, sigma ~ h2))
predict(fit.h2)

## specifying newdata for h2 - equivalent to predict(fit.h2)
predict(fit.h2, newdata = data.frame(h2 = factor(c('f','m'))))

## conditional likelihood fit of preceding model
## estimate of sex ratio does not change
fit.CL.h2 <- secr.fit(morning, hcov = "sex", mask = mmask, trace = FALSE,
    CL = TRUE, model = list(g0 ~ h2, sigma ~ h2))
predict(fit.CL.h2)

## did sexes differ in detection parameters?
fit.CL.0 <- secr.fit(morning, hcov = "sex", mask = mmask, trace = FALSE,
    CL = TRUE, model = list(g0 ~ 1, sigma ~ 1))
```

```
LR.test(fit.CL.h2, fit.CL.0)

## did sex ratio deviate from 1:1?
fit.CL.h2.50 <- secr.fit(morning, hcov = "sex", mask = mmask, trace = FALSE,
    CL = TRUE, model = list(g0 ~ h2, sigma ~ h2), fixed = list(pmix = 0.5))
LR.test(fit.CL.h2, fit.CL.h2.50)

## did sexes show extra-compensatory variation in lambda0?
## (Efford and Mowat 2014)
fit.CL.a0 <- secr.fit(morning, hcov = "sex", mask = mmask, trace = FALSE,
    CL = TRUE, model = list(a0 ~ 1, sigma ~ h2))
LR.test(fit.CL.h2, fit.CL.a0)

## trend in ovenbird sex ratio, assuming sex-specific detection
omask <- make.mask(traps(ovenCH), buffer = 300, nx = 32)
fit.sextrend <- secr.fit(ovenCH, model = list(g0~h2, sigma~h2, pmix~Session),
    hcov = "Sex", CL = TRUE, mask = omask, trace = FALSE)
predict(fit.sextrend)[1:5]


## End(Not run)
```

---

head *First or Last Part of an Object*

Description

Returns the first or last parts of secr objects

Usage

```
## S3 method for class 'mask'
head(x, n=6L, ...)
## S3 method for class 'Dsurface'
head(x, n=6L, ...)
## S3 method for class 'traps'
head(x, n=6L, ...)
## S3 method for class 'capthist'
head(x, n=6L, ...)
## S3 method for class 'mask'
tail(x, n=6L, ...)
## S3 method for class 'Dsurface'
tail(x, n=6L, ...)
## S3 method for class 'traps'
tail(x, n=6L, ...)
## S3 method for class 'capthist'
tail(x, n=6L, ...)
```

Arguments

x	'mask', 'traps' or 'capthist' object
n	a single integer. If positive, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the n last/first number of elements of x.
...	other arguments passed to `subset`

Details

These custom S3 methods retain the class of the target object, unlike the default methods applied to 'mask', 'Dsurface', 'traps' or 'capthist' objects.

Value

An object of the same class as x, but (usually) fewer rows.

See Also

`head`, `tail`

Examples

```
head(possummask)
```

---

homerange *Home Range Statistics*

Description

Some ad hoc measures of home range size may be calculated in **secr** from capture–recapture data:

`dbar` is the mean distance between consecutive capture locations, pooled over individuals (e.g. Efford 2004). `moves` returns the raw distances.

`MMDM` (for 'Mean Maximum Distance Moved') is the average maximum distance between detections of each individual i.e. the observed range length averaged over individuals (Otis et al. 1978).

`ARL` (or 'Asymptotic Range Length') is obtained by fitting an exponential curve to the scatter of observed individual range length vs the number of detections of each individual (Jett and Nichols 1987: 889).

`RPSV` (for 'Root Pooled Spatial Variance') is a measure of the 2-D dispersion of the locations at which individual animals are detected, pooled over individuals (cf Calhoun and Casby 1958, Slade and Swihart 1983).

`moves` reports the distance between successive detections of each animal.

`centroids` reports the averaged coordinates of each animal's detections

`ORL` reports the observed range length of each animal, the maximum distance between any two detections.

t2r2 computes Schoener's measure of serial correlation (Swihart and Slade 1985, 1987). The expected value of t2r2 for independent detections is about 2.0.

trapsPerAnimal tabulates the number of animals recorded at 1, 2, ..., K detectors

Usage

```
dbar(capthist, userdist = NULL, mask = NULL, telemetry = NULL)
MMDM(capthist, min.recapt = 1, full = FALSE, userdist = NULL, mask = NULL,
    telemetry = NULL)
ARL(capthist, min.recapt = 1, plt = FALSE, full = FALSE, userdist = NULL,
    mask = NULL, telemetry = NULL)
moves(capthist, userdist = NULL, mask = NULL, names = FALSE, telemetry = NULL)
RPSV(capthist, CC = FALSE, telemetry = NULL)
ORL(capthist, userdist = NULL, mask = NULL, telemetry = NULL)
t2r2(capthist, telemetry = NULL)
centroids(capthist, telemetry = NULL)
trapsPerAnimal(capthist)
```

Arguments

capthist	object of class capthist
userdist	function or matrix with user-defined distances
mask	habitat mask passed to userdist function, if required
names	logical; should results be ordered alphanumerically by row names?
min.recapt	integer minimum number of recaptures for a detection history to be used
plt	logical; if TRUE observed range length is plotted against number of recaptures
full	logical; set to TRUE for detailed output
CC	logical for whether to use Calhoun and Casby formula
telemetry	logical; if TRUE then calculation uses the telemetryxy attribute

Details

dbar is defined as –

$$\overline{d} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n_i-1} \sqrt{(x_{i,j} - x_{i,j+1})^2 + (y_{i,j} - y_{i,j+1})^2}}{\sum_{i=1}^{n} (n_i - 1)}$$

When CC = FALSE, RPSV is defined as –

$$RPSV = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{n_i} [(x_{i,j} - \overline{x}_i)^2 + (y_{i,j} - \overline{y}_i)^2]}{\sum_{i=1}^{n} (n_i - 1) - 1}}$$

.

Otherwise (CC = TRUE), RPSV uses the formula of Calhoun and Casby (1958) with a different denominator –

$$s = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n_i}[(x_{i,j}-\overline{x}_i)^2+(y_{i,j}-\overline{y}_i)^2]}{2\sum_{i=1}^{n}(n_i-1)}}$$

.

The Calhoun and Casby formula (offered from 2.9.1) correctly estimates σ when trapping is on an infinite, fine grid, and is preferred for this reason. The original RPSV (CC = FALSE) is retained as the default for compatibility with previous versions of **secr**.

RPSV has a specific role as a proxy for detection scale in inverse-prediction estimation of density (Efford 2004, 2023).

RPSV is used in autoini to obtain plausible starting values for maximum likelihood estimation.

MMDM and ARL discard data from detection histories containing fewer than min.recapt+1 detections.

Schoener's measure (t2r2) is defined as t^2/r^2 where

$$t^2 = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n_i-1}[(x_{i,j}-x_{i,j+1})^2+(y_{i,j}-y_{i,j+1})^2]}{\sum_{i=1}^{n}(n_i-1)},\quad r^2 = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n_i}[(x_{i,j}-\overline{x}_i)^2+(y_{i,j}-\overline{y}_i)^2]}{\sum_{i=1}^{n}(n_i-1)},$$

where $(\bar{x}_i, \bar{y}_i)$ is the arithmetic mean location of individual i.

The userdist option is included for exotic non-Euclidean cases (see e.g. secr.fit details). RPSV is not defined for non-Euclidean distances.

If capthist comprises standalone telemetry data (all detector 'telemetry') or the telemetry argument is set (from secr 5.5.0) then calculations are performed on the telemetry coordinates. If capthist combines telemetry data and conventional detections ('multi', 'proximity' etc.) then only the conventional data are summarised.

Movements are reliably reported by moves only if there is a maximum of one detection per animal per occasion. The sequence of detections within any occasion is not known; where these occur the sequence used by moves is arbitrary (sequence follows detector index).

Value

For dbar, MMDM, ARL and RPSV –

Scalar distance in metres, or a list of such values if capthist is a multi-session list.

The full argument may be used with MMDM and ARL to return more extensive output, particularly the observed range length for each detection history.

For moves –

List with one component for each animal, a vector of distances, or numeric(0) if the animal is detected only once. A list of such lists if capthist is a multi-session list.

For centroids –

For a single-session capthist, a matrix of two columns, the x- and y-coordinates of the centroid of the detections of each animal. The number of detections is returned as the attribute 'Ndetections', a 1-column matrix.

For a multi-session capthist, a 3-D array as before, but with a third dimension for the session. Centroid coordinates are missing (NA) if the animal was not detected in a session. The attribute 'Ndetections' with the number of detections per animal and session is a matrix.

For `trapsPerAnimal` –

A vector with the number of animals detected at k detectors.

Note

All measures are affected by the arrangement of detectors. `dbar` is also affected quite strongly by serial correlation in the sampled locations. Using `dbar` with 'proximity' detectors raises a problem of interpretation, as the original sequence of multiple detections within an occasion is unknown. RPSV is a value analogous to the standard deviation of locations about the home range centre.

The value returned by `dbar` for 'proximity' or 'count' detectors is of little use because multiple detections of an individual within an occasion are in arbitrary order.

Inclusion of these measures in the **secr** package does not mean they are recommended for general use! It is usually better to use a spatial parameter from a fitted model (e.g., σ of the half-normal detection function). Even then, be careful that σ is not 'contaminated' with behavioural effects (e.g. attraction of animal to detector) or 'detection at a distance'.

The argument 'names' was added in 3.0.1. The default `names = FALSE` causes a change in behaviour from that version onwards.

References

Calhoun, J. B. and Casby, J. U. (1958) Calculation of home range and density of small mammals. Public Health Monograph. No. 55. U.S. Government Printing Office.

Efford, M. G. (2004) Density estimation in live-trapping studies. *Oikos* **106**, 598–610.

Efford, M. G. (2023) ipsecr: An R package for awkward spatial capture–recapture data. *Methods in Ecology and Evolution* In press.

Jett, D. A. and Nichols, J. D. (1987) A field comparison of nested grid and trapping web density estimators. *Journal of Mammalogy* **68**, 888–892.

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**, 1–135.

Slade, N. A. and Swihart, R. K. (1983) Home range indices for the hispid cotton rat (*Sigmodon hispidus*) in Northeastern Kansas. *Journal of Mammalogy* **64**, 580–590.

Swihart, R. K. and Slade, N. A. (1985) Testing for independence of observations in animal movements. *Ecology* **66**, 1176–1184.

Swihart, R. K. and Slade, N. A. (1987) A test for independence of movements as shown by live trapping. *American Midland Naturalist* **117**, 204–207.

See Also

`autoini`

Examples

```
dbar(captdata)
RPSV(captdata)
RPSV(captdata, CC = TRUE)

centr <- centroids(captdata)
plot(traps(captdata), border = 20 )
text(centr[,1], centr[,2], attr(centr, 'Ndetections'))
text(centr[,1]+2, centr[,2]+3, rownames(captdata), cex = 0.6,
    adj = 0)
```

---

hornedlizard *Flat-tailed Horned Lizard Dataset*

Description

Data from multiple searches for flat-tailed horned lizards (*Phrynosoma mcalli*) on a plot in Arizona, USA.

Usage

```
hornedlizardCH
```

Details

The flat-tailed horned lizard (*Phrynosoma mcalli*) is a desert lizard found in parts of southwestern Arizona, southeastern California and northern Mexico. There is considerable concern about its conservation status. The species is cryptically coloured and has the habit of burying under the sand when approached, making it difficult or impossible to obtain a complete count (Grant and Doherty 2007).

K. V. Young conducted a capture–recapture survey of flat-tailed horned lizards 25 km south of Yuma, Arizona, in the Sonoran Desert. The habitat was loose sand dominated by creosote bush and occasional bur-sage and Galletta grass. A 9-ha plot was surveyed 14 times over 17 days (14 June to 1 July 2005). On each occasion the entire 300 m x 300 m plot was searched for lizards. Locations within the plot were recorded by handheld GPS. Lizards were captured by hand and marked individually on their underside with a permanent marker. Marks are lost when the lizard sheds, but this happens infrequently and probably caused few or no identification errors during the 2.5-week study.

A total of 68 individuals were captured 134 times. Exactly half of the individuals were recaptured at least once.

Royle and Young (2008) analysed the present dataset to demonstrate a method for density estimation using data augmentation and MCMC simulation. They noted that the plot size was much larger than has been suggested as being practical in operational monitoring efforts for this species, that the plot was chosen specifically because a high density of individuals was present, and that high densities typically correspond to less movement in this species. The state space in their analysis was a square comprising the searched area and a 100-m buffer (J. A. Royle pers. comm.).

The detector type for these data is 'polygonX' and there is a single detector (the square plot). The data comprise a capture history matrix (the body of `hornedlizardCH`) and the x-y coordinates of each positive detection (stored as an attribute that may be displayed with the 'xy' function); the 'traps' attribute of `hornedlizardCH` contains the vertices of the plot. See secr-datainput.pdf for guidance on data input.

Non-zero entries in a polygonX capture-history matrix indicate the number of the polygon containing the detection. In this case there was just one polygon, so entries are 0 or 1. No animal can appear more than once per occasion with the polygonX detector type, so there is no need to specify 'binomN = 1' in secr.fit.

Object	Description
hornedlizardCH	single-session capthist object
FTHL.0	fitted model (see Examples)
FTHL.CL	fitted model (see Examples)

Source

Royle and Young (2008) and J. A. Royle (pers. comm.), with additional information from K. V. Young (pers. comm.).

References

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

Grant, T. J. and Doherty, P. F. (2007) Monitoring of the flat-tailed horned lizard with methods incorporating detection probability. *Journal of Wildlife Management* **71**, 1050–1056

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.

See Also

`capthist`, `detector`, `reduce.capthist`

Examples

```
plot(hornedlizardCH, tracks = TRUE, varycol = FALSE,
    lab1 = TRUE, laboff = 6, border = 10, title =
    "Flat-tailed Horned Lizards (Royle & Young 2008)")

table(table(animalID(hornedlizardCH)))
traps(hornedlizardCH)

## show first few x-y coordinates
head(xy(hornedlizardCH))

## Not run:
```

```
## Compare default (Poisson) and binomial models for number
## caught
FTHL.0 <- secr.fit(hornedlizardCH, buffer = 80)
FTHL.bn <- secr.fit(hornedlizardCH, buffer = 80, details =
    list(distribution = "binomial"))
collate(FTHL.0, FTHL.bn)[,,,"D"]

## Conditional likelihood
FTHL.CL <- secrpoly.fit(hornedlizardCH, buffer = 80, CL = TRUE)

## Collapse occasions (does not run faster)
hornedlizardCH.14 <- reduce(hornedlizardCH, newoccasions =
    list(1:14), outputdetector = "polygon")
FTHL14.fit <- secr.fit(hornedlizardCH.14, binomN = 14)


## End(Not run)
```

---

housemouse *House mouse live trapping data*

Description

Data of H. N. Coulombe from live trapping of feral house mice (*Mus musculus*) in a salt marsh, California, USA.

Usage

```
housemouse
```

Details

H. N. Coulombe conducted a live-trapping study on an outbreak of feral house mice in a salt marsh in mid-December 1962 at Ballana Creek, Los Angeles County, California. A square 10 x 10 grid was used with 100 Sherman traps spaced 3 m apart. Trapping was done twice daily, morning and evening, for 5 days.

The dataset was described by Otis et al. (1978) and distributed with their CAPTURE software. Otis et al. (1978 p. 62, 68) cited Coulombe's unpublished 1965 master's thesis from the University of California, Los Angeles, California.

The data are provided as a single-session `capthist` object. There are two individual covariates: sex (factor levels 'f', 'm') and age class (factor levels 'j', 'sa', 'a'). The sex of two animals is not available (NA); it is necessary to drop these records for analyses using 'sex' unless missing values are specifically allowed, as in `hcov`.

The datasets were originally in the CAPTURE 'xy complete' format which for each detection gives the 'column' and 'row' numbers of the trap (e.g. ' 9 5' for a capture in the trap at position (x=9,

y=5) on the grid). Trap identifiers have been recoded as strings with no spaces by inserting zeros (e.g. '0905' in this example).

Sherman traps are designed to capture one animal at a time, but the data include 30 double captures and one occasion when there were 4 individuals in a trap at one time. The true detector type therefore falls between 'single' and 'multi'. Detector type is set to 'multi' in the distributed data objects.

Otis et al. (1978) report various analyses including a closure test on the full data, and model selection and density estimation on data from the mornings only.

Source

File 'examples' distributed with program CAPTURE.

References

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**, 1–135.

Examples

```
plot(housemouse, title = paste("Coulombe (1965), Mus musculus,",
    "California salt marsh"), border = 5, rad = 0.5,
    gridlines = FALSE)

morning <- subset(housemouse, occ = c(1,3,5,7,9))
summary(morning)

## drop 2 unknown-sex mice
known.sex <- subset(housemouse, !is.na(covariates(housemouse)$sex))

## reveal multiple captures
table(trap(housemouse), occasion(housemouse))

## Not run:

## assess need to distinguish morning and afternoon samples
housemouse.0 <- secr.fit (housemouse, buffer = 20)
housemouse.ampm <- secr.fit (housemouse, model = g0~tcov, buffer = 20,
    timecov = c(0,1,0,1,0,1,0,1,0,1))
AIC(housemouse.0, housemouse.ampm)


## End(Not run)
```

Internal *Internal Functions*

Description

Functions called internally by **secr** and exported but not usually called by users.

Usage

```
boundarytoSF (poly, ...)

Dfn2(designD, beta = NULL, ...)
```

Arguments

poly	data to define one or more polygons
designD	dataframe of density design data (output from D.designdata)
beta	numeric vector of beta values (see Details for NULL)
...	other arguments passed to sf::st_sfc or sf::st_as_sfc (e.g. st_crs)

Details

The function boundarytoSF converts various possible polygon input formats to a standard form (sfc).

Possible inputs are:

Input	From	Note
2-column matrix or dataframe	base R	
SpatialPolygons	**sp**	
SpatialPolygonsDataFrame	**sp**	
SpatVector	**terra**	
sf	**sf**	geometry type POLYGON or MULTIPOLYGON
sfc	**sf**	geometry type POLYGON or MULTIPOLYGON

Matrix input defines a single polygon.

Dfn2 is supplied automatically as 'details' argument Dfn in secr.fit when the switch Dlambda is set to TRUE for the multi-session trend reparameterization of density. Dfn2 uses beta = NULL to return the required number of density coefficients (beta parameters) in the model.

Value

boundarytoSF – Spatial object of **sf** class sfc, containing a geometry set of type POLYGON or MULTIPOLYGON. NULL input results in NULL output.

Dfn2 – Vector of density values on the link scale, suitable for the internal array (mask x groups x sessions).

References

Hijmans, R. J. (2022) terra: Spatial Data Analysis. R package version 1.5-14. https://rspatial.org/terra/

Pebesma, E. (2018) Simple features for R: standardized support for spatial vector data. *The R Journal* **10(1)**, 439–446. doi:10.32614/RJ2018009

Pebesma, E.J. and Bivand, R. S. (2005) Classes and methods for spatial data in R. *R News* **5(2)**, 9–13. https://cran.r-project.org/doc/Rnews/Rnews_2005-2.pdf.

See Also

`pointsInPolygon`, secr-spatialdata.pdf,

`predictDlambda`, secr-trend.pdf,

Examples

```
poly <- cbind(x = c(0,6,6,0,0), y = c(0,0,6,6,0))
boundarytoSF(poly)
```

`intervals` *Work with Open Population data*

Description

Functions for data manipulation

Usage

```
intervals(object, ...)
intervals(object) <- value
sessionlabels(object, ...)
sessionlabels(object) <- value
```

Arguments

`object`	capthist object
`value`	vector of intervals or primary session labels
`...`	other arguments (not used)

Details

`intervals` extracts the 'interval' attribute if it exists.

The attribute 'intervals' is set automatically by the secr function `join`.

`sessionlabels` provides session names for the primary sessions encoded in a "single-session" capthist object (e.g., the result of `join`) that has an intervals attribute. The names are used by some summary functions in the package **openCR** (M. Efford unpubl.) (`m.array`, `JS.counts`).

The function `session` has a different purpose: labelling sessions in a multi-session capthist object. However, `session` names of multi-session input are used automatically by `join` to construct the `sessionlabels` attribute of the resulting single-session object.

Value

For `intervals`, a numeric vector of time intervals, one less than the number of occasions (secondary sessions).

For `sessionlabels`, a character vector of primary session names.

Note

There is a naming conflict with the intervals function in **nlme**.

Examples

```
singlesessionCH <- join(ovenCH)
intervals(singlesessionCH)
sessionlabels(singlesessionCH)
```

---

join — *Combine or Split Sessions of capthist Object*

Description

Make a single-session capthist object from a list of single-session objects, or a multi-session capthist object.

Usage

```
join(object, remove.dupl.sites = TRUE, tol = 0.001, sites.by.name = FALSE,
    drop.sites = FALSE, intervals = NULL, sessionlabels = NULL,
    timevaryingcov = NULL)

unjoin(object, intervals, ...)
```

Arguments

`object`	list of single-session objects, or a multi-session capthist object [`join`], or a single-session capthist object [`unjoin`]
`remove.dupl.sites`	logical; if TRUE then a single record is retained for each trap site used in multiple input sessions
`tol`	absolute distance in metres within which sites are considered identical
`sites.by.name`	logical; if TRUE and `remove.dupl.sites` then duplicate sites are inferred from row names rather than x-y coordinates
`drop.sites`	logical; if TRUE then site information is discarded
`intervals`	vector of times between sessions (join) or occasions (unjoin; zero indicates same session)
`sessionlabels`	vector of session names
`timevaryingcov`	character vector of covariate names
`...`	other arguments passed to `subset.capthist`

Details

`join` The input sessions are assumed to be of the same detector type and to have the same attributes (e.g., covariates should be present for all or none).

The number of occasions (columns) in the output is equal to the sum of the number of occasions in each input.

Duplicates may be defined either as sites within a given distance (`tol`) or sites with the same name (`sites.by.name = TRUE`). Using site names is faster.

For non-spatial analyses it is efficient to drop the third dimension and discard the traps attribute (`drop.sites = TRUE`).

A new dataframe of individual covariates is formed using the covariates for the first occurrence of each animal.

If `timevaryingcov` is given then for each name a new covariate is generated for each session and populated with values observed in that session, or NA if the animal was not detected. A 'timevaryingcov' (list) attribute is created that associates each set of new session-specific columns with the corresponding old name, so that it may be used in formulae (see `timevaryingcov`).

Attributes xy and signal are handled appropriately, as is trap usage.

`unjoin` The input grouping of occasions (columns) into sessions is specified via `intervals`. This is a vector of length one less than the number of occasions (columns) in `object`. Elements greater than zero indicate a new session.

The `intervals` argument may be omitted if `object` has a valid 'intervals' attribute, as in the output from `join`.

Value

For `join`, a single-session capthist object. The vector attribute 'intervals' records the distinction between occasions that are adjacent in the input (interval = 0) and those that are in consecutive sessions (e.g., interval = 1); 'intervals' has length one less than the number of occasions.

For `unjoin`, a multi-session capthist object. Sessions are named with integers.

Note

Do not confuse unjoin with split.capthist which splits by row (animal) rather than by column (occasion).

Occasions survive intact; to pool occasions use reduce.capthist.

join was modified in version 2.9.5 to check whether the components of 'object' all used the same detectors ('traps') (putting aside differences in usage). If the traps are identical and remove.dupl.sites = TRUE then the resulting 'capthist' uses the common list of detectors, with a usage attribute formed by concatenating the usage columns of the input. This is faster than the previous filtering algorithm using 'tol'; the older algorithm is still used if the traps differ.

Problems may be encountered with large datasets. These may be alleviated by setting sites.by.name = TRUE (if matching sites have matching names, avoiding the need for coordinate matching) or drop.sites = TRUE (if only non-spatial data are required for openCR).

See Also

MS.capthist, rbind.capthist

Examples

```
joined.ovenCH <- join (ovenCH)
summary(joined.ovenCH)
attr(joined.ovenCH, "intervals")

summary(unjoin(joined.ovenCH))

## Not run:

## suppose the 5-year ovenbird covariates include a column for weight
## (here generated as random numbers)
for (i in 1:5) covariates(ovenCH[[i]])$wt <- runif(nrow(ovenCH[[i]]))
## construct single-session version of data for openCR
## identify 'wt' as varying across years
ovenCHj <- join(ovenCH, timevaryingcov = 'wt')
head(covariates(ovenCHj))
timevaryingcov(ovenCHj)
## Use example: openCR.fit(ovenCHj, model = p~wt)


## End(Not run)
```

---

kfn *Overlap Index*

Description

Computes the overlap index of Efford et al. (2016) from various inputs, including fitted models.

Usage

```
kfn(object)
```

Arguments

`object`	fitted secr model, numeric vector, matrix, dataframe

Details

`kfn` simply computes $k = \sigma\sqrt{D}/100$, where σ is the sigma parameter of a fitted halfnormal detection function and D is the corresponding density estimate. The factor of 1/100 adjusts for the units used in **secr** (sigma in metres; D in animals per hectare).

Input may be in any of these forms

1. vector with D and sigma in the first and second positions.
2. matrix with each row as in (1)
3. dataframe such as produced by `predict.secr` with rows 'D' and 'sigma', and column 'estimate'.
4. fitted **secr** model
5. a list of any of the above

Value

Numeric vector with elements 'D', 'sigma' and 'k', or a matrix with these columns.

Note

The index should not be taken too literally as a measure of overlap: it represents the overlap expected *if* activity centres are randomly distributed and *if* home ranges have bivariate normal utilisation. Thus it does *not* measure overlap due to social behaviour etc. except as that affects home range size.

The index may be estimated directly using the sigmak parameterization (i.e., when sigmak appears in the model for `secr.fit`). This provides SE and confidence limits for sigmak (= k). However, the directly estimated value of sigmak lacks the unit correction and is therefore 100 $\times$ the value from `kfn`.

References

Efford, M. G., Dawson, D. K., Jhala, Y. V. and Qureshi, Q. (2016) Density-dependent home-range size revealed by spatially explicit capture–recapture. *Ecography* **39**, 676–688.

See Also

`predict.secr`, `secr.fit`, `details`

Examples

```
kfn(secrdemo.0)

## compare
## fitk <- secr.fit(captdata, model = sigmak~1, buffer = 100, trace = FALSE)
## predict(fitk)
```

`list.secr.fit` *Fit Multiple SECR Models*

Description

This function is a wrapper for `secr.fit` that allows multiple models to be fitted.

Usage

```
list.secr.fit (..., constant = list(), prefix = "fit", names = NULL)
```

Arguments

`...`	varying arguments of `secr.fit`
`constant`	list of named arguments held constant
`prefix`	character prefix for automatic names
`names`	character names of output

Details

The ... argument may be one or several vectors of the same length that refer to a different named argument of `secr.fit`. `secr.fit` is called with the constant arguments plus the first value in each vector, then the second value, etc. The logic follows `mapply`.

Each of the ... arguments may also be a named argument with a single value, although the compound values should be wrapped in list(), passed by name (in quotes), or placed in the 'constant' list to avoid misinterpretation. For example, the capthist argument of `secr.fit` should be be wrapped in list() or " " if it is placed outside 'constant'.

'prefix' is used only if 'names' is not supplied.

Value

An `secrlist` of the successful model fits (see `secr.fit`).

In the special case that 'constant' specifies 'details' with component LLonly = TRUE, the list of values from `secr.fit` without checking or modification.

Note

This function replaces the previous function `par.secr.fit`: since the introduction of multi-threading in **secr** 4.0 it is no longer efficient to use parallel worker processes.

See Also

`secr.fit`, `AIC.secr`, `predict.secr`

Examples

```
## Not run:

# fit two detection models
fits <- list.secr.fit (model = c(g0~1, g0~b), constant = list(captdata, trace = FALSE))
AIC(fits)

# alternatively,
fits <- list.secr.fit ('captdata', model = c(g0~1, g0~b), trace = FALSE)
AIC(fits)

# replacing par.derived and par.region.N:

lapply(fits, derived)
lapply(fits, region.N)


## End(Not run)
```

`LLsurface` *Plot Likelihood Surface*

Description

LLsurface is a generic function to calculate log likelihood over a grid of values of two coefficients (beta parameters) from a fitted model and optionally make an approximate contour plot of the log likelihood surface.

A method is provided for secr objects.

Usage

```
LLsurface(object, ...)

## S3 method for class 'secr'
LLsurface(object, betapar = c("g0", "sigma"), xval = NULL,
    yval = NULL, centre = NULL, realscale = TRUE, plot = TRUE,
    plotfitted = TRUE, ncores = NULL, ...)
```

Arguments

`object`	fitted model, `secr` object output from `secr.fit`
`betapar`	character vector giving the names of two beta parameters
`xval`	vector of numeric values for x-dimension of grid
`yval`	vector of numeric values for y-dimension of grid
`centre`	vector of central values for all beta parameters
`realscale`	logical. If TRUE input and output of x and y is on the untransformed (inverse-link) scale.
`plot`	logical. If TRUE a contour plot is produced
`plotfitted`	logical. If TRUE the MLE from `object` is shown on the plot (+)
`ncores`	integer number of threads for parallel processing
`...`	other arguments passed to `contour`

Details

`centre` is set by default to the fitted values of the beta parameters in `object`. This has the effect of holding parameters other than those in `betapar` at their fitted values.

If `xval` or `yval` is not provided then 11 values are set at equal spacing between 0.8 and 1.2 times the values in `centre` (on the 'real' scale if `realscale` = TRUE and on the 'beta' scale otherwise).

Contour plots may be customized by passing graphical parameters through the ... argument.

Setting `ncores` = `NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

Value

A matrix of the log likelihood evaluated at each grid point (rows x, columns y), invisibly if `plot` = `TRUE`. Failed evaluations return NA.

Note

`LLsurface` works for named 'beta' parameters rather than 'real' parameters. The default `realscale` = `TRUE` only works for beta parameters that share the name of the real parameter to which they relate i.e. the beta parameter for the base level of the real parameter. This is because link functions are defined for real parameters not beta parameters.

The contours are approximate because they rely on interpolation. See Examples for a more reliable way to compare the likelihood at the MLE with nearby points on the surface.

Examples

```
## Not run:

LLsurface(secrdemo.CL, xval = seq(0.16,0.40,0.02),
    yval = 25:35, nlevels = 20)

## now verify MLE
```

```
## click on MLE and apparent `peak'
if (interactive()) {
    xy <- locator(2)
    LLsurface(secrdemo.CL, xval = xy$x, yval = xy$y, plot = FALSE)
}


## End(Not run)
```

---

logit *Logit Transformation*

Description

Transform real values to the logit scale, and the inverse.

Usage

```
logit(x)
invlogit(y)
```

Arguments

x	vector of numeric values in (0,1) (possibly a probability)
y	vector of numeric values

Details

The logit transformation is defined as $\mathrm{logit}(x) = \log(\frac{x}{1-x})$ for $x \in (0, 1)$.

Value

Numeric value on requested scale.

Note

`logit` is equivalent to `qlogis`, and `invlogit` is equivalent to `plogis` (both R functions in the **stats** package). `logit` and `invlogit` are used in **secr** because they are slightly more robust to bad input, and their names are more memorable!

Examples

```
logit(0.5)
invlogit(logit(0.2))
```

`logmultinom` *Multinomial Coefficient of SECR Likelihood*

Description

Compute the constant multinomial component of the SECR log likelihood

Usage

```
logmultinom(capthist, grp = NULL)
```

Arguments

`capthist`	`capthist` object
`grp`	factor defining group membership, or a list (see Details)

Details

For a particular dataset and grouping, the multinomial coefficient is a constant; it does not depend on the parameters and may be ignored when maximizing the likelihood to obtain parameter estimates. Nevertheless, the log likelihood reported by `secr.fit` includes this component *unless* the detector type is 'signal', 'polygon', 'polygonX', 'transect' or 'transectX' (from 2.0.0).

If `grp` is NULL then all animals are assumed to belong to one group. Otherwise, the length of `grp` should equal the number of rows of `capthist`.

`grp` may also be any vector that can be coerced to a factor. If `capthist` is a multi-session capthist object then `grp` should be a list with one factor per session.

If capture histories are not assigned to groups the value is the logarithm of

$$\binom{n}{n_1, ..., n_C} = \frac{n!}{n_1! n_2! ... n_C!}$$

where n is the total number of capture histories and n_1 ... n_C are the frequencies with which each of the C unique capture histories were observed.

If capture histories are assigned to G groups the value is the logarithm of

$$\prod_{g=1}^{G} \frac{n_g!}{n_{g1}! n_{g2}! ... n_{gC_g}!}$$

where n_g is the number of capture histories of group g and n_{g1} ... n_{gC_g} are the frequencies with which each of the C_g unique capture histories were observed for group g.

For multi-session data, the value is the sum of the single-session values. Both session structure and group structure therefore affect the value computed. Users will seldom need this function.

Value

The numeric value of the log likelihood component.

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., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture–recapture: likelihood-based methods. In: D. L. Thompson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer. Pp. 255–269.

See Also

stoatDNA

Examples

```
## no groups
logmultinom(stoatCH)
```

LR.test *Likelihood Ratio Test*

Description

Compute likelihood ratio test to compare two fitted models, one nested within the other.

Usage

```
LR.test(model1, model2)
```

Arguments

model1	fitted model
model2	fitted model

Details

The fitted models must be of a class for which there is a logLik method (e.g., 'secr' or 'lm'). Check with `methods("logLik")`.

The models must be nested (no check is performed - this is up to the user), but either model1 or model2 may be the more general model.

The models must also be compatible by the criteria of `AICcompatible`.

The test statistic is twice the difference of the maximized likelihoods. It is compared to a chi-square distribution with df equal to the number of extra parameters in the more complex model.

Value

Object of class 'htest', a list with components

statistic	value the test statistic
parameter	degrees of freedom of the approximate chi-squared distribution of the test statistic
p.value	probability of test statistic assuming chi-square distribution
method	character string indicating the type of test performed
data.name	character string with names of models compared

See Also

AICcompatible, AIC.secr, score.test

Examples

```
## two pre-fitted models
AIC (secrdemo.0, secrdemo.b)
LR.test  (secrdemo.0, secrdemo.b)
```

---

make.capthist *Construct capthist Object*

Description

Form a capthist object from a data frame of capture records and a traps object.

Usage

```
make.capthist(captures, traps, fmt = c("trapID", "XY"), noccasions = NULL,
    covnames = NULL, bysession = TRUE, sortrows = TRUE,
   cutval = NULL, tol = 0.01, snapXY = FALSE, noncapt = "NONE", signalcovariates)
```

Arguments

captures	dataframe of capture records in one of two possible formats (see Details)
traps	object of class traps describing an array of passive detectors
fmt	character string for capture format.
noccasions	number of occasions on which detectors were operated
covnames	character vector of names for individual covariate fields
bysession	logical, if true then ID are made unique by session
sortrows	logical, if true then rows are sorted in ascending order of animalID
cutval	numeric, threshold of signal strength for 'signal' detector type

tol	numeric, snap tolerance in metres
snapXY	logical; if TRUE then fmt = 'XY' uses nearest trap within tol for non-polygon detectors
noncapt	character value; animal ID used for 'no captures'
signalcovariates	character vector of field names from 'captures'

Details

make.capthist is the most flexible way to prepare data for secr.fit. See read.capthist for a more streamlined way to read data from text files for common detector types. Each row of the input data frame captures represents a detection on one occasion. The capture data frame may be formed from a text file with read.table.

Input formats are based on the Density software (Efford 2012; see also secr-datainput.pdf). If fmt = "XY" the required fields are (session, ID, occasion, x, y) in that order. If fmt = "trapID" the required fields are (session, ID, occasion, trap), where trap is the numeric index of the relevant detector in traps. session and ID may be character-, vector- or factor-valued; other required fields are numeric. Fields are matched by position (column number), *not* by name. Columns after the required fields are interpreted as individual covariates that may be continuous (e.g., size) or categorical (e.g., age, sex).

If captures has data from multiple sessions then traps may be either a list of traps objects, one per session, or a single traps object that is assumed to apply throughout. Similarly, noccasions may be a vector specifying the number of occasions in each session.

Covariates are assumed constant for each individual; the first non-missing value is used. The length of covnames should equal the number of covariate fields in captures.

bysession takes effect when the same individual is detected in two or more sessions: TRUE results in one capture history per session, FALSE has the effect of generating a single capture history (this is not appropriate for the models currently provided in **secr**).

Deaths are coded as negative values in the occasion field of captures. Occasions should be numbered 1, 2, ..., noccasions. By default, the number of occasions is the maximum value of 'occasion' in captures.

Signal strengths may be provided in the fifth (fmt = trapID) or sixth (fmt = XY) columns. Detections with signal strength missing (NA) or below 'cutval' are discarded.

A session may result in no detections. In this case a null line is included in captures using the animal ID field given by noncapt, the maximum occasion number, and any trapID (e.g. "sess1 NONE 5 1" for a 5-occasion session) (or equivalently "sess1 NONE 5 10 10" for fmt = XY).

Nonspatial data (Session, AnimalID, Occasion and possibly individual covariates) may be entered by omitting the 'traps' argument or setting it to NULL.

Value

An object of class capthist (a matrix or array of detection data with attributes for detector positions etc.). For 'single' and 'multi' detectors this is a matrix with one row per animal and one column per occasion (dim(capthist)=c(nc,noccasions)); each element is either zero (no detection) or a detector number (the row number in traps *not* the row name). For 'proximity' detectors capthist is an array of values {-1, 0, 1} and dim(capthist)=c(nc,noccasions,ntraps). The number of animals nc is

determined from the input, as is noccasions if it is not specified. traps, covariates and other data are retained as attributes of capthist.

Deaths during the experiment are represented as negative values in capthist.

For 'signal' and 'signalnoise' detectors, the columns of captures identified in signalcovariates are saved along with signal strength measurements in the attribute 'signalframe'.

If the input has data from multiple sessions then the output is an object of class c("capthist", "list") comprising a list of single-session capthist objects.

Note

make.capthist requires that the data for captures and traps already exist as R objects. To read data from external (text) files, first use read.table and read.traps, or try read.capthist for a one-step solution.

Prior to **secr** 4.4.0, occasional valid records for "multi" and "single" detectors were rejected as duplicates.

From **secr** 4.5.0, 'snapXY' works for transects as well as point detectors.

References

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

See Also

capthist, traps, read.capthist, secr.fit, sim.capthist

Examples

```
## peek at demonstration data
head(captXY)
head(trapXY)

demotraps <- read.traps(data = trapXY)
demoCHxy  <- make.capthist (captXY, demotraps, fmt = "XY")

demoCHxy            ## print method for capthist
plot(demoCHxy)      ## plot method for capthist
summary(demoCHxy)   ## summary method for capthist


## To enter `count' data without manually repeating rows
## need a frequency vector f, length(f) == nrow(captXY)
n <- nrow(captXY)
f <- sample (1:5, size = n, prob = rep(0.2,5), replace = TRUE)
## repeat rows as required...
captXY <- captXY[rep(1:n, f),]
counttraps <- read.traps(data = trapXY, detector = "count")
countCH  <- make.capthist (captXY, counttraps, fmt = "XY")
```

`make.lacework` *Construct Lacework Detector Design*

Description

A lacework design comprises a square grid with detectors placed at regular distances along the grid lines (Efford unpubl.). This requires fewer detectors than uniform coverage at close spacing and is simpler than clustered designs, while providing good spatial coverage and protection from alignment bias (Efford 2019).

Usage

```
make.lacework(region, spacing = c(100, 20), times = NULL, origin = NULL,
    rotate = 0, radius = NULL, detector = "multi", keep.design = TRUE)
```

Arguments

`region`	dataframe or SpatialPolygonsDataFrame with coordinates of perimeter
`spacing`	numeric 2-vector with major (grid) and minor spacings, or minor spacing only
`times`	numeric ratio major:minor spacing if spacing length 1
`origin`	numeric vector giving x- and y-cooordinates of fixed grid origin (origin is otherwise random)
`rotate`	numeric; number of degrees by which to rotate design clockwise about centroid of region bounding box
`radius`	numeric; detectors are dropped if they are further than this from a crossing
`detector`	character detector type – see `make.grid`
`keep.design`	logical; if TRUE then input argument values are retained

Details

It is tidy for the major spacing (`spacing[1]`) to be a multiple of the minor spacing (`spacing[2]`); precisely one detector is then placed at each grid intersection. This outcome may also be achieved by providing only the minor spacing in the `spacing` argument and specifying an integer value for `times`.

In general it is better not to specify `origin`. Specifying both `origin` and `rotate` may result in incomplete coverage, as the desired grid is relative to the bounding box of the *rotated* region.

Set `radius` < spacing[1]/2 to break lacework into multiple cross-shaped arrays centred on the intersections (crossing points) and truncated at `radius` metres (assuming you follow advice and express all linear measurements in metres).

The number of detectors should not exceed 5000.

Value

An **secr** traps object. The attribute 'crossings' is a 2-column matrix with the coordinates of the intersection points. If `keep.design` is TRUE then the input argument values are retained in attribute 'design' (a list with first component `function = 'make.lacework'`).

References

Efford, M. G. (2019) Non-circular home ranges and the estimation of population density. *Ecology* **100**, e02580. doi:10.1002/ecy.2580

See Also

`make.systematic`

Examples

```
trps <- make.lacework(possumarea, c(1000,100), rotate = 45, detector = 'proximity')
plot(trps, gridspace = 1000)
lines(possumarea)
points(attr(trps, 'crossings'), pch = 16)
```

`make.mask` *Build Habitat Mask*

Description

Construct a habitat mask object for spatially explicit capture-recapture. A mask object is a set of points with optional attributes.

Usage

```
make.mask(traps, buffer = 100, spacing = NULL, nx = 64, ny = 64, type =
    c("traprect", "trapbuffer", "pdot", "polygon", "clusterrect",
   "clusterbuffer", "rectangular", "polybuffer"), poly = NULL, poly.habitat = TRUE,
    cell.overlap = c("centre","any","all"), keep.poly = TRUE, check.poly = TRUE,
    pdotmin = 0.001, random.origin = FALSE,
    ...)
```

Arguments

`traps`	object of class `traps`
`buffer`	width of buffer in metres
`spacing`	spacing between grid points (metres)
`nx`	number of grid points in 'x' direction
`ny`	number of grid points in 'y' direction (type = 'rectangular')

type	character string for method
poly	bounding polygon to which mask should be clipped (see Details)
poly.habitat	logical for whether poly represents habitat or its inverse (non-habitat)
cell.overlap	character string for cell vertices used to determine overlap with 'poly'
keep.poly	logical; if TRUE any bounding polygon is saved as the attribute 'polygon'
check.poly	logical; if TRUE a warning is given for traps that lie outside a bounding polygon
pdotmin	minimum detection probability for inclusion in mask when type = "pdot" (optional)
random.origin	logical; if TRUE the mask coordinates are jittered
...	additional arguments passed to pdot when type = "pdot"

Details

The 'traprect' method constructs a grid of points in the rectangle formed by adding a buffer strip to the minimum and maximum x-y coordinates of the detectors in traps. Both 'trapbuffer' and 'pdot' start with a 'traprect' mask and drop some points.

The 'trapbuffer' method restricts the grid to points within distance buffer of any detector.

The 'pdot' method restricts the grid to points for which the net detection probability $p.(\mathbf{X})$ (see pdot) is at least pdotmin. Additional parameters are used by pdot (detectpar, noccasions). Set these with the ... argument; otherwise make.mask will silently use the arbitrary defaults. pdot is currently limited to a halfnormal detection function.

The 'clusterrect' method constructs a grid of rectangular submasks centred on 'clusters' of detectors generated with trap.builder (possibly indirectly by make.systematic). The 'clusterbuffer' method resembles 'trapbuffer', but is usually faster when traps are arranged in clusters because it starts with a 'clusterrect' mask.

The 'rectangular' method constructs a simple rectangular mask with the given nx, ny and spacing.

The 'polybuffer' method constructs a mask by buffering around the polygon specified in the 'poly' argument. If that inherits from 'SpatialPolygons' or 'sfc' then the buffering is performed with sf::st_buffer. Otherwise, buffering is approximate, based on the distance to points on an initial discretized mask enclosed by 'poly' (points at half the current 'spacing').

If poly is specified, points outside poly are dropped (unless type = "polybuffer"). The default is to require only the centre to lie within poly; use cell.overlap = "all" to require all cell corners to lie within poly, or cell.overlap = "any" to accept cells with any corner in poly. The 'polygon' method places points on a rectangular grid clipped to the polygon (buffer is not used). Thus 'traprect' is equivalent to 'polygon' when poly is supplied. poly may be either

- a matrix or dataframe of two columns interpreted as x and y coordinates, or
- an object from package 'sf' with polygon geometries, or
- a SpatialPolygons or SpatialPolygonsDataFrame object as defined in the package 'sp', possibly imported by reading a shapefile.

If spacing is not specified then it is determined by dividing the range of the x coordinates (including any buffer) by nx.

random.origin shifts the origin of the mask by a uniform random displacement within a spacing x spacing grid cell, while ensuring that the mask also satisfies the buffer requirement. random.origin

is available only for 'traprect', 'trapbuffer', 'polygon', and 'rectangular' types, and spacing must be specified.

Value

An object of class mask. When keep.poly = TRUE, poly and poly.habitat are saved as attributes of the mask.

Note

A warning is displayed if type = "pdot" and the buffer is too small to include all points with $p. >$ pdotmin.

A habitat mask is needed to fit an SECR model and for some related computations. The default mask settings in secr.fit may be good enough, but it is preferable to use make.mask to construct a mask in advance and to pass that mask as an argument to secr.fit.

The function bufferContour displays the extent of one or more 'trapbuffer' zones - i.e. the effect of buffering the detector array with varying strip widths.

See Also

mask, read.mask, subset.mask, pdot, bufferContour, deleteMaskPoints, as.mask

Examples

```
temptrap <- make.grid(nx = 10, ny = 10, spacing = 30)

## default method: traprect
tempmask <- make.mask(temptrap, spacing = 5)
plot(tempmask)
summary (tempmask)

## make irregular detector array by subsampling
## form mask by `trapbuffer' method
temptrap <- subset (temptrap, sample(nrow(temptrap), size = 30))
tempmask <- make.mask (temptrap, spacing = 5, type = "trapbuffer")
plot (tempmask)
plot (temptrap, add = TRUE)

## Not run:

## form mask by "pdot" method
temptrap <- make.grid(nx = 6, ny = 6)
tempmask <- make.mask (temptrap, buffer = 150, type = "pdot",
    pdotmin = 0.0001, detectpar = list(g0 = 0.1, sigma = 30),
    noccasions = 4)
plot (tempmask)
plot (temptrap, add = TRUE)

## Using an ESRI polygon shapefile for clipping (shapefile
## polygons may include multiple islands and holes).
```

```
library(sf)
shpfilename <- system.file("extdata/possumarea.shp", package = "secr")
possumarea <- st_read(shpfilename)

possummask2 <- make.mask(traps(possumCH), spacing = 20,
    buffer = 250, type = "trapbuffer", poly = possumarea)
par(mar = c(1,6,6,6), xpd = TRUE)
plot (possummask2, ppoly = TRUE)
plot(traps(possumCH), add = TRUE)
par(mar = c(5,4,4,2) + 0.1, xpd = FALSE)

## if the polygon delineates non-habitat ...
seaPossumMask <- make.mask(traps(possumCH), buffer = 1000,
    type = "traprect", poly = possumarea, poly.habitat = FALSE)
plot(seaPossumMask)
plot(traps(possumCH), add = TRUE)
## this mask is not useful!


## End(Not run)
```

`make.spcosa` *Construct Spatial Coverage Design*

Description

A spatial coverage design places one cluster of detectors in each compact subregion of a region of interest. Equal subregions are determined by *k*-means clustering of pixels (Walvoort et al. 2010).

Usage

```
make.spcosa(n, cluster, region, rotation = 0, randomize = FALSE, maxtries = 100,
    keep.mask = FALSE, ...)
```

Arguments

`n`	integer number of subregions
`cluster`	traps object defining a cluster of detectors
`region`	boundary of region of interest
`rotation`	numeric angular rotation of each cluster (negative for random)
`randomize`	logical; if TRUE then cluster is located at random within subregion
`maxtries`	integer maximum attempts to find random location
`keep.mask`	logical; if TRUE then a habitat mask with covariate 'stratum'
`...`	other arguments passed to `make.mask`

Details

The region may be specified in any form acceptable as the 'poly' argument of `make.mask` (see also boundarytoSF).

The ... argument determines the coarseness of the discretization used to define the subregions, via the 'nx' or 'spacing' arguments of `make.mask`.

By default (randomize = FALSE) clusters are centred at subregion centroids. Otherwise (randomize = TRUE) clusters are centred in a randomly selected cell of each subregion, subject to the constraint that all detectors fall within the subregion. An error results if no cluster meeting the constraint is found in 'maxtries' attempts.

Slightly different partitions of 'poly' are generated depending on the value of the random seed, so for consistency this should be first set with `set.seed`.

The argument 'rotation' is applied separately to each cluster, as in `trap.builder` and unlike the argument 'rotate' of `make.systematic`.

Value

A traps object with n x nrow(cluster) detectors.

References

Walvoort, D., Brus, D., and de Gruijter, J. (2010) An R package for spatial coverage sampling and random sampling from compact geographical strata by k-means. *Computers & Geosciences* 36:1261–1267.

See Also

`make.mask`, `trap.builder`, `boundarytoSF`, traps

Examples

```
# check package 'spcosa' installed
if (requireNamespace("spcosa")) {

# preliminaries
polygonfile <- system.file("extdata/possumarea.txt", package = "secr")
poly <- read.table(polygonfile, header = TRUE)
subgrid <- make.grid(3,3, spacing = 80)
set.seed(123)

# nx and keep.mask refer to the discretized region of interest
tr <- make.spcosa(n = 5, subgrid, poly, nx = 32, randomize = TRUE,
          keep.mask = TRUE)

plot(attr(tr,'mask'), dots = FALSE, cov = 'stratum', legend = FALSE)
plot(tr, add = TRUE)
lines(poly)

}
```

make.systematic *Construct Systematic Detector Design*

Description

A rectangular grid of clusters within a polygonal region.

Usage

```
make.systematic(n, cluster, region, spacing = NULL, origin = NULL,
    originoffset = c(0,0), chequerboard = c('all','black','white'),
    order = c('x', 'y', 'xb', 'yb'), rotate = 0,  centrexy = NULL,
    keep.design = TRUE, ...)
```

Arguments

n	integer approximate number of clusters (see Details)
cluster	traps object defining a single cluster
region	dataframe or SpatialPolygonsDataFrame with coordinates of perimeter
spacing	scalar distance between cluster centres
origin	vector giving x- and y-cooordinates of fixed grid origin (origin is otherwise random)
originoffset	numeric; 2-vector (x,y offsets); see Details
chequerboard	logical; if not 'all' then alternate clusters are omitted
order	character; sort order for clusters (see Details)
rotate	numeric; number of degrees by which to rotate entire design clockwise about centroid of region bounding box
centrexy	numeric; 2-vector for centre of rotation, if any
keep.design	logical; if TRUE then input argument values are retained
...	arguments passed to trap.builder

Details

region may be any shape.

region may be one of the spatial classes described in boundarytoSF. Otherwise, region should be a dataframe with columns 'x' and 'y'.

spacing may be a vector with separate values for spacing in x- and y- directions. If spacing is provided then n is ignored.

If n is a scalar, the spacing of clusters is determined from the area of the bounding box of region divided by the requested number of clusters (this does not necessarily result in exactly n clusters). If n is a vector of two integers these are taken to be the number of columns and the number of rows.

After preparing a frame of cluster centres, `make.systematic` calls `trap.builder` with method = 'all'; ... allows the arguments 'rotation', 'edgemethod', 'plt', and 'detector' to be passed. Setting the `trap.builder` arguments `frame`, `method`, and `samplefactor` has no effect.

Note the distinction between argument `rotate` and the `trap.builder` argument `rotation` that is applied separately to each cluster.

If `origin` is not specified then a random uniform origin is chosen within a box (width = spacing) placed with its bottom left corner at the bottom left of the bounding box of `region`, shifted by `originoffset`. Before version 3.1.8 the behaviour of `make.systematic` was equivalent to `originoffset = c(wx,wy)` where `wx,wy` are the cluster half widths.

`chequerboard = "black"` retains black 'squares' and `chequerboard = "white"` retains white 'squares', where the lower left cluster in the candidate rectangle of cluster origins is black, as on a chess board. The effect is the same as increasing spacing by sqrt(2) and rotating through 45 degrees.

`order` determines the ordering of clusters in the resulting traps object. The options are a subset of those for `ID` argument of `make.grid`:

Option	Sort order
x	column-dominant
y	row-dominant
xb	column-dominant boustrophedonical (alternate columns reversed)
yb	row-dominant boustrophedonical (alternate rows reversed)

`rotate` rotates the array about the given centre (default is centroid of the bounding box of `region`).

Value

A single-session 'traps' object.

From 3.2.0 these additional attributes are set –

origin	coordinates of grid origin
centres	coordinates of true cluster centres (cf `cluster.centres`)
originbox	vertices of rectangular spatial sampling frame for random origin

From 4.2.0 if `keep.design` is TRUE then the input argument values are retained in attribute 'design' (a list with first component `function = 'make.systematic'`).

Note

Do not confuse with the simpler function `make.grid`, which places single detectors in a rectangular array.

See Also

`trap.builder`, `make.lacework`, `cluster.centres`

Examples

```
mini <- make.grid(nx = 2, ny = 2, spacing = 100)
region <- cbind(x=c(0,2000,2000,0), y=c(0,0,2000,2000))
temp <- make.systematic(25, mini, region, plt = TRUE)
temp <- make.systematic(c(6, 6), mini, region, plt = TRUE,
    rotation = -1)

## Example using shapefile "possumarea.shp" in
## "extdata" folder. By default, each cluster is
## a single multi-catch detector

## Not run:

library(sf)
shpfilename <- system.file("extdata/possumarea.shp", package = "secr")
possumarea <- st_read(shpfilename)

possumgrid <- make.systematic(spacing = 100, region =
    possumarea, plt = TRUE)

## or with 2 x 2 clusters
possumgrid2 <- make.systematic(spacing = 300,
    cluster = make.grid(nx = 2, ny = 2, spacing = 100),
    region = possumarea, plt = TRUE, edgemethod =
    "allinside")
## label clusters
text(cluster.centres(possumgrid2), levels(clusterID
    (possumgrid2)), cex=0.7)

## If you have GPSBabel installed and on the Path
## then coordinates can be projected and uploaded
## to a GPS with `writeGPS', which also requires the
## package `proj4'. Defaults are for a Garmin GPS
## connected by USB.

if (interactive()) {
    writeGPS(possumgrid, proj = "+proj=nzmg")
}


## End(Not run)
```

`make.traps` *Build Detector Array*

Description

Construct a rectangular array of detectors (trapping grid) or a circle of detectors or a polygonal search area.

Usage

```
make.grid(nx = 6, ny = 6, spacex = 20, spacey = spacex, spacing = NULL,
    detector = "multi", originxy = c(0,0), hollow = FALSE, ID = "alphay",
    leadingzero = TRUE, markocc = NULL, centre = FALSE)

make.circle (n = 20, radius = 100, spacing = NULL,
    detector = "multi", originxy = c(0,0), IDclockwise = TRUE,
    leadingzero = TRUE, markocc = NULL)

make.poly (polylist = NULL, x = c(-50,-50,50,50),
    y = c(-50,50,50,-50), exclusive = FALSE, verify = TRUE)

make.transect (transectlist = NULL, x = c(-50,-50,50,50),
    y = c(-50,50,50,-50), exclusive = FALSE)

make.telemetry (xy = c(0,0))
```

Arguments

nx	number of columns of detectors
ny	number of rows of detectors
spacex	distance between detectors in ‘x’ direction (nominally in metres)
spacey	distance between detectors in ‘y’ direction (nominally in metres)
spacing	distance between detectors (x and y directions)
detector	character value for detector type - "single", "multi" etc.
originxy	vector origin for x-y coordinates
hollow	logical for hollow grid
ID	character string to control row names
leadingzero	logical; if TRUE numeric rownames are padded with leading zeros
markocc	integer vector of marking or sighting codes; see markocc
centre	logical; if TRUE grid is centred on originxy
n	number of detectors
radius	radius of circle (nominally in metres)
IDclockwise	logical for numbering of detectors
polylist	list of dataframes with coordinates for polygons
transectlist	list of dataframes with coordinates for transects
x	x coordinates of vertices
y	y coordinates of vertices
exclusive	logical; if TRUE animal can be detected only once per occasion
verify	logical if TRUE then the resulting traps object is checked with verify
xy	vector with coordinates of arbitrary point (e.g., centroid of fixes)

Details

make.grid generates coordinates for nx.ny traps at separations spacex and spacey. If spacing is specified it replaces both spacex and spacey. The bottom-left (southwest) corner is at originxy. For a hollow grid, only detectors on the perimeter are retained. By default, identifiers are constructed from a letter code for grid rows and an integer value for grid columns ("A1", "A2",...). 'Hollow' grids are always numbered clockwise in sequence from the bottom-left corner. Other values of ID have the following effects:

ID	Effect
numx	column-dominant numeric sequence
numy	row-dominant numeric sequence
numxb	column-dominant boustrophedonical numeric sequence (try it!)
numyb	row-dominant boustrophedonical numeric sequence
alphax	column-dominant alphanumeric
alphay	row-dominant alphanumeric
xy	combine column (x) and row(y) numbers

'xy' adds leading zeros as needed to give a string of constant length with no blanks.

make.circle generates coordinates for n traps in a circle centred on originxy. If spacing is specified then it overrides the radius setting; the radius is adjusted to provide the requested straightline distance between adjacent detectors. Traps are numbered from the trap due east of the origin, either clockwise or anticlockwise as set by IDclockwise.

Polygon vertices may be specified with x and y in the case of a single polygon, or as polylist for one or more polygons. Each component of polylist is a dataframe with columns 'x' and 'y'. polylist takes precedence. make.poly automatically closes the polygon by repeating the first vertex if the first and last vertices differ.

Transects are defined by a sequence of vertices as for polygons, except that they are not closed.

make.telemetry builds a simple traps object for the 'telemetry' detector type. The attribute 'telemetrytype' is set to "independent".

Specialised functions for arrays using a triangular grid are described separately (make.tri, clip.hex).

Value

An object of class traps comprising a data frame of x- and y-coordinates, the detector type ("single", "multi", or "proximity" etc.), and possibly other attributes.

Note

Several methods are provided for manipulating detector arrays - see traps.

References

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

See Also

read.traps,detector, trap.builder,make.systematic, print.traps, plot.traps, traps, make.tri, addTelemetry

Examples

```
demo.traps <- make.grid()
plot(demo.traps)

## compare numbering schemes
par (mfrow = c(2,4), mar = c(1,1,1,1), xpd = TRUE)
for (id in c("numx", "numy", "alphax", "alphay", "numxb",
    "numyb"))
{
    temptrap <- make.grid(nx = 7, ny = 5, ID = id)
    plot (temptrap, border = 10, label = TRUE, offset = 7,
        gridl = FALSE)
}

temptrap <- make.grid(nx = 7, ny = 5, hollow = TRUE)
plot (temptrap, border = 10, label = TRUE, gridl = FALSE)

plot(make.circle(n = 20, spacing = 30), label = TRUE, offset = 9)
summary(make.circle(n = 20, spacing = 30))

## jitter locations randomly within grid square
## and plot over `mask'
# see also ?gridCells

tr0 <- tr <- make.grid(nx = 7, ny = 7, spacing = 30)
tr[] <- jitter(unlist(tr), amount = spacing(tr)/2)
plot(as.mask(tr0), dots = FALSE, mesh = 'white')
plot(tr, add = TRUE)
```

---

make.tri *Build Detector Array on Triangular or Hexagonal Grid*

Description

Construct an array of detectors on a triangular grid and optionally select a hexagonal subset of detectors.

Usage

```
make.tri (nx = 10, ny = 12, spacing = 20, detector = "multi",
    originxy = c(0,0))

clip.hex (traps, side = 20, centre = c(50, 60*cos(pi/6)),
    fuzz = 1e-3, ID = "num", ...)
```

Arguments

nx	number of columns of detectors
ny	number of rows of detectors
spacing	distance between detectors (x and y directions)
detector	character value for detector type - "single", "multi" etc.
originxy	vector origin for x-y coordinates
traps	traps object
side	length of hexagon side
centre	x-y coordinates of hexagon centre
fuzz	floating point fuzz value
ID	character string to control row names
...	other parameters passed to subset.traps (not used)

Details

`make.tri` generates coordinates for `nx.ny` traps at separations `spacing`. The bottom-left (south-west) corner is at `originxy`. Identifiers are numeric. See `make.grid` for further explanation.

`clip.hex` clips a grid of detectors, retaining only those within a bounding hexagon. Detectors are re-labelled according to `ID` as follows:

ID	Effect
NULL	no change
num	numeric sequence
alpha	letter for‘shell’; number within shell

Value

An object of class `traps` comprising a data frame of x- and y-coordinates, the detector type ("single", "multi", or "proximity" etc.), and possibly other attributes.

Note

Several methods are provided for manipulating detector arrays - see `traps`.

See Also

`make.grid`, `detector`

Examples

```
tri.grid <- make.tri(spacing = 10)
plot(tri.grid, border = 5)

hex <- clip.hex(tri.grid, side = 30, ID = "alpha")
plot (hex, add = TRUE, detpar = list(pch = 16, cex = 1.4),
    label = TRUE, offset = 2.5 )
```

---

makeStart *Initial Parameter Values*

Description

makeStart() wraps the code previously internal to secr.fit(). It will not usually be called directly.

Usage

```
makeStart(start = NULL, parindx, capthist, mask, detectfn, link,
    details = NULL, fixed = NULL, CL = FALSE, anypoly = FALSE,
    anytrans = FALSE, alltelem = FALSE, sighting = FALSE)
```

Arguments

start	optional starting values as described for secr.fit
parindx	list with one component per real parameter giving corresponding indices of coefficients (beta parameters)
capthist	capthist object including capture data and detector (trap) layout
mask	mask object or (for a multi-session analysis) a list of mask objects, one for each session
detectfn	integer code or character string for shape of detection function 0 = halfnormal, 1 = hazard rate etc. – see detectfn
link	list of link function names ("log", "logit", "identity") for each real parameter
details	list of additional arguments (see secr.fit)
fixed	list with optional components corresponding to real parameters giving the scalar value to which the parameter is to be fixed
CL	logical, if true then model does not include density
anypoly	logical, TRUE if any polygon detectors
anytrans	logical, TRUE if any transect detectors
alltelem	logical, TRUE if any telemetry detectors
sighting	logical, TRUE if any sighting data

Details

If 'start' is a previously fitted model then only 'start' and 'parindx' are required.

Value

Numeric vector with one value for each coefficient (beta parameter) in model.

See Also

secr.fit

Examples

```
makeStart(secrdemo.0, list(D = 1, g0 = 2:3, sigma = 4))
```

---

mask *Mask Object*

Description

Encapsulate a habitat mask for spatially explicit capture–recapture. See also secr-habitatmasks.pdf.

Details

A habitat mask serves four main purposes in spatially explicit capture–recapture. Firstly, it defines an outer limit to the area of integration; habitat beyond the mask may be occupied, but animals there should have negligible chance of being detected (see pdot and below). Secondly, it distinguishes sites in the vicinity of the detector array that are 'habitat' (i.e. have the potential to be occupied) from 'non-habitat'. Thirdly, it discretizes continuous habitat as a list of points. Each point is notionally associated with a cell (pixel) of uniform density. Discretization allows the SECR likelihood to be evaluated by summing over grid cells. Fourthly, the x-y coordinates of the mask and any habitat covariates may be used to build spatial models of density. For example, a continuous or categorical habitat covariate 'cover' measured at each point on the mask might be used in a formula for density such as D ~cover.

In relation to the first purpose, the definition of 'negligible' is fluid. Any probability less than 0.001 seems OK in the sense of not causing noticeable bias in density estimates, but this depends on the shape of the detection function (fat-tailed functions such as 'hazard rate' are problematic). New tools for evaluating masks appeared in **secr** 1.5 (mask.check, esaPlot), and suggest.buffer automates selection of a buffer width.

Mask points are stored in a data frame with columns 'x' and 'y'. The number of rows equals the number of points.

Possible mask attributes

Attribute	Description
type	'traprect', 'trapbuffer', 'pdot', 'polygon', 'clusterrect', 'clusterbuffer' (see make.mask) or 'user'

polygon	vertices of polygon defining habitat boundary, for type = 'polygon'
pdotmin	threshold of p.(X) for type = 'pdot'
covariates	dataframe of site-specific covariates
meanSD	data frame with centroid (mean and SD) of x and y coordinates
area	area (ha) of the grid cell associated with each point
spacing	nominal spacing (metres) between adjacent points
boundingbox	data frame of 4 rows, the vertices of the bounding box of all grid cells in the mask

Attributes other than `covariates` are generated automatically by `make.mask`. Type 'user' refers to masks input from a text file with `read.mask`.

A virtual S4 class 'mask' is defined to allow the definition of a method for the generic function `raster` from the **raster** package.

Note

A habitat mask is needed by `secr.fit`, but one will be generated automatically if none is provided. You should be aware of this and check that the default settings (e.g. `buffer`) are appropriate.

See Also

`make.mask`, `read.mask`, `mask.check`, `esaPlot`, `suggest.buffer`, `secr.fit`

`mask.check` *Mask Diagnostics*

Description

`mask.check` evaluates the effect of varying buffer width and mask spacing on either the likelihood or density estimates from secr.fit().

Usage

```
mask.check(object, buffers = NULL, spacings = NULL, poly = NULL,
    LLonly = TRUE, realpar = NULL, session = 1, file = NULL,
    drop = "", tracelevel = 0, ...)
```

Arguments

`object`	object of class 'capthist' or 'secr'
`buffers`	vector of buffer widths
`spacings`	vector of mask spacings
`poly`	matrix of two columns, the x- and y-coordinates of a bounding polygon (optional)
`LLonly`	logical; if TRUE then only the log likelihood is computed

realpar	list of parameter values
session	vector of session indices (used if object spans multiple sessions)
file	name of output file (optional)
drop	character vector: names of fitted secr object to omit
tracelevel	integer for level of detail in reporting (0,1,2)
...	other arguments passed to secr.fit

Details

Masks of varying buffer width and spacing are constructed with the 'trapbuffer' method in make.mask, using the detector locations ('traps') from either a capthist object or a previous execution of secr.fit. Default values are provided for buffers and spacings if object is of class 'secr' (respectively c(1, 1.5, 2) and c(1, 0.75, 0.5) times the values in the existing mask). The default for buffers will not work if a detector is on the mask boundary, as the inferred buffer is then 0.

Variation in the mask may be assessed for its effect on –

- the log-likelihood evaluated for given values of the parameters (LLonly = TRUE)
- estimates from maximizing the likelihood with each mask (LLonly = FALSE)

realpar should be a list with one named component for each real parameter (see Examples). It is relevant only if LLonly = TRUE. realpar may be omitted if object is of class 'secr'; parameter values are then extracted from object.

session should be an integer or character vector suitable for indexing sessions in object, or in object$capthist if object is a fitted model. Each session is considered separately; a model formula that refers to session or uses session covariates will cause an error.

If file is specified then detailed results (including each model fit when LLonly = FALSE) are saved to an external .RData file. Loading this file creates or overwrites object(s) in the workspace: mask.check.output if LLonly = TRUE, otherwise mask.check.output and mask.check.fit. For multiple sessions these are replaced by lists with one component per session (mask.check.outputs and mask.check.fits). The drop argument is passed to trim and applied to each fitted model; use it to save space, at the risk of limiting further computation on the fitted models.

tracelevel>0 causes more verbose reporting of progress during execution.

The ... argument may be used to override existing settings in object - for example, a conditional likelihood fit (CL = T) may be selected even if the original model was fitted by maximizing the full likelihood.

Value

Array of log-likelihoods (LLonly = TRUE) or estimates (LLonly = FALSE) for each combination of buffers and spacings. The array has 3 dimensions if LLonly = FALSE and both buffers and spacings have multiple levels; otherwise it collapses to a matrix. Rows generally represent buffers, but rows represent spacings if a single buffer is specified.

Warning

mask.check() may fail if object is a fitted 'secr' model and a data object named in the original call of secr.fit() (i.e. object$call) is no longer in the working environment (secr.fit arguments capthist, mask, verify & trace are exempt). Fix by any of (1) applying mask.check directly to the 'capthist' object, specifying other arguments (buffers, spacings, realpar) as needed, (2) re-fitting the model and running mask.check in the same environment, (3) specifying the offending argument(s) in ..., or (4) re-creating the required data objects(s) in the working environment, possibly from saved inputs in object (e.g., mytimecov <- myfit$timecov).

Note

When LLonly = TRUE the functionality of mask.check resembles that of the 'Tools | ML SECR log likelihood' menu option in Density 5. The help page in Density 5 for ML SECR 2-D integration (see index) may be helpful.

Warning messages from secr.fit are suppressed. 'capthist' data provided via the object argument are checked with verify.capthist if tracelevel > 0.

The likelihood-only method is fast, but not definitive. It is reasonable to aim for stability in the third decimal place of the log likelihood. Slight additional variation in the likelihood may cause little change in the estimates; the only way to be sure is to check these by setting LLonly = FALSE.

The performance of a mask depends on the detection function; be sure to set the detectfn argument appropriately. The hazard rate function has a fat tail that can be problematic.

When provided with an 'secr' object, mask.check constructs a default vector of buffer widths as multiples of the buffer used in object *even though that value is not saved explicitly*. For this trick, detector locations in traps(object$capthist) are compared to the bounding box of object$mask; the base level of buffer width is the maximum possible within the bounding box.

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*. Department of Mathematics and Statistics, University of Otago, Dunedin, New Zealand. https://www.otago.ac.nz/density/.

See Also

esaPlot, make.mask, secr.fit

Examples

```
## Not run:

## from a capthist object, specifying almost everything
mask.check (possumCH, spacings = c(20, 30), buffers =c(200, 300),
    realpar = list(g0 = 0.2, sigma = 50), CL = TRUE)

## from a fitted model, using defaults
mask.check (stoat.model.HN)
```

```
## LL did not change with varying buffer (rows) or spacing (cols):
##         78.125  58.59375   39.0625
## 1000 -144.0015 -144.0015 -144.0015
## 1500 -144.0017 -144.0017 -144.0017
## 2000 -144.0017 -144.0017 -144.0017

## fit new models for each combination of buffer & spacing,
## and save fitted models to a file
mask.check (stoat.model.HN, buffers = 1500, spacings =
    c(40,60,80), LLonly = FALSE, file = "test", CL = TRUE)

## look in more detail at the preceding fits
## restores objects `mask.check.output' and `mask.check.fit'
load("test.RData")
lapply(mask.check.fit, predict)
lapply(mask.check.fit, derived)

## multi-session data
mask.check(ovenbird.model.1, session = c("2005","2009"))

## clipping mask
txtfilename <- system.file("extdata/possumarea.txt", package = "secr")
possumarea <- read.table(txtfilename, header = TRUE)
mask.check (possum.model.0, spacings = c(20, 30), buffers =
    c(200, 300), poly = possumarea, LLonly = FALSE,
    file = "temp", CL = TRUE)

## review fitted models
load ("temp.RData")
par(mfrow = c(2,2), mar = c(1,4,4,4))
for (i in 1:4) {
    plot(traps(mask.check.fit[[i]]$capthist), border = 300,
        gridlines = FALSE)
    plot(mask.check.fit[[i]]$mask, add = TRUE)
    lines(possumarea)
    text ( 2698618, 6078427, names(mask.check.fit)[i])
    box()
}
par(mfrow = c(1,1), mar = c(5,4,4,2) + 0.1)    ## defaults


## End(Not run)
```

MCgof *Monte Carlo Goodness-of-fit for SECR Models*

Description

MCgof implements and extends the Monte Carlo resampling method of Choo et al. (2024) to emulate Bayesian posterior predictive checks (Gelman et al. 1996, Royle et al. 2014). Initial results suggest

the approach is more informative than the deviance-based test proposed by Borchers and Efford (2008) and implemented in secr.test. However, the tests have limited power.

MCgof is under development. The structure of the output may change and bugs may be found. See Warning below for exclusions.

Usage

```
## S3 method for class 'secr'
MCgof(object, nsim = 100, statfn = NULL, testfn = NULL, seed = NULL,
    ncores = 1, clustertype = c("PSOCK", "FORK"), usefxi = TRUE,
    useMVN = TRUE, Ndist = NULL, quiet = FALSE, debug = FALSE, ...)

## S3 method for class 'secrlist'
MCgof(object, nsim = 100, statfn = NULL, testfn = NULL, seed = NULL,
    ncores = 1, clustertype = c("PSOCK", "FORK"), usefxi = TRUE,
    useMVN = TRUE, Ndist = NULL, quiet = FALSE, debug = FALSE, ...)
```

Arguments

object	secr fitted model or secrlist object
nsim	integer number of replicates
statfn	function to extract summary statistics from capture histories
testfn	function to compare observed and expected counts
seed	integer seed
ncores	integer for number of parallel cores
clustertype	character cluster type for parallel::makeCluster
usefxi	logical; if FALSE then AC are simulated de novo from the density process rather than using information on the detected individuals
useMVN	logical; if FALSE parameter values are fixed at the MLE rather than drawn from multivariate normal distribution
Ndist	character; distribution of number of unobserved AC (optional)
quiet	logical; if FALSE then a progress bar (ncores=1) and final timing are shown
debug	integer; if >0 then the browser is started at one of 4 points in code
...	other arguments passed to testfn

Details

At each replicate parameter values are sampled from the multivariate-normal sampling distribution of the fitted model. The putative location of each detected individual is drawn from the spatial distribution implied by its observations and the resampled parameters (see fxi); locations of undetected individuals are simulated from the complement of pdot(x) times D(x).

New detections are simulated under the model for individuals at the simulated locations, along with the expected numbers. Detections form a capthist object, a 3-D array with dimensions for individual i, occasion j and detector $k*$. Thus for each replicate and detected individual there are the original

observations y_{ijk}, simulated observations Y_{ijk}, and expected counts $\mathrm{E}(y_{ijk})$. Two discrepancy statistics are calculated for each replicate – observed vs expected counts, and simulated vs expected counts – and a record is kept of which of these discrepancy statistics is the larger (indicating poorer fit).

* Notation differs slightly from Choo et al. (2024), using j for occasion and k for detector to be consistent with usage in **secr** and elsewhere (e.g., Borchers and Fewster 2016).

The default discrepancy (`testfn`) is the Freeman-Tukey statistic as in Choo et al. (2024) and Royle et al. (2014) (see also Brooks, Catchpole and Morgan 2000). The statistic has this general form for M counts y_m with expected value $\mathrm{E}(y_m)$:

$$T = \sum_{m=1}^{m=M} \left(\sqrt{y_m} - \sqrt{E(y_m)}\right)^2.$$

The key output of `MCgof` is the proportion of replicates in which the simulated discrepancy exceeds the observed discrepancy. For perfect fit this will be about 0.5, and for poor fit it will approach zero.

By default, tests are performed separately for three types of count: the numbers of detections of each individual (yi), at each detector (yk), and for each individual at each detector (yik) extracted by the default `statfn` from the margins of the observed and simulated capture histories.

$y_{ik} = \sum_j y_{ijk}$	individual x detector
$y_i = \sum_j \sum_k y_{ijk}$	individual
$y_k = \sum_i \sum_j y_{ijk}$	detector

Parallel processing is offered using multiple cores (CPUs) through the package **parallel** when ncores > 1. This differs from the usual multithreading paradigm in **secr** and does not rely on the environment variable set by `setNumThreads` *except* that, if ncores = NULL, ncores will be set to the value from `setNumThreads`. The cluster type "FORK" is available only on Unix-like systems; it can require large amounts of memory, but is generally fast. A small value of ncores>1 may be optimal, especially with cluster type "PSOCK".

'usefxi' and 'useMVN' may be used to drop key elements of the Choo et al. (2024) approach - they are provided for demonstration only.

'Ndist' refers to the distribution of the number of unobserved AC, conditional on the expected number $q = D^*A - n$ where D^* is the resampled density, A the mask area, and n the number of detected individuals. By default 'Ndist' depends on the distribution component of the 'details' argument of the fitted model ("poisson" for Poisson n, "fixed"" for binomial n).

'debug' may be used to view intermediate data at certain points in MCgof() numbered 1 to 5. Examine the code of secr:::MCgof.secr or secr:::simfxiAC for these points. Debugging requires 'ncores = 1'.

The `RNGkind` of the random number generator is set internally for consistency across platforms.

The ... argument may be used to pass 'np' and 'verbose' to 'Fletcher.chat' used as 'testfn'.

Value

Invisibly returns an object of class 'MCgof' with components -

`nsim`	as input
`statfn`	as input or default
`testfn`	as input or default
`all`	list of outputs: for each statistic, a 3 x nsim matrix. Rows correspond to Tobs, Tsim, and a binary indicator for Tsim > Tobs
`proctime`	execution time in seconds

For secrlist input the value returned is a list of 'MCgof' objects.

Warning

Not all models are covered and some are untested. These models are specifically excluded -

1. multi-session models
2. models with groups
3. conditional likelihood
4. polygon, transect, telemetry or signal detectors
5. non-binary behavioural responses

Notes

This implementation extends the work of Choo et al. (2024) in these respects -

1. detector types 'multi' and 'count' are allowed
2. the model may include variation among detectors
3. the model may include behavioural responses
4. 2-class finite mixture and hybrid mixture models are both allowed.

Author(s)

Murray Efford and Yan Ru Choo

References

Borchers, D. L. and Efford, M. G. (2008) Spatially explicit maximum likelihood methods for capture–recapture studies. *Biometrics* **64**, 377–385. doi:10.1111/j.15410420.2007.00927.x

Borchers, D. L. and Fewster, R. M. (2016) Spatial capture–recapture models. *Statistical Science* **31**, 219–232. doi:10.1214/16sts557

Brooks, S. P., Catchpole, E. A. and Morgan, B. J. T. (2000) Bayesian animal survival estimation. *Statistical Science* **15**, 357–376.

Choo, Y. R., Sutherland, C. and Johnston, A. (2024) A Monte Carlo resampling framework for implementing goodness-of-fit tests in spatial capture-recapture model *Methods in Ecology and Evolution* **15**, doi:10.1111/2041210X.14386.

Gelman, A., Meng, X.-L., and Stern, H. (1996) Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica* **6**, 733–807.

Royle, J. A., Chandler, R. B., Sollmann, R. and Gardner, B. (2014) *Spatial capture–recapture.* Academic Press.

See Also

Parallel, secr.test, plot.MCgof, hist.MCgof, summary.MCgof

Examples

```
tmp <- MCgof(secrdemo.0)
summary(tmp)
par(mfrow = c(1,3), pty = 's')
plot(tmp)
```

---

modelAverage *Averaging of SECR Models Using Akaike's Information Criterion*

Description

AIC- or AICc-weighted average of estimated 'real' or 'beta' parameters from multiple fitted secr models, and the tabulation of estimates.

Usage

```
## S3 method for class 'secr'
modelAverage(object, ..., realnames = NULL, betanames = NULL, newdata = NULL,
    alpha = 0.05, dmax = 10, covar = FALSE, average = c("link", "real"),
    criterion = c("AIC","AICc"), CImethod = c("Wald", "MATA"), chat = NULL)

## S3 method for class 'secrlist'
modelAverage(object, ..., realnames = NULL, betanames = NULL, newdata = NULL,
    alpha = 0.05, dmax = 10, covar = FALSE, average = c("link", "real"),
    criterion = c("AIC","AICc"), CImethod = c("Wald", "MATA"), chat = NULL)
```

Arguments

object	secr or secrlist object
...	other secr objects
realnames	character vector of real parameter names
betanames	character vector of beta parameter names
newdata	optional dataframe of values at which to evaluate models
alpha	alpha level for confidence intervals
dmax	numeric, the maximum AIC or AICc difference for inclusion in confidence set
covar	logical, if TRUE then return variance-covariance matrix
average	character string for scale on which to average real parameters

`criterion`	character, information criterion to use for model weights
`CImethod`	character, type of confidence interval (see Details)
`chat`	numeric optional variance inflation factor for quasi-AIC weights

Details

Models to be compared must have been fitted to the same data and use the same likelihood method (full vs conditional). If `realnames` = NULL and `betanames` = NULL then all real parameters will be averaged; in this case all models must use the same real parameters. To average beta parameters, specify `betanames` (this is ignored if a value is provided for `realnames`). See `predict.secr` for an explanation of the optional argument `newdata`; `newdata` is ignored when averaging beta parameters.

Model-averaged estimates for parameter θ are given by

$$\hat{\theta} = \sum_k w_k \hat{\theta}_k$$

where the subscript k refers to a specific model and the w_k are AIC or AICc weights (see `AIC.secr` for details). Averaging of real parameters may be done on the link scale before back-transformation (`average="link"`) or after back-transformation (`average="real"`).

Models for which dAIC > `dmax` (or dAICc > `dmax`) are given a weight of zero and effectively are excluded from averaging.

Also,

$$\mathrm{var}(\hat{\theta}) = \sum_k w_k (\mathrm{var}(\hat{\theta}_k | \beta_k) + \beta_k^2)$$

where $\hat{\beta}_k = \hat{\theta}_k - \hat{\theta}$ and the variances are asymptotic estimates from fitting each model k. This follows Burnham and Anderson (2004) rather than Buckland et al. (1997).

Two methods are offered for confidence intervals. The default 'Wald' uses the above estimate of variance. The alternative 'MATA' (model-averaged tail area) avoids estimating a weighted variance and is thought to provide better coverage at little cost in increased interval length (Turek and Fletcher 2012). Turek and Fletcher (2012) also found averaging with AIC weights (here `criterion = 'AIC'`) preferable to using AICc weights, even for small samples. `CImethod` does not affect the reported standard errors.

If 'chat' is provided then quasi-AIC or quasi-AICc weights are used, depending on the value of 'criterion'.

Value

For `modelAverage`, an array of model-averaged estimates, their standard errors, and a $100(1-\alpha)\%$ confidence interval. The interval for real parameters is backtransformed from the link scale. If there is only one row in `newdata` or beta parameters are averaged or averaging is requested for only one parameter then the array is collapsed to a matrix. If `covar = TRUE` then a list is returned with separate components for the estimates and the variance-covariance matrices.

Note

`modelAverage` replaces the deprecated function `model.average` whose name conflicted with a method in **RMark**.

References

Buckland S. T., Burnham K. P. and Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.

Burnham, K. P. and Anderson, D. R. (2002) *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Second edition. New York: Springer-Verlag.

Burnham, K. P. and Anderson, D. R. (2004) Multimodel inference - understanding AIC and BIC in model selection. *Sociological Methods & Research* **33**, 261–304.

Turek, D. and Fletcher, D. (2012) Model-averaged Wald confidence intervals. *Computational statistics and data analysis* **56**, 2809–2815.

See Also

modelAverage, AIC.secr, secr.fit, collate

Examples

```
## Compare two models fitted previously
## secrdemo.0 is a null model
## secrdemo.b has a learned trap response

modelAverage(secrdemo.0, secrdemo.b)
modelAverage(secrdemo.0, secrdemo.b, betanames = c("D","g0","sigma"))

## In this case we find the difference was actually trivial...
## (subscripting of output is equivalent to setting fields = 1)
```

---

ms *Multi-session Objects*

Description

Logical function to distinguish objects that span multiple sessions

Usage

```
  ## Default S3 method:
ms(object, ...)
  ## S3 method for class 'mask'
ms(object, ...)
  ## S3 method for class 'secr'
ms(object, ...)
```

Arguments

object	any object
...	other arguments (not used)

Details

The test applied varies with the type of object. The default method uses `inherits(object, "list")`.

Value

logical, TRUE if `object` contains data for multiple sessions

See Also

`capthist`, `mask`, `secr.fit`

Examples

```
ms(ovenCH)
ms(ovenbird.model.1)
ms(ovenCH[[1]])
```

---

newdata — *Create Default Design Data*

Description

Internal function used to generate a dataframe containing design data for the base levels of all predictors in an secr object.

Usage

```
## Default S3 method:
makeNewData(object, all.levels = FALSE, ...)
## S3 method for class 'secr'
makeNewData(object, all.levels = FALSE, bytrap = FALSE, ...)
```

Arguments

`object`	fitted secr model object
`all.levels`	logical; if TRUE then all levels of factors are included
`bytrap`	logical; if TRUE then all detectors are included
`...`	other arguments (not used)

Details

`makeNewData` is used by `predict` in lieu of user-specified 'newdata'. There is seldom any need to call the function `makeNewData` directly.

With 'bytrap' there is a row for each detector, with columns for the corresponding levels of any detector covariates that appear in the model.

Value

A dataframe with one row for each session and group, and columns for the predictors used by `object$model`.

See Also

`predict.secr`, `secr.fit`

Examples

```
## from previously fitted model
makeNewData(secrdemo.b)
```

---

nk — *Number per detector*

Description

Functions described here compute the observed (nk) or expected (Enk) number of individuals detected at each detector.

The function chat.nk computes an erroneous estimate of the overdispersion of activity centres (Bischof et al. 2020; Efford and Fletcher 2025).

`Enk` uses exact formulae for 'multi', 'proximity' and 'count' detector types. Other types may be simulated by setting a positive value for 'nrepl', which should be large (e.g., nrepl = 10000).

Usage

```
nk(capthist)

Enk(D, mask, traps, detectfn = 0, detectpar = list(g0 = 0.2,
    sigma = 25, z = 1), noccasions = NULL, binomN = NULL,
    userdist = NULL, ncores = NULL, nrepl = NULL)

chat.nk(object, nsim = NULL, ...)
```

Arguments

`capthist`	secr `capthist` object
`D`	numeric density, either scalar or vector of length nrow(mask)
`mask`	single-session habitat mask
`traps`	`traps` object
`detectfn`	integer code for detection function q.v.
`detectpar`	a named list giving a value for each parameter of detection function

noccasions	number of sampling intervals (occasions)
binomN	integer code for discrete distribution (see secr.fit)
userdist	user-defined distance function or matrix (see userdist)
ncores	integer number of threads
nrepl	integer number of replicates for E(nk) by simulation (optional)
object	fitted secr model or dataframe (see Warnings for restrictions)
nsim	integer number of c-hat values to simulate (optional)
...	other arguments passed to Fletcher.chat (verbose, type)

Details

If traps has a usage attribute then noccasions is set accordingly; otherwise it must be provided.

The environment variable RCPP_PARALLEL_NUM_THREADS determines the number of parallel threads. It is set to the value of ncores, unless that is NULL (see setNumThreads).

chat.nk may be used to simulate $\hat{c}$ values under the given model (set nsim > 0). The ... argument may include 'ncores = x' (x>1) to specify parallel processing of simulations - the speed up is large on unix-like machines for which the cluster type of makeCluster is "FORK" rather than "PSOCK". If 'ncores' is not provided then the value returned by setNumThreads() is used.

Value

For nk, a vector of observed counts, one for each detector in traps(capthist).

For Enk, a vector of expected counts, one for each detector in traps.

For chat.nk, usually a list comprising –

expected.nk	expected number at each detector
nk	observed number at each detector
stats	vector of summary statistics: mean(expected.nk), var(expected.nk), mean(nk), var(nk), nu (=df), X2/nu
chat	$\hat{c}$ (Fletcher or Wedderburn depending on 'type')

There are two variations –

If 'verbose = FALSE' then only the numeric value of $\hat{c}$ is returned (a vector of 2 values if 'type = "both"').

If chat.nk is called with 'nsim > 0' then the output is a list comprising –

type	from input
nsim	from input
sim.chat	vector of simulated $\hat{c}$
chat	$\hat{c}$ (Fletcher or Wedderburn depending on 'type')
p	probability of observing $\hat{c}$ as large as this (from rank of chat among sim.chat)

Warning

The variance inflation factor given by `chat.nk` was shown by Efford and Fletcher (2025) to be inadequate and should not be used. For replicate spatial samples, `chat.nj` is a better alternative.

These functions are experimental and do not work with polygon-like and single-catch detectors. No allowance is made for modelled variation in detection parameters with respect to occasion, detector or animal; this includes mixture models (e.g., g0~h2).

Versions before 4.5.11 did not correctly compute expected counts for multi-catch detectors.

References

Bischof, R., P. Dupont, C. Milleret, J. Chipperfield, and J. A. Royle. 2020. Consequences of ignoring group association in spatial capture–recapture analysis. *Wildlife Biology* wlb.00649. doi:10.2981/wlb.00649

Efford, M. G. and D. Fletcher. 2025. Effect of spatial overdispersion on confidence intervals for population density estimated by spatial capture-recapture. *bioRxiv* https://doi.org/10.1101/2024.03.12.584742

Fletcher, D. (2012) Estimating overdispersion when fitting a generalized linear model to sparse data. *Biometrika* **99**, 230–237.

Wedderburn, R. W. M. (1974) Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika* **61**, 439–47.

See Also

`secr`, `make.mask`, `Detection functions`, `Fletcher.chat`, `chat.nj`

Examples

```
  temptrap <- make.grid()
  msk <- make.mask(temptrap)
  ## expected number of individuals per detector (multi-catch)
  Enk (D = 5, msk, temptrap, detectpar = list(g0 = 0.2, sigma = 25),
    noccasions = 5)
```

```
# useful plotting function for simulated chat (nsim>0)
plotchat <- function(chat, head = '', breaks = seq(0.5,2.5,0.05)) {
    hist(chat$sim.chat, xlim = range(breaks), main = head, xlab = 'c-hat',
        breaks = breaks, cex.main = 1, yaxs = 'i')
    abline(v = chat$chat, lwd = 1.5, col = 'blue')
}
```

nontarget *Non-target Data*

Description

Non-target detections and interference events may be recorded in a binary detector x occasion matrix attached as an attribute to a single-session capthist object, or to each component of a multi-session capthist object.

Models fitted by **secr** make no use of these data. They may be used in **ipsecr**.

From **secr** 4.5.6, a summary of nontarget data is reported by the `summary` method for capthist objects, and the `verify` method reports clashes between detections and nontarget data.

Extraction and replacement funcions are provided from **secr** 4.5.7 on.

Usage

```
nontarget(object, ...)
nontarget(object) <- value
```

Arguments

`object`	capthist object
`...`	other argments (not used)
`value`	numeric binary matrix (rows = detectors, columns = occasions)

Details

The order of rows should match the order of detectors in `traps(object)`. Matrix entries should be zero for trap x occasion combinations that were not used (see usage) or for which there is a corresponding detection.

`value` is coerced to a matrix before assignment.

Value

For `nontarget()`, a matrix or list of matrices.

See Also

`plot.capthist`, `summary.capthist`

Examples

```
set.seed(123)
ch <- captdata

# traps that caught something
caught <- t(apply(ch, 2:3, sum))

# construct artificial nontarget data
# (positive for half the traps that caught nothing)
nontarget(ch) <- (1-caught) * (runif(500)>0.5)

head(caught)
head(nontarget(ch))

# the summary method recognises the 'nontarget' attribute
summary(ch)$nontarget
```

---

occasionKey *Key to Petal Plot*

Description

Displays a graphic key to the occasions corresponding to petals in a petal plot.

Usage

```
occasionKey(capthist, noccasions, rad = 3, x, y, px = 0.9, py = 0.9,
                          title = 'Occasion', ...)
```

Arguments

capthist	single-session capthist object
noccasions	number of petals (if capthist not provided)
rad	distance of petal centre from key centre
x	numeric x coordinate for centre of key
y	numeric y coordinate for centre of key
px	x position as fraction of user coordinates
py	y position as fraction of user coordinates
title	character
...	other arguments passed to text

Details

Either capthist or noccasions is required. It is assumed that a plot exists.

Graphic arguments in ... are applied to both the title and the occasion numbers.

Value

The key will be added to an existing plot. No value is returned.

See Also

plot.capthist

Examples

```
plot(captdata, border = 50)
occasionKey(captdata, rad = 8, cex = 0.8)
```

---

ovenbird *Ovenbird Mist-netting Dataset*

Description

Data from a multi-year mist-netting study of ovenbirds (*Seiurus aurocapilla*) at a site in Maryland, USA.

Usage

```
ovenCH
ovenCHp
ovenbird.model.1
ovenbird.model.D
ovenbird.model.Dl
ovenmask
```

Details

From 2005 to 2009 D. K. Dawson and M. G. Efford conducted a capture–recapture survey of breeding birds in deciduous forest at the Patuxent Research Refuge near Laurel, Maryland, USA. The forest was described by Stamm, Davis & Robbins (1960), and has changed little since. Analyses of data from previous mist-netting at the site by Chan Robbins were described in Efford, Dawson & Robbins (2004) and Borchers & Efford (2008).

Forty-four mist nets (12 m long, 30-mm mesh) spaced 30 m apart on the perimeter of a 600-m x 100-m rectangle were operated for approximately 9 hours on each of 9 or 10 non-consecutive days during late May and June in each year. Netting was passive (i.e. song playback was not used to lure birds into the nets). Birds received individually numbered bands, and both newly banded and previously banded birds were released at the net where captured. Sex was determined in the hand from the presence of a brood patch (females) or cloacal protuberance (males). A small amount of extra netting was done by other researchers after the main session in some years.

This dataset comprises all records of adult (after-hatch-year) ovenbirds caught during the main session in each of the five years 2005–2009. One ovenbird was killed by a predator in the net in

2009, as indicated by a negative net number in the dataset. Sex was determined in the hand from the presence of a brood patch (females) or cloacal protuberance (males). Birds are listed by their band number (4-digit prefix, '.', and 5-digit number).

The data are provided as a multi-session `capthist` object 'ovenCHp'. Sex is coded as a categorical individual covariate ("M" or "F").

Recaptures at the same site within a day are not included in this dataset, so `ovenCHp` has detector type 'proximity'. Previous versions of **secr** provided only a trimmed version of these data, retaining only one capture per bird per day (`ovenCH` with detector type 'multi'). That may be obtained from `ovenCHp` as shown in the examples.

Although several individuals were captured in more than one year, no use is made of this information in the analyses presently offered in **secr**.

An analysis of the data for males in the first four years showed that they tended to avoid nets after their first capture within a season (Dawson & Efford 2009). While the species was present consistently, the number of detections in any one year was too small to give reliable estimates of density; pooling of detection parameters across years helped to improve precision.

Included with the data are a mask and three models fitted to `ovenCH` as in Examples.

Object	Description
ovenCH	multi-session capthist object (as multi-catch)
ovenCHp	multi-session capthist object (as binary proximity)
ovenbird.model.1	fitted secr model – null
ovenbird.model.D	fitted secr model – trend in density across years
ovenbird.model.Dl	fitted secr model – trend in density across years, Dlambda parameterization
ovenmask	mask object

Source

D. K. Dawson (<ddawson@usgs.gov>) and M. G. Efford unpublished data.

References

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

Dawson, D. K. and Efford, M. G. (2009) Bird population density estimated from acoustic signals. *Journal of Applied Ecology* **46**, 1201–1209.

Efford, M. G., Dawson, D. K. and Robbins C. S. (2004) DENSITY: software for analysing capture-recapture data from passive detector arrays. *Animal Biodiversity and Conservation* **27**, 217–228.

Stamm, D. D., Davis, D. E. and Robbins, C. S. (1960) A method of studying wild bird populations by mist-netting and banding. *Bird-Banding* **31**, 115–130.

See Also

`capthist`

Examples

```
## commands used to create ovenCH from the input files
## "netsites0509.txt" and "ovencapt.txt"
## for information only - these files not distributed

# netsites0509 <- read.traps(file = "netsites0509.txt",
#    skip = 1, detector = "proximity")
# temp <- read.table("ovencapt.txt", colClasses=c("character",
#     "character", "numeric", "numeric", "character"))
# ovenCHp <- make.capthist(temp, netsites0509, covnames = "Sex")
# ovenCHp <- reduce(ovenCHp, dropunused = FALSE) # drop repeat detections

par(mfrow = c(1,5), mar = c(1,1,4,1))
plot(ovenCHp, tracks = TRUE, varycol = TRUE)
par(mfrow = c(1,1), mar = c(5,4,4,2) + 0.1)  ## defaults

counts(ovenCHp, "n")

## Not run:

## trimmed version of data - for consistency with earlier versions

ovenCH <- reduce(ovenCHp, outputdetector = "multi", dropunused = FALSE)

## array constant over years, so build mask only once
## "pdot" type is no longer favoured, but kept here for consistency
ovenmask <- make.mask(traps(ovenCH)[["2005"]], type = "pdot",
    buffer = 400, spacing = 15, detectpar = list(g0 = 0.03,
    sigma = 90), nocc = 10)

## fit constant-density model
ovenbird.model.1 <- secr.fit(ovenCH, mask = ovenmask)

## fit temporal trend in density (Session capitalized)
ovenbird.model.D <- secr.fit(ovenCH, mask = ovenmask,
    model = list(D ~ Session))

## fit temporal trend in density, Dlambda parameterization (Efford 2025 Appendix J)
ovenbird.model.Dl <- secr.fit(ovenCH, mask = ovenmask,
    model = list(D ~ 1), details = list(Dlambda = TRUE))
predictDlambda(ovenbird.model.Dl)

## compare compatible pre-fitted models
AIC(ovenbird.model.1, ovenbird.model.D)


## End(Not run)
```

ovensong *Ovenbird Acoustic Dataset*

Description

Data from an acoustic survey of ovenbirds (*Seiurus aurocapilla*) at a site in Maryland, USA.

Usage

```
signalCH
ovensong.model.1
ovensong.model.2
```

Details

In June 2007 D. K. Dawson and M. G. Efford used a moving 4-microphone array to survey breeding birds in deciduous forest at the Patuxent Research Refuge near Laurel, Maryland, USA. The data for ovenbirds were used to demonstrate a new method for analysing acoustic data (Dawson and Efford 2009). See `ovenbird` for mist-netting data from the same site over 2005–2009, and for other background.

Over five days, four microphones were placed in a square (21-m side) centred at each of 75 points in a rectangular grid (spacing 50 m); on each day points 100 m apart were sampled sequentially. Recordings of 5 minutes duration were made in .wav format on a 4-channel digital sound recorder.

The data are estimates of average power on each channel (microphone) for the first song of each ovenbird distinguishable in a particular 5-minute recording. Power was estimated with the sound analysis software Raven Pro 1.4 (Charif et al. 2008), using a window of 0.7 s duration and frequencies between 4200 and 5200 Hz, placed manually at the approximate centre of each ovenbird song. Sometimes this frequency range was obscured by insect noise so an alternative 1000-Hz range was measured and the values were adjusted by regression.

The data are provided as a single-session, single-occasion `capthist` object `signalCH`. The 'signal' attribute contains the power measurement in decibels for each detected sound on each channel where the power threshold is exceeded. As the threshold signal (attribute `cutval = 35`) is less than any signal value in this dataset, all detection histories are complete (1,1,1,1) across microphones. For analysis Dawson and Efford applied a higher threshold that treated weaker signals as 'not detected' (see Examples).

The row names of `signalCH` (e.g. "3755AX") are formed from a 4-digit number indicating the sampling location (one of 75 points on a 50-m grid) and a letter A–D to distinguish individual ovenbirds within a 5-minute recording; 'X' indicates power values adjusted by regression.

The default model for sound attenuation is a log-linear decline with distance from the source (linear decline on dB scale). Including a spherical spreading term in the sound attenuation model causes the likelihood surface to become multimodal in this case. Newton-Raphson, the default maximization method in `secr.fit`, is particularly inclined to settle on a local maximum; in the example below we use a set of starting values that have been found by trial and error to yield the global maximum.

Two fitted models are included (see Examples for details).

Object	Description
signalCH	capthist object
ovensong.model.1	fitted secr model – spherical spreading
ovensong.model.2	fitted secr model – no spherical spreading

Source

D. K. Dawson (<ddawson@usgs.gov>) and M. G. Efford unpublished data.

References

Charif, R. A., Waack, A. M. and Strickman, L. M. (2008) Raven Pro 1.3 User's Manual. Cornell Laboratory of Ornithology, Ithaca, New York.

Dawson, D. K. and Efford, M. G. (2009) Bird population density estimated from acoustic signals. *Journal of Applied Ecology* **46**, 1201–1209.

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.

See Also

capthist, ovenbird, Detection functions

Examples

```
summary(signalCH)
traps(signalCH)
signal(signalCH)

## apply signal threshold
signalCH.525 <- subset(signalCH, cutval = 52.5)

## Not run:

## models with and without spherical spreading
omask <- make.mask(traps(signalCH), buffer = 200)
ostart <- c(log(20), 80, log(0.1), log(2))
ovensong.model.1 <- secr.fit( signalCH.525, mask = omask,
    start = ostart, detectfn = 11 )
ovensong.model.2 <- secr.fit( signalCH.525, mask = omask,
    start = ostart, detectfn = 10 )


## End(Not run)

## compare fit of models
AIC(ovensong.model.1, ovensong.model.2)

## density estimates, dividing by 75 to allow for replication
collate(ovensong.model.1, ovensong.model.2)[1,,,"D"]/75
```

```
## plot attenuation curves cf Dawson & Efford (2009) Fig 5
pars1 <- predict(ovensong.model.1)[c("beta0", "beta1"), "estimate"]
pars2 <- predict(ovensong.model.2)[c("beta0", "beta1"), "estimate"]
attenuationplot(pars1, xval=0:150, spherical = TRUE, ylim = c(40,110))
attenuationplot(pars2, xval=0:150, spherical = FALSE, add = TRUE,
    col = "red")
## spherical spreading only
pars1[2] <- 0
attenuationplot(pars1, xval=0:150, spherical = TRUE, add = TRUE, lty=2)
```

`OVpossum` *Orongorongo Valley Brushtail Possums*

Description

A dataset from long-term capture-recapture trapping of brushtail possums *Trichosurus vulpecula* in New Zealand.

Usage

```
OVpossumCH
```

Format

A multi-session capthist object of 6 sessions. Sessions are labeled 49–54, corresponding to February 1996, June 1996, September 1996, February 1997, June 1997 and September 1997.

Details

Brushtail possums are 2-4 kg largely arboreal marsupials that have become pests of forests and farmland in New Zealand since their introduction from Australia in the nineteenth century. Their population dynamics in mixed native forest have been studied by capture-recapture in the Orongorongo Valley near Wellington since 1966 (e.g. Crawley 1973, Efford 1998, Efford and Cowan 2004).

From 1996 to 2006, a grid of 167 traps set on the ground at 30-m spacing was operated in an area of about 14 ha for 5 consecutive days three times each year (Efford and Cowan 2004). Each trap could catch only one animal, with rare exceptions when a young 'backrider' entered the trap with its mother. All animals were tagged and tattooed for individual identification and released at the site of capture.

A broad shingle riverbed forms a natural boundary on two sides of the study grid. Much of the grid lies on a gently sloping old alluvial fan and recent terraces, but to the southeast the valley side rises steeply and, except where cut by streams, supports beech forest (*Nothofagus truncata* and *Nothofagus solandri solandri*) rather than the mixed broadleaf forest of the valley floor.

This dataset relates to six five-day trapping sessions in 1996 and 1997, a time of high and declining density. Possums are long-lived (up to about 15 years) and as adults restrict their movements to a

home range of 1-10 ha. Breeding is seasonal, resulting in an influx of newly independent juveniles in the first trapping of each calendar year.

The dataset includes individual covariates not provided by Efford (2012): 'Sex' (F or M) and 'Age-class' (1 for first year, 2 for older).

A coarse habitat map is provided for the immediate vicinity of the trapping grid as the shapefile 'OVforest.shp' in the package 'extdata' folder. This distinguishes two forest classes ('beech' and 'non-beech'), and leaves out the shingle riverbed. The distinction between 'beech' and 'non-beech' is mapped only to a distance of about 120 m from the outermost traps. A text file 'leftbank.txt' in the same folder contains the x- and y- coordinates of the adjoining bank of the Orongorongo River. All coordinates relate to the old New Zealand Map Grid (NZMG), since replaced by the New Zealand Transverse Mercator grid (NZTM2000).

The example code shows how to import the shapefile as a **sf** 'simple features' object and use it to construct a mask for `secr.fit`.

Source

Efford (2012) and unpublished data.

References

Crawley, M. C. (1973) A live-trapping study of Australian brush-tailed possums, *Trichosurus vulpecula* (Kerr), in the Orongorongo Valley, Wellington, New Zealand. *Australian Journal of Zoology* **21**, 75–90.

Efford, M. G. (1998) Demographic consequences of sex-biased dispersal in a population of brushtail possums. *Journal of Animal Ecology* **67**, 503–517.

Efford, M. G. (2012) DENSITY 5.0: software for spatially explicit capture-recapture. Department of Mathematics and Statistics, University of Otago, Dunedin, New Zealand. https://www.otago.ac.nz/density/

Efford, M. G. and Cowan, P. E. (2004) Long-term population trend of *Trichosurus vulpecula* in the Orongorongo Valley, New Zealand. In: *The Biology of Australian Possums and Gliders*. Edited by R. L. Goldingay and S. M. Jackson. Surrey Beatty & Sons, Chipping Norton. Pp. 471–483.

Ward, G. D. (1978) Habitat use and home range of radio-tagged opossums *Trichosurus vulpecula* (Kerr) in New Zealand lowland forest. In: *The ecology of arboreal folivores*. Edited by G. G. Montgomery. Smithsonian Institute Press. Washington, D.C. Pp. 267–287.

Examples

```
## Not run:

library(sf)

summary(OVpossumCH, terse = TRUE)
ovtrap <- traps(OVpossumCH[[1]])

## retrieve and plot the forest map
OVforest <- st_read(system.file("extdata/OVforest.shp", package = "secr"))
## omit forest across the river by selecting only 1,2
OVforest <- OVforest[1:2,]
```

```
forestcol <- terrain.colors(6)[c(4,2)]
plot(st_as_sfc(OVforest), col = forestcol)
plot(ovtrap, add = TRUE)

## construct a mask
ovmask <- make.mask(ovtrap, buffer = 120, type = 'trapbuffer',
    poly = OVforest, spacing = 7.5, keep.poly = FALSE)
ovmask <- addCovariates(ovmask, OVforest)

## display mask
par(mar = c(0,0,0,8))
plot(ovmask, covariate = 'forest', dots = FALSE, col = forestcol)
plot(ovtrap, add = TRUE)

## add the left bank of the Orongorongo River
lines(read.table(system.file("extdata/leftbank.txt", package = "secr")))


## End(Not run)
```

Parallel *Multi-core Processing*

Description

From version 4.0 **secr** uses multi-threading in C++ (package **RcppParallel**, Allaire et al. 2021) to speed likelihood evaluation and hence model fitting in `secr.fit`. Detection histories are distributed over threads. Setting `ncores = NULL` in functions with multi-threading uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

These functions use multi-threading and call `setNumThreads` internally:

- `autoini`
- `confint.secr`
- `derived.secr`
- `esa.secr`
- `fxi` and related functions
- `pdot`
- `region.N`
- `score.test`
- `secr.fit`

These functions use multi-threading without calling `setNumThreads`:

- `LLsurface.secr`
- `mask.check`

expected.n
secr.test
sim.secr

Other functions may use multithreading indirectly through a call to one of these functions. Examples are suggest.buffer (autoini), esaPlot (pdot), and bias.D (pdot).

NOTE: The mechanism for setting the number of threads changed between versions 4.1.0 and 4.2.0. The default number of cores is now capped at 2 to meet CRAN requirements. Setting ncores = NULL previously specified one less than the maximum number of cores.

Earlier versions of **secr** made more limited use of multiple cores (CPUs) through the package **parallel**. The functions par.secr.fit, par.derived, and par.region.N are now deprecated because they were too slow. list.secr.fit replaces par.secr.fit

'Unit' refers to the unit of work sent to each worker process. As a guide, a 'large' benefit means >60% reduction in process time with 4 CPUs.

parallel offers several different mechanisms, bringing together the functionality of **multicore** and **snow**. The mechanism used by **secr** is the simplest available, and is expected to work across all operating systems. Technically, it relies on Rscript and communication between the master and worker processes is *via* sockets. As stated in the **parallel** documentation "Users of Windows and Mac OS X may expect pop-up dialog boxes from the firewall asking if an R process should accept incoming connections". You may possibly get warnings from R about closing unused connections. These can safely be ignored.

Use parallel::detectCores() to get an idea of how many cores are available on your machine; this may (in Windows) include virtual cores over and above the number of physical cores. See RShowDoc("parallel", package = "parallel") in core R for explanation.

In secr.fit the output component 'proctime' misrepresents the elapsed processing time when multiple cores are used.

Warning

It appears that multicore operations in **secr** using **parallel** may fail if the packages **snow** and **snowfall** are also loaded. The error message is obscure:

"Error in UseMethod("sendData") : no applicable method for 'sendData' applied to an object of class "SOCK0node""

References

Allaire, J. J., Francois, R., Ushey, K., Vandenbrouck, G., Geelnard, M. and Intel (2021) RcppParallel: Parallel Programming Tools for 'Rcpp'. R package version 5.1.2. https://CRAN.R-project.org/package=RcppParallel.

Examples

```
## Not run:

sessionInfo()
# R version 4.3.0 (2023-04-21 ucrt)
```

```
# Platform: x86_64-w64-mingw32/x64 (64-bit)
# Running under: Windows 11 x64 (build 22621)
# on Dell-XPS 8950 Intel i7-12700K
# ...
# see stackoverflow suggestion for microbenchmark list argument
# https://stackoverflow.com/questions/32950881/how-to-use-list-argument-in-microbenchmark

library(microbenchmark)
options(digits = 4)

## benefit from multi-threading in secr.fit

jobs <- lapply(seq(2,8,2), function(nc)
    bquote(suppressWarnings(secr.fit(captdata, trace = FALSE, ncores = .(nc)))))
microbenchmark(list = jobs, times = 10, unit = "seconds")
# [edited output]
# Unit: seconds
# expr     min      lq   mean median     uq    max neval
# ncores = 2 1.75880 2.27978 2.6680 2.7450 3.0960 3.4334    10
# ncores = 4 1.13549 1.16280 1.6120 1.4431 2.0041 2.4018    10
# ncores = 6 0.88003 0.98215 1.2333 1.1387 1.5175 1.6966    10
# ncores = 8 0.78338 0.90033 1.5001 1.0406 1.2319 4.0669    10

## sometimes (surprising) lack of benefit with ncores>2

msk <- make.mask(traps(ovenCH[[1]]), buffer = 300, nx = 25)
jobs <- lapply(c(1,2,4,8), function(nc)
    bquote(secr.fit(ovenCH, trace = FALSE, ncores = .(nc), mask = msk)))
microbenchmark(list = jobs, times = 10, unit = "seconds")
# [edited output]
# Unit: seconds
# expr     min      lq   mean median     uq    max neval
# ncores = 1 12.5010 13.4951 15.674 15.304 16.373 21.723    10
# ncores = 2 10.0363 11.8634 14.407 13.726 16.782 22.966    10
# ncores = 4  8.6335 10.3422 13.085 12.449 15.729 17.914    10
# ncores = 8  8.5286  9.9008 10.751 10.736 10.796 14.885    10

## and for simulation...

jobs <- lapply(seq(2,8,2), function(nc)
    bquote(sim.secr(secrdemo.0, nsim = 20, tracelevel = 0,  ncores = .(nc))))
microbenchmark(list = jobs, times = 10, unit = "seconds")
# [edited output]
# Unit: seconds
# expr    min     lq   mean median     uq     max neval
# ncores = 2 48.610 49.932 59.032 52.485 54.730 119.905    10
# ncores = 4 29.480 29.996 30.524 30.471 31.418  31.612    10
# ncores = 6 22.583 23.594 24.148 24.354 24.644  25.388    10
# ncores = 8 19.924 20.651 25.581 21.002 21.696  51.920    10

## and log-likelihood surface

jobs <- lapply(seq(2,8,2), function(nc)
```

```
    bquote(suppressMessages(LLsurface(secrdemo.0,  ncores = .(nc)))))
microbenchmark(list = jobs, times = 10, unit = "seconds")
# [edited output]
# Unit: seconds
# expr    min     lq   mean median     uq    max neval
# ncores = 2 20.941 21.098 21.290 21.349 21.471 21.619    10
# ncores = 4 14.982 15.125 15.303 15.263 15.449 15.689    10
# ncores = 6 13.994 14.299 14.529 14.342 14.458 16.515    10
# ncores = 8 13.597 13.805 13.955 13.921 14.128 14.353    10


## End(Not run)
```

---

pdot *Net Detection Probability*

Description

Compute spatially explicit net probability of detection for individual(s) at given coordinates (pdot).

Usage

```
pdot(X, traps, detectfn = 0, detectpar = list(g0 = 0.2,
    sigma = 25, z = 1), noccasions = NULL, binomN = NULL,
    userdist = NULL, ncores = NULL)

CVpdot(..., conditional = FALSE)
```

Arguments

X	vector or 2-column matrix of coordinates
traps	traps object
detectfn	integer code for detection function q.v.
detectpar	a named list giving a value for each parameter of detection function
noccasions	number of sampling intervals (occasions)
binomN	integer code for discrete distribution (see secr.fit)
userdist	user-defined distance function or matrix (see userdist)
ncores	integer number of threads
...	arguments passed to pdot
conditional	logical; if TRUE then computed mean and CV are conditional on detection

Details

If traps has a usage attribute then noccasions is set accordingly; otherwise it must be provided.

The probability computed is $p.(\mathbf{X}) = 1 - \prod_k \{1 - p_s(\mathbf{X}, k)\}^S$ where the product is over the detectors in traps, excluding any not used on a particular occasion. The per-occasion detection function p_s is halfnormal (0) by default, and is assumed not to vary over the S occasions.

From 4.6.11, the detection parameters g0, lambda0 and sigma for point detectors may be detector- and occasion-specific. This is achieved by providing a vector of values that is replicated internally to fill a matrix with dimensions ntraps x noccasions (i.e. in trap order for occasion 1, then occasion 2 etc.)

For detection functions (10) and (11) the signal threshold 'cutval' should be included in detectpar, e.g., detectpar = list(beta0 = 103, beta1 = -0.11, sdS = 2, cutval = 52.5).

The calculation is not valid for single-catch traps because $p.(\mathbf{X})$ is reduced by competition between animals.

userdist cannot be set if 'traps' is any of polygon, polygonX, transect or transectX. if userdist is a function requiring covariates or values of parameters 'D' or 'noneuc' then X must have a covariates attribute with the required columns.

Setting ncores = NULL uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see setNumThreads).

CVpdot returns the expected mean and CV of pdot across the points listed in X, assuming uniform population density. X is usually a habitat mask. See Notes for details.

Value

For pdot, a vector of probabilities, one for each row in X.

For CVpdot, a named vector with elements 'meanpdot' and 'CVpdot'.

Note

CVpdot computes the mean μ and variance V of the location-specific overall detection probability $p.(\mathbf{X})$ as follows.

$$\mu = \int p.(\mathbf{X}) f(\mathbf{X}) d\mathbf{X},$$

$$V = \int p.(\mathbf{X})^2 f(\mathbf{X}) d\mathbf{X} - \mu^2.$$

For uniform density and conditional = FALSE, $f(\mathbf{X})$ is merely a scaling factor independent of $\mathbf{X}$.

If conditional = TRUE then $f(\mathbf{X}) = p.(\mathbf{X}) / \int p.(\mathbf{X}) d\mathbf{X}$.

The coefficient of variation is CV = $\sqrt{V}/\mu$.

See Also

secr, make.mask, Detection functions, pdotContour, CV

Examples

```
## Not run:

  temptrap <- make.grid()
  ## per-session detection probability for an individual centred
  ## at a corner trap. By default, noccasions = 5.
  pdot (c(0,0), temptrap, detectpar = list(g0 = 0.2, sigma = 25),
    noccasions = 5)

  msk <- make.mask(temptrap, buffer = 100)
  CVpdot(msk, temptrap, detectpar = list(g0 = 0.2, sigma = 25),
    noccasions = 5)

## End(Not run)
```

PG *Telemetry Fixes in Polygons*

Description

For a telemetry dataset, either as a standalone capthist object with detector type 'telemetryonly' or the xylist attribute of a combined capthist object resulting from `addTelemetry`, determine the proportion of fixes of each individual that lie within a set of polygons. Typically used to obtain the proportion of fixes on a trapping grid, hence 'proportion on grid'.

Usage

```
PG(CH, poly = NULL, includeNULL = FALSE, plt = FALSE, ...)
```

Arguments

`CH`	capthist object including telemetry locations
`poly`	polygon object (see `boundarytoSF`)
`includeNULL`	logical; if TRUE then missing values are returned for animals without telemetry data
`plt`	logical; if TRUE then poly and telemetry locations are plotted
`...`	other arguments passed to `bufferContour`

Details

By default `poly` is obtained by applying `bufferContour` with arguments ... to the traps attribute of `CH`. Note that either a positive `buffer` argument or `convex = TRUE` is needed for the polygon to have area > 0.

If `plt = TRUE`, `bufferContour` is used to plot `poly` and the points are overplotted (open circles outside, filled circles inside). To control the framing of the plot, create an initial plot (e.g., with plot.traps, setting the `border` argument) and use `add = TRUE` (see Examples).

Value

Numeric vector of proportions. If `includeNULL = TRUE` length equal to number of animals (rows) in `CH`; otherwise length is the number of animals for which there is telemetry data (because xylist may cover only a subset of animals in `CH`).

References

Grant, T. J. and Doherty, P. F. (2007) Monitoring of the flat-tailed horned lizard with methods incorporating detection probability. *Journal of Wildlife Management* **71**, 1050–1056

See Also

`addTelemetry`, `bufferContour`, `pointsInPolygon`

Examples

```
## Not run:
olddir <- setwd('d:/density communication/combining telemetry and secr/possums')
CvilleCH <- read.capthist('CVILLE summer captures 4occ.txt',
                          'CVILLE detectors summer 4occ.txt',
                          detector = 'single')
CvilleGPS <- read.telemetry('CVILLE GPS Combined 4occ.txt')
CvilleGPSnew <- read.telemetry('CVILLE summer GPS New occasions.txt')
setwd(olddir)

CvilleBoth <- addTelemetry(CvilleCH, CvilleGPSnew)
plot(CvilleBoth, border = 400)
PG(CvilleBoth, buffer = 100, convex = TRUE, plt = TRUE, add = TRUE,
    col = 'red')

#################################################################
## this code computes an area-adjusted density estimate
## cf Grant and Doherty 2007
PGD <- function (CH, estimator = 'h2', ...) {
    pg <- PG(CH, ...)
    PGbar <- mean(pg)
    N <- closedN(CH, estimator)
    A <- polyarea(bufferContour(traps(CH), ...)[[1]])
    Dhat <- N$Nhat / A * PGbar
    varDhat <- (N$Nhat^2 * var(pg) + PGbar^2 * N$seNhat^2) / A^2
    c(Dhat = Dhat, seDhat = sqrt(varDhat))
}
plot(traps(CvilleBoth), border = 400)
PGD(CvilleBoth, buffer = 0, convex = TRUE, plt = TRUE, add = TRUE)
PGD(CvilleBoth, est='null', buffer = 0, convex = TRUE, plt = FALSE)

#################################################################
## this code generates a PG summary for telemetry records randomly
## translated and rotated, keeping the centres within a habitat mask

randomPG <- function(CH, poly = NULL, mask, reorient = TRUE, nrepl = 1,
                     seed = 12345, ...) {
```

```
    moveone <- function(xy, newcentre) {
        xy <- sweep(xy,2,apply(xy,2,mean))
        if (reorient)  ## random rotation about centre
            xy <- rotate(xy, runif(1)*360)
        sweep(xy,2,unlist(newcentre), "+")
    }
    onerepl <- function(r) {   ## r is dummy for replicate
        centres <- sim.popn(D = D, core = mask, model2D = "IHP",
                            Ndist = "fixed")
        xyl <- mapply(moveone, xyl, split(centres, rownames(centres)))
        attr(CH, 'xylist') <- xyl  ## substitute random placement
        PG(CH = CH , poly = poly, plt = FALSE, ...)
    }
    set.seed(seed)
    if (!requireNamespace('sf')) stop ("requires package sf")
    if (is.null(poly)) {
        poly <- bufferContour (traps(CH), ...)
        poly <- lapply(poly, as.matrix)
        poly <- sf::st_sfc(sf::st_polygon(poly))
    }
    xyl <- telemetryxy(CH)
    D <- length(xyl) / maskarea(mask)
    sapply(1:nrepl, onerepl)
}

mask <- make.mask (traps(CvilleBoth), buffer = 400, type = "trapbuffer")
pg <- randomPG (CvilleBoth, mask = mask, buffer = 100, convex = TRUE,
    nrepl = 20)
apply(pg, 1, mean)
##################################################################


## End(Not run)
```

`plot.capthist` *Plot Detection Histories*

Description

Display a plot of detection (capture) histories or telemetry data over a map of the detectors.

Usage

```
## S3 method for class 'capthist'
plot(x, rad = 5, hidetraps = FALSE, tracks = FALSE,
    title = TRUE, subtitle = TRUE, add = FALSE, varycol = TRUE,
   icolours = NULL, randcol = FALSE, lab1cap = FALSE, laboffset = 4, ncap = FALSE,
   splitocc = NULL, col2 = "green", type = c("petal", "n.per.detector", "n.per.cluster",
        "sightings", "centres", "telemetry", "nontarget"),
    cappar = list(cex = 1.3, pch = 16, col = "blue"),
```

```
    trkpar = list(col = "blue", lwd = 1),
    labpar = list(cex = 0.7, col = "black"), ...)

plotMCP(x, add = FALSE, col = "black", fill = NA, lab1cap = FALSE, laboffset = 4,
    ncap = FALSE, ...)
```

Arguments

x	an object of class capthist
rad	radial displacement of dot indicating each capture event from the detector location (used to separate overlapping points)
hidetraps	logical indicating whether trap locations should be displayed
tracks	logical indicating whether consecutive locations of individual animals should be joined by a line
title	logical or character string for title
subtitle	logical or character string for subtitle
add	logical for whether to add to existing plot
varycol	logical for whether to distinguish individuals by colour
icolours	vector of individual colours (when varycol = TRUE), or colour scale (non-petal plots)
randcol	logical to use random colours (varycol = TRUE)
lab1cap	logical for whether to label the first capture of each animal
laboffset	distance by which to offset labels from points
ncap	logical to display the number of detections per trap per occasion
splitocc	optional occasion from which second colour is to be used
col2	second colour (used with splitocc)
type	character string ("petal", "n.per.detector" or "n.per.cluster")
cappar	list of named graphical parameters for detections (passed to par)
trkpar	list of named graphical parameters for tracks (passed to par)
labpar	list of named graphical parameters for labels (passed to par)
...	arguments passed to plot.traps
col	vector of line colour numbers or names (plotMCP only)
fill	vector of fill colour numbers or names (plotMCP only)

Details

By default, a 'petal' plot is generated in the style of Density (Efford 2012) using eqscplot from the MASS library.

If type = "n.per.detector" or type = "n.per.cluster" the result is a colour-coded plot of the number of individuals at each unit, pooled over occasions.

If type = "sightings" the sightings of unmarked animals are displayed on a petal-like plot (requires mark-resight data) (see also sightingPlot).

If type = "centres" then a single point is plotted for each animal, jittered on each axis by a random amount (limits +/- rad/2).

If type = "telemetry" and the 'telemetryxy' attribute is not NULL then the telemetry locations are plotted.

If type = "nontarget" and the 'nontarget' attribute is not NULL then the nontarget captures or interference events are plotted.

If title = FALSE no title is displayed; if title = TRUE, the session identifer is used for the title.

If subtitle = FALSE no subtitle is displayed; if subtitle = TRUE, the subtitle gives the numbers of occasions, detections and individuals.

If x is a multi-session capthist object then a separate plot is produced for each session. Use par(mfrow = c(nr, nc)) to allow a grid of plots to be displayed simultaneously (nr rows x nc columns).

These arguments are used only for petal plots: rad, tracks, varycol, randcol, lab1cap, laboffset, ncap, splitocc, col2, trkpar, and labpar. Call occasionKey to add a key to the petals. From 5.0.1 a warning is issued if rad exceeds 3% of the detector span in either x- or y-dimensions.

If icolours = NULL and varycol = TRUE then a vector of colours is generated automatically as topo.colors((nrow(x)+1) * 1.5). If there are too few values in icolours for the number of individuals then colours will be re-used.

plotMCP plots minimum convex polygons of individual location data over a base plot of detector locations. Usually the data are telemetry locations in the xylist attribute of the capthist object; if this is not present and x is a polygon search capthist then the individual xy data are plotted.

Value

For type = "petal", the number of detections in x. For type = "sightings", the number of sightings of unmarked animals in x. For type = "n.per.detector" or type = "n.per.cluster", a dataframe with data for a legend (see Examples).

plotMCP invisibly returns a list in which each component is a 2-column (x,y) dataframe of boundary coordinates for one individual.

References

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

See Also

capthist, occasionKey, plot.traps, sightingPlot

Examples

```
demotrap <- make.grid()
tempcapt <- sim.capthist(demotrap,
```

```
    popn = list(D = 5, buffer = 50),
    detectpar = list(g0 = 0.15, sigma = 30))
plot(tempcapt, border = 10, rad = 3, tracks = TRUE,
    lab1cap = TRUE, laboffset = 2.5)

## type = n.per.cluster

## generate some captures
testregion <- data.frame(x = c(0,2000,2000,0),
    y = c(0,0,2000,2000))
popn <- sim.popn (D = 10, core = testregion, buffer = 0,
    model2D = "hills", details = list(hills = c(-2,3)))
t1 <- make.grid(nx = 1, ny = 1)
t1.100 <- make.systematic (cluster = t1, spacing = 100,
    region = testregion)
capt <- sim.capthist(t1.100, popn = popn, noccasions = 1)

## now plot captures ...
temp <- plot(capt, title = "Individuals per cluster",
    type = "n.per.cluster", hidetraps = FALSE,
    gridlines = FALSE, cappar = list(cex = 1.5))

if (interactive()) {
    ## add legend; click on map to place top left corner
    legend (locator(1), pch = 21, pt.bg = temp$colour,
        pt.cex = 1.3, legend = temp$legend, cex = 0.8)
}

## Not run:

## try varying individual colours - requires RColorBrewer
library(RColorBrewer)
plot(infraCH[[2]], icolours = brewer.pal(12, "Set3"), tracks = TRUE,
    bg = "black", cappar = list(cex = 2), border = 10, rad = 2,
    gridlines = FALSE)

## generate telemetry data
te <- make.telemetry()
tr <- make.grid(detector = "proximity")
totalpop <- sim.popn(tr, D = 20, buffer = 100)
tepop <- subset(totalpop, runif(nrow(totalpop)) < 0.05)
teCH <- sim.capthist(te, popn = tepop, renumber=FALSE, detectfn = "HHN",
    detectpar = list(lambda0 = 3, sigma = 25))
plot(teCH, type = 'telemetry', tracks = TRUE)

## simple "centres" example
## polygon data require 'hazard' detection function 14:19
CH <- sim.capthist(make.poly(), nocc = 20, detectfn = 'HHN',
    detectpar = list(lambda0 = 1, sigma = 10))
plot(CH, cappar = list(col = 'orange'), varycol = FALSE, border = 10)
plot(CH, type = 'centres', add = TRUE, rad = 0)

## End(Not run)
```

plot.mask *Plot Habitat Mask, Density or Resource Surface*

Description

Plot a habitat mask either as points or as an image plot. Colours maybe used to show the value of one mask covariate.

Usage

```
## S3 method for class 'mask'
plot(x, border = 20, add = FALSE, covariate = NULL, axes = FALSE,
    dots = TRUE, col = "grey", breaks = 10, meshcol = NA, ppoly = TRUE,
    polycol = "red", legend = TRUE, ...)

## S3 method for class 'Dsurface'
plot(x, covariate, group = NULL, plottype =
    "shaded", scale = 1, ...)

## S3 method for class 'Rsurface'
plot(x, covariate = "Resource", plottype =
    "shaded", scale = 1, ...)

spotHeight (object, prefix = NULL, dec = 2, point = FALSE, text = TRUE,
    sep = ", ",  session = 1, scale = 1, ...)
```

Arguments

x, object	mask or Dsurface object
border	width of blank display border (metres)
add	logical for adding mask points to an existing plot
covariate	name (as character string in quotes) or column number of a covariate to use for colouring
axes	logical for plotting axes
dots	logical for plotting mask points as dots, rather than as square pixels
col	colour(s) to use for plotting
breaks	an integer or a numeric vector – see cut
meshcol	colour for pixel borders (NA for none)
ppoly	logical for whether the bounding polygon should be plotted (if 'poly' specified)
polycol	colour for outline of polygon (ppoly = TRUE)
legend	logical; if TRUE a legend is plotted

...	other arguments passed to eqscplot (in the case of plot.mask), plot.mask (in the case of plot.Dsurface and plot.Rsurface), and points or text (in the case of spotHeight)
group	group for which plot required, if more than 1
plottype	character string c("dots", "shaded", "contour", "persp")
scale	numeric multiplier for density or other numeric covariate (see Dsurface)
prefix	character vector for name(s) of covariate(s) to retrieve
dec	number of decimal places for rounding density
point	logical for whether to plot point
text	logical for whether to place density label on plot
sep	character separator for elements if length(prefix)>1
session	session number or identifier

Details

The argument dots of plot.mask selects between two distinct types of plot (dots and shaded (coloured) pixels).

plot.Dsurface and plot.Rsurface offer contour and perspective plots in addition to the options in plot.mask. It may take some experimentation to get what you want - see contour and persp.

For plot.Dsurface the default value of 'covariate' is 'D' unless the Dsurface has a 'parameter' attribute of 'noneuc',

If using a covariate or Dsurface or Rsurface to colour dots or pixels, the col argument should be a colour vector of length equal to the number of levels (the default palette from 2.9.0 is terrain.colors, and this palette will also be used whenever there are too few levels in the palette provided; see Notes for more on palettes). Border lines around pixels are drawn in 'meshcol'. Set this to NA to eliminate pixel borders.

If a covariate is specified in a call to plot.Dsurface then that covariate will be plotted instead of density. This is a handy way to contour a covariate (contouring is not available in plot.mask).

If 'breaks' is an integer then the range of the covariate is divided into this number of equal intervals. Alternatively, 'breaks' may be a vector of break points (length one more than the number of intervals). This gives more control and often 'prettier'

spotHeight may be used to interrogate a plot produced with plot.Dsurface or plot.Rsurface, or by plot.mask if the mask has covariates. prefix defaults to 'density.' for Dsurface objects and to '' (all covariates) for mask objects. The predicted density or covariate at the nearest point is returned when the user clicks on the plot. Multiple values may be displayed (e.g., prefix = c("lcl","ucl") if Dsurface includes confidence limits). Click outside the mask or hit the Esc key to end. spotHeight deals with one session at a time.

Legend plotting is enabled only when a covariate is specified. It uses legend when dots = TRUE and strip.legend otherwise.

Value

If `covariate` is specified and `plottype = "shaded"` then `plot.mask` invisibly returns a character vector of the intervals defined by 'breaks' (useful for plotting a legend).

If `plottype = "persp"` then `plot.mask` invisibly returns a the perspective matrix that may be used to add to the plot with `trans3d`.

`spotHeight` invisibly returns a dataframe of the extracted values and their coordinates.

Note

`plot.mask()` acquired the argument 'legend' in version 2.9.0, and other changes (e.g., `breaks = 10`) may alter the output.

Contouring requires a rectangular grid; if a Dsurface is not rectangular then plot.Dsurface with `plottype = "contour"` triggers a call to `rectangularMask`.

The colour palettes `topo.colors`, `heat.colors` and `terrain.colors` may be viewed with the `demo.pal` function in the Examples code of their help page palettes.

The package **RColorBrewer** is a good source of palettes. Try `display.brewer.all()` and e.g., `col = brewer.pal(7, "YlGn")`.

See Also

`colours`, `mask`, `Dsurface`, `rectangularMask`, `contour` `persp` `strip.legend`

Examples

```
# simple

temptrap <- make.grid()
tempmask <- make.mask(temptrap)
plot (tempmask)

## Not run:

## restrict to points over an arbitrary detection threshold,
## add covariate, plot image and overlay traps

tempmask <- subset(tempmask, pdot(tempmask, temptrap,
    noccasions = 5)>0.001)
covariates (tempmask) <- data.frame(circle =
    exp(-(tempmask$x^2 + tempmask$y^2)/10000) )
plot (tempmask, covariate = "circle", dots = FALSE, axes = TRUE,
    add = TRUE, breaks = 8, col = terrain.colors(8), mesh = NA)
plot (temptrap, add = TRUE)

## add a legend
par(cex = 0.9)
covrange <- range(covariates(tempmask)$circle)
step <- diff(covrange)/8
colourlev <- terrain.colors(9)
zlev <- format(round(seq(covrange[1],covrange[2],step),2))
```

```
legend (x = "topright", fill = colourlev, legend = zlev,
    y.intersp = 0.8, title = "Covariate")

title("Colour mask points with p.(X) > 0.001")
mtext(side=3,line=-1, "g0 = 0.2, sigma = 20, nocc = 5")

## Waitarere possum density surface extrapolated across region

regionmask <- make.mask(traps(possumCH), buffer = 1000, spacing = 10,
    poly = possumremovalarea)
dts <- distancetotrap(regionmask, possumarea)
covariates(regionmask) <- data.frame(d.to.shore = dts)
shorePossums <- predictDsurface(possum.model.Ds, regionmask)

## plot as coloured pixels with white lines
colourlev <- terrain.colors(7)
plot(shorePossums, breaks = seq(0,3.5,0.5), plottype = "shaded",
    poly = FALSE, col = colourlev, mesh = NA)
plot(traps(possumCH), add = TRUE, detpar = list(col = "black"))
polygon(possumremovalarea)

## check some point densities
spotHeight(shorePossums, dec = 1, col = "black")

## add a legend
zlev <- format(seq(0,3,0.5), digits = 1)
legend (x = "topright", fill = colourlev, legend =
    paste(zlev,"--"), y.intersp = 1, title = "Density / ha")

## End(Not run)
```

---

plot.MCgof *Methods for MCgof Objects*

Description

Plot, summary and print methods for MCgof objects.

Usage

```
## S3 method for class 'MCgof'
plot(x, counts = 'all', overlay = NULL, maxT = NULL,
    main = NULL, cex = 0.9, ...)

## S3 method for class 'MCgof'
hist(x, counts = 'all', maxT = NULL, main = NULL,
    cex = 0.9, ...)
```

```
## S3 method for class 'MCgof'
summary(object, ...)

## S3 method for class 'MCgof'
print(x, ...)
```

Arguments

x	MCgof object
counts	character vector of marginal counts for which statistics are to be plotted
overlay	MCgof object
maxT	numeric maximum plotted value of statistic
main	character vector of labels (see Details)
cex	numeric size of labels and points
...	other arguments passed by the plot method to points for plotting overlay
object	MCgof object

Details

We start with a 3-D capthist array with dimensions corresponding to individuals (i), occasions (j) and detectors (k). The possible marginal counts for the default 'statfn' in MCgof are designated –

Count	Margin	Cell value
yik	individual x detector	$y_{ik} = \sum_j y_{ijk}$
yi	individual	$y_i = \sum_j \sum_k y_{ijk}$
yk	detector	$y_k = \sum_j \sum_i y_{ijk}$

The plot method displays a scatterplot of discrepancies for observed and simulated data (one point per replicate) (Gelman et al. 1996).

If 'overlay' is provided then the results are overlaid on the initial plot. Points should be distinguished by specifying a different colour (col) or symbol (pch) with the ... argument.

'main' is a vector of labels used as headers; the names should include all components of 'statfn'. Setting main = "" suppresses headers.

The hist method displays a histogram of the ratio Tobs/Tsim.

Value

The summary method returns a matrix of values in which the columns correspond to the different statistics (default yik, yi, yk) and the rows are

- median discrepancy Tobs
- median discrepancy Tsim
- proportion Tobs>Tsim
- number of valid results

References

Gelman, A., Meng, X.-L., and Stern, H. (1996) Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica* **6**, 733–807.

See Also

MCgof

`plot.popn` *Plot Population Object*

Description

Display animal locations from a popn object.

Usage

```
## S3 method for class 'popn'
plot(x, add = FALSE, frame = TRUE,
    circles = NULL, collapse = FALSE, seqcol = NULL, ...)
```

Arguments

x	object of class popn
add	logical to add points to an existing plot
frame	logical to add frame or polygon within which points were simulated
circles	vector giving the radii if circles are to be plotted
collapse	logical; if TRUE then multiple sessions are overlaid
seqcol	color used for first detection when collapse = TRUE (optional)
...	arguments passed to eqscplot and points or symbols

Details

If circles is provided then a circle of the given radius is plotted for each animal using the symbols function. The arguments fg and bg may be used to control the colour of the perimeter and the fill of each circle (see Examples).

For a multi-session popn with turnover, collapse = TRUE allows successive locations to be joined with (type = 'o' or type = 'l').

seqcol may be a single color, a vector of colours (one per session), or a vector of two colours, one for the first and one for all later sessions in which each animal was detected.

If ... includes 'col' then 'collapse' must be specified to avoid confusion, even for single-session data (see Examples).

See Also

popn, sim.popn

Examples

```
temppopn <- sim.popn(D = 5, expand.grid(
    x = c(0,100), y = c(0,100)))
# specify collapse to avoid partial match of col
plot(temppopn, pch = 16, collapse = FALSE, col = "blue")

plot(temppopn, circles = 20, bg = "tan", fg = "white")
plot(temppopn, pch = 16, cex = 0.5, add = TRUE)
```

---

plot.secr *Plot Detection Functions*

Description

Plot detection functions using estimates of parameters in an secr object, or as provided by the user.

Usage

```
## S3 method for class 'secr'
plot(x, newdata = NULL, add = FALSE,
    sigmatick = FALSE, rgr = FALSE, limits = FALSE, alpha = 0.05,
    xval = 0:200, ylim = NULL, xlab = NULL, ylab = NULL, ...)

## S3 method for class 'secrlist'
plot(x, newdata = NULL, add = FALSE,
    sigmatick = FALSE, rgr = FALSE, limits = FALSE, alpha = 0.05,
    xval = 0:200, ylim = NULL, xlab = NULL, ylab = NULL, ...,
    overlay = TRUE)

detectfnplot (detectfn, pars, details = NULL, add = FALSE,
    sigmatick = FALSE, rgr = FALSE, hazard = FALSE, xval = 0:200, ylim = NULL,
    xlab = NULL, ylab = NULL, ...)

attenuationplot (pars, add = FALSE, spherical = TRUE,
    xval = 0:200, ylim = NULL, xlab = NULL, ylab = NULL, ...)
```

Arguments

x	an secr object
newdata	dataframe of data to form estimates
add	logical to add curve(s) to an existing plot

sigmatick	logical; if TRUE the scale parameter sigma is shown by a vertical line
rgr	logical; if TRUE a scaled curve r.g(r) is plotted instead of g(r)
hazard	logical; if TRUE the hazard of detection is plotted instead of probability
limits	logical; if TRUE pointwise confidence limits are drawn
alpha	alpha level for confidence intervals
xval	vector of distances at for which detection to be plotted
ylim	vector length 2 giving limits of y axis
xlab	label for x axis
ylab	label for y axis
...	arguments to pass to lines
overlay	logical; if TRUE then automatically add = TRUE for plots after the first
detectfn	integer code or character string for shape of detection function 0 = halfnormal etc. – see detectfn
pars	list, vector or matrix of parameter values
details	list of ancillary parameters
spherical	logical for whether to include spherical spreading term

Details

newdata is usually NULL, in which case one curve is plotted for each session and group. Otherwise, predict.secr is used to form estimates and plot a curve for each row in newdata.

If axis labels are not provided they default to 'Distance (m)' and 'Detection probability' or 'Detection lambda'.

detectfnplot is an alternative in which the user nominates the type of function and provides parameter values. pars maybe a list as from detectpar; it is first coerced to a numeric vector with unlist. Parameter values must be in the expected order (e.g. g0, sigma, z). If pars is a matrix then a separate curve is plotted with the parameter values in each row.

For detectfnplot the signal threshold parameters 'cutval' and 'spherical' should be provided in details (see examples).

Approximate confidence limits for g(r) are calculated using a numerical first-order delta-method approximation to the standard error at each xval. The distribution of g(r) is assumed to be normal on the logit scale for non-hazard functions (detectfn 0:13). For hazard detection functions (detectfn 14:18) the hazard is assumed (from version 3.1.1) to be distributed normally on the log scale. Limits are back-transformed to the probability scale g(r).

attenuationplot plots the expected decline in signal strength with distance, given parameters β_0 and β_1 for a log-linear model of sound attenuation.

Value

plot.secr invisibly returns a dataframe of the plotted values (or a list of dataframes in the case that newdata has more than one row).

See Also

Detection functions, plot, secr

Examples

```
plot (secrdemo.b, xval = 0:100, ylim = c(0, 0.4))
## Add recapture probability
plot (secrdemo.b, newdata = data.frame(b = 1), add = TRUE,
    col = "red")

## signal strength detection: 70dB at source, attenuation
## 0.3dB/m, sdS 5dB; detection threshold 40 dB.
detectfnplot (detectfn = 10, c(70, -0.3, 5), details =
    list(cutval = 40))

## add a function with louder source and spherical spreading...
detectfnplot (detectfn = 11, c(110, -0.3, 5), details =
    list(cutval = 40), add = TRUE, col = "red")

## matching sound attenuation curves; `spherical-only' dashed line
attenuationplot (c(70, -0.3), spherical = FALSE, ylim=c(-10,110))
attenuationplot (c(110, 0), spherical = TRUE, add=TRUE, lty=2)
attenuationplot (c(110, -0.3), spherical = TRUE, add = TRUE,
    col = "red")
```

---

plot.traps *Plot traps Object*

Description

Map the locations of detectors (traps).

Usage

```
## S3 method for class 'traps'
plot(x, border = 100, label = FALSE, offset = c(6,6), add = FALSE,
  hidetr = FALSE, detpar = list(), txtpar = list(), bg = "white",
  gridlines = !add, gridspace = 100, gridcol = "grey",
  markused = FALSE, markvarying = FALSE, markvertices = FALSE,
  labelclusters = FALSE, frame = NULL, ...)
```

Arguments

x	a traps object
border	width of blank margin around the outermost detectors
label	logical indicating whether a text label should appear by each detector
offset	vector displacement of label from point on x and y axes

add	logical to add detectors to an existing plot
hidetr	logical to suppress plotting of detectors
detpar	list of named graphical parameters for detectors (passed to par)
txtpar	list of named graphical parameters for labels (passed to par)
bg	background colour
gridlines	logical for plotting grid lines
gridspace	spacing of gridlines
gridcol	colour of gridlines
markused	logical to distinguish detectors used on at least one occasion
markvarying	logical to distinguish detectors whose usage varies among occasions
markvertices	logical or 0,1,2 for plotting transect or polygon points
labelclusters	logical to label clusters
frame	data defining a boundary polygon (see boundarytoSF
...	arguments to pass to eqscplot

Details

offset may also be a scalar value for equal displacement on the x and y axes. The hidetr option is most likely to be used when plot.traps is called by plot.capthist. See par and colours for more information on setting graphical parameters. The initial values of graphical parameters are restored on exit.

Axes are not labeled. Use axis and mtext if necessary.

markvertices determines whether the vertices of each transect or polygon will be emphasised by overplotting a point symbol (detpar$pch). Value may be logical (TRUE, FALSE) or integer (0 = no points, 1 = terminal vertices only, 2 = all vertices).

From 4.4.0, polygon detectors are shaded with detpar$col and outlined (border) with detpar$fg. Use detpar$col = NA for no shading (transparent polygons).

labelclusters requires x to have attributes 'clusterID' and 'clustertrap'. The label is placed near the first detector in each cluster.

A boundary polygon is plotted in black if frame is not NULL.

Value

None

See Also

plot, traps, clusterID

Examples

```
temptrap <- make.grid()
plot (temptrap, detpar = list(pch = 16, col = "blue"),
   label = TRUE, offset = 7)
```

plotMaskEdge *Outline Around Mask Cells*

Description

Plots the outer edge of a mask.

Usage

```
plotMaskEdge(mask, plt = TRUE, add = FALSE, ...)
```

Arguments

mask	secr habitat mask object
plt	logical; if TRUE the edge is plotted
add	logical; if TRUE the line is added to an existing plot
...	other line plotting arguments passed to segments

Details

May be slow.

Value

A numeric matrix of 4 columns is returned invisibly. The columns may be used as arguments x0, y0, x1, y1 in a call to segments().

Note

A bug in **secr** <3.2.2 caused some internal lines to appear when the mask spacing was not an integer.

See Also

gridCells

Examples

```
## Not run:
plot(possummask)
plotMaskEdge (possummask, add = TRUE)

## End(Not run)
```

pmixProfileLL *Mixture Model Check*

Description

Compute the profile likelihood of a finite mixture model for a user-specified range of values for the mixing parameter. This provides a check on multimodality.

Usage

```
pmixProfileLL(CH, model = list(g0 ~ h2, sigma ~ h2), CL = TRUE, pmvals = seq(0.01,
 0.99, 0.01), pmi = 5, ...)
```

Arguments

CH	capthist object
model	model as in secr.fit
CL	logical as in in secr.fit
pmvals	numeric vector of values for mixing parameter 'pmix'
pmi	integer index of 'pmix' in vector of coefficients (beta parameters) for the specified model
...	other arguments passed to secr.fit

Details

See secr-finitemixtures.pdf.

Choosing the wrong value for pmi results in the error message "invalid fixed beta - require NP-vector". The easiest way to find the value of pmi is to inspect the output from a previously fitted mixture model - either count the coefficients or check fit$parindx$pmix (for a model named 'fit'). It is assumed that 'pmix' is the last real parameter in the model, and that pmix is constant.

Value

Numeric vector of profile likelihoods.

Note

This is slow to execute and the results are hard to interpret. Use only if you are confident.

Examples

```
## Not run:

pmvals <- seq(0.02,0.99,0.02)
mask <- make.mask(traps(ovenCH[[1]]), nx = 32, buffer = 100)

## only g0 ~ h2, so reduce pmi from 5 to 4
outPL <- pmixProfileLL(ovenCH[[1]], model = list(g0~h2),
    mask = mask, pmvals, CL = TRUE, trace = FALSE, pmi = 4)

plot(pmvals, outPL, xlim = c(0,1),
xlab = 'Fixed pmix', ylab = 'Profile log-likelihood')


## End(Not run)
```

`pointsInPolygon` *Points Inside Polygon*

Description

Determines which of a set of points lie inside a closed polygon or at least one of a set of polygons

Usage

```
pointsInPolygon(xy, poly, logical = TRUE)
```

Arguments

`xy`	2-column matrix or dataframe of x-y coordinates for points to assess
`poly`	2-column matrix or dataframe containing perimeter points of polygon, or a SpatialPolygonsDataFrame object from package **sp**, or a 'mask' object (see Warning)
`logical`	logical to control the output when 'poly' is a mask (see Details)

Details

If `poly` is a SpatialPolygonsDataFrame object then the method `over` is used from **sp**. This allows multiple polygons and polygons with holes.

If `poly` is an secr 'mask' object then `xy` is discretized and matched to the cells in `poly`. If `logical = FALSE` then the returned value is a vector of integer indices to the row in 'poly' corresponding to each row of 'xy'; otherwise the result is a vector of logical values.

Otherwise, the algorithm is adapted from some code posted on the S-news list by Peter Perkins (23/7/1996). The polygon should be closed (last point same as first).

Value

Vector of logical or integer values, one for each row in `xy`

Warning

If poly is a mask object then its cells must be aligned to the x- and y- axes

See Also

over

Examples

```
## 100 random points in unit square
xy <- matrix(runif(200), ncol = 2)
## triangle centred on (0.5, 0.5)
poly <- data.frame(x = c(0.2,0.5,0.8,0.2), y = c(0.2,0.8,0.2,0.2))
plot(xy, pch = 1 + pointsInPolygon(xy, poly))
lines(poly)
```

---

polyarea *Area of Polygon(s)*

Description

Area of a single closed polygon (simple x-y coordinate input) or of multiple polygons, possibly with holes.

Usage

```
polyarea(xy, ha = TRUE)
```

Arguments

xy	dataframe or list with components 'x' and 'y', or a SpatialPolygons or SpatialPolygonsDataFrame object from package **sp**, or an **sf** object with polygon data
ha	logical if TRUE output is converted from square metres to hectares

Details

For sf, sfc, SpatialPolygons or SpatialPolygonsDataFrame objects, the package **sf** is used.

Value

A scalar.

See Also

bufferContour

Examples

```
polyarea(make.grid(hollow = TRUE))
```

---

popn *Population Object*

Description

Encapsulate the locations of a set of individual animals.

Details

An object of class popn records the locations of a set of individuals, together with ancillary data such as their sex. Often used for a realisation of a spatial point process (e.g. homogeneous Poisson) with known density (intensity). Locations are stored in a data frame with columns 'x' and 'y'.

A popn object has attributes

covariates	data frame with numeric, factor or character variables to be used as individual covariates
model2D	2-D distribution ("poisson", "cluster", "IHP", "linear" etc.)
Ndist	distribution of number of individuals ("poisson", "fixed")
boundingbox	data frame of 4 rows, the vertices of the rectangular area

The number of rows in `covariates` must match the length of `x` and `y`. See `sim.popn` for more information on `Ndist` and `model2D`.

Note

The popn class is used only occasionally: it is not central to spatially explicit capture recapture.

See Also

`sim.popn`, `plot.popn`, `transformations`

possum *Brushtail Possum Trapping Dataset*

Description

Data from a trapping study of brushtail possums at Waitarere, North Island, New Zealand.

Usage

```
possumCH
possumarea
possumremovalarea
possummask
possum.model.0
possum.model.Ds
```

Details

Brushtail possums (*Trichosurus vulpecula*) are an unwanted invasive species in New Zealand. Although most abundant in forests, where they occasionally exceed densities of 15 / ha, possums live wherever there are palatable food plants and shelter.

Efford et al. (2005) reported a live-trapping study of possums in *Pinus radiata* plantation on coastal sand dunes. The 300-ha site at Waitarere in the North Island of New Zealand was a peninsula, bounded on one side by the sea and on two other sides by the Manawatu river. 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 centred at random points within the 300-ha area. Animals were tagged and released daily for 5 days in April 2002. Subsequently, leg-hold trapping was conducted on a trapping web centred on each square (data not reported here), and strenuous efforts were made to remove all possums by cyanide poisoning and further leghold trapping across the entire area. This yielded a density estimate of 2.26 possums / ha.

Traps could catch at most one animal per day. The live-trapped animals comprised 46 adult females, 33 adult males, 10 immature females and 11 immature males; sex and/or age were not recorded for 4 individuals (M. Coleman unpubl. data). These counts do not sum to the number of capture histories - see Note. One female possum was twice captured at two sites on one day, having entered a second trap after being released; one record in each pair was selected arbitrarily and discarded.

The data are provided as a single-session `capthist` object 'possumCH'. 'possummask' is a matching mask object - see Examples. Two fitted models are provided for illustration.

The dataframe `possumarea` contains boundary coordinates of a habitat polygon that is used to clip `possummask` at the shore (from secr 1.5). `possumarea` comprises a single polygon representing the extent of terrestrial vegetation to the west, north and east, and an arbitrary straight southern boundary. The boundary is also included as a shapefile and as a text file ('possumarea.shp' etc. and 'possumarea.txt' in the package 'extdata' folder). See Examples in `make.mask`.

The dataframe `possumremovalarea` contains boundary coordinates of another polygon, the nominal removal area of Efford et al. (2005 Fig. 1) (from secr 2.3).

Object	Description
possumCH	capthist object
possummask	mask object
possumarea	habitat perimeter
possumremovalarea	nominal boundary of removal region
possum.model.0	fitted secr model – null
possum.model.Ds	fitted secr model – distance to shore

Note

A significant problem with the data used by Efford et al. (2005) was noticed recently. Five capture histories in possumCH are for animals that had lost a previous tag. A further three histories may also have been animals that were tagged previously or mis-recorded. Analyses that treat each previously tagged animal as a new individual are in error (this includes the published analyses, the pre-fitted models described here, and those in Chapter 11 of The SECR book). All eight questionable histories are now indicated in possumCH with the logical covariate 'prev.tagged'.

Methods have not yet been developed to adjust for tag loss in SECR models.

Source

Landcare Research, New Zealand.

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., Dawson, D. K. and Robbins C. S. (2004) DENSITY: software for analysing capture-recapture data from passive detector arrays. *Animal Biodiversity and Conservation* **27**, 217–228.

Efford, M. G., Warburton, B., Coleman, M. C. and Barker, R. J. (2005) A field test of two methods for density estimation. *Wildlife Society Bulletin* **33**, 731–738.

See Also

capthist

Examples

```
plot(possummask)
plot(possumCH, tracks = TRUE, add = TRUE)
plot(traps(possumCH), add = TRUE)
lines(possumarea)
summary(possumCH)

## compare & average pre-fitted models
AIC(possum.model.0, possum.model.Ds)
modelAverage(possum.model.0, possum.model.Ds)

## Not run:
```

```
## Roughly estimate tag-loss error by dropping dubious histories
## i.e. restrict to "not previously tagged"
NPT <- !covariates(possumCH)$prev.tagged
possum.model.0.NPT <- secr.fit(subset(possumCH,NPT), mask =
    possummask, trace = FALSE)
predict(possum.model.0)[1,2]/ predict(possum.model.0.NPT)[1,2]
## ...about 9%

## End(Not run)
```

`predict.secr` *SECR Model Predictions*

Description

Evaluate a spatially explicit capture–recapture model. That is, compute the 'real' parameters corresponding to the 'beta' parameters of a fitted model for arbitrary levels of any variables in the linear predictor.

Usage

```
## S3 method for class 'secr'
predict(object, newdata = NULL, realnames = NULL, type = c("response", "link"),
    se.fit = TRUE, alpha = 0.05, savenew = FALSE, ...)

## S3 method for class 'secrlist'
predict(object, newdata = NULL, realnames = NULL, type = c("response", "link"),
    se.fit = TRUE, alpha = 0.05, savenew = FALSE, ...)

## S3 method for class 'secr'
detectpar(object, ..., byclass = FALSE, bytrap = FALSE)
```

Arguments

`object`	`secr` object output from `secr.fit`, or list of `secr` objects (secrlist)
`newdata`	optional dataframe of values at which to evaluate model
`realnames`	character vector of real parameter names
`type`	character; type of prediction required. The default ("response") provides estimates of the 'real' parameters.
`se.fit`	logical for whether output should include SE and confidence intervals
`alpha`	alpha level for confidence intervals
`savenew`	logical for whether newdata should be saved
`...`	other arguments passed to `makeNewData`

byclass	logical; if TRUE values are returned for each latent class in a mixture model, or class in a hybrid mixture (hcov) model
bytrap	logical; if TRUE values are returned for each detector

Details

The variables in the various linear predictors are described in secr-models.pdf and listed for the particular model in the vars component of object.

Optional newdata should be a dataframe with a column for each of the variables in the model (see 'vars' component of object). If newdata is missing then a dataframe is constructed automatically.

Default newdata are for a naive animal on the first occasion; numeric covariates are set to zero and factor covariates to their base (first) level. From secr 3.1.4 the argument 'all.levels' may be passed to makeNewData; if TRUE then the default newdata includes all factor levels.

realnames may be used to select a subset of parameters.

Standard errors for parameters on the response (real) scale are by the delta method (Lebreton et al. 1992), and confidence intervals are backtransformed from the link scale.

The value of newdata is optionally saved as an attribute.

detectpar is used to extract the detection parameter estimates from a simple model to pass to functions such as esaPlot. detectpar calls predict.secr. Parameters will be evaluated by default at base levels of the covariates, although this may be overcome by passing a one-line newdata to predict via the ... argument. Groups and mixtures are a headache for detectpar: it merely returns the estimated detection parameters of the first group or mixture.

If the 'a0' parameterization has been used in secr.fit (i.e., object$details$param == 3) then detectpar automatically backtransforms (a0, sigma) to (g0, sigma) or (lambda0, sigma) depending on the value of object$detectfn.

Value

When se.fit = FALSE, a dataframe identical to newdata except for the addition of one column for each 'real' parameter. Otherwise, a list with one component for each row in newdata. Each component is a dataframe with one row for each 'real' parameter (density, g0, sigma, b) and columns as below

link	link function
estimate	estimate of real parameter
SE.estimate	standard error of the estimate
lcl	lower 100(1–alpha)% confidence limit
ucl	upper 100(1–alpha)% confidence limit

When newdata has only one row, the structure of the list is 'dissolved' and the return value is one data frame.

For detectpar, a list with the estimated values of detection parameters (e.g., g0 and sigma if detectfn = "halfnormal"). In the case of multi-session data the result is a list of lists (one list per session). For models that include spatial variation in detection parameters (i.e. XXXXxy = sigmaxy, lambda0xy, a0xy or sigmakxy) the reported value is the product of XXXX and XXXXxy evaluated at the mask centre (coordinates in attr(fit$mask, 'meanSD')).

Note

predictDsurface should be used for predicting density at many points from a model with spatial variation. This deals automatically with scaling of x- and y-coordinates, and is much is faster than predict.secr. The resulting Dsurface object has its own plot method.

The argument 'scaled' was removed from both predict methods in version 2.10 as the scaleg0 and scalesigma features had been superceded by other parameterisations.

Overdispersion results in confidence intervals that are too narrow. See adjustVarD for a partial solution.

References

Lebreton, J.-D., Burnham, K. P., Clobert, J. and Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case studies. *Ecological Monographs* **62**, 67–118.

See Also

secr.fit, predictDsurface, adjustVarD, makeNewData

Examples

```
## load previously fitted secr model with trap response
## and extract estimates of `real' parameters for both
## naive (b = 0) and previously captured (b = 1) animals

predict (secrdemo.b, newdata = data.frame(b = 0:1))

## OR from secr 3.1.4
predict (secrdemo.b, all.levels = TRUE)

temp <- predict (secrdemo.b, all.levels = TRUE, save = TRUE)
attr(temp, "newdata")

detectpar(secrdemo.0)
```

---

predictDsurface *Predict Density Surface*

Description

Predict density at each point on a raster mask from an secr model fitted by maximising the full likelihood. derivedDsurface is the equivalent function for models fitted by maximising the conditional likelihood.

Usage

```
predictDsurface(object, mask = NULL, se.D = FALSE, cl.D = FALSE, alpha =
0.05, parameter = 'D')
```

Arguments

`object`	fitted secr object
`mask`	secr mask object
`se.D`	logical for whether to compute prediction SE
`cl.D`	logical for whether to compute confidence limits
`alpha`	alpha level for 100(1 – alpha)% confidence intervals
`parameter`	character for real parameter to predict

Details

Predictions use the linear model for density or other spatial parameters ('noneuc', 'sigmaxy' etc.) on the link scale in the fitted secr model 'object', or the fitted user-defined function, if that was specified in `secr.fit`.

If 'mask' is NULL then predictions are for the mask component of 'object'.

SE and confidence limits are computed only if specifically requested. They are not available for user-defined density functions.

Density is adjusted automatically for the number of clusters in 'mashed' models (see `mash`).

If `object` is a relative density model then `derivedDsurface` first computes the intercept.

Value

Object of class 'Dsurface' inheriting from 'mask'. Predicted densities are added to the covariate dataframe (attribute 'covariates') as column(s) with prefix 'D.' If the model uses multiple groups, multiple columns will be distinguished by the group name (e.g., "D.F" and "D.M"). If groups are not defined the column is named "D.0".

For multi-session models the value is a multi-session mask.

The pointwise prediction SE is saved as a covariate column prefixed 'SE.' (or multiple columns if multiple groups). Confidence limits are likewise saved with prefixes 'lcl.' and 'ucl.'.

See Also

`plot.Dsurface`, `secr.fit`, `predict.secr`, `derivedDsurface`

Examples

```
## use canned possum model
shorePossums <- predictDsurface(possum.model.Ds)
par(mar = c(1,1,1,6))
plot(shorePossums, plottype = "shaded", polycol = "blue", border = 100)
plot(traps(possumCH), detpar = list(col = "black"), add = TRUE)
par(mar = c(5,4,4,2) + 0.1)  ## reset to default
## extract and summarise
summary(covariates(shorePossums))

## Not run:
```

```
## extrapolate to a new mask; add covariate needed by model; plot
regionmask <- make.mask(traps(possumCH), buffer = 1000, spacing = 10,
    poly = possumremovalarea)
dts <- distancetotrap(regionmask, possumarea)
covariates(regionmask) <- data.frame(d.to.shore = dts)
regionPossums <- predictDsurface(possum.model.Ds, regionmask,
    se.D = TRUE, cl.D = TRUE)
par(mfrow = c(1,2), mar = c(1,1,1,6))
plot(regionPossums, plottype = "shaded", mesh = NA, breaks = 20)
plot(regionPossums, plottype = "contour", add = TRUE)
plot(regionPossums, covariate = "SE", plottype = "shaded",
    mesh = NA, breaks = 20)
plot(regionPossums, covariate = "SE", plottype = "contour",
    add = TRUE)

## confidence surfaces
plot(regionPossums, covariate = "lcl", breaks = seq(0,3,0.2),
    plottype = "shaded")
plot(regionPossums, covariate = "lcl", plottype = "contour",
    add = TRUE, levels = seq(0,2.7,0.2))
title("lower 95% surface")
plot(regionPossums, covariate = "ucl", breaks=seq(0,3,0.2),
    plottype = "shaded")
plot(regionPossums, covariate = "ucl", plottype = "contour",
    add = TRUE, levels = seq(0,2.7,0.2))
title("upper 95% surface")

## annotate with CI
par(mfrow = c(1,1))
plot(regionPossums, plottype = "shaded", mesh = NA, breaks = 20)
plot(traps(possumCH), add = TRUE, detpar = list(col = "black"))

if (interactive()) {
    spotHeight(regionPossums, dec = 1, pre = c("lcl","ucl"), cex = 0.8)
}

## perspective plot
pm <- plot(regionPossums, plottype = "persp", box = FALSE, zlim =
    c(0,3), phi=30, d = 5, col = "green", shade = 0.75, border = NA)
lines(trans3d (possumremovalarea$x, possumremovalarea$y,
     rep(1,nrow(possumremovalarea)), pmat = pm))

par(mfrow = c(1,1), mar = c(5, 4, 4, 2) + 0.1)  ## reset to default

## compare estimates of region N
## grid cell area is 0.01 ha
sum(covariates(regionPossums)[,"D.0"]) * 0.01
region.N(possum.model.Ds, regionmask)


## End(Not run)
```

print.capthist *Print Detections*

Description

Print method for capthist objects.

Usage

```
## S3 method for class 'capthist'
print(x, ..., condense = FALSE, sortrows = FALSE)
```

Arguments

x	capthist object
...	arguments to pass to print.default
condense	logical, if true then use condensed format for 3-D data
sortrows	logical, if true then sort output by animal

Details

The condense option may be used to format data from proximity detectors in a slightly more readable form. Each row then presents the detections of an individual in a particular trap, dropping rows (traps) at which the particular animal was not detected.

Value

Invisibly returns a dataframe (condense = TRUE) or array in the format printed.

See Also

print, capthist

Examples

```
## simulated detections of simulated default population of 5/ha
print(sim.capthist(make.grid(nx=5,ny=3)))
```

`print.secr` *Print or Summarise secr Object*

Description

Print results from fitting a spatially explicit capture–recapture model or generate a list of summary values.

Usage

```
## S3 method for class 'secr'
 print(x, newdata = NULL, alpha = 0.05, deriv = FALSE, call = TRUE, ...)
## S3 method for class 'secr'
 summary(object, newdata = NULL, alpha = 0.05, deriv = FALSE, ...)
```

Arguments

`x`	`secr` object output from `secr.fit`
`object`	`secr` object output from `secr.fit`
`newdata`	optional dataframe of values at which to evaluate model
`alpha`	alpha level
`deriv`	logical for calculation of derived D and esa
`call`	logical; if TRUE the call is printed
`...`	other arguments optionally passed to derived.secr

Details

Results from `print.secr` are potentially complex and depend upon the analysis (see below). Optional `newdata` should be a dataframe with a column for each of the variables in the model. If `newdata` is missing then a dataframe is constructed automatically. Default `newdata` are for a naive animal on the first occasion; numeric covariates are set to zero and factor covariates to their base (first) level. Confidence intervals are 100 (1 – alpha) % intervals.

call	the function call (optional)
version,time	secr version, date and time fitting started, and elapsed time
Detector type	‘single’, ‘multi’, ‘proximity’ etc.
Detector number	number of detectors
Average spacing	
x-range	
y-range	
New detector type	as fitted when details$newdetector specified
N animals	number of distinct animals detected
N detections	number of detections
N occasions	number of sampling occasions
Mask area	

Model	model formula for each 'real' parameter
Fixed (real)	fixed real parameters
Detection fn	detection function type (halfnormal or hazard-rate)
N parameters	number of parameters estimated
Log likelihood	log likelihood
AIC	Akaike's information criterion
AICc	AIC with small sample adjustment (Burnham and Anderson 2002)
Beta parameters	coef of the fitted model, SE and confidence intervals
vcov	variance-covariance matrix of beta parameters
Real parameters	fitted (real) parameters evaluated at base levels of covariates
Derived parameters	derived estimates of density and mean effective sampling area (optional)

Derived parameters (see `derived`) are computed only if `deriv = TRUE`.

Value

The `summary` method constructs a list of outputs similar to those printed by the `print` method, but somewhat more concise and re-usable:

versiontime	secr version, and date and time fitting started
traps	detector summary
capthist	capthist summary
mask	mask summary
modeldetails	miscellaneous model characteristics (CL etc.)
AICtable	single-line output of AIC.secr
coef	table of fitted coefficients with CI
predicted	predicted values ('real' parameter estimates)
derived	output of derived.secr (optional)

References

Burnham, K. P. and Anderson, D. R. (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. Second edition. New York: Springer-Verlag.

See Also

`AIC.secr`, `secr.fit`

Examples

```
## load & print previously fitted null (constant parameter) model
print(secrdemo.0)

summary(secrdemo.0)

## combine AIC tables from list of summaries
do.call(AIC, lapply(list(secrdemo.b, secrdemo.0), summary))
```

```
## Not run:

print(secrdemo.CL, deriv = TRUE)


## End(Not run)
```

`print.traps` *Print Detectors*

Description

Print method for `traps` objects.

Usage

```
## S3 method for class 'traps'
print(x, ...)
```

Arguments

`x`	`traps` object
`...`	arguments to pass to `print.default`

See Also

`print`, `traps`

Examples

```
print(make.grid(nx = 5, ny = 3))
```

`randomHabitat` *Random Landscape*

Description

The Modified Random Cluster algorithm of Saura and Martinez-Millan (2000) is used to generate a mask object representing patches of contiguous 'habitat' cells (pixels) within a 'non-habitat' matrix ('non-habitat' cells are optionally dropped). Spatial autocorrelation (fragmentation) of habitat patches is controlled via the parameter 'p'. 'A' is the expected proportion of 'habitat' cells.

`randomDensity` is a wrapper for `randomHabitat` that may be used as input to `sim.popn`.

Usage

```
randomHabitat(mask, p = 0.5, A = 0.5, directions = 4, minpatch = 1,
drop = TRUE, covname = "habitat", plt = FALSE, seed = NULL)

randomDensity(mask, parm)
```

Arguments

mask	secr mask object to use as template
p	parameter to control fragmentation
A	parameter for expected proportion of habitat
directions	integer code for adjacency (rook's move 4 or queen's move 8)
minpatch	integer minimum size of patch
drop	logical for whether to drop non-habitat cells
covname	character name of covariate when drop = FALSE
plt	logical for whether intermediate stages should be plotted
seed	either NULL or an integer that will be used in a call to set.seed
parm	list of arguments for randomHabitat, with added argument D

Details

Habitat is simulated within the region defined by the cells of mask. The region may be non-rectangular.

The algorithm comprises stages A-D:

A. Randomly select proportion p of cells from the input mask

B. Cluster selected cells with any immediate neighbours as defined by directions

C. Assign clusters to 'non-habitat' (probability 1–A) and 'habitat' (probability A)

D. Cells not in any cluster from (B) receive the habitat class of the majority of the <=8 adjacent cells assigned in (C), if there are any; otherwise they are assigned at random (with probabilities 1–A, A).

Fragmentation declines, and cluster size increases, as p increases up to the 'percolation threshold' which is about 0.59 in the default case (Saura and Martinez-Millan 2000 p.664).

If minpatch > 1 then habitat patches of less than minpatch cells are converted to non-habitat, and vice versa. This is likely to cause the proportion of habitat to deviate from A.

If drop = FALSE a binary-valued (0/1) covariate with the requested name is included in the output mask, which has the same extent as the input. Otherwise, non-habitat cells are dropped and no covariate is added.

The argument 'parm' for randomDensity is a list with average density D and an optional subset of named values to override the defaults (p = 0.5, A = 0.5, directions = 4, minpatch = 1, plt = FALSE, seed = NULL). 'rescale' is a further optional component of 'parm'; if 'rescale = TRUE' then the pixel-specific densities are adjusted upwards by the factor 1/A to maintain the same expected number of activity centres as if the nominal density applied throughout. Arguments 'mask' and 'drop' of randomHabitat are substituted automatically.

Value

For randomHabitat –

An object of class 'mask'. By default (`drop = TRUE`) this has fewer rows (points) than the input mask.

The attribute "type" is a character string formed from `paste('MRC p=',p, ' A=',A, sep='')`.

The RNG seed is stored as attribute 'seed' (see secrRNG).

For randomDensity –

A vector of cell-specific densities.

Note

Single-linkage clustering and adjacency operations use functions 'clump' and 'adjacency' of the package **raster**; 'clump' also requires package **igraph0** (**raster** still uses this deprecated version). Optional plotting of intermediate stages (`plt = TRUE`) uses the plot method for rasterLayers in **raster**.

A non-rectangular input mask is padded out to a rectangular rasterLayer for operations in **raster**; cells added as padding are ultimately dropped.

The procedure of Saura and Martinez-Millan (2000) has been followed as far as possible, but this implementation may not match theirs in every detail.

This implementation allows only two habitat classes. The parameter A is the *expected* value of the habitat proportion; the *realised* habitat proportion may differ quite strongly from A, especially for large p (e.g., $p > 0.5$).

Anisotropy is not implemented; it would require skewed adjacency filters (i.e. other than rook- or queen-move filters) that are not available in **raster**.

Gaussian random fields provide an alternative method for simulating random habitats (e.g., rLGCP option in `sim.popn`).

References

Hijmans, R. J. and van Etten, J. (2011) raster: Geographic analysis and modeling with raster data. R package version 1.9-33. https://CRAN.R-project.org/package=raster.

Saura, S. and Martinez-Millan, J. (2000) Landscape patterns simulation with a modified random clusters method. *Landscape Ecology*, **15**, 661–678.

See Also

`mask`, `make.mask`, `sim.popn`

Examples

```
## Not run:

tempmask <- make.mask(nx = 100, ny = 100, spacing = 20)
mrcmask <- randomHabitat(tempmask, p = 0.4, A = 0.4)
plot(mrcmask, dots = FALSE, col = "green")
pop <- sim.popn(10, mrcmask, model2D = "IHP")
```

```
plot(pop, add = TRUE)

# OR
plot(sim.popn(D = randomDensity, core = tempmask, model2D = "IHP",
    details = list(D = 10, p = 0.4, A = 0.4, plt = TRUE)),
    add = TRUE, frame = FALSE)

## plot intermediate steps A, C, D
opar <- par(mfrow = c(1,3))
mrcmask <- randomHabitat(tempmask, p = 0.4, A = 0.4, plt = TRUE)
par(opar)

## keep non-habitat cells
mrcmask <- randomHabitat(tempmask, p = 0.4, A = 0.4, drop = FALSE)
plot(mrcmask, covariate = "habitat", dots = FALSE,
    col = c("grey","green"), breaks = 2)

## effect of purging small patches
opar <- par(mfrow=c(1,2))
mrcmask <- randomHabitat(tempmask, p = 0.4, A = 0.4, minpatch = 1)
plot(mrcmask, dots = FALSE, col  ="green")
mrcmask <- randomHabitat(tempmask, p = 0.4, A = 0.4, minpatch = 5)
plot(mrcmask, dots = FALSE, col  ="green")
par(opar)

## End(Not run)
```

---

raster *Create a RasterLayer Object from Mask or Dsurface*

Description

Methods to convert **secr** object to a RasterLayer object.

Usage

```
## S4 method for signature 'mask'
raster(x, covariate, values = 1, crs = NA)

## S4 method for signature 'Dsurface'
raster(x, covariate, values = 1, crs = NA)
## S4 method for signature 'mask'
rast(x, covariate, values = 1, crs = "")
## S4 method for signature 'Dsurface'
rast(x, covariate, values = 1, crs = "")
```

Arguments

`x`	mask or Dsurface object
`covariate`	character name of covariate to provide values for RasterLayer
`values`	numeric values for RasterLayer
`crs`	character or object of class CRS. Optional PROJ.4 type description of a Coordinate Reference System (map projection).

Details

There are two ways to specify the values to be used. If `covariate` is provided then the values of the corresponding covariate of the mask or Dsurface are used. Otherwise, `values` is duplicated to the required number of rows.

The resulting RasterLayer may optionally include a PROJ.4 map projection defined via `crs`. The specification may be very simple (as in the example below) or complex, including an explicit datum and other arguments. Projections are used by **sf**, **terra**, **raster**, **sp** and other packages. See `raster` for further explanation and links.

The S3 classes 'mask' and 'Dsurface' are defined in **secr** as virtual S4 classes. This enables these extensions to the list of S4 methods defined in **raster** and **terra**.

Although these methods work 'standalone', it is currently necessary to load the **terra** or **raster** package to do much with the result (e.g., plot it).

Value

RasterLayer (raster)
SpatRaster (rast)

Note

Prior to **secr** 2.9.5 these methods could fail unpredictably because an intermediate array was badly dimensioned due to truncation of a floating point value.

See Also

`raster`, `rast`

Examples

```
## Not run:

shorePossums <- predictDsurface(possum.model.Ds)
tmp <- raster(shorePossums, covariate = "D.0")
library(raster)
plot(tmp, useRaster = FALSE)

## alternative with same result
tmp <- raster(shorePossums, values = covariates(shorePossums)$D.0)

## set the projection
```

```
## here the crs PROJ.4 spec refers simply to the old NZ metric grid
tmp <- raster(shorePossums, "D.0", crs = "+proj=nzmg")
## check the projection
proj4string(tmp)

# mask to rast
dtsrast <- rast(possummask, covariate = 'd.to.shore')
plot(dtsrast)

# Dsurface to rast
class(shorePossums) <- c('mask', 'data.frame') # or change "mask" to "Dsurface" above
Drast <- rast(shorePossums, covariate = 'D.0')
plot(Drast)


## End(Not run)
```

`rbind.capthist` *Combine capthist Objects*

Description

Form a single `capthist` object from two or more compatible `capthist` objects.

Usage

```
  MS.capthist(...)
  append.capthist(..., synchronous = TRUE)

  ## S3 method for class 'capthist'
rbind(..., renumber = TRUE, pool = NULL, verify = TRUE)
```

Arguments

`...`	one or more `capthist` objects or lists of `capthist` objects
`synchronous`	logical; if TRUE occasions are assumed to coincide
`renumber`	logical, if TRUE assigns new composite individual ID
`pool`	list of vectors of session indices or names
`verify`	logical, if TRUE the output is checked with `verify`

Details

`MS.capthist` concatenates the sessions in the input objects as one multi-session capthist object. Each session may use a different detector array (traps) and a different number of sampling occasions. Session names are derived implicitly from the inputs, or may be given explicitly (see Examples); if any name is duplicated, all will be replaced with sequential integers. The ... argument may include lists of single-session capthist objects.

`append.capthist` constructs a single-session capthist assuming the inputs are independent capthist objects. Individuals and detectors are renumbered. Occasions are assumed to be synchronous by default. This is an alternative to rbind when the detector array varies.

The `rbind` method for capthist objects is used to pool capture data from more than one session into a single session. The number of rows in the output session is the sum of the number of rows in the input sessions (i.e. each animal appears in only one session). Sessions to be pooled with `rbind` must have the same number of capture occasions and use the same detectors (traps).

For `rbind.capthist`, the ... argument may be

1. A series of single-session capthist objects, which are pooled to form one new single-session object, or
2. One multi-session capthist object, when the components of 'pool' are used to define combinations of old sessions; e.g. `pool = list(A=1:3, B=4:5)` produces an object with two sessions (named 'A' and 'B') from 5 old ones. If `pool = NULL` (the default) then all the sessions are pooled to form one single-session capthist object.

The names of arguments other than ... should be given in full. If `renumber = TRUE` (the default), the session name will be prepended to the animal ID before pooling: animals 1, 2 and 3 in Session A will become A.1, A.2 and A.3, while those in Session B become B.1, B.2 and B.3. This ensures that each animal has a unique ID. If `renumber = FALSE`, the animal IDs will not change.

Other attributes (xy, signal) are handled appropriately. If the signal threshold (attribute 'cutval') differs among sessions, the maximum is used and detections of lower signal strength are discarded.

The use of `rbind.capthist` to concatenate sessions is now deprecated: use `MS.capthist`.

Although `MS.capthist` looks like an S3 method, it isn't. The full function name must be used. `rbind.capthist` became an S3 method in **secr** 3.1, so it is called as `rbind` alone.

Value

For `MS.capthist`, a multi-session object of class 'capthist' with number of sessions equal to the number of sessions in the objects in ....

For `append.capthist`, a single-session of class 'capthist'. When 'synchronous' the dimensions are $(\sum n_i, \max S_i, \sum K_i)$ where (n_i, S_i, K_i) are the numbers of individuals, occasions and detectors in the i-th input. Otherwise the dimensions are $(\sum n_i, \sum S_i, \sum K_i)$. An index to the original component is saved in the traps covariate 'sub'.

For `rbind.capthist`, either an object of class 'capthist' with one session formed by pooling the sessions in the input objects, or a capthist object with more than one session, each formed by pooling groups of sessions defined by the 'pool' argument. Covariate columns that appear in all input sessions are retained in the output.

See Also

`capthist`, `subset.capthist`

Examples

```
## extend a multi-session object
## we fake the 2010 data by copying from 2005
## note how we name the appended session
```

```
fakeCH <- ovenCH[["2005"]]
MS.capthist(ovenCH, "2010" = fakeCH)

## simulate sessions for 2-part mixture
temptrap <- make.grid(nx = 8, ny = 8)
temp1 <- sim.capthist(temptrap,
    detectpar = list(g0 = 0.1, sigma = 40))
temp2 <- sim.capthist(temptrap,
    detectpar = list(g0 = 0.2, sigma = 20))

## concatenate sessions
temp3 <- MS.capthist(large.range = temp1, small.range = temp2)
summary(temp3)
## session-specific movement statistic
RPSV(temp3)

## pool sessions
temp4 <- rbind(temp1, temp2)
summary(temp4)
RPSV(temp4)

## compare mixture to sum of components
## note `detectors visited' is not additive for 'multi' detector
## nor is `detectors used'
(summary(temp1)$counts +  summary(temp2)$counts) -
    summary(temp4)$counts

## Not run:

## compare two different model fits
tempfit3 <- secr.fit(temp3, CL = TRUE, buffer = 150, model = list
    (g0 ~ session, sigma ~ session), trace = FALSE)
predict(tempfit3)

## if we can tell which animals had large ranges...
covariates(temp4) <- data.frame(range.size = rep(c("large",
    "small"), c(nrow(temp1), nrow(temp2))))
tempfit4 <- secr.fit(temp4, CL = TRUE, buffer = 150, model = list
    (g0 ~ range.size, sigma ~ range.size), trace = FALSE)
predict(tempfit4, newdata = data.frame(range.size = c("large",
    "small")))

## polygon data
pol1 <- make.poly()
pol2 <- make.poly(x = c(50,50,150,150))
ch1 <- sim.capthist(pol1, popn = list(D = 30), detectfn = 'HHN',
   detectpar = list(lambda0 = 0.3))
ch2 <- sim.capthist(pol2, popn = list(D = 30), detectfn = 'HHN',
   detectpar = list(lambda0 = 0.3))
plot(ch1); plot(pol2, add = TRUE); plot(ch2, add = TRUE)


## End(Not run)
```

rbind.popn *Combine popn Objects*

Description

Form a single popn object from two or more existing popn objects, or a list.

Usage

```
## S3 method for class 'popn'
 rbind(..., renumber = TRUE)
```

Arguments

...	one or more popn objects
renumber	logical for whether row names in the new object should be set to the row indices

Details

An attempt to combine objects will fail if they conflict in their `covariates` attributes.

From **secr** 3.1 this is an S3 method and list input is not allowed.

Value

An object of class popn with number of rows equal to the sum of the rows in the input objects.

See Also

popn

Examples

```
## generate and combine two subpopulations
trapobj <- make.grid()
p1 <- sim.popn(D = 3, core = trapobj)
p2 <- sim.popn(D = 2, core = trapobj)
covariates(p1) <- data.frame(size = rep("small", nrow(p1)))
covariates(p2) <- data.frame(size = rep("large", nrow(p2)))
pop <- rbind(p1,p2)

## or
pop <- do.call(rbind, list(p1,p2))
```

rbind.traps *Combine traps Objects*

Description

Form a single traps object from two or more existing traps objects.

Usage

```
## S3 method for class 'traps'
rbind(..., renumber = TRUE, addusage, checkdetector = TRUE, suffix = TRUE)
```

Arguments

...	one or more traps objects
renumber	logical for whether row names in the new object should be set to the row indices
addusage	integer vector; if specified and the inputs lack usage attributes then a binary usage attribute will be generated with the given number of occasions for each input
checkdetector	logical; if TRUE then variation in the detector attribute triggers a warning
suffix	logical; if TRUE then suffix to the row names indicates source

Details

An attempt to combine objects will fail if they conflict in their covariates attributes. Differences in the usage attribute are handled as follows. If usage is missing for all inputs and addusage = TRUE is specified then usage codes are generated automatically (positive for the specified number of occasions). If usage is specified for one input but not other(s), the missing values are constructed assuming all detectors were operated for the maximum number of occasions in any input. If inputs differ in the number of 'usage' columns (occasions), the smaller matrices are padded with 'zero' columns to the maximum number of columns in any input.

...may be a single multi-session traps object (from 2.10.0).

By default (and always prior to 3.1.1) row names include a suffix (e.g., ".1", or ".2") to indicate the original object (first, second etc.). A suffix is added automatically to all names if any name is duplicated, and a warning is generated.

Value

An object of class traps with number of rows equal to the sum of the rows in the input objects.

See Also

traps, subset.traps

Examples

```
## nested hollow grids
hollow1 <- make.grid(nx = 8, ny = 8, hollow = TRUE)
hollow2 <- shift(make.grid(nx = 6, ny = 6, hollow = TRUE),
    c(20, 20))
nested <- rbind (hollow1, hollow2)
plot(nested, gridlines = FALSE, label = TRUE)
```

---

read.capthist *Import or export data*

Description

Data in the DENSITY formats for capture data and trap layouts may be imported as a `capthist` object for analysis in **secr**. Data in a `capthist` object may also be exported in these formats for use in DENSITY (Efford 2012). `read.capthist` inputs data from text files and constructs a `capthist` object in one step using the functions `read.traps` and `make.capthist`. Data may also be read from Excel spreadsheets if the package **readxl** is installed (see secr-datainput.pdf).

Usage

```
read.capthist(captfile, trapfile, detector = "multi", fmt = c("trapID","XY"),
    noccasions = NULL, covnames = NULL, trapcovnames = NULL,
    cutval = NULL, verify = TRUE, noncapt = "NONE", tol = 0.01, snapXY = FALSE,
    markocc = NULL, ...)

write.capthist(object, filestem = deparse(substitute(object)),
     sess = "1", ndec = 2, covariates = FALSE, tonumeric = TRUE, ...)
```

Arguments

`captfile`	name of capture data file
`trapfile`	name of trap layout file or (for a multi-session captfile) a vector of file names, one for each session
`detector`	character value for detector type (‘single’, ‘multi’, ‘proximity’, etc.)
`fmt`	character value for capture format (‘trapID’ or ‘XY’)
`noccasions`	number of occasions on which detectors were operated
`covnames`	character vector of names for individual covariate fields in ‘captfile’
`trapcovnames`	character vector of names for detector covariate fields in ‘trapfile’
`cutval`	numeric, threshold of signal strength for ‘signal’ detector type
`verify`	logical if TRUE then the resulting capthist object is checked with `verify`
`noncapt`	character value; animal ID used for ‘no captures’
`tol`	numeric, snap tolerance in metres

snapXY	logical; if TRUE then fmt = 'XY' uses nearest trap within tol
markocc	integer vector distinguishing marking occasions (1) from sighting occasions (0)
...	other arguments passed to read.table, write.table and count.fields
object	capthist object with the captures and trap locations to export
filestem	character value used to form names of output files
sess	character session identifier
ndec	number of digits after decimal point for x,y coordinates
covariates	logical or a character vector of covariates to export
tonumeric	logical for whether factor and character covariates should be converted to numeric values on output

Details

read.capthist

captfile should record one detection on each line. A detection comprises a session identifier, animal identifier, occasion number (1, 2,...,*S* where *S* is the number of occasions), and a detector identifier (fmt = "trapID") or X- and Y-coordinates (fmt = "XY"). Each line of trapfile has a detector identifier and its X- and Y-coordinates. In either file type the identifiers (labels) may be numeric or alphanumeric values. Values should be separated by blanks or tabs unless (i) the file name ends in '.csv' or (ii) sep = ",," is passed in ..., in which case commas are assumed. Blank lines and any text after '#' are ignored. For further details see secr-datainput.pdf, make.capthist and 'Data formats' in the help for DENSITY.

The noccasions argument is needed only if there were no detections on the final occasion; it may be a positive integer (constant across all sessions) or a vector of positive integers, one for each session. covnames is needed only when captfile includes individual covariates. Likewise for trapcovnames and detector covariates. Values of noccasions and covnames are passed directly to make.capthist, and trapcovnames is passed to read.traps.

A session identifier is required even for single-session capture data. In the case of data from multiple sessions, trapfile may be a vector of file names, one for each session.

Additional data may be coded as for DENSITY. Specifically, captfile may include extra columns of individual covariates, and trapfile may code varying usage of each detector over occasions and detector covariates.

markocc is needed only if sightings of unmarked animals are potentially recorded on some occasions. If the data span multiple sessions with differing combinations of marking and sighting occasions then markocc may be a list with one vector per session.

The function read.telemetry is a simplified version of read.capthist for telemetry data.

write.capthist

For a single-session analysis, DENSITY requires one text file of capture data and one text file with detector coordinates (the 'trap layout' file). write.capthist constructs names for these files by appending 'capt.txt' and 'trap.txt' to filestem which defaults to the name of the capthist object. If filestem is empty then output goes to the console.

If object contains multiple sessions with differing traps then a separate trap layout file is exported for each session and each file name includes the session name. All capture data are exported to one

file regardless of the number of sessions. The DENSITY format used is 'TrapID' except when x-y coordinates are specific to a detection (i.e., polygon and transect detectors).

`covariates` controls the export of both detector and individual covariates. If it is TRUE or FALSE then it is taken to apply to both. A vector of covariate names is used as a lookup for both detector and capthist covariate fields: covariates are exported if their name matches; this may be used to export any combination of (uniquely named) detector and capthist covariates.

Existing text files will be replaced without warning. In the case of a multi-session capthist file, session names are taken from `object` rather than `sess`. Session names are truncated to 17 characters with blanks and commas removed.

To export data in comma-delimited ('.csv') format, pass `sep = ","` in ... . The resulting files have extension '.csv' rather than '.txt' and may be opened with spreadsheet software.

Warning

`write.capthist` does not work for mark–resight data.

Note

The original DENSITY formats accommodate 'single', 'multi' and 'proximity' data. Data for the newer detector types ('count', 'signal', 'polygon', 'polygonX', 'transect', 'transectX' and 'telemetryonly') may be input using the DENSITY formats with minor variations. They may also be output with `write.capthist`, but a warning is given that DENSITY does not understand these data types. See `detector` and secr-datainput.pdf for more.

The ... argument is useful for some special cases. For example, if your input uses ';' instead of '#' for comments (';' is also valid in DENSITY) then set `comment.char = ";"` in `read.capthist`.

In a similar fashion, write comma- or tab-separated values by setting `sep = ","` or `sep = "\t"` respectively.

The arguments of `count.fields` are a subset of those of `read.table` so ... is limited to any of {sep, quote, skip, blank.lines.skip, comment.char}.

If you fail to set `fmt` correctly in `read.capthist` then the error message from `verify` may be uninformative.

References

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

See Also

`read.telemetry`, `read.traps`, `make.capthist`, `write.captures`, `write.traps`, `read.table`

Examples

```
## export ovenbird capture histories
## the files "ovenCHcapt.txt" and "ovenCHtrap.txt" are
## placed in the current folder (check with getwd() or dir())
```

```
## Not run:
write.capthist(ovenCH)

## End(Not run)
```

---

read.mask *Read Habitat Mask From File*

Description

Read coordinates of points on a habitat mask from a text file.

Usage

```
read.mask(file = NULL, data = NULL, spacing = NULL, columns = NULL, ...)
```

Arguments

file	character string with name of text file
data	dataframe
spacing	spacing of grid points in metres
columns	character vector naming the columns to save as covariates
...	other arguments to pass to read.table

Details

For file input, the x and y coordinates are usually the first two values on each line, separated by white space. If the file starts with a line of column headers and 'header = TRUE' is passed to read.table in the ... argument then 'x' and 'y' need not be the first two fields.

data is an alternative input route if the x and y coordinates already exist in R as columns in a dataframe. Only one of data or file should be specified.

The grid cell size spacing should be provided if known. If it is not provided then an attempt is made to infer it from the minimum spacing of points. This can be slow and may demand more memory than is available. In rare cases (highly fragmented masks) it may also yield the wrong answer.

From 2.3.0, additional columns in the input are saved as covariates. The default (columns = NULL) is to save all columns.

Value

object of class mask with type 'user'

Note

read.mask creates a single-session mask. If used in secr.fit with a multi-session capthist object a single-session mask will be replicated to the number of sessions. This is appropriate if all sessions relate to the same geographical region. If the 'sessions' relate to different regions you will need to construct a multi-session mask as a list of single-session masks (e.g. mask <- list(mask1, mask2, mask3)).

See Also

mask

Examples

```
## Replace file name with a valid local name and remove `#'
# read.mask (file = "c:\\myfolder\\mask.txt",
# spacing = 3, header = TRUE)
## "mask.txt" should have lines like this
# x   y
# 265 265
# 268 265
# ...
```

---

read.telemetry *Import Telemetry Fixes*

Description

A shortcut function for constructing a telemetry capthist object from a file of telemetry fixes. Telemetry data are generally similar in format to polygon data (see also addTelemetry).

Usage

```
read.telemetry(file = NULL, data = NULL, covnames = NULL, verify = TRUE, ...)
```

Arguments

file	character name of text file
data	data.frame containing coordinate data (alternative to file)
covnames	character vector of names for individual covariates
verify	logical for whether to check input
...	other arguments passed to countfields, read.table etc.

Details

Input data may be in a text file (argument file) or a dataframe (argument data). Data should be in the XY format for function 'read.capthist' i.e. the first 5 columns should be Session, ID, Occasion, X, Y. Further columns are treated as individual covariates.

No 'traps' input is required. A traps object is generated automatically.

Value

An secr capthist object including attribute 'telemetryxy' with the x-y coordinates, and a 'traps' object with detector type = 'telemetry'

See Also

addTelemetry, read.capthist

Examples

```
## example file of telemetry data included in secr
filename <- system.file("extdata/telemetrydemo.txt", package = "secr")

## peek at raw data
head(readLines(filename))

teCH <- read.telemetry(filename)

plot(teCH)
head(teCH)
RPSV(teCH, CC = TRUE)

fit <- secr.fit(teCH, detectfn = 'HHN', trace = FALSE,
    details = list(safeLL = TRUE, uselog = TRUE), buffer = 100)
predict(fit)
```

---

read.traps *Read Detector Data From File*

Description

Construct an object of class traps with detector locations from a text file or data frame. Usage per occasion and covariates may be included. Data may also be read from an Excel spreadsheet (see secr-datainput.pdf).

Usage

```
read.traps(file = NULL, data = NULL, detector = "multi", covnames =
NULL, binary.usage = TRUE, markocc = NULL, trapID = NULL, ...)
```

Arguments

`file`	character string with name of text file
`data`	data frame of detector coordinates
`detector`	character string for detector type
`covnames`	character vector of names for detector covariate fields
`binary.usage`	logical; if FALSE will read usage fields as continuous effort
`markocc`	integer vector distinguishing marking occasions (1) from sighting occasions (0)
`trapID`	character column containing detector names (see Details)
`...`	other arguments to pass to `read.table`

Details

Reads a text file in which the first column is a character string (see Note) identifying a detector and the next two columns are its x- and y-coordinates, separated by white space. The coordinates optionally may be followed by a string of codes '0' or '1' indicating whether the detector was operated on each occasion. Trap-specific covariates may be added at the end of the line preceded by '/'. This format is compatible with the Density software (Efford 2012), except that all detectors are assumed to be of the same type (usage codes greater than 1 are treated as 1), and more than one covariate may be specified.

If `file` is missing then x-y coordinates will be taken instead from `data`, which should include columns 'x' and 'y'. Row names of `data` are read as detector identifiers unless `trapID` is specified. This option does not allow for `covariates` or `usage`, but they maybe added later.

`detector` specifies the behaviour of the detector following Efford et al. (2009). 'single' refers to a trap that is able to catch at most one animal at a time; 'multi' refers to a trap that may catch more than one animal at a time. For both 'single' and 'multi' detectors a trapped animals can appear at only one detector per occasion. Detectors of type 'proximity', such as camera traps and hair snags for DNA sampling, allow animals to be recorded at several detectors on one occasion. See `detector` for further detector types.

For polygon and transect detector types, each line corresponds to a vertex and starts with a code to identify the polygon or transect (hence the same code appears on 2 or more lines). For input from a dataframe the code column should be named 'polyID'. Also, usage and covariates are for the polygon or transect as a whole and not for each vertex. Usage 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.

Value

An object of class `traps` comprising a data frame of x- and y-coordinates, the detector type ('single', 'multi', 'proximity', 'count', 'polygon' etc.), and possibly other attributes.

Note

Detector names, which become row names in the traps object, should not contain underscores.

Prior to 4.3.1 the function did not read usage or covariates from xls or data input.

References

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

See Also

traps, make.grid, detector

Examples

```
## Not run:
## "trap.txt" should have lines like this
# 1     365     365
# 2     365     395
# 3     365     425
# etc.
## in following, replace file name with a valid local name
filename <- paste0(system.file("extdata", package = "secr"), '/trap.txt')
tr1 <- read.traps (filename, detector = "proximity")
summary(tr1)

## Or if we have a dataframe of coordinates...
mytrapdf <- data.frame(x = c(365,365,365), y = c(365,395,425),
                       row.names = c('A','B','C'))
mytrapdf
#     x   y
# A 365 365
# B 365 395
# C 365 425
## ...then we can convert it to a `traps' object with
tr2 <- read.traps(data = mytrapdf)
summary(tr2)


## End(Not run)
```

---

rectangularMask *Rectangular Mask*

Description

Convert a mask or Dsurface with an irregular outline into a mask or Dsurface with a rectangular outline and the same bounding box. This enables contour plotting.

Usage

```
rectangularMask(mask)
```

Arguments

mask	object of class mask or Dsurface

Details

The covariates of new points are set to missing. The operation may be reversed (nearly) with `subset(rectmask, attr(rectmask, "OK"))`.

The results are unpredictable if the mask has been rotated.

Value

A rectangular mask or Dsurface with the same 'area', 'boundingbox', 'meanSD', 'polygon' and 'polygon.habitat' attributes as `mask`. A logical vector attribute 'OK' is added identifying the points inherited from `mask`.

See Also

`plot.Dsurface`

Examples

```
rMask <- rectangularMask(possummask)
plot(rMask)
plot(possummask, add = TRUE, col = "blue")
```

---

reduce *Combine Columns*

Description

Combine columns in a matrix-like object to create a new data set using the first non-zero value.

Usage

```
reduce (object, ...)
## Default S3 method:
reduce(object, columns, ...)
```

Arguments

object	object that may be coerced to a matrix
columns	list in which each component is a vector of subscripts for columns to be pooled
...	other arguments (not used currently)

Details

The first element of `columns` defines the columns of `object` for the first new column, the second for the second new column etc. This is a generic method. More useful methods exist for `capthist` and `traps` objects.

Value

A matrix with number of columns equal to `length(columns)`.

See Also

`capthist`, `reduce.capthist`, `reduce.traps`

Examples

```
## matrix with random zeros
temp <- matrix(runif(20), nc = 4)
temp[sample(20,10)] <- 0
temp

reduce(temp, list(1:2, 3:4))
```

---

reduce.capthist *Combine Occasions Or Detectors*

Description

Use these methods to combine data from multiple occasions or multiple detectors in a `capthist` or `traps` object, creating a new data set and possibly converting between detector types.

Usage

```
## S3 method for class 'traps'
reduce(object, newtraps = NULL, newoccasions = NULL, span = NULL,
    rename = FALSE, newxy = c('mean', 'first'), ...)

## S3 method for class 'capthist'
reduce(object, newtraps = NULL, span = NULL, rename =
    FALSE, newoccasions = NULL, by = 1, outputdetector = NULL,
    select = c("last","first","random"), dropunused = TRUE, verify
    = TRUE, sessions = NULL, ...)
```

Arguments

`object`	`traps` or `capthist` object
`newtraps`	list in which each component is a vector of subscripts for detectors to be pooled
`newoccasions`	list in which each component is a vector of subscripts for occasions to be pooled
`span`	numeric maximum span in metres of new detector
`rename`	logical; if TRUE the new detectors will be numbered from 1, otherwise a name will be constructed from the old detector names
`newxy`	character; coordinates when detectors grouped with 'newtraps'
`by`	number of old occasions in each new occasion
`outputdetector`	character value giving detector type for output (defaults to input)
`select`	character value for method to resolve conflicts
`dropunused`	logical, if TRUE any never-used detectors are dropped
`verify`	logical, if TRUE the `verify` function is applied to the output
`sessions`	vector of session indices or names (optional)
`...`	other arguments passed by reduce.capthist to reduce.traps, or by reduce.traps to hclust

Details

reduce.traps –

Grouping may be specified explicitly via `newtraps`, or implicitly by `span`.

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

`newxy = 'first'` selects the coordinates of the first detector in a group defined by 'newtraps', rather then the average of all detectors in group.

reduce.capthist –

The first component of `newoccasions` defines the columns of `object` for new occasion 1, the second for new occasion 2, etc. If `newoccasions` is NULL then all occasions are output. Subscripts in a component of `newoccasions` that do not match an occasion in the input are ignored. When the output detector is one of the trap types ('single', 'multi'), reducing capture occasions can result in locational ambiguity for individuals caught on more than one occasion, and for single-catch traps there may also be conflicts between individuals at the same trap. The method for resolving conflicts among 'multi' detectors is determined by `select` which should be one of 'first', 'last' or 'random'. With 'single' detectors `select` is ignored and the method is: first, randomly select* one trap per animal per day; second, randomly select* one animal per trap per day; third, when collapsing multiple days use the first capture, if any, in each trap.

Usage data in the `traps` attribute are also pooled if present; usage is summed over contributing occasions and detectors. If there is no 'usage' attribute in the input, and `outputdetector` is one of 'count', 'polygon', 'transect' and 'telemetry', a homogeneous (all-1's) 'usage' attribute is first generated for the input.

* i.e., in the case of a single capture, use that capture; in the case of multiple 'competing' captures draw one at random.

If newoccasions is not provided then old occasions are grouped into new occasions as indicated by the by argument. For example, if there are 15 old occasions and by = 5 then new occasions will be formed from occasions 1:5, 6:10, and 11:15. A warning is given when the number of old occasions is not a multiple of by as then the final new occasion will comprise fewer old occasions.

dropunused = TRUE has the possibly unintended effect of dropping whole occasions on which there were no detections.

A special use of the by argument is to combine all occasions into one for each session in a multi-session dataset. This is done by setting by = "all".

reduce.capthist may be used with non-spatial capthist objects (NULL 'traps' attribute) by setting verify = FALSE.

Value

reduce.traps –

An object of class traps with detectors combined according to newtraps or span. The new object has an attribute 'newtrap', a vector of length equal to the original number of detectors. Each element in newtrap is the index of the new detector to which the old detector was assigned (see Examples).

The object has no clusterID or clustertrap attribute.

reduce.capthist –

An object of class capthist with number of occasions (columns) equal to length(newoccasions); detectors may simulataneously be aggregated as with reduce.traps. The detector type is inherited from object unless a new type is specified with the argument outputdetector.

Warning

The argument named 'columns' was renamed to 'newoccasions' in version 2.5.0, and arguments were added to reduce.capthist for the pooling of detectors. Old code should work as before if all arguments are named and 'columns' is changed.

Note

The reduce method may be used to re-assign the detector type (and hence data format) of a capthist object without combining occasions or detectors. Set the object and outputdetector arguments and leave others at their default values.

Automated clustering can produce unexpected outcomes. In particular, there is no guarantee that clusters will be equal in size. You should inspect the results of reduce.traps especially when using span.

reduce.traps is not implemented for polygons or transects.

The function discretize converts polygon data to point-detector (multi, proximity or count) data.

See Also

capthist, subset.capthist, discretize, hclust, cutree

Examples

```
tempcapt <- sim.capthist (make.grid(nx = 6, ny = 6), nocc = 6)
class(tempcapt)

pooled.tempcapt <- reduce(tempcapt, newocc = list(1,2:3,4:6))
summary (pooled.tempcapt)

pooled.tempcapt2 <- reduce(tempcapt, by = 2)
summary (pooled.tempcapt2)

## collapse multi-session dataset to single-session 'open population'
onesess <- join(reduce(ovenCH, by = "all"))
summary(onesess)

# group detectors within 60 metres
plot (traps(captdata))
plot (reduce(captdata, span = 60), add = TRUE)

# plot linking old and new
old <- traps(captdata)
new <- reduce(old, span = 60)
newtrap <- attr(new, "newtrap")
plot(old, border = 10)
plot(new, add = TRUE, detpar = list(pch = 16), label = TRUE)
segments (new$x[newtrap], new$y[newtrap], old$x, old$y)

## Not run:

# compare binary proximity with collapsed binomial count
# expect TRUE for each year
for (y in 1:5) {
    CHA <- abs(ovenCHp[[y]])   ## abs() to ignore one death
    usage(traps(CHA)) <- matrix(1, 44, ncol(CHA))
    CHB <- reduce(CHA, by = 'all', output = 'count')
    # summary(CHA, terse = TRUE)
    # summary(CHB, terse = TRUE)
    fitA <- secr.fit(CHA, buffer = 300, trace = FALSE)
    fitB <- secr.fit(CHB, buffer = 300, trace = FALSE, binomN = 1, biasLimit = NA)
    A <- predict(fitA)[,-1]
    B <- predict(fitB)[,-1]
    cat(y, ' ', all(abs(A-B)/A < 1e-5), '\n')
}
## multi-session fit
## expect TRUE overall
CHa <- ovenCHp
for (y in 1:5) {
    usage(traps(CHa[[y]])) <- matrix(1, 44, ncol(CHa[[y]]))
    CHa[[y]][,,] <- abs(CHa[[y]][,,])
}
CHb <- reduce(CHa, by = 'all', output = 'count')
summary(CHa, terse = TRUE)
summary(CHb, terse = TRUE)
```

```
fita <- secr.fit(CHa, buffer = 300, trace = FALSE)
fitb <- secr.fit(CHb, buffer = 300, trace = FALSE, binomN = 1, biasLimit = NA)
A <- predict(fita)[[1]][,-1]
B <- predict(fitb)[[1]][,-1]
all(abs(A-B)/A < 1e-5)


## End(Not run)
```

---

region.N *Population Size*

Description

Estimate the expected and realised populations in a region, using a fitted spatially explicit capture–recapture model. Density is assumed to follow an inhomogeneous Poisson process in two dimensions. Expected N is the volume under a fitted density surface; realised N is the number of individuals within the region for the current realisation of the process (cf Johnson et al. 2010; see Note).

Usage

```
region.N(object, ...)

## S3 method for class 'secr'
region.N(object, region = NULL, spacing = NULL, session = NULL,
    group = NULL, se.N = TRUE, alpha = 0.05, loginterval = TRUE,
    keep.region = FALSE, nlowerbound = TRUE, RN.method = "poisson",
    pooled.RN = FALSE, ncores = NULL, ...)

## S3 method for class 'secrlist'
region.N(object, region = NULL, spacing = NULL, session = NULL,
    group = NULL, se.N = TRUE, alpha = 0.05, loginterval = TRUE,
    keep.region = FALSE, nlowerbound = TRUE, RN.method = "poisson",
    pooled.RN = FALSE, ncores = NULL, ...)
```

Arguments

object	secr object output from secr.fit
region	mask object defining the possibly non-contiguous region for which population size is required, or vector polygon(s) (see Details)
spacing	spacing between grid points (metres) if region mask is constructed on the fly
session	character session
group	group – for future use

se.N	logical for whether to estimate SE($\hat{N}$) and confidence interval
alpha	alpha level for confidence intervals
loginterval	logical for whether to base interval on log(N)
keep.region	logical for whether to save the raster region
nlowerbound	logical for whether to use n as lower bound when computing log interval for realised N
RN.method	character string for method used to calculate realised N (RN) and its sampling variance. 'poisson' or 'MSPE'.
pooled.RN	logical; if TRUE the estimate of realised N for a multi-session model is computed as if for combined sampling with all detectors (see Details)
ncores	integer number of threads to be used for parallel processing
...	other arguments (not used)

Details

If the density surface of the fitted model is flat (i.e. `object$model$D == ~1` or `object$CL == TRUE`) then $E(N)$ is simply the density multiplied by the area of `region`, and the standard error is also a simple product. In the conditional likelihood case, the density and standard error are obtained by first calling `derived`.

If, on the other hand, the density has been modelled then the density surface is predicted at each point in `region` and $E(N)$ is obtained by discrete summation. Pixel size may have a minor effect on the result - check by varying `spacing`. Sampling variance is determined by the delta method, using a numerical approximation to the gradient of $E(N)$ with respect to each beta parameter.

The region may be defined as a mask object (if omitted, the mask component of `object` will be used). Alternatively, `region` may be a SpatialPolygonsDataFrame object (see package **sp**), and a raster mask will be constructed on the fly using the specified spacing. See `make.mask` for an example importing a shapefile to a SpatialPolygonsDataFrame.

Note: The option of specifying a polygon rather than a mask for `region` does not work if the density model in `object` uses spatial covariates: these must be passed in a mask.

Group-specific N has yet to be implemented.

Population size is adjusted automatically for the number of clusters in 'mashed' models (see `mash`). However, the population size reported is that associated with a single cluster unless `regionmask` is specified.

`pooled.RN = TRUE` handles the special case of a multi-session model in which the region of interest spans several patches (i.e., sampling in each session is localised within `region`. This is not yet fully implemented.

Setting `ncores = NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

From version 5.5.0, detections on sighting and telemetry occasions are excluded in the computation of realised N, with a warning.

Realised N is not estimable for telemetry type 'marking' (see `addTelemetry`) because it depends on the probability that an animal is marked, which the model does not describe. For such models 'R.N' and `n` are returned as NA; expected N is unaffected.

Value

If `se.N = FALSE`, the numeric value of expected population size, otherwise, a dataframe with rows 'E.N' and 'R.N', and columns as below.

estimate	estimate of N (expected or realised, depending on row)
SE.estimate	standard error of estimated N
lcl	lower 100(1–alpha)% confidence limit
ucl	upper 100(1–alpha)% confidence limit
n	total number of individuals detected

For multiple sessions, the value is a list with one component per session, each component as above.

If `keep.region = TRUE` then the mask object for the region is saved as the attribute 'region' (see Examples).

The area in hectares of the region is saved as attribute 'regionarea'.

Note

The estimates of expected and realised N are generally very similar, or identical, but realised N usually has lower estimated variance, especially if the n detected animals comprise a large fraction.

Realised N is given by $R(N) = n + \int_B (1 - p.(X))D(X)dX$ (the second term represents undetected animals). This definition strictly holds only when region B is at least as large as the region of integration used to fit the model; only with this condition can we be sure all n detected animals have centres within B. The sampling variance of $R(N)$, technically a mean square prediction error (Johnson et al. 2010), is approximated by summing the expected Poisson variance of the true number of undetected animals and a delta-method estimate of its sampling variance, obtained as for $E(N)$.

By default, a shortcut is used to compute the sampling variance of realised N. With this option (RN.method = 'poisson') the sampling variance is the sampling variance of $E(N)$ minus the estimate of $E(N)$ (representing Poisson process variance). This has been found to give reliable confidence intervals in simulations (Efford and Fewster 2013).

If RN.method is neither 'MSPE' nor 'poisson' (ignoring case) then the estimate of expected N is also used for realised N, and the 'poisson' shortcut variance is used.

Johnson et al. (2010) use the notation $\mu(B)$ for expected N and $N(B)$ for realised N in region B.

In our case, the relative SE (CV) of $\mu(B)$ is the same as that for the estimated density D if D has been estimated using the Poisson distribution option in `secr.fit` or `derived()`. If D has been estimated with the binomial distribution option, its relative SE for simple models will be the same as that of $N(B)$, assuming that B is the full extent of the original mask.

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. and Fewster, R. M. (2013) Estimating population size by spatially explicit capture–recapture. *Oikos* **122**, 918–928.

Johnson, D. S., Laake, J. L. and Ver Hoef, J. M. (2010) A model-based approach for making ecological inference from distance sampling data. *Biometrics* **66**, 310–318.

See Also

secr.fit, derived, make.mask, expected.n, closedN

Examples

```
## Not run:

## routine examples using arbitrary mask from model fit
region.N(secrdemo.0)
region.N(secrdemo.CL)
region.N(ovenbird.model.D)

## region defined as vector polygon
## retain and plot region mask
temp <- region.N(possum.model.0, possumarea, spacing = 40,
    keep.region = TRUE)
temp
plot (attr(temp, "region"), type = "l")

## End(Not run)
```

RMarkInput *Convert Data to RMark Input Format*

Description

A single-session capthist object is formed by RMarkInput into a dataframe that may be passed directly to RMark.

Usage

```
RMarkInput(object, grouped = FALSE, covariates = TRUE)

unRMarkInput(df, covariates = TRUE)
```

Arguments

object	secr capthist object
grouped	logical for whether to replace each group of identical capture histories with a single line
covariates	logical or character vector; see Details
df	dataframe with fields ‘ch’ and ‘freq’

Details

To convert a multi-session object first collapse the sessions with `join`.

If `covariates` is TRUE then all columns of individual covariates in the input are appended as columns in the output. If `covariates` is a character-valued vector then only the specified covariates will be appended.

If both `grouped` and `covariates` are specified in `RMarkInput`, `grouped` will be ignored, with a warning.

Value

For RMarkInput –

Dataframe with fields ch and freq. 'ch' is a character string of 0's and 1's. If `grouped = FALSE` the rownames are retained and the value of 'freq' is 1 or -1. Negative values of 'freq' indicate removal.

The dataframe also includes individual covariates specified with `covariates`.

The attribute 'intervals' is copied from 'object', if present; otherwise it is set to a vector of zeros (indicating a closed-population sample).

For unRMarkInput –

A single-session capthist object with no traps attribute and hence no detector type (i.e. non-spatial capture histories). Covariates are copied as requested.

From **secr** 4.6.6, missing values (.) in input capture histories are converted to NA in the output, with a warning. The resulting capthist is unusable until the NAs are removed.

Note

In versions before 2.4.0, a spurious occasion was added by `RMarkInput` when grouped = FALSE. Thanks to Jeff Stetz for spotting this.

The default value for `grouped` changed to FALSE in secr 2.4.0.

References

Laake, J. and Rexstad E. (2008) Appendix C. RMark - an alternative approach to building linear models in MARK. In: Cooch, E. and White, G. (eds) Program MARK: A Gentle Introduction. 6th edition. Most recent edition available at www.phidot.org/software/mark/docs/book/.

See Also

`join`

Examples

```
## ovenCH is a 5-year mist-netting dataset
ovenRD <- RMarkInput (join(ovenCH))
head(ovenRD)

unRMarkInput(ovenRD)

RMarkInput(deermouse.ESG, covariates = FALSE, grouped = TRUE)
```

```
RMarkInput(deermouse.ESG, covariates = TRUE)

## Not run:
## fit robust-design model in RMark (MARK must be  installed)
library(RMark)
MarkPath <- 'c:/MARK'    ## adjust for your installation
ovenRD.data <- process.data(ovenRD, model = "Robust",
    time.interval = attr(ovenRD, "intervals"))
ovenRD.model <- mark(data = ovenRD.data, model = "Robust",
    model.parameters = list(p = list(formula = ~1, share = TRUE),
    GammaDoublePrime = list(formula = ~1),
    GammaPrime = list(formula = ~1),
    f0 = list(formula = ~1)))
cleanup(ask = FALSE)

## End(Not run)
```

RSE *RSE from Fitted Model*

Description

Precision of parameter estimates from an SECR model, expressed as relative standard error.

Usage

```
RSE(fit, parm = NULL, newdata = NULL)
```

Arguments

fit	secr or openCR fitted model or secrlist
parm	character; names of one or more real parameters (default all)
newdata	dataframe of covariates for predict.secr

Details

The relative standard error (RSE) of parameter θ is $RSE(\hat{\theta}) = \widehat{SE}(\theta)/\hat{\theta}$.

For a parameter estimated using a log link with single coefficient β, the RSE is also $\mathrm{RSE}(\hat{\theta}) = \sqrt{\exp(\mathrm{var}(\beta)) - 1}$. This formula is used wherever applicable.

Value

Named vector of RSE, or matrix if newdata has more than one row.

Note

The less explicit abbreviation CV has been used for the same quantity (sometimes expressed as a percentage). CV is used also for the relative standard deviation of a distribution.

From version 5.5.0, the 'fit' argument may be an secrlist. Results are provided for each component.

References

Efford, M. G. and Boulanger, J. 2019. Fast evaluation of study designs for spatially explicit capture–recapture. *Methods in Ecology and Evolution* **10**, 1529–1535.

See Also

CV

Examples

```
RSE(secrdemo.0)
```

Rsurface *Smoothed Resource Surface*

Description

Creates a smoothed resource surface from a covariate of a mask. Smoothing entails summing the value in each pixel weighted by a detection kernel centred on the focal pixel. The detection kernel represents home-range utilization with spatial scale sigma. The resulting surface is equivalent to the denominator used by Royle et al. (2013) to normalize site-specific detection.

Usage

```
Rsurface(mask, sigma, usecov = NULL, alpha2 = 1, detectfn = 'HHN', z = 1,
 inverse = FALSE, scale = TRUE)
```

Arguments

mask	secr habitat mask object (single-session)
sigma	numeric spatial scale of home range model
alpha2	numeric coefficient of spatial covariate
usecov	character name of resource covariate
detectfn	integer or character code for detection function
z	numeric shape parameter of home range model
inverse	logical; if TRUE the reciprocal of smoothed resource is returned
scale	logical; not used

Details

detectfn may be uniform ('UN') or one of the cumulative-hazard functions ('HHN', 'HHR', 'HEX', 'HAN', 'HCG') (or integer codes 4, 14:18; see detectfn).

The default 'HHN' corresponds to a halfnormal function on the hazard scale, or a bivariate circular normal home range.

If usecov is not named then it takes the value 1.0 for all points on the mask and zero otherwise.

The Rsurface can be used implicitly to normalize detection probability when fitting a model with detector-specific covariate equal to usecov (see details, but the process is intricate and not fully documented).

Value

An object with class c('Rsurface', 'mask', 'data.frame') and covariate 'Resource' (other covariates are retained from the input mask). The attribute 'scale' is 1.0 if scale = FALSE; otherwise it is the average of the resource over the masked area.

Note

Consider a focal pixel **s** and another point in the habitat mask **x**, with distance $d = |x - s|$. Weights are given by a kernel $f(d)$. Typically the kernel will be halfnormal $f(d) = \exp(-d^2/(2\sigma^2))$ (detectfn = 'HHN') or exponential $f(d) = \exp(-d/\sigma)$ (detectfn = 'HEX') (see detectfn for other possibilities).

If $z(x)$ represents the covariate value at point **x**, the summed resource availability at **s** is given by

$$R(s) = \sum_x f(d)\ \exp(\alpha_2\, z(x)).$$

This corresponds to the denominator of eqn 4 in Royle et al. (2013).

By default, the numerical values reported by Rsurface are not raw R values. If scale = TRUE, values are standardized by dividing by the mean: $R'(s) = R(s)/(\sum_s R(s)/n)$ where n is the number of pixels. Values of $R'(s)$ are centred on 1.0.

If inverse = TRUE, the numeric values are $1/R'(s)$ or $1/R(s)$ as determined by scale.

References

Royle, J. A., Chandler, R. B., Sun, C. C. and Fuller, A. K. (2013) Integrating resource selection information with spatial capture–recapture. *Methods in Ecology and Evolution* **4**, 520–530.

See Also

mask, plot.Rsurface, spotHeight, details

Examples

```
## create binary covariate (0 outside habitat)
msk <- make.mask(traps(possumCH), buffer = 800)
```

```
covariates(msk) <- data.frame(z = as.numeric(pointsInPolygon
    (msk,possumarea)))

## derive and plot "resource availability"
Rs <- Rsurface(msk, sigma = 100, usecov = 'z')
plot(Rs, plottype = 'contour', col = topo.colors(10))
lines(possumarea)

if (interactive()) {
    spotHeight(Rs, dec = 2)
}
```

---

score.test *Score Test for SECR Models*

Description

Compute score tests comparing a fitted model and a more general alternative model.

Usage

```
score.test(secr, ..., betaindex = NULL, trace = FALSE, ncores = NULL, .relStep = 0.001,
    minAbsPar = 0.1)

score.table(object, ..., sort = TRUE, dmax = 10)
```

Arguments

secr	fitted secr model
...	one or more alternative models OR a fitted secr model
trace	logical. If TRUE then output one-line summary at each evaluation of the likelihood
ncores	integer number of threads for parallel processing
.relStep	see fdHess
minAbsPar	see fdHess
betaindex	vector of indices mapping fitted values to parameters in the alternative model
object	score.test object or list of such objects
sort	logical for whether output rows should be in descending order of AIC
dmax	threshold of dAIC for inclusion in model set

Details

Score tests allow fast model selection (e.g. Catchpole & Morgan 1996). Only the simpler model need be fitted. This implementation uses the observed information matrix, which may sometimes mislead (Morgan et al. 2007). The gradient and second derivative of the likelihood function are evaluated numerically at the point in the parameter space of the second model corresponding to the fit of the first model. This operation uses the function `fdHess` of the **nlme** package; the likelihood must be evaluated several times, but many fewer times than would be needed to fit the model. The score statistic is an approximation to the likelihood ratio; this allows the difference in AIC to be estimated.

Covariates are inferred from components of the reference model `secr`. If the new models require additional covariates these may usually be added to the respective component of `secr`.

Mapping of parameters between the fitted and alternative models sometimes requires user intervention via the `betaindex` argument. For example `betaindex` = c(1,2,4) is the correct mapping when comparing the null model (D∼ 1, g0∼ 1, sigma∼ 1) to one with a behavioural effect on g0 (D∼ 1, g0∼ b, sigma∼ 1).

The arguments `.relStep` and `minAbsPar` control the numerical gradient calculation and are passed directly to `fdHess`. More investigation is needed to determine optimal settings.

`score.table` summarises one or more score tests in the form of a model comparison table. The ... argument here allows the inclusion of additional score test objects (note the meaning differs from `score.test`). Approximate AIC values are used to compute relative AIC model weights for all models within dmax AIC units of the best model.

If `ncores = NULL` then the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS is used (see `setNumThreads`).

Value

An object of class 'score.test' that inherits from 'htest', a list with components

`statistic`	the value the chi-squared test statistic (score statistic)
`parameter`	degrees of freedom of the approximate chi-squared distribution of the test statistic (difference in number of parameters H0, H1)
`p.value`	probability of test statistic assuming chi-square distribution
`method`	a character string indicating the type of test performed
`data.name`	character string with null hypothesis, alternative hypothesis and arguments to function call from fit of H0
`H0`	simpler model
`np0`	number of parameters in simpler model
`H1`	alternative model
`H1.beta`	coefficients of alternative model
`AIC`	Akaike's information criterion, approximated from score statistic
`AICc`	AIC with small-sample adjustment of Hurvich & Tsai 1989

If ... defines several alternative models then a list of score.test objects is returned.

The output from `score.table` is a dataframe with one row per model, including the reference model.

Note

This implementation is experimental. The AIC values, and values derived from them, are approximations that may differ considerably from AIC values obtained by fitting and comparing the respective models. Use of the observed information matrix may not be optimal.

Weights were based on AICc rather than AIC prior to version 5.0.0.

References

Catchpole, E. A. and Morgan, B. J. T. (1996) Model selection of ring-recovery models using score tests. *Biometrics* **52**, 664–672.

Hurvich, C. M. and Tsai, C. L. (1989) Regression and time series model selection in small samples. *Biometrika* **76**, 297–307.

McCrea, R. S. and Morgan, B. J. T. (2011) Multistate mark-recapture model selection using score tests. *Biometrics* **67**, 234–241.

Morgan, B. J. T., Palmer, K. J. and Ridout, M. S. (2007) Negative score test statistic. *American statistician* **61**, 285–288.

See Also

AIC, LR.test

Examples

```
## Not run:
    AIC (secrdemo.0, secrdemo.b)
    st <- score.test (secrdemo.0, g0 ~ b)
    st
    score.table(st)

    ## adding a time covariate to separate occasions (1,2) from (3,4,5)
    secrdemo.0$timecov <- data.frame(t2 = factor(c(1,1,2,2,2)))
    st2 <- score.test (secrdemo.0, g0 ~ t2)
    score.table(st,st2)

## End(Not run)
```

---

secr-defunct *Defunct Functions in Package* **secr**

Description

These functions are no longer available in **secr**.

Usage

```
# Defunct in 5.0.0

derivedSystematic()

# Defunct in 4.6.2 (2023-09-30)

model.average()

# Defunct in 4.5.10 (2023-03-10)

ip.secr()
pfn()

# Defunct in 4.4.2 (2021-05-04)

make.newdata()

# Defunct in 4.4.0 (2021-05-01)

secr.make.newdata()

# Defunct in 4.0.0 (2019-10-27)

read.SPACECAP()
write.SPACECAP()
```

Details

Some of these have stubs which report that they are defunct, but most have been removed completely (apart from being documented here).

`model.average` is replaced by a method for 'secr' and 'secrlist' objects of the generic `modelAverage`. The internal code is essentially the same for `model.average` and `modelAverage.secrlist`. The generic avoids a name conflict with RMark and is also used in **openCR**.

`ip.secr` and `pfn` have been superceded by `ipsecr.fit` and `proxy.ms` in package **ipsecr**.

Internal functions `secr.make.newdata` and `make.newdata` were replaced with `makeNewData` generic from 4.4.2.

SPACECAP was removed from the CRAN archive on 2019-08-31.

See Also

`secr-deprecated`

secr-deprecated *Deprecated Functions in Package* **secr**

Description

These functions will be removed from future versions of **secr** or have been renamed.

par.secr.fit (arglist, ncores = 1, seed = NULL, trace = TRUE, logfile = "logfile.txt", prefix = "fit.", LB = FALSE, save.intermediate = FALSE)

par.derived (secrlist, ncores = 1, ...)

par.region.N (secrlist, ncores = 1, ...)

fx.total (object, sessnum = 1, mask = NULL, ncores = NULL, ...)

fxi.secr (object, i = NULL, sessnum = 1, X = NULL, ncores = NULL, ...)

buffer.contour (traps, buffer, nx = 64, convex = FALSE, ntheta = 100, plt = TRUE, add = FALSE, poly = NULL, poly.habitat = TRUE, fill = NULL, ...)

pdot.contour (traps, border = NULL, nx = 64, detectfn = 0, detectpar = list(g0 = 0.2, sigma = 25, z = 1), noccasions = NULL, binomN = NULL, levels = seq(0.1, 0.9, 0.1), poly = NULL, poly.habitat = TRUE, plt = TRUE, add = FALSE, fill = NULL, ...)

esa.plot (object, max.buffer = NULL, spacing = NULL, max.mask = NULL, detectfn, detectpar, noccasions, binomN = NULL, thin = 0.1, poly = NULL, poly.habitat = TRUE, session = 1, plt = TRUE, type = c('density', 'esa', 'meanpdot', 'CVpdot'), n = 1, add = FALSE, overlay = TRUE, conditional = FALSE, ...)

fxi.contour (object, i = 1, sessnum = 1, border = 100, nx = 64, levels = NULL, p = seq(0.1,0.9,0.1), plt = TRUE, add = FALSE, fitmode = FALSE, plotmode = FALSE, fill = NULL, output = c('list','sf','SPDF'), ncores = NULL, ...)

fxi.mode (object, i = 1, sessnum = 1, start = NULL, ncores = NULL, ...)

Details

Since the introduction of multi-threading in **secr** 4.0 it is no longer efficient to use parallel worker processes.

`list.secr.fit` replaces `par.secr.fit`.

Functions par.derived and par.region.N can be replaced by a simple call of lapply (see Examples in `list.secr.fit`).

Some functions have been renamed to avoid the ambiguous ".":

Old	New
fxi.secr	`fxi` (S3 generic with method for secr objects)
fxi.contour	`fxiContour`
fxi.mode	`fxiMode`
fx.total	`fxTotal`
esa.plot	`esaPlot`
pdot.contour	`pdotContour`

buffer.contour bufferContour

See Also

secr-defunct, list.secr.fit, secr-version5

---

secr-version5 *Changes in* **secr** *5.0*

Description

This document explains changes in **secr** 5.0. Version 5.0 is compatible in most respects with earlier versions, but a few names and one default have been changed without warning. See the NEWS file for a complete list of the changes over time.

Potential Gotchas

New generics:

Several new generic functions are defined, with methods specifically for 'secr' fitted models (esa, fxi, fxTotal).

New names:

Some functions with "." in their name have been renamed to avoid confusion with methods for generics.

Where possible, the old names have been deprecated (tagged with a warning), and will continue to work for a while.

Old	New
buffer.contour	bufferContour
esa.plot	esaPlot
fxi.contour	fxiContour
fxi.mode	fxiMode
fxi.secr	fxi (generic)
fx.total	fxTotal (generic)
pdot.contour	pdotContour

fxi.secr has been replaced by the generic fxi. Thus instead of
fxi.secr(secrdemo.0, i = 1, X = c(365,605)) use
fxi(secrdemo.0, i = 1, X = c(365,605)).

New default:

AIC and related functions now default to criterion = "AIC" instead of criterion = "AICc".

Some of us have been uneasy for a long time about blanket use of the AICc small-sample adjustment to AIC (Hurvich and Tsai 1989). Royle et al. (2014) expressed doubts because the sample size itself is poorly defined. AICc is widely used, but AIC may be better for model averaging even when samples are small (Turek and Fletcher 2012; Fletcher 2019, p. 60).

New features

New data `blackbearCH`**:**

secr 5.0 includes a new black bear DNA hair snag dataset from the Great Smoky Mountains, Tennessee (thanks to J. Laufenberg, F. van Manen and J. Clark).

Goodness-of-fit `MCgof`**:**

The method of Choo et al. (2024) for emulating the Bayesian p-value goodness-of-fit test (Gelman 1996, Royle et al. 2014) has been implemented as the generic `MCgof` with a method for 'secr' fitted models. I thank Yan Ru Choo for his assistance.

This is a new approach and should be used with caution. Bugs may yet be found, and the power of the tests is limited.

Extended capability

These extensions allow `MCgof` to cover a wider range of models:

- `detectpar` optionally returns values for each detector
- `pdot` accepts detector- and occasion-specific detection parameters

Changes behind the scenes

The code for area-search and transect-search models (detector types 'polygonX', 'polygon', 'transectX', 'transect') has been streamlined with a view to removing it to another package. Simulation for these models (functions `sim.capthist`, `sim.detect`) will remain in **secr**, but uses native R functions rather than RcppNumerical of Qiu et al. (2023).

The undocumented detection function 'HPX' has been removed.

References

Choo, Y. R., Sutherland, C. and Johnston, A. (2024) A Monte Carlo resampling framework for implementing goodness-of-fit tests in spatial capture-recapture models. *Methods in Ecology and Evolution* **15**, 1653–1666. doi:10.1111/2041210X.14386.

Efford, M. G. (2024) secr: Spatially explicit capture-recapture models. R package version 5.0.0. https://CRAN.R-project.org/package=secr

Fletcher, D. (2019) *Model averaging*. SpringerBriefs in Statistics. Berlin: Springer-Verlag.

Hurvich, C. M. and Tsai, C. L. (1989) Regression and time series model selection in small samples. *Biometrika* **76**, 297–307.

Gelman, A., Meng, X.-L., and Stern, H. (1996) Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica* **6**, 733–807.

Qiu, Y., Balan, S., Beall, M., Sauder, M., Okazaki, N. and Hahn, T. (2023) RcppNumerical: 'Rcpp' Integration for Numerical Computing Libraries. R package version 0.6-0. https://CRAN.R-project.org/package=RcppNumerical

Royle, J. A., Chandler, R. B., Sollmann, R. and Gardner, B. (2014) *Spatial capture–recapture*. Academic Press.

Turek, D. and Fletcher, D. (2012) Model-averaged Wald confidence intervals. *Computational statistics and data analysis* **56**, 2809–2815. doi:10.1002/ecs2.3385

See Also

secr-deprecated, secr-defunct

---

secr.design.MS *Construct Detection Model Design Matrices and Lookups*

Description

Internal functions used by secr.fit.

Usage

```
secr.design.MS (capthist, models, timecov = NULL, sessioncov = NULL,
    groups = NULL, hcov = NULL, dframe = NULL, naive = FALSE, CL = FALSE,
   keep.dframe = FALSE, full.dframe = FALSE, ignoreusage = FALSE, contrasts = NULL, ...)

make.lookup (tempmat)

insertdim (x, dimx, dims)
```

Arguments

capthist	capthist object
models	list of formulae for parameters of detection
timecov	optional dataframe of values of time (occasion-specific) covariate(s).
sessioncov	optional dataframe of values of session-specific covariate(s).
groups	optional vector of one or more variables with which to form groups. Each element should be the name of a factor variable in the covariates attribute of capthist.
hcov	character name of an individual (capthist) covariate for known class membership in h2 models
dframe	optional data frame of design data for detection parameters
naive	logical if TRUE then modelled detection probability is for a naive animal (not caught previously); if FALSE then detection probability is contingent on individual's history of detection
CL	logical; TRUE for model to be fitted by maximizing the conditional likelihood
keep.dframe	logical; if TRUE the dataframe of design data is included in the output
full.dframe	logical; if FALSE then padding rows are purged from output dframe (ignored if keep.dframe = FALSE)
ignoreusage	logical; if TRUE any usage attribute of traps(capthist) is ignored
contrasts	contrast specification as for model.matrix
...	other arguments passed to the R function model.matrix

tempmat	matrix for which row lookup required
x	vector of character, numeric or factor values
dimx	vector of notional dimensions for x to fill in target array
dims	vector of notional dimensions of target array

Details

These are internal **secr** functions that you are unlikely ever to use.

Each real parameter is notionally different for each unique combination of session, individual, occasion, detector and latent class, i.e., for R sessions, n individuals, S occasions, K detectors and M latent classes there are *potentially* $R \times n \times S \times K \times M$ different values. Actual models always predict a *much* reduced set of distinct values, and the number of rows in the design matrix is reduced correspondingly; a parameter index array allows these to retrieved for any combination of session, individual, occasion and detector.

The `keep.dframe` option is provided for the rare occasions that a user may want to check the data frame that is an intermediate step in computing each design matrix with `model.matrix` (i.e. the data argument of `model.matrix`).

... may be used to pass `contrasts.arg` to `model.matrix`.

Value

For `secr.design.MS`, a list with the components

designMatrices	list of reduced design matrices, one for each real detection parameter
parameterTable	index to row of the reduced design matrix for each real detection parameter; dim(parameterTable) = c(uniquepar, np), where uniquepar is the number of unique combinations of paramater values (uniquepar < $RnSKM$) and np is the number of parameters in the detection model.
PIA	Parameter Index Array - index to row of parameterTable for a given session, animal, occasion and detector; dim(PIA) = c(R,n,S,K,M)
R	number of sessions

If `models` is empty then all components are NULL except for PIA which is an array of 1's (M set to 1).

Optionally (`keep.dframe = TRUE`) -

dframe	dataframe of design data, one column per covariate, one row for each c(R,n,S,K,M). For multi-session models n, S, and K refer to the maximum across sessions
validdim	list giving the valid dimensions (n, S, K, M) before padding

For `make.lookup`, a list with components

lookup	matrix of unique rows
index	indices in lookup of the original rows

For `insertdim`, a vector with length prod(dims) containing the values replicated according to dimx.

See Also

D.designdata, model.matrix

Examples

```
secr.design.MS (captdata, models = list(g0 = ~b))$designMatrices
secr.design.MS (captdata, models = list(g0 = ~b))$parameterTable

## peek at design data constructed for learned response model
head(captdata)
temp <- secr.design.MS (captdata, models = list(g0 = ~b),
    keep.dframe = TRUE)
a1 <- temp$dframe$animal == 1 &  temp$dframe$detector %in% 8:10
temp$dframe[a1,]

## ... and trap specific learned response model
temp <- secr.design.MS (captdata, models = list(g0 = ~bk),
    keep.dframe = TRUE)
a1 <- temp$dframe$animal == 1 &  temp$dframe$detector %in% 8:10
temp$dframe[a1,]

## place values 1:6 in different dimensions
insertdim(1:6, 1:2, c(2,3,6))
insertdim(1:6, 3, c(2,3,6))
```

---

secr.fit *Spatially Explicit Capture–Recapture*

Description

Estimate animal population density with data from an array of passive detectors (traps) by fitting a spatial detection model by maximizing the likelihood. Data must have been assembled as an object of class capthist. Integration is by summation over the grid of points in mask.

Usage

```
secr.fit (capthist, model = list(), mask = NULL, buffer = NULL,
    CL = FALSE, detectfn = NULL, binomN = NULL, start = NULL, link = list(),
    fixed = list(), timecov = NULL, sessioncov = NULL, hcov = NULL,
    groups = NULL, dframe = NULL, details = list(), method =
    "Newton-Raphson", verify = TRUE, biasLimit = 0.01, trace = NULL,
    ncores = NULL, ...)
```

Arguments

capthist	capthist object including capture data and detector (trap) layout
model	list with optional components each symbolically defining a linear predictor for one real parameter using formula notation
mask	mask object or (for a multi-session analysis) a list of mask objects, one for each session
buffer	scalar mask buffer radius if mask not specified (default 100 m)
CL	logical, if true then the model is fitted by maximizing the conditional likelihood
detectfn	integer code or character string for shape of detection function 0 = halfnormal, 1 = hazard rate etc. – see detectfn
binomN	integer code for distribution of counts (see Details)
start	vector of initial values for beta parameters, or secr object from which they may be derived
link	list with optional components corresponding to 'real' parameters (e.g., 'D', 'g0', 'sigma'), each a character string in {"log", "logit", "identity", "sin"} for the link function of one real parameter
fixed	list with optional components corresponding to real parameters giving the scalar value to which the parameter is to be fixed
timecov	optional dataframe of values of time (occasion-specific) covariate(s).
sessioncov	optional dataframe of values of session-specific covariate(s).
hcov	character name of individual covariate for known membership of mixture classes.
groups	optional vector of one or more variables with which to form groups. Each element should be the name of a factor variable in the covariates attribute of capthist. See Note.
dframe	optional data frame of design data for detection parameters
details	list of additional settings, mostly model-specific (see Details)
method	character string giving method for maximizing log likelihood
verify	logical, if TRUE the input data are checked with verify
biasLimit	numeric threshold for predicted relative bias due to buffer being too small
trace	logical, if TRUE then output each evaluation of the likelihood, and other messages
ncores	integer number of threads to use for parallel processing
...	other arguments passed to the maximization function

Details

secr.fit fits a SECR model by maximizing the likelihood. The likelihood depends on the detector type ("multi", "proximity", "count", "polygon" etc.) of the traps attribute of capthist (Borchers and Efford 2008, Efford, Borchers and Byrom 2009, Efford, Dawson and Borchers 2009, Efford 2011). The 'multi' form of the likelihood is also used, with a warning, when detector type = "single" (see Efford et al. 2009 for justification).

The default model is null (model = list(D~1, g0~1,sigma~1) for detectfn = 'HN' and CL = FALSE), meaning constant density and detection probability). The set of variables available for use in linear predictors includes some that are constructed automatically (t, T, b, B, bk, Bk, k, K), group (g), and others that appear in the covariates of the input data. See also usage for varying effort, timevaryingcov to construct other time-varying detector covariates, and secr-models.pdf and secr-overview.pdf for more on defining models.

buffer and mask are alternative ways to define the region of integration (see mask). If mask is not specified then a mask of type "trapbuffer" will be constructed automatically using the specified buffer width in metres.

hcov is used to define a hybrid mixture model, used especially to model sex differences (see hcov). (Allows some animals to be of unknown class).

The length of timecov should equal the number of sampling occasions (ncol(capthist)). Arguments timecov, sessioncov and groups are used only when needed for terms in one of the model specifications. Default link is list(D="log",g0="logit", sigma="log").

If start is missing then autoini is used for D, g0 and sigma, and other beta parameters are set initially to arbitrary values, mostly zero. start may be a previously fitted model. In this case, a vector of starting beta values is constructed from the old (usually nested) model and additional betas are set to zero. Mapping of parameters follows the default in score.test, but user intervention is not allowed. From 2.10.0 the new and old models need not share all the same 'real' parameters, but any new real parameters, such as 'pmix' for finite mixture models, receive a starting value of 0 on the link scale (remembering e.g., invlogit(0) = 0.5 for parameter 'pmix').

binomN (previously a component of details) determines the distribution that is fitted for the number of detections of an individual at a particular detector, on a particular occasion, when the detectors are of type 'count', 'polygon' or 'transect':

- binomN > 1 — binomial with size binomN
- binomN = 1 — binomial with size determined by usage
- binomN = 0 — Poisson

The default with these detectors is to fit a Poisson distribution.

details is used for various specialized settings. An incomplete list follows; see details for the full list and description.

autoini	session to use for starting values (default 1)
centred	centre x-y coordinates
chat	overdispersion of sighting counts Tu, Tm (and shape, for 'marking')
chatmethod	"simulate" (default) or "analytic"
chatonly	compute overdispersion for Tu and Tm, then exit
contrasts	coding of factor predictors
convexpolygon	allows non-convex polygons (slower)
Dlambda	switch density reparameterization to trend model
distribution	binomial vs Poisson N
externalqx	name of mask covariate as offset for relative density
fastproximity	special handling of binary proximity detectors
fixedbeta	specify fixed beta parameter(s)
grain	grain argument of RcppParallel::parallelFor

hessian	variance method
ignoreusage	override usage in traps object of capthist
knownmarks	known or unknown number of marked animals in sighting-only model
LLonly	compute one likelihood for values in `start`
maxdistance	distance threshold for selective mask
miscparm	starting values for extra parameters fitted via userdist function
newdetector	detector type to override detector(traps(capthist))
nsim	number of simulations to compute mark-resight overdispersion
param	optional parameterisation code
safeLL	log-sum-exp algorithm to avoid underflow
savecall	optionally suppress saving of call
saveprogress	save results to RDS file at given frequency
telemetryint	method for integration over activity centres of telemetered animals
uselog	log-sum likelihood to avoid underflow, mainly telemetry
userdist	user-provided distance function or matrix

Setting `ncores = NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

A mark-resight model is fitted if the `markocc` attribute of the capthist 'traps' object includes sighting occasions. See the vignette secr-markresight.pdf for a full account.

If `method = "Newton-Raphson"` then `nlm` is used to maximize the log likelihood (minimize the negative log likelihood); otherwise `optim` is used with the chosen method ("BFGS", "Nelder-Mead", etc.). If maximization fails a warning is given appropriate to the method.

`method = "none"` may be used to skip likelihood maximization and compute only the hessian for the current dataset at the values in start, and the corresponding variance-covariance matrix of beta parameters. The computation uses fdHess from **nlme**. See also `secr.refit` (secr >= 5.2.1).

If `verify` = TRUE then `verify` is called to check capthist and mask; analysis is aborted if "errors" are found. Some conditions that trigger an "error" are benign (e.g., no detections in some sessions of a multi-session study of a sparse population); use `verify = FALSE` to avoid the check. See also Note.

If `buffer` is used rather than `mask`, and `biasLimit` is valid, then the estimated density is checked for bias due to the choice of buffer. A warning is generated when `buffer` appears to be too small (predicted RB(D-hat) > `biasLimit`, default 1% relative bias). The prediction uses `bias.D`. No check is performed when `mask` is specified, when `biasLimit` is 0, negative or NA, or when the detector type is "polygon", "transect", "polygonX" or "transectX".

Function `list.secr.fit` is a way to fit several models at once.

Value

When `details$LLonly = TRUE` a single log-likelihood is returned, with attributes

`npar`	number of parameters to be estimated,
`preptime`	elapsed setup time in seconds,
`LLtime`	elapsed time for single likelihood evaluation, exclusive of setup.

Otherwise, `secr.fit` returns an object of class secr representing the fitted SECR model. This has components

`call`	function call)
`capthist`	saved input
`mask`	saved input
`detectfn`	saved input
`CL`	saved input
`timecov`	saved input
`sessioncov`	saved input
`hcov`	saved input
`groups`	saved input
`dframe`	saved input
`designD`	design matrix for density model; may be NULL
`designNE`	design matrix for noneuc model; may be NULL
`design`	reduced design matrices for detection parameters, parameter table and parameter index array for actual animals (see `secr.design.MS`)
`design0`	reduced design matrices for detection parameters, parameter table and parameter index array for 'naive' animal (see `secr.design.MS`)
`start`	vector of starting values for beta parameters
`link`	list with one component for each real parameter (typically 'D', 'g0', 'sigma'),giving the name of the link function used for each real parameter.
`fixed`	saved input
`parindx`	list with one component for each real parameter giving the indices of the 'beta' parameters associated with each real parameter
`model`	saved input
`details`	saved input
`vars`	vector of unique variable names in `model`
`betanames`	names of beta parameters
`realnames`	names of fitted (real) parameters
`fit`	list describing the fit (output from `nlm` or `optim`)
`beta.vcv`	variance-covariance matrix of beta parameters
`smoothsetup`	list of objects specifying smooths in **mgcv**
`learnedresponse`	logical; TRUE if any learned response in detection model
`version`	secr version number
`starttime`	character string of date and time at start of fit
`proctime`	processor time for model fit, in seconds

The environment variable RCPP_PARALLEL_NUM_THREADS is updated if an integer value is provided for `ncores`.

Warning

** Mark-resight data formats and models are experimental in secr 2.10.0 and subject to change **

Note

One system of units is used throughout **secr**. Distances are in metres and areas are in hectares (ha). The unit of density is animals per hectare. 1 ha = 10000 m^2 = 0.01 km^2. To convert density to animals / km^2, multiply by 100.

When you display an 'secr' object by typing its name at the command prompt, you implicitly call its 'print' method `print.secr`, which in turn calls `predict.secr` to tabulate estimates of the 'real' parameters. Confidence limits (lcl, ucl) are for a 100(1-alpha)% interval, where alpha defaults to 0.05 (95% interval); alpha may be varied in `print.secr` or `predict.secr`.

`AIC`, `logLik` and `vcov` methods are also provided. Take care with using AIC: not all models are comparable (see Notes section of `AIC.secr`) and large differences in AIC may relate to trivial differences in estimated density.

`derived` is used to compute the derived parameters 'esa' (effective sampling area) and 'D' (density) for models fitted by maximizing the conditional likelihood (CL = TRUE).

Components 'version' and 'starttime' were introduced in version 1.2.7, and recording of the completion time in 'fitted' was discontinued.

The Newton-Raphson algorithm is fast, but it sometimes fails to compute the information matrix correctly, causing some or all standard errors to be set to NA. This usually indicates a major problem in fitting the model, and parameter estimates should not be trusted. See Troubleshooting.

The component D in output was replaced with N from version 2.3. Use `region.N` to obtain SE or confidence intervals for N-hat, or to infer N for a different region.

Prior to version 2.3.2 the buffer bias check could be switched off by setting `verify = FALSE`. This is now done by setting `biasLimit = 0` or `biasLimit = NA` .

Groups defined by the 'interaction' of multiple individual covariates gave unreliable results before version 5.3.0. See https://github.com/MurrayEfford/secr/issues/9.

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. (2004) Density estimation in live-trapping studies. *Oikos* **106**, 598–610.

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture–recapture: likelihood-based methods. In: D. L. Thompson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer. Pp. 255–269.

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.

See Also

Detection functions, `AIC.secr`, `capthist`, `details`, `derived`, `hcov`, `mask`, `list.secr.fit`, `predict.secr`, `print.secr`, `region.N`, `secr.refit`, Speed tips Troubleshooting userdist usage, `vcov.secr`, `verify`,

Examples

```
## Not run:

## construct test data (array of 48 `multi-catch' traps)

detectors <- make.grid (nx = 6, ny = 8, detector = "multi")
detections <- sim.capthist (detectors, popn = list(D = 10,
    buffer = 100), detectpar = list(g0 = 0.2, sigma = 25))

## fit & print null (constant parameter) model
secr0 <- secr.fit (detections)
secr0   ## uses print method for secr

## compare fit of null model with learned-response model for g0

secrb <- secr.fit (detections, model = g0~b)
AIC (secr0, secrb)

## typical result

##                  model   detectfn npar    logLik     AIC    AICc dAIC   AICwt
## secr0 D~1 g0~1 sigma~1 halfnormal    3 -347.1210 700.242 700.928 0.000 0.7733
## secrb D~1 g0~b sigma~1 halfnormal    4 -347.1026 702.205 703.382 2.454 0.2267

## End(Not run)
```

---

secr.refit *Repeat or Resume Model Fit*

Description

From 5.2.1 it is possible to save a part-fitted model to a file and resume fitting with secr.refit. The function may also be used to re-fit a completed model, or to repeat just the variance estimation (Hessian) step.

Usage

```
secr.refit(object, ...)
```

Arguments

object	secr object (a fitted model) or character value naming a saved progress RDS file (e.g. progress.RDS)
...	arguments of secr.fit to override those in object

Details

There are two common uses for the function.

1. To resume fitting of a partly fitted model from an RDS file saved with `saveprogress`.

2. To recompute variances of a fitted model. The numerical method is fdHess from package **nlme**. Often this delivers variance estimates when the default algorithm failed.

The ... argument is used to override previous settings (arguments of `secr.fit`); commonly method = "none" to recompute variances. A new 'details' argument may be incomplete: specified components replace those in the original (see example resetting 'saveprogress' below).

The 'start' argument may not be overridden: it is set by `secr.refit` from the previous fit. The 'binomN' argument is elevated automatically from details$binomN if required.

A third use of `secr.refit` is to vary the model structure. This requires that 'object' is a fitted model. A change in model structure is implied when the ... argument replaces any of 'model', 'CL', 'detectfn', 'link', 'fixed', 'hcov', or 'groups'. New starting values are then determined in `secr.fit` by a call to `makeStart`, and may not precisely follow the previous fit.

Also, `secr.refit` will compute a full model from a relative density model (i.e. estimate the intercept) - just set CL = FALSE. If method = "none" then new variances are computed using fdHess. Note this does *not* work if the model includes individual covariates, which are prohibited in a full-likelihood model.

Value

A fitted secr model.

Note

In the case of a relative density model (i.e. a model was specified for D when CL = TRUE), a warning will be generated "overriding provided fixedbeta[1] for D". This can be ignored.

See Also

`secr.fit`, saveprogress

Examples

```
# Repeat variance step using nlme::fdHess
# method = "none" skips likelihood maximization
secr.refit(secrdemo.0, method = "none")

# Save data and coefficients after every 10 evaluations to a file
# with the default name 'progress.RDS'.
fit1 <- secr.fit(captdata, CL = TRUE, details = list(saveprogress = 10))

# Review progress. In this example, fit1 ran to completion,
# but the log stops at a multiple of 10 evaluations.
attr(readRDS('progress.RDS'), 'log')
```

```
# Restart at last line in log, suppressing overwrite of progress log
secr.refit('progress.RDS', details = list(saveprogress = FALSE))

# cleanup
unlink("progress.RDS")

# Example using secr.refit() to complete a relative density model
# fitCL <- secr.fit(OVpossumCH[[1]], buffer = 120, detectfn = 'HEX',
#    CL = TRUE, model = D~x+y)
# fit <- secr.refit(fitCL, CL = FALSE, method = "none")
# coef(fitCL)
# coef(fit)
# predict(fit)
```

---

secr.test *Goodness-of-Fit Test*

Description

Simple Monte-Carlo goodness-of-fit tests for full-likelihood SECR models. The approach is to calculate a statistic from either the raw data or a fitted model, and to relate this to the distribution of the statistic under the original fitted model. The distribution is estimated by simulating data from the model, and possibly re-fitting the model to each simulated dataset.

The suitability of different test statistics has yet to be assessed. An alternative and possibly more informative set of tests is provided in function MCgof that implements the approach of Choo et al. (2024) for a range of models.

Usage

```
secr.test(object, nsim = 99, statfn, fit = FALSE, seed =
NULL, ncores = NULL, tracelevel = 1, ...)
```

Arguments

object	a fitted secr model
nsim	integer number of replicates
statfn	function to compute a numeric vector of one or more statistics from a single-session 'capthist' object or from a fitted model (see Details)
fit	logical; if TRUE the model is re-fitted to each simulated dataset
seed	either NULL or an integer that will be used in a call to set.seed
ncores	integer number of threads for parallel processing
tracelevel	see sim.secr
...	other arguments passed to statfn, if needed

Details

The test statistic(s) may be computed either on a dataset or on a fitted model, as determined by the argument `fit`. The single argument expected by `statfn` should be either a 'capthist' object (`fit = FALSE`) or an 'secr' object (`fit = TRUE`).

The default statistic when `fit = FALSE` is the proportion of individuals observed on only one occasion, which is equivalent to `statfn = function(CH) c(f1 = sum(apply(abs(CH) > 0,1,sum) == 1) / nrow(CH))`. Repeat detections on one occasion at the same or different detectors are not counted. The default statistic is therefore not appropriate for some data, specifically from 'count' or 'polygon' detectors with few occasions or only one.

The default statistic when `fit = TRUE` is the deviance divided by the residual degrees of freedom (i.e., `statfn = function(object) c(devdf = deviance(object) / df.residual(object))`).

The reported probability (p) is the rank of the observed value in the vector combining the observed value and simulated values, divided by (nsim + 1). Ranks are computed with `rank` using the default `ties.method = "average"`.

Simulations take account of the usage attribute of detectors in the original capthist object, given that usage was defined and ignoreusage was not set.

Setting `ncores = NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

`statfn` may return a vector of statistics for each observed or simulated dataset or model: then the value of each statistic will be calculated for every simulated dataset, and summarised. If `fit = TRUE` the vector of statistics may include both those computed on the raw data (object$capthist) and the fitted model.

Value

An object of class `c('secrtest','list')` with components:

`object`	as input
`nsim`	as input
`statfn`	as input
`fit`	as input
`seed`	as input
`output`	list comprising the simulated values, the observed value(s), and estimated probabilities

For multi-session input when `fit = FALSE`, 'output' is a list in which each session provides one component.

Print and plot methods are provided for 'secrtest' objects.

Note

`simulate.secr` is always used to simulate the raw data, but `simulate.secr` does not work for all types of fitted model. Models fitted by maximizing the likelihood conditional on n (`CL = TRUE` in `secr.fit`) potentially include individual covariates whose distribution in the population is unknown. This precludes simulation, and conditional-likelihood models in general are therefore not covered by `secr.test`.

Other exclusions include exotic non-binary behavioural responses ("bn", "bkn", "bkc", "Bkc" - but these are generally undocumented in any case).

If `fit = TRUE` then `sim.secr` is used.

At each simulation a new population is generated across the extent of the original mask. If the extent is unduly large then time will be wasted simulating the possibility of detection for many essentially undetectable animals. This is an argument for keeping the mask tight - large enough only to avoid mask-induced bias.

References

Choo, Y. R., Sutherland, C. and Johnston, A. (2024) A Monte Carlo resampling framework for implementing goodness-of-fit tests in spatial capture-recapture model *Methods in Ecology and Evolution* **15**, 1653–1666. doi:10.1111/2041210X.14386.

See Also

`MCgof`, `print.secrtest`, `plot.secrtest`, `simulate.secr`, `sim.secr`, `deviance.secr`

Examples

```
## Not run:

secr.test(secrdemo.0, nsim = 99)

secr.test(ovenbird.model.1, nsim = 20)

## example combining raw data summary and model fit
## assumes single-session
bothfn <- function(object) {
    CH <- object$capthist
    f1 <- sum(apply(abs(CH) > 0, 1, sum) == 1) / nrow(CH)
    devdf <- deviance(object) / df.residual(object)
    c(f1 = f1, devdf = devdf)
}
test <- secr.test (secrdemo.0, nsim = 19, statfn = bothfn, fit = TRUE)
test
plot(test, main = '')


## End(Not run)
```

`secrdemo` *SECR Models Fitted to Demonstration Data*

Description

Demonstration data from program Density are provided as text files in the 'extdata' folder, as raw dataframes (`trapXY`, `captXY`), and as a combined `capthist` object (`captdata`) ready for input to `secr.fit`.

The fitted models are objects of class `secr` formed by

```
secrdemo.0 <- secr.fit (captdata)
secrdemo.b <- secr.fit (captdata, model = list(g0 = ~b))
secrdemo.CL <- secr.fit (captdata, CL = TRUE)
```

Usage

```
data(secrdemo)
```

Details

The raw data are 235 fictional captures of 76 animals over 5 occasions in 100 single-catch traps 30 metres apart on a square grid with origin at (365,365).

Dataframe `trapXY` contains the data from the Density input file 'trap.txt', and `captXY` contains the data from 'capt.txt' (Efford 2012).

The fitted models use a halfnormal detection function and the likelihood for multi-catch traps (expect estimates of g0 to be biased because of trap saturation Efford et al. 2009). The first is a null model (i.e. parameters constant) and the second fits a learned trap response.

Object	Description
captXY	data.frame of capture data
trapXY	data.frame of trap locations
captdata	capthist object
secrdemo.0	fitted secr model – null
secrdemo.b	fitted secr model – g0 trap response
secrdemo.CL	fitted secr model – null, conditional likelihood

References

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

See Also

`capthist`, `read.capthist`

Examples

```
## Not run:

## navigate to folder with raw data files
olddir <- setwd (system.file("extdata", package="secr"))

## construct capthist object from raw data
captdata  <- read.capthist ("capt.txt", "trap.txt", fmt = "XY", detector = "single")

## generate demonstration fits
secrdemo.0 <- secr.fit (captdata)
secrdemo.CL <- secr.fit (captdata, CL = TRUE)
secrdemo.b <- secr.fit (captdata, model = list(g0 ~ b))

## restore previous setting
setwd(olddir)

## End(Not run)

## display the null model fit, using the print method for secr
secrdemo.0

## compare fit of models
AIC(secrdemo.0, secrdemo.b)

## display estimates for the two models (single session)
collate(secrdemo.0, secrdemo.b)[1,,,]
```

---

secrRNG *Random Number Seed*

Description

The use of random number seeds in **secr** is explained.

Random numbers in R

R provides several kinds of random number generator (RNG) in the **base** package (see RNG). These are used both explicitly, in functions such as `runif` and `rnorm`, and implicitly (`sample`).

A seed suitable for any kind of RNG is held in a vector of 626 integers named `.Random.seed`. The vector is not to be modified directly by users. Instead, to start a reproducible stream of random numbers, the user calls `set.seed` with a single non-null integer argument. This has the effect of initialising `.Random.seed`. The value of `.Random.seed` may nevertheless be stored and restored to reset the RNG state.

`set.seed` with a NULL argument initialises `.Random.seed` to an indeterminate (time- and process-dependent) value. The same happens if a random number function is called before `.Random.seed` has been set.

Handling of RNG seed for simulation in package stats

The 'official' approach to setting and storing the RNG seed is shown in code and documentation for the generic function `simulate` in the **stats** package.

- The generic has argument 'seed' with default NULL.
- If 'seed' is non-null then `set.seed` is called.
- The returned value has an attribute "seed" whose value is either (i) if specified, the integer value of the 'seed' argument (with its own attribute "kind" from RNGkind), or (ii) the original vector `.Random.seed`.
- On exit the RNG state in `.Random.seed` is reset to the value that applied when the function was called.

For NULL seed input, the saved RNGstate may be used to reset .Random.seed (see Examples).

Use of random numbers in secr

Many functions in **secr** call on random numbers, sometimes in unexpected places. For example `autoini` selects a random sample to thin points and speed computation. In most functions there is no provision for direct control of the random number state: users won't usually care, and if they do then `set.seed` may be called for the particular R session.

However, control of the RNG seed is required for reproducible data generation in simulation functions. These functions typically have a 'seed' argument that is used internally in a call to `set.seed`. Handling of seeds in the simulation functions of **secr** largely follows `stats::simulate` as described in the preceding section.

The relevant functions are –

Function	Default	Saved attribute	Note
`randomHabitat`	NULL	seed or RNGstate	
`secr.test`	NULL	seed or RNGstate	calls and retains seed from `simulate.secr`
`sim.capthist`	NULL	seed or RNGstate	
`sim.resight`	NULL	seed or RNGstate	seed may be passed in ... argument
`sim.popn`	NULL	seed or RNGstate	
`sim.secr`	NULL	seed or RNGstate	
`simulate.secr`	NULL	seed or RNGstate	S3 method called by `sim.secr`

Setting `seed = NULL` in any of these functions has the effect of continuing the existing random number stream; it is not the same as calling `set.seed(NULL)`.

Parallel processing

Two models are used for parallel processing in **secr**, corresponding to multi-threading with package **RcppParallel** (e.g. `secr.fit`) and parallel cores in package **parallel** (e.g. `chat.nk`).

In the **parallel** model the L'Ecuyer pseudorandom generator is used to provide a separate random number stream for each core (see `clusterSetRNGStream`).

When using **Rcpp** the state of the random number generator is set in C++ with the call `RNGScope scope;` that automatically resets the state of the generator on exit (Eddelbuettel 2013 p. 115).

Random number streams in separate **RcppParallel** threads are (probably) not independent. Thus there are potential issues with RNG calls in multi-threaded code. However, in **secr** 4.0 all RNG calls in C++ code are outside multi-threaded contexts, with the exception of simulations allowing for overdispersion in mark–resight estimates (Rcpp exported function sightingchatcpp). The implications for mark-resight estimates have not been explored, and it is unclear whether more elaborate solutions are needed.

References

Eddelbuettel, D. 2013. Seamless R and C++ integration with Rcpp. Springer.

See Also

Parallel, set.seed, simulate, sim.capthist, sim.popn, sim.resight, secr.test, simulate.secr

Examples

```
## Not run:

lmfit <- lm(speed ~ dist, data = cars)

## 1. NULL seed
r1 <- simulate(lmfit, seed = NULL)
r2 <- simulate(lmfit, seed = NULL)
## restore RNGstate, assuming RNGkind unchanged
.Random.seed <- attr(r1, "seed")
r3 <- simulate(lmfit, seed = NULL)
r1[1:6,1]
r2[1:6,1]
r3[1:6,1]

## 2. explicit seed
r4 <- simulate(lmfit, seed = 123)
r5 <- simulate(lmfit, seed = attr(r4, "seed"))
r4[1:6,1]
r5[1:6,1]


## End(Not run)
```

---

secrtest *Goodness-of-fit Test Results*

Description

S3 class for results from secr.test.

Usage

```
## S3 method for class 'secrtest'
print(x, terse = TRUE, ...)
## S3 method for class 'secrtest'
plot(x, stat, ...)
```

Arguments

x	secrtest object from secr.test
terse	logical; if TRUE only p values are displayed
stat	character; names of statistics to plot (default: all)
...	other arguments passed to hist by plot.secrtest

Details

An 'secrtest' object is output from secr.test.

plot.secrtest plots a histogram of the simulated values.

If plot.secrtest is applied to an object with more than one statistic then multiple plots are produced, so a multi-figure layout should be prepared (par(mfrow = c(1,2)) for 2 plots side by side). Include the hist argument main = '' to suppress the ugly plot labels, and ensure each statistic is named by statfn so that the x-axis is labelled correctly (See the Examples in help for secr.test).

See Also

secr.test

Examples

```
## Not run:

tmp <- secr.test(ovenbird.model.1)
if (inherits(tmp, 'secrtest')) {
    tmp  ## terse print
    print(tmp, terse = FALSE)
    par(mfrow = c(1,5))
    plot(tmp, main = '', xlim=c(0,1), breaks=seq(0,1,0.05))
    par(mfrow = c(1,1))    ## reset to default
}


## End(Not run)
```

session *Session Vector*

Description

Extract or replace the session names of a `capthist` object.

Usage

```
session(object, ...)
session(object) <- value
```

Arguments

`object`	object with 'session' attribute e.g. `capthist`
`value`	character vector or vector that may be coerced to character, one value per session
`...`	other arguments (not used)

Details

Replacement values will be coerced to character.

Value

a character vector with one value for each session in `capthist`

Note

Like Density, **secr** uses the term 'session' for a closed-population sample. A session usually includes data from several closely-spaced capture occasions (often consecutive days). Each 'primary session' in the 'robust' design of Pollock (1982) would be treated as a session in **secr**. **secr** also uses 'session' for independent subsets of the capture data distinguished by characteristics other than sampling time (as above). For example, two grids trapped simultaneously could be analysed as distinct sessions if (i) they were far enough apart that there was negligible prospect of the same animal being caught on both grids, and (ii) there was interest in comparing estimates from the two grids, or fitting a common detection model.

The log likelihood for a session model is the sum of the separate session log likelihoods. Although this assumes independence of sampling, parameters may be shared across sessions, or session-specific parameter values may be functions of session-level covariates. For many purposes, 'sessions' are equivalent to 'groups'. For multi-session models the detector array and mask are specified separately for each session. Group models are therefore generally simpler to implement. On the other hand, sessions offer more flexibility in defining and evaluating between-session models, including trend models.

References

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

See Also

capthist

Examples

```
session(captdata)
```

---

setNumThreads *Number of Threads*

Description

Set or report the number of cores to be used for multi-threaded operations. A wrapper for the RcppParallel function setThreadOptions (Allaire et al. 2019).

Usage

```
setNumThreads(ncores, ...)
```

Arguments

ncores	integer number of threads to use
...	other arguments passed to RcppParallel::setThreadOptions, specifically stackSize

Details

If ncores is NULL then the current value of the environment variable RCPP_PARALLEL_NUM_THREADS is used. RCPP_PARALLEL_NUM_THREADS defaults to 2 at the start of a session (assuming at least 2 logical cores available).

Calling setNumThreads() with no arguments is a handy way to check how many threads are in use.

The value of RCPP_PARALLEL_NUM_THREADS is also reset when a multi-threaded function such as secr.fit is called with a non-NULL value of the ncores argument. This value applies in later calls of secr.fit with ncores = NULL until changed.

Value

The new value of the environment variable RCPP_PARALLEL_NUM_THREADS.

Note

The mechanism for setting the number of threads changed between versions 4.1.0 and 4.2.0. The default number of cores is now capped at 2 to meet CRAN requirements. Setting ncores = NULL previously specified one less than the number of available cores.

References

Allaire, J. J., Francois, R., Ushey, K., Vandenbrouck, G., Geelnard, M. and Intel (2019) RcppParallel: Parallel Programming Tools for 'Rcpp'. R package version 4.4.4. https://CRAN.R-project.org/package=RcppParallel.

See Also

Parallel, `setThreadOptions` `Sys.getenv`

Examples

```
# determine current number of threads

setNumThreads()

## Not run:

# set new number of threads
setNumThreads(7)

# a call to secr.fit that specifies 'ncores' also sets the
# number of threads, as we see here

fit <- secr.fit(captdata, trace = FALSE, ncores = 8)
setNumThreads()


## End(Not run)
```

`shareFactorLevels` *Fix Inconsistent Covariates*

Description

Factor covariates can give trouble in multi-session models if the levels differ among sessions. A warning is provided by `verify.capthist`. This function forces factor covariates to use the same levels.

Usage

```
shareFactorLevels(object, columns = NULL, stringsAsFactors = TRUE)
```

Arguments

`object`	multi-session capthist object or list of traps or mask objects
`columns`	indices of columns to fix (default all)
`stringsAsFactors`	logical; if TRUE then character columns are converted to factor

Details

Factor-valued covariates are coerced to use the same set of levels for each session of a multi-session capthist object or each component of a list of traps or masks. The combined level set is the union of all levels in separate sessions. The order of levels follows the default in `factor` (alphabetical according to current locale).

Setting `stringsAsFactors = TRUE` causes character-valued columns to be converted to factors.

Value

An object of the same class as input. A single-session object is passed unchanged except for possible conversion of character values to factor (`stringsAsFactors = TRUE`).

See Also

`verify.capthist`

---

sighting *Sighting Attributes*

Description

Extract or replace the `markocc` attribute of a `traps` object that distinguishes marking occasions from sighting occasions. Also, extract or replace the attributes Tu, Tm and Tn of a capthist object, used for storing counts of sightings. All attributes are optional, but Tu, Tm and Tn require markocc to be specified. The attribute marked is the animal-level analogue of markocc: it records, for each animal and occasion, the fraction of the occasion for which the animal can be identified as marked, for example while a telemetry collar was working (telemetry type 'marking').

Usage

```
markocc(object, ...)
markocc(object) <- value
sighting(object)
Tu(object, ...)
Tu(object) <- value
Tm(object, ...)
Tm(object) <- value
Tn(object, ...)
Tn(object) <- value
marked(object, ...)
marked(object) <- value
```

Arguments

`object`	`traps` object (`markocc`) or capthist object (`Tu`, `Tm`, `Tn`, `marked`)
`value`	numeric matrix of detectors x occasions, or a vector (see Details); for `marked` a matrix of animals x non-telemetry occasions
`...`	other arguments (not used)

Details

For replacement of markocc, 'value' should be a vector of integers indicating the occasions on which animals are sighted only (0) or marked or recaptured (1).

For replacement of Tu, Tm or Tn, 'value' may be a scalar (total count) or a detectors x occasions matrix.

For replacement of marked, 'value' is a numeric matrix with one row per animal and one column per non-telemetry occasion (the same columns as for Tu), or NULL. The attribute is analogous to detector usage, which records the extent to which each detector was operating on each occasion: marked records the extent to which each animal was identifiable as marked on each occasion. Each element is a known, non-negative multiplier of the usage of the detectors for that animal on that occasion; usually 0 (e.g. collar not working) or 1, or a fraction if the status changed during the occasion. The default (NULL) is 1 for all animals. In telemetry type 'marking' a collared animal contributes cues to the identified counts and Tm only while marked; at other times its cues are part of the unmarked sightings Tu.

Value

markocc(object) returns the markocc vector of the traps object. markocc(object) may be NULL.

Tu, Tm, Tn and marked return the respective attributes of a capthist object, or NULL if they are unspecified.

sighting(object) returns TRUE if the markocc attribute indicates at least one sighting-only occasion.

See Also

traps, addSightings, sightingPlot, secr-markresight.pdf

---

signal *Signal Fields*

Description

Extract or replace signal attributes of a 'capthist' object.

Usage

```
signalframe(object)
signalframe(object) <- value

## S3 method for class 'capthist'
signal(object, ...)
## S3 method for class 'capthist'
noise(object, ...)

signal(object) <- value
noise(object) <- value
```

Arguments

`object`	a 'capthist' object
`value`	replacement value (see Details)
`...`	other arguments (not used)

Details

Signal attributes of a 'capthist' object are stored in a dataframe called the signalframe. This has one row per detection. The signalframe includes the primary field 'signal' and an unlimited number of other fields. To extract the signal field alone use the signal method.

These functions extract data on detections, ignoring occasions when an animal was not detected. Detections are ordered by occasion, animalID and trap.

Replacement values must precisely match `object` in number of detections and in their order.

Value

For `signalframe` , a dataframe containing signal data and covariates, one row per detection. The data frame has one row per detection. See `signalmatrix` for a matrix with one row per cue and columns for different microphones.

For `signal` and `noise`, a numeric vector with one element per detection.

If `object` has multiple sessions, the result is a list with one component per session.

See Also

`capthist`, `signalmatrix`

Examples

```
## ovensong dataset has very simple signalframe
head(signalframe(signalCH))
```

---

signalmatrix *Reformat Signal Data*

Description

Produce sound x microphone matrix, possibly with sound covariates as extra columns.

Usage

```
signalmatrix(object, noise = FALSE, recodezero = FALSE,
    prefix = "Ch", signalcovariates = NULL, names = NULL)
```

Arguments

`object`	object inheriting from secr class 'capthist'
`noise`	logical; if TRUE, noise is extracted instead of signal
`recodezero`	logical; if TRUE zero signals are set to NA
`prefix`	character value used to form channel names
`signalcovariates`	character vector of covariate names from signalframe to add as columns
`names`	character vector of column names

Details

This function extracts signal or noise data from a capthist object, where it is stored in the 'signalframe' attribute, as a more natural sound x microphone table. There is no equivalent replacement function.

The `signalcovariates` argument may be used to specify additional columns of the signal frame to collapse and add as columns to the right of the actual signal data. Ordinarily there will be multiple rows in signalframe for each row in the output; the covariate value is taken from the first matching row.

If `names` is not provided, column names are constructed from the detector names. If the length of `names` is less than the number of columns, simple numerical names are constructed.

Value

A dataframe with dim = c(n,K+j) where n is the number of separate sounds, K is the number of microphones, and j is the number of covariates (by default j = 0).

See Also

`ovensong`

Examples

```
## use 'secr' ovenbird data
signalmatrix(signalCH)
```

`sim.capthist` *Simulate Detection Histories*

Description

Create a set of capture or marking-and-resighting histories by simulated sampling of a 2-D population using an array of detectors.

Usage

```
sim.capthist(traps, popn = list(D = 5, buffer = 100,
    Ndist = "poisson"), detectfn = 0, detectpar = list(),
    noccasions = 5, nsessions = 1, binomN = NULL, exactN = NULL,
    p.available = 1, renumber = TRUE, seed = NULL,
    maxperpoly = 100, chulltol = 0.001, userdist = NULL,
    savepopn = FALSE)
sim.resight(traps, popn = list(D = 5, buffer = 100, Ndist = "poisson"), ...,
   pID = 1, unmarked = TRUE, nonID = TRUE,  unresolved = FALSE, unsighted = TRUE,
    pmark = 0.5, Nmark = NULL, markingmask = NULL)
```

Arguments

traps	traps object with the locations and other attributes of detectors
popn	locations of individuals in the population to be sampled, either as a popn object (see sim.popn) or a list with the named components.
detectfn	integer code or character string for shape of detection function 0 = halfnormal etc. – see detectfn
detectpar	list of values for named parameters of detection function
noccasions	number of occasions to simulate
nsessions	number of sessions to simulate
binomN	integer code for distribution of counts (see Details)
exactN	integer number of telemetry fixes per occasion
p.available	vector of one or two probabilities (see Details)
renumber	logical for whether output rows should labeled sequentially (TRUE) or retain the numbering of the population from which they were drawn (FALSE)
seed	either NULL or an integer that will be used in a call to set.seed
maxperpoly	integer maximum number of detections of an individual in one polygon or transect on any occasion
chulltol	numeric buffer (m) for polygon around telemetry locations
userdist	user-defined distance function or matrix (see details)
savepopn	logical; if TRUE then the popn (input or simulated) is saved as an attribute
...	arguments to pass to sim.capthist
pID	probability of individual identification for marked animals
unmarked	logical, if TRUE unmarked individuals are not recorded during 'sighting'
nonID	logical, if TRUE then unidentified marked individuals are not recorded during 'sighting'
unresolved	logical, if TRUE then individuals of unresolved mark status are not recorded during 'sighting'
unsighted	logical, if TRUE and sighting only then capthist includes all-zero histories
pmark	numeric probability that an individual is 'pre-marked' (see Details)
Nmark	number of individuals to be 'pre-marked' (see Details)
markingmask	mask object

Details

If popn is not of class 'popn' then a homogeneous Poisson population with the desired density (animals/ha) is first simulated over the rectangular area of the bounding box of traps plus a buffer of the requested width (metres). The detection algorithm depends on the detector type of traps. For 'proximity' detectors, the actual detection probability of animal *i* at detector *j* is the naive probability given by the detection function. For 'single' and 'multi' detectors the naive probability is modified by competition between detectors and, in the case of 'single' detectors, between animals. See Efford (2004) and other papers below for details.

Detection parameters in detectpar are specific to the detection function, which is indicated by detectfn. Parameters may vary with time - for this provide a vector of length noccasions. The g0 parameter may vary both by time and detector - for this provide a matrix with noccasions rows and as many columns as there are detectors. The default detection parameters are list(g0 = 0.2, sigma = 25, z = 1).

The default is to simulate a single session. This may be overridden by providing a list of populations to sample (argument popn) or by specifying nsessions > 1 (if both then the number of sessions must match). Using nsessions > 1 results in replicate samples of populations with the same density etc. as specified directly in the popn argument.

binomN determines the statistical distribution of the number of detections of an individual at a particular 'count' detector or polygon on a particular occasion. A Poisson distribution is indicated by binomN = 0; see secr.fit for more. The distribution is always Bernoulli (binary) for 'proximity' and 'signal' detectors.

If exactN is not specified or zero then the number of telemetry fixes is a random variable determined by the other detection settings.

p.available specifies temporary non-availability for detection in multi-session simulations. If a single probability is specified then temporary non-availability is random (independent from session to session). If two probabilities are given then non-availability is Markovian (dependent on previous state) and the two values are for animals available and not available at the preceding session. In the Markovian case, availability in the first session is assigned at random according to the equilibrium probability p2 / (1 - p1 + p2). Incomplete availability is not implemented for sampling lists of populations.

detectpar may include a component 'truncate' for the distance beyond which detection probability is set to zero. By default this value is NULL (no specific limit).

detectpar may also include a component 'recapfactor' for a general learned trap response. For 'single' and 'multi' detector types the probability of detection changes by this factor for all occasions after the occasion of first capture. Attempted use with other detector types causes an error. If recapfactor x g(d) > 1.0, g(d) is truncated at 1.0. A site-specific response (bk) is indicated by setting detectpar$bk = TRUE. Other types of response (e.g. Markovian B) are not allowed.

If popn is specified by an object of class 'popn' then any individual covariates will be passed on; the covariates attribute of the output is otherwise set to NULL.

The random number seed is managed as in simulate.

chulltol is used only when simulating telemetry locations. By default, a new 'traps' polygon is generated as the convex hull of the simulated locations, with a slight (1 mm) added buffer to ensure boundary points are within the polygon. Buffering is suppressed if chulltol is NA or negative.

userdist cannot be set if 'traps' is any of polygon, polygonX, transect or transectX.

sim.resight generates mark-resight data. The 'markocc' attribute of 'traps' indicates the occasions which are for sighting-only (0) or marking and recapture (1). The number of occasions is determined by markocc. sim.capthist is first called with the arguments 'traps' and .... The detector type of 'traps' should be 'proximity' or 'count' for sighting occasions (markocc = 0). The detector type need not be the same for marking and sighting occasions ('multi' is allowed on marking occasions). If ...includes a non-null 'seed' the random seed is reset in sim.resight and not passed to sim.capthist.

A special case arises when all occasions are sighting-only. Then it is assumed that individuals in the population are marked prior to the start of sampling with a known spatial distribution (i.e. marking does not follow a spatial detection model). By default, animals throughout the buffered area are pre-marked with probability pmark. If Nmark is specified then a sample of size Nmark will be selected for marking, overriding pmark.

The marked population may be restricted to a subset of the space spanned by popn by specifying markingmask, which may have a further covariate 'marking' to vary the intensity of marking.

Value

For sim.capthist, an object of class capthist, a 3-dimensional array with additional attributes. Rows represent individuals and columns represent occasions; the third dimension, codes the number of detections at each detector (zero or one for trap detectors ('single', 'multi') and binary proximity detectors.

The initial state of the R random number generator is stored in the 'seed' attribute.

For sim.resight, an object of class capthist for which the traps object has a markocc attribute (marking occasions), and there are further attributes Tu (sightings of unmarked animals) and Tm (sightings of marked but not identified animals).

Note

External code is called to speed the simulations. The present version assumes a null model, i.e., naive detection probability is constant except for effects of distance and possibly time (using vector-valued detection parameters from 1.2.10). You can, however, use rbind.capthist to combine detections of population subclasses (e.g. males and females) simulated with different parameter values. This is not valid for detector type "single" because it fails to allow for competition for traps between subclasses. Future versions may allow more complex models.

truncate has no effect (i) when using a uniform detection function with radius (sigma) <= truncate and (ii) with signal strength detection (detectfn 10, 11). Note that truncated detection functions are provided for de novo simulation, but are not available when fitting models with in secr.fit or simulating from a fitted model with sim.secr.

maxperpoly limits the size of the array allocated for detections in C code; an error results if the is number is exceeded.

Prior to 2.10.0 sim.resight interpreted length-2 vectors of detection parameters as referring to marking and sighting occasions; this feature has been discontinued.

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. (2004) Density estimation in live-trapping studies. *Oikos* **106**, 598–610.

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

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.

See Also

sim.popn, capthist, traps, popn, Detection functions, simulate, Tu, Tm

Examples

```
## simple example
## detector = "multi" (default)
temptrap <- make.grid(nx = 6, ny = 6, spacing = 20)
sim.capthist (temptrap, detectpar = list(g0 = 0.2, sigma = 20))

## with detector = "proximity", there may be more than one
## detection per individual per occasion
temptrap <- make.grid(nx = 6, ny = 6, spacing = 20, detector =
    "proximity")
summary(sim.capthist (temptrap, detectpar = list(g0 = 0.2,
    sigma = 20)))

## marking on occasions 1, 3 only
temptrap <- make.grid(nx = 6, ny = 6, spacing = 20, detector = 'proximity')
markocc(temptrap) <- c(1,0,1,0,0)
CH <- sim.resight (temptrap, detectpar = list(g0 = 0.2, sigma = 20))
summary(CH)

## multiple sessions
grid4 <- make.grid(nx = 2, ny = 2)
temp <- sim.capthist (grid4, popn = list(D = 1), nsessions = 20)
summary(temp, terse = TRUE)

## unmarked or presence types
# grid <- make.grid(nx = 10, ny = 10, detector = "unmarked")
# CH <- sim.capthist (grid, noccasions = 5)
# CH
## "presence" and "unmarked" data are stored as "count" data
## behaviour is controlled by detector type, e.g.
# detector(traps(CH)) <- "presence"
# CH
```

sim.popn *Simulate 2-D Population*

Description

Simulate a point process representing the locations of individual animals.

Usage

```
sim.popn (D, core, buffer = 100, model2D = c("poisson", "cluster",
  "IHP", "coastal", "hills", "linear", "even", "rLGCP", "rThomas"),
  buffertype = c("rect", "concave", "convex"), poly = NULL,
  covariates = list(sex = c(M = 0.5, F = 0.5)), number.from = 1,
  Ndist = c("poisson", "fixed", "specified"), nsessions = 1, details = NULL,
  seed = NULL, keep.mask = model2D %in% c("IHP", "linear"),
  Nbuffer = NULL, age = FALSE, ...)

tile(popn, method = "reflect")
```

Arguments

D	density animals / hectare (10 000 m^2) (see Details for IHP case)
core	data frame of points defining the core area
buffer	buffer radius about core area
model2D	character string for 2-D distribution
buffertype	character string for buffer type
poly	bounding polygon (see Details)
covariates	list of named covariates or function to generate covariates
number.from	integer ID for animal
Ndist	character string for distribution of number of individuals
nsessions	number of sessions to simulate
details	optional list with additional parameters
seed	either NULL or an integer that will be used in a call to `set.seed`
keep.mask	logical; if TRUE and model2D %in% c('IHP','linear') then `core` is saved as the attribute "mask"
Nbuffer	numeric number of individuals to simulate (possibly non-integer)
age	logical; if TRUE then age covariate added for multisession popn with turnover
...	arguments passed to subset if poly is not NULL
popn	popn object
method	character string "reflect" or "copy"

Details

core must contain columns 'x' and 'y'; a traps object is suitable. For buffertype = "rect", animals are simulated in the rectangular area obtained by extending the bounding box of core by buffer metres to top and bottom, left and right. This box has area A. If model2D = 'poisson' the buffer type may also be 'convex' (points within a buffered convex polygon) or 'concave' (corresponding to a mask of type 'trapbuffer'); these buffer types use bufferContour.

Covariates may be specified in either of two ways. In the first, each element of covariates defines a categorical (factor) covariate with the given probabilities of membership in each class. In the second, the 'covariates' argument is a function (or a character value naming a function) that takes a dataframe of x and y coordinates as its sole argument; the function should return a dataframe with the same number of rows that will be used as the covariates attribute (**secr** >= 4.6.7).

A notional random covariate 'sex' is generated by default.

Ndist should usually be 'poisson' or 'fixed'. The number of individuals N has expected value DA. If DA is non-integer then Ndist = "fixed" results in $N \in \{\mathrm{trunc}(DA), \mathrm{trunc}(DA) + 1\}$, with probabilities set to yield DA individuals on average. The option 'specified' is undocumented; it is used in some open-population simulations.

If model2D = "cluster" then the simulated population approximates a Neyman-Scott clustered Poisson distribution. Ancillary parameters are passed as components of details: details$mu is the expected number of individuals per cluster and details$hsigma is the spatial scale (σ) of a 2-D kernel for location within each cluster. The algorithm is

1. Determine the number of clusters (parents) as a random Poisson variate with $\lambda = DA/\mu$
2. Locate each parent by drawing uniform random x- and y-coordinates
3. Determine number of offspring for each parent by drawing from a Poisson distribution with mean mu
4. Locate offspring by adding random normal error to each parent coordinate
5. Apply toroidal wrapping to ensure all offspring locations are inside the buffered area

A special cluster option is selected if details$clone = "constant": then each parent is cloned exactly details$mu times.

Toroidal wrapping is a compromise. The result is more faithful to the Neyman-Scott distribution if the buffer is large enough that only a small proportion of the points are wrapped.

If model2D = "IHP" then an inhomogeneous Poisson distribution is simulated. core should be a habitat mask and D should be one of –

- a vector of length equal to the number of cells (rows) in core,
- the name of a covariate in core that contains cell-specific densities (animals / hectare),
- a function to generate the intensity of the distribution at each mask point, or
- a constant.

If a function, D should take two arguments, a habitat mask and a list of parameter values ('core' and 'details' are passed internally as these arguments). A function may be given by name (i.e. as a character value); this is required when nsessions>1.

The number of individuals in each cell is either (i) Poisson-distributed with mean DA where A is the cell area (an attribute of the mask) (Ndist = "poisson") or (ii) multinomial with size DA and

relative cell probabilities given by D (`Ndist = "fixed"`). `buffertype` and `buffer` are ignored, as the extent of the population is governed entirely by the mask in `core`.

If `model2D = "linear"` then a linear population is simulated as for `model2D = "IHP"`, except that `core` should be a linearmask object from package **secrlinear**, and density (D) is expressed in animals per km. The documentation of **secrlinear** should be consulted for further detail (e.g. the wrapper function `sim.linearpopn`).

If `model2D = "coastal"` then a form of inhomogeneous Poisson distribution is simulated in which the x- and y-coordinates are drawn from independent Beta distributions. Default parameters generate the 'coastal' distribution used by Fewster and Buckland (2004) for simulations of line-transect distance sampling (x ~ Beta(1, 1.5), y ~ Beta(5, 1), which places 50% of the population in the 'northern' 13% of the rectangle). The four Beta parameters may be supplied in the vector component Beta of the 'details' list (see Examples). The Beta parameters (1,1) give a uniform distribution. Coordinates are scaled to fit the limits of a sampled rectangle, so this method assumes buffertype = "rect".

If `model2D = "hills"` then a form of inhomogeneous Poisson distribution is simulated in which intensity is a sine curve in the x- and y- directions (density varies symmetrically between 0 and 2 x D along each axis). The number of hills in each direction (default 1) is determined by the 'hills' component of the 'details' list (e.g. details = list(hills=c(2,3)) for 6 hills). If either number is negative then alternate rows will be offset by half a hill. Displacements of the entire pattern to the right and top are indicated by further elements of the 'hills' component (e.g. details = list(hills=c(1,1,0.5,0.5)) for 1 hill shifted half a unit to the top right; coordinates are wrapped, so the effect is to split the hill into the four corners). Negative displacements are replaced by runif(1). Density is zero at the edge when the displacement vector is (0,0) and rows are not offset.

If `model2D = "even"` then the buffered area is divided into square cells with side sqrt(10000/D) and one animal is located at a random uniform location within each cell. If the height or width is not an exact multiple of the cell side then one whole extra row or column of cells is added; animals located at random in these cells are discarded if they fall outside the original area.

From **secr** 4.6.2, `sim.popn` provides an interface to two simulation functions from **spatstat** (Baddeley et al. 2015): `rLGCP` and `rThomas`.

If `model2D = "rLGCP"` then a log-gaussian Cox process is simulated within the buffered area. Function `rLGCP` in recent versions of **spatstat** uses internal code rather than Schlather et al. (2015) as previously. The algorithm first constructs the log spatial intensity as a realisation of a Gaussian random field; one realisation of an IHP with that intensity is then simulated.

The parameters for `model2D = "rLGCP"` are the scalar density (D) and the variance and spatial scale of the random field (passed as details arguments 'var' and 'scale'). The variance is on the log scale; the mean on the log scale is computed internally as mu = log(D) - var/2. var = 0 results in a random uniform (Poisson) distribution. When details$saveLambda = TRUE, the discretized intensity function is saved as the attribute "Lambda", a habitat mask with covariate "Lambda" that may be used to construct further IHP realisations (see Examples).

If `model2D = "rThomas"` then a Thomas process is simulated. This is a special case of the Neyman-Scott process in which each parent gives rise to a Poisson number of offspring (see Notes). The expected number of offspring per parent and the spatial scatter about each parent are specified by the details arguments 'mu' and 'scale'. Argument 'kappa' of `rThomas` (density of parent process) is computed as D/mu/1e4. Other arguments remain at their defaults, including 'expand' (4 * scale). A dataframe of parent locations is saved in attribute 'parents'. The intensity surface for each realisation is saved in attribute 'Lambda' when details$saveLambda = TRUE.

Clipping to a polygon (poly) is supported for both rLGCP and rThomas in secr >= 5.2.2, but care is needed. The boundary coordinates must be given anticlockwise and not include duplicates (even first and last).

Conditional simulation (Ndist = 'fixed') is supported for both rLGCP and rThomas in secr >= 5.2.2. The n.cond argument of the respective **spatstat** functions is set internally by 'sim.popn'. The rejection algorithm may fail. Set details$verbose = TRUE for detailed reporting.

If `poly` is specified, points outside `poly` are dropped. `poly` may be one of the types descrbed in `boundarytoSF`.

The `subset` method is called internally when `poly` is used; the ... argument may be used to pass values for `keep.poly` and `poly.habitat`.

Multi-session populations may be generated with `nsessions > 1`. Multi-session populations may be independent or generated by per capita turnover from a starting population. In the 'independent' case (`details$lambda` not specified) D or Nbuffer may be a vector of length equal to `nsessions`. Turnover is controlled by survival, growth rate and movement parameters provided as components of `details` and described in turnover. The optional covariate 'age' is the number of sessions from the session of recruitment.

The random number seed is managed as in `simulate.lm`.

Function `tile` replicates a popn pattern by either reflecting or copying and translating it to fill a 3 x 3 grid.

Value

An object of class `c("popn", "data.frame")` a data frame with columns 'x' and 'y'. Rows correspond to individuals. Individual covariates (optional) are stored as a data frame attribute. The initial state of the R random number generator is stored in the 'seed' attribute.

If `model2D = "linear"` the output is of class `c("linearpopn", "popn", "data.frame")`.

If `model2D = "IHP"` or `model2D = "linear"` the value of `core` is stored in the 'mask' attribute.

Notes

`model2D = "rThomas"` and `model2D = "cluster"` (the builtin Neyman-Scott implementation) are equivalent. There may be some subtle differences. The **spatstat** implementation is usually to be preferred.

References

Baddeley, A., Rubak, E., and Turner, R. 2015. Spatial Point Patterns: Methodology and Applications with R. Chapman and Hall/CRC Press, London. ISBN 9781482210200, https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/p/book/9781482210200/.

Fewster, R. M. and Buckland, S. T. 2004. Assessment of distance sampling estimators. In: S. T. Buckland, D. R. Anderson, K. P. Burnham, J. L. Laake, D. L. Borchers and L. Thomas (eds) *Advanced distance sampling*. Oxford University Press, Oxford, U. K. Pp. 281–306.

Schlather, M., Malinowski, A., Menck, P. J., Oesting, M. and Strokorb, K. 2015. Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software*, **63**, 1–25. URL https://www.jstatsoft.org/v63/i08/.

See Also

popn, plot.popn, randomHabitat, turnover, simulate

Examples

```
temppop <- sim.popn (D = 10, expand.grid(x = c(0,100), y =
    c(0,100)), buffer = 50)

## plot, distinguishing "M" and "F"
plot(temppop, pch = 1, cex= 1.5,
    col = c("green","red")[covariates(temppop)$sex])

## add a continuous covariate
## assumes covariates(temppop) is non-null
covariates(temppop)$size <- rnorm (nrow(temppop), mean = 15, sd = 3)
summary(covariates(temppop))

## Neyman-Scott cluster distribution (see also rThomas)
par(xpd = TRUE, mfrow=c(2,3))
for (h in c(5,15))
for (m in c(1,4,16)) {
    temppop <- sim.popn (D = 10, expand.grid(x = c(0,100),
        y = c(0,100)), model2D = "cluster", buffer = 100,
        details = list(mu = m, hsigma = h))
    plot(temppop)
    text (50,230,paste(" mu =",m, "hsigma =",h))
}
par(xpd = FALSE, mfrow=c(1,1))   ## defaults

## Inhomogeneous Poisson distribution
xy <- secrdemo.0$mask$x + secrdemo.0$mask$y - 900
tempD <- xy^2 / 1000
plot(sim.popn(tempD, secrdemo.0$mask, model2D = "IHP"))

## Coastal distribution in 1000-m square, homogeneous in
## x-direction
arena <- data.frame(x = c(0, 1000, 1000, 0),
    y = c(0, 0, 1000, 1000))
plot(sim.popn(D = 5, core = arena, buffer = 0, model2D =
    "coastal", details = list(Beta = c(1, 1, 5, 1))))

## Hills
plot(sim.popn(D = 100, core = arena, model2D = "hills",
    buffer = 0, details = list(hills = c(-2,3,0,0))),
    cex = 0.4)

## tile demonstration
pop <- sim.popn(D = 100, core = make.grid(), model2D = "coastal")
par(mfrow = c(1,2), mar = c(2,2,2,2))
plot(tile(pop, "copy"))
polygon(cbind(-100,200,200,-100), c(-100,-100,200,200),
    col = "red", density = 0)
```

```
title("copy")
plot(tile(pop, "reflect"))
polygon(cbind(-100,200,200,-100), c(-100,-100,200,200),
    col = "red", density = 0)
title("reflect")

## Not run:

## simulate from inhomogeneous fitted density model

regionmask <- make.mask(traps(possumCH), type = "polygon",
    spacing = 20, poly = possumremovalarea)
dts <- distancetotrap(regionmask, possumarea)
covariates(regionmask) <- data.frame(d.to.shore = dts)
dsurf <- predictDsurface(possum.model.Ds, regionmask)
possD <- covariates(dsurf)$D.0
posspop <- sim.popn(D = possD, core = dsurf, model = "IHP")
plot(regionmask, dots = FALSE, ppoly = FALSE)
plot(posspop, add = TRUE, frame = FALSE)
plot(traps(possumCH), add = TRUE)

## randomHabitat demonstration
## - assumes igraph has been installed

# The wrapper function randomDensity may be passed to generate
# a new habitat map each time sim.popn is called. The `details' argument
# of sim.popn is passed to randomDensity as the `parm' argument.

if (requireNamespace("igraph")) {

tempmask <- make.mask(nx = 100, ny = 100, spacing = 20)
pop <- sim.popn(D = randomDensity, core = tempmask, model2D = "IHP",
    details = list(D = 10, p = 0.4, A = 0.5))
plot(attr(pop, 'mask'), cov = 'D', dots = FALSE)
plot(pop, add = TRUE)

}

## rLGCP demonstration
## - assumes spatstat has been installed

if (requireNamespace("spatstat")) {
    msk <- make.mask(traps(captdata))
    # details argument 'spacing' ensures core matches Lambda below
    pop <- sim.popn(D = 20, core = msk, buffer = 0,
        model2D = "rLGCP", details = list(var=1, scale = 30, saveLambda = TRUE),
        seed = 1234)
    plot(pop)
    plot(traps(captdata), add = TRUE)

    # another IHP realisation from same LGCP intensity surface
    lgcp <- attr(pop, 'Lambda')
    pop2 <- sim.popn(D = 'Lambda', core = lgcp, model2D = "IHP")
```

```
    plot (lgcp, covariate = "Lambda", dots = FALSE)
    plot (pop2, add = TRUE, frame = FALSE)

    # check input and output masks match
    summary(lgcp)
    summary(msk)
}


## End(Not run)
```

---

sim.secr *Simulate From Fitted secr Model*

Description

Simulate a spatially distributed population, sample from that population with an array of detectors, and optionally fit an SECR model to the simulated data.

Usage

```
## S3 method for class 'secr'
simulate(object, nsim = 1, seed = NULL, maxperpoly = 100,
    chat = 1, poponly = FALSE, ...)

sim.secr(object, nsim = 1, extractfn = function(x) c(deviance =
    deviance(x), df = df.residual(x)), seed = NULL, maxperpoly = 100,
    data = NULL, tracelevel = 1, hessian = c("none", "auto", "fdHess"),
    start = "true", ncores = NULL, ...)

sim.detect(object, popnlist, maxperpoly = 100, renumber = TRUE,
    expected = FALSE, dropzeroCH = TRUE)
```

Arguments

object	a fitted secr model
nsim	integer number of replicates
seed	either NULL or an integer that will be used in a call to `set.seed`
maxperpoly	integer maximum number of detections of an individual in one polygon or transect on any occasion
chat	real value for overdispersion parameter
poponly	logical; if TRUE then only populations are simulated
ncores	integer number of threads used by `secr.fit`
extractfn	function to extract output values from fitted model

data	optional list of simulated data saved from previous call to simulate.secr
tracelevel	integer for level of detail in reporting (0,1,2)
hessian	character or logical controlling the computation of the Hessian matrix
start	vector of starting 'beta' values for secr.fit, NULL or 'true'
...	other arguments (not used by simulate, passed to 'extractfn' by sim.secr)
popnlist	list of popn objects
renumber	logical; if TRUE then output animals are renumbered
expected	logical; if TRUE then the array of expected counts is saved as an attribute
dropzeroCH	logical; if TRUE then all-zero capture histories are dropped

Details

For each replicate, simulate.secr calls sim.popn to generate session- and group-specific realizations of the (possibly inhomogeneous) 2-D Poisson distribution fitted in object, across the habitat mask(s) in object. Group subpopulations are combined using rbind.popn within each session; information to reconstruct groups is retained in the individual-level factor covariate(s) of the resulting popn object (corresponding to object$groups). Unless 'poponly = TRUE' each population is then sampled using the fitted detection model and detector (trap) array(s) in object.

The random number seed is managed as in simulate.lm.

Certain model types are not supported by simulate.secr. These include models fitted using conditional likelihood (object$CL = TRUE), telemetry models and exotic behavioural response models.

Detector type is determined by detector(traps(object$capthist)).

sim.secr is a wrapper function. If data = NULL (the default) then it calls simulate.secr to generate nsim new datasets. If data is provided then nsim is taken to be length(data). secr.fit is called to fit the original model to each new dataset. Results are summarized according to the user-provided function extractfn. The default extractfn returns the deviance and its degrees of freedom; a NULL value for extractfn returns the fitted secr objects after trimming to reduce bulk. Simulation uses the detector type of the data, even when another likelihood is fitted (this is the case with single-catch data, for which a multi-catch likelihood is fitted). Warning messages from secr.fit are suppressed.

extractfn should be a function that takes an secr object as its only argument.

tracelevel=0 suppresses most messages; tracelevel=1 gives a terse message at the start of each fit; tracelevel=2 also sets 'details$trace = TRUE' for secr.fit, causing each likelihood evaluation to be reported.

hessian controls computation of the Hessian matrix from which variances and covariances are obtained. hessian replaces the value in object\$details. Options are "none" (no variances), "auto" (the default) or "fdhess" (see secr.fit). It is OK (and faster) to use hessian="none" unless extractfn needs variances or covariances. Logical TRUE and FALSE are interpreted by secr.fit as "auto" and "none".

If ncores = NULL then the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS is used (see setNumThreads).

sim.capthist is a more direct way to simulate data from a null model (i.e. one with constant parameters for density and detection), or from a time-varying model.

sim.detect is a function used internally that will not usually be called directly.

Value

For `simulate.secr`,

- if poponly = TRUE, a list of populations ('popn' objects)
- if poponly = FALSE, a list of data sets ('capthist' objects). This list has class `c("secrdata", "list")`

The initial state of the random number generator (roughly, the value of .Random.seed) is stored as the attribute 'seed'.

The value from `sim.secr` depends on `extractfn`: if that returns a numeric vector of length `n.extract` then the value is a matrix with `dim = c(nsim, n.extract)` (i.e., the matrix has one row per replicate and one column for each extracted value). Otherwise, the value returned by `sim.secr` is a list with one component per replicate (strictly, an object of class = `c("secrlist", "list")`). Each simulated fit may be retrieved *in toto* by specifying `extractfn = identity`, or slimmed down by specifying `extractfn = NULL` or `extractfn = trim`, which are equivalent.

For either form of output from `sim.secr` the initial state of the random number generator is stored as the attribute 'seed'.

For `sim.detect` a list of 'capthist' objects.

Warning

`sim.secr` does not work for mark–resight models.

Note

The value returned by `simulate.secr` is a list of 'capthist' objects; if there is more than one session, each 'capthist' is itself a sort of list .

The classes 'secrdata' and 'secrlist' are used only to override the ugly and usually unwanted printing of the seed attribute. However, a few other methods are available for 'secrlist' objects (e.g. `plot.secrlist`).

The default value for `start` in `sim.secr` is the previously fitted parameter vector. Alternatives are NULL or a (complete) user-constructed vector of beta values.

See Also

`sim.capthist`, `secr.fit`, `simulate`, `secr.test`

Examples

```
## Not run:

## previously fitted model
simulate(secrdemo.0, nsim = 2)

## The following has been superceded by secr.test()

## this would take a long time...
sims <- sim.secr(secrdemo.0, nsim = 99)
```

```
deviance(secrdemo.0)
devs <- c(deviance(secrdemo.0),sims$deviance)
quantile(devs, probs=c(0.95))
rank(devs)[1] / length(devs)

## to assess bias and CI coverage
extrfn <- function (object) unlist(predict(object)["D",-1])
sims <- sim.secr(secrdemo.0, nsim = 50, hessian = "auto",
    extractfn = extrfn)
sims

## with a larger sample, could get parametric bootstrap CI
quantile(sims[,1], c(0.025, 0.975))


## End(Not run)
```

`simOU.capthist` *Simulate Detections For Ornstein-Uhlenbeck Movement*

Description

The usual SECR model ignores the sequential locations of an individual within its home range. Movement models predict serial correlation of detections in space. The Ornstein-Uhlenbeck (OU) model is a convenient example that over long durations leads to a bivariate normal distribution of locations.

Movements of a single animal according to the OU model are simulated in discrete time with `simOU`.

Detections of a population of individuals with pre-defined activity centres are simulated with `simOU.capthist`. Detection happens when the location of an individual at time t (occasion t) is within distance epsilon ϵ of a detector.

Usage

```
simOU(xy, tau, sigma, noccasions, start = NULL)

simOU.capthist(traps, popn, detectpar, noccasions, seed = NULL,
    savepopn = FALSE, savepath = FALSE, ...)
```

Arguments

`xy`	numeric vector of x,y coordinates for one animal, or 2-column matrix
`tau`	numeric parameter of serial correlation = $1/\beta$
`sigma`	numeric spatial scale parameter
`noccasions`	integer number of time steps
`start`	numeric vector of x,y coordinates for initial location (optional)

traps	secr traps object
popn	secr popn object or a 2-column matrix of x-y coordinates of activity centres
detectpar	list with values of detection parameters epsilon, sigma, and tau
seed	either NULL or an integer that will be used in a call to `set.seed`
savepopn	logical; if TRUE the population is saved as an attribute
savepath	logical; if TRUE the movement paths are saved as an attribute
...	other arguments passed to `reduce.capthist`

Details

The first location for `simOU` by default is drawn at random from the asymptotic distribution.

The detection parameters are:

epsilon	radius within which individual detected with certainty
sigma	spatial scale of asymptotic bivariate normal
tau	serial correlation parameter 1/beta

In a final step, 'simOU.capthist' uses the reduce method for capthist objects to coerce the simulated capthist object to the detector type of the traps argument. This step is skipped to save time if 'traps' has detector type "multi" and ... is empty.

The ... argument may be used to pass the 'by' argument to reduce.capthist. For example, 'by = "ALL"' collapses the initially binary data for a single detector on noccasions to a single integer. Alternatively, 'by = 10' collapses the original occasions in groups of 10. Data will be lost unless the input traps object has detector type 'count'.

The default 'verify' step in reduce.capthist may also be suppressed (set `verify = FALSE` in `simOU.capthist()`).

The x- and y-dimensions are simulated separately, assuming circularity. The distribution of location on the x axis at time $t+1$ conditional on the location at time t is then

$$x_{t+1}|x_t \sim N(\mu_x + e^{\frac{-1}{\tau}(x_t-\mu_x)},\ \sigma^2(1-e^{\frac{-2}{\tau}}),$$

where $\mu = (\mu_x, \mu_y)$ is the long-term activity centre and τ (tau) is a parameter for the strength of serial correlation ($\tau = 1/\beta$ in other formulations). The scale of the long-term (asymptotic) bivariate normal home range is governed by σ as usual. Steps are implicitly of length 1 occasion so Δt is omitted.

Value

simOU - matrix of locations dim = c(noccasions, 2), or a list of these, one per row of xy

simOU.capthist - single-session capthist object for **secr**

See Also

`sim.capthist`

Examples

```
# one animal
locs <- simOU(c(0,0), 20, 1, 100)
par(pty = 's')
plot(locs, type = 'o', xlim = c(-2.5,2.5), ylim = c(-2.5,2.5))
points(0,0, pch = 16, col = 'red')



# simulate some capture data
set.seed(123)
grid <- make.grid(8, 8, spacing = 2)
pop <- sim.popn(D = 1000, core = grid, buffer = 4)
ch <- simOU.capthist(grid, pop, detectpar=list(tau = 50, sigma = 1,
    epsilon = 0.25), noccasions = 100, savepath = TRUE)

# plot simulated capthist (blue dots), movements (grey), and AC (triangles)
plot(pop, pch = 2, cex = 1, border = 4)
plot(ch, rad = 0.2, tracks = TRUE, varycol = FALSE, add = TRUE,
    gridlines = FALSE)
symbols(grid$x, grid$y, circles = rep(0.25,64), inches = FALSE, fg = 'red',
    add = TRUE)
sapply(attr(ch, 'path'), lines, col = 'grey')

# fit a misspecified HHN model
# density estimates show a small negative bias
# sigma estimates show a large negative bias
# lambda0 cannot be compared
fit <- secr.fit(capthist = ch, buffer = 8, detectfn = 14, trace = FALSE)
predict(fit)
```

---

skink *Skink Pitfall Data*

Description

Data from a study of skinks (*Oligosoma infrapunctatum* and *O. lineoocellatum*) in New Zealand.

Usage

```
infraCH
lineoCH
LStraps
```

Details

Lizards were studied over several years on a steep bracken-covered hillside on Lake Station in the Upper Buller Valley, South Island, New Zealand. Pitfall traps (sunken cans baited with a morsel of fruit in sugar syrup) were set in two large grids, each 11 x 21 traps nominally 5 meters apart, surveyed by tape and compass (locations determined later with precision surveying equipment - see Examples). Three diurnal lizard species were trapped: *Oligosoma infrapunctatum*, *O. lineoocellatum* and *O. polychroma* (Scincidae). The smallest species *O. polychroma* was seldom caught and these data are not included. The two other species are almost equal in average size (about 160 mm total length); they are long-lived and probably mature in their second or third year. The study aimed to examine their habitat use and competitive interactions.

Traps were set for 12 3-day sessions over 1995–1996, but some sessions yielded very few captures because skinks were inactive, and some sessions were incomplete for logistical reasons. The data are from sessions 6 and 7 in late spring (17–20 October 1995 and 14–17 November 1995). Traps were cleared daily; the few skinks present when traps were closed on the morning of the fourth day are treated as Day 3 captures. Individuals were marked uniquely by clipping one toe on each foot. Natural toe loss caused some problems with long-term identification; captures were dropped from the dataset when identity was uncertain. Released animals were occasionally recaptured in a different trap on the same day; these records were also discarded.

The data are provided as two two-session `capthist` objects 'infraCH' and 'lineoCH'. Also included is 'LStraps', the `traps` object with the coordinates and covariates of the trap sites (these data are also embedded in each of the `capthist` objects). Pitfall traps are multi-catch traps so `detector(LStraps)` = 'multi'.

Habitat data for each trap site are included as a dataframe of trap covariates in `LStraps`. Ground cover and vegetation were recorded for a 1-m radius plot at each trap site. The dataframe also gives the total number of captures of each species by site on 31 days between April 1995 and March 1996, and the maximum potential annual solar radiation calculated from slope and aspect (Frank and Lee 1966). Each site was assigned to a habitat class by fuzzy clustering (Kaufman and Rousseauw 1990; package **cluster**) of a distance matrix using the ground cover, vegetation and solar radiation variables. Sites in class 1 were open with bare ground or low-canopy vegetation including the heath-like *Leucopogon fraseri* and grasses; sites in class 2 had more-closed vegetation, lacking *Leucopogon fraseri* and with a higher canopy that often included *Coriaria arborea*. Site variables are listed with definitions in the attribute `habitat.variables` of `LStraps` (see Examples).

Object	Description
infraCH	multi-session capthist object *O. infrapunctatum*
lineoCH	multi-session capthist object *O. lineoocellatum*
LStraps	traps object – Lake Station grids

Source

M. G. Efford, B. W. Thomas and N. J. Spencer unpublished data.

References

Efford, M. G., Spencer, N. J., Thomas, B. W., Mason, R. F. and Williams, P. In prep. Distribution of sympatric skink species in relation to habitat.

Frank, E. C. and Lee , R. (1966) Potential solar beam irradiation on slopes. *United States Forest Service Research Paper* RM-118.

Kaufman, L. and Rousseauw, P. J. (1990) *Finding groups in data: an introduction to cluster analysis.* John Wiley & Sons, New York.

Spencer, N. J., Thomas, B. W., Mason, R. F. and Dugdale, J. S. (1998) Diet and life history variation in the sympatric lizards *Oligosoma nigriplantare polychroma* and *Oligosoma lineoocellatum*. New Zealand Journal of Zoology 25: 457–463.

See Also

capthist, covariates

Examples

```
summary (infraCH)
summary (lineoCH)

## check mean distance to nearest trap etc.
summary(LStraps)

## LStraps has several site covariates; terse descriptions are in
##  an extra attribute that may be displayed thus
attr(LStraps, "habitat.variables")

## For density modelling we need covariate values at each point in the
## habitat mask. This requires both on-grid interpolation and
## extrapolation beyond the grids. One (crude) possibility is to
## extrapolate a mask covariate from a covariate of the nearest trap:

LSmask <- make.mask(LStraps, buffer = 30, type = "trapbuffer")
temp <- nearesttrap(LSmask, LStraps)
habclass <- covariates(LStraps)$class[temp]
habclass <- factor (habclass, levels = c(1,2))
covariates(LSmask) <- data.frame(habclass)

## plot mask with colour-coded covariate
par(fg = "white")  ## white pixel borders
plot (LSmask, covariate = "habclass", dots = FALSE, axes = FALSE,
    col = c("yellow", "green"), border = 0)
plot(LStraps, add = TRUE, detpar = list(pch = 16))
par(fg = "black")    ## default
```

---

smooths *Smooth Terms in SECR Models*

Description

From version 2.9.0, the model formulae provided to `secr.fit` may include smooth terms as specified for the **mgcv** function 'gam', with some restrictions. Smooth terms may be used for both density and detection parameters.

Details

The specification of smooth terms is explained in `formula.gam`. Only a subset of options are relevant to 'secr'. Penalized splines are not available. The smooth function may be 's' or 'te'.

The 'wiggliness' of the curve is controlled by the argument k, which in this implementation is set by the user. The argument 'fx' should be set to TRUE.

See also the example in Chapter 11 of The SECR book.

Background

Regression splines are a very flexible way to represent non-linear responses in generalized additive models (e.g., **mgcv**, Wood 2006). Borchers and Kidney (2014) have shown how they may be used to model 2-dimensional trend in density in **secrgam**, an R package that extends **secr**. Their approach is to use **mgcv** to construct regression spline basis functions from mask x- and y-coordinates, and possibly additional mask covariates, and to pass these as covariates to **secr**. The idea of using **mgcv** to construct the basis functions is applied within **secr** from version 2.9.

Smooth semi-parametric responses are also useful for modelling variation in detection parameters such as g0 and sigma over time, or in response to individual or detector-level covariates, when (1) a linear or other parametric response is arbitrary or implausible, and (2) sampling spans a range of times or levels of the covariate(s).

For a concrete example, consider a population sampled monthly for a year (i.e., 12 'sessions'). If home range size varies seasonally then the parameter sigma may vary in a more-or-less sinusoidal fashion. A linear trend is obviously inadequate, and a quadratic is not much better. However, a sine curve is hard to fit (we would need to estimate its phase, amplitude, mean and spatial scale) and assumes the increase and decrease phases are equally steep. An extreme solution is to treat month as a factor and estimate a separate parameter for each level (month). A smooth (semi-parametric) curve may capture the main features of seasonal variation with fewer parameters.

Warning

This implementation of smooth models results in large fitted objects, on account of the need to store setup information from **mgcv**. It is also vulnerable to future changes in **mgcv**.

Expect that the implementation will change in later versions of **secr**, and that smooth models fitted in the this version will not necessarily be compatible with `predict` and `predictDsurface` in later versions.

Note

Setting the intercept of a smooth to zero is not a canned option in **mgcv**, and is not offered in **secr**. It may be achieved by placing a knot at zero and hacking the matrix of basis functions to drop the corresponding column, plus some more jiggling.

References

Borchers, D. L. and Kidney, D. (2014) *Flexible density surface estimation for spatially explicit capture–recapture surveys*. Technical Report, University of St Andrews.

Wood, S. N. (2006) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC.

See Also

formula.gam

Examples

```
## Not run:

## smooth density surface
possum.model.sxy <- secr.fit(possumCH, mask = possummask,
    model = D ~ s(x,y, k = 6, fx = TRUE), trace = FALSE)
fittedsurface <- predictDsurface(possum.model.sxy)
par(mar = c(1,1,1,6))
plot(fittedsurface)
plot(fittedsurface, plottype = 'contour', add = TRUE)
par(mar = c(5,4,4,2) + 0.1)  ## reset to default

## Now try smooth on g0

## For the smooth we use 'Session' which is coded numerically (0:4)
## rather than the factor 'session' ('2005', '2006', '2007', '2008',
## '2009')

ovenbird.model.g0 <- secr.fit(ovenCH, mask = ovenmask,
    model = g0 ~ session, trace = FALSE)
ovenbird.model.sg0 <- secr.fit(ovenCH, mask = ovenmask,
    model = g0 ~ s(Session, k = 3, fx = TRUE), trace = FALSE)

AIC(ovenbird.model.g0, ovenbird.model.sg0)

## Or over occasions within a session...

fit.sT3 <- secr.fit(captdata, model = g0 ~ s(T, k = 3, fx = TRUE),
    trace = FALSE)
pred <- predict(fit.sT3, newdata = data.frame(T = 0:4))

plot(sapply(pred, '[', 'g0', 'estimate'))


## End(Not run)
```

---

snip *Slice Transect Into Shorter Sections*

Description

This function splits the transects in a ‘transect’ or ‘transectX’ traps object into multiple shorter sections. The function may also be applied directly to a capthist object based on transect data. This makes it easy to convert detection data collected along linear transects to point detection data (see Example).

Usage

```
snip(object, from = 0, by = 1000, length.out = NULL, keep.incomplete = TRUE, tol = 0.01)
```

Arguments

object	secr 'traps' or 'capthist' object based on transects
from	numeric starting posiiton (m)
by	numeric length of new transects (m)
length.out	numeric number of new transects, as alternative to 'by'
keep.incomplete	logical; if TRUE then initial or terminal sections of each original transect that are less than 'by' will be retained in the output
tol	numeric tolerance for xyontransect (capthist only)

Details

If a positive `length.out` is specified, `by` will be computed as `(transectlength(object) - from) / length.out`.

Value

A 'traps' or 'capthist' object, according to the input. If `keep.incomplete == FALSE` animals and detections from the

Warning

`snip` does not work for mark–resight data.

See Also

`transectlength`, `discretize`

Examples

```
x <- seq(0, 4*pi, length = 41)
temptrans <- make.transect(x = x*100, y = sin(x)*300)
plot (snip(temptrans, by = 200), markvertices = 1)

## Not run:

## simulate some captures
tempcapt <- sim.capthist(temptrans, popn = list(D = 2,
   buffer = 300), detectfn = 'HHN', binomN = 0,
   detectpar = list(lambda0 = 0.5, sigma = 50))

## snip capture histories
tempCH <- snip(tempcapt, by = 20)
```

```
## collapse from 'transect' to 'count', discarding location within transects
tempCH <- reduce(tempCH, outputdetector = "count")

## fit secr model and examine H-T estimates of density
## fails with detectfn = 'HN'
fit <- secr.fit(tempCH, buffer = 300, CL = TRUE, detectfn = 'HHN', trace = FALSE)
derived(fit)

## also, may split an existing transect into equal lengths
## same result:
plot(snip(temptrans, by = transectlength(temptrans)/10),
    markvertices = 1)
plot(snip(temptrans, length.out = 10), markvertices = 1)


## End(Not run)
```

`sort.capthist` *Sort Rows of capthist or mask Object*

Description

Rows are sorted by fields in covariates or by a provided sort key of length equal to the number of rows.

Usage

```
## S3 method for class 'capthist'
sort(x, decreasing = FALSE, by = "",
    byrowname = TRUE,...)

## S3 method for class 'mask'
sort(x, decreasing = FALSE, by = "",
    byrowname = TRUE,...)
```

Arguments

`x`	`capthist` object
`decreasing`	logical. Should the sort be increasing or decreasing?
`by`	character vector (names of covariates) or data frame whose columns will be used as sort keys
`byrowname`	logical. Should row name be used as a final sort key?
`...`	other arguments (not used)

Details

For multi-session `capthist` objects only the named covariate form is suitable as the number of rows varies between sessions.

If requested, rows are sorted by rowname within by. The effect of the defaultsis to sort by rowname.

The attribute markingpoints of a mask object is removed if present, as it is no longer meaningful.

Value

`capthist` or `mask` object with sorted rows; any relevant attributes are also sorted (covariates, signal, xy)

See Also

`capthist`

Examples

```
sort(ovenCH, by = "Sex")
covariates(ovenCH)[["2005"]]
covariates(sort(ovenCH, by = "Sex"))[["2005"]]
```

---

spacing — *Detector or Mask Spacing*

Description

Extract or replace the spacing attribute of a detector array or mask.

Usage

```
spacing(object, ...)
spacing(object) <- value

## S3 method for class 'traps'
spacing(object, ..., recalculate = FALSE)
## S3 method for class 'mask'
spacing(object, ..., recalculate = FALSE)
```

Arguments

`object`	object with 'spacing' attribute e.g. `traps`
`value`	numeric value for spacing
`...`	other arguments (not used)
`recalculate`	logical; if TRUE compute average spacing afresh

Details

The 'spacing' attribute of a detector array is the average distance from one detector to the nearest other detector.

The attribute was not always set by `make.grid()` and `read.traps()` in versions of **secr** before 1.5.0. If the attribute is found to be NULL then `spacing` will compute it on the fly.

Value

scalar numeric value of mean spacing, or a vector if `object` has multiple sessions

See Also

`traps`

Examples

```
temptrap <- make.grid(nx = 6, ny = 8)
spacing(temptrap)
```

`speed` *Speed Tips*

Description

A list of ways to make `secr.fit` run faster.

Use an appropriate mask

Check the extent and spacing of the habitat mask that you are using. Execution time is roughly proportional to the number of mask points (`nrow(mymask)`). Default settings can lead to very large masks for detector arrays that are elongated 'north-south' because the number of points in the east-west direction is fixed. Compare results with a much sparser mask (e.g., nx = 32 instead of nx = 64).

Use conditional likelihood

If you don't need to model variation in density over space or time then consider maximizing the conditional likelihood in secr.fit (CL = TRUE). This reduces the complexity of the optimization problem, especially where there are several sessions and you want session-specific density estimates (by default, derived() returns a separate estimate for each session even if the detection parameters are constant across sessions).

Model selection

Do you really need to fit all those complex models? Chasing down small decrements in AIC is so last-century. Remember that detection parameters are mostly nuisance parameters, and models with big differences in AIC may barely differ in their density estimates. This is a good topic for further research - we seem to need a 'focussed information criterion' (Claeskens and Hjort 2008) to discern the differences that matter. Be aware of the effects that can really make a difference: learned responses (b, bk etc.) and massive unmodelled heterogeneity.

Use score.test() to compare nested models. At each stage this requires only the more simple model to have been fitted in full; further processing is required to obtain a numerical estimate of the gradient of the likelihood surface for the more complex model, but this is much faster than maximizing the likelihood. The tradeoff is that the score test is only approximate, and you may want to later verify the results using a full AIC comparison.

Break problem down

Suppose you are fitting models to multiple separate datasets that fit the general description of 'sessions'. If you are fitting separate detection parameters to each session (i.e., you do not need to pool detection information), and you are not modelling trend in density across sessions, then it is much quicker to fit each session separately than to try to do it all at once. See Examples.

Mash replicated clusters of detectors

If your detectors are arranged in similar clusters (e.g., small square grids) then try the function `mash`.

Reduce sparse 'proximity' data to 'multi'

Full data from 'proximity' detectors has dimensions n x S x K (n is number of individuals, S is number of occasions, K is number of traps). If the data are sparse (i.e. multiple detections of an individual on one occasion are rare) then it is efficient to treat proximity data as multi-catch data (dimension n x S, maximum of one detection per occasion). Use `reduce(proxCH, outputdetector = "multi")`.

Use multiple cores when applicable

Most computers these days have multiple processors and these will be used by **secr** if the user sets `ncores` greater than one in `secr.fit`, `sim.secr` and some other functions. If `ncores = NULL` then the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS is used (see `setNumThreads`).

Avoid covariates with many levels

Categorical (factor) covariates with many levels and continuous covariates that take many values are not handled efficiently in secr.fit, and can dramatically slow down analyses and increase memory requirements.

Set appropriate typsize

Setting typsize manually in the call of ‘secr.fit‘ can speed up fitting when magnitudes on the link scale are very different (for example, when an identity link is used for density and density is very small or very large).

Simulations

Model fitting is not needed to assess power. The precision of estimates from secr.fit can be predicted without laboriously fitting models to simulated datasets. Just use `method = "none"` to obtain the asymptotic variance at the known parameter values for which data have been simulated (e.g. with sim.capthist()).

Suppress computation of standard errors by derived(). For a model fitted by conditional likelihood (CL = TRUE) the subsequent computation of derived density estimates can take appreciable time. If variances are not needed (e.g., when the aim is to predict the bias of the estimator across a large number of simulations) it is efficient to set se.D = FALSE in derived().

It is tempting to save a list with the entire ‘secr’ object from each simulated fit, and to later extract summary statistics as needed. Be aware that with large simulations the overheads associated with storage of the list can become very large. The solution is to anticipate the summary statistics you will want and save only these.

References

Claeskens, G. and Hjort N. L. (2008) *Model Selection and Model Averaging*. Cambridge: Cambridge University Press.

Examples

```
## Not run:

## compare timing of combined model with separate single-session models
## for 5-session ovenbird mistnetting data: 2977/78 = 38-fold difference

setNumThreads(7)

system.time(fit1 <- secr.fit(ovenCH, buffer = 300, trace = FALSE,
    model = list(D ~ session, g0 ~ session, sigma ~ session)))
##    user  system elapsed
## 1837.71   31.81  730.56

system.time(fit2 <- lapply (ovenCH, secr.fit, buffer = 300, trace = FALSE))
##   user  system elapsed
##  43.74    0.46   11.13

## ratio of density estimates
collate(fit1)[,1,1,"D"] / sapply(fit2, function(x) predict(x)["D","estimate"])
## session=2005 session=2006 session=2007 session=2008 session=2009
##    1.0000198    1.0000603    0.9999761    0.9999737    0.9999539


## End(Not run)
```

stoatDNA *Stoat DNA Data*

Description

Data of A. E. Byrom from a study of stoats (*Mustela erminea*) in New Zealand. Individuals were identified from DNA in hair samples.

Usage

```
stoatCH
stoat.model.HN
stoat.model.EX
```

Details

The data are from a pilot study of stoats in red beech (*Nothofagus fusca*) forest in the Matakitaki Valley, South Island, New Zealand. Sticky hair-sampling tubes ($n = 94$) were placed on a 3-km x 3-km grid with 500-m spacing between lines and 250-m spacing along lines. Tubes were baited with rabbit meat and checked daily for 7 days, starting on 15 December 2001. Stoat hair samples were identified to individual using DNA microsatellites amplified by PCR from follicular tissue (Gleeson et al. 2010). Six loci were amplified and the mean number of alleles was 7.3 per locus. Not all loci could be amplified in 27% of samples. A total of 40 hair samples were collected (Gleeson et al. 2010), but only 30 appear in this dataset; the rest presumably did not yield sufficient DNA for genotyping.

The data are provided as a single-session `capthist` object 'stoatCH'. Hair tubes are 'proximity' detectors which allow an individual to be detected at multiple detectors on one occasion (day), but there are no multiple detections in this dataset and for historical reasons the data are provided as detector type 'multi'. Two pre-fitted models are included: `stoat.model.HN` and `stoat.model.EX`.

Object	Description
stoatCH	capthist object
stoat.model.HN	fitted secr model – null, halfnormal detection function
stoat.model.EX	fitted secr model – null, exponential detection function

Note

The log-likelihood values reported for these data by `secr.fit` differ by a constant from those published by Efford et al. (2009) because the earlier version of DENSITY used in that analysis did not include the multinomial coefficient, which in this case is log(20!) or about +42.336. The previous analysis also used a coarser habitat mask than the default in **secr** (32 x 32 rather than 64 x 64) and this slightly alters the log-likelihood and ΔAIC values.

Fitting the hazard-rate detection function previously required the shape parameter z (or b) to be fixed, but the model can be fitted in **secr** without fixing z. However, the hazard rate function can cause problems owing to its long tail, and it is not recommended. The check on the buffer width, usually applied automatically on completion of secr.fit, causes an error and must be suppressed with biasLimit = NA (see Examples).

Gleeson et al. (2010) address the question of whether there is enough variability at the sampled microsatellite loci to distinguish individuals. The reference to 98 sampling sites in that paper is a minor error (A. E. Byrom pers. comm.).

Source

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

References

Gleeson, D. M., Byrom, A. E. and Howitt, R. L. J. (2010) Non-invasive methods for genotyping of stoats (*Mustela erminea*) in New Zealand: potential for field applications. *New Zealand Journal of Ecology* **34**, 356–359.

See Also

capthist, Detection functions, secr.fit

Examples

```
summary(stoatCH)

## Not run:

stoat.model.HN <- secr.fit(stoatCH, buffer = 1000, detectfn = 0)

# this generates an error unless we use biasLimit = NA
# to suppress the default bias check

stoat.model.EX <- secr.fit(stoatCH, buffer = 1000, detectfn = 2)
confint(stoat.model.HN, "D")
## Profile likelihood interval(s)...
##         lcl        ucl
## D 0.01275125 0.04055662


## End(Not run)

## plot fitted detection functions
xv <- seq(0,800,10)
plot(stoat.model.EX, xval = xv, ylim = c(0,0.12), limits = FALSE,
    lty = 2)
plot(stoat.model.HN, xval = xv, limits = FALSE, lty = 1, add = TRUE)
```

```
## review density estimates
collate(stoat.model.HN, stoat.model.EX,
    realnames = "D", perm = c(2,3,4,1))
modelAverage(stoat.model.HN, stoat.model.EX, realnames = "D")
```

`strip.legend` *Colour Strip Legend*

Description

This function is used with shaded plots to display a legend.

Usage

```
strip.legend(xy, legend, col, legendtype = c("breaks", "intervals", "other"),
  tileborder = NA, height = 0.5, width = 0.06, inset = 0.06, text.offset = 0.02,
  text.cex = 0.9, xpd = TRUE, scale = 1, title = "", box = NA, box.col = par()$bg)
```

Arguments

`xy`	location of legend (see Details)
`legend`	character vector (see Details)
`col`	vector of colour values
`legendtype`	character
`tileborder`	colour of lines around each tile in the colour strip. Use NA for none.
`height`	height of colour strip as a fraction of the plot dimensions
`width`	width of colour strip as a fraction of the plot dimensions
`inset`	spacing between legend and outside plot boundary, as a fraction of the plot dimensions
`text.offset`	spacing between colour strip and text, as a fraction of the plot dimensions
`text.cex`	size of text font
`xpd`	logical, if TRUE the legend will use the margins of the plot
`scale`	numeric; each value x will be displayed as scale * x
`title`	text displayed above legend
`box`	colour of frame, if framed, otherwise NA
`box.col`	colour of background, if framed, otherwise ignored

Details

The location of the legend is determined by xy which may be one of the character values "topright", "topleft", "bottomright", "bottomleft", "right", "left", or the x-y coordinates (in user units) of the top-left corner of the colour strip. Coordinates may be given as a vector or a list, and the output from locator(1) is suitable.

For more on colours, see notes in plot.mask and colors and terrain.colors

If legendtype = 'breaks' then labels are placed at the class boundaries; otherwise, the labels are centred vertically. If legendtype = 'breaks' or legendtype = 'intervals' then numeric values are extracted from the input, otherwise the text strings in legend are used as provided.

The legend itself may be provided as a vector of values or as the class labels output from plot.mask. Class labels are generated by cut in the form '(0,20]', '(20,40]', etc. These are parsed to construct either breaks (0,20,40,...) or intervals ('0-20', '20-40',...) as requested in the legendtype argument.

box may also be TRUE/FALSE; if TRUE the foreground colour is used par()$fg.

Value

Invisibly returns a vector of user coordinates for the left, right, bottom and top of the colour strip.

Note

From **secr** 2.9.0, the default behaviour of plot.mask is to call strip.legend to display a legend in the top right of the plot, labeled at breaks.

See Also

plot.mask

Examples

```
temptrap <- make.grid()
tempmask <- make.mask(temptrap)
covariates (tempmask) <- data.frame(circle =
    exp(-(tempmask$x^2 + tempmask$y^2)/10000) )
tmpleg <- plot (tempmask, covariate = "circle", dots = FALSE,
    breaks = 10, legend = FALSE)
strip.legend (xy = 'topright', col = terrain.colors(10),
    legend = tmpleg, title = "Test plot")

if (interactive()) {
    ## a custom axis using the returned values
    par(mar = c(2,2,2,6))
    plot (tempmask, covariate = "circle", dots = FALSE,
        breaks = 10, legend = FALSE)
    b <- strip.legend (locator(1), col = terrain.colors(10),
        legendtype = "other", legend = "    ", title = "Test plot",
         height = 0.3, box = NA)
    axis(side = 4, pos = b[2]+5, at = seq(b[4], b[3], length = 3),
        lab = seq(0,1,0.5), las = 1, tck = -0.02)
```

```
    par(mar = c(5,4,4,2) + 0.1)  ## reset to default
}
```

---

subset.capthist *Subset or Split capthist Object*

Description

Create a new capthist object or list of objects by selecting rows (individuals), columns (occasions) and traps from an existing capthist object.

Usage

```
## S3 method for class 'capthist'
subset(x, subset = NULL, occasions = NULL, traps = NULL,
    sessions = NULL, cutval = NULL, dropnullCH = TRUE, dropnullocc = FALSE,
    dropunused = TRUE, droplowsignals = TRUE, dropNAsignals = FALSE,
    cutabssignal = TRUE, renumber = FALSE, ...)

## S3 method for class 'capthist'
split(x, f, drop = FALSE, prefix = "S", bytrap = FALSE,
 byoccasion = FALSE, bysession = FALSE, ...)
```

Arguments

x	object of class capthist
subset	vector of subscripts to select rows (individuals) (see Details for variations)
occasions	vector of subscripts to select columns (occasions)
traps	vector of subscripts to select detectors (traps)
sessions	vector of subscripts to select sessions
cutval	new threshold for signal strength
dropnullCH	logical for whether null (all-zero) capture histories should be dropped
dropnullocc	logical for whether occasions with no detections should be dropped
dropunused	logical for whether never-used detectors should be dropped
droplowsignals	logical for whether cutval should be applied at each microphone rather than to sound as a whole
dropNAsignals	logical for whether detections with missing signal should be dropped
cutabssignal	logical for whether to apply cutval to absolute signal strength or the difference between signal and noise
renumber	logical for whether row.names should be replaced with sequence number in new capthist
f	factor or object that may be coerced to a factor

drop	logical indicating if levels that do not occur should be dropped (if f is a factor)
prefix	a character prefix to be used for component names when values of f are numeric
bytrap	logical; if TRUE then each level of f identifies traps to include
byoccasion	logical; if TRUE then each level of f identifies occasions to include
bysession	logical; if TRUE then each level of f identifies sessions of a multisession capthist to include
...	other arguments passed to subset.capthist (split.capthist) or to optional subset function (subset.capthist)

Details

Subscript vectors may be either logical- (length equal to the relevant dimension of x), character- or integer-valued. Subsetting is applied to attributes (e.g. covariates, traps) as appropriate. The default action is to include all animals, occasions, and detectors if the relevant argument is omitted.

When traps is provided, detections at other detectors are set to zero, as if the detector had not been used, and the corresponding rows are removed from traps. If the detector type is 'proximity' then selecting traps also reduces the third dimension of the capthist array.

split generates a list in which each component is a capthist object. Each component corresponds to a level of f. Multi-session capthists are accepted in **secr** >= 4.4.0; f should then be a list of factors with one component per session and the same levels in all.

To combine (pool) occasions use reduce.capthist. There is no equivalent of unlist for lists of capthist objects.

The effect of droplowsignals = FALSE is to retain below-threshold measurements of signal strength on all channels (microphones) as long as the signal is above cutval on at least one. In this case all retained sounds are treated as detected on all microphones. This fails when signals are already missing on some channels.

Subsetting is awkward with multi-session input when the criterion is an individual covariate. See the Examples for one way this can be tackled.

Value

capthist object with the requested subset of observations, or a list of such objects (i.e., a multi-session capthist object). List input results in list output, except when a single session is selected.

Warning

split.capthist does not work for mark–resight data.

See Also

capthist, rbind.capthist, reduce.capthist

Examples

```
tempcapt <- sim.capthist (make.grid(nx = 6, ny = 6), noccasions = 6)
summary(subset(tempcapt, occasions = c(1,3,5)))

## Consider `proximity' detections at a random subset of detectors
## This would not make sense for `multi' detectors, as the
## excluded detectors influence detection probabilities in
## sim.capthist.

tempcapt2 <- sim.capthist (make.grid(nx = 6, ny = 6,
    detector = "proximity"), noccasions = 6)
tempcapt3 <- subset(tempcapt2, traps = sample(1:36, 18,
    replace = FALSE))
summary(tempcapt3)
plot(tempcapt3)

tempcapt4 <- split (tempcapt2, f = sample (c("A","B"),
    nrow(tempcapt2), replace = TRUE))
summary(tempcapt4)

## Split out captures on alternate rows of a grid
tempcapt5 <- split(captdata, f = rep(1:2, 50), bytrap = TRUE)
summary(tempcapt5)

## Divide one session into two by occasion
tempcapt6 <- split(captdata, f = factor(c(1,1,2,2,2)), byoccasion = TRUE)
summary(tempcapt6)

## Applying a covariate criterion across all sessions of a
## multi-session capthist object e.g. selecting male ovenbirds from the
## 2005--2009 ovenCH dataset. We include a restriction on occasions
## to demonstrate the use of 'MoreArgs'. Note that mapply() creates a
## list, and the class of the output must be restored manually.

ovenCH.males <- mapply(subset, ovenCH,
    subset = lapply(ovenCH, function(x) covariates(x)$Sex == "M"),
    MoreArgs = list(occasions = 1:5))
class(ovenCH.males) <- class(ovenCH)
summary(ovenCH.males, terse = TRUE)

## A simpler approach using a function to define subset
subsetfn <- function(x, sex) covariates(x)$Sex == sex
ovenCH.males <- subset(ovenCH, subset = subsetfn, sex = "M")
summary(ovenCH.males, terse = TRUE)
```

---

subset.mask *Subset, Split or Combine Mask Objects*

Description

Retain selected rows of a `mask` object.

Usage

```
## S3 method for class 'mask'
subset(x, subset, ...)

## S3 method for class 'mask'
split(x, f, drop = FALSE, clusters = NULL, ...)

## S3 method for class 'mask'
rbind(...)
```

Arguments

`x`	`mask` object
`subset`	numeric or logical vector to select rows of mask
`f`	factor or object that may be coerced to a factor
`drop`	logical; if TRUE then levels that do not occur are dropped (if f is a factor)
`clusters`	list of traps objects, each defining a cluster (alternative to f)
`...`	two or more `mask` objects (rbind only)

Details

The subscripts in `subset` may be of type integer, character or logical as described in `Extract`.

The `split` method may use either a factor f with one value for each row or a list of clusters, each a traps object. The output mask corresponding to each cluster is the subset of the original mask points that lie within buffer of a trap within the cluster; buffer is computed as the maximum distance between a mask point in `x` and any detector in `clusters`. Sub-masks specified with `clusters` may overlap.

Covariates are ignored by `rbind.mask`.

Value

For `subset`, an object of class 'mask' with only the requested subset of rows and 'type' attribute set to 'subset'.

For `split`, a list of mask objects.

For `rbind`, an object of class 'mask' with all unique rows from the masks in . . . , and 'type' attribute set to 'rbind'.

Warning

The spacing attribute is carried over from the input (it is not updated automatically). In the case of very sparse masks (i.e. those with isolated points) this may lead to an unexpected value for this attribute. (Automatic updating requires excessive computation time and/or memory for very large masks).

See Also

mask

Examples

```
tempmask <- make.mask(make.grid())
OK <- (tempmask$x + tempmask$y) > 100
tempmask <- subset(tempmask, subset = OK)
plot(tempmask)
```

---

subset.popn *Subset popn Object*

Description

Retain selected rows of a popn object.

Usage

```
## S3 method for class 'popn'
subset(x, subset = NULL, sessions = NULL, poly = NULL,
    poly.habitat = TRUE, keep.poly = TRUE, renumber = FALSE, ...)
```

Arguments

x	popn object
subset	vector to subscript the rows of x
sessions	vector to subscript sessions if x is a multi-session population
poly	bounding polygon (see Details)
poly.habitat	logical for whether poly represents habitat or its inverse (non-habitat)
keep.poly	logical; if TRUE any bounding polygon is saved as the attribute 'polygon'
renumber	logical for whether to renumber rows in output
...	arguments passed to other functions

Details

The subscripts in subset may be of type integer, character or logical as described in Extract. By default, all rows are retained.

In the case of a multi-session popn object (a list of populations), subset may be a list with one component for the subscripts in each new session.

If poly is specified, points outside poly are dropped. poly may be one of the types descrbed in boundarytoSF.

Value

An object of class popn with only the requested subset of rows. Subsetting is applied to the covariates attribute if this is present. Attributes 'Ndist' and 'model2D' are set to NULL.

See Also

popn

Examples

```
temppop <- sim.popn (D = 10, expand.grid(x = c(0,100), y =
    c(0,100)), buffer = 50)
## 50% binomial sample of simulated population
temppops <- subset(temppop, runif(nrow(temppop)) < 0.5)
plot(temppop)
plot(temppops, add = TRUE, pch = 16)
```

---

subset.traps *Subset traps Object*

Description

Retain selected rows of a traps object.

Usage

```
## S3 method for class 'traps'
subset(x, subset = NULL, occasions = NULL, ...)
## S3 method for class 'traps'
split(x, f, drop = FALSE, prefix = "S", byoccasion = FALSE, ...)
```

Arguments

x	traps object
subset	vector to subscript the rows of x
occasions	vector to subscript columns in usage(x)
...	arguments passed to other functions or to optional subset function (subset.traps)
f	factor or object that may be coerced to a factor
drop	logical indicating if levels that do not occur should be dropped (if f is a factor)
prefix	a character prefix to be used for component names when values of f are numeric
byoccasion	logical ; if TRUE then f is used to split occasions

Details

The subscripts in `subset` may be of type integer, character or logical as described in `Extract`. By default, all rows are retained.

In the case of 'polygon' and 'transect' detectors, subsetting is done at the level of whole polygons or transects. `subset` should therefore have the same length as `levels(polyID(x))` or `levels(transectID(x))`.

`split` generates a list in which each component is a `traps` object. Each component corresponds to a level of `f`. The argument `x` of `split` cannot be a list (i.e. `x` must be a single-session traps object).

If the levels of `f` are numeric, from version 2.10.3 a leading zero is inserted in the names of the output list to maintain the sort order.

Value

An object of class `traps` with only the requested subset of rows. Subsetting is applied to `usage` and `covariates` attributes if these are present.

Splitting with `byoccasion = TRUE` produces a list of traps objects, each with usage codes for a subset of occasions. Traps not used on any occasion within a session are automatically dropped from that session.

Warning

`split.traps` does not work for mark–resight data.

See Also

`traps`, `rbind.traps`

Examples

```
## odd-numbered traps only, using modulo operator
temptrap <- make.grid(nx = 7, ny = 7)
t2 <- subset(temptrap, as.logical(1:nrow(temptrap) %% 2))
plot(t2)

## this works also for even number of rows, but must change 'outer' call
temptrap <- make.grid(nx = 8, ny = 8)
t3 <- subset(temptrap, !as.logical(outer(1:8,1:8,'+')%%2))
plot(t3)
```

`suggest.buffer` *Mask Buffer Width*

Description

Determines a suitable buffer width for an integration mask. The 'buffer' in question defines a concave polygon around a detector array constructed using `make.mask` with `type = "trapbuffer"`. The method relies on an approximation to the bias of maximum likelihood density estimates (M. Efford unpubl).

`bufferBiasCheck` is the post hoc test implemented in `secr.fit` when 'biasLimit' is not NA.

Usage

```
suggest.buffer(object, detectfn = NULL, detectpar = NULL,
    noccasions = NULL, ignoreusage = FALSE, ncores = NULL, RBtarget = 0.001,
    interval = NULL, binomN = NULL, ...)

bias.D (buffer, traps, detectfn, detectpar, noccasions, binomN = NULL,
    control = NULL)

bufferBiasCheck(object, buffer, biasLimit)
```

Arguments

`object`	single-session 'secr', 'traps' or 'capthist' object
`detectfn`	integer code or character string for shape of detection function 0 = halfnormal etc. – see detectfn
`detectpar`	list of values for named parameters of detection function – see detectpar
`noccasions`	number of sampling occasions
`ignoreusage`	logical for whether to discard usage information from `traps(capthist)`
`ncores`	integer number of threads to use for parallel processing
`RBtarget`	numeric target for relative bias of density estimate
`interval`	a vector containing the end-points of the interval to be searched
`binomN`	integer code for distribution of counts (see `secr.fit`)
`...`	other argument(s) passed to `bias.D`
`buffer`	vector of buffer widths
`traps`	'traps' object
`control`	list of mostly obscure numerical settings (see Details)
`biasLimit`	numeric threshold for predicted relative bias due to buffer being too small

Details

The basic input style of `suggest.buffer` uses a 'traps' object and a detection model specified by 'detectpar', 'detectfn' and 'noccasions', plus a target relative bias (RB). A numerical search is conducted for the buffer width that is predicted to deliver the requested RB. If `interval` is omitted it defaults to (1, 100S) where S is the spatial scale of the detection function (usually `detectpar$sigma`). An error is reported if the required buffer width is not within `interval`. This often happens with heavy-tailed detection functions (e.g., hazard-rate): choose another function, a larger `RBtarget` or a wider `interval`.

Setting `ncores = NULL` uses the existing value from the environment variable RCPP_PARALLEL_NUM_THREADS (see `setNumThreads`).

Convenient alternative input styles are –

- `secr` object containing a fitted model. Values of 'traps', 'detectpar', 'detectfn' and 'noccasions' are extracted from `object` and any values supplied for these arguments are ignored.

- capthist object containing raw data. If detectpar is not supplied then autoini is used to get 'quick and dirty' values of g0 and sigma for a halfnormal detection function. noccasions is ignored. autoini tends to underestimate sigma, and the resulting buffer also tends to be too small.

bias.D is called internally by suggest.buffer.

Value

suggest.buffer returns a scalar value for the suggested buffer width in metres, or a vector of such values in the case of a multi-session object.

bias.D returns a dataframe with columns buffer and RB.D (approximate bias of density estimate using finite buffer width, relative to estimate with infinite buffer).

bufferBiasCheck returns a logical value, FALSE if the predicted bias exceeds biasLimit.

Note

The algorithm in bias.D uses one-dimensional numerical integration of a polar approximation to site-specific detection probability. This uses a further 3-part linear approximation for the length of contours of distance-to-nearest-detector (r) as a function of r.

The approximation seems to work well for a compact detector array, but it should not be taken as an estimate of the bias for any other purpose: do *not* report RB.D as "the relative bias of the density estimate". RB.D addresses only the effect of using a finite buffer. The effect of buffer width on final estimates should be checked with mask.check.

The default buffer type in make.mask, and hence in secr.fit, is 'traprect', not 'trapbuffer', but a buffer width that is adequate for 'trapbuffer' is always adequate for 'traprect'.

control contains various settings of little interest to the user.

The potential components of control are –

method = 1 code for method of modelling *p.(X)* as a function of buffer (*q(r)*)

bfactor = 20 *q(r)* vs *p.(X)* calibration mask buffer width in multiples of trap spacing

masksample = 1000 maximum number of points sampled from calibration mask

spline.df = 10 effective degrees of freedom for smooth.spline

ncores = NULL integer number of cores

See Also

mask, make.mask, mask.check, esaPlot

Examples

```
## Not run:

temptraps <- make.grid()
detpar <-  list(g0 = 0.2, sigma = 25)
suggest.buffer(temptraps, "halfnormal", detpar, 5)
```

```
suggest.buffer(secrdemo.0)

suggest.buffer(ovenCH[[1]])

RB <- bias.D(50:150, temptraps, "halfnormal", detpar, 5)
plot(RB)

detpar <-  list(g0 = 0.2, sigma = 25, z=5)
RB <- bias.D(50:150, temptraps, "hazard rate", detpar, 5)
lines(RB)

## compare to esa plot
esaPlot (temptraps, max.buffer = 150, spacing = 4, detectfn = 0,
  detectpar = detpar, noccasions = 5, type = "density")

## compare detection histories and fitted model as input
suggest.buffer(captdata)
suggest.buffer(secrdemo.0)


## End(Not run)
```

---

summary.capthist *Summarise Detections*

Description

Concise description of capthist object.

Usage

```
## S3 method for class 'capthist'
summary(object, terse = FALSE, moves = FALSE, tpa = FALSE, ...)

## S3 method for class 'summary.capthist'
print(x, ...)

counts(CHlist, counts = "M(t+1)")
```

Arguments

object	capthist object
terse	logical; if TRUE return only summary counts
moves	logical; if TRUE then summary includes detected movements
tpa	logical; if TRUE then summary includes number of detectors per animal
x	summary.capthist object

...	arguments passed to other functions
CHlist	capthist object, especially a multi-session object
counts	character vector of count names

Details

These counts are reported by summary.capthist

n	number of individuals detected on each occasion
u	number of individuals detected for the first time on each occasion
f	number of individuals detected exactly f times
M(t+1)	cumulative number of individuals detected
losses	number of individuals reported as not released on each occasion
detections	number of detections, including within-occasion 'recaptures'
traps visited	number of detectors at which at least one detection was recorded
traps set	number of detectors, excluding any 'not set' in usage attribute of traps attribute

The last two rows are dropped if the data are nonspatial (object has no traps attribute).

Movements are as reported by moves. When terse = TRUE the number of non-zero moves is reported. The temporal sequence of detections at 'proximity' and 'count' detectors is not recorded in the capthist object, so the movement statistics are not to be taken too seriously. The problem is minimised when detections are sparse (seldom more than one per animal per occasion), and does not occur with 'single' or 'multi' detectors.

The 'tpa' option provides the frequency distribution of detectors per animal. When terse = TRUE the number of animals at >= 2 detectors is reported ('Animal2').

counts may be used to return the specified counts in a compact session x occasion table. If more than one count is named then a list is returned with one component for each type of count.

Value

From summary.capthist, an object of class summary.capthist, a list with at least these components

detector	detector type ("single", "multi", "proximity" etc.)
ndetector	number of detectors
xrange	range of x coordinates of detectors
yrange	range of y coordinates of detectors
spacing	mean distance from each trap to nearest other trap
counts	matrix of summary counts (rows) by occasion (columns). See Details.
dbar	mean recapture distance
RPSV	root pooled spatial variance

or, when `terse = TRUE`, a vector (single session) or dataframe (multiple sessions) of counts (Occasions, Detections, Animals, Detectors, and optionally Moves and Animals2).

A summary of individual covariates is provided if these are present (from **secr** 4.0.1).

A summary of interference/non-target captures is provided if there is a nontarget attribute (from **secr** 4.5.5).

See Also

dbar, RPSV, capthist

Examples

```
temptrap <- make.grid(nx = 5, ny = 3)
summary(sim.capthist(temptrap))
summary(sim.capthist(temptrap))$counts["n",]
summary(captdata, moves = TRUE)
```

`summary.mask` *Summarise Habitat Mask*

Description

Concise summary of a `mask` object.

Usage

```
## S3 method for class 'mask'
summary(object, ...)
## S3 method for class 'summary.mask'
print(x, ...)
```

Arguments

`object`	`mask` object
`x`	`summary.mask` object
`...`	other arguments (not used)

Details

The bounding box is the smallest rectangular area with edges parallel to the x- and y-axes that contains all points and their associated grid cells. A print method is provided for objects of class `summary.mask`.

Value

Object of class 'summary.mask', a list with components

detector	character string for detector type ("single","multi","proximity")
type	mask type ("traprect", "trapbuffer", "pdot", "polygon", "user", "subset")
nmaskpoints	number of points in mask
xrange	range of x coordinates
yrange	range of y coordinates
meanSD	dataframe with mean and SD of x, y, and each covariate
spacing	nominal spacing of points
cellarea	area (ha) of grid cell associated with each point
bounding box	dataframe with x-y coordinates for vertices of bounding box
covar	summary of each covariate

See Also

mask

Examples

```
tempmask <- make.mask(make.grid())
## left to right gradient
covariates (tempmask) <- data.frame(x = tempmask$x)
summary(tempmask)
```

---

summary.popn *Summarise Simulated Population*

Description

Concise summary of a popn object.

Usage

```
## S3 method for class 'popn'
summary(object, collapse = FALSE, ...)
## S3 method for class 'summary.popn'
print(x, ...)
```

Arguments

object	popn object
collapse	logical; if TRUE multi-session popn objects are treated as a single open population
x	summary.popn object
...	other arguments (not used)

Details

By default each component of a multisession object is summarised separately. If `collapse = TRUE` then turnover and movements are collated across sessions, matching individuals by rownames.

Value

For `summary.popn`, an object of class 'summary.popn' with various components. For a multisession object and collapse = TRUE the descriptors include the numbers of new individuals (recruits) and lost individuals (deaths), and matrices showing the status of each animal in each session ('status' codes 0 not recruited yet; 1 alive; -1 dead) and movement from previous session if alive then ('movements').

See Also

`sim.popn`

Examples

```
grid <- make.grid(8,8)
turnover <- list(phi = 0.8, lambda = 1)
pop <- sim.popn(Nbuffer = 200, core = grid, buffer = 200, Ndist = 'fixed',
    nsessions = 5, details = turnover)
summary(pop, collapse = TRUE)
```

---

summary.traps *Summarise Detector Array*

Description

Concise description of `traps` object.

Usage

```
## S3 method for class 'traps'
summary(object, getspacing = TRUE, covariates = FALSE, ...)
## S3 method for class 'summary.traps'
print(x, terse = FALSE, ...)
```

Arguments

`object`	`traps` object
`getspacing`	logical to calculate spacing of detectors from scratch
`covariates`	logical; if true each covariate is summarised
`x`	`summary.traps` object
`terse`	if TRUE suppress printing of usage and covariate summary
`...`	arguments passed to other functions

Details

When `object` includes both categorical (factor) covariates and `usage`, usage is tabulated for each level of the covariates.

Computation of `spacing` (mean distance to nearest trap) is slow and may hit a memory limit when there are many traps. In this case, turn off the computation with `getspacing` = FALSE.

Value

An object of class `summary.traps`, a list with elements

`detector`	`detector` type ("single", "multi", "proximity" etc.)
`ndetector`	number of detectors
`xrange`	range of x coordinates
`yrange`	range of y coordinates
`spacing`	mean distance from each trap to nearest other trap
`usage`	table of usage by occasion
`covar`	summary of covariates

See Also

`print`, `traps`

Examples

```
demo.traps <- make.grid()
summary(demo.traps)   ## uses print method for summary.traps object
```

---

timevaryingcov *Time-varying Covariates*

Description

Extract or replace time varying covariates

Usage

```
timevaryingcov(object, ...)
timevaryingcov(object) <- value
```

Arguments

`object`	an object of class `traps` or `capthist`
`value`	a list of named vectors
`...`	other arguments (not used)

Details

The timevaryingcov attribute is a list of one or more named vectors. Each vector identifies a subset of columns of covariates(object), one for each occasion. If character values are used they should correspond to covariate names.

In **secr** models, time-varying covariates are restricted to traps objects. Time-varying (session-specific) individual covariates may be used in **openCR**. The following remarks apply to time-varying traps covariates.

The name of the vector may be used in a model formula; when the model is fitted, the value of the trap covariate on a particular occasion is retrieved from the column indexed by the vector.

For replacement, if `object` already has a `usage` attribute, the length of each vector in `value` must match exactly the number of columns in `usage(object)`.

When converting a multi-session capthist object into a robust-design "single-session" object with function `join` the argument 'timevaryingcov' is used to collate covariate values across sessions in a form suitable for inclusion in **openCR** models (see `join`).

Value

`timevaryingcov(object)` returns the timevaryingcov attribute of `object` (may be NULL).

Note

It is usually better to model varying effort directly, via the usage attribute (see secr-varyingeffort.pdf).

Models for data from detectors of type 'multi', 'polygonX' or 'transectX' take much longer to fit when detector covariates of any sort are used.

Time-varying covariates are not available with the (default) 'fastproximity' option.

See secr-varyingeffort.pdf for input of detector covariates from a file.

See Also

`join`

Examples

```
# make a trapping grid with simple covariates
temptrap <- make.grid(nx = 6, ny = 8, detector = "multi")
covariates (temptrap) <- data.frame(matrix(
    c(rep(1,48*3),rep(2,48*2)), ncol = 5))
head(covariates (temptrap))

# identify columns 1-5 as daily covariates
timevaryingcov(temptrap) <- list(blockt = 1:5)
timevaryingcov(temptrap)

## Not run:

# default density = 5/ha, noccasions = 5
CH <- sim.capthist(temptrap, detectpar = list(g0 = c(0.15, 0.15,
    0.15, 0.3, 0.3), sigma = 25))
```

```
fit.1 <- secr.fit(CH, trace = FALSE)
fit.tvc2 <- secr.fit(CH, model = g0 ~ blockt, trace = FALSE)

# because variation aligns with occasions, we get the same with:
fit.t2 <- secr.fit(CH, model = g0 ~ tcov, timecov = c(1,1,1,2,2),
    trace = FALSE)

predict(fit.t2, newdata = data.frame(tcov = 1:2))
predict(fit.tvc2, newdata = data.frame(blockt = 1:2))

# now model some more messy variation
covariates (traps(CH))[1:10,] <- 3
fit.tvc3 <- secr.fit(CH, model = g0 ~ blockt, trace = FALSE)

AIC(fit.tvc2, fit.t2, fit.tvc3)
# fit.tvc3 is the 'wrong' model


## End(Not run)
```

`transformations` *Transform Point Array*

Description

Flip (reflect), rotate or slide (translate) an array of points. Methods are provided for 'traps' objects that ensure other attributes are retained. The methods may be used with `rbind.traps` to create complex geometries.

Usage

```
flip (object, lr = FALSE, tb = FALSE, ...)
rotate (object, degrees, centrexy = NULL, ...)
shift (object, shiftxy, ...)

## S3 method for class 'traps'
 flip(object, lr = FALSE, tb = FALSE, ...)
## S3 method for class 'traps'
rotate(object, degrees, centrexy = NULL, ...)
## S3 method for class 'traps'
shift(object, shiftxy, ...)

## S3 method for class 'popn'
 flip(object, lr = FALSE, tb = FALSE, ...)
## S3 method for class 'popn'
rotate(object, degrees, centrexy = NULL, ...)
## S3 method for class 'popn'
```

```
shift(object, shiftxy, ...)

## S3 method for class 'mask'
shift(object, shiftxy, ...)
```

Arguments

object	a 2-column matrix or object that can be coerced to a matrix
lr	either logical for whether array should be flipped left-right, or numeric value for x-coordinate of axis about which it should be flipped left-right
tb	either logical for whether array should be flipped top-bottom, or numeric value for y-coordinate of axis about which it should be flipped top-bottom
degrees	clockwise angle of rotation in degrees
centrexy	vector with xy coordinates of rotation centre
shiftxy	vector of x and y displacements
...	other arguments (not used)

Details

`flip` reflects points about a vertical or horizontal axis. Logical values for `lr` or `tb` indicate that points should be flipped about the mean on the relevant axis. Numeric values indicate the particular axis value(s) about which points should be flipped. The default arguments result in no change.

`shift` shifts the location of each point by the desired amount on each axis.

`rotate` rotates the array about a designated point. If `centrexy` is NULL then rotation is about (0,0) (`rotate.default`), the array centre (`rotate.traps`), or the centre of the bounding box (`rotate.popn`).

Value

A matrix or object of class 'traps' or 'popn' with the coordinates of each point transformed as requested.

See Also

`traps`, `popn`

Examples

```
temp <- matrix(runif (20) * 2 - 1, nc = 2)

## flip
temp2 <- flip(temp, lr = 1)
plot(temp, xlim=c(-1.5,4), ylim = c(-1.5,1.5), pch = 16)
points (temp2, pch = 1)
arrows (temp[,1], temp[,2], temp2[,1], temp2[,2], length = 0.1)
abline(v = 1, lty = 2)
```

```
## rotate
temp2 <- rotate(temp, 25)
plot(temp, xlim=c(-1.5,1.5), ylim = c(-1.5,1.5), pch = 16)
points (0,0, pch=2)
points (temp2, pch = 1)
arrows (temp[,1], temp[,2], temp2[,1], temp2[,2], length = 0.1)

## shiftxy
temp2 <- shift(temp, c(0.1, 0.1))
plot(temp, xlim=c(-1.5,1.5), ylim = c(-1.5,1.5), pch = 16)
points (0,0, pch=2)
points (temp2, pch = 1)
arrows (temp[,1], temp[,2], temp2[,1], temp2[,2], length = 0.1)

## flip.traps
par(mfrow = c(1,2), xpd = TRUE)
traps1 <- make.grid(nx = 8, ny = 6, ID = "numxb")
traps2 <- flip (traps1, lr = TRUE)
plot(traps1, border = 5, label = TRUE, offset = 7, gridl = FALSE)
plot(traps2, border = 5, label = TRUE, offset = 7, gridl = FALSE)
par(mfrow = c(1,1), xpd = FALSE)

## rotate.traps
hollow1 <- make.grid(nx = 8, ny = 8, hollow = TRUE)
nested <- rbind (hollow1, rotate(hollow1, 45, c(70, 70)))
plot(nested, gridlines = FALSE)

## shift.traps
hollow1 <- make.grid(nx = 8, ny = 8, hollow = TRUE)
hollow2 <- shift(make.grid(nx = 6, ny = 6, hollow = TRUE), c(20, 20))
nested <- rbind (hollow1, hollow2)
plot(nested, gridlines = FALSE, label = TRUE)
```

`trap.builder` *Complex Detector Layouts*

Description

Construct detector layouts comprising small arrays (clusters) replicated across space, possibly at a probability sample of points.

Usage

```
trap.builder (n = 10, cluster, region = NULL, frame = NULL,
    method = c("SRS", "GRTS", "all", "rank"),
   edgemethod = c("clip", "allowoverlap", "allinside", "anyinside", "centreinside"),
    samplefactor = 2, ranks = NULL, rotation = NULL, detector,
   exclude = NULL, exclmethod = c("clip", "alloutside", "anyoutside", "centreoutside"),
    plt = FALSE, add = FALSE, ...)
```

```
mash (object, origin = c(0,0), clustergroup = NULL, ...)

cluster.counts (object)

cluster.centres (object)
```

Arguments

n	integer number of clusters (ignored if method = "all")
cluster	traps object
region	bounding polygon(s)
frame	data frame of points used as a finite sampling frame
method	character string (see Details)
edgemethod	character string (see Details)
samplefactor	oversampling to allow for rejection of edge clusters (multiple of n)
ranks	vector of relative importance (see Details)
rotation	angular rotation of each cluster about centre (degrees)
detector	character detector type (see detector)
exclude	polygon(s) from which detectors are to be excluded
exclmethod	character string (see Details)
plt	logical: should array be plotted?
add	logical: add to existing plot
object	single-session multi-cluster capthist object, or traps object for cluster.centres
origin	new coordinate origin for detector array
clustergroup	list of vectors subscripting the clusters to be mashed
...	other arguments passed by trap.builder to spsurvey::grts (e.g., mindis) and by mash to make.capthist (e.g., sortrows)

Details

The detector array in cluster is replicated n times and translated to centres sampled from the area sampling frame in region or the finite sampling frame in frame. Each cluster may be rotated about its centre either by a fixed number of degrees (rotation positive), or by a random angle (rotation negative).

If the cluster argument is not provided then single detectors of the given type are placed according to the design.

The sampling frame is finite (the points in frame) whenever frame is not NULL. If region and frame are both specified, sampling uses the finite frame but sites may be clipped using the polygon.

region and exclude may be a two-column matrix or dataframe of x-y coordinates for the boundary, or one of the other polygon sources listed in boundarytoSF (these allow multiple polygons).

method may be "SRS", "GRTS", "all" or "rank". "SRS" takes a simple random sample (without replacement in the case of a finite sampling frame). "GRTS" takes a spatially representative sample using the 'generalized random tessellation stratified' (GRTS) method of Stevens and Olsen (2004). "all" replicates cluster across all points in the finite sampling frame. "rank" selects n sites from frame on the basis of their ranking on the vector 'ranks', which should have length equal to the number of rows in frame; ties are resolved by drawing a site at random.

Options for edgemethod are –

edgemethod	Description
"clip"	reject any individual detectors outside region
"allowoverlap"	no action
"allinside"	reject whole cluster if any component is outside region
"anyinside"	reject whole cluster if no component is inside region
"centreinside"	reject whole cluster if centre outside region, and clip to region

Similarly, exclmethod may be "clip" (reject individual detectors), "alloutside" (reject whole cluster if any component is outside exclude) etc. Sufficient additional samples ((samplefactor--1) * n) must be drawn to allow for replacement of any rejected clusters; otherwise, an error is reported ('not enough clusters within polygon').

GRTS samples require function grts in version >= 5.3.0 of package **spsurvey** (Dumelle et al. 2022). More sophisticated stratified designs may be specified by using grts directly.

mash collapses a multi-cluster capthist object as if all detections were made on a single cluster. The new detector coordinates in the 'traps' attribute are for a single cluster with (min(x), min(y)) given by origin. clustergroup optionally selects one or more groups of clusters to mash; if length(clustergroup) > 1 then a multisession capthist object will be generated, one 'session' per clustergroup. By default, all clusters are mashed.

mash discards detector-level covariates and occasion-specific 'usage', with a warning.

cluster.counts returns the number of *distinct* individuals detected per cluster in a single-session multi-cluster capthist object.

cluster.centres returns the centroid of the detector locations in each cluster. When clusters have been truncated these differ from the attribute centres set by make.systematic.

Value

trap.builder produces an object of class 'traps'.

plt = TRUE causes a plot to be displayed, including the polygon or finite sampling frame as appropriate.

mash produces a capthist object with the same number of rows as the input but different detector numbering and 'traps'. An attribute 'n.mash' is a vector of the numbers recorded at each cluster; its length is the number of clusters. An attribute 'centres' is a dataframe containing the x-y coordinates of the cluster centres. The predict method for secr objects and the function derived both recognise and adjust for mashing.

cluster.counts returns a vector with the number of individuals detected at each cluster.

cluster.centres returns a dataframe of x- and y-coordinates.

Note

The function make.systematic should be used to generate systematic random layouts. It calls trap.builder.

The sequence number of the cluster to which each detector belongs, and its within-cluster sequence number, may be retrieved with the functions clusterID and clustertrap.

References

Dumelle, M., Kincaid, T. M., Olsen, A. R., and Weber, M. H. (2021). spsurvey: Spatial Sampling Design and Analysis. R package version 5.2.0.

Stevens, D. L., Jr., and Olsen, A. R. (2004) Spatially-balanced sampling of natural resources. *Journal of the American Statistical Association* **99**, 262–278.

See Also

make.grid, traps, make.systematic, clusterID, clustertrap

Examples

```
## solitary detectors placed randomly within a rectangle
tempgrid <- trap.builder (n = 10, method = "SRS",
    region = cbind(x = c(0,1000,1000,0),
    y = c(0,0,1000,1000)), plt = TRUE)

## one detector in each 100-m grid cell -
## a form of stratified simple random sample
## see also Examples in ?make.grid

origins <- expand.grid(x = seq(0, 900, 100),
    y = seq(0, 1100, 100))
XY <- origins + runif(10 * 12 * 2) * 100
temp <- trap.builder (frame = XY, method = "all",
    detector = "multi")
## same as temp <- read.traps(data = XY)
plot(temp, border = 0)  ## default grid is 100 m

## Not run:

## simulate some data
## regular lattice of mini-arrays
minigrid <- make.grid(nx = 3, ny = 3, spacing = 50,
    detector = "proximity")
tempgrid <- trap.builder (cluster = minigrid , method =
    "all", frame = expand.grid(x = seq(1000, 5000, 2000),
    y = seq(1000, 5000, 2000)), plt = TRUE)
tempcapt <- sim.capthist(tempgrid, popn = list(D = 10))
cluster.counts(tempcapt)
cluster.centres(tempgrid)

## "mash" the CH
summary(mash(tempcapt))
```

```
## compare timings (estimates are near identical)
tempmask1 <- make.mask(tempgrid, type = "clusterrect",
    buffer = 200, spacing = 10)
fit1 <- secr.fit(tempcapt, mask = tempmask1, trace = FALSE)

tempmask2 <- make.mask(minigrid, spacing = 10)
fit2 <- secr.fit(mash(tempcapt), mask = tempmask2, trace = FALSE)
## density estimate is adjusted automatically
## for the number of mashed clusters (9)

predict(fit1)
predict(fit2)
fit1$proctime
fit2$proctime

## SRS excluding detectors from a polygon

region <- cbind(x = c(0,6000,6000,0,0), y = c(0,0,6000,6000,0))
exclude <- cbind(x = c(3000,7000,7000,3000,3000), y = c(2000,2000,4000,4000,2000))
newgrid <- trap.builder (n = 40, cluster = minigrid,
    method = "SRS", edgemethod = "allinside", region = region,
    exclude = exclude, exclmethod = "alloutside",
    plt = TRUE)

## two-phase design: preliminary sample across region,
## followed by selection of sites for intensive grids

arena <- data.frame(x = c(0,2000,2000,0), y = c(0,0,2500,2500))
t1 <- make.grid(nx = 1, ny = 1)
t4 <- make.grid(nx = 4, ny = 4, spacing = 50)
singletraps <- make.systematic (n = c(8,10), cluster = t1,
    region = arena)
CH <- sim.capthist(singletraps, popn = list(D = 2))
plot(CH, type = "n.per.cluster", title = "Number per cluster")
temp <- trap.builder(10, frame = traps(CH), cluster = t4,
    ranks = cluster.counts(CH), method = "rank",
    edgemethod = "allowoverlap", plt = TRUE, add = TRUE)

## GRTS sample of mini-grids within a rectangle
## GRTS uses package 'spsurvey' >= 5.3.0

minigrid <- make.grid(nx = 3, ny = 3, spacing = 50,
    detector = "proximity")
region <- cbind(x = c(0,6000,6000,0,0), y = c(0,0,6000,6000,0))

if (requireNamespace("spsurvey", versionCheck = list(version = ">=5.3.0"))) {

  tempgrid <- trap.builder (n = 20, cluster = minigrid, region = region,
      plt = TRUE, method = "GRTS")

  # specifying minimum distance between cluster origins
  tempgrid2 <- trap.builder (n = 20, cluster = minigrid, region = region,
```

```
      plt = TRUE, method = "GRTS", mindis = 500, maxtry = 10)
  # use spsurvey::warnprnt() to view warnings (e.g., maxtry inadequate)

}


## End(Not run)
```

---

traps *Detector Array*

Description

An object of class traps encapsulates a set of detector (trap) locations and related data. A method of the same name extracts or replaces the traps attribute of a capthist object.

Usage

```
traps(object, ...)
traps(object) <- value
```

Arguments

object	a capthist object.
value	traps object to replace previous.
...	other arguments (not used).

Details

An object of class traps holds detector (trap) locations as a data frame of x-y coordinates. Trap identifiers are used as row names. The required attribute 'detector' records the type of detector ("single", "multi" or "proximity" etc.; see detector for more).

Other possible attributes of a traps object are:

spacing	mean distance to nearest detector
spacex	
spacey	
covariates	dataframe of trap-specific covariates
clusterID	identifier of the cluster to which each detector belongs
clustertrap	sequence number of each trap within its cluster
usage	a traps x occasions matrix of effort (may be binary 0/1)
markocc	integer vector distinguishing marking occasions (1) from sighting occasions (0)
newtrap	vector recording aggregation of detectors by reduce.traps

If usage is specified, at least one detector must be 'used' (usage non-zero) on each occasion.

Various array geometries may be constructed with functions such as make.grid and make.circle, and these may be combined or placed randomly with trap.builder.

Note

Generic methods are provided to select rows (subset.traps), combine two or more arrays (rbind.traps), aggregate detectors (reduce.traps), shift an array (shift.traps), or rotate an array (rotate.traps).

The attributes usage and covariates may be extracted or replaced using generic methods of the same name.

References

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture-recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch and M. J. Conroy (eds) *Modeling Demographic Processes in Marked Populations*. Springer, New York. Pp. 255–269.

See Also

make.grid, read.traps, rbind.traps, reduce.traps, plot.traps, secr.fit, spacing, detector, covariates, trap.builder, as.mask

Examples

```
demotraps <- make.grid(nx = 8, ny = 6, spacing = 30)
demotraps    ## uses print method for traps
summary (demotraps)

plot (demotraps, border = 50, label = TRUE, offset = 8,
    gridlines=FALSE)

## generate an arbitrary covariate `randcov'
covariates (demotraps) <- data.frame(randcov = rnorm(48))

## overplot detectors that have high covariate values
temptr <- subset(demotraps, covariates(demotraps)$randcov > 0.5)
plot (temptr, add = TRUE,
    detpar = list (pch = 16, col = "green", cex = 2))
```

---

traps.info *Detector Attributes*

Description

Extract or replace attributes of an object of class ‘traps’.

Usage

```
polyID(object)
polyID(object) <- value
transectID(object)
transectID(object) <- value
searcharea(object)
transectlength(object)
```

Arguments

`object`	a 'traps' object
`value`	replacement value (see Details)

Details

The 'polyID' and 'transectID' functions assign and extract the attribute of a 'traps' object that relates vertices (rows) to particular polygons or transects. The replacement value should be a factor of length equal to nrow(object).

The 'searcharea' of a 'polygon' traps object is a vector of the areas of the component polygons in hectares. This value is read-only.

The 'transectlength' of a 'transect' traps object is a vector of the lengths of the component transects in metres. This value is read-only.

Value

`polyID` - a factor with one level per polygon. If the object does not have a polyID attribute then a factor with one level for each detector. `searcharea` - numeric value of polygon areas, in hectares. `transectlength` - numeric value of transect lengths, in metres.

See Also

`traps`

Examples

```
## default is a single polygon
temp <- make.grid(detector = "polygon", hollow = TRUE)
polyID(temp)
plot(temp)

## split in two
temp <- make.grid(detector = "polygon", hollow = TRUE)
polyID(temp) <- factor(rep(c(1,2),rep(10,2)))
plot(temp)
```

Trend *Density Trend*

Description

Functions for multi-session density trend analysis.

Usage

```
predictDlambda(object, alpha = 0.05)
```

Arguments

object	multi-session secr object output from secr.fit
alpha	alpha level for confidence intervals

Details

Usage is described in secr-trend.pdf. Briefly, setting details argument 'Dlambda' in 'secr.fit causes the density model (D~xxx) to be interpreted as a session-specific trend model with parameters for the initial density (D1) and each subsequent session-on-session change in density $\lambda[t] = D[t+1]/D[t]$.

Value

A table of session-specific estimates (initial D, subsequent $\lambda[t]$) with SE and confidence intervals.

See Also

predictDsurface, secr.fit

Examples

```
# a model with constant lambda
msk <- make.mask(traps(ovenCH[[1]]), buffer = 300, nx = 25)
fit <- secr.fit(ovenCH, model = D~1, mask = msk, trace = FALSE,
                details = list(Dlambda = TRUE), ncores = 2)
predictDlambda(fit)
```

trim *Drop Unwanted List Components*

Description

Drop unwanted components from a `list` object, usually to save space.

Usage

```
## Default S3 method:
trim(object, drop, keep)
## S3 method for class 'secr'
trim(object, drop = c("call", "mask", "designD", "designNE",
    "design", "design0"), keep = NULL)
## S3 method for class 'secrlist'
trim(object, drop = c("call", "mask", "designD", "designNE",
    "design", "design0"), keep = NULL)
```

Arguments

`object`	a list object
`drop`	vector identifying components to be dropped
`keep`	vector identifying components to be kept

Details

`drop` may be a character vector of names or a numeric vector of indices. If both `drop` and `keep` are given then the action is conservative, dropping only components in `drop` and not in `keep`.

Be warned that some further operations on fitted secr objects become impossible once you have discarded the default components.

Value

a list retaining selected components.

Examples

```
names(secrdemo.0)
names(trim(secrdemo.0))

object.size(secrdemo.0)
object.size(trim(secrdemo.0))

object.size(trim(secrlist(secrdemo.0, secrdemo.b)))
```

Troubleshooting *Problems in Fitting SECR Models*

Description

Although secr.fit is quite robust, it does not always work. Inadequate data or an overambitious model occasionally cause numerical problems in the algorithms used for fitting the model, or problems of identifiability, as described for capture–recapture models in general by Gimenez et al. (2004). Here are some tips that may help you.

This page has largely been superceded by secr-troubleshooting.pdf.

The log-likelihood values shown with trace = TRUE are all NA

Most likely you have infeasible starting values for the parameters. try some alternatives, specifying them manually with the start argument.

secr.fit finishes, but some or all of the variances are missing

This usually means the model did not fit and the estimates should not be trusted. Extremely large variances or standard errors also indicate problems.

- Try another maximization method (method = "Nelder-Mead" is more robust than the default). The same maximum likelihood should be found regardless of method, so AIC values are comparable across methods.
- Repeat the maximization with different starting values. You can use secr.fit(..., start = last.model) where last.model is a previously fitted secr object.
- If you think the estimates are right but there was a problem in computing the variances, try re-running secr.fit() with the previous model as starting values (see preceding) and set method = "none". This bypasses maximization and computes the variances afresh using fdHess from **nlme**.
- Try a finer mask (e.g., vary argument nx in make.mask). Check that the extent of the mask matches your data.
- The maximization algorithms work poorly when the beta coefficients are of wildly different magnitude. This may happen when using covariates: ensure beta coefficients are similar (within a factor of 5–10 seems adequate, but this is not based on hard evidence) by scaling any covariates you provide. This can be achieved by setting the typsize argument of nlm or the parscale control argument of optim.
- Examine the model. Boundary values (e.g., g0 near 1.0) may give problems. In the case of more complicated models you may gain insight by fixing the value of a difficult-to-estimate parameter (argument fixed).

See also Chapter 11 of The SECR book.

secr.fit finishes with warning nlm code 3

This condition does not invariably indicate a failure of model fitting. Proceed with caution, checking as suggested in the preceding section.

secr.fit crashes part of the way through maximization

A feature of the maximization algorithm used by default in `nlm` is that it takes a large step in the parameter space early on in the maximization. The step may be so large that it causes floating point underflow or overflow in one or more real parameters. This can be controlled by passing the 'stepmax' argument of `nlm` in the `...` argument of `secr.fit` (see first example). See also the previous point about scaling of covariates.

secr.fit demands more memory than is available

This is a problem particularly when using individual covariates in a model fitted by maximizing the conditional likelihood. The memory required is then roughly proportional to the product of the number of individuals, the number of occasions, the number of detectors and the number of latent classes (for finite-mixture models). When maximizing the full-likelihood, substitute 'number of groups' for 'number of individuals'. [The limit is reached in external C used for the likelihood calculation, which uses the R function 'R_alloc'.]

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

Estimates from mixture models appear unstable

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

References

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

See Also

`secr.fit`, `Memory-limits`

---

turnover *Specifying a Dynamic Population*

Description

`sim.popn` can simulate a multi-session population with known between-session survival, recruitment and movement probabilities. The parameter settings to achieve this are passed to `sim.popn` in its 'details' argument. Components of 'details' that are relevant to turnover are described below; see `sim.popn` for others.

Multi-session populations are generated in `sim.popn` whenever its argument 'nsessions' is greater than 1. If `details$lambda` remains NULL (the default) then the population for each successive session is generated de novo from the given density model (model2D, D etc.). If a value is specified for `details$lambda` then only the first population is generated de novo; remaining populations are generated iteratively with probabilistic mortality, recruitment and movement as described here.

Turnover components of `sim.popn` details argument

Component	Description	Default
`phi`	per capita survival rate ϕ	0.7
`survmodel`	probability model for number of survivors	“binomial"
`lambda`	finite rate of increase $\lambda = N_{t+1}/N_t$	none
`recrmodel`	probability model for number of recruits	“poisson"
`superN`	optional superpopulation size for ‘multinomial’ recruitment model	see below
`Nrecruits`	number of recruits to add at t+1 for ‘specified’ recruitment model	0
`movemodel`	“static", “uncorrelated", “normal", “exponential", “t2D" or a user function	“static"
`move.a`	first parameter of movement kernel (replacing sigma.m)	0
`move.b`	second parameter of movement kernel	1
`edgemethod`	treatment of animals that cross the boundary	“wrap"
`sigma.m`	deprecated in 3.2.1; use move.a	0
`wrap`	deprecated in 3.1.6; use edgemethod	TRUE i.e. edgemethod = “wrap"

Survival

Survival is usually thought of as a Bernoulli process (outcome 0 or 1 for each individual) so the number of survivors S is a binomial variable (survmodel = "binomial"). Another approach is to fix the proportion surviving, but this can be done exactly only when ϕN is an integer. A (slightly ad hoc) solution is to randomly choose between the two nearest integers with probability designed in the long term (over many sessions) to give the required ϕ (survmodel = "discrete").

Population growth and recruitment

Per capita recruitment (f) is the difference between lambda and phi ($f = \lambda - \phi$), which must be non-negative (phi > lambda causes an error). The number of recruits B is a random variable whose probability distribution is controlled by details$recrmodel:

Value	Probability model
"constantN"	Exact replacement of animals that die (B = $N_t - S$)
"binomial"	Binomial number of recruits (B ~ bin(N_t, f)
"poisson"	Poisson number of recruits (B ~ pois(fN_t))
"discrete"	Minimum-variance number of recruits (see Survival)
"multinomial"	The POPAN model, conditioning on superpopulation size (e.g., Schwarz and Arnason 1996))
"specified"	Add the number of recruits specified in Nrecruits (may be vector)

In the case of binomial recruitment there is a maximum of one recruit per existing individual, so lambda <= (phi+1). Multinomial recruitment requires a value for the superpopulation size. This may be provided as the details component "superN". If not specified directly, a value is inferred by projecting a trial initial (simulated) population using the specified phi and lambda.

Specifying the integer number of recruits in each year (recrmodel ‘specified’) overrides the value of lambda, but a non-null value should be given for lambda.

Movement

Individuals may shift their home range centre between sessions. Movement probability is governed by a circular kernel specified by 'movemodel' and the parameter values 'move.a' and 'move.b' (optional). By default there is no movement between sessions (movemodel = "static"). Other options are

"IND"	"uncorrelated"	individuals are randomly assigned a new, independent location within the buffered area
"BVN"	"normal"	bivariate normal (Gaussian) kernel with parameter move.a (previously called sigma.m)
"BVE"	"exponential"	negative exponential (Laplace) kernel with parameter move.a
"BVT"	"t2D"	circular 2-dimensional t-distribution with scale parameter move.a and shape parameter move.b = d
"RDE"		exponential distribution of radial distance (Ergon & Gardner, 2014)
"RDG"		gamma distribution of radial distance (Ergon & Gardner, 2014)
"RDL"		log-normal distribution of radial distance (Ergon & Gardner, 2014)
		(parameterized with move.a = exp(mu), move.b = 1/CV^2 = 1 / (exp(SD^2) - 1)

The package **openCR** >=1.4.0 provides functions for constructing and plotting these kernels and summarising their properties (`make.kernel`; `plot` and `summary` methods for kernel objects). The **secr** function `extractMoves` is useful for checking simulations of movement.

Models IND, BVN, BVE, and RDE may also be zero-inflated (suffix "zi"). The parameter 'move.a' (INDzi) or 'move.b' (BVNzi, BVEzi, RDEzi) is the zero-inflation probability. See Examples.

In **secr** <3.2.1 sigma.m was also used to indicate two special cases; these continue to work but may be discontinued in the future:

sigma.m = 0 corresponds to movemodel = 'static'

sigma.m < 0 corresponds to movemodel = 'uncorrelated'

In **secr** >= 4.4.0, the 'movemodel' component may also be a user-provided function with these characteristics: two or three arguments, the first being the number of centres to be moved (e.g., n) and the others parameters of the dispersal distribution (e.g., a,b); the function should return a matrix of n rows and 2 columns, the displacements in the x- and y-directions. The output is a set of random points from the bivariate dispersal kernel. The function will be called with the current number of centres and parameter values move.a and move.b as needed.

If movement takes an animal across the boundary of the arena (buffered area) in `sim.popn` the component "edgemethod" comes into play. By default, locations are toroidally wrapped i.e. the animal re-joins the population on the opposing edge. Other options are "clip" (discard), "clipandreplace" (assign new identity at original location), "stop" (stop just inside the boundary), "reflect" (bounce off edges to the limit of the dispersal), "normalize" = "truncate" (truncate kernel and scale probability to 1.0) and "none" (allow centres outside the buffered area). The "normalize" option (new in **secr** 4.3.3) can take longer as it repeatedly relocates each individual until its destination lies within the bounding box, up to a maximum of 500 attempts.

References

Clark, J. S, Silman, M., Kern, R., Macklin, E. and HilleRisLambers, J. (1999) Seed dispersal near and far: patterns across temperate and tropical forests. *Ecology* **80**, 1475–1494.

Nathan , R., Klein, E., Robledo-Arnuncio, J. J. and Revilla, E. (2012) Dispersal kernels: a review. In: J Clobert et al. *Dispersal Ecology and Evolution*. Oxford University Press. Pp 187–210.

See Also

sim.popn, extractMoves

Examples

```
par (mfrow = c(2,3), mar = c(1,1,1,1))

## birth and death only
grid <- make.grid(nx = 7, ny = 4, detector = 'proximity', spacing = 10)
pop <- sim.popn (Nbuffer = 100, core = grid, nsessions = 6,
    details = list(lambda = 0.8, phi = 0.6))
sapply(pop, nrow)  ## how many individuals?
plot(pop)

## movement only
pop2 <- sim.popn (Nbuffer = 100, core = grid, nsessions = 6,
    details = list(lambda = 1, phi = 1, movemodel = 'normal',
    move.a = 10, edgemethod = "wrap"))
pop3 <- sim.popn (Nbuffer = 100, core = grid, nsessions = 6,
    details = list(lambda = 1, phi = 1, movemodel = 'normal',
    move.a = 10, edgemethod = "clip"))
pop4 <- sim.popn (Nbuffer = 100, core = grid, nsessions = 10,
    details = list(lambda = 1, phi = 1, movemodel = 'normal',
    move.a = 10, edgemethod = "stop"))
sapply(pop2, nrow)  ## how many individuals?
plot(pop2)

## show effect of edgemethod --
## first session blue, last session red
cols <- c('blue',rep('white',4),'red')
par (mfrow=c(1,2))
plot(pop2, collapse = TRUE, seqcol = cols)
plot(pop3, collapse = TRUE, seqcol = cols)

## zero-inflated movement
## move.b is zero-inflation probability
pop5 <- sim.popn (Nbuffer = 1000, core = grid, nsessions = 6,
     details = list(lambda = 1, phi = 1, movemodel = 'RDEzi',
         move.a = 50, move.b = 0.5, edgemethod = "none"))
mean(do.call(rbind,extractMoves(pop5))$d)   # approx 50 * 0.5
```

---

updateCH *Update Old capthist Format*

Description

Before version 3.0, the internal data format for data from exclusive detectors (single, multi, proximityX, transectX) was a matrix with one row per detected animal and one column per sampling

occasion; each cell was either zero or the number of the detector at which the animal was detected (with switched sign if the animal died). The format for data from proximity and other detectors was a 3-dimensional array (third dimension corresponding to detectors) that allowed more than one detection per animal per occasion.

From secr 3.0 all capthist data use the 3-D format internally. This simplifies a lot of the coding, and enables mixing of detector types within a session. The constraint that only one detection is allowed per animal per occasion at exclusive detectors is imposed by verify().

The data input functions (read.capthist etc.) automatically generate objects in the new format. Objects created and saved under earlier versions should be converted if they relate to the 'exclusive' detector types listed above.

Usage

```
updateCH(object)
```

Arguments

`object`	capthist object

Details

The function reduce.capthist is applied with the nominal detector type as the outputdetector.

Value

Object with same class as the input.

Updating has the side effect of discarding invalid supernumerary detections (e.g. if there were two detections of an animal on one occasion, only one will be included).

Examples

```
# if we had the old ovenCH !
sapply(ovenCH, dim)
sapply(updateCH(ovenCH), dim)
```

`usage` *Detector Usage*

Description

Extract or replace usage (effort) information of a `traps` object (optional).

Usage

```
## S3 method for class 'traps'
usage(object, noccasions = NULL, ...)

usage(object) <- value
```

Arguments

`object`	`traps` object
`noccasions`	integer number of occasions (optional)
`value`	numeric matrix of detectors x occasions
`...`	other arguments (not used)

Details

In secr versions before 2.5.0, usage was defined as a binary value (1 if trap k used on occasion s, zero otherwise).

In later versions, usage may take nonnegative real values and will be interpreted as effort. This corresponds to the constant T_s used for the duration of sampling by Borchers and Efford (2008). Effort is modelled as a known linear coefficient of detection probability on the hazard scale (secr-varyingeffort.pdf; Efford et al. 2013).

For replacement of usage, various forms are possible for `value`:

- a matrix in which the number of rows of `value` exactly matches the number of traps *K* in `object`

- a vector of two values, the usage (typically 1) and the number of occasions *S* (a *K* x *S* matrix will be filled with the first value)

- a vector of *R+1* values where *R* is the number of sessions in a multi-session object and elements *2..R+1* correspond to the numbers of occasions *S1, S2,...* in each session

- the usage only (typically 1) (only works when replacing an existing usage matrix with known number of occasions).

Value

usage(object) returns the usage matrix of the `traps` object. `usage(object)` may be NULL. If `noccasions` is provided and there is no pre-existing matrix then a matrix of all ones will be generated.

Note

At present, assignment of usage to the traps objects of a multisession capthist object results in the loss of session names from the latter.

References

Efford, M. G., Borchers D. L. and Mowat, G. (2013) Varying effort in capture–recapture studies. *Methods in Ecology and Evolution* **4**, 629–636.

See Also

`traps`, `usagePlot`, `read.capthist`, `addSightings`

Examples

```
demo.traps <- make.grid(nx = 6, ny = 8)
## random usage over 5 occasions
usage(demo.traps) <- matrix (sample(0:1, 48*5, replace = TRUE,
    p = c(0.5,0.5)), nc = 5)
usage(demo.traps)
summary(demo.traps)

usage(traps(ovenCH)) <- c(1,9,10,10,10,10)
## restore lost names
names(ovenCH) <- 2005:2009
```

---

usagePlot *Plot usage, detections or sightings.*

Description

usagePlot displays variation in effort (usage) over detectors as a bubble plot (circles with radius scaled so that area is proportional to effort).

sightingPlot displays spatial variation in the number of sightings at each detector as a bubble plot (circles with radius scaled so that area is proportional to either the average number per occasion or the total over occasions.

Usage

```
usagePlot(object, add = FALSE, occasions = NULL, col = "black", fill =
FALSE, scale = 2, metres = TRUE, rad = 5, ...)

sightingPlot(object, type = c("Detections", "Tu", "Tm", "Tn", "Ta"), add = FALSE,
occasions = "ALL", mean = TRUE, col = "black", fill = FALSE, scale = 2,
metres = TRUE, dropunused = TRUE, title = type, legend = c(1, 2, 4, 8),
px = 0.95, py = 0.95, ...)
```

Arguments

object	traps object with usage attribute
add	logical; if FALSE plot.traps is called to create a base plot
occasions	integer number(s) of the occasion(s) for which effort is plotted, "ALL", or NULL
col	character or integer colour value
fill	logical; if TRUE the circle is filled with the line colour
scale	numeric value used to scale radius
metres	logical; if TRUE scale is a value in metres (see Details)

rad	numeric; radial displacement of symbol centre for each occasion from true detector location (metres)
...	other arguments passed to plot.traps
type	character to choose among sighting types and detections of marked animals
mean	logical; if TRUE then the plotted value is the average over occasions, otherwise the sum
dropunused	logical; if TRUE then detectors are omitted when they were unused on occasions
title	character
legend	numeric values for which legend circles will be drawn
px	legend x position as fraction of user coordinates
py	legend y position as fraction of user coordinates

Details

The behaviour of usagePlot is described first. By default (occasion = NULL) circles representing usage on each occasion are plotted around the detector location at distance rad, as in the petal plot of plot.capthist. Otherwise, the usage on a single specified occasion, or summed over occasions (length(occasion)>1, or occasion = "ALL"), is plotted as a circle centred at the detector location.

Package **sp** provides an alternative to usagePlot (see Examples).

sightingPlot may be used to display either detections of marked animals (whether or not occasions refers to sighting occasions) or any of the sighting attributes (unmarked sightings 'Tu', marked, unidentified sightings 'Tm', other uncertain sightings 'Tn', or the aggregate of sightings and detections 'Ta').

If py is of length 2 then the values determine the vertical spread of symbols in the legend.

For both functions –

The metres argument switches between two methods. If metres = TRUE, the symbols function is used with inches = FALSE to plot circles with radius scaled in the units of object (i.e. metres; scale is then the radius in metres of the symbol for a detector with usage = 1.0). Otherwise, plotting uses points; this has the advantage of producing better filled circles, but a suitable value of scale must be found by trial and error.

Value

No value is returned by usagePlot.

sightingPlot invisibly returns a 'traps' object with a covariate 'f' holding the plotted values.

See Also

usage, symbols, bubble, sightings

Examples

```
simgrid <- make.grid(nx = 10, ny = 10, detector = "proximity")
usage(simgrid) <- matrix(rep(1:10, 50), nrow = 100, ncol = 5)
usagePlot(simgrid, border = 20, scale = 1.5, fill = FALSE,
  metres = FALSE)

# It is hard to get the legend just right
# here is one attempt
legend (x = -50, y = 185, legend = c(1,2,5,10), pch = 1, pt.cex =
  c(1,2,5,10)^0.5 * 1.5, x.intersp = 3, y.intersp = 1.8, adj = 1,
  bty = "n", title = "Usage")

usagePlot(simgrid, occasion = NULL, border = 20, scale = 1.5, fill = FALSE,
  metres = FALSE)

## Not run:
# bubble plot in package 'sp'
library(sp)
simgrid$usage <- usage(simgrid)[,1]  ## occasion 1
class(simgrid) <- "data.frame"
coordinates(simgrid) <- c("x","y")
bubble(simgrid)

## End(Not run)
```

`userdist` *Non-Euclidean Distances*

Description

Non-Euclidean distances have a variety of uses, some obscure. You probably do not need them unless you have data from linear habitats, covered in the package **secrlinear**. On the other hand, they open up some intriguing possibilities for the advanced user. The key is to provide an appropriate value for the component 'userdist' of the `details` argument of `secr.fit`.

`details$userdist` is either a function to compute distances between detectors and mask points, or a pre-computed matrix of such distances. Pre-computing assumes the matrix is static (i.e. fixed and not dependent on any estimated coefficients). The functions `edist` and `nedist` are useful for computing static matrices of Euclidean or non-Euclidean distances (the latter is useful when there are barriers to movement).

If `details$userdist` is a function then it should take the form –

```
distfn(xy1, xy2, mask)
```

where the arguments are –

- **xy1** 2-column matrix of x-y coordinates of k detectors
- **xy2** 2-column matrix of x-y coordinates of m mask points
- **mask** habitat mask defining a non-Euclidean habitat geometry

Details

The matrix returned by the function must have exactly k rows and m columns. The function name may be almost anything you like.

The non-Euclidean habitat geometry may or may not require access to local density (D), local (mask) covariates, and the estimation of additional coefficients (beta variables). In order that secr.fit can assemble these data, there is a mechanism for the user to indicate which, if any, variables are required: when called with no arguments the function should return a character vector of variable names. These may include covariates of 'mask', the dynamically computed density 'D', and a special real parameter 'noneuc' for which one or more coefficients will be fitted.

'noneuc' is like 'D' in that it may be modelled as a function of any mask covariates, session, Session, x, y, etc. The actual meaning attributed to 'noneuc' depends entirely on how it is used inside the function.

The function may require no variables and not require estimation of additional coefficients. This is the case for a simple linear geometry as described in documentation for the package 'secrlinear'.

Value	Interpretation
''	no covariates etc. required
'D'	density at each mask point
'noneuc'	a multi-purpose real parameter defined for each mask point
c('D', 'noneuc')	both of the preceding
c('noneuc','habclass')	both noneuc and the mask covariate 'habclass'

The last case does not estimate a coefficient for habclass, it merely makes the raw value available to whatever algorithm you implement.

The 'xy2' and 'mask' parameters of the userdist function overlap in practice: xy1 and xy2 only define the points between which distances are required, whereas mask is a carrier for any and all additional information needed by the algorithm.

Full documentation of the **secr** capability for non-Euclidean distances is in the separate document secr-noneuclidean.pdf, which includes example code for the analysis of Sutherland et al. (2015).

Compatibility

User-specified distances are compatible with some but not all features of **secr**. Functions with a 'userdist' argument are certainly compatible, and others may be.

With a static userdist, `region.N` will generally not calculate population size for a region other than the original mask. If you want to supply a new mask in the 'region' argument, replace x$details$userdist with a distance matrix appropriate to the new mask, where 'x' is the name of the fitted model.

User-specified distances cannot be used with polygon or transect detectors.

When using `sim.capthist` to simulate detections of a new population from `sim.popn` you must provide `userdist` as a function rather than a matrix. This is because new animals are not restricted to locations on the 'mask' grid.

References

Sutherland, C., Fuller, A. K. and Royle, J. A. (2015) Modelling non-Euclidean movement and landscape connectivity in highly structured ecological networks. *Methods in Ecology and Evolution* **6**, 169–177.

See Also

details, secr.fit, nedist

Examples

```
## see secr-noneuclidean.pdf
```

---

utility *Utility Functions*

Description

Minor functions.

Usage

```
getMeanSD(xy)
maskarea(mask, sessnum = 1)
masklength(mask, sessnum = 1)
masksize(mask, sessnum = 1)
edist(xy1, xy2)
nedist(xy1, xy2, mask, inf = Inf, ...)
rlnormCV(n, mean, cv)
```

Arguments

xy	2-column matrix or dataframe
xy1	2-column matrix or dataframe
xy2	2-column matrix or dataframe
mask	mask or linearmask object
sessnum	integer; for multi-session masks, the number of the session
inf	numeric value to use for +infinity
...	other arguments for transition
n	number of observations
mean	mean on natural scale
cv	coefficient of variation on natural scale

Details

getmeanSD is used by make.mask to standardize mask coordinates.

For masklength the input should be a linear mask from **secrlinear**.

edist computes the Euclidean distance between each point in xy1 and each point in xy2. (This duplicates the functionality of 'rdist' in package **fields**).

nedist computes the non-Euclidean distance between each point in xy1 and each point in xy2, in two dimensions. The calculation uses **gdistance** (van Etten 2017; see also Csardi & Nepusz 2006): a transition layer is formed representing the connections between adjacent points in mask. By default, points within a 16-point neighbourhood are considered 'adjacent'. Distances are obtained by Dijkstra's (1959) algorithm as least cost paths through the graph of all points in the mask.

nedist has some subtle options. If 'mask' is missing then the transition layer will be formed from 'xy2'. If 'mask' has a covariate named 'noneuc' then this will be used to weight distances. The ... argument of nedist allows the user to vary arguments of transition (defaults transitionFunction = mean and directions = 16). Be warned this can lead to unexpected results! Point pairs that are completely separated receive the distance +Inf unless a finite value is provided for the argument 'inf'. See secr-noneuclidean.pdf for uses of nedist.

rlnormCV is a wrapper for rlnorm that computes its meanlog and sdlog arguments from the mean and CV on the natural scale:

$\mathrm{sdlog} = \sqrt{\log(\mathrm{CV}^2 + 1)}$, and

$\mathrm{meanlog} = \log(\mathrm{mean}) - \mathrm{sdlog}^2/2$.

Value

For getMeanSD, a dataframe with columns 'x' and 'y' and two rows, mean and SD.

For maskarea, the summed area of mask cells in hectares (ha).

For masklength, the summed length of mask cells in kilometers (km).

For masksize, whichever of area or length is appropriate.

For edist and nedist, a matrix with dim = c(nrow(xy1), nrow(xy2)).

For rlnormCV a vector of random deviates.

References

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, 269–271.

Csardi, G. and Nepusz, T. (2006) The igraph software package for complex network research. *InterJournal*, **1695**. https://igraph.org

van Etten, J. (2017) R package gdistance: Distances and routes on geographical grids. *Journal of Statistical Software*, **76(1)**, 1–21. doi:10.18637/jss.v076.i13

Examples

```
getMeanSD(possummask)
```

vcov.secr *Variance - Covariance Matrix of SECR Parameters*

Description

Variance-covariance matrix of beta or real parameters from fitted secr model.

Usage

```
## S3 method for class 'secr'
vcov(object, realnames = NULL, newdata = NULL,
    byrow = FALSE, ...)
```

Arguments

object	secr object output from the function secr.fit
realnames	vector of character strings for names of 'real' parameters
newdata	dataframe of predictor values
byrow	logical for whether to compute covariances among 'real' parameters for each row of new data, or among rows for each real parameter
...	other arguments (not used)

Details

By default, returns the matrix of variances and covariances among the estimated model coefficients (beta parameters).

If realnames and newdata are specified, the result is either a matrix of variances and covariances for each 'real' parameter among the points in predictor-space given by the rows of newdata or among real parameters for each row of newdata. Failure to specify newdata results in a list of variances only.

Value

A matrix containing the variances and covariances among beta parameters on the respective link scales, or a list of among-parameter variance-covariance matrices, one for each row of newdata, or a list of among-row variance-covariance matrices, one for each 'real' parameter.

See Also

vcov, secr.fit, print.secr

Examples

```
## previously fitted secr model
vcov(secrdemo.0)
```

`verify` *Check SECR Data*

Description

Check that the data and attributes of an object are internally consistent to avoid crashing functions such as `secr.fit`

Usage

```
## Default S3 method:
verify(object, report, ...)
## S3 method for class 'traps'
verify(object, report = 2, ...)
## S3 method for class 'capthist'
verify(object, report = 2, tol = 0.01, ...)
## S3 method for class 'mask'
verify(object, report = 2, ...)
```

Arguments

`object`	an object of class ‘traps’, ‘capthist’ or ‘mask’
`report`	integer code for level of reporting to the console. 0 = no report, 1 = errors only, 2 = full.
`tol`	numeric tolerance for deviations from transect line (m)
`...`	other arguments (not used)

Details

Checks are performed specific to the class of ‘object’. The default method is called when no specific method is available (i.e. class not ‘traps’, ‘capthist’ or ‘mask’), and does not perform any checks.

`verify.capthist`

1. No ‘traps’ component
2. Invalid ‘traps’ component reported by verify.traps
3. No live detections
4. Missing values not allowed in capthist
5. Live detection(s) after reported dead
6. Empty detection histories (except concurrent telemetry and all-sighting data)
7. More than one capture in single-catch trap(s)
8. More than one detection per detector per occasion at proximity detector(s)
9. Signal detector signal(s) less than threshold or invalid threshold
10. Number of rows in ‘traps’ object not compatible with reported detections

11. Number of rows in dataframe of individual covariates differs from capthist
12. Number of occasions in usage matrix differs from capthist
13. Detections at unused detectors
14. Number of coordinates does not match number of detections ('polygon', 'polygonX', 'transect' or 'transectX' detectors)
15. Coordinates of detection(s) outside polygons ('polygon' or 'polygonX' detectors)
16. Coordinates of detection(s) do not lie on any transect ('transect' or 'transectX' detectors)
17. Row names (animal identifiers) not unique
18. Inconsistencies in resighting data
19. Inconsistencies in telemetry data
20. Invalid non-target data
21. Levels of factor covariate(s) differ between sessions

`verify.traps`

1. Missing detector coordinates not allowed
2. Number of rows in dataframe of detector covariates differs from expected
3. Number of detectors in usage matrix differs from expected
4. Occasions with no used detectors
5. Polygons overlap
6. Polygons concave east-west ('polygon' detectors)
7. PolyID missing or not factor
8. Polygon detector is concave in east-west direction
9. Levels of factor trap covariate(s) differ between sessions

`verify.mask`

1. Valid x and y coordinates
2. Number of rows in covariates dataframe differs from expected
3. Levels of factor mask covariate(s) differ between sessions

Earlier errors may mask later errors: fix & re-run.

Value

A list with the component `errors`, a logical value indicating whether any errors were found. If `object` contains multi-session data then session-specific results are contained in a further list component `bysession`.

Full reporting is the same as 'errors only' except that a message is posted when no errors are found.

See Also

`capthist`, `secr.fit`, `shareFactorLevels`

Examples

```
verify(captdata)

## create null (complete) usage matrix, and mess it up
temptraps <- make.grid()
usage(temptraps) <- matrix(1, nr = nrow(temptraps), nc = 5)
usage(temptraps)[,5] <- 0
verify (temptraps)

## create mask, and mess it up
tempmask <- make.mask(temptraps)
verify(tempmask)
tempmask[1,1] <- NA
verify(tempmask)
```

`write.captures` *Write Data to Text File*

Description

Export detections or detector layout or mask to a text file in format suitable for input to DENSITY.

Usage

```
write.captures(object, file = "", deblank = TRUE, header = TRUE,
    append = FALSE, sess = "1", ndec = 2, covariates = FALSE, tonumeric
    = TRUE, ...)

write.traps(object, file = "", deblank = TRUE,  header = TRUE,
    ndec = 2, covariates = FALSE, ...)

write.mask(object, file = "", header = TRUE, ndec = 2, covariates = TRUE, ...)
```

Arguments

`object`	`capthist` or `traps` object
`file`	character name of output file
`deblank`	logical; if TRUE remove any blanks from character string used to identify detectors
`header`	logical; if TRUE output descriptive header
`append`	logical; if TRUE output is appended to an existing file
`sess`	character session identifier
`ndec`	number of digits after decimal point for x,y coordinates
`covariates`	logical or a character vector of covariates to export

tonumeric	logical for whether factor and character covariates should be converted to numeric values on output
...	other arguments passed to write.table

Details

Existing file will be replaced without warning if append = FALSE. In the case of a multi-session capthist file, session names are taken from object rather than sess.

write.capthist is generally simpler to use if you want to export both the capture data and trap layout from a capthist object.

By default individual covariates are not exported. When exported they are repeated for each detection of an individual. Factor covariates are coerced to numeric before export.

For write.mask, header = TRUE also causes column names to be exposed.

Value

None

See Also

as.data.frame.capthist

Examples

```
write.captures (captdata)
```

---

writeGPS *Upload to GPS*

Description

Upload a set of point locations as waypoints to a GPS unit connected by USB or via a serial port. Intended primarily for detector locations in a traps object. Uses the GPSBabel package which must have been installed. Coordinates are first inverse-projected to latitude and longitude using function st_transform from **sf**.

Usage

```
writeGPS(xy, o = "garmin", F = "usb:", proj = "+proj=nzmg")
```

Arguments

xy	2-column matrix or dataframe of x-y coordinates
o	character output format (see GPSBabel documentation)
F	character for destination (see Details)
proj	character string describing projection

Details

This function is derived in part from `readGPS` in **maptools**.

For users of Garmin GPS units, useful values of `o` are "garmin" for direct upload via USB or serial ports, and "gdb" for a file in Mapsource database format.

`F` may be "usb:" or "com4:" etc. for upload via USB or serial ports, or the name of a file to create.

The `proj` argument may be complex. For further information see the Examples and the vignette secr-spatialdata.pdf. If `proj` is an empty string then coordinates are assumed already to be latitudes (column 1) and longitudes (column 2).

Waypoint names are derived from the rownames of `xy`.

Value

No value is returned. The effect is to upload waypoints to an attached GPS or file.

Note

GPSBabel is available free online. Remember to add it to the path. On Windows this means following something like Settings > Control panel > System > Advanced settings > Environment variables > (select Path) Edit and adding ";C:/Program Files (x86)/gpsbabel" to the end (without the quotes).

See Also

`make.systematic`

Examples

```
## Example using shapefile "possumarea.shp" in
## "extdata" folder. As 'cluster' is not specified,
## the grid comprises single multi-catch detectors.

## Not run:

## test for availability of GPSBabel

if (nzchar(Sys.which("gpsbabel"))) {

library(sf)
shpfilename <- system.file("extdata/possumarea.shp", package = "secr")
possumarea <- st_read(shpfilename)

possumgrid <- make.systematic(spacing = 100, region = possumarea,
    plt = TRUE)

## May upload directly to GPS...
# writeGPS(possumgrid, proj = "+proj=nzmg")

## ...or save as Mapsource file
writeGPS(possumgrid, o = "gdb", F = "tempgrid.gdb",
```

```
    proj = "+proj=nzmg")

## If `region' had been specified in another projection we
## would need to specify this as in Proj.4. Here is a
## hypothetical example for New Zealand Transverse Mercator
## with datum NZGD2000 (EPSG:2193)

NZTM <- paste("+proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996",
    "+x_0=1600000 +y_0=10000000 +ellps=GRS80",
    " +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")

# writeGPS(possumgridNZTM, o = "gdb", F = "tempNZTM.txt",
#    proj = NZTM)

## Or to upload coordinates from UTM Zone 18 in eastern
## Maryland, USA...

# writeGPS(MarylandUTMgrid, proj =
#    "+proj=utm +zone=18 +ellps=WGS84")

}


## End(Not run)
```

Index

[.secrlist (AIC.secr), 16

addCovariates, 8, *93*
addSightings, 10, *13*, *15*, *262*, *329*
addTelemetry, 12, *61*, *64*, *139*, *172*, *216, 217*, *226*
adjustVarD, *79*, *82*, *94*, *196*
adjustVarD (chat), 34
AIC, *19*, *235*
AIC.secr, 16, *20*, *120*, *125*, *151, 152*, *201*, *247*
AIC.secrlist (AIC.secr), 16
AICcompatible, *17–19*, 20, *124, 125*
alive (capthist.parts), 33
all.equal, *20*
alongtransect (capthist.parts), 33
animalID (capthist.parts), 33
append.capthist (rbind.capthist), 207
ARL (homerange), 105
arrows, *89*
as.array (as.data.frame), 21
as.data.frame, 21
as.data.frame.capthist, *340*
as.mask, 22, *131*, *319*
as.popn, 23
attenuationplot (plot.secr), 183
autoini, 24, *108*, *166*, *244*, *255*, *303*
axis, *186*

bias.D, *167*, *245*
bias.D (suggest.buffer), 301
binCovariate, 26, *91*
blackbear, 27
blackbearCH (blackbear), 27
boundarytoSF, *58*, *133, 134*, *171*, *186*, *272*, *299*, *314*
boundarytoSF (Internal), 113
bubble, *331*
buffer.contour (secr-deprecated), 237
bufferBiasCheck (suggest.buffer), 301
bufferContour, *131*, *171, 172*, *190*, *238*, *270*
bufferContour (contour), 49
BUGS, 29

capped (detector), 71
captdata, *7*
captdata (secrdemo), 252
capthist, *8*, *15*, *25*, *30*, 31, *31*, *34*, *41*, *43, 44*, *106*, *110*, *123*, *126, 127*, *141*, *153*, *160*, *163*, *175*, *193*, *199*, *208*, *221*, *223*, *243*, *247*, *253*, *259*, *263*, *268*, *282*, *287*, *292*, *296*, *304*, *306*, *338*
capthist.parts, 33
captXY (secrdemo), 252
centroids (homerange), 105
chat, 34
chat.nj, *35*, *79*, *82*, *94*, *156*
chat.nj (chat), 34
chat.nk, *255*
chat.nk (nk), 154
circular, 36
clip.hex, *138*
clip.hex (make.tri), 139
clone, 39
closedN, 40, *228*
closure.test, 43, *43*, *57*
cluster, *34*, *36*, 44, *80*
cluster.centres, *45*, *135*
cluster.centres (trap.builder), 313
cluster.counts, *45*
cluster.counts (trap.builder), 313
clusterID, *186*, *316*, *318*
clusterID (cluster), 44
clusterID<- (cluster), 44
clusterSetRNGStream, *255*
clustertrap, *316*, *318*
clustertrap (cluster), 44
clustertrap<- (cluster), 44
coef.secr, 45, *66*
collate, 46, *152*
colors, *294*

colours, *179*, *186*
confint.secr, 48, *55*, *166*
contour, 49, *97*, *121*, *178*, *179*
Coulombe (housemouse), 111
count (detector), 71
counts (summary.capthist), 304
covariates, *27*, 52, *282*, *318*, *319*
covariates<- (covariates), 52
cut, *26*, *177*, *294*
cutree, *222*, *223*
CV, 53, *170*, *231*
CVa (CV), 53
CVa0 (CV), 53
CVpdot, *54*, *85*, *86*
CVpdot (pdot), 169

D.designdata, 55, *113*, *242*
dbar, *25*, *306*
dbar (homerange), 105
deermouse, 56
defunct (secr-defunct), 235
deleteMaskPoints, 57, *131*
derived, *14*, *54*, 59, *62*, *63*, *80*, *82*, *201*, *228*, *247*
derived.secr, *35*, *166*
derivedCluster, *45*
derivedCluster (empirical.varD), 79
derivedDcoef, *14*
derivedDcoef (derivedDsurface), 62
derivedDfit (derivedDsurface), 62
derivedDsurface, *14*, 62, *196*, *197*
derivedExternal (empirical.varD), 79
derivedMash (empirical.varD), 79
derivednj (empirical.varD), 79
derivedSession (empirical.varD), 79
derivedSystematic (secr-defunct), 235
details, *14*, *54*, 64, *107*, *118*, *232*, *244*, *247*, *265*, *334*
detectfn, *24*, *37*, *38*, *50*, 69, *85*, *141*, *184*, *232*, *243*, *265*, *266*, *302*
detectfnplot, *38*, *71*
detectfnplot (plot.secr), 183
Detection functions, *247*
Detection functions (detectfn), 69
detector, *69*, 71, *110*, *139*, *140*, *214*, *218*, *219*, *222*, *305*, *309*, *314*, *318*, *319*
detector<- (detector), 71
detectpar, *184*, *239*, *302*
detectpar (predict.secr), 194
deviance, 72
deviance.secr, *19*, *252*
df.residual (deviance), 72
Dfn2 (Internal), 113
discretize, 74, *223*, *285*
distancetotrap, 76
Dsurface, 77, *178*, *179*
Dsurface-class (Dsurface), 77

edist, *332*
edist (utility), 334
effort (usage), 328
ellipse.bvn (ellipse.secr), 78
ellipse.secr, 78
empirical.varD, *62*, 79
Enk (nk), 154
esa, *24*, *82*, *238*
esa (derived), 59
esa.plot (secr-deprecated), 237
esa.secr, *166*
esaPlot, *46*, 84, *142*, *143*, *145*, *167*, *237*, *238*, *303*
expected.n, *35*, 87, *167*, *228*
Extract, *298*, *299*, *301*
extractMoves, 89, *326*, *327*

factor, *261*
FAQ, 90
fdHess, *60*, *61*, *233*, *234*
fillNA, *9*, 92
fixedbeta (details), 64
Fletcher.chat, *35*, *36*, 93, *155*, *156*
flip (transformations), 311
formula.gam, *283*, *284*
FTHL.0 (hornedlizard), 109
FTHL.CL (hornedlizard), 109
fx.total (secr-deprecated), 237
fxi, 95, *98*, *99*, *147*, *166*, *237*, *238*
fxi.contour (secr-deprecated), 237
fxi.mode (secr-deprecated), 237
fxiContour, *99*, *237*, *238*
fxiContour (fxi), 95
fxiMode, *237*, *238*
fxiMode (fxi), 95
fxTotal, *97*, 98, *237*, *238*

getMeanSD (utility), 334
gridCells, 100, *187*
GSM (blackbear), 27

hclust, *222, 223*
hcov, 101, *111*, *244*, *247*
head, 104, *105*
hist.MCgof, *150*
hist.MCgof *(*plot.MCgof*)*, 180
homerange, 105
hornedlizard, 109
hornedlizardCH, *31*
hornedlizardCH *(*hornedlizard*)*, 109
housemouse, 111

identical, *20*
infraCH *(*skink*)*, 280
insertdim *(*secr.design.MS*)*, 240
integrate, *37*
interference *(*nontarget*)*, 157
Internal, 113
intervals, 114
intervals<- *(*intervals*)*, 114
invlogit *(*logit*)*, 122
ip.secr *(*secr-defunct*)*, 235

join, *64*, 115, *115*, *229*, *310*

kfn, 117

legend, *178*
lineoCH *(*skink*)*, 280
list.secr.fit, 119, *167*, *237, 238*, *245*, *247*
LLonly *(*details*)*, 64
LLsurface, 120
LLsurface.secr, *166*
locator, *294*
logit, 122
logLik.secr *(*AIC.secr*)*, 16
logmultinom, 123
LR.test, *19*, 124, *235*
LStraps *(*skink*)*, 280

make.capthist, *33*, 125, *213, 214*
make.circle, *318*
make.circle *(*make.traps*)*, 136
make.grid, *100*, *128*, *135*, *140*, *219*, *316*, *318, 319*
make.grid *(*make.traps*)*, 136
make.lacework, 128, *135*
make.lookup *(*secr.design.MS*)*, 240
make.mask, *9*, *23*, *36*, *50, 51*, *58*, *76*, *85, 86*, 129, *132, 133*, *143*, *145*, *156*, *170*, *192*, *204*, *226*, *228*, *303*, *323*, *335*
make.newdata *(*secr-defunct*)*, 235
make.poly *(*make.traps*)*, 136
make.spcosa, 132
make.systematic, *129, 130*, *133*, 134, *139*, *315, 316*, *341*
make.telemetry, *15*
make.telemetry *(*make.traps*)*, 136
make.transect *(*make.traps*)*, 136
make.traps, 136
make.tri, *138*, 139, *139*
makeCluster, *155*
makeNewData, *194–196*
makeNewData *(*newdata*)*, 153
makeStart, 141, *249*
mapply, *119*
marked, *14, 15*
marked *(*sighting*)*, 261
marked<- *(*sighting*)*, 261
markocc, *10, 11*, *13*, *15*, *137*, *245*, *318*
markocc *(*sighting*)*, 261
markocc<- *(*sighting*)*, 261
mash, *45*, *80*, *197*, *226*, *289*, *324*
mash *(*trap.builder*)*, 313
mask, *8*, *23*, *25*, *33*, *55*, *85, 86*, *131*, *141*, 142, *153*, *179*, *203, 204*, *216*, *232*, *243, 244*, *247*, *270*, *299*, *301*, *303*, *307*
mask-class *(*mask*)*, 142
mask.check, *86*, *142*, 143, *143*, *166*, *303*
maskarea *(*utility*)*, 334
masklength *(*utility*)*, 334
masksize *(*utility*)*, 334
maxdistance *(*details*)*, 64
MCgof, 146, *181, 182*, *239*, *250*, *252*
miscparm *(*details*)*, 64
MMDM *(*homerange*)*, 105
model.average *(*secr-defunct*)*, 235
model.matrix, *65*, *240–242*
modelAverage, *18–20*, *47*, 150, *152*, *236*
moves, *305*
moves *(*homerange*)*, 105
ms, 152
MS.capthist, *32, 33*, *117*
MS.capthist *(*rbind.capthist*)*, 207
mtext, *186*
multi *(*detector*)*, 71
Multi-core processing *(*Parallel*)*, 166

ncores *(*Parallel*)*, 166
nearesttrap *(*distancetotrap*)*, 76

nedist, *332*, *334*
nedist (utility), 334
newdata, 153
nk, 154
nlm, *96*, *245*
noise (signal), 262
noise<- (signal), 262
noneuc (userdist), 332
nontarget, 157, *306*
nontarget<- (nontarget), 157

occasion (capthist.parts), 33
occasionKey, 158, *175*
optim, *245*
ORL (homerange), 105
ovenbird, *7*, 159, *162*, *163*
ovenCH (ovenbird), 159
ovenCHp (ovenbird), 159
ovenmask (ovenbird), 159
ovensong, 162, *264*
over, *190*
OVpossum, 164
OVpossumCH (OVpossum), 164

palettes, *179*
par, *186*
par.derived, *61*, *167*
par.derived (secr-deprecated), 237
par.region.N, *167*
par.region.N (secr-deprecated), 237
par.secr.fit, *167*
par.secr.fit (secr-deprecated), 237
Parallel, *150*, 166, *256*, *260*
parallel, *167*
param (details), 64
pdot, *50*, *51*, *86*, *87*, *98*, *99*, *130*, *131*, *142*, *166*, 169, *239*
pdot.contour (secr-deprecated), 237
pdotContour, *97*, *170*, *237*, *238*
pdotContour (contour), 49
persp, *178*, *179*
pfn (secr-defunct), 235
PG, 171
pgamma, *70*
plogis, *122*
plot, *185*, *186*
plot.capthist, *157*, *159*, 173, *331*
plot.Dsurface, *77*, *99*, *197*, *220*
plot.Dsurface (plot.mask), 177
plot.mask, *23*, 177, *294*
plot.MCgof, *150*, 180
plot.popn, 182, *191*, *273*
plot.Rsurface, *232*
plot.Rsurface (plot.mask), 177
plot.secr, 183
plot.secrlist (plot.secr), 183
plot.secrtest, *252*
plot.secrtest (secrtest), 256
plot.traps, *139*, *175*, 185, *319*
plotMaskEdge, *100*, 187
plotMCP (plot.capthist), 173
pmixProfileLL, 188
points, *181*
pointsInPolygon, *114*, *172*, 189
polyarea, 190
polygon (detector), 71
polygonX (detector), 71
polyID, *34*
polyID (traps.info), 319
polyID<- (traps.info), 319
popn, *23*, *183*, 191, *210*, *265*, *268*, *273*, *300*, *312*
population size (region.N), 225
possum, 192
possumarea (possum), 192
possumCH (possum), 192
possummask (possum), 192
possumremovalarea (possum), 192
predict.secr, *35*, *62*, *118*, *120*, *151*, *154*, 194, *197*, *230*, *247*
predict.secrlist (predict.secr), 194
predictDlambda, *114*
predictDlambda (Trend), 321
predictDsurface, *62*, *63*, *77*, 196, *196*, *321*
print, *199*, *202*, *309*
print.capthist, 199
print.default, *199*, *202*
print.Dsurface (Dsurface), 77
print.MCgof (plot.MCgof), 180
print.secr, *19*, *62*, 200, *247*, *336*
print.secrtest, *252*
print.secrtest (secrtest), 256
print.summary.capthist (summary.capthist), 304
print.summary.mask (summary.mask), 306
print.summary.popn (summary.popn), 307
print.summary.traps (summary.traps), 308

print.traps, *139*, 202
proximity (detector), 71

qlogis, *122*

random numbers (secrRNG), 254
randomDensity (randomHabitat), 202
randomHabitat, 202, *255*, *273*
rank, *251*
rast, *206*
rast (raster), 205
rast,Dsurface-method (raster), 205
rast,mask-method (raster), 205
raster, 205, *206*
raster,Dsurface-method (raster), 205
raster,mask-method (raster), 205
rbind.capthist, *33*, *117*, 207, *267*, *296*
rbind.mask (subset.mask), 297
rbind.popn, 210, *276*
rbind.traps, 211, *301*, *311*, *319*
read.capthist, *8*, *10*, *11*, *13*, *33*, *126*, *127*, 212, *217*, *253*, *329*
read.DA (BUGS), 29
read.mask, *9*, *131*, *143*, 215
read.SPACECAP (secr-defunct), 235
read.table, *10*, *11*, *214*
read.telemetry, *13*, *15*, *213*, *214*, 216
read.traps, *9*–*11*, *139*, *214*, 217, *319*
rectangularMask, *179*, 219
reduce, 220
reduce.capthist, *33*, *75*, *110*, *117*, 221, *221*, *279*, *296*
reduce.traps, *221*, *318*, *319*
reduce.traps (reduce.capthist), 221
region.N, *14*, *42*, *43*, *61*, *88*, *166*, 225, *247*
rlnorm, *335*
rlnormCV (utility), 334
RMarkInput, 228
RNG, *204*, *254*
RNGkind, *148*
rnorm, *254*
rotate (transformations), 311
rotate.traps, *319*
RPSV, *24*, *306*
RPSV (homerange), 105
RSE, *54*, 230
RShowDoc, *72*
Rsurface, 231
runif, *254*

safeLL (details), 64
sample, *254*
save, *91*
saveprogress, *249*
saveprogress (details), 64
score.table (score.test), 233
score.test, *19*, *55*, *125*, *166*, 233, *244*
searcharea (traps.info), 319
secr, *36*, *156*, *170*, *185*
secr (secr-package), 6
secr-defunct, 235, *240*
secr-deprecated, 237, *240*
secr-package, 6
secr-version5, 238, *238*
secr.design.MS, *55*, 240, *246*
secr.fit, *6*, *8*, *14*, *17*, *19*, *20*, *24*, *25*, *33*, *46*, *47*, *50*, *55*, *62*, *68*, *73*, *85*, *103*, *113*, *118*–*120*, *127*, *141*–*143*, *145*, *152*–*155*, *166*, *169*, *188*, *196*, *197*, *201*, *216*, *228*, *240*, 242, *248*, *249*, *255*, *266*, *276*, *277*, *288*, *292*, *301*, *302*, *319*, *321*, *324*, *332*, *334*, *336*, *338*
secr.make.newdata (secr-defunct), 235
secr.refit, *64*, *67*, *245*, *247*, 248, *248*
secr.test, *147*, *150*, *167*, 250, *255*–*257*, *277*
secrdemo, 252
secrlist, *119*, *147*
secrlist (AIC.secr), 16
secrRNG, *204*, 254
secrtest, 256
seed (secrRNG), 254
segments, *187*
session, *115*, 258
session<- (session), 258
sessionlabels (intervals), 114
sessionlabels<- (intervals), 114
set.seed, *254*, *256*
setNumThreads, *25*, *48*, *60*, *87*, *91*, *96*, *98*, *121*, *148*, *155*, *166*, *170*, *226*, *234*, *245*, *251*, 259, *276*, *289*, *302*
setThreadOptions, *260*
shareFactorLevels, 260, *338*
shift (transformations), 311
shift.traps, *319*
sighting, 261
sightingPlot, *175*, *262*
sightingPlot (usagePlot), 330

sightings, *331*
sightings (sighting), 261
signal, 262
signal<- (signal), 262
signalCH (ovensong), 162
signalframe (signal), 262
signalframe<- (signal), 262
signalmatrix, *34*, 263, *263*
sim.capthist, *24*, *30*, *33*, *127*, *255*, *256*, 264, *277*, *279*, *333*
sim.detect (sim.secr), 275
sim.popn, *23*, *40*, *89*, *183*, *191*, *202*, *204*, *255*, *256*, *265*, *268*, 269, *276*, *308*, *324*, *327*, *333*
sim.resight, *11*, *255*, *256*
sim.resight (sim.capthist), 264
sim.secr, *73*, *167*, *250*, *252*, *255*, 275
simOU, *70*
simOU (simOU.capthist), 278
simOU.capthist, 278
simulate, *255*, *256*, *266*, *268*, *273*, *277*
simulate (sim.secr), 275
simulate.secr, *252*, *255*, *256*
single (detector), 71
skink, *27*, *93*, 280
smooths, 282
snip, *74*, *75*, 284
sort.capthist, 286
sort.mask (sort.capthist), 286
spacing, *100*, 287, *318*, *319*
spacing<- (spacing), 287
speed, 288
Speed tips, *91*, *247*
Speed tips (speed), 288
split.capthist, *117*
split.capthist (subset.capthist), 295
split.mask (subset.mask), 297
split.traps (subset.traps), 300
spotHeight, *232*
spotHeight (plot.mask), 177
st_crs, *100*, *113*
st_transform, *340*
stoat.model.EX (stoatDNA), 291
stoat.model.HN (stoatDNA), 291
stoatCH (stoatDNA), 291
stoatDNA, *124*, 291
strip.legend, *178*, *179*, 293
subset, *272*
subset.capthist, *33*, *116*, *208*, *223*, 295
subset.mask, *58*, *131*, 297
subset.popn, 299
subset.traps, *211*, 300, *319*
suggest.buffer, *142*, *143*, *167*, 301
summary, *157*
summary.capthist, *157*, 304
summary.Dsurface (Dsurface), 77
summary.mask, 306
summary.MCgof, *150*
summary.MCgof (plot.MCgof), 180
summary.popn, 307
summary.secr (print.secr), 200
summary.traps, 308
symbols, *331*
Sys.getenv, *260*

t2r2 (homerange), 105
tail, *105*
tail.capthist (head), 104
tail.Dsurface (head), 104
tail.mask (head), 104
tail.traps (head), 104
telemetered, *15*
telemetered (capthist.parts), 33
telemetrytype (addTelemetry), 12
telemetrytype<- (addTelemetry), 12
telemetryxy, *13*, *15*
telemetryxy (capthist.parts), 33
telemetryxy<- (capthist.parts), 33
terrain.colors, *294*
text, *158*
tile (sim.popn), 269
timevaryingcov, *53*, *116*, *244*, 309
timevaryingcov<- (timevaryingcov), 309
Tm, *11*, *268*
Tm (sighting), 261
Tm<- (sighting), 261
Tn, *11*
Tn (sighting), 261
Tn<- (sighting), 261
trans3d, *179*
transect (detector), 71
transectID (traps.info), 319
transectID<- (traps.info), 319
transectlength, *285*
transectlength (traps.info), 319
transectX (detector), 71
transformations, *191*, 311

transition, *334, 335*
trap *(*capthist.parts*)*, 33
trap.builder, *34*, *45*, *130*, *133*, *135*, *139*, 313, *318, 319*
traps, *8*, *23*, *33*, *45*, *72*, *127*, *133*, *138–140*, *186*, *202*, *211*, *219*, *262*, *268*, *288*, *301*, *308, 309*, *312*, *316*, 318, *320*, *329*
traps object *(*traps*)*, 318
traps.info, 319
traps<- *(*traps*)*, 318
trapsPerAnimal *(*homerange*)*, 105
trapXY *(*secrdemo*)*, 252
Trend, 321
trim, *144*, *276*, 322
Troubleshooting, *90*, *247*, 323
Tu, *11*, *268*
Tu *(*sighting*)*, 261
Tu<- *(*sighting*)*, 261
turnover, *89*, *182*, *272, 273*, 324

uniroot, *25*, *48, 49*
unjoin *(*join*)*, 115
unRMarkInput *(*RMarkInput*)*, 228
updateCH, *32*, 327
usage, *25*, *50*, *91*, *155*, *157*, *170*, *244*, *247*, *262*, *310*, *318*, 328, *331*
usage<- *(*usage*)*, 328
usagePlot, *329*, 330
uselog *(*details*)*, 64
userdist, *68*, *155*, *169*, *247*, 332
utility, 334

vcov, *336*
vcov.secr, *66*, *247*, 336
verify, *10*, *30, 31*, *137*, *157*, *207*, *212*, *243*, *245*, *247*, 337
verify.capthist, *12*, *145*, *261*
version5 *(*secr-version5*)*, 238

write.capthist, *340*
write.capthist *(*read.capthist*)*, 212
write.captures, *214*, 339
write.DA *(*BUGS*)*, 29
write.mask *(*write.captures*)*, 339
write.SPACECAP *(*secr-defunct*)*, 235
write.traps, *214*
write.traps *(*write.captures*)*, 339
writeGPS, 340

xy *(*capthist.parts*)*, 33
xy2CH *(*addTelemetry*)*, 12
xy<- *(*capthist.parts*)*, 33