Package 'secr'

June 17, 2011

Type Package

Title Spatially explicit capture-recapture

Version 2.1.0

Depends R (>= 2.12.0), abind, utils

Suggests MASS, nlme, sp, maptools, gpclib, spsurvey, rgdal, rgeos

Date 2011-06-17

Author Murray Efford

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

Description Functions to estimate the density 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.

License GPL (>=2)

LazyLoad yes

LazyData yes

LazyDataCompression xz

URL http://www.otago.ac.nz/density

R topics documented:

`secr-package` *Spatially Explicit Capture-Recapture Models*

Description

Functions to estimate the density 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:	2.1.0
Date:	2011-06-17
License:	GNU General Public License Version 2 or later

Data comprise the locations of detectors (traps, searched areas, etc.) 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 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 by maximizing either the full likelihood or the likelihood conditional on the number of individuals observed (n). Conditional likelihood models are limited to homogeneous Poisson density, but allow continuous individual covariates for detection. Fitting (function `secr.fit`) creates an object of class `secr`. Generic methods (plot, print, summary, etc.) are provided for each object class.

A link at the bottom of this and every help page takes you to the help index. Several vignettes complement the help pages:

`../doc/secr-overview.pdf`	general introduction
`../doc/secr-datainput.pdf`	data formats and input functions
`../doc/secr-polygondetectors.pdf`	using polygon and transect detector types
`../doc/secr-sound.pdf`	analysing data from microphone arrays
`../doc/secr-finitemixtures.pdf`	mixture models for individual heterogeneity

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. Feedback on the software is also welcome, including suggestions for additional documentation or new features consistent with the overall design.

Acknowledgements

David Borchers made these methods possible with his work on the likelihood, and I'm grateful for his continuing advice. Jeff Laake provided encouragement and reviewed an early version. Ray

Brownrigg got my Windows code running under Unix. Deanna Dawson edited some of the documentation (the cleaner bits!) and her support and collaboration were important throughout. Tiago Marques suggested many improvements to the help pages.

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.

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 or transect searches. Unpublished manuscript.

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.

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.

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)
```

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.

Usage

```
## S3 method for class 'secr'
AIC(object, ..., sort = TRUE, k = 2, dmax = 10)
## S3 method for class 'secrlist'
AIC(object, ..., sort = TRUE, k = 2, dmax = 10)
## S3 method for class 'secr'
logLik(object, ...)
```

Arguments

`object`	`secr` object output from the function `secr.fit`, or a list of such objects with class c('list','secrlist')
`...`	other `secr` objects
`sort`	logical for whether rows should be sorted by ascending AICc
`k`	numeric, the penalty per parameter to be used; always k = 2 in this method
`dmax`	numeric, the maximum AIC difference for inclusion in confidence set

Details

Models to be compared must have been fitted to the same data and use the same likelihood method (full vs conditional).

AIC with small sample adjustment is given by

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

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

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

Models for which dAICc > `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 `model.average` (see also Buckland et al. 1997, Burnham and Anderson 2002).

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

Value

A data frame with one row per model. By default, rows are sorted by ascending AICc.

`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)
`dAICc`	difference between AICc of this model and the one with smallest AICc
`AICwt`	AICc model weight

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

Note

It is not be meaningful to compare models by AIC if they relate to different data or habitat masks. For example, 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. Likewise, models fitted by the conditional likelihood (`CL = TRUE`) and full likelihood (`CL = FALSE`) methods cannot be compared.

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

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.

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

See Also

`model.average`, `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)
```

autoini *Initial Parameter Values for SECR*

Description

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

Usage

```
autoini(capthist, mask, detectfn = 0, thin = 0.2)
```

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

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.

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.

A proportion 1–thin of the points in the mask may be discarded at random to speed execution.

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 may in future include an option to use RPSV instead of dbar.

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

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

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; 'NA' where not detected
U2	matrix of y-coordinates, dimensioned as Y; 'NA' where not detected

References

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

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')
str(lzdata)
## Now convert to capthist
tempcapt <- read.DA(lzdata, Y = 'H', xcoord = 'X',
    ycoord = 'Y', buffer = 0, units = 100)
summary(tempcapt)
setwd(olddir)

## Not run:
plot(tempcapt)
secr.fit(tempcapt)
## 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`.

For 'single' and 'multi' detectors, `capthist` is a matrix with one row per animal and one column per occasion (i.e. dim(capthist) = c(nc, noccasions)); each element is either zero (no detection) or a detector number. For other detectors ('proximity', 'count', 'signal' etc.), `capthist` is an array of values and dim(capthist) = c(nc, noccasions, ntraps); 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)
- signal – signal strength values, one per detection ('signal' only)
- detectedXY – dataframe of coordinates for location within polygon ('polygon' only)

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`.

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)
occasion(object)
trap(object, names = TRUE)
alive(object)
alongtransect(object, tol = 0.01)
xy(object)
```

```
xy(object) <- value
signal(object)
signal(object) <- value
unmarked(object)
unmarked(object) <- value
```

Arguments

`object`	a 'capthist' object
`names`	if FALSE the values returned are numeric indices rather than names
`tol`	tolerance for snapping to transect line (m)
`value`	replacement value (see Details)

Details

These functions extract data on detections, ignoring occasions when an animal was not detected. Detections are ordered by occasion, animalID and trap.

`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.

`unmarked` extracts or replaces the counts of unmarked animals that are saved as attribute 'Tu' of `object`.

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'.

For `unmarked`, a matrix with one row per detector and one column for each occasion. The number of occasions should match the number of columns in `object` (noting that `object` has no rows).

For `signal`, 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`, `polyID`

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))
```

`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).

Usage

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

circular.p (r = 1, detectfn = 0, sigma = 1, detectpar = NULL)
```

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, 1 = hazard rate etc. – see detectfn
`sigma`	spatial scale parameter of detection function
`detectpar`	named list of detection function parameters

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).

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 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).

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.

See Also

`detectfn`, `detectfnplot`

Examples

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

## halfnormal, hazard-rate and exponential
circular.r ()
circular.r (detectfn = 1, detectpar = list(sigma = 1, z = 4))
circular.r (detectfn = 2)

plot(seq(0, 5, 0.01), circular.p(r = seq(0, 5, 0.01)),
    type = 'l', xlab = 'Radius (multiples of sigma)', ylab = 'Probability')
lines(seq(0, 5, 0.01), circular.p(r = seq(0, 5, 0.01), detectfn = 2),
    type = 'l', col = 'red')
lines(seq(0, 5, 0.01), circular.p(r = seq(0, 5, 0.01), 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.
```

`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

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; please see Chao and Shen (2010) for details.

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. and Shen, T.-J. (2010) Program SPADE (Species Prediction And Diversity Estimation). Program and User's Guide available online at `http://chao.stat.nthu.edu.tw`.

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.

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

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

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`.

Value

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

`clusterID(object)` may be NULL.

See Also

`traps`, `trap.builder`, `derived.cluster`, `cluster.counts`, `cluster.centres`

Examples

```
## 81 4-detector clusters
mini <- make.grid(nx = 2, ny = 2)
tempgrid <- trap.builder (cluster = mini , method = 'all',
    frame = expand.grid(x = seq(100, 900, 100), y = seq(100,
    900, 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`, `esa.plot`

Examples

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

`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, ...)
```

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
`...`	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.

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. `buffer.contour` adds a conventional ‘boundary strip’ to a detector (trap) array, where `buffer` equals the strip width.

Usage

```
pdot.contour(traps, border = NULL, nx = 64, detectfn = 0,
    detectpar = list(g0 = 0.2, sigma = 25, z = 1), noccasions = 5,
    binomN = NULL, levels = seq(0.1, 0.9, 0.1), poly = NULL, plt = TRUE,
    add = FALSE, ...)

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

Arguments

`traps`	`traps` object
`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 c ode 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)
`plt`	logical to plot contours
`add`	logical to add contour(s) to an existing plot
`...`	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

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

if `convex = FALSE`, `buffer.contour` constructs a mask and contours the points on the basis of distance to the nearest trap at the levels given in `buffer`.

if `convex = TRUE`, `buffer.contour` 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, 2009).

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`.

Note

The precision (smoothness) of the fitted line in `buffer.contour` 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).

References

Efford, M. G. (2009) *DENSITY 4.4: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand http://www.otago.ac.nz/density.

See Also

`pdot`, `make.mask`

Examples

```
possumtraps <- traps(possumCH)

plot(possumtraps, border = 270)
pdot.contour(possumtraps, detectfn = 0, nx = 128, detectpar =
    detectpar(possum.model.1), levels = c(0.1, 0.01, 0.001),
    add = TRUE)

## clipping to polygon
olddir <- setwd(system.file("extdata", package = "secr"))
possumarea <- read.table('possumarea.txt', header = TRUE)
oldpar <- par(xpd = TRUE, mar = c(1,6,6,6))
plot(possumtraps, border = 400, gridlines = FALSE)
pdot.contour(possumtraps, detectfn = 0, nx = 256, detectpar =
    detectpar(possum.model.1), levels = c(0.1, 0.01, 0.001),
    add = TRUE, poly = possumarea, col = 'blue')
lines(possumarea)
setwd(olddir)
par(oldpar)

## convex and concave buffers
plot(possumtraps, border = 270)
buffer.contour(possumtraps, buffer = 100, add = TRUE, col = 'blue')
buffer.contour(possumtraps, buffer = 100, convex = TRUE, add = TRUE)
```

```
## areas
buff.concave <- buffer.contour(possumtraps, buffer = 100,
    plt = FALSE)
buff.convex  <- buffer.contour(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 <- buffer.contour(possumtraps, buffer = 100,
    nx = 128, plt = FALSE)
sum (sapply(buff.concave2, polyarea))
```

`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.

Examples

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

D.designdata *Construct Density Design Data*

Description

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

Usage

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

Arguments

`mask`	`mask` object.
`Dmodel`	formula for density model
`grps`	vector of group names
`sessionlevels`	vector of character values for session names
`sessioncov`	optional dataframe of values of session-specific covariate(s).

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. 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.

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

```
data(deermouse)
```

Details

Two datasets of V. H. Reid were described by Otis et al. (1978) and distributed with their CAPTURE software (now available from http://www.mbr-pwrc.usgs.gov/software.html). 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.

Some fitted `secr` models are included (ESG.0, ESG.b, ESG.t, ESG.h2, WSG.0, WSG.b, WSG.t, WSG.h2, each with the indicated effect on g0). Otis et al. (1978) draw attention to the tendency of *Peromyscus* to become 'trap happy', and we observe that models with a behavioural response (ESG.b, WSG.b) have the lowest AIC among those fitted here.

Some components of the fitted secr models have been removed with `trim` to save space.

Object	Description
deermouse.ESG	capthist object, East Stuart Gulch
deermouse.WSG	capthist object, Wet Swizer Gulch
ESG.0	fitted secr model – ESG null
ESG.b	fitted secr model – ESG trap response g0
ESG.h2	fitted secr model – ESG finite mixture g0
ESG.t	fitted secr model – ESG time-varying g0
WSG.0	fitted secr model – WSG null
WSG.b	fitted secr model – WSG trap response g0
WSG.h2	fitted secr model – WSG finite mixture g0
WSG.t	fitted secr model – WSG time-varying g0

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))
```

---

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, sessnum = NULL, groups = NULL, alpha = 0.05,
    se.esa = FALSE, se.D = TRUE, loginterval = TRUE,
    distribution = NULL)
esa(object, sessnum = 1, beta = NULL, real = NULL, noccasions = NULL)
```

Arguments

`object`	`secr` object output from `secr.fit`, or an object of class `c('list','secrlist')`
`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
`beta`	vector of fitted parameters on transformed (link) scale
`real`	vector of 'real' parameters
`noccasions`	integer number of sampling occasions (see Details)

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.

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 `object$fit$par` 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.

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'.

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:

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

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

References

Borchers, D. L. and Efford, M. G. (2007) Supplements to Biometrics paper. Available online at http://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

```
## 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))
```

`detectfn` *Detection Functions*

Description

A detection function relates the probability of detection to the distance of a detector from a point. The reference point is usually thought of as an animal's home-range centre. In **secr** only simple 2- or 3-parameter functions are used. Each type of function is identified by a numeric code (see below). In most cases the name may also be used (as a quoted string).

Some functions such as (4) uniform are defined only for simulation as the pose problems for maximum likelihood.

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 function (8), '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).

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 $g(d) = 1 - F\{(c - \beta_0)/sdS\}$

The hazard-rate detection function was described by Hayes and Buckland (1983). The compound halfnormal detection function follows Efford and Dawson (2009). The signal strength and binary signal strength functions are from Efford et al. (2009).

Code	Name	Parameters	Function
0	halfnormal	g0, sigma	$g(d) = g_0 \exp\left(\frac{-d^2}{2\sigma^2}\right)$
1	hazard rate	g0, sigma, z	$g(d) = g_0[1 - \exp\{-({}^{d}/_{\sigma})^{-z}\}]$
2	exponential	g0, sigma	$g(d) = g_0 \exp\{-({}^{d}/_{\sigma})\}$
3	compound halfnormal	g0, sigma, z	$g(d) = g_0[1 - \{1 - \exp\left(\frac{-d^2}{2\sigma^2}\right)\}^z]$
4	uniform	g0, sigma	$g(d) = g_0, d <= \sigma; g(d) = 0,$ otherwise
5	w exponential	g0, sigma, w	$g(d) = g_0, d < w; g(d) = g_0 \exp\left(-\frac{d-w}{\sigma}\right)$, otherwise
6	annular normal	g0, sigma, w	$g(d) = g_0 \exp\{\frac{-(d-w)^2}{2\sigma^2}\}$
7	cumulative lognormal	g0, sigma, z	$g(d) = g_0[1 - F\{(d - \mu)/s\}]$

8	cumulative gamma	g0, sigma, z	$g(d) = g_0\{1 - G(d; k, \theta)\}$
9	binary signal strength	b0, b1	$g(d) = 1 - F\{-(b_0 + b_1 d)\}$
10	signal strength	beta0, beta1, sdS	$g(d) = 1 - F[\{c - (\beta_0 + \beta_1 d)\}/s]$
11	signal strength spherical	beta0, beta1, sdS	$g(d) = 1 - F[\{c - (\beta_0 + \beta_1 (d-1) - 10\log_{10} d^2)\}/s]$

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.

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

See Also

`detectfnplot`, `secr detection models`

`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', '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 `../doc/secr-overview.pdf`). Polygon and transect detector types are for area and linear searches as described in `../doc/secr-polygondetectors.pdf` and Efford (2011). The 'signal' detector type is used for acoustic data as described in `../doc/secr-sound.pdf`.

Value

character string for detector type

References

Efford, M. G. (2011) Estimation of population density by spatially explicit capture–recapture with area or transect searches. Unpublished manuscript.

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)
```

`distancetotrap` *Distance To Nearest Detector*

Description

Compute 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

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.

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)
```

ellipse.secr *Confidence ellipse*

Description

Plot joint confidence ellipse for two parameters of secr model

Usage

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

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

Details

A confidence ellipse is calculated from the asymptotic variance-covariance matrix of the beta parameters (coefficients), and optionally plotted.

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.

Value

A list containing the x and y coordinates is returned invisibly

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 `derived.nj`. 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.

Usage

```
derived.nj ( nj, esa, se.esa,  method = 'SRS', xy = NULL,
    alpha = 0.05, loginterval = TRUE )

derived.mash ( object, sessnum = NULL,  method = 'SRS',
    alpha = 0.05, loginterval = TRUE)

derived.cluster ( object, sessnum = NULL,  method = 'SRS',
    alpha = 0.05, loginterval = TRUE)

derived.session ( object,  method = 'SRS', xy = NULL,
    alpha = 0.05, loginterval = TRUE )

derived.external ( object, sessnum = NULL, nj, cluster, buffer = 100,
    mask = NULL, noccasions = NULL,  method = 'SRS', xy = NULL,
    alpha = 0.05, loginterval = TRUE)
```

Arguments

`object`	fitted secr model
`nj`	vector of number observed in each sampling unit (cluster)
`esa`	scalar 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)
`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

derived.cluster 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.).

derived.mash 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.

derived.nj accepts a vector of counts (nj), along with $\hat{a}$ and $\widehat{SE}(\hat{a})$. The argument esa may include both $\hat{a}$ and $\widehat{SE}(\hat{a})$) - any form will do if it can be coerced to a vector of length 2.

derived.external 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.

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

Spatial variance may be calculated assuming simple random sampling (method = 'SRS') or using the neighbourhood variance estimator recommended by Stevens and Olsen (2003) for generalized random tessellation stratified (GRTS) samples and implemented in package **spsurvey** (method = 'local'). For 'local' variance estimates, the centre of each replicate must be provided in xy, except where centres may be inferred from the data.

Value

A dataframe with one row and the columns –

estimate	Horvitz-Thompson-like estimate of population density
SE.estimate	SE of density 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

In versions before 2.1, the functionality of derived.nj and derived.session was provided by empirical.VarD, which has been removed.

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:

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

where $\widehat{\text{var}}(n) = \frac{J}{J-1} \sum_{j=1}^{J} (n_j - n/J)^2$. The estimate of $\text{var}(\hat{a})$ is model-based while that of $\text{var}(n)$ 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 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.

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

See Also

`derived`, `esa`

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.

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)
derived.nj(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.


## Not run:

## 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)
```

```
## 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)
derived.mash(fit1)

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

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)
derived(fit2)
derived.mash(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')
derived.external (fit2, nj = nj, cluster = t1, buffer = 100,
    noccasions = 1)

## Review plots
base.plot <- function() {
    eqscplot( testregion, axes = FALSE, xlab = '',
        ylab = '', type = 'n')
    polygon(testregion)
}
par(mfrow = c(1,3), xpd = T, 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()
cluster.plot (capt2a, add = TRUE,title = 'Extensive survey')
base.plot()
plot(capt2, add = TRUE, title = 'Intensive survey')
```

```
## End(Not run)
```

---

esa.plot *Mask Buffer Diagnostic Plot*

Description

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

Usage

```
esa.plot (object, max.buffer = NULL, spacing = NULL, max.mask = NULL,
    detectfn, detectpar, noccasions, binomN = NULL, thin = 0.1,
    poly = NULL, session = 1, plt = TRUE, as.density = TRUE, n = 1,
    add = FALSE, overlay = TRUE, ...)
```

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)
`session`	vector of session indices (used if `object` spans multiple sessions)
`plt`	logical to plot results
`as.density`	logical; if TRUE the y-axis is n / esa
`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
`...`	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. esa.plot 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 $d_k(\mathbf{X}) <=$ buffer, where c is the area associated with each cell.

The default (as.density = TRUE) 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 esa.plot 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.

esa.plot calls the internal function esa.plot.secr 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 ($p.(\mathbf{X})$)

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.

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, make.mask, mask.check, detection functions

Examples

```
## with previously fitted model
esa.plot(secrdemo.0)

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

esa.plot(trps,,, msk, 0, detectpar, nocc = 5, thin = 0.002, plt = FALSE)
```

`esa.plot.secr` *Mask Buffer Diagnostic Plot (internal)*

Description

Internal function used to plot effective sampling area (Borchers and Efford 2008) as a function of increasing buffer width given an 'secr' object

Usage

```
esa.plot.secr (object, max.buffer = NULL, max.mask = NULL,
    thin = 0.1, poly = NULL, session = 1, plt = TRUE, as.density
    = TRUE, add = FALSE, overlay = TRUE, ...)
```

Arguments

`object`	`secr` object output from `secr.fit`
`max.buffer`	maximum width of buffer in metres
`max.mask`	`mask` object
`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)
`session`	vector of session indices (used if `object` spans multiple sessions)
`plt`	logical to plot results
`as.density`	logical; if TRUE the y-axis is n / esa
`add`	logical to add line to an existing plot
`overlay`	logical; if TRUE then automatically `add = TRUE` for plots after the first
`...`	graphical arguments passed to plot() and lines()

Details

esa.plot.secr provides a wrapper for esa.plot that is called internally from esa.plot when it is presented with an secr object. Arguments of esa.plot such as detectfn are inferred from the fitted model.

If max.mask is not specified then a maximal mask of type 'trapbuffer' is constructed using max.buffer and the spacing of the mask in object. In this case, if max.buffer is not specified then it is set either to the width of the existing plot (add = TRUE) or to 10 x sigma-hat from the fitted model in object (add = FALSE).

Value

see esa.plot

See Also

esa.plot, mask, pdot, make.mask, mask.check, detection functions

---

expected.n — *Expected Number of Individuals*

Description

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

Usage

```
expected.n(object, session = NULL, group = NULL, bycluster
    = TRUE, splitmask = 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 separate detectors
splitmask	logical for computation method (see Details)

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, clustered then each detector will be treated as a cluster, with a warning.

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

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

Value

Vector of expected count with length equal to the number of detectors or the number of clusters. For multi-session data, a list of such vectors.

Warning

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

See Also

region.N

Examples

```
expected.n(secrdemo.0)
expected.n(ovenbird.model.D)

## Not run:
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)

## 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)
```

FAQ *Frequently Asked Questions, And Others*

Description

A place for hints and miscellaneous advice.

How do I install secr?

Follow the usual procedure for installing binaries from CRAN, or ...

Under Windows, install the package binary from the Rgui. Save the file secr_2.1.x.zip to a local folder ('x' is the release number) and use the menu option "Packages | Install packages(s) from local zip files...".

For other systems you may need to install the source package secr_2.1.x.tar.gz.

Whatever your system, you also need to get the package **abind** (use Packages | Install package(s)... in Windows to download from CRAN). Other required packages (**MASS**, **nlme**, **stats**) should be available as part of your R installation.

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.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 documents:

```
RShowDoc ('secr-manual', package='secr')
```

```
RShowDoc ('secr-overview', package='secr')
```

See below for more R tips.

How should I report a problem?

If you get really stuck or find something you think is a bug then please report the problem. There is a support forum at http://www.phidot.org/forum under 'DENSITY|secr'. Please read the FAQ there before posting.

You may be asked to send an actual dataset - ideally, the simplest one that exhibits the problem. The correct address for this is <density.software@otago.ac.nz>. 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 threshold of probability for capture histories (smaller values are all set to minprob), whereas Density has no explicit limit. In the current version the default minprob has been reduced from 1e-20 to 1e-50. If you find a discrepancy with Density it may be worth lowering minprob even further.

How can I speed up model fitting and model selection?

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).

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).

Do you really need to fit that complex model? 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.

If your detectors are arranged in similar clusters (e.g., small square grids) then try the function `mash`.

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.

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 (http://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.

`flip` *Flip Points*

Description

Flip an array of points about a vertical or horizontal axis.

Usage

```
flip (object, lr = F, tb = F, ...)
```

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
`...`	other arguments (not used)

Details

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.

This is a generic function. A method is provided for `traps` objects.

Value

A matrix with the coordinates of each point reflected about the desired axis or axes.

Author(s)

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

See Also

traps, rotate.traps

Examples

```
temp <- matrix(runif (20) * 2 - 1, nc = 2)
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)
```

`flip.traps` *Flip Detector Array*

Description

Flip a detector array about a vertical or horizontal axis.

Usage

```
## S3 method for class 'traps'
 flip(object, lr = F, tb = F, ...)
```

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
`...`	other arguments (not used)

Details

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.

Value

Object of class `traps` with the coordinates of each point reflected about the desired axis or axes.

Author(s)

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

See Also

traps, rotate.traps, shift.traps

Examples

```
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, lab = TRUE, offset = 7, gridl = FALSE)
plot(traps2, border = 5, lab = TRUE, offset = 7, gridl = FALSE)
```

---

fxi *Probability Density of Home Range Centre*

Description

Display contours of the probability density function for the estimated location of one or more range centres (f(Xi|wi)), compute values for particular points X, or compute mode of pdf.

Usage

```
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, normal = TRUE, ...)
fxi.secr(object, i = 1, sessnum = 1, X, normal = TRUE)
fxi.mode(object, i = 1, sessnum = 1, start = NULL, ...)
```

Arguments

`object`	a fitted secr model
`i`	integer or character vector of individuals for which to plot contours, or a single individual as input to other functions
`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
`normal`	logical; should values of pdf be normalised?
`start`	vector of x-y coordinates for maximization
`...`	additional arguments passed to `contour` or `nlm`

Details

fxi.contour computes contours of 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 np.

fxi.secr computes the probability density for a single detection history; X may contain coordinates for one or several points; a dataframe or vector (x then y) will be coerced to a matrix.

fxi.mode finds the maximum of the pdf for a single detection history (i.e. n is of length 1). fxi.mode calls nlm.

fxi.contour with fitmode = TRUE uses fxi.mode to find the maximum of each pdf. 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 optionally normalised by dividing by the integral of Pr(wi|X) over the habitat mask in object.

If start is not provided then the first detector site is used, but this is not guaranteed to work.

The ... argument gives additional control over a contour plot; for example, set drawlabels = FALSE to suppress contour labels.

Value

fxi.contour –

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).

fxi.secr –

Vector of probability densities

fxi.mode –

List with components 'x' and 'y'

Note

These functions only work with homogeneous Poisson density models and do not allow incomplete usage (some detectors not used on some occasions).

References

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

See Also

pdot.contour, contour

Examples

```
fxi.secr(secrdemo.0, i = 1, X = c(365,605))

plot(secrdemo.0$capthist)
## contour first 5 detection histories
fxi.contour (secrdemo.0, i = 1:5, add = TRUE,
    plotmode = TRUE, drawlabels = FALSE)
```

---

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.

`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.

`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).

Usage

```
dbar(capthist)
RPSV(capthist)
MMDM(capthist, min.recapt = 1, full = FALSE)
ARL(capthist, min.recapt = 1, plt = FALSE, full = FALSE)
moves(capthist)
```

Arguments

`capthist`	object of class `capthist`
`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

Details

`dbar` is defined as

$$\overline{d} = \frac{\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n_i-1} \sqrt{(x_{i,j} - x_{i,j+1})^2 + (y_{i,j} - y_{i,j+1})^2}}{\sum\limits_{i=1}^{n} (n_i - 1)}$$

RPSV is defined as

$$RPSV = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{n_i} [(x_{i,j} - \overline{x}_i)^2 + (y_{i,j} - \overline{y}_i)^2]}{\sum_{i=1}^{n} (n_i - 1) - 1}}$$

dbar and RPSV have a specific role as proxies for detection scale in inverse-prediction estimation of density (Efford 2004; see ip.secr).

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.

Value

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.

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'.

References

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

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.

See Also

autoini

Examples

```
dbar(captdata)
RPSV(captdata)
```

`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

```
data(hornedlizard)
```

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 `../doc/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

Source

Royle and Young (2008) and J. A. Royle (pers. comm.), with additional information from K. V. Young (pers. comm.).

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

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

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.fit <- secr.fit(hornedlizardCH)
FTHLbn.fit <- secr.fit(hornedlizardCH, details =
    list(distribution = 'binomial'))
collate(FTHL.fit, FTHLbn.fit)[,,,'D']

## Collapse occasions (does not run faster)
hornedlizardCH.14 <- reduce(hornedlizardCH, columns =
    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

```
data(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 (now available from http://http://www.mbr-pwrc.usgs.gov/software.html). Otis et al. (1978 p. 62, 68) cite 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'.

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. We include several secr models fitted to the 'morning' data (`morning.0`, `morning.b` etc.). Of these, a model including individual heterogeneity in both g0 and sigma has the lowest AIC.

Some components of the fitted secr models have been removed with `trim` to save space.

Object	Description
housemouse	capthist object
housemouse.0	fitted secr model – null
housemouse.ampm	fitted secr model – g0 differs morning vs afternoon
housemouse.ampmh2h2	fitted secr model – as above, finite mixture g0, sigma
morning.0	fitted secr model – morning data only, null
morning.0h2	fitted secr model – mornings, null g0, finite mixture sigma
morning.b	fitted secr model – mornings, trap response g0
morning.h2	fitted secr model – mornings, finite mixture g0
morning.h2h2	fitted secr model – mornings, finite mixture g0, sigma
morning.t	fitted secr model – mornings, day-specific g0

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))

AIC(morning.0, morning.b, morning.t, morning.h2, morning.0h2, morning.h2h2)

## assess need to distinguish morning and afternoon samples
## Not run:
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)
```

---

ip.secr *Spatially Explicit Capture-Recapture by Inverse Prediction*

Description

Estimate population density by simulation and inverse prediction (Efford 2004; Efford, Dawson & Robbins 2004). A restricted range of SECR models may be fitted (detection functions with more than 2 parameters are not supported, nor are covariates).

Usage

```
ip.secr (capthist, predictorfn = pfn, predictortype = 'null',
    detectfn = 0, mask = NULL, start = NULL, boxsize = 0.1,
    centre = 3,  min.nsim = 10, max.nsim = 2000, CVmax = 0.002,
    var.nsim = 1000, maxbox = 5, ...)

pfn(capthist, N.estimator)
```

Arguments

`capthist`	capthist object including capture data and detector (trap) layout
`predictorfn`	a function with two arguments (the first a capthist object) that returns a vector of predictor values

`predictortype`
: value (usually character) passed as the second argument of predictorfn

`detectfn`
: integer code or character string for shape of detection function 0 halfnormal, 2 exponential, 3 uniform) – see detectfn

`mask`
: optional habitat mask to limit simulated population

`start`
: vector of np initial parameter values (density, g0 and sigma)

`boxsize`
: scalar or vector of length np for size of design as fraction of central parameter value

`centre`
: number of centre points in simulation design

`min.nsim`
: minimum number of simulations per point

`max.nsim`
: maximum number of simulations per point

`CVmax`
: tolerance for precision of points in predictor space

`var.nsim`
: number of additional simulations to estimate variance-covariance matrix

`maxbox`
: maximum number of attempts to 'frame' solution

`...`
: further arguments passed to sim.popn

`N.estimator`
: character value indicating population estimator to use

Details

'Inverse prediction' uses methods from multivariate calibration (Brown 1982). The goal is to estimate population density (D) and the parameters of a detection function (usually g0 and sigma) by 'matching' statistics from `predictorfn(capthist)` (the target vector) and statistics from simulations of a 2-D population using the postulated detection model. Statistics (see Note) are defined by the predictor function, which should return a vector equal in length to the number of parameters (np = 3). Simulations of the 2-D population use `sim.popn`. The simulated population is sampled with `sim.capthist` according to the detector type (e.g., 'single' or 'multi') and detector layout specified in traps(capthist).

... may be used to control aspects of the simulation by passing named arguments (other than D) to `sim.popn`. The most important arguments of `sim.popn` to keep an eye on are 'buffer' and 'Ndist'. 'buffer' defines the region over which animals are simulated (unless `mask` is specified) - the region should be large enough to encompass all animals that might be caught. 'Ndist' controls the number of individuals simulated within the buffered or masked area. The default is 'poisson'. Use 'Ndist = fixed' to fix the number in the buffered or masked area A at $N = DA$. This conditioning reduces the estimated standard error of $\hat{D}$, but conditioning is not always justified - seek advice from a statistician if you are unsure.

The simulated 2-D distribution of animals is Poisson by default. There is no 'even' option as in Density.

Simulations are conducted on a factorial experimental design in parameter space - i.e. at the vertices of a cuboid 'box' centred on the working values of the parameters, plus an optional number of centre points. The size of the 'box' is specified as a fraction of the working values, so for example the limits on the density axis are D*(1–boxsize) and D*(1+boxsize) where D* is the working value of D. For g0, this computation uses the odds transformation (g0/(1–g0)). `boxsize` may be a vector defining different scaling on each parameter dimension.

A multivariate linear model is fitted to predict each set of simulated statistics from the known parameter values. The number of simulations at each design point is increased (doubled) until the residual standard error divided by the central value is less than CVmax for all parameters. An error occurs if max.nsim is exceeded.

Once a model with sufficient precision has been obtained, a new working vector of parameter estimates is 'predicted' by inverting the linear model and applying it to the target vector. A working vector is accepted as the final estimate when it lies within the box; this reduces the bias from using a linear approximation to extrapolate a nonlinear function. If the working vector lies outside the box then a new design is centred on value for each parameter in the working vector.

Once a final estimate is accepted, further simulations are conducted to estimate the variance-covariance matrix. These also provide a parametric bootstrap sample to evaluate possible bias. Set var.nsim = 0 to suppress the variance step.

See Efford et al. (2004) for another description of the method, and Efford et al. (2005) for an application.

The value of `predictortype` is passed as the second argument of the chosen `predictorfn`. By default this is `pfn`, for which the second argument (`N.estimator`) is a character value from c('n', 'null','zippin','jackknife'), corresponding respectively to the number of individuals caught (Mt+1), and $\hat{N}$ from models M0, Mh and Mb of Otis et al. (1978).

If not provided, the starting values are determined automatically with `autoini`.

Linear measurements are assumed to be in metres and density in animals per hectare (10 000 m^2).

Value

For `ip.secr`, a list comprising

`call`	the function call
`IP`	dataframe with estimated density ha^{-1}, g0 and sigma (m)
`vcov`	variance-covariance matrix of estimates
`ip.nsim`	total number of simulations
`variance.bootstrap`	dataframe summarising simulations for variance estimation
`proctime`	processor time (seconds)

For `pfn`, a vector of numeric values corresponding to $\hat{N}$, $\hat{p}$, and `RPSV`, a measure of the spatial scale of individual detections.

Note

Each statistic is expected to have a monotonic relationship with one parameter when the other parameters are held constant. Typical statistics are -

Statistic	Parameter
$\hat{N}$	D
$\hat{p}$	g_0
$RPSV$	σ

where $\hat{N}$ and $\hat{p}$ are estimates of population size and capture probability from the naive application of a nonspatial population estimator, and $RPSV$ is a trap-revealed measure of the scale of movement.

This method provides nearly unbiased estimates of the detection parameter g0 when data are from single-catch traps (likelihood-based estimates of g0 are biased in this case - Efford, Borchers & Byrom 2009).

The implementation largely follows that in Density, and it may help to consult the Density online help. There are some differences: the M0 and Mb estimates of population-size in `ip.secr` can take non-integer values; the simulation design used by `ip.secr` uses odds(g0) rather than g0; the default boxsize and CVmax differ from those in Density 4.4. There is no provision in `ip.secr` for two-phase estimation, using a different experimental design at the second phase. If you wish you can achieve the same effect by using the estimates as starting values for a second call of `ip.secr` (see examples).

Maximum likelihood estimates from `secr.fit` are preferable in several respects to estimates from inverse prediction (speed*; more complex models; tools for model selection). `ip.secr` is provided for checking estimates of g0 from single-catch traps, and for historical continuity.

* `autoini` with thin = 1 provides fast estimates from a simple halfnormal model if variances are not required.

References

Brown, P. J. (1982) Multivariate calibration. *Journal of the Royal Statistical Society, Series B* **44**, 287–321.

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. 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 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.

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**.

See Also

`capthist`, `secr.fit`, `RPSV`, `autoini`, `sim.popn`, `detection functions`

Examples

```
## Not run:
## these calculations may take several minutes

## default settings
ip.secr (captdata)

## coarse initial fit, no variance step
ip1 <- ip.secr (captdata, boxsize = 0.2, CVmax=0.01, var=0)
## refined fit
ip2 <- ip.secr (captdata, start = ip1$IP[,'estimate'],
    boxsize = 0.1, CVmax=0.002, var=1000)
ip2

## improvise another predictor function (dbar instead of RPSV)
pfn2 <- function (capthist, v) {  ## v is not used
    sumni <- sum(capthist!=0)    ## total detections
    n <- nrow(capthist)          ## number of individuals
    nocc <- ncol(capthist)       ## number of occasions
    c(N = n, p = sumni/n/nocc, dbar = dbar(capthist))
}
ip.secr (captdata, predictorfn = pfn2)
```

```
## End(Not run)
```

`LLsurface.secr` *Plot likelihood surface*

Description

Calculate log likelihood over a grid of values of two beta parameters from a fitted secr model and optionally make an approximate contour plot of the log likelihood surface.

Usage

```
LLsurface.secr(object, betapar = c("g0", "sigma"), xval = NULL,
    yval = NULL, centre = NULL, realscale = TRUE, plot = TRUE,
    plotfitted = TRUE, ...)
```

Arguments

`object`	`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 (+)
`...`	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.

Value

Invisibly returns a matrix with the log likelihood evaluated at each grid point

Note

`LLsurface.secr` 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.secr(secrdemo.CL, xval = seq(0.16,0.40,0.02),
    yval = 25:35, nlevels = 20)

## now verify MLE
## click on MLE and apparent 'peak'
xy <- locator(2)
temp <- LLsurface.secr(secrdemo.CL, xval = xy$x,
     yval = xy$y, plot = FALSE)
temp

## 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 for SECR Models*

Description

Compute likelihood ratio test to compare two fitted models, one nested within the other.

Usage

```
LR.test(secr1, secr2)
```

Arguments

`secr1`	fitted secr model
`secr2`	fitted secr model

Details

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.

The models must be nested (no check is performed - this is up to the user), but either secr1 or secr2 may be the more general 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 secr models compared

See Also

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 = "trapID", noccasions = NULL,
    covnames = NULL, bysession = TRUE, sortrows = TRUE,
    cutval = NULL, tol = 0.01, noncapt = 'NONE')
```

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. Valid values are 'XY' and 'trapID'.
`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, tolerance in metres when assigning coordinates for 'transect' detector type
`noncapt`	character value; animal ID used for 'no 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 2009; see also `../doc/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).

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`.

If the input has data from multiple sessions then the output is an object of class c('list','capthist') 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.

References

Efford, M. G. (2009) *Density 4.4: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand. `http://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.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,
    type = "traprect", poly = NULL, keep.poly = TRUE,
    check.poly = TRUE, pdotmin = 0.001, ...)
```

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
`type`	character string for method to use (‘traprect’, ‘trapbuffer’, ‘pdot’, ‘polygon’, ‘clusterrect’, ‘clusterbuffer’)
`poly`	bounding polygon to which mask should be clipped (see Details)
`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)
`...`	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.

If `poly` is specified, points outside `poly` are dropped. 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
- a SpatialPolygonsDataFrame object as defined in the package 'sp', possibly from reading a shapefile with readShapePoly() from package 'maptools'.

If `spacing` is not specified then it is determined by dividing the range of the x coordinates (including any buffer) by `nx`.

Value

an object of class `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 `buffer.contour` 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`, `subset.mask`, `pdot`, `buffer.contour`

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)

## 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).
## Requires the 'maptools' package of Nicholas J. Lewin-Koh, Roger
## Bivand, and others; 'maptools' uses the 'sp' package of spatial
## classes by Ed Pebesma and Roger Bivand.

## Not run:
require(maptools)
setwd(system.file("extdata", package = "secr"))
possumarea <- readShapePoly('possumarea')  ## possumarea.shp etc.
possummask2 <- make.mask(traps(possumCH), spacing = 20,
    buffer = 250, type = 'trapbuffer', poly = possumarea)
oldpar <- par(mar = c(1,6,6,6), xpd = TRUE)
plot (possummask2, ppoly = TRUE)
plot(traps(possumCH), add = T)
par(oldpar)

## End(Not run)
```

`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, ...)
```

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)
`...`	arguments passed to `trap.builder`

Details

`region` may be any shape. The **sp** class SpatialPolygonsDataFrame is useful for complex shapes and input from shapefiles using **maptools** (see Examples). 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.

Value

A single-session 'traps' object.

Note

Do not confuse with the simpler function `make.grid`, which places single detectors in a rectangular array.

See Also

`trap.builder`, `cluster.centres`, `readShapePoly`

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:
require(maptools)
setwd(system.file("extdata", package = "secr"))
```

```
possumarea <- readShapePoly('possumarea')
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.

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 = 20, spacing = NULL,
    detector = "multi", originxy = c(0,0), hollow = F,
    ID = 'alphay')

make.circle (n = 20, radius = 100, spacing = NULL,
    detector = "multi", originxy = c(0,0), IDclockwise = T)

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)
```

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
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

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.

Specialised functions for arrays using a triangular grid are described separately (make.tri, clip.hex).

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.

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. (2007) *Density 4.1: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand. http://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`, `print.traps`, `plot.traps`, `traps`, `make.tri`

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, lab = TRUE, offset = 7,
        gridl = FALSE)
}

temptrap <- make.grid(nx = 7, ny = 5, hollow = TRUE)
plot (temptrap, border = 10, lab = TRUE, gridl = FALSE)

plot(make.circle(n = 20, spacing = 30), lab = TRUE, offset = 9)
summary(make.circle(n = 20, spacing = 30))
```

---

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 (southwest) 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),
    lab = TRUE, offset = 2.5 )
```

---

mask *Mask Object*

Description

Encapsulate a habitat mask for spatially explicit capture–recapture.

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`, `esa.plot`), 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 $p.(\mathbf{X})$ 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`.

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`, `esa.plot`, `suggest.buffer`, `secr.fit`, `secr density models`

`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 4. The help page in Density 4 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. (2009) *DENSITY 4.4: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand `http://www.otago.ac.nz/density`.

See Also

`esa.plot`, `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
olddir <- setwd(system.file("extdata", package = "secr"))
possumarea <- read.table('possumarea.txt', header = TRUE)
setwd(olddir)
data (possum)
mask.check (possum.model.1, spacings = c(20, 30), buffers =
    c(200, 300), poly = possumarea, LLonly = FALSE,
    file = 'temp', CL = TRUE)

## review fitted models
load ('temp.RData')
oldpar <- par(mfrow = c(2,2), mar = c(1,4,4,4), xpd = FALSE)
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(oldpar)


## End(Not run)
```

`model.average` *Model averaging for SECR Models*

Description

AICc-weighted average of estimated 'real' or 'beta' parameters from multiple fitted secr models.

Usage

```
model.average(..., realnames = NULL, betanames = NULL, newdata = NULL,
    alpha = 0.05, dmax = 10, covar = FALSE, average = 'link')

collate (..., realnames = NULL, betanames = NULL, newdata = NULL,
    scaled = FALSE, alpha = 0.05, perm = 1:4, fields = 1:4)
```

Arguments

`...`	`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
`scaled`	logical for scaling of sigma and g0 (see Details)

alpha	alpha level for confidence intervals
dmax	numeric, the maximum AIC 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
perm	permutation of dimensions in output from collate
fields	vector to restrict summary fields in output

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 weights with small sample adjustment (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 dAICc > dmax are given a weight of zero and effectively are excluded from averaging.

Also,

$$\text{var}(\hat{\theta}) = \sum_k w_k (\text{var}(\hat{\theta}_k | \beta_k) + \beta_k^2)$$

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).

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.

The argument scaled applies only to the detection parameters g0 and sigma, and only to models fitted with scalesigma or scaleg0 switched on. If scaled is TRUE then each estimate is multiplied by its scale factor (1/D^0.5 and 1/sigma^2 respectively).

Value

For model.average, 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.

For collate, 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 ('D', 'g0' etc.). For particular comparisons it often helps to reorder the dimensions with the perm argument.

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.

See Also

`AIC.secr`, `secr.fit`

Examples

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

model.average(secrdemo.0, secrdemo.b)
model.average(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)

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

`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

```
data(ovenbird)
```

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). Recaptures within a day are not included in this dataset, so each bird occurs at most once per day and the detector type is 'multi' rather than 'proximity'. Although several individuals were captured in more than one year, no use is made of this information in the analyses presently offered in **secr**.

The data are provided as a multi-session `capthist` object 'ovenCH'. Sex is coded as a categorical individual covariate ('M' or 'F').

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 in press). 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 four models fitted as in Examples. Some components of the fitted secr models have been removed with `trim` to save space.

Object	Description
ovenCH	multi-session capthist object
ovenbird.model.1	fitted secr model – null
ovenbird.model.1b	fitted secr model – g0 net shyness
ovenbird.model.1T	fitted secr model – g0 time trend within years
ovenbird.model.h2	fitted secr model – g0 finite mixture
ovenbird.model.D	fitted secr model – trend in density across years
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

```
## Not run:

## 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 = 'multi')
temp <- read.table('ovencapt.txt', colClasses=c('character',
    'character', 'numeric', 'numeric', 'character'))
ovenCH <- make.capthist(temp, netsites0509, covnames=c('Sex','Age'))


## End(Not run)
```

```
oldpar <- par(mfrow = c(1,5), mar = c(1,1,4,1))
plot(ovenCH, tracks = TRUE, varycol = TRUE)
par(oldpar)

counts(ovenCH, 'n')

## Not run:

## array constant over years, so build mask only once
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)
ovenbird.model.1

## fit net avoidance model
ovenbird.model.1b <- secr.fit(ovenCH, mask = ovenmask, model =
    list(g0~b))
ovenbird.model.1b

## fit model with time trend in detection
ovenbird.model.1T <- secr.fit(ovenCH, mask = ovenmask, model =
    list(g0 ~ T))
ovenbird.model.1T

## fit model with 2-class mixture for g0
ovenbird.model.h2 <- secr.fit(ovenCH, mask = ovenmask, model =
    list(g0~h2))
ovenbird.model.h2


## End(Not run)

## compare & average pre-fitted models
AIC (ovenbird.model.1, ovenbird.model.1b, ovenbird.model.1T,
    ovenbird.model.h2)
model.average (ovenbird.model.1,ovenbird.model.1b, ovenbird.model.1T,
    ovenbird.model.h2, realnames='D')

## select one year to plot
plot(ovenbird.model.1b, newdata = data.frame(session = '2005',
    b = 0))
```

`ovensong` *Ovenbird Acoustic Dataset*

Description

Data from an acoustic survey of ovenbirds (*Seiurus aurocapilla*) at a site in Maryland, USA.

Usage

```
data(ovensong)
```

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). Some components of the fitted secr models have been removed with `trim` to save space.

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)
```

---

pdot *Net Detection Probability*

Description

Compute spatially explicit net probability of detection for individual(s) at given coordinates.

Usage

```
pdot(X, traps, detectfn = 0, detectpar = list(g0 = 0.2,
     sigma = 25, z = 1), noccasions = 5, binomN = NULL)
```

Arguments

X	coordinates
traps	traps object
detectfn	integer code for detection function q.v.
detectpar	a list giving a value for each named parameter of detection function
noccasions	number of intervals (occasions)
binomN	integer code for discrete distribution (see secr.fit

Details

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.

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.

Value

A vector of probabilities, one for each row in X.

See Also

secr, make.mask, detection functions, pdot.contour

Examples

```
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))
```

---

plot.capthist *Plot Detection Histories*

Description

Display a plot of detection (capture) histories 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 = 'petal',
    cappar = list(cex = 1.3, pch = 16, col = "blue"),
    trkpar = list(col = "blue", lwd = 1),
    labpar = list(cex = 0.7, col = "black"), ...)
```

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 to be passed to `lines` if tracks are plotted

Details

By default, a 'petal' plot is generated in the style of Density (Efford 2007) 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 `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`.

If `icolours = NULL` and `varycol = TRUE` then a vector of colours is generated automatically as terrain.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.

Value

For `type = 'petal'`, the number of detections in `x`. For `type = 'n.per.detector'` or `type = 'n.per.cluster'`, a dataframe with data for a legend (see Examples).

References

Efford, M. G. (2007) *Density 4.1: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand. http://www.otago.ac.nz/density

See Also

`capthist`

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', hidetr = FALSE, gridl = FALSE,
    cappar = list(cex = 1.5))

## Not run:
## 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 = 0.6)

## End(Not run)
```

`plot.mask` *Plot Habitat Mask*

Description

Plot a habitat mask either as points or as an `image` plot. Colours may be used to show the value of one mask covariate.

Usage

```
## S3 method for class 'mask'
plot(x, border = 20, add = F, covariate = NULL, axes = F,
     dots = T, col = "grey", breaks = 12, ppoly = T,
     polycol = "red", ...)
```

Arguments

`x`	`mask` 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
`breaks`	number of levels to use when cutting continuous covariate for plotting
`col`	colour(s) to use for plotting
`ppoly`	logical for whether the bounding polygon should be plotted (applies only if mask type = 'polygon')
`polycol`	colour for outline of polygon (`ppoly` = TRUE)
`...`	other arguments passed to `eqscplot`

Details

The argument `dots` selects between two distinct types of plot. If using a covariate to colour points, the `col` argument should be a colour vector of length equal to the number of levels.

Border lines around pixels are drawn in the current foreground colour (par('fg')). Set this to NA with `par(fg=NA)` to eliminate borders, but remember to reset it when you've finished.

See Also

`colours`, `mask`

Examples

```
# simple

temptrap <- make.grid()
tempmask <- make.mask(temptrap)
plot (tempmask)

## restrict to points over an arbitrary detection threshold,
## add covariate, plot image and overlay traps

tempmask <- subset(tempmask, pdot(tempmask,temptrap)>0.001)

covariates (tempmask) <- data.frame(circle =
    exp(-(tempmask$x^2 + tempmask$y^2)/10000) )

par(fg='white')
```

```
plot (tempmask, covariate = 'circle', dots = FALSE, axes = TRUE,
    add = TRUE, breaks = 8, col = terrain.colors(8))
par(fg='black')

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 <- formatC(seq(covrange[1],covrange[2],step), format='f',
    digits=2, width=4)
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')
```

`plot.popn` *Plot popn Object*

Description

Display animal locations from a `popn` object.

Usage

```
## S3 method for class 'popn'
plot(x, add = FALSE, frame = TRUE, circles = NULL, ...)
```

Arguments

`x`	object of class `popn`
`add`	logical to add points to an existing plot
`frame`	logical to add frame within which points were simulated
`circles`	vector giving the radii if circles are to be plotted
`...`	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).

See Also

`popn`, `sim.popn`

Examples

```
temppopn <- sim.popn(D = 5, expand.grid(
    x = c(0,100), y = c(0,100)))
plot(temppopn, pch = 16, 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, 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)
`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 is assumed to be normal on the logit scale; limits are back-transformed from that scale.

`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 = TRUE, gridspace = 100, gridcol = "grey",
  markused = FALSE, markvarying = FALSE, ...)
```

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
`...`	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.

Value

None

See Also

`plot`, `traps`

Examples

```
temptrap <- make.grid()
plot (temptrap, detpar = list(pch = 16, col = 'blue'),
   label = TRUE, offset = 7)
```

`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**
`ha`	logical if TRUE output is converted from square metres to hectares

Details

For SpatialPolygons or SpatialPolygonsDataFrame objects it is necessary to have installed the packages **sp** and **rgeos**.

Value

A scalar.

See Also

`buffer.contour`

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')
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

```
data(possum)
```

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 294-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 294-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). 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 ('possum.model.1' & 'possum.model.1b') are provided for illustration.

From secr 1.5 a dataframe `possumarea` is included to indicate the boundary of potential habitat, and this is used to clip `possummask`. `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`.

Clipping of the mask, introduced in secr 1.5, slightly changes the fitted models (and hence the density estimates).

Object	Description
possumCH	capthist object
possummask	mask object
possumarea	habitat perimeter
possum.model.1	fitted secr model – null
possum.model.1b	fitted secr model – trap response g0

Some components of the fitted secr models have been removed with `trim` to save space.

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

```
## Not run:

## code used to construct dataset
setwd('d:\density secr 2.1\package data\possum')
possumCH <- read.capthist('foxton.txt', 'foxtraps.txt', detector = 'single')
## drop within-day duplicates of tag 5861 in traps 11,12, to conform DENSITY
possumCH['5861',] <- c(134, 133, 134, 133, 135)
trps <- traps(possumCH)
covariates(traps(possumCH)) <- data.frame(grid =
    factor(substring(rownames(trps),1,2)))
possumarea <- read.table ('possumarea.txt', header = TRUE)
possummask <- make.mask(trps, buffer = 300, type='pdot', pdotmin =
    0.001, detectpar = list(g0=0.2, sigma=60), spacing = 10, poly =
    possumarea)
possum.model.1 <- secr.fit(possumCH, mask = possummask, trace=F)
possum.model.1b <- secr.fit(possumCH, mask = possummask, model =
    list(g0~b), trace=F)
save(possumCH, possummask, possumarea, possum.model.1, possum.model.1b,
    file='d:\density secr 2.0\secr\data\possum.RData')

## End(Not run)

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.1, possum.model.1b)
model.average(possum.model.1, possum.model.1b)
```

---

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, se.fit = TRUE, alpha = 0.05,
    savenew = FALSE, scaled = FALSE, ...)

detectpar (object, ...)
```

Arguments

`object`	`secr` object output from `secr.fit`
`newdata`	optional dataframe of values at which to evaluate model
`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
`scaled`	logical for scaling of sigma and g0 (see Details)
`...`	other arguments

Details

The variables in the various linear predictors are described in secr models 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.

Standard errors are by the delta method (Lebreton et al. 1992). Confidence intervals are backtransformed from the link scale.

The argument `scaled` applies only to the detection parameters g0 and sigma, and only to models fitted with `scalesigma` or `scaleg0` switched on. If `scaled` is TRUE then each estimate is multiplied by its scale factor (1/D^0.5 and 1/sigma^2 respectively).

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 `esa.plot`. `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.

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).

References

Lebreton, J.-D., Burnham, K. P., Clobert, J., 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`

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))

temp <- predict (secrdemo.b, newdata = data.frame(b=0:1),
    save = TRUE)
attr(temp, 'newdata')

detectpar(secrdemo.0)
```

`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.mask` *Print Habitat Mask*

Description

Print the x-y coordinates of points in a `mask` object

Usage

```
## S3 method for class 'mask'
print(x, ...)
```

Arguments

`x`	`mask` object
`...`	arguments passed to other functions

Author(s)

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

See Also

`mask`, `summary.mask`

Examples

```
tempmask <- make.mask(make.grid())
print(tempmask)
```

`print.secr` *Print secr Object*

Description

Print results from fitting a spatially explicit capture–recapture model.

Usage

```
## S3 method for class 'secr'
 print(x, newdata = NULL, alpha = 0.05, deriv = FALSE, ...)
```

Arguments

`x`	`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
`...`	other arguments (not used currently)

Details

Results 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
time	date and time fitting started
N animals	number of distinct animals detected
N captures	number of detections
N occasions	number of sampling occasions
N detectors	number of detectors
Detector type	'single', 'multi', 'proximity' etc.
Model	model formula for each 'real' parameter
Fixed	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

Derived parameters (see `derived`) are computed only for models fitted by maximizing the conditional likelihood (CL = TRUE).

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)
print(secrdemo.CL, deriv = TRUE)
```

---

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))
```

rbind.capthist — *Combine capthist Objects*

Description

Form a single capthist object from two or more compatible capthist objects.

Usage

```
rbind.capthist(..., renumber = TRUE, pool = NULL)
MS.capthist(...)
```

Arguments

...	one or more simple capthist objects (i.e., single-session)
renumber	assign new composite individual ID: sourceobject.oldID
pool	list of vectors of indices

Details

In its simplest usage, the source objects in ... each provide detection histories from a single session, and the result is a single-session object. For this to work the objects must be compatible. capthist objects are compatible if they use the same detectors (traps) and have consistent covariates and other attributes.

If the ... argument includes at least one multi-session capthist object then the elements will be formed into a single multi-session capthist object. If ... is a single multi-session object then the components of pool are used to define combinations of old sessions (e.g. pool = list(1:3, 4:5) forms two new sessions from 5 old ones).

Although rbind.capthist looks like an S3 method, it isn't. The full function name must be used.

MS.capthist treats each source object as the data for a separate session. Compatibility is not required. The ... argument may include lists of single-session capthist objects regardless of whether the list has the class ('list','capthist').

Value

For rbind.capthist, an object of class capthist with number of rows equal to the sum of the rows in the input objects.

For MS.capthist, a multi-session object of class capthist with number sessions equal to the number of objects in ....

See Also

capthist, subset.capthist

Examples

```
## simulate 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))
temp3 <- rbind.capthist(temp1, temp2)

## compare mixture to sum of components
## note 'traps visited' is not additive for 'multi' detector
## nor is 'traps set'
(summary(temp1)$counts +  summary(temp2)$counts) -
    summary(temp3)$counts

## assemble a multi-session object
## we fake the 2010 data by copying from 2005
## note must provide name each session
fakeCH <- ovenCH[['2005']]
MS.capthist(ovenCH, '2010' = fakeCH)
```

---

rbind.popn — *Combine popn Objects*

Description

Form a single popn object from two or more existing popn objects, or a list.

Usage

```
rbind.popn(..., renumber = TRUE)
```

Arguments

...	one or more popn objects, or a single list of 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. This is not an S3 method.

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.popn(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)
```

Arguments

`...`	one or more `traps` 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. Differences in the `usage` attribute are handled as follows. 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.

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, lab = 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 2009). `read.capthist` inputs data from text files and constructs a `capthist` object in one step using the functions `read.traps` and `make.capthist`.

Usage

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

write.capthist(object, filestem = deparse(substitute(object)),
     sess = "1", ndec = 2, covariates = FALSE, ...)
```

Arguments

`captfile`	name of capture data file
`trapfile`	name of trap layout file
`detector`	character value for detector type ('single', 'multi' etc.)
`fmt`	character value for capture format ('XY' or 'trapID')
`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'
`...`	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

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 `../doc/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.

`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.

Note

The DENSITY formats accommodate 'single', 'multi' and 'proximity' data. Data for the newer detector types ('count', 'signal', 'polygon', 'polygonX', 'transect' and 'transectX') 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 `../doc/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. (2009) *DENSITY 4.4: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand http://www.otago.ac.nz/density.

See Also

`read.traps`, `make.capthist`, `write.captures`, `write.traps`, `read.table`

Examples

```
## export ovenbird capture histories
write.capthist(ovenCH)
```

`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, ...)
```

Arguments

`file`	character string with name of text file
`data`	dataframe
`spacing`	spacing of grid points in metres
`...`	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.

If the grid cell size `spacing` 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.

Value

object of class `mask` with type 'user'

Note

The package **SPACECAP** uses a 'state-space' file in 'csv' text format with columns 'X_COORD', 'Y_COORD' and 'HABITAT'. Such a file may be input directly to `read.mask`; rows with HABITAT != 1 are dropped.

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.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.

Usage

```
read.traps(file = NULL, data = NULL, detector = "multi", covnames =
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
`...`	other arguments to pass to `read.table`

Details

Reads a text file in which the first column is a character string 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 2007), 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`. 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.

References

Efford, M. G. (2007) *Density 4.1: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand. http://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

```
## Replace file name with a valid local name and remove '#'
# read.traps ('c:\\myfolder\\mytraps.txt', detector='proximity')
## 'mytraps.txt' should have lines like this
# 1      365      365
# 2      365      395
# 3      365      425
# etc.
```

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, 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. A method exists for objects of class `capthist`.

Value

A matrix with number of columns equal to `length(columns)`.

See Also

`capthist`, `reduce.capthist`

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 Convert Detector Types*

Description

Combine columns (occasions) in a `capthist` object to create a new data set, possibly converting between detector types

Usage

```
## S3 method for class 'capthist'
reduce(object, columns = NULL, outputdetector =
    detector(traps(object)), select='last', dropunused = TRUE,
    verify = TRUE, sessions = NULL, by = 1, ...)
```

Arguments

`object`	`capthist` object
`columns`	list in which each component is a vector of subscripts for occasions to be pooled
`outputdetector`	character value giving detector type for output
`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)
`by`	number of old occasions in each new occasion
`...`	other arguments (not used currently)

Details

The first component of `columns` defines the columns of `object` for new occasion 1, the second for new occasion 2, etc. If `columns` is NULL then all occasions are output. Subscripts in a component of `columns` 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; a trap is considered 'used' on a pooled occasion if it was used on any contributing occasion.

* i.e., in the case of a single capture, use that capture; in the case of multiple 'competing' captures draw one at random.

If `columns` 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.

Value

An object of class `capthist` with number of columns equal to `length(occasions)`. The detector type is inherited from `object` unless a new type is specified with the argument `outputdetector`.

See Also

`capthist`, `subset.capthist`

Examples

```
tempcapt <- sim.capthist (make.grid(nx = 6, ny = 6), nocc = 6)
class(tempcapt)

pooled.tempcapt <- reduce(tempcapt, col = list(1,2:3,4:6))
summary (pooled.tempcapt)

pooled.tempcapt2 <- reduce(tempcapt, by = 2)
summary (pooled.tempcapt2)
```

`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, region = NULL, spacing = NULL, session = NULL,
    group = NULL, se.N = TRUE, alpha = 0.05, loginterval = TRUE,
    keep.region = FALSE)
```

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

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.

Group-specific N has yet to be implemented.

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	$\hat{N}$ (expected or realised, depending on row)
SE.estimate	standard error of $\hat{N}$
lcl	lower 100(1–alpha)% confidence limit
ucl	upper 100(1–alpha)% confidence limit
n	total number of individuals detected
E.n	expected number of individuals detected from region

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).

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 estimated as $R(N) = n + \int_B (1 - p.(X))D(X)dX$ (the second term represents undetected animals). 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 $E(N)$.

The expected number of individuals detected from region B is $E(n) = \int_B p.(X)D(X)dX$. $E(n) < n$ is a sign that the region did not contain all detected animals, and that other results are unreliable.

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.

The coding of region.N is somewhat complicated and we may yet discover scenarios not allowed for in this version (2.1.0). Please treat it as provisional and report any problems.

References

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

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

Examples

```
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.1, possumarea, spacing = 40,
    keep.region = TRUE)
temp
plot (attr(temp, 'region'))
```

---

rotate *Rotate Points*

Description

Rotate a set of points.

Usage

```
rotate (object, degrees, centrexy = NULL, ...)
```

Arguments

object	object that may be coerced to a numeric matrix
degrees	clockwise angle of rotation in degrees
centrexy	vector with xy coordinates of rotation centre
...	other arguments (not used)

Details

The first column of `object` holds the x coordinates of the points and the second holds the y coordinates. If `centrexy` is NULL then rotation is about (0,0).

A generic method, introduced for the class-specific method `rotate.traps`.

Value

A matrix with the location of each point rotated about the centre.

Author(s)

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

See Also

shift

Examples

```
temp <- matrix(runif (20) * 2 - 1, nc = 2)
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)
```

---

rotate.traps *Rotate Detectors*

Description

Rotate detectors while retaining other attributes.

Usage

```
## S3 method for class 'traps'
rotate(object, degrees, centrexy = NULL, ...)
```

Arguments

object	object of class traps
degrees	clockwise angle of rotation in degrees
centrexy	vector with x,y coordinates of point about which to rotate
...	other arguments (not used).

Details

May be used with flip.traps, rbind.traps and shift.traps to create complex geometries.

Value

An object of class traps with the location of each detector rotated about the centre.

Author(s)

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

See Also

traps, shift.traps

Examples

```
hollow1 <- make.grid(nx = 8, ny = 8, hollow = TRUE)
hollow2 <- make.grid(nx = 8, ny = 8, hollow = TRUE)
nested <- rbind (hollow1, rotate(hollow2, 45, c(70, 70)))
plot(nested, gridlines = FALSE)
```

---

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)

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
`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 AICc
`dmax`	threshold of dAICc 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.

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).

`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 AICc values are used to compute relative AIC model weights for all models within dmax AICc units of the best model.

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.

`score.test` cannot be used to compare models that differ in the arguments `scalesigma` or `scaleg0`.

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.

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)
```

```
## End(Not run)
```

`secr.design.MS` *Construct Detection Model Design Matrices and Lookups*

Description

Internal function used by `secr.fit`.

Usage

```
secr.design.MS(capthist, models, timecov = NULL, sessioncov = NULL,
    groups = NULL, dframe = NULL, naive = FALSE, bygroup = FALSE,
    ...)
```

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`.
`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
`bygroup`	logical if TRUE then the individual dimension of the parameter matrix is automatically collapsed to one row per group; if FALSE then the full dimensionality is retained (one row per individual)
`...`	other arguments passed to the R function `model.matrix`

Details

This is an internal **secr** function that you are unlikely ever to use. ... may be used to pass `contrasts.arg` to `model.matrix`.

Each real parameter is notionally different for each unique combination of session, individual, occasion and detector, i.e., for R sessions, n individuals, S occasions and K detectors there are *potentially* $R \times n \times S \times K$ 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.

Value

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 < $RnSK$) 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)

See Also

`D.designdata`

Examples

```
secr.design.MS (captdata, models = list(g0 = ~b))$designMatrices
secr.design.MS (captdata, models = list(g0 = ~b))$parameterTable
```

`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(D~1, g0~1, sigma~1),
    mask = NULL, buffer = NULL, CL = FALSE, detectfn = NULL,
    binomN = NULL, start = NULL, link = list(), fixed = list(),
    timecov = NULL, sessioncov = NULL, groups = NULL,
    dframe = NULL, details = list(), method = 'Newton-Raphson',
    verify = TRUE, trace = NULL, ...)
```

Arguments

`capthist`	`capthist` object including capture data and detector (trap) layout
`mask`	`mask` object
`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 'D', 'g0', 'sigma' and 'z', each a character string in {'log', 'logit', 'identity', 'sin'} for the link function of the relevant real parameter
fixed	list with optional components corresponding to each 'real' parameter (e.g., 'D', 'g0', 'sigma'), the scalar value to which parameter is to be fixed
model	list with optional components 'D', 'g0', 'sigma' and 'z', each symbolically defining a linear predictor for the relevant real parameter using formula notation
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.
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
trace	logical, if TRUE then output each evaluation of the likelihood, and other messages
...	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 (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), group (g), and others that appear in the covariates of the input data. See secr models and ../doc/secr-overview.pdf for more on defining models.

buffer and mask are alternative ways to define the region of integration (see mask).

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 nested model. In this case, a vector of starting beta values is constructed from the nested model and additional betas are set to zero. Mapping of parameters follows the default in score.test, but user intervention is not allowed.

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 Bernoulli
- binomN = 0 Poisson
- binomN < 0 negative binomial with size abs(binomN) – see `dnbinom`

The default with these detectors is to fit a Poisson distribution. The 'size' parameter of the negative binomial is not estimated: it must be supplied. `binomN` should be an integer unless negative.

`details` is used for various specialized settings –

`details$distribution` specifies the distribution of the number of individuals detected; 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'. See also Note.

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

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

`details$scalesigma` = TRUE causes sigma to be scaled by $\mathrm{D}^{-0.5}$.

`details$scaleg0` = TRUE causes g0 to be scaled by sigma^{-2}.

`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$param` = 1 causes the Gardner & Royle parameterisation of the detection model (*p0*, σ; Gardner et al. 2009) to be used for multi-catch detectors (default 0 for Borchers and Efford). This parameterisation does not allow detector covariates.

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. A feature of `nlm` is that it takes a large step early on in the maximization that may cause 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).

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.

Value

The function `secr.fit` returns an object of class secr. This has components

`call`	function call (as character string prior to secr 1.5)
`capthist`	saved input
`mask`	saved input
`detectfn`	saved input
`CL`	saved input
`timecov`	saved input
`sessioncov`	saved input

groups	saved input
dframe	saved input
design	reduced design matrices, parameter table and parameter index array for actual animals (see secr.design.MS)
design0	reduced design matrices, 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
D	if CL = FALSE, array of predicted densities of each group at each mask point in each session, dim(D) = c(nrow(mask), ngroups, nsessions), otherwise NULL
version	secr version number
starttime	character string of date and time at start of fit
proctime	processor time for model fit, in seconds

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.

print, AIC, vcov, and predict methods are provided. 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).

The component 'distribution' of argument 'details' 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.

Components 'version' and 'starttime' were introduced in version 1.2.7, and recording of the completion time in 'fitted' was discontinued.

A warning is generated when 'buffer' appears to be too small (predicted RB(D-hat) > 0.01). The prediction uses bias.D. No check is performed when 'mask' is specified, when 'verify' = FALSE, or when the detector type is 'polygon', 'transect', 'polygonX' or 'transectX'.

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. The alternative method = 'BFGS' often works better in these cases, or use details = list(hessian = 'fdhess').

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 or transect searches. Unpublished manuscript.

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.

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

See Also

capthist, mask, detection functions, print.secr, vcov.secr, AIC.secr, derived, predict.secr, verify

Examples

```
## 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
## stepmax is passed to nlm (not usually needed)

secr0 <- secr.fit (detections, stepmax = 50)
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 dAICc  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
```

---

secr.make.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

```
secr.make.newdata(object)
```

Arguments

`object`	fitted secr model object

Details

`secr.make.newdata` is used by `predict` in lieu of user-specified 'newdata'. There is seldom any need to call `secr.make.newdata` directly.

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
secr.make.newdata(secrdemo.b)
```

`secr.model` *Spatially Explicit Capture-Recapture Models*

Description

A family of capture–recapture models (e.g. SECR) may include submodels that constrain variation in core parameters and include the effects of covariates. The language of generalised linear models is convenient for describing submodels (e.g., Huggins 1989, Lebreton et al. 1992). Each parameter is treated as a linear combination of effects on its transformed ('link') scale. This is useful for combining effects because, given a suitable link function, any combination maps to a feasible value of the parameter. The logit scale has this property for probabilities in (0,1), and the natural log scale works for positive parameters i.e. (0, +Inf).

Submodels for spatially explicit capture–recapture in **secr** are defined symbolically using the R formula notation. A separate linear predictor is used for each core parameter. Core parameters are 'real' parameters in the terminology of MARK, and **secr** uses that term to reduce confusion. Four real parameters are commonly modelled in **secr**: D (density), g0, sigma and z. Only the last three real parameters, the ones jointly defining detection probability as a function of location, can be estimated directly when the model is fitted by maximizing the conditional likelihood. D is then a derived parameter. 'z' is a shape parameter used only when the detection function requires three parameters. Other real parameters are used for acoustic models (beta0, beta1; `../doc/secr-sound.pdf`) and for the mixture proportion (pmix) in finite mixture models (`../doc/secr-finitemixtures.pdf`).

Each real parameter has a linear predictor of the form

y = X * beta,

where y is vector of parameter values on the link scale, X is a design matrix of predictor values, beta is a vector of coefficients, and '*' stands for matrix multiplication. The elements of beta are estimated when we fit the model; in MARK these are called 'beta parameters' to distinguish them from the 'real' parameter values in y. X has one column for each element of beta. To repeat: there is an X and a beta for each real parameter; elsewhere in the documentation we use 'beta' to refer to the vector got by concatenating *all* the parameter-specific beta's. We now describe design matrices in more detail.

[Some variations on the basic SECR model do not fit easily into this framework. An example is the choice of detection function (halfnormal vs hazard-rate). These are treated as higher-level choices.]

Design matrices

The design matrix contains a column of '1's (for the constant or intercept term) and additional columns as needed to describe the effects in the submodel. Depending on the model, these may be continuous predictors (e.g. air temperature to predict occasion-to-occasion variation in g0), indicator variables (e.g. 1 if animal i was caught before occasion s, 0 otherwise), or coded factor levels.

Within `secr.fit`, a design matrix is constructed automatically from the input data (`capthist`) and the model formula (e.g. `model$g0`) in a 2-stage process. First, a data frame is built containing 'design data' with one column for each variable in the formula. Second, the R function `model.matrix()` is used to construct the design matrix. This process is hidden from the user. The design matrix will have at least one more column than the design data, and more if the formula includes interactions or factors with more than two levels. For a good description of the general approach see the documentation for RMark (Laake and Rexstad 2008). The key point is that the necessary design data can be either extracted from the inputs (`capthist` and `mask`) or generated automatically (e.g. indicator of previous capture, mentioned in the previous paragraph).

Real parameters fall into two groups: density (D) and detection (g0, sigma and z). Density and detection parameters are subject to different types of effect, so they use different design matrices and are described separately here `secr detection models` and here `secr density models`.

Note

The structure of **secr** precludes certain types of model. Unlike density, detection parameters (g0, sigma etc.) cannot be modelled as varying in space *per se*, whether continuously or discretely (e.g. as a function of habitat class). However, such variation may be modelled between detectors or between sessions. As an example, consider a measure of vegetation cover in a 50-m circle centred on each detector. This may be used as a detector covariate in models for g0 or sigma. A 'detector-centred' view of habitat effects is almost as sensible as an 'animal-centred' view; the one reservation is that the spatial scale (radius of the circle) is arbitrary rather than being driven by sigma as you might like. Perhaps this could be fixed in future versions by computing the trap covariate 'on the fly' from covariates in the habitat mask, given the current magnitude of sigma.

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. Available online at http://www.phidot.org.

---

secr.model.density *Density Models*

Description

SECR can fit an inhomogeneous Poisson model to describe the distribution of animals. This may be viewed as a surface of expected density across the study area.

The log likelihood is evaluated in `secr.fit` by summing values at points on a 'habitat mask'. Each point in a habitat mask represents a grid cell of potentially occupied habitat (their combined area may be almost any shape and may include disjunct patches). The full design matrix for density (D) has one row for each point in the mask. The design matrix has one column for the intercept (constant) term and one for each predictor.

Predictors may be based on Cartesian coordinates (e.g. 'x' for an east-west trend), a continuous habitat variable (e.g. vegetation cover) or a categorical (factor) habitat variable. Predictors must be known for all points in the mask (non-habitat excluded). The variables 'x', 'y', 'x2', 'y2', 'xy', 'session' and 'g' are provided automatically. Other covariates should be named columns in the 'covariates' attribute of the habitat mask.

The fitted model for density is linear on the link scale (see the `link` argument of `secr.fit`. The default link for density is 'log'.

Variable	Description	Data source
x	x-coordinate	automatic
y	y-coordinate	automatic
x2	x-coordinate^2	automatic
y2	y-coordinate^2	automatic
xy	x-coordinate * y-coordinate	automatic
session	session factor	automatic
Session	session number 0:(R-1)	automatic
g	group factor	automatic
[user]	mask covariate	`covariates (mask)` as named in formula

The submodel for density (D) is a named component of the list used in the `model` argument of `secr.fit`. It is expressed in R formula notation by appending terms to $\sim$.

Note

Note that no density model is fitted when `secr.fit` is called with CL = TRUE.

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 models, secr detection models, secr.fit

Examples

```
list(D = ~ 1)      ## constant density (homogeneous Poisson)
list(D = ~ x)      ## east-west trend
list(D = ~ cover) ## requires 'cover' as a mask covariate
```

`secr.model.detection` *Models for Detection Parameters*

Description

For spatially explicit capture–recapture estimation of a closed population, we model the detection of individual i on occasion s at detector k. Given n observed individuals on S occasions at K detectors there are therefore $n.S.K$ detection probabilities of interest. We can think of these as elements of a 3-dimensional array. Strictly, we are also interested in the detection probabilities of unobserved individuals, but these are estimated only by extrapolation from those observed so we do not consider them in the array.

In a null (constant) model, all $n.S.K$ detection probabilities are the same. The conventional sources of variation in capture probability (Otis et al. 1978) appear as variation in the n dimension ('individual heterogeneity' h), in the S dimension ('time variation' t) or as a particular interaction in these two dimensions ('behavioural response to capture' b). Combined effects are possible.

Spatially explicit capture–recapture introduces two sorts of additional complexity. Firstly, detection probability is no longer a scalar (even for a particular animal, occasion and detector combination); it is described by the detection function, which may have two parameters (e.g. g0, sigma for half-normal), three parameters (e.g. g0, sigma, z for the hazard-rate function), or potentially more.

Secondly, many more types of variation are possible. Any of the parameters of the detection function may vary with respect to individual (i), occasion (s) or detector (k). For example, there may be a covariate associated with trap location that influences detection probability.

The full design matrix for each detection submodel has one row for each combination of i, s and k (animal, occasion and trap). Allowing a distinct probability for each animal (the 'n' dimension) may seem excessive, as continuous individual-specific covariates are feasible only when a model is fitted by maximizing the conditional likelihood (cf Huggins 1989). However, the full $n.S.K$ array is convenient for coding both group membership (Lebreton et al. 1992, Cooch and White 2008) and experience of capture, even when individual-level heterogeneity cannot be modelled.

Variation between 'sessions' and between latent classes in a finite mixture adds two further dimensions: in principle there is an $n.S.K$ array for each latent class (classes are numbered 1..M), and an $n.S.K.M$ array for each session (sessions are numbered 1..R). The full design matrix has $n.S.K.M.R$ rows. We do not expand on this here.

Specifying effects on detection parameters

Effects on parameters of detection probability are specified with R formulae using standard variable names or named covariates supplied by the user. The formula for each detection parameter (g0, sigma, z) may be constant ($\sim$ 1, the default) or some combination of terms in standard R formula notation (see `formula`).

Variable	Description	Data source	Dim
t	time factor (one level for each occasion)	automatic	S
T	time trend (integer covariate 0:(S-1))	automatic	S

tcov	default time covariate	timecov[,1]	S
kcov	default trap covariate	covariates (traps)[,1]	K
b	learned response	`capthist`	$n.S$
B	transient (Markovian) response	`capthist`	$n.S$
g	group	see below	n
h2	2-class mixture	–	2
h3	3-class mixture	–	2
session	session factor (one level for each session)	automatic	R
Session	session number 0:(R-1)	automatic	R
[user]	individual covariate	`covariates (capthist)`	n
[user]	session covariate	`sessioncov`	R
[user]	time covariate	`timecov`	S
[user]	detector covariate	`covariates (traps)`	K

The classic 'learned response' is a step change following first detection; this is implemented with the predictor variable 'b' which is FALSE up to and including the time of first capture and TRUE afterwards. An alternative is a response that depends only on detection at the last opportunity ('B').

Groups ('g') are defined by the interaction of the `capthist` categorical (factor) individual covariates identified in `secr.fit` argument 'groups'. Groups are redundant with conditional likelihood because individual covariates of whatever sort (continuous or categorical) may be included freely in the model.

Individual heterogeneity ('h' in the notation of Otis et al. 1978) may modelled by treating any detection parameter as a 2-part or 3-part finite mixture e.g. g0 $\sim$ h2. See `../doc/secr-finitemixtures.pdf`.

Any other variable name appearing in a formula is assumed to refer to a user-defined predictor. These will be interpreted by searching for name matches in the dataframes of individual, session, time and trap covariates, in that order (remembering that individual covariates other than groups are allowed only when the model is fitted by maximizing the conditional likelihood). The type of the predictor is inferred from the data frame in which it first occurs. Thus if the model included the formula 'g0 ~ wetness', and 'wetness' was a column in the data frame of time covariates (timecov), then 'wetness' would be interpreted as a time covariate, and a column of the same name in covariates(traps) would be ignored. In this case, renaming the column in timecov would expose the traps covariate, and 'wetness' would be interpreted as an attribute of detectors, rather than sample intervals. This is a good reason to give covariates distinctive names!

The design matrix for detection parameters may also be provided manually in the argument `dframe`. This feature is untested.

The submodels for 'g0', 'sigma' and 'z' are named components of the `model` argument of `secr.fit`. They are expressed in R formula notation by appending terms to $\sim$. The name of the response may optionally appear on the left hand side of the formula (e.g. g0$\sim$ b).

Note

The parameter 'z' was previously called 'b'; it was renamed to avoid confusion with the predictor b used in a formula for a learned trap response.

References

Cooch, E. and White, G. (eds) (2008) *Program MARK: A Gentle Introduction.* 6th edition. Available online at `http://www.phidot.org`.

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

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

Lebreton, J.-D., Burnham, K. P., Clobert, J., 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 models, secr density models, secr.fit

Examples

```
## constant (null) model
list(g0 = ~1, sigma = ~1)

## both detection parameters change after first capture
list(g0 = ~b, sigma = ~b)

## group-specific parameters; additive time effect on g0
## groups are defined via the 'groups' argument of secr.fit
list(g0 = ~ g + t, sigma = ~ g)

## g0 depends on trap-specific covariate
list(g0 = ~ kcov)
```

---

secrdemo *SECR Models Fitted to Demonstration Data*

Description

Demonstration data from program Density are provided both 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 2007).

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

Source

Efford, M.G. (2007) Density 4.1: software for spatially explicit capture-recapture. Department of Zoology, University of Otago, Dunedin, New Zealand. http://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

```
## construct a traps object from raw trap data
## this dataset uses fmt = 'XY' (x-y coordinates included in
## both trap and capture files), but fmt = 'trapID' (capture file
## uses trap identifier) is simpler

temptrap <- read.traps(data = trapXY, detector = 'single')
plot(temptrap)

## construct a capthist object
captdata <- make.capthist(captXY, temptrap, fmt='XY')
plot(captdata, tracks = TRUE)

## display the null model fit, using the print method for secr
secrdemo.0
```

---

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.

Sessions are a recent addition to **secr** and the documentation and testing of session capability is therefore less advanced than for other features.

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)
```

`shift` *Shift Points*

Description

Translate an array of points.

Usage

```
shift (object, shiftxy, ...)
```

Arguments

`object`	a 2-column matrix or object that can be coerced to a matrix
`shiftxy`	vector of x and y displacements
`...`	other arguments (not used)

Details

This is a generic function. The default method is redundant, but the method for `traps` objects may be useful.

Value

A matrix with the location of each point shifted by the desired amount.

Author(s)

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

See Also

`rotate`, `flip`

Examples

```
temp <- matrix(runif (20) * 2 - 1, nc = 2)
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)
```

shift.traps *Shift Detectors*

Description

Translate detectors while retaining other attributes.

Usage

```
## S3 method for class 'traps'
shift(object, shiftxy, ...)
```

Arguments

`object`	object of class `traps`
`shiftxy`	vector with displacements in x and y directions
`...`	other arguments (not used)

Details

May be used with `rbind.traps` and `rotate.traps` to create complex geometries.

Value

An object of class `traps` with the location of each detector shifted by the desired amount.

Author(s)

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

See Also

`traps`, `rotate.traps`, `flip.traps`

Examples

```
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, lab = TRUE)
```

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, renumber = TRUE, seed = NULL)
sim.resight(traps, ..., q = 1, pID = 1, unmarked = TRUE,
    nonID = TRUE)
```

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 or a list with named components 'D' (density) and 'buffer'
`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)
`renumber`	logical for whether output rows should labeled sequentially (TRUE) or retain the numbering of the population from which they were drawn (FALSE)
`seed`	an object specifying if and how the random number generator should be initialized ('seeded')
`...`	arguments to pass to `sim.capthist`
`q`	number of marking occasions
`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'

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 a numeric code (`detectfn`). Parameters may vary with time - for this provide a vector of length `noccasions`. 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.

`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).

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`.

`sim.resight` generates mark-resight data for 'q' marking occasions followed by 'noccasions – q' sighting occasions. `sim.capthist` is first called with the arguments 'traps' and .... The detector type must be 'proximity'. The 'usage' attribute of `traps` is ignored at present, so the same detectors are operated on all occasions. Any detection-parameter vector of length 2 in ... is interpreted as providing differing constant values for the marking and sighting phases.

Value

For `sim.capthist`, an object of class `capthist`, a matrix or 3-dimensional array with additional attributes. Rows represent individuals and columns represent occasions; the third dimension, used when detector type = 'proximity', codes presence or absence at each detector. For trap detectors ('single', 'multi') each entry in `capthist` is either zero (no detection) or the sequence number of the trap.

The initial state of the R random number generator is stored in the 'seed' attribute.

For `sim.resight`, an object of class `capthist`, always a 3-dimensional array, with additional attributes Tu and Tm containing counts of 'unmarked' and 'marked, not identified' sightings.

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`.

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`

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)))

## multiple sessions
grid4 <- make.grid(nx = 2, ny = 2)
temp <- sim.capthist (grid4, popn = list(D = 1), nsessions = 20)
summary(temp, terse = TRUE)
```

`sim.popn` *Simulate 2-D Population*

Description

Simulate a Poisson process representing the locations of individual animals.

Usage

```
sim.popn (D, core, buffer = 100, model2D = 'poisson',
    buffertype = 'rect', covariates = list(sex = c(M=0.5, F=0.5)),
    number.from = 1, Ndist = 'poisson', nsession = 1, details =
    NULL, seed = NULL)
```

Arguments

D	density animals / hectare (10 000 m\^2)
core	data frame of points defining the core area
buffer	buffer radius about core area
model2D	character string for 2-D distribution ('poisson', 'cluster', 'IHP', 'coastal')
buffertype	character string for buffer type
covariates	list of named covariates
number.from	integer ID for animal
Ndist	character string for distribution of number of individuals
nsession	number of sessions to simulate
details	optional list with additional parameters
seed	value for setting .Random.seed - either NULL or an integer

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.

A notional random covariate 'sex' is generated by default.

Each element of `covariates` defines a categorical (factor) covariate with the given probabilities of membership in each class. No mechanism is provided for generating continuous covariates, but these may be added later (see Examples).

Ndist may 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.

If `model2D = 'cluster'` then the simulated population approximates a Neyman-Scott clustered Poisson distribution. Ancillary parameters are passed as components of `details`: details$mu is the fixed number of individuals per cluster and details$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. Generate mu offspring for each parent and locate them by adding random normal error to each parent coordinate
4. Apply toroidal wrapping to ensure all offspring locations are inside the buffered area

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 a vector of length equal to the number of cells (rows) in `core`. The number of individuals in each cell is Poisson-distributed with mean DA where A is the cell area (an attribute of the mask). `buffertype` and `buffer` are ignored, as the extent of the population is governed entirely by the mask in `core`.

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.

The random number seed is managed as in `simulate.lm`.

Value

An object of class 'popn', 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.

Note

Other `buffertypes` will be defined later (e.g. convex hull, concave)

References

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.

See Also

`popn`, `plot.popn`, `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
oldpar <- 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(oldpar)

## 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)
```

---

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, chat = 1, ...)

sim.secr(object, nsim = 1, extractfn = function(x) c(deviance =
    deviance(x), df = df.residual(x)), seed = NULL, data = NULL,
    tracelevel = 1, hessian = "none", start = object$fit$par)
```

Arguments

`object`	an secr object
`nsim`	number of replicates
`seed`	value for setting .Random.seed - either NULL or an integer
`chat`	real value for overdispersion parameter
`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
...	other arguments (not used)

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). 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.

simulate.secr does not yet work with models fitted using conditional likelihood (object$CL = TRUE). Detector type is determined by detector(traps(object$capthist)), which should be one of 'single', 'multi', 'proximity', 'areasearch' or 'count'.

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'.

sim.capthist is a more direct way to simulate data from a null model (i.e. one with constant parameters for density and detection).

Value

For simulate.secr, a list of data sets ('capthist' objects). This list has class c('list','secrdata'); 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('list','secrlist')). 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'.

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 `object$start`.

See Also

`sim.capthist`, `secr.fit`, `simulate`

Examples

```
## previously fitted model
simulate(secrdemo.0, nsim = 2)

## Not run:

## 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)
```

`skink` *Skink Pitfall Data*

Description

Data from a study of skinks (*Oligosoma infrapunctatum* and *O. lineoocellatum*) in New Zealand.

Usage

```
data(skink)
```

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
oldpar <- 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(oldpar)
```

`sort.capthist` *Sort Rows of capthist 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,...)
```

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.

Value

`capthist` 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']]
```

SPACECAP *Exchange data with SPACECAP package*

Description

Data in a single-session **secr** `capthist` object may be written directly to the 'csv' format used by **SPACECAP**, a package for Bayesian spatially explicit capture–recapture (Singh et al. 2010). Data in csv format may also be read to construct a `capthist` object for analysis in **secr**.

Usage

```
write.SPACECAP(object, mask = NULL, buffer = 100, ndec = 2,
    filestem = "")
read.SPACECAP(AC, TD, detector = "proximity", session = "1")
```

Arguments

`object`	`capthist` object with the captures and trap locations to export
`mask`	`mask` object to use for state-space file
`buffer`	width of buffer in metres to use when creating a mask if none is specified
`ndec`	number of digits after decimal point for coordinates of mask on output
`filestem`	character value used to form names of output files
`AC`	character value giving name of ‘animal capture’ .csv file
`TD`	character value giving name of ‘trap deployment’ .csv file
`detector`	detector type (‘proximity’ or ‘count’)
`session`	character value to use as session name

Details

If successful, `write.SPACECAP` creates three output files with names ending in ‘AC.csv’,‘TD.csv’ and ’SS.csv’. These are respectively the ‘Animal Capture’, ‘Trap Deployment’ and ‘State-Space’ files required by **SPACECAP**.

Value

`write.SPACECAP` is used for its side effect of writing the required csv files. `read.SPACECAP` returns a `capthist` object.

Note

State-space csv files may be imported with `read.mask`.

References

Singh, P., Gopalaswamy, A. M., Royle, A. J., Kumar, N. S. and Karanth, K. U. (2010) SPACECAP: A program to estimate animal abundance and density using Bayesian spatially explicit capture-recapture models. Version 1.0. Wildlife Conservation Society - India Program, Centre for Wildlife Studies, Bangalure, India.

See Also

`capthist`, `mask`, `read.mask`

Examples

```
## Not run:

## coerce data to proximity detector type for export
demo <- reduce(captdata, output = 'proximity')
write.SPACECAP (demo, filestem = 'demo')

## now read back the data just exported...
temp <- read.SPACECAP ('demoAC.csv', 'demoTD.csv')
temp <- reduce(temp, output = 'single')
summary (temp)
summary (captdata)
## should match exactly

## End(Not run)
```

`spacing` *Detector or Mask Spacing*

Description

Extract or replace the spacing attribute of a detector array or mask.

Usage

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

Arguments

`object`	object with 'spacing' attribute e.g. `traps`
`value`	numeric value for spacing
`...`	other arguments (not used)

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)
```

`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

```
data(stoatDNA)
```

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 treated as 'proximity' detectors which allow an individual to be detected at multiple detectors on one occasion (day), although there are no multiple detections in this dataset. Three pre-fitted models are included: `stoat.model.HN`, `stoat.model.HZ`, and `stoat.model.EX` (with halfnormal, hazard-rate and negative exponential detection functions, respectively). Some components of the fitted secr models have been removed with `trim` to save space.

Object	Description
stoatCH	capthist object
stoat.model.EX	fitted secr model – null, exponential detection function
stoat.model.HN	fitted secr model – null, halfnormal detection function
stoat.model.HZ	fitted secr model – null, hazard-rate 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.

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. Available on-line at `http://www.newzealandecology.org`.

See Also

`capthist`, `detection functions`, `secr.fit`

Examples

```
summary(stoatCH)

## Not run:
stoat.model.HN <- secr.fit(stoatCH, buffer = 1000, detectfn = 0)
stoat.model.HZ <- secr.fit(stoatCH, buffer = 1000, detectfn = 1)
stoat.model.EX <- secr.fit(stoatCH, buffer = 1000, detectfn = 2)
confint(stoat.model.HN, 'D')
## Profile likelihood interval(s)...
##          lcl        ucl
## D 0.01381419 0.04563511

## 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)
plot(stoat.model.HZ, xval = xv, limits = FALSE, lty = 3, add = TRUE)

## review density estimates
collate(stoat.model.HZ, stoat.model.HN, stoat.model.EX,
    realnames='D', perm=c(2,3,4,1))
model.average(stoat.model.HZ, stoat.model.HN, stoat.model.EX,
    realnames='D')
```

---

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, dropnull = TRUE, dropunused =
    TRUE, renumber = FALSE, ...)
## S3 method for class 'capthist'
split(x, f, drop = FALSE, prefix = 'S', ...)
```

Arguments

`x`	object of class `capthist`
`subset`	vector of subscripts to select rows (individuals)
`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
`dropnull`	logical for whether null (all-zero) capture histories and occasions with no detections should be dropped
`dropunused`	logical for whether never-used detectors should be dropped
`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
`...`	other arguments (not used currently)

Details

Subscript vectors may be either logical (length equal to the relevant dimension of `x`) or integer-valued. Subsetting is applied to attributes (e.g. `covariates`, `traps`) as appropriate. The default action is to include all rows, columns and traps 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`.

To combine (pool) occasions use `reduce.capthist`. There is no equivalent of `unlist` for lists of `capthist` objects.

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.

See Also

`capthist`, `rbind.capthist`, `reduce.capthist`

Examples

```
tempcapt <- sim.capthist (make.grid(nx=6, ny=6), nocc=6)
summary(subset(tempcapt, occ=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'), nocc = 6)
tempcapt3 <- subset(tempcapt2, traps = sample(1:36, 18,
    replace=FALSE))
summary(tempcapt3)
plot(tempcapt3)

split (tempcapt2, f = sample (c('A','B'), nrow(tempcapt2),
    replace = TRUE))
```

`subset.mask` *Subset Mask Object*

Description

Retain selected rows of a `mask` object.

Usage

```
## S3 method for class 'mask'
subset(x, subset, ...)

## S3 method for class 'mask'
rbind(...)
```

Arguments

`x`	`mask` object
`subset`	numeric or logical vector to select rows of mask
`...`	two or more `mask` objects (rbind only)

Details

The subscripts in `subset` may be of type integer, character or logical as described in `Extract`.

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 `rbind`, an object of class 'mask' with all unique rows from the masks in ..., and 'type' attribute set to 'rbind'.

See Also

`mask`

Examples

```
tempmask <- make.mask(make.grid())
OK <- (tempmask$x + tempmask$y) > 100
tempmask <- subset(tempmask, subset = OK)
plot(tempmask)
```

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', ...)
```

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
`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

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.

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.

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)
```

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 (Efford and Marques unpubl).

Usage

```
suggest.buffer(object, detectfn = NULL, detectpar = NULL,
    noccasions = NULL, RBtarget = 0.001, interval = NULL, ...)

bias.D (buffer, traps, detectfn, detectpar, noccasions,
    control = NULL)
```

Arguments

`object`	'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
`RBtarget`	numeric target for relative bias of density estimate
`interval`	a vector containing the end-points of the interval to be searched
`...`	other argument(s) passed to `bias.D`
`buffer`	vector of buffer widths
`traps`	'traps' object
`control`	list of mostly obscure numerical settings (see Details)

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`.

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`.

The package **gpclib** may be used for more accurate estimates of the length of buffer contours (this does not appear to be critical). Some components of `control` are specific to this part of the algorithm (`ntheta`, `ninterp`, `maxinterp`).

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).

Note

The algorithm in `bias.D` uses one-dimensional numerical integration of a polar approximation to site-specific detection probability. By default, and when **gpclib** is not available, 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`
`use.gpclib = FALSE`	logical to use **gpclib** if installed
`ntheta = 60`	integer value for smoothness of contours (straight-line approximation to arc lengths)
`ninterp = 5`	number of points to interpolate between trapspacing/2 and trapspacing/2 ^0.5 on the contour-length vs buffer curve

See Also

`mask`, `make.mask`, `mask.check`, `esa.plot`

Examples

```
temptraps <- make.grid()
```

```
detpar <-  list(g0 = 0.2, sigma = 25)
suggest.buffer(temptraps, 'halfnormal', detpar, 5)

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
## Not run:
esa.plot (temptraps, max.buffer = 150, spacing = 4, detectfn = 0,
  detectpar = detpar, noccasions = 5, as.density = F)

## End(Not run)

## compare detection histories and fitted model as
suggest.buffer(ovenCH)
suggest.buffer(ovenbird.model.1)
```

---

summary.capthist *Summarise Detections*

Description

Concise description of `capthist` object.

Usage

```
## S3 method for class 'capthist'
summary(object, terse = FALSE, ...)

## S3 method for class 'summary.capthist'
print(x, ...)

counts(CHlist, counts = 'M(t+1)')
```

Arguments

`object`	`capthist` object
`terse`	logical; provide only summary counts for multi-session object
`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

`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` and `object` contains multiple sessions, a dataframe of counts per session.

See Also

`dbar`, `RPSV`, `capthist`

Examples

```
temptrap <- make.grid(nx = 5, ny = 3)
summary(sim.capthist(temptrap))
summary(sim.capthist(temptrap))$counts['n',]
```

`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.traps — *Summarise Detector Array*

Description

Concise description of `traps` object.

Usage

```
## S3 method for class 'traps'
summary(object, getspacing = TRUE, ...)
## S3 method for class 'summary.traps'
print(x, terse = FALSE, ...)
```

Arguments

`object`	`traps` object
`getspacing`	logical to calculate spacing of detectors from scratch
`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
```

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 = F, tb = F, ...)
rotate (object, degrees, centrexy = NULL, ...)
shift (object, shiftxy, ...)

## S3 method for class 'traps'
 flip(object, lr = F, tb = F, ...)
## 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 = F, tb = F, ...)
## S3 method for class 'popn'
rotate(object, degrees, centrexy = NULL, ...)
## S3 method for class 'popn'
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
oldpar <- 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, lab = TRUE, offset = 7, gridl = FALSE)
plot(traps2, border = 5, lab = TRUE, offset = 7, gridl = FALSE)
par(oldpar)

## 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, lab = 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 = 'SRS', edgemethod = 'clip', samplefactor = 2,
    ranks = NULL, rotation = NULL, detector, 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
`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`)
`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

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` may be a two-column matrix or dataframe of x-y coordinates for the boundary, or a SpatialPolygonsDataFrame object from **sp**.

`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.

`edgemethod` may be 'clip' (reject individual detectors), 'allowoverlap' (no action) or 'allinside' (reject whole cluster if any component is outside `region`). 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').

The package **sp** is required. GRTS samples require function `grts` in package **spsurvey** of Olsen and Kincaid. Much more sophisticated sampling 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.

Value

`trap.builder` produces an object of class 'traps'. A covariate 'cluster' retains the sequence number of the cluster from which a location is derived. A covariate 'trapnum' retains the within-cluster sequence number of the detector.

`method = 'GRTS'` causes messages to be displayed regarding the stratum (always 'None'), and the initial, current and final number of levels from the GRTS algorithm.

`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 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.

References

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

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)

## GRTS sample of mini-grids within a rectangle
## edgemethod = 'allinside' avoids truncation at edge
minigrid <- make.grid(nx = 3, ny = 3, spacing = 50,
    detector = 'proximity')
tempgrid <- trap.builder (n = 20, cluster = minigrid,
    method = 'GRTS', edgemethod = 'allinside', region =
    cbind(x = c(0,6000,6000,0), y = c(0,0,6000,6000)),
    plt = TRUE)

## one detector in each 100-m grid cell -
## a form of stratified simple random sample
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')
plot(temp, border = 0)  ## default grid is 100 m

## regular lattice of mini-arrays
tempgrid <- trap.builder (cluster = minigrid , method =
    'all', frame = expand.grid(x = seq(1000, 5000, 2000),
    y = seq(1000, 5000, 2000)), plt = TRUE)

## simulate some data
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)
## Not run:
tempmask1 <- make.mask(tempgrid, type = 'clusterrect',
```

```
    buffer = 200, spacing = 10)
secr.fit(tempcapt, mask = tempmask1)         ## 241.52 s

tempmask2 <- make.mask(minigrid, spacing = 10)
secr.fit(mash(tempcapt), mask = tempmask2)   ## 2.85 s
## density estimate is adjusted automatically
## for the number of mashed clusters (9)

## End(Not run)

## two-phase design: preliminary sample across region,
## followed by selection of sites for intensive grids
## Not run:
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)

## 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 average `spacing`, trap-specific covariates (`covariates`), and a matrix of binary (0/1) codes indicating whether each detector was used on each occasion (`usage`). If usage is specified, at least one detector must be 'used' 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`), shift an array (`shift.traps`), and to 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. (2007) *Density 4.1: software for spatially explicit capture–recapture*. Department of Zoology, University of Otago, Dunedin, New Zealand. http://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`, `plot.traps`, `secr.fit`, `spacing`, `detector`, `covariates`, `trap.builder`

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. `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)
```

`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('mask', 'design', 'design0', 'D'),
    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))
```

usage Detector Usage

Description

Extract or replace usage information of a `traps` object.

Usage

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

Arguments

`object`	a `traps` object
`value`	a matrix of traps x occasions 1 if trap[i] used on occasion[j], zero otherwise.
`...`	other arguments (not used)

Details

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

Value

usage(object) returns the usage matrix of the `traps` object. `usage(object)` may be NULL.

See Also

`traps`

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)
```

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

```
## previosuly 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. More than one capture in single-catch trap(s)
7. More than one detection per detector per occasion at proximity detector(s)
8. Signal detector signal(s) less than threshold or invalid threshold
9. Number of rows in 'traps' object not compatible with reported detections
10. Number of rows in dataframe of individual covariates differs from capthist
11. Number of occasions in usage matrix differs from capthist
12. Detections at unused detectors
13. Number of coordinates does not match number of detections ('polygon', 'polygonX', 'transect' or 'transectX' detectors)
14. Coordinates of detection(s) outside polygons ('polygon' or 'polygonX' detectors)
15. Coordinates of detection(s) do not lie on any transect ('transect' or 'transectX' detectors)

`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

`verify.mask`

1. Valid x and y coordinates
2. Number of rows in covariates dataframe differs from expected

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`

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 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,  ...)

write.traps(object, file = '', deblank = TRUE,  header = TRUE,
    ndec = 2, covariates = FALSE, ...)
```

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
`...`	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.

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 `project` from **rgdal**.

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, http://www.remotesensing.org/geotiff/proj_list/ and the help for related package **rgdal**. 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 from http://www.gpsbabel.org/. 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). Or ';C:/Program Files/gpsbabel' on 32-bit systems.

See Also

`make.systematic`, `readGPS`, `project`

Examples

```
## Example using shapefile "possumarea.shp" in
## "extdata" folder. As 'cluster' is not specified,
## the grid comprises single multi-catch detectors.

## Not run:
require(maptools)
setwd(system.file("extdata", package = "secr"))
possumarea <- readShapePoly('possumarea')
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
## an example for New Zealand Transverse Mercator with
## datum NZGD2000.

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