Package ‘openCR’

January 17, 2022

Type Package
Title Open Population Capture-Recapture
Version 2.2.2
Depends R (>= 3.5.0), secr (>= 4.4.0)
Imports utils, MASS, nlme, parallel, stats, Rcpp (>= 0.12.14),
       stringr, plyr, abind, methods, RcppParallel (>= 5.1.1)
LinkingTo Rcpp, RcppParallel, BH
Suggests knitr, RMark, rgdal, R2ucare, rmarkdown, testthat
VignetteBuilder knitr
Date 2022-01-17
Description Functions for non-spatial and spatial open-population capture-recapture analysis.
License GPL (>= 2)
LazyData yes
LazyDataCompression xz
SystemRequirements GNU make
URL https://www.otago.ac.nz/density/
BugReports https://github.com/MurrayEfford/openCR/issues/
NeedsCompilation yes
Author Murray Efford [aut, cre] (<https://orcid.org/0000-0001-5231-5184>)
Maintainer Murray Efford <murray.efford@otago.ac.nz>
Repository CRAN
Date/Publication 2022-01-17 06:02:42 UTC

R topics documented:

  openCR-package ............................................................ 3
  age.matrix ............................................................... 4
  AIC.openCR ............................................................... 5
R topics documented:

classMembership ..................................................  8
cloned.fit ..........................................................  9
cumMove ............................................................ 10
derived .............................................................. 12
dipperCH ............................................................. 14
expected.d .......................................................... 16
Field vole ............................................................ 17
gonodontisCH ........................................................ 20
Internal ............................................................... 22
JS.counts ............................................................ 24
JS.direct ........................................................... 26
LL.surface ........................................................... 27
make.kernel ........................................................ 29
make.table .......................................................... 32
makeNewData ....................................................... 33
matchscale .......................................................... 34
Microtus ............................................................. 35
miscellaneous ....................................................... 38
modelAverage ....................................................... 39
Movement models .................................................. 41
moving.fit .......................................................... 43
openCR-defunct ..................................................... 45
openCR-deprecated ................................................. 45
openCR.design ....................................................... 46
openCR.fit .......................................................... 48
openCRlist .......................................................... 54
par.openCR.fit ...................................................... 55
pkernel ............................................................... 57
plot.derivedopenCR ................................................. 58
plot.openCR ........................................................ 59
PPNpossums ........................................................ 61
predict.openCR ..................................................... 62
print.derivedopenCR .............................................. 63
print.openCR ........................................................ 64
read.inp ............................................................. 67
rev.capthist ........................................................ 68
simulation ........................................................... 69
squeeze .............................................................. 73
strata ................................................................. 74
stratify ............................................................. 75
ucare.cjs ............................................................ 76

Index ................................................................. 78
Description

Functions for non-spatial open population analysis by Cormack-Jolly-Seber (CJS) and Jolly-Seber-Schwarz-Arnason (JSSA) methods, and by spatially explicit extensions of these methods. The methods build on Schwarz and Arnason (1996), Borchers and Efford (2008) and Pledger et al. (2010) (see vignette for more comprehensive references and likelihood). The parameterisation of JSSA recruitment is flexible (options include population growth rate $\lambda$, per capita recruitment $f$ and seniority $\gamma$). Spatially explicit analyses may assume home-range centres are fixed or allow dispersal between primary sessions according to various probability kernels, including bivariate normal (BVN) and bivariate t (BVT).

Details

Package: openCR  
Type: Package  
Version: 2.2.2  
Date: 2022-01-17  
License: GNU General Public License Version 2 or later

Data are observations of marked individuals from a ‘robust’ sampling design (Pollock 1982). Primary sessions may include one or more secondary sessions. Detection histories are assumed to be stored in an object of class ‘capthist’ from the package secr. Grouping of occasions into primary and secondary sessions is coded by the ‘intervals’ attribute (zero for successive secondary sessions).

A few test datasets are provided (microtusCH, FebpossumCH, dipperCH, gonodontisCH, fieldvoleCH) and some from secr are also suitable e.g. ovenCH and 0VpossumCH.

Models are defined using symbolic formula notation. Possible predictors include both pre-defined variables (b, session etc.), corresponding to ‘behaviour’ and other effects, and user-provided covariates.

Models are fitted by numerically maximizing the likelihood. The function openCR.fit creates an object of class openCR. Generic methods (print, AIC, etc.) are provided for each object class.

A link at the bottom of each help page takes you to the help index.

See openCR-vignette.pdf for more.

Author(s)

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

References


See Also

openCR.fit, capthist, ovenCH

Examples

```R
## Not run:
## a CJS model is fitted by default
openCR.fit(ovenCH)

## End(Not run)
```

---

**age.matrix**

### Session-specific Ages

#### Description

A matrix showing the age of each animal at each secondary session (occasion).

#### Usage

```R
age.matrix(capthist, initialage = 0, minimumage = 0, maximumage = 1, collapse = FALSE)
```

#### Arguments

- `capthist`: single-session capthist object
- `initialage`: numeric or character name of covariate with age at first detection (optional)
- `minimumage`: integer minimum age
- `maximumage`: integer maximum age
- `collapse`: logical; if TRUE then values for each individual are collapsed as a string with no spaces
Details

age.matrix is used by openCR.design for the predictors ‘age’ and ‘Age’.
Computations use the intervals attribute of capthist, which may be non-integer.
Ages are inferred for occasions before first detection, back to the minimum age.

Value

Either a numeric matrix with dimensions (number of animals, number of secondary occasions) or if collapse = TRUE a character matrix with one column.

See Also

openCR.design

Examples

age.matrix(join(ovenCH), maximumage = 2, collapse = TRUE)
Arguments

- **object**: openCR object output from the function `openCR.fit`, or `openCRlist`.
- ... other openCR 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.
- **use.rank**: logical; if TRUE the number of parameters is based on the rank of the Hessian matrix.
- **svtol**: minimum singular value (eigenvalue) of Hessian used when counting non-redundant parameters.
- **criterion**: character, criterion to use for model comparison and weights.
- **n**: integer effective sample size.

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

$$AIC_c = -2 \log \left( L(\hat{\theta}) \right) + 2K + \frac{2K(K + 1)}{n - K - 1}$$

where $K$ is the number of “beta” parameters estimated. By default, the effective sample size $n$ is the number of individuals observed at least once (i.e. the number of rows in `capthist`). This differs from the default in MARK which for CJS models is the sum of the sizes of release cohorts (see `m.array`).

Model weights are calculated as

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

Models for which $\Delta \text{AIC} > d\text{max}$ are given a weight of zero and are excluded from the summation.

Model weights may be used to form model-averaged estimates of real or beta parameters with `modelAverage` (see also Buckland et al. 1997, Burnham and Anderson 2002).

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

Value

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

- **model**: character string describing the fitted model.
- **npar**: number of parameters estimated.
- **rank**: rank of Hessian.
- **logLik**: maximized log likelihood.
- **AIC**: Akaike’s Information Criterion.
AIC\textsubscript{openCR} 7

AICc  \hspace{0.5cm} \text{AIC with small-sample adjustment of Hurvich & Tsai (1989)}

dAICc  \hspace{0.5cm} \text{difference between AICc of this model and the one with smallest AIC}

AIC\text{wt}  \hspace{0.5cm} \text{AICc model weight}

\texttt{logLik.openCR} returns an object of class ‘logLik’ that has attribute \texttt{df} (degrees of freedom = number of estimated parameters).

\textbf{Note}

The default criterion is AIC, not AICc as in \texttt{secr} 3.1.

Computed values differ from MARK for various reasons. MARK uses the number of observations, not the number of capture histories when computing AICc. It is also likely that MARK will count parameters differently.

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

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

\textbf{References}


\textbf{See Also}

\texttt{AIC, openCR.fit, print.openCR, LR.test}

\textbf{Examples}

\begin{verbatim}
## Not run:
m1 <- openCR.fit(ovenCH, type = 'JSSAf')
m2 <- openCR.fit(ovenCH, type = 'JSSAf', model = list(p~session))
AIC(m1, m2)

## End(Not run)
\end{verbatim}
classMembership

Class Membership Probability for Mixture Models

Description

Finite mixture models treat class membership as a latent random variable. The probability of an individual’s membership in each class may be inferred retrospectively from the relative likelihoods.

Usage

## S3 method for class 'openCR'
classMembership(object, fullCH = NULL, ...)

Arguments

- object: fitted model of class openCR
- fullCH: captathist object (optional)
- ...: other arguments (not used)

Details

It is assumed that the input model includes finite mixture terms (h2 or h3). As the detection histories are saved in compressed ("squeezed") form in openCR objects the original animal identifiers are lost and the order of animals may change. These may be restored by providing fullCH.

No class can be assigned from a CJS model for animals detected only in the final session.

Value

Matrix with one row per individual and columns for each class and the class number of the most likely class.

Note

In earlier versions openCR.fit always computed class membership and saved it in component ‘posterior’ of the fitted model. classMembership replaces that functionality.

See Also

openCR.fit, squeeze
Examples

```r
## Not run:
jch <- join(ovenCH)
fit <- openCR.fit(ovenCH, model=p~h2)
classMembership(fit, jch)

## End(Not run)
```

---

**cloned.fit**

*Cloning to Evaluate Identifiability*

### Description

The identifiability of parameters may be examined by refitting a model with cloned data (each capture history replicated `nclone` times). For identifiable parameters the estimated variances are proportional to $1/nclone$.

### Usage

```r
cloned.fit(object, nclone = 100, newdata = NULL, linkscale = FALSE)
```

### Arguments

- **object**: previously fitted openCR object
- **nclone**: integer number of times to replicate each capture history
- **newdata**: optional dataframe of values at which to evaluate model
- **linkscale**: logical; if TRUE then comparison uses SE of linear predictors

### Details

The key output is the ratio of SE for estimates from the uncloned and cloned datasets, adjusted for the level of cloning (`nclone`). For identifiable parameters the ratio is expected to be 1.0.

Cloning is not implemented for spatial models.

The comparison may be done either on the untransformed scale (using approximate SE) or on the link scale.
cumMove

Value

Dataframe with columns* –
- estimate: original estimate
- SE.estimate: original SE
- estimate.xxx: cloned estimate (xxx = nclone)
- SE.estimate.xxx: cloned SE
- SE.ratio: SE.estimate / SE.estimate.xxx / sqrt(nclone)

* 'estimate' becomes 'beta' when linkscale = TRUE.

References


See Also

openCR.fit

Examples

```r
## Not run:
fit <- openCR.fit(dipperCH)
cloned.fit(fit)
## End(Not run)
```

cumMove

Probability Distribution After Movement

Description

Compute the compounding effect of a random walk defined by a discrete kernel. The number of steps and the edge algorithm are specified by the user. The final distribution may be summed for points lying within an arbitrary polygon. This is a simple way to compute the expected proportion remaining within a particular region (i.e. not “emigrating”).

Usage

```r
cumMove(X, mask, kernel, edgemethod = c("truncate", "wrap", "none"), nstep = 1, mqarray = NULL, settlecov = NULL)
proportionInPolygon(mask, poly, cov = "pm")
```
cumMove

Arguments

- `x`  initial location(s) (see Details)
- `mask`  habitat mask
- `kernel`  kernel object
- `edgemethod`  character
- `nstep`  non-negative integer
- `mqarray`  integer array of lookup indices
- `settlecov`  character name of covariate of mask
- `poly`  a polygon (see Details)
- `cov`  character name of covariate of mask

Details

The input `X` may be -

- a vector of length 2 for the coordinates of a single point
- a mask with covariate 'pm' representing the initial distribution
- a SpatialPolygons object from `sp`. Animals are assumed initially to be distributed uniformly across mask points that lie within the polygon.

The default edgemethod truncates the kernel at the edge and re-normalizes the cell probabilities so that all destinations lie within the boundary of the mask.

`settlecov` may name a covariate of `mask` that has settlement weights in range 0–1.

For `proportionInPolygon`, the input mask may be the output from `cumMove`. The polygon `poly` may be specified as for `pointsInPolygon` (e.g., SpatialPolygons object or 2-column matrix of coordinates) or as a list with components x and y. A list of polygon specifications is also accepted.

`mqarray` is computed automatically if not provided. Precomputing the array can save time but is undocumented.

Value

For `cumMove` - a mask object with initial probability distribution in covariate 'pm0' and final distribution in covariate 'pm'.

For `proportionInPolygon` - vector of the summed weights (probabilities) for cells centred in the polygon(s) as a proportion of all non-missing weights.

See Also

`make.kernel`, `pointsInPolygon`
Examples

```r
sp <- 10
msk <- make.mask(nx = 51, ny = 51, type = 'rect', spacing = sp,
                  buffer = 0)
k <- make.kernel('BVN', 20, spacing = sp, move.a = 50, clip = TRUE,
                  sparse = TRUE)

# initial distribution a central point
X <- apply(msk, 2, mean)
par(mfrow = c(1,4), mar = c(1,1,2,1))
for (step in 0:2) {
  X <- cumMove(X, msk, k, nstep = min(step,1))
  plot(X, cov = 'pm', dots = FALSE, legend = FALSE, breaks =
       seq(0, 0.006, 0.0001))
  mtext(side = 3, line = 0, paste('Step', step), cex = 0.9)
  contour(
    x = unique(X$x),
    y = unique(X$y),
    z = matrix(covariates(X)$pm, nrow = length(unique(X$x))),
    levels = c(0.0002),
    drawlabels = FALSE, 
    add = TRUE)
}

## Not run:
# initial distribution across a polygon
X0 <- matrix(c(200,200,300,300,200,200,300,300,200,200), ncol = 2)
X <- X0
par(mfrow = c(1,4), mar = c(1,1,2,1))
for (step in 0:3) {
  X <- cumMove(X, msk, k, nstep = min(step,1))
  plot(X, cov = 'pm', dots = FALSE, legend = FALSE, breaks =
       seq(0, 0.006, 0.0001))
  mtext(side = 3, line = 0, paste('Step', step), cex = 0.9)
  contour(
    x = unique(X$x),
    y = unique(X$y),
    z = matrix(covariates(X)$pm, nrow = length(unique(X$x))),
    levels = c(0.0002),
    drawlabels = FALSE, 
    add = TRUE)
}

polygon(X0)
proportionInPolygon(X, X0)
```

## End(Not run)

---

**Derived Parameters From openCR Models**
Description

For .CL openCR models, compute the superpopulation size or density. For all openCR models, compute the time-specific population size or density from the estimated superpopulation size and the turnover parameters.

Usage

```r
## S3 method for class 'openCR'
derived(object, newdata = NULL, all.levels = FALSE, Dscale = 1,
    HTbysession = FALSE, ...)
## S3 method for class 'openCRlist'
derived(object, newdata = NULL, all.levels = FALSE, Dscale = 1,
    HTbysession = FALSE, ...)
opencR.esa(object, bysession = FALSE, stratum = 1)
opencR.pdot(object, bysession = FALSE, stratum = 1)
```

Arguments

- `object` fitted openCR model
- `newdata` optional dataframe of values at which to evaluate model
- `all.levels` logical; passed to `makeNewData` if newdata not specified
- `Dscale` numeric to scale density
- `HTbysession` logical; Horvitz-Thompson estimates by session (see Details)
- `...` other arguments (not used)
- `bysession` logical; if TRUE then esa or pdot is computed separately for each session
- `stratum` integer

Details

Derived estimates of density and superD are multiplied by Dscale. Use Dscale = 1e4 for animals per 100 sq. km. openCR.esa and openCR.pdot are used internally by derived.openCR.

If HTbysession then a separate H-T estimate is derived for each primary session; otherwise a H-T estimate of the superpopulation is used in combination with turnover parameters (phi, beta) to obtain session-specific estimates. Results are often identical.

The output is an object with its own print method (see `print.derivedopenCR`).

The code does not yet allow user-specified newdata.

Value

derived returns an object of class `c("derivedopenCR","list"), list with these components:

- `totalobserved` number of different individuals detected
- `parameters` character vector; names of parameters in model (excludes derived parameters)
- `superN` superpopulation size (non-spatial models only)
superD superpopulation density (spatial models only)
estimates data frame of counts and estimates
Dscale numeric multiplier for printing densities

If newdata has multiple levels then the value is a list of such objects, one for each level.
openCR.pdot returns a vector of experiment-wide detection probabilities under the fitted model
(one for each detected animal).
openCR.esa returns a vector of effective sampling areas under the fitted model (one for each de-
tected animal). If `bysession = TRUE` the result is a list with one component per session.

Note

Prior to 1.4.5, openCR.esa did not expand the result for squeezed capture histories (freq>1) and did
not return a list when bysession = TRUE.

See Also

openCR.fit, print.derivedopenCR

Examples

```r
## Not run:
# override default method to get true ML for L1
L1CL <- openCR.fit(ovenCH, type = 'JSSAlCL', method = 'Nelder-Mead')
predict(L1CL)
derived(L1CL)

## compare to above
L1 <- openCR.fit(ovenCH, type = 'JSSA1', method = 'Nelder-Mead')
predict(L1)
derived(L1)

## End(Not run)
```

dipperCH *Dippers*

Description

Lebreton et al. (1992) demonstrated Cormack-Jolly-Seber methods with a dataset on European
Dipper (*Cinclus cinclus*) collected by Marzolin (1988) and the data have been much used since then. Dippers were captured annually over 1981–1987. We use the version included in the RMark
package (Laake 2013).
Usage

dipperCH

Format

The format is a single-session `secr` capthist object. As these are non-spatial data, the `traps` attribute is NULL.

Details

Dippers were sampled in 1981–1987.

Source

MARK example dataset ‘ed.inp’. Also RMark (Laake 2013). See Examples.

References


See Also

read.inp

Examples

m.array(dipperCH)

```r
## Not run:
# From file 'ed.inp' in MARK input format
datadir <- system.file('extdata', package = 'openCR')
dipperCH <- read.inp(paste0(datadir, '/ed.inp'), grouplabel='sex',
grouplevels = c('Male', 'Female'))
intervals(dipperCH) <- rep(1,6)
sessionlabels(dipperCH) <- 1981:1987  # labels only

# or extracted from the RMark package with this code
if (require(RMark)) {
  if (all (nchar(Sys.which(c('mark.exe','mark64.exe', 'mark32.exe'))) < 2))
```
stop("MARK executable not found; set e.g. MarkPath <- c:/Mark/")
data(dipper) # retrieve dataframe of dipper capture histories
dipperCH2 <- unRMarkInput(dipper) # convert to secr capthist object
intervals(dipperCH2) <- rep(1,6)
sessionlabels(dipperCH2) <- 1981:1987 # labels only
} else message("RMark not found")

# The objects dipperCH and dipperCH2 differ in the order of factor levels for 'sex'
## End(Not run)

---

**expected.d**

**Expected Distance Moved**

### Description

Movement models in openCR differ in their parameterisation so direct comparison can be difficult. The expected distance moved is a convenient statistic common to all models. This function computes the expected distance from various inputs, including fitted models.

### Usage

```r
expected.d(movementmodel, move.a, move.b, truncate = Inf, mask = NULL, min.d = 1e-4, ...)
```

### Arguments

- **movementmodel**: character or function or kernel or openCR object
- **move.a**: numeric parameter of kernel
- **move.b**: numeric parameter of kernel
- **truncate**: radius of truncation
- **mask**: habitat mask object
- **min.d**: numeric lower bound of integration (see Details)
- **...**: other arguments passed to `make.kernel` if input is a fitted model

### Details

The input `movementmodel` may be

- fitted openCR model
- user kernel function \( g(r) \)
- kernel object
- character name of kernel model see Movement models
If truncate (R) is finite or movementmodel is a function then the expected value is computed by numerical integration \( E(d) = \int_0^R r \cdot f(r) \, dr \). In the event that \( f(0) \) is not finite, \( \min_d \) is used as the lower bound.

mask is used only for ‘uncorrelated’ and ‘uncorrelatedzi’ movement. For these models the expected movement is merely the average distance between points on the mask, weighted by \( (1-zi) \) if zero-inflated (uncorrelatedzi).

The ...argument is useful for (i) selecting a session from a fitted model, or (ii) specifying the upper or lower confidence limits from a single-parameter fitted model via the ‘stat’ argument of make.kernel.

Value

A numeric value (zero for ‘static’ model, NA if model unrecognised).

See Also

Movement models, make.kernel, pkernel, qkernel

Examples

```r
expected.d('BVT', move.a = 20, move.b = 1)
expected.d('BVT', move.a = 20, move.b = 1, truncate = 300)
k <- make.kernel(movementmodel = 'BVT', spacing = 10, move.a = 20, move.b = 1)
expected.d(k)
```

---

**Field vole**

**Kielder Field Voles**

### Description

Captures of *Microtus agrestis* on a large grid in a clearcut within Kielder Forest, northern England, June–August 2000 (Ergon and Gardner 2014). Robust-design data from four primary sessions of 3–5 secondary sessions each.

### Usage

`fieldvoleCH`

### Format

The format is a multi-session secr capthist object. Attribute ‘ampm’ codes for type of secondary session (am, pm).
Details

Ergon and Lambin (2013) provided a robust design dataset from a trapping study on field voles *Microtus agrestis* in a clearcut within Kielder Forest, northern England – see also Ergon et al. (2011), Ergon and Gardner (2014) and Reich and Gardner (2014). The study aimed to describe sex differences in space-use, survival and dispersal among adult voles. Data were from one trapping grid in summer 2000.

Trapping was on a rectangular grid of 192 multi-catch (Ugglan Special) traps at 7-metre spacing. Traps were baited with whole barley grains and carrots; voles were marked with individually numbered ear tags.

Four trapping sessions were conducted at intervals of 21 to 23 days between 10 June and 15 August. Traps were checked at about 12 hour intervals (6 am and 6 pm).

The attribute ‘ampm’ is a data.frame with a vector of codes, one per secondary session, to separate am and pm trap checks (1 = evening, 2 = morning). The four primary sessions had respectively 3, 5, 4 and 5 trap checks.

Ergon and Gardner (2014) restricted their analysis to adult voles (118 females and 40 males). Histories of five voles (ma193, ma239, ma371, ma143, ma348) were censored part way through the study because they died in traps (T. Ergon pers. comm.).

Source

Data were retrieved from DRYAD (Ergon and Lambin (2013) for openCR. Code for translating the DRYAD ASCII file into a capthist object is given in Examples.

References


Examples

```r
summary(fieldvoleCH, terse = TRUE)
m.array(fieldvoleCH)
JS.counts(fieldvoleCH)
attr(fieldvoleCH, 'ampm')
```
## Not run:

```r
maleCH <- subset(fieldvoleCH, function(x) covariates(x) == 'M')
fit <- openCR.fit(maleCH)
predict(fit)
```

```r
# Read data object from DRYAD ASCII file

datadir <- system.file('extdata', package = 'openCR')
EG <- dget(paste0(datadir,'/ergonandgardner2013.rdat'))
```

```r
# construct capthist object
onesession <- function (sess) {
  mat <- EG$H[, , sess]
  id <- as.numeric(row(mat))
  occ <- as.numeric(col(mat))
  occ[mat<0] <- -occ[mat<0]
  trap <- abs(as.numeric(mat))
  matrow <- rownames(mat)
  df <- data.frame(session = rep(sess, length(id)),
                  ID = matrow[id],
                  occ = occ,
                  trapID = trap,
                  sex = c('F', 'M')[EG$gr],
                  row.names = 1:length(id))
  # retain captures (trap>0)
  df[df$trapID>0, , drop = FALSE]
}
```

```r
tr <- read.traps(data = data.frame(EG$X), detector = "multi")
```

```r
# recode matrix as mixture of zeros and trap numbers
EG$H <- EG$H - 1
```

```r
# code censored animals with negative trap number
# two ways to recognise censoring
censoredprimary <- which(EG$K < 4)
censoredsecondary <- which(apply(EG$J, 1, function(x) any(x-c(3, 5, 4, 5) < 0)))
censored <- unique(c(censoredprimary, censoredsecondary))
rownames(EG$H)[censored]
```

```r
# [1] "ma193" "ma239" "ma371" "ma143" "ma348"
censorocc <- apply(EG$H[censored, , ], 1, function(x) which.max(cumsum(x)))
censor3 <- ((censorocc-1) %% 5) + 1 # session
censor2 <- censorocc - (censor3-1) * 5 # occasion within session
censori <- cbind(censored, censor2, censor3)
EG$H[censori] <- -EG$H[censori]
```

```r
lch <- lapply(1:4, onesession)
ch <- make.capthist(do.call(rbind,lch), tr=tr, covnames='sex')
```

```r
# apply intervals in months
intervals(ch) <- EG$dt
```

```r
fieldvoleCH <- ch
```
# extract time covariate - each secondary session was either am (2) or pm (1)
# EG$tod
# 1 2 3 4 5
# 1 2 1 2 NA NA
# 2 2 1 2 1 1
# 3 2 1 2 1 NA
# 4 2 1 2 1 2
# Note consecutive pm trap checks in session 2
ampm <- split(EG$tod, 1:4)
ampm <- lapply(ampm, na.omit)
attr(fieldvoleCH, 'ampm') <- data.frame(ampm = unlist(ampm))

## End(Not run)

gonodontisCH  Gonodontis Moths

Description
Non-spatial open-population capture–recapture data of Bishop et al. (1978) for nonmelanic male
Gonodontis bidentata at Cressington Park, northwest England.

Usage
gonodontisCH

Format
The format is a single-session secr capthist object. As these are non-spatial data, the traps attribute is NULL.

Details
The data are from a study of the relative fitness of melanic and nonmelanic morphs of the moth
Gonodontis bidentata at several sites in England (Bishop et al. 1978). Crosbie (1979; see also
Crosbie and Manly 1985) selected a subset of the Bishop et al. data (nonmelanic males from
Cressington Park) to demonstrate innovations in Jolly-Seber modelling, and the same data were
used by Link and Barker (2005) and Schofield and Barker (2008). The present data are those used
by Crosbie (1979) and Link and Barker (2005).
Male moths were attracted to traps which consisted of a cage containing phermone-producing fe-
male surrounded by an enclosure which the males could enter but not leave. New virgin females
were usually added every 1 to 4 days. Moths were marked at each capture with a date-specific mark
in enamel paint or felt-tip pen on the undersurface of the wing. Thus, although moths at Cressington
Park were not marked individually, each moth was a flying bearer of its own capture history.
The data comprise 689 individual capture histories for moths captured at 8 traps operated over 17 days (24 May–10 June 1970). The traps were in a square that appears have been about 40 m on a side. The location of captures is not included in the published data. All captured moths appear to have been marked and released (i.e. there were no removals recorded). All captures on Day 17 were recaptures; it is possible that unmarked moths were not recorded on that day.

Both Table 1 and Appendix 1 (microfiche) of Bishop et al. (1978) refer to 690 capture histories of nonmelanics at Cressington Park. In the present data there are only 689, and there are other minor discrepancies. Also, Crosbie and Manly (1985: Table 1) refer to 82 unique capture histories (“distinct cmr patterns”) when there are only 81 in the present dataset (note that two moths share 00000000000000011).

Source

Richard Barker provided an electronic copy of the data used by Link and Barker (2005), copied from Crosbie (1979).

References


Examples

```r
summary(gonodontisCH)
m.array(gonodontisCH)

## Not run:
# compare default (CJS) estimates from openCR, MARK
fit <- openCR.fit(gonodontisCH)
predict(fit)

if (require(RMark)) {
  MarkPath <- 'c:/Mark/' # customize as needed
  if (!all (nchar(Sys.which(c('mark.exe','mark64.exe', 'mark32.exe')))) < 2)) {
    mothdf <- RMarkInput(gonodontisCH)
    mark(mothdf)
    cleanup(ask = FALSE)
} else message("mark.exe not found")
```
### Internal Functions

#### Description

Functions called by openCR.fit when details$R == TRUE, and some others

#### Usage

- `prwi (type, n, x, jj, cumss, nmix, w, fi, li, openval, PIA, PIAJ, intervals, CJSp1)`
- `prwisecr (type, n, x, nc, jj, kk, mm, nmix, cumss, w, fi, li, gk, openval, PIA, PIAJ, binomN, Tsk, intervals, h, hindex, CJSp1, moveargsi, movementcode, sparsekernel, edgencode, usermodel, kernel = NULL, mqarray = NULL, cellsize = NULL, r0)`
- `PCH1 (type, x, nc, cumss, nmix, openval0, PIA0, PIAJ, intervals)`
- `PCH1secr (type, individual, x, nc, jj, cumss, kk, mm, openval0, PIA0, PIAJ, gk0, binomN, Tsk, intervals, moveargsi, movementcode, sparsekernel, edgencode, usermodel, kernel, mqarray, cellsize, r0)`
- `pradelloglik (type, w, openval, PIAJ, intervals)`
- `cyclic.fit (... maxcycle = 10, tol = 1e-5, trace = FALSE)`

#### Arguments

- `type` character
- `n` integer index of capture history
- `x` integer index of latent class
- `jj` integer number of primary sessions
- `cumss` integer vector cumulative number of secondary sessions at start of each primary session
- `nmix` integer number of latent classes
- `w` array of capture histories
- `fi` integer first primary session
li
openval
PIA
PIAJ
intervals
h
hindex
CJSp1
moveargsi
movementcode
sparsekernel
decode
usermodel
kernel
mqarray
cellsize
r0
gk
Tsk
openval0
PIA0
individual
gk0
nc
kk
mm
binomN
... maxcycle
tol
trace

Details

cyclic.fit implements cyclic fixing more or less as described by Schwarz and Arnason (1996) and used by Pledger et al. (2010). The intention is to speed up maximization when there are many (beta) parameters. However, fitting is slower than with a single call to openCR.fit, and the function is here only as a curiosity (it is not exported in 1.2.0).
Value

cyclic.fit returns a fitted model object of class ‘openCR’.
Other functions return numeric components of the log likelihood.

References


See Also

*openCR.fit*

Examples

```r
## Not run:
openCR::cyclic.fit(capthist = dipperCH, model = list(p~t, phi~t), tol = 1e-5, trace = TRUE)
## End(Not run)
```

---

**JS.counts**

*Summarise Non-spatial Open-population Data*

Description

Simple conventional summaries of data held in secr ‘capthist’ objects.

Usage

```r
JS.counts(object, primary.only = TRUE, stratified = FALSE)
m.array(object, primary.only = TRUE, never.recaptured = TRUE, 
       last.session = TRUE, stratified = FALSE)
bd.array(beta, phi)
```
Arguments

- **object**: secr capthist object or similar
- **primary.only**: logical; if TRUE then counts are tabulated for primary sessions
- **stratified**: logical; if TRUE then sessions of multisession object summarised separately
- **never.recaptured**: logical; if TRUE then a column is added for animals never recaptured
- **last.session**: logical; if TRUE releases are reported for the last session
- **beta**: numeric vector of entry probabilities, one per primary session
- **phi**: numeric vector of survival probabilities, one per primary session

Details

The input is a capthist object representing a multi-session capture–recapture study. This may be (i) a single-session capthist in which occasions are understood to represent primary sessions, or (ii) a multi-session capthist object that is automatically converted to a single session object with `join` (any secondary sessions (occasions) are first collapsed with `reduce(object, by = 'all')`), or (iii) a multi-session capthist object in which sessions are interpreted as strata.

The argument **primary.only** applies for single-session input with a robust-design structure defined by the `intervals`. **last.session** results in a final row with no recaptures.

If the covariates attribute of **object** includes a column named ‘freq’ then this is used to expand the capture histories.

Conventional Jolly–Seber estimates may be computed with `JS.direct`.

`bd.array` computes the probability of each possible combination of birth and death times (strictly, the primary session at which an animal was first and last available for detection), given the parameter vectors beta and phi. These cell probabilities are integral to JSSA models.

* this may fail with nonspatial data.

Value

For `JS.counts`, a data.frame where rows correspond to sessions and columns hold counts as follows:

- **n**: number of individuals detected
- **R**: number of individuals released
- **m**: number of previously marked individuals
- **r**: number of released individuals detected in later sessions
- **z**: number known to be alive (detected before and after) but not detected in current session

For `m.array`, a table object with rows corresponding to release cohorts and columns corresponding to first–recapture sessions. The size of the release cohort is shown in the first column. Cells in the lower triangle have value NA and print as blank by default.

See Also

`join, JS.direct`
Examples

JS.counts(ovenCH)
m.array(ovenCH)

## Not run:

## probabilities of b,d pairs
fit <- openCR.fit(ovenCH, type = 'JSSAbCL')
beta <- predict(fit)$b$estimate
phi <- predict(fit)$phi$estimate
bd.array(beta, phi)

## End(Not run)

---

**JS.direct**

*Jolly–Seber Estimates*

**Description**

Non-spatial open-population estimates using the conventional closed-form Jolly–Seber estimators (Pollock et al. 1990).

**Usage**

JS.direct(object)

**Arguments**

- **object** secr capthist object or similar

**Details**

Estimates are the session-specific Jolly-Seber estimates with no constraints.
The reported SE of births (B) differ slightly from those in Pollock et al. (1990), and may be in error.

**Value**

A dataframe in which the first 5 columns are summary statistics (counts from JS.counts) and the remaining columns are estimates:

- **p** capture probability
- **N** population size
- **phi** probability of survival to next sample time
B  number of recruits at next sample time

Standard errors are in fields prefixed ‘se’; for N and B these include only sampling variation and
omit population stochasticity. The covariance of successive phi-hat is in the field ‘covphi’.

References


See Also

  * JS.counts

Examples

```
# cf Pollock et al. (1990) Table 4.8
JS.direct(microtusCH)
```

---

**LLsurface**  
*Plot Likelihood Surface*

**Description**

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

This is a method for the generic function LLsurface defined in *secr*.

**Usage**

```r
## S3 method for class 'openCR'
LLsurface(object, betapar = c("phi", "sigma"), xval = NULL, yval = NULL,
  centre = NULL, realscale = TRUE, plot = TRUE, plotfitted = TRUE, ncores = NULL, ...)
```

**Arguments**

- `object`  
  openCR object output from openCR.fit

- `betapar`  
  character vector giving the names of two beta parameters

- `xval`  
  vector of numeric values for x-dimension of grid

- `yval`  
  vector of numeric values for y-dimension of grid

- `centre`  
  vector of central values for all beta parameters

- `realscale`  
  logical. If TRUE input and output of x and y is on the untransformed (inverse-
  link) scale.
plot  logical. If TRUE a contour plot is produced
plotfitted  logical. If TRUE the MLE from object is shown on the plot (+)
ncores  integer number of cores available for parallel processing
...  other arguments passed to contour

Details

centre is set by default to the fitted values of the beta parameters in object. This has the effect of holding parameters other than those in betapar at their fitted values.

If xval or yval is not provided then 11 values are set at equal spacing between 0.8 and 1.2 times the values in centre (on the ‘real’ scale if realscale = TRUE and on the ‘beta’ scale otherwise).

Contour plots may be customized by passing graphical parameters through the ...argument.

The value of ncores is passed to openCR.fit.

Value

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

Note

LLsurface.openCR 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.

Handling of multiple threads was changed in version 1.5.0 to align with LLsurface.secr.

The contours are approximate because they rely on interpolation.

See Also

LLsurface.secr

Examples

# not yet
make.kernel

Discrete Movement Kernel

Description

Functions to create, plot and summarise a discrete representation of a movement kernel.

Usage

```r
make.kernel(movementmodel = c("BVN", "BVE", "BVC", "BVT", "RDE", "RDG", "RDL", "UNI"),
            kernelradius = 10, spacing, move.a, move.b,
            sparsekernel = FALSE, clip = FALSE, normalize = TRUE,
            stat = c('estimate', 'lcl', 'ucl'), session = 1, r0 = 1/sqrt(pi), ...)
```

```r
## S3 method for class 'kernel'
plot(x, type = "kernel", contour = FALSE, levels = NULL, text = FALSE,
     title = NULL, add = FALSE, xscale = 1, ...)  
## S3 method for class 'kernel'
summary(object, ...)
```

Arguments

- `movementmodel`: character or function or openCR object
- `kernelradius`: integer radius of kernel in grid cells
- `spacing`: numeric spacing between cell centres
- `move.a`: numeric parameter of kernel
- `move.b`: numeric parameter of kernel
- `sparsekernel`: logical; if TRUE then only cardinal and intercardinal axes are included
- `clip`: logical; if TRUE then corner cells are removed
- `normalize`: logical; if TRUE then cell values are divided by their sum
- `stat`: character; predicted statistic to use for move.a (openCR object only)
- `session`: integer; session for move.a, move.b if input is fitted model
- `r0`: numeric; effective radius of zero cell for movement models
- `x`: kernel object from `make.kernel`
- `type`: character; plot style (see Details)
- `contour`: logical; if TRUE then contour lines are overlaid on any plot
- `levels`: numeric vector of contour levels
- `text`: logical; if TRUE then cell probabilities are overprinted, rounded to 3 d.p.
- `title`: character; if NULL a title is constructed automatically
add logical; if TRUE a line is added to an existing plot (types "gr", "fr", "Fr")
xscale numeric multiplier for distance axis (0.001 for distances in km)
... other arguments passed to predict.openCR (make.kernel) or plot.mask (plot type "kernel")
or lines ( plot types "gr", "fr", "Fr") (not used by summary method)
object kernel object from make.kernel

Details

A kernel object is a type of mask with cell probabilities stored in the covariate ‘kernelp’. All kernels are truncated at kernelradius x spacing.

The movement model may also be a function or a previously fitted openCR model that includes movement. If a fitted openCR object, parameter values and kernel attributes are derived from that object and other arguments are ignored.

The parameter ‘move.a’ is a scale parameter in metres, except for the UNIzi and INDzi models for which it is the zero-inflation parameter (‘move.b’ is the zero-inflation parameter for BVNzi, BVEzi and RDEzi).

‘Sparse’ kernels include only those grid cells that lie on 4 axes (N-S, E-W, NW-SE, NE-SW); cell probabilities are adjusted to maintain nearly the same distance distribution as the non-sparse equivalents.

Movement models are listed in Movement models and further described in the vignettes openCR-vignette.pdf.

Plot type may be one or more of –

- ‘kernel’ coloured 2-D depiction
- ‘gr’ cross-section through the origin of \( g(r) \) (the 2-D kernel)
- ‘fr’ continuous probability density \( f(r) \)
- ‘Fr’ cumulative probability distribution \( F(r) \)

Type “kernel” by default includes an informative title with font size from the graphical parameter ‘cex.main’. Set title = "" to suppress the title.

Useful properties of theoretical (not discretized) kernels may be recovered with matchscale, pkernel,
dkernel and qkernel.

The obscure argument \( r0 \) controls the value assigned to the central cell of a discretized kernel. For positive \( r0 \) the value is \( F(r0*cellsize) \), where \( F \) is the cumulative probability distribution of distance moved. Otherwise the cell is assigned the value \( g(0)*cellarea \), where \( g() \) is the 2-D kernel probability density (this fails where \( g(0) \) is undefined or infinite).

Value

make.kernel returns an object of class c('kernel','mask','data.frame').

The kernel object has attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
movementmodel  saved input
K2 saved kernelradius
move.a  saved input
move.b  saved input
distribution empirical cumulative distribution function

The empirical cumulative distribution is a dataframe with columns for the sorted cell radii ‘r’ and the associated cumulative probability ‘cumprob’ (one row per cell).

summary.kernel returns an object with these components, displayed with the corresponding print method.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k2</td>
<td>kernel radius in mask cells</td>
</tr>
<tr>
<td>spacing</td>
<td>cell width</td>
</tr>
<tr>
<td>ncells</td>
<td>number of cells in kernel</td>
</tr>
<tr>
<td>movementmodel</td>
<td>movement model code</td>
</tr>
<tr>
<td>move.a</td>
<td>first (scale) parameter</td>
</tr>
<tr>
<td>move.b</td>
<td>second (shape) parameter</td>
</tr>
<tr>
<td>mu</td>
<td>mean of logs (RDL only; from move.a)</td>
</tr>
<tr>
<td>s</td>
<td>SD of logs (RDL only; from move.b)</td>
</tr>
<tr>
<td>expectedmove</td>
<td>mean movement (untruncated)</td>
</tr>
<tr>
<td>expectedmovetr</td>
<td>mean movement (truncated at kernel radius)</td>
</tr>
<tr>
<td>expectedmoveemp</td>
<td>mean computed directly from kernel cell values as sum(r.p)</td>
</tr>
<tr>
<td>ptruncated</td>
<td>proportion of theoretical distribution truncated at radius</td>
</tr>
<tr>
<td>expectedq50</td>
<td>theoretical (untruncated) median</td>
</tr>
<tr>
<td>expectedq90</td>
<td>theoretical (untruncated) 90th percentile</td>
</tr>
<tr>
<td>expectedq50tr</td>
<td>theoretical truncated median</td>
</tr>
<tr>
<td>expectedq90tr</td>
<td>theoretical truncated 90th percentile</td>
</tr>
</tbody>
</table>

The empirical mean in expectedmoveemp is usually the most pertinent property of a fitted kernel.

Note

The plot method for kernels supercedes the function plotKernel that has been removed.

References


See Also

Movement models, mask, matchscale, dkernel, pkernel, qkernel

Examples

```r
k <- make.kernel(movementmodel = 'BVT', spacing = 10, move.a = 20, move.b = 1)
summary(k)

# read a previously fitted movement model packaged with 'openCR'
fit <- readRDS(system.file("exampledata", "spmOV.RDS", package = "openCR"))
k <- make.kernel(fit)
plot(k)
if (interactive()) {
  spotHeight(k, dec = 3) # click on points; Esc to exit
}
```

make.table

Tabulate Estimates From Multiple Models

Description

Session-specific estimates of real parameters (p, phi, etc.) are arranged in a rectangular table.

Usage

```r
make.table(fits, parm = "phi", fields = "estimate", strata = 1, collapse = FALSE, ...)
```

Arguments

- `fits`: openCRlist object
- `parm`: character name of real parameter estimate to tabulate
- `fields`: character column from predict (estimate, SE.estimate, lcl, ucl)
- `strata`: integer; indices of strata to report
- `collapse`: logical; if TRUE stratum-specific results are collapsed to single table
- `...`: arguments passed to `predict.openCRlist`

Details

The input will usually be from `par.openCR.fit`.

Value

A table object.
makeNewData

Create Default Design Data

Description

Internal function used to generate a dataframe containing design data for the base levels of all predictors in an openCR object.

Usage

## S3 method for class 'openCR'
makeNewData(object, all.levels = FALSE, ...)

Arguments

object   fitted openCR model object
all.levels   logical; if TRUE then all covariate factor levels appear in the output
...   other arguments (not used)

Details

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

Examples

## Not run:

arglist <- list(
    constant = list(capthist = ovenCHp, model = phi~1),
    session.specific = list(capthist = ovenCHp, model = phi~session)
)
fits <- par.openCR.fit(arglist, trace = FALSE)
print(make.table(fits), na = ".")

## End(Not run)
Value

A dataframe with one row for each session, and columns for the predictors used by `object$model`.

See Also

`openCR.fit`

Examples

```r
## Not run:

## null example (no covariates)
ovenCJS <- openCR.fit(ovenCH)
makeNewData(ovenCJS)

## End(Not run)
```

matchscale

Match Kernel

Description

Finds scale parameter (move.a) of a movement model that corresponds to desired quantile, or expected distance moved.

Usage

```r
matchscale(movementmodel, q = 40, expected = NULL, p = 0.5, lower = 1e-05, upper = 1e+05, move.b = 1, truncate = Inf)
```

Arguments

- `movementmodel`: character (see Movement models and openCR-vignettes.pdf)
- `q`: desired quantile (distance moved)
- `expected`: numeric expected distance moved
- `p`: cumulative probability
- `move.b`: shape parameter of movement kernel
- `lower`: lower bound interval to search
- `upper`: upper bound interval to search
- `truncate`: numeric q value at which distribution truncated
**Details**

The default behaviour is to find the movement parameter for the given combination of q and p.

The alternative, when a value is provided for ‘expected’, is to find the movement parameter corresponding to the given expected distance.

The truncate argument must be specified for movementmodel ‘UNIzi’. For movementmodel ‘UNI’ there is no parameter and the radius of truncation is varied to achieve the requested quantile q corresponding to cumulative probability p, or the desired expected distance.

**Value**

Numeric value for move.a (scale parameter or zero-inflation in the case of ‘UNIzi’) or truncation radius (‘UNI’).

**See Also**

Movement models, pkernel, make.kernel, expected.d

**Examples**

```r
matchscale(‘BVN’, 40, 0.5)
matchscale(‘BVT’, 40, 0.5, move.b = 1)
matchscale(‘BVT’, 40, 0.5, move.b = 5)
matchscale(‘BVT’, move.b = 5, expected = 10)
```

---

**Description**

Captures of *Microtus pennsylvanicus* at Patuxent Wildlife Research Center, Laurel, Maryland, June–December 1981. Collapsed (primary session only) data for adult males and adult females, and full robust-design data for adult males. Nichols et al. (1984) described the field methods and analysed a superset of the present data.

**Usage**

```r
microtusCH
microtusFCH
microtusMCH
microtusFMCH
microtusRDCH
```
Format

The format is a single-session secr capthist object. As these are non-spatial data, the traps attribute is NULL.

Details

Voles were caught in live traps on a 10 x 10 grid with traps 7.6 m apart. Traps were baited with corn. Traps were set in the evening, checked the following morning, and locked open during the day. Voles were ear-tagged with individually numbered fingerling tags. The locations of captures were not included in the published data.

Data collection followed Pollock’s robust design with five consecutive days of trapping each month for six months (27 June 1981–8 December 1981). The data are for “adult” animals only, defined as those weighing at least 22g. Low capture numbers on the last two days of the second primary session (occasions 9 and 10) are due to a raccoon interfering with traps (Nichols et al. 1984). Six adult female voles and ten adult male voles were not released; their final captures are coded as -1 in the respective capthist objects.

microtusRDCH is the full robust-design dataset for adult males ((Williams et al. 2002 Table 19.1). microtusFCH and microtusMCH are the collapsed datasets (binary at the level of primary session) for adult females and adult males from Williams et al. (2002 Table 17.5); microtusFMCH combines them and includes the covariate ‘sex’.

microtusCH is a combined-sex version of the data with different lineage (see below).

The ‘intervals’ attribute was assigned for microtusRDCH to distinguish primary sesssions (interval 1 between primary sessions; interval 0 for consecutive secondary sessions within a primary session). True intervals (start of one primary session to start of next) were 35, 28, 35, 28 and 34 days. See Examples to add these manually.

Williams, Nichols and Conroy (2002) presented several analyses of these data.

Program JOLLY (Hines 1988, Pollock et al. 1990) included a combined-sex version of the primary-session data that was used by Pollock et al. (1985) and Pollock et al. (1990)*. The numbers of voles released each month in the JOLLY dataset JLYEXMPL differ by 0–3 from the sum of the male and female data from Williams et al. (2002) (see Examples). Some discrepancies may have been due to voles for which sex was not recorded. The JOLLY version matches Table 1 of Nichols et al. (1984). The JOLLY version is distributed here as the object microtusCH.

Differing selections of data from the Patuxent study were analysed by Nichols et al. (1992) and Bonner and Schwarz (2006).

* There is a typographic error in Table 4.7 of Pollock et al. (1990): ri for the first period should be 89.

Source

<table>
<thead>
<tr>
<th>Object</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>microtusCH</td>
<td>Text file JLYEXMPL distributed with Program JOLLY (Hines 1988; see also Examples)</td>
</tr>
<tr>
<td>microtusFCH</td>
<td>Table 17.5 in Williams, Nichols and Conroy (2002)</td>
</tr>
<tr>
<td>microtusMCH</td>
<td>Table 17.5 in Williams, Nichols and Conroy (2002)</td>
</tr>
<tr>
<td>microtusFMCH</td>
<td>Table 17.5 in Williams, Nichols and Conroy (2002)</td>
</tr>
<tr>
<td>microtusRDCH</td>
<td>Table 19.1 in Williams, Nichols and Conroy (2002) provided as text file by Jim Hines</td>
</tr>
</tbody>
</table>
References


Examples

```r
# cf Williams, Nichols and Conroy Table 17.6
m.array(microtusFCH)
m.array(microtusMCH)

## Not run:
# cf Williams, Nichols and Conroy Fig. 17.2
fitfm <- openCR.fit(microtusFMCH, model = list(p~1, phi ~ session + sex))
maledat <- expand.grid(sex = factor("M", levels = c("F", "M")), session = factor(1:6))
plot(fitfm, ylim=c(0,1), type = "o")
plot(fitfm, newdata = maledat, add = TRUE, xoffset = 0.1, pch = 16, type = "o")

# adjusting for variable interval
intervals(microtusCH) <- c(35,28,35,28,34) / 30
intervals(microtusRDCH)[intervals(microtusRDCH)>0] <- c(35,28,35,28,34) / 30

# The text file JLYEXMPL distributed with JOLLY is in the extdata folder of the R package
# The microtusCH object may be rebuilt as follows
datadir <- system.file("extdata", package = "openCR")
JLYdf <- read.table(paste0(datadir,"/JLYEXMPL"), skip = 3,
colClasses = c('character','numeric'))
names(JLYdf) <- c('ch', 'freq')
JLYdf$freq[grep('\2', JLYdf$ch)] <- -JLYdf$freq[grep('\2', JLYdf$ch)]
JLYdf$ch <- gsub ('\2', '\1', JLYdf$ch)
microtusCH <- unRMarkInput(JLYdf)
```
# Compare to combined-sex data from Williams et al. Table 17.5
JS.counts(microtusCH) - JS.counts(microtusFMCH)

## End(Not run)

### Data Manipulation

#### Description

Miscellaneous functions

#### Usage

```r
primarysessions(intervals)
secondarysessions(intervals)
```

#### Arguments

- `intervals` numeric vector of intervals for time between secondary sessions a of robust design

#### Details

These functions are used internally.

#### Value

- `primarysessions` – Integer vector with the number of the primary session to which each secondary session belongs.
- `secondarysessions` – Integer vector with secondary sessions numbered sequentially within primary sessions.

#### Examples

```r
int <- intervals(join(ovenCH))
primary <- primarysessions(int)
primary

# number of secondary sessions per primary
table(primary)
```
Averaging of OpenCR Models Using Akaike's Information Criterion

Description

AIC- or AICc-weighted average of estimated ‘real’ or ‘beta’ parameters from multiple fitted openCR models.

The \texttt{modelAverage} generic is imported from \texttt{secr (>= 4.5.0)}.

Usage

\begin{verbatim}
## S3 method for class 'openCR'
modelAverage(object, ..., realnames = NULL, betanames = NULL,
             newdata = NULL, alpha = 0.05, dmax = 10, covar = FALSE, average = c("link",
             "real"), criterion = c("AIC","AICc"), CImethod = c("Wald", "MATA"))

## S3 method for class 'openCRlist'
modelAverage(object, ..., realnames = NULL, betanames = NULL,
             newdata = NULL, alpha = 0.05, dmax = 10, covar = FALSE, average = c("link",
             "real"), criterion = c("AIC","AICc"), CImethod = c("Wald", "MATA"))
\end{verbatim}

Arguments

- \texttt{object} openCR or openCRlist objects
- \texttt{...} other openCR objects (\texttt{modelAverage.openCR()} only)
- realnames character vector of real parameter names
- betanames character vector of beta parameter names
- newdata optional dataframe of values at which to evaluate models
- alpha alpha level for confidence intervals
- dmax numeric, the maximum AIC or AICc difference for inclusion in confidence set
- covar logical, if \texttt{TRUE} then return variance-covariance matrix
- average character string for scale on which to average real parameters
- criterion character, information criterion to use for model weights
- CImethod character, type of confidence interval (see Details)
Details

Models to be compared must have been fitted to the same data and use the same likelihood method (full vs conditional). If \texttt{realnames} = \texttt{NULL} and \texttt{betanames} = \texttt{NULL} then all real parameters will be averaged; in this case all models must use the same real parameters. To average beta parameters, specify \texttt{betanames} (this is ignored if a value is provided for \texttt{realnames}). See \texttt{predict.openCR} for an explanation of the optional argument \texttt{newdata}; \texttt{newdata} is ignored when averaging beta parameters.

Model-averaged estimates for parameter $\theta$ are given by

$$\hat{\theta} = \sum_k w_k \hat{\theta}_k$$

where the subscript $k$ refers to a specific model and the $w_k$ are AIC or AICc weights (see \texttt{AIC.openCR} 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 $d\text{AIC} > d_{\text{max}}$ (or $d\text{AICc} > d_{\text{max}}$) 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).

Two methods are offered for confidence intervals. The default ‘Wald’ uses the above estimate of variance. The alternative ‘MATA’ (model-averaged tail area) avoids estimating a weighted variance and is thought to provide better coverage at little cost in increased interval length (Turek and Fletcher 2012). Turek and Fletcher (2012) also found averaging with AIC weights (here \texttt{criterion} = ‘AIC’) preferable to using AICc weights, even for small samples. \texttt{CImethod} does not affect the reported standard errors.

Value

A list (one component per parameter) 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 \texttt{newdata} or beta parameters are averaged or averaging is requested for only one parameter then the array is collapsed to a matrix. If \texttt{covar} = \texttt{TRUE} then a list is returned with separate components for the estimates and the variance-covariance matrices.

References


Movement models

See Also

AIC.openCR, make.table, openCR.fit, openCRlist

Examples

```r
## Compare two models fitted previously

cjs1 <- openCR.fit(dipperCH, model=p~1)
cjs2 <- openCR.fit(dipperCH, model=p~session)
AIC(cjs1, cjs2)
modelAverage(cjs1, cjs2)

## or

cjs12 <- openCRlist(cjs1, cjs2)
modelAverage(cjs12)
```

---

Movement models

List of Movement Models

## Description

Movement of activity centres between primary sessions is modelled in openCR as a random walk with step length governed by a circular probability kernel. The argument `movementmodel` defines the kernel in several functions. More detail is provided in the vignettes openCR-vignette.pdf.

## Movement models in openCR 2.2

Kernel models:

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BVN</td>
<td>bivariate normal</td>
<td>move.a</td>
</tr>
<tr>
<td>BVE</td>
<td>bivariate Laplace</td>
<td>move.a</td>
</tr>
<tr>
<td>BVC</td>
<td>bivariate Cauchy distribution</td>
<td>move.a</td>
</tr>
<tr>
<td>BVT</td>
<td>bivariate t-distribution (2Dt of Clark et al. 1999)</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>RDE</td>
<td>exponential distribution of distance moved cf Ergon and Gardner (2014)</td>
<td>move.a</td>
</tr>
<tr>
<td>RDG</td>
<td>gamma distribution of distance moved cf Ergon and Gardner (2014)</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>RDL</td>
<td>log-normal distribution of distance moved cf Ergon and Gardner (2014)</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>RDLS*</td>
<td>log-sech distribution of distance moved (Van Houtan et al. 2007)</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>UNI</td>
<td>uniform within kernel radius, zero outside</td>
<td>(none)</td>
</tr>
<tr>
<td>BVNZi</td>
<td>zero-inflated BVN</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>BVEzi</td>
<td>zero-inflated BVE</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>RDEzi</td>
<td>zero-inflated RDE</td>
<td>move.a, move.b</td>
</tr>
<tr>
<td>UNIZi</td>
<td>zero-inflated UNI</td>
<td>move.a</td>
</tr>
</tbody>
</table>
Movement models

* incomplete implementation

Kernel-free models (buffer dependent):

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND</td>
<td>independent relocation within habitat mask (Gardner et al. 2018)</td>
<td>(none)</td>
</tr>
<tr>
<td>INDzi</td>
<td>zero-inflated IND</td>
<td>move.a</td>
</tr>
</tbody>
</table>

Relationships among models

Some models may be derived as special cases of others, for example

<table>
<thead>
<tr>
<th>General</th>
<th>Condition</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>BVT</td>
<td>large move.b (df ∞)</td>
<td>BVN</td>
</tr>
<tr>
<td>BVT</td>
<td>move.b = 0.5 (df 1)</td>
<td>BVC</td>
</tr>
<tr>
<td>RDG</td>
<td>move.b = 1</td>
<td>RDE</td>
</tr>
<tr>
<td>RDG</td>
<td>move.b = 2</td>
<td>BVE</td>
</tr>
<tr>
<td>BVNzi</td>
<td>large move.a</td>
<td>UNIzi</td>
</tr>
</tbody>
</table>

RDL and RDG are almost indistinguishable when move.b > 2.

Deprecated names of movement models

These old names appeared in earlier releases. They still work, but may be removed in future.

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>BVN</td>
</tr>
<tr>
<td>exponential</td>
<td>BVE</td>
</tr>
<tr>
<td>t2D</td>
<td>BVT</td>
</tr>
<tr>
<td>frE</td>
<td>RDE</td>
</tr>
<tr>
<td>frG</td>
<td>RDG</td>
</tr>
<tr>
<td>frL</td>
<td>RDL</td>
</tr>
<tr>
<td>uniform</td>
<td>UNI</td>
</tr>
<tr>
<td>frEzi</td>
<td>RDEzi</td>
</tr>
<tr>
<td>uniformzi</td>
<td>UNIzi</td>
</tr>
</tbody>
</table>

Additional movement models that may be removed without notice

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>annular</td>
<td>non-zero only at centre and edge cells (after clipping at kernelradius)</td>
<td>move.a</td>
</tr>
<tr>
<td>annularR</td>
<td>non-zero only at centre and a ring of cells at radius R</td>
<td>move.a, move.b</td>
</tr>
</tbody>
</table>

“annularR” uses a variable radius (R = move.b x kernelradius x spacing) and weights each cell according to the length of arc it intersects; “annularR” is not currently allowed in openCR. fit. For the ‘annular’ models ‘move.a’ is the proportion at the centre (probability of not moving).
References


See Also

make.kernel, gkernel, dkernel, pkernel, qkernel

moving.fit

Moving Window Functions

Description

Apply a function to successive multi-session windows from a capthist object. The default function is openCR.fit, but any function may be used whose first argument accepts a capthist object.

Usage

```
moving.fit (..., width = 3, centres = NULL, filestem = NULL, trace = FALSE, FUN = openCR.fit)
```

```
extractFocal (ocrlist, ...)
```

Arguments

- `...`: named arguments passed to `openCR.fit` (see Details)
- `width`: integer; moving window width (number of primary sessions)
- `centres`: integer; central sessions of windows to consider
- `filestem`: character or `NULL`; stem used to form filenames for optional intermediate output
- `trace`: logical; if `TRUE` a status message is given at each call of `FUN`
- `FUN`: function to be applied to successive capthist objects
- `ocrlist`: `openCRlist` object returned by `moving.fit` when `FUN = openCR.fit`
moving.fit

Details

moving.fit applies FUN to successive multi-session subsets of the data in the capthist argument. width should be an odd integer. centres may be used to restrict the range of windows considered; the default is to use all complete windows (width%/%2 + 1...).

If a filestem is specified then each result is output to a file that may be loaded with load. This is useful if fitting takes a long time and analyses may be terminated before completion.

eXtractFocal returns the focal-session (central) estimates from a moving.fit with FUN = openCR.fit. The ... argument is passed to predict.openCR; it may be used, for example, to choose a different alpha level for confidence intervals.

extractFocal is untested for complex models (e.g. finite mixtures).

Value

A list in which each component is the output from FUN applied to one subset. The window width is saved as attribute ‘width’.

See Also

openCR.fit

Examples

```r
## number of individuals detected
moving.fit(capthist = OVpossumCH, FUN = nrow)

## Not run:
## if package R2ucare installed
if (requireNamespace("R2ucare"))
  moving.fit(capthist = OVpossumCH, FUN = ucare.cjs, width = 5, tests = "overall_CJS")

## using default FUN = openCR.fit
mf1 <- moving.fit(capthist = OVpossumCH, type = 'JSSAfCL',
  model = list(p~t, phi~t))
lapply(mf1, predict)
exFocal(mf1)

msk <- make.mask(traps(OVpossumCH[[1]]), nx = 32)
mf2 <- moving.fit(capthist = OVpossumCH, mask = msk, type = 'JSSAsecrfCL')
exFocal(mf2)

## End(Not run)
```
Defunct Functions in Package openCR

Description
These functions are no longer available in openCR.

Usage

# Defunct in 2.1.0
openCR.make.newdata()

# Defunct in 2.0.0
plotKernel()

Details
Internal function openCR.make.newdata was replaced with a method for the openCR class of the generic makeNewData.
plotKernel was replaced with a plot method for the kernel class.

See Also
openCRdeprecated

openCRdeprecated

Deprecation in Package openCR

Description
These functions will be removed from future versions of openCR.

Usage

# Deprecated in 2.2.0

# None

See Also
openCRdefunct
openCR.design

Design Data for Open population Models

Description

Internal function used by openCR.fit.

Usage

openCR.design(capthist, models, type, naive = FALSE, stratumcov = NULL,
sessioncov = NULL, timecov = NULL, agecov = NULL,
dframe = NULL, contrasts = NULL, initialage = 0,
minimumage = 0, maximumage = 1, CJSp1 = FALSE, ...)

Arguments

capthist       single-session capthist object
models         list of formulae for parameters of detection
type           character string for type of analysis "CJS", "JSSAfCL" etc. (see openCR.fit)
naive          logical if TRUE then modelled parameter is for a naive animal (not caught previously)
timecov        optional vector or dataframe of values of occasion-specific covariate(s).
stratumcov     optional dataframe of values of stratum-specific covariate(s)
sessioncov     optional dataframe of values of session-specific covariate(s)
agecov         optional dataframe of values of age-specific covariate(s)
dframe         optional data frame of design data for detection parameters
contrasts       contrast specification as for model.matrix
initialage     numeric or character (name of individual covariate containing initial ages)
minimumage     numeric; ages younger than minimum are truncated up
maximumage     numeric; ages older than maximum are truncated down
CJSp1          logical; if TRUE detection is modelled on first primary session in CJS models
...            other arguments passed to the R function model.matrix

Details

This is an internal openCR 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 individual, secondary session, detector and latent class, i.e., for \( n \) individuals, \( S \) secondary sessions, \( K \) detectors and \( m \) latent classes there are potentially \( n \times S \times K \times m \) different values. Actual models always predict
a much reduced set of distinct values, and the number of rows in the design matrix is reduced correspondingly; a parameter index array allows these to retrieved for any combination of individual, session and detector.

openCR.design is less tolerant than openCR.fit regarding the inputs ‘capthist’ and ‘models’. Model formulae are processed by openCR.fit to a standard form (a named list of formulae) before they are passed to openCR.design, and multi-session capthist objects are automatically ‘reduced’ and ‘joined’ for open-population analysis.

If timecov is a single vector of values (one for each secondary session) then it is treated as a covariate named ‘tcov’. If sessioncov is a single vector of values (one for each primary session) then it is treated as a covariate named ‘scov’.

The initialage and maximumage arguments are usually passed via the openCR.fit ‘details’ argument.

agecov may be used to group ages. It should have length (or number of rows) equal to maximumage + 1.

Value

A list with the components

designMatrices list of reduced design matrices, one for each parameter

parameterTable index to row of the reduced design matrix for each parameter; dim(parameterTable) = c(uniquepar, np), where uniquepar is the number of unique combinations of parameter values (uniquepar < nSKM) and np is the number of parameters in the detection model.

PIA Parameter Index Array - index to row of parameterTable for a given animal, occasion and latent class; dim(PIA) = c(n,S,K,M)

validlevels for J primary sessions, a logical matrix of np rows and J columns, mostly TRUE, but FALSE for impossible combinations e.g. CJS recapture probability in session 1 (validlevels["p",1]) unless CJSp1 = TRUE, or CJS final survival probability (validlevels["phi",J]). Also, validlevels["b",1] is FALSE with type = "JSSA..." because of the constraint that entry parameters sum to one.

Note

The component validlevels is TRUE in many cases for which a parameter is redundant or confounded (e.g. validlevels["phi",J-1]); these are sorted out ‘post hoc’ by examining the fitted values, their asymptotic variances and the eigenvalues of the Hessian matrix.

See Also

openCR.fit

Examples

```r
## this happens automatically in openCR.fit
ovenCH1 <- join(reduce(ovenCH, by = "all", newtraps=list(1:44)))
```
openCR.design (ovenCH1, models = list(p = ~1, phi = ~session),
interval = c(1,1,1,1), type = "CJS")

openCR.fit

**Fit Open Population Capture–Recapture Model**

**Description**

Nonspatial or spatial open-population analyses are performed on data formatted for 'secr'. Several parameterisations are provided for the nonspatial Jolly-Seber Schwarz-Arnason model ('JSSA', also known as 'POPAN'). Corresponding spatial models are designated 'JSSAsecr'. The prefix 'PLB' (Pradel-Link-Barker) is used for versions of the JSSA models that are conditional on the number observed. Cormack-Jolly-Seber (CJS) models are also fitted.

**Usage**

```r
openCR.fit (capthist, type = "CJS", model = list(p~1, phi~1, sigma~1),
distribution = c("poisson", "binomial"), mask = NULL,
detectfn = c("HHN", "HHR", "HEX", "HAN", "HCG", "HVP", "HPX"), binomN = 0,
c("truncated", "wrap", "none"), kernelradius = 30, sparsekernel = TRUE,
start = NULL, link = list(), fixed = list(), stratumcov = NULL,
sessioncov = NULL, timecov = NULL, agecov = NULL, dframe = NULL,
dframe0 = NULL, details = list(), method = "Newton-Raphson", trace = NULL,
ncores = NULL, stratified = FALSE, ...)
```

**Arguments**

- `capthist`: capthist object from 'secr'
- `type`: character string for type of analysis (see Details)
- `model`: list with optional components, each symbolically defining a linear predictor for the relevant real parameter using formula notation. See Details for names of real parameters.
- `distribution`: character distribution of number of individuals detected
- `mask`: single-session mask object; required for spatial (secr) models
- `detectfn`: character code
- `binomN`: integer code for distribution of counts (see `secr.fit`)
- `movementmodel`: character; model for movement between primary sessions (see Details)
- `edgemethod`: character; method for movement at edge of mask (see Details)
- `kernelradius`: integer; radius in mask cells of discretized kernel (movement models only)
- `sparsekernel`: logical; if TRUE then only cardinal and intercardinal axes are included
**openCR.fit**

- **start**: vector of initial values for beta parameters, or fitted model(s) from which they may be derived.
- **link**: list with named components, each a character string in {"log", "logit", "loglog", "identity", "sin", "mlogit"} for the link function of the relevant real parameter.
- **fixed**: list with optional components corresponding to each ‘real’ parameter, the scalar value to which parameter is to be fixed.
- **stratumcov**: optional dataframe of values of stratum-specific covariate(s).
- **sessioncov**: optional dataframe of values of session-specific covariate(s).
- **timecov**: optional dataframe of values of occasion-specific covariate(s).
- **agecov**: optional dataframe of values of age-specific covariate(s).
- **dframe**: optional data frame of design data for detection parameters (seldom used).
- **dframe0**: optional data frame of design data for detection parameters of naive (undetected) animals (seldom used).
- **details**: list of additional settings (see Details).
- **method**: character string giving method for maximizing log likelihood.
- **trace**: logical or integer; output log likelihood at each evaluation, or at some lesser frequency as given.
- **ncores**: integer number of cores for parallel processing (see Details).
- **stratified**: logical; if TRUE then sessions of capthist interpreted as independent strata.
- **...**: other arguments passed to join().

**Details**

The permitted nonspatial models are CJS, Pradel, Pradelg, JSSAbCL = PLBb, JSSAfCL = PLBf, JSSAgCL = PLBg, JSSAIcL = PLBl, JSSA, JSSAg, JSSAI, JSSAB and JSSAN.

The permitted spatial models are CJSsecr, JSSAsecrbCL = PLBsecrb, JSSAsecrCL = PLBsecr, JSSAsecrgCL = PLBsecrg, JSSAsecrlCL = PLBsecrl, JSSAsecrf, JSSAsecrg, JSSAsecrB, JSSAsecrN, secrCL, and secrD.

See openCR-vignette.pdf for a table of the ‘real’ parameters associated with each model type.

Parameterisations of the JSSA models differ in how they include recruitment: the core parameterisations express recruitment either as a per capita rate (‘f’), as a finite rate of increase for the population (‘l’ for lambda) or as per-occasion entry probability (‘b’ for the classic JSSA beta parameter, aka PENT in MARK). Each of these models may be fitted by maximising either the full likelihood, or the likelihood conditional on capture in the Huggins (1989) sense, distinguished by the suffix ‘CL’.

Full-likelihood JSSA models may also be parameterized in terms of the time-specific absolute recruitment (BN, BD) or the time-specific population size(N) or density (D).

‘secrCL’ and ‘secrD’ are closed-population spatial models.

Data are provided as secr ‘capthist’ objects, with some restrictions. For nonspatial analyses, ‘capthist’ may be single-session or multi-session, with any of the main detector types. For spatial analyses ‘capthist’ should be a single-session dataset of a point detector type (‘multi’, ‘proximity’ or ‘count’) (see also details$distribution below). In openCR the occasions of a single-session dataset are treated as open-population temporal samples except that occasions separated by an interval of
zero (0) are from the same primary session (multi-session input is collapsed to single-session if necessary).

Model formulae may include the pre-defined terms ‘session’, ‘Session’, ‘h2’, and ‘h3’ as in secr. ‘session’ is the name given to primary sampling times in ‘secr’, so a fully time-specific CJS model is `list(p ~ session, phi ~ session)`. ‘t’ is a synonym of ‘session’. ‘Session’ is for a trend over sessions. ‘h2’ and ‘h3’ allow finite mixture models.

Learned (behavioural) responses (‘b’, ‘B’, etc.) were redefined and extended in version 1.3.0. The vignette should be consulted for current definitions.

Formulae may also include named occasion-specific and session-specific covariates in the dataframe arguments ‘timecov’ and ‘sessioncov’ (occasion = secondary session of robust design). Named age-specific covariates in ‘agecov’ are treated similarly. Individual covariates present as an attribute of the ‘capthist’ input may be used in CJS and .CL models. Groups are not supported in this version, but may be implemented via a factor-level covariate in .CL models.

`distribution` specifies the distribution of the number of individuals detected; this may be conditional on the population size (or number in the masked area) (“binomial”) or unconditional (“poisson”). `distribution` affects the sampling variance of the estimated density. The default is "poisson" as in secr.

Movement models are listed in Movement models. Their use is described in the vignette.

`edgemethod` controls movement probabilities at the mask edge in spatial models that include movement. "none" typically causes bias in estimates; "wrap" wraps kernel probabilities to the opposing edge of a rectangular mask; "truncate" scales the values of an edge-truncated kernel so that they always sum to 1.0 (safer and more general than "wrap").

The `mlogit` link function is used for the JSSA (POPAN) entry parameter ‘b’ (PENT in MARK) and for mixture proportions, regardless of `link`.

Spatial models use one of the hazard-based detection functions (see `detectfn`) and require data from independent point detectors (secr detector types ‘multi’, ‘proximity’ or ‘count’).

Code is executed in multiple threads unless the user specifies `ncores = 1` or there is only one core available or `details$R == TRUE`. Setting `ncores = NULL` uses the existing value from the environment variable `RCPP_PARALLEL_NUM_THREADS` (see `setNumThreads`) or 2 if that has not been set.

Optional stratification was introduced in openCR 2.0.0. See openCR-vignette.pdf for details.

The `...` argument may be used to pass a vector of unequal intervals to join (interval), or to vary the tolerance for merging detector sites (tol).

The `start` argument may be

- a vector of beta parameter values, one for each of the NP beta parameters in the model
- a named vector of beta parameter values in any order
- a named list of one or more real parameter values
- a single fitted secr or openCR model whose real parameters overlap with the current model
- a list of two fitted models

In the case of two fitted models, the values are melded. This is handy for initialising an open spatial model from a closed spatial model and an open non-spatial model. If a beta parameter appears in both models then the first is used.
**openCR.fit**

Details is a list used for various specialized settings –

<table>
<thead>
<tr>
<th>Component</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoini</td>
<td>1</td>
<td>Number of the session used to determine initial values of D, lambda0 and sigma (secr types only)</td>
</tr>
<tr>
<td>CJSp1</td>
<td>FALSE</td>
<td>Modified CJS model including initial detection (estimable with robust design and many spatial models)</td>
</tr>
<tr>
<td>contrasts</td>
<td>NULL</td>
<td>Value suitable for the ‘contrasts.arg’ argument of <code>model.matrix</code> used to specify the coding of factor predictors</td>
</tr>
<tr>
<td>control</td>
<td>list()</td>
<td>Components may be named arguments of <code>nlm</code>, or passed intact as argument ‘control’ of <code>optim</code> -</td>
</tr>
<tr>
<td>debug</td>
<td>0</td>
<td>debug=1 prints various intermediate values; debug&gt;=2 interrupts execution by calling browser()</td>
</tr>
<tr>
<td>fixedbeta</td>
<td>NULL</td>
<td>Vector with one element for each coefficient (beta parameter) in the model. Only 'NA' coefficients</td>
</tr>
<tr>
<td>grain</td>
<td>1</td>
<td>Obscure setting for multithreading - see RcppParallel package</td>
</tr>
<tr>
<td>hessian</td>
<td>&quot;auto&quot;</td>
<td>Computation of the Hessian matrix from which variances and covariances are obtained. Options</td>
</tr>
<tr>
<td>ignoreusage</td>
<td>FALSE</td>
<td>Overrides usage in traps object of capthist</td>
</tr>
<tr>
<td>initialage</td>
<td>0</td>
<td>Numeric (uniform age at first capture) or character value naming an individual covariate; see age.matrix</td>
</tr>
<tr>
<td>initialstratum</td>
<td>1</td>
<td>Number of stratum to use for finding default starting values (cf autoini in secr)</td>
</tr>
<tr>
<td>LLonly</td>
<td>TRUE</td>
<td>TRUE causes the function to return a single evaluation of the log likelihood at the initial values,</td>
</tr>
<tr>
<td>minimumage</td>
<td>0</td>
<td>Sets a minimum age; see age.matrix</td>
</tr>
<tr>
<td>maximumage</td>
<td>1</td>
<td>Sets a maximum age; older animals are recycled into this age class; see age.matrix</td>
</tr>
<tr>
<td>multinom</td>
<td>FALSE</td>
<td>Include the multinomial constant in the reported log-likelihood.</td>
</tr>
<tr>
<td>r0</td>
<td>0.5</td>
<td>effective radius of zero cell in movement kernel (multiple of cell width)</td>
</tr>
<tr>
<td>R</td>
<td>FALSE</td>
<td>Switch from the default C++ code to slower functions in native R (useful for debugging; not all)</td>
</tr>
<tr>
<td>squeeze</td>
<td>TRUE</td>
<td>Apply squeeze to capthist before analysis. Non-spatial models fit faster, because histories often</td>
</tr>
</tbody>
</table>

If `method = "Newton-Raphson"` then `nlm` is used to maximize the log likelihood (minimize the negative log likelihood); otherwise `optim` is used with the chosen method ("BFGS", "Nelder-Mead", etc.). If maximization fails a warning is given appropriate to the method. `method = "none"` may be used to compute or re-compute the variance-covariance matrix at given starting values (i.e. providing a previously fitted model as the value of start).

Parameter redundancies are common in open-population models. The output from `openCR.fit` includes the singular values (eigenvalues) of the Hessian - a useful post-hoc indicator of redundancy (e.g., Gimenez et al. 2004). Eigenvalues are scaled so the largest is 1.0. Very small scaled values represent redundant parameters - in my experience with simple JSSA models a threshold of 0.00001 seems effective.

[There is an undocumented option to fix specific ‘beta’ parameters.]

**Value**

If `details$LLonly == TRUE` then a numeric vector is returned with logLik in position 1, followed by the named coefficients.

Otherwise, an object of class ‘openCR’ with components

- **call**: function call
- **capthist**: saved input (unique histories; see covariates(capthist)$freq for frequencies)
- **type**: saved input
- **model**: saved input
- **distribution**: saved input
The environment variable RCPP_PARALLEL_NUM_THREADS is updated with the value of ncores if provided.
Note

Different parameterisations lead to different model fits when used with the default `model` argument in which each real parameter is constrained to be constant over time.

The JSSA implementation uses summation over feasible 'birth' and 'death' times for each capture history, following Pledger et al. (2010). This enables finite mixture models for individual capture probability (not fully tested), flexible handling of additions and losses on capture (aka removals) (not yet programmed), and ultimately the extension to 'unknown age' as in Pledger et al. (2009).

openCR uses the generalized matrix inverse ‘ginv’ from the MASS package rather than ‘solve’ from base R, as this seems more robust to singularities in the Hessian. Also, the default maximization method is ‘BFGS’ rather than ‘Newton-Raphson’ as BFGS appears more robust in the presence of redundant parameters.

Earlier versions of openCR.fit computed latent class membership probabilities for each individual in finite mixture models and saved them in component 'posterior'. Now see classMembership for that functionality.

From 1.5.0 onwards the number of threads uses the environment variable RCPP_PARALLEL_NUM_THREADS, as in secr.fit. This may be set once in a session with secr::setNumThreads.

The default movement arguments changed in openCR 2.1.1. Now kernelradius = 30, sparsekernel = TRUE.

References


See Also

classMembership.openCR, derived.openCR, openCR.design, par.openCR.fit, predict.openCR, summary.openCR

Examples

## Not run:

## CJS default
openCR.fit(ovenCH)

## POPAN Jolly-Seber Schwarz-Arnason, lambda parameterisation
L1 <- openCR.fit(ovenCH, type = 'JSSA1')
predict(L1)

JSSA1 <- openCR.fit(ovenCH, type = 'JSSAf')
JSSA2 <- openCR.fit(ovenCH, type = 'JSSAf', model = list(phi~t))
JSSA3 <- openCR.fit(ovenCH, type = 'JSSAf', model = list(p~t,phi~t))
AIC (JSSA1, JSSA2, JSSA3)
predict(JSSA1)

RMdata <- RMarkInput (join(reduce(ovenCH, by = "all")))
if (require(RMark)) {
  MarkPath <- '\c:/Mark/'
  if (!all (nchar(Sys.which(c('mark.exe', 'mark64.exe', 'mark32.exe'))) < 2)) {
    openCHtest <- process.data(RMdata, model = 'POPAN')
    openCHPOPAN <- mark(data = openCHtest, model = 'POPAN',
                       model.parameters = list(p = list(formula = ~1),
                       pent = list(formula = ~1),
                       Phi = list(formula = ~1)))
    popan.derived(openCHtest, openCHPOPAN)
    cleanup(ask = FALSE)
  } else message("mark.exe not found")
} else message("RMark not found")

## End(Not run)

---

**openCRlist**  
*Bundle openCR Models*

**Description**
Fitted models are bundled together for convenience.

**Usage**

```r
desc

# S3 method for class 'openCRlist'
x[i]
```

**Arguments**

- `...`  
- `x`  
- `i`  

- `openCR objects`
- `openCRlist`
- `indices`
par.openCR.fit

Details
	onopenCRlist forms a special list (class ‘openCRlist’) of fitted model (openCR) objects. This may be used as an argument of AIC, predict, make.table etc.

Methods are provided for the generic function c and list extraction ‘[]’.

Value
	onopenCRlist object

See Also

AIC.openCR, predict.openCR, make.table

Examples

---

```r
## Not run:
fit0 <- openCR.fit(dipperCH)
fitt <- openCR.fit(dipperCH, model=phi~t)
fits <- openCRlist(fit0, fitt)
AIC(fits)
make.table(fits, ‘phi’)

## End(Not run)
```

par.openCR.fit

Fit Multiple openCR Models

Description

This function is a wrapper for openCR.fit.

Usage

```r
par.openCR.fit(arglist, ncores = 1, seed = 123, trace = FALSE, logfile = NULL, prefix = ””)
```

Arguments

- `arglist`: list of argument lists for secr.fit or a character vector naming such lists
- `ncores`: integer number of cores used by parallel::makeClusters()
- `seed`: integer pseudorandom number seed
- `trace`: logical; if TRUE intermediate output may be logged
- `logfile`: character name of file to log progress reports
- `prefix`: character prefix for names of output
Details

In openCR >= 1.5.0, setting ncores > 1 is deprecated and triggers a warning: multithreading makes it faster to set ncores = 1 in par.secr.fit.

trace overrides any settings in arglist.

It is convenient to provide the names of the capthist and mask arguments in each component of arglist as character values (i.e. in quotes); objects thus named are exported from the workspace to each worker process (see Examples).

Using ncores>1 is obsolete under the multithreading regime in openCR >= 1.5.0. It is usually slower than ncores = 1. If used it has these effects:

– worker processes are generated using the parallel package,
– one model is fitted on each worker, and
– if no logfile name is provided then a temporary file name will be generated in tempdir().

Value

For par.openCR.fit - openCRlist of model fits (see openCR.fit and openCRlist). Names are created by prefixing prefix to the names of argslist. If trace is TRUE then the total execution time and finish time are displayed.

Note

Any attempt in arglist to set ncores > 1 for a particular openCR fit was ignored in openCR < 1.5.0. Now it is allowed.

See Also

openCR.fit, Parallel, make.table, openCRlist

Examples

```r
# Not run:
m1 <- list(capthist = ovenCH, model = list(p~1, phi~1))
m2 <- list(capthist = ovenCH, model = list(p~session, phi~1))
m3 <- list(capthist = ovenCH, model = list(p~session, phi~session))
setNumThreads(7)  # on quadcore Windows PC
fits <- par.openCR.fit (c('m1','m2','m3'), ncores = 1)
AIC(fits)
```

# End(Not run)
pkernel

Kernel Distribution Functions

Description

Distribution of distance moved for each of the main movement kernels. Theoretical probability density, cumulative distribution function, and quantile function (inverse of the cumulative distribution function).

Usage

pkernel(q, movementmodel = c("BVN", "BVE", "BVC", "BVT", "RDE", "RDG", "RDL"), move.a, move.b, truncate = Inf, lower.tail = TRUE)
dkernel(r, movementmodel = c("BVN", "BVE", "BVC", "BVT", "RDE", "RDG", "RDL"), move.a, move.b, truncate = Inf)
qkernel(p, movementmodel = c("BVN", "BVE", "BVC", "BVT", "RDE", "RDG", "RDL"), move.a, move.b, truncate = Inf, lower.tail = TRUE)
gkernel(r, movementmodel = c("BVN", "BVE", "BVC", "BVT", "RDE", "RDG", "RDL"), move.a, move.b, truncate = Inf)

Arguments

p numeric vector of cumulative probabilities (0.5 for median)
r numeric vector of distance moved
q numeric vector of quantiles (distance moved)
movementmodel character (see Movement models and openCR-vignette.pdf)
move.a numeric parameter of movement kernel
move.b numeric parameter of movement kernel
truncate numeric q value at which distribution truncated
lower.tail logical; if TRUE (default), probabilities are P[X <= x] otherwise, P[X > x].

Details

Some formulae are given in openCR-vignette.pdf. gkernel gives the 2-D probability density of the bivariate kernel \( g(r) = f(r)/(2\pi r) \); the remaining functions describe the distribution of distance moved \( f(r) \).

Computation of qkernel for movementmodel = 'BVE' uses numerical root finding (function unroot). Truncation (truncate = limit for finite limit) adjusts probabilities upwards by \( 1/pkernel(limit, ..., truncate = limit) \) so that pkernel(limit, ..., truncate = limit) equals 1.0. By default the distribution is not truncated.
Value

For pkernel –
Vector of cumulative probabilities corresponding to q. The cumulative probability is 1.0 for q > truncate.

For dkernel –
Vector of probability density at radial distance r (zero for r > truncate).

For qkernel –
Vector of quantiles (distances moved) corresponding to cumulative probabilities p.

For gkernel –
Vector of 2-D probability density at radial distance r (zero for r > truncate).

See Also

Movement models, make.kernel, matchscale

Examples

# plot 3 distributions chosen with matchscale to intersect at p = 0.5
q <- 0:100
plot(q, pkernel(q, 'BVN', 34), type = 'l', ylab = 'Cumulative probability')
lines(q, pkernel(q, 'BVT', move.a = 104, move.b = 5), col = 'darkgreen', lwd = 2)
lines(q, pkernel(q, 'BVT', move.a = 40, move.b = 1), col = 'orange', lwd = 2)
points(40, 0.5, pch = 16)
legend(62, 0.36, lty=1, lwd = 2, col = c('black', 'darkgreen', 'orange'),
      legend = c('BVN sigma=34', 'BVT a=104, b=5', 'BVT a=40, b=1'))

# median
abline(v = qkernel(0.5, 'BVN', 34))
Arguments

- `x`: openCR object from openCR.fit
- `par`: character names of parameter to plot
- `add`: logical; if TRUE then points are added to an existing plot
- `xoffset`: numeric offset to be added to all x values
- `ylim`: numeric vector of limits on y-axis
- `useintervals`: logical; if TRUE then x values are spaced according to the intervals attribute
- `intermediate.x`: logical; if TRUE then turnover parameters are plotted at the mid point on the x axis of the interval to which they relate
- `...`: other arguments passed to `plot`, `points` and `segments`

Details

If `ylim` is not provided it is set automatically.
Confidence intervals are not available in this version.

Value

The x coordinates (including `xoffset`) are returned invisibly.

See Also

- `plot.openCR`

Examples

```r
## Not run:
fit <- openCR.fit(dipperCH, type='JSSAfCL', model = phi~session)
der <- derived(fit)
plot(der,'N', pch = 16, cex = 1.3)

## End(Not run)
```

Description

Session-specific estimates of the chosen parameter are plotted.
plot.openCR

Usage

## S3 method for class 'openCR'
plot(x, par = "phi", newdata = NULL, add = FALSE, xoffset = 0, ylim = NULL,
     useintervals = TRUE, CI = TRUE, intermediate.x = TRUE, alpha = 0.05, stratum = 1, ...)

Arguments

- **x**: openCR object from openCR.fit
- **par**: character names of parameter to plot
- **newdata**: dataframe of predictor values for predict (optional)
- **add**: logical; if TRUE then points are added to an existing plot
- **xoffset**: numeric offset to be added to all x values
- **ylim**: numeric vector of limits on y-axis
- **useintervals**: logical; if TRUE then x values are spaced according to the intervals attribute
- **CI**: logical; if TRUE then 1-alpha confidence intervals are plotted
- **intermediate.x**: logical; if TRUE then turnover parameters are plotted at the mid point on the x axis of the interval to which they relate
- **alpha**: numeric confidence level default (alpha = 0.05) is 95% interval
- **stratum**: numeric; stratum to plot if more than one
- **...**: other arguments passed to plot, points and segments

Details

If ylim is not provided it is set automatically.

For customization you may wish to prepare a base plot with plot(... , type = 'n') and use add = TRUE.

Value

The x coordinates (including xoffset) are returned invisibly.

See Also

predict, plot.derivedopenCR

Examples

## Not run:

fit <- openCR.fit(join(ovenCH), type='CJS', model = phi~session)
plot(fit,'phi', pch = 16, cex=1.3, yl=c(0,1))

## End(Not run)
Description

A subset of brushtail possum (Trichosurus vulpecula) data from the Orongorongo Valley live-trapping study of Efford (1998) and Efford and Cowan (2005) that was used by Pledger, Pollock and Norris (2003, 2010). The OVpossumCH dataset in secr is a different selection of data from the same study. Consult ?OVpossumCH for more detail.

The data comprise captures in February of each year from 1980 to 1988.

Usage

FebpossumCH

Format

The format is a 9-session secr capthist object. Capture locations are not included.

Details

The data are captures of 448 animals (175 females and 273 males) over 9 trapping sessions comprising 4–10 occasions each. All were independent of their mothers, but age was not otherwise distinguished. The individual covariate sex takes values ’F’ or ’M’.

Pledger, Pollock and Norris (2010) fitted 2-class finite mixture models for capture probability p and apparent survival phi, with or without allowance for temporal (between year) variation, using captures from only the first day of each trapping session. The first-day data relate to 270 individuals (115 females and 155 males).

Source


References


Examples

summary(FebpossumCH)
m.array(FebpossumCH)
JS.counts(FebpossumCH)

FebD1CH <- subset(FebpossumCH, occasion = 1)

## Not run:
# reading the text file 'poss8088.data'

datadir <- system.file('extdata', package = 'openCR')
poss8088df <- read.table(paste0(datadir,'/poss8088.data'), header = TRUE)
capt <- poss8088df[,c('session', 'id', 'day', 'day', 'sex')]

# duplication of day is a trick to get a dummy trapID column in the right place
# this is needed because make.capthist does not have nonspatial option
capt$day.1[] <- 1

# keep only February samples
capt <- capt[ capt$session %% 3 == 1, ]

# build nonspatial secr capthist object using dummy trapping grid
FebpossumCH <- make.capthist(capt, make.grid(1,2, ID='numx'))
# discard dummy traps objects
for (i in 1:9) attr(FebpossumCH[[i]], 'traps') <- NULL
names(FebpossumCH) <- 1980:1988
sessionlabels(FebpossumCH) <- 1980:1988

## End(Not run)

predict.openCR  

Description

Evaluate an openCR 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 'openCR'
predict(object, newdata = NULL, se.fit = TRUE, alpha = 0.05, savenew = FALSE, ...)
## S3 method for class 'openCRlist'
predict(object, newdata = NULL, se.fit = TRUE, alpha = 0.05, savenew = FALSE, ...)
print.derivedopenCR

Arguments

  object       openCR object output from openCR.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
  savenew     logical; if TRUE then newdata is saved as an attribute
  ...         other arguments passed to makeNewData

Details

Predictions are provided for each row in ‘newdata’. The default (constructed by makeNewData) is to limit those rows to the first-used level of factor predictors; to include all levels pass all.levels = TRUE to makeNewData in the ...argument.

See Also

AIC.openCR, openCR.fit

Examples

  ## Not run:
  c1 <- openCR.fit(ovenCH, type='CJS', model=phi~session)
  predict(c1)

  ## End(Not run)

print.derivedopenCR  Print Method for Derived Estimates

Description

Formats output from derived.openCR.

Usage

  ## S3 method for class 'derivedopenCR'
  print(x, Dscale = NULL, legend = FALSE, ...)


print.openCR

Arguments

- **x**: openCR object output from openCR.fit
- **object**: openCR object output from openCR.fit
- **newdata**: optional dataframe of values at which to evaluate model
- **alpha**: alpha level
- **svtol**: threshold for non-null eigenvalues when computing numerical rank
- **deriv**: logical; if TRUE then table of derived parameters is calculated
- **...**: other arguments passed to derived.openCR by summary.openCR

Details

By default (i.e. when not not specified in the in the ...argument), row.names = FALSE and digits = 4.

See Also

derived.openCR

---

**print.openCR**

*Print or Summarise openCR Object*

Description

Print results from fitting a spatially explicit capture–recapture model, or generate a list of summary data.

Usage

```r
## S3 method for class 'openCR'
print(x, newdata = NULL, alpha = 0.05, svtol = 1e-5,...)
## S3 method for class 'openCR'
summary(object, newdata = NULL, alpha = 0.05, svtol = 1e-5, deriv = FALSE, ...)
```

Arguments

- **x**: openCR object output from openCR.fit
- **object**: openCR object output from openCR.fit
- **newdata**: optional dataframe of values at which to evaluate model
- **alpha**: alpha level
- **svtol**: threshold for non-null eigenvalues when computing numerical rank
- **deriv**: logical; if TRUE then table of derived parameters is calculated
- **...**: other arguments passed to derived.openCR by summary.openCR

---
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 sessions
number of sampling occasions

Model
model formula for each ‘real’ parameter

Fixed
fixed real parameters

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

Eigenvalues
scaled eigenvalues of Hessian matrix (maximum 1.0)

Numerical rank
number of eigenvalues exceeding svtol

vcov
variance-covariance matrix of beta parameters

Real parameters
fitted (real) parameters evaluated at base levels of covariates

AICc is computed with the default sample size (number of individuals) and parameter count (use.rank = FALSE).

Value

The summary method constructs a list of outputs similar to those printed by the print method, but somewhat more concise and re-usable:

version
secr version, and date and time fitting started

time
date and time fitting started

traps*
detector summary

capthist
capthist summary (primary and secondary sessions, numbers of animals and detections)

intervals
intervals between primary sessions

mask*
mask summary

modeldetails
miscellaneous model characteristics (type etc.)

AICtable
single-line output of AIC.openCR

coef
table of fitted coefficients with CI

predicted
predicted values (‘real’ parameter estimates)

derived
output of derived.openCR (optional)

* spatial models only

References


See Also

AIC.openCR, openCR.fit
read.inp

Examples

```r
## Not run:
c1 <- openCR.fit(ovenCH, type='CJS', model=phi~session)
c1

## End(Not run)
```

read.inp

Import Data from RMark Input Format

Description

read.inp forms a capthist object from a MARK input (.inp) file.

Usage

```r
read.inp(filename, ngroups = 1, grouplabel = 'group', grouplevels = NULL,
covnames = NULL, skip = 0)
```

Arguments

- `filename` character file name including `.inp`.
- `ngroups` integer number of group columns in input
- `grouplabel` character
- `grouplevels` vector with length equal to number of groups
- `covnames` character vector of additional covariates names, one per covariate column
- `skip` integer number of lines to skip at start of file

Details

Comments bracketed with `/*` and `*/` will be removed automatically.

If `grouplevels` is specified then `ngroups` is taken from the number of levels (`ngroups` is overridden). An individual covariate is output, named according to `grouplabel`. The order of levels in `grouplevels` should match the order of the group frequency columns in the input. This also determines the ordering of levels in the resulting covariate.

Value

A single-session capthist object with no traps attribute.
rev.capthist

See Also
- RMarkInput, unRMarkInput

Examples

```r
datadir <- system.file('extdata', package = 'openCR')
dipperCH <- read.inp(paste0(datadir, '/ed.inp'), ngroups = 2)
summary(dipperCH)
```

---

rev.capthist  
Reverse Primary Sessions

Description

The `rev` method for capthist objects reverses the order of the primary sessions while retaining the order of secondary sessions within each primary session.

Usage

```r
## S3 method for class 'capthist'
rev(x)
```

Arguments

- `x`  
  multi-session capthist object from secr

Details

`rev()` is used to demonstrate 'reversed time' analyses (Nichols 2016) in which seniority (gamma) is estimated as reversed-time survival (phi) The approach is numerically equivalent to direct modelling of seniority (see Examples). Direct modelling allows more control and is more intuitive.

If `x` is not overtly multi-session and has no intervals attribute then each occasion is treated as a primary session.

Value

Capthist object with same observations as input, but re-ordered. The order of attributes `sessionlabels` and `intervals` is also reversed. A default intervals attribute is added if the input lacks one.

References

Examples

```r
summary(rev(ovenCH), terse = TRUE)

# These three models give the same result for gamma except for
# gamma(1982) which is confounded with p and not separately estimable:

## Not run:
dipperPradel <- openCR.fit(dipperCH, type = "Pradelg", model = list(p~t, phi~t, gamma~t))
revdipper <- openCR.fit(rev(dipperCH), model=list(p~t, phi~t))
dipperJSSA <- openCR.fit(dipperCH, type='JSSAgCL', model=list(p~t, phi~t, gamma~t))

predict(dipperPradel)$gamma
predict(revdipper)$phi
predict(dipperJSSA)$gamma

## End(Not run)
```

---

**simulation**

**Simulate Capture Histories**

Description

Generate non-spatial or spatial open-population data and fit models.

Usage

```r
sim.nonspatial (N, turnover = list(), p, nsessions, noccasions = 1, intervals = NULL,
recapfactor = 1, seed = NULL, savepopn = FALSE, ...)
runsim.nonspatial (nrepl = 100, seed = NULL, ncores = NULL, fitargs = list(),
extractfn = predict, ...)
runsim.spatial (nrepl = 100, seed = NULL, ncores = NULL, popargs = list(),
detargs = list(), fitargs = list(), extractfn = predict, intervals = NULL)
sumsims (sims, parm = 'phi', session = 1, dropifnoSE = TRUE, svtol = NULL,
maxcode = 3, true = NULL)
runsim.RMark (nrepl = 100, model = "CJS", model.parameters = NULL, extractfn,
seed = NULL, ...)
```
Arguments

N  integer population size

turnover  list as described for turnover

p  numeric detection probability

nsessions  number of primary sessions

noccasions  number of secondary sessions per primary session

intervals  intervals between primary sessions (see Details)

recapfactor  numeric multiplier for capture probability after first capture

seed  random number seed see random numbers

savepopn  logical; if TRUE the generated population is saved as an attribute of the capthist object

...  other arguments passed to sim.popn (sim.nonspatial) or sim.nonspatial (run.sims)

nrepl  number of replicates

ncores  integer number of cores to be used for parallel processing (see Details)

popargs  list of arguments for sim.popn

detargs  list of arguments for sim.capthist

fitargs  list of arguments for openCR.fit

extractfn  function applied to each fitted openCR model

sims  list output from runsim.nonspatial or runsim.spatial

parm  character name of parameter to summarise

session  integer vector of session numbers to summarise

dropifnoSE  logical; if TRUE then replicates are omitted when SE missing for parm

svtol  numeric; minimum singular value (eigenvalue) considered non-zero

maxcode  integer; maximum accepted value of convergence code

true  true value of requested parm in given session

model  character; RMark model type

model.parameters  list with RMark model specification (see ?mark)

Details

For sim.nonspatial – If intervals is specified then the number of primary and secondary sessions is inferred from intervals and nsessions and noccasions are ignored. If N and p are vectors of length 2 then subpopulations of the given initial size are sampled with the differing capture probabilities and the resulting capture histories are combined.

runsim.spatial is a relatively simple wrapper for sim.popn, sim.capthist, and openCR.fit. Some arguments are set automatically: the sim.capthist argument 'renumber' is always FALSE; argument 'seed' is ignored within 'popargs' and 'detargs'; if no 'traps' argument is provided in 'detargs' then 'core' from 'popargs' will be used; detargs$popn and fitargs$capthist are derived
from the preceding step. The 'type' specified in fitargs may refer to a non-spatial or spatial open-population model ("CJS", "JSSAsecrfCL" etc.). If the intervals argument is specified it is used to set the intervals attribute of the simulated capthist object; turnover parameters in sim.popn are not scaled by intervals.

Control of parallel processing changed in openCR 1.5.0 to conform to secr. In runsim.nonspatial and runsim.spatial, if ncores is NULL (the default) then the number of cores used for multithreading by openCR.fit is controlled by the environment variable RCPP_PARALLEL_NUM_THREADS. Use the secr function setNumThreads to set RCPP_PARALLEL_NUM_THREADS to a value greater than the default (2, from openCR 1.5 onwards).

Otherwise, (ncores specified in runsim.nonspatial or runsim.spatial) 'ncores' is set to 1 for each replicate and the replicates are split across the specified number of cores.

sumsim assumes output from runsim.nonspatial and runsim.spatial with 'extractfn = predict' or 'extractfn = summary'. Missing SE usually reflects non-identifiability of a parameter or failure of maximisation, so these replicates are dropped by default. If svtol is specified then the rank of the Hessian is determined by counting eigenvalues that exceed svtol, and replicates are dropped if the rank is less than the number of beta parameters. A value of 1e-5 is suggested for svtol in AIC.openCR, but smaller values may be appropriate for larger models (MARK has its own algorithm for this threshold).

Replicates are also dropped by sumsims if the convergence code exceeds 'maxcode'. The maximisation functions nlm (used for method = 'Newton-Raphson', the default), and optim (all other methods) return different convergence codes; their help pages should be consulted. The default is to accept code = 3 from nlm, although the status of such maximisations is ambiguous.

### Value

- **sim.nonspatial** – A capthist object representing an open-population sample
- **runsim.nonspatial and runsim.spatial** – List with one component (output from extractfn) for each replicate. Each component also has attributes 'eigH' and 'fit' taken from the output of openCR.fit. See Examples to extract convergence codes from 'fit' attribute.
- **sumsim** – Data.frame with rows 'estimate', 'SE.estimate', 'lcl', 'ucl', 'RSE', 'CI.length' and columns for median, mean, SD and n. If 'true' is specified there are additional rows are 'Bias' and 'RB', and columns for 'rRMSE' and 'COV'.

### See Also

- sim.popn, sim.capthist

### Examples

```r
## Not run:
cores <- 2 # for CRAN check; increase as available
```
ch <- sim.nonspatial(100, list(phi = 0.7, lambda = 1.1), p = 0.3, nsessions = 8, noccasions=2)
openCR.fit(ch, type = 'CJS')

turnover <- list(phi = 0.85, lambda = 1.0, recrmodel = 'constantN')
set.seed(123)

## using type = 'JSSAlCL' and extractfn = predict
fitarg <- list(type = 'JSSAlCL', model = list(p~t, phi~t, lambda~t))
out <- runsim.nonspatial(nrepl = 100, N = 100, cores = cores, turnover = turnover,
p = 0.2, recapfactor = 4, nsessions = 10, noccasions = 1, fitargs = fitarg)
sumsims(out, 'lambda', 1:10)

## using type = 'Pradelg' and extractfn = derived
## homogeneous p
fitarg <- list(type = 'Pradelg', model = list(p~t, phi~t, gamma~t))
outg <- runsim.nonspatial(nrepl = 100, N = 100, cores = cores, turnover = turnover,
p = c(0.3,0.9), recapfactor = 4, nsessions = 10, noccasions = 1,
fitargs = fitarg, extractfn = derived)
apply(sapply(outg, function(x) x$estimates$lambda),1,mean)

## 2-class mixture for p
outg2 <- runsim.nonspatial(nrepl = 100, N = c(50,50), cores = cores, turnover = turnover,
p = c(0.3,0.9), recapfactor = 1, nsessions = 10, noccasions = 1,
fitargs = fitarg, extractfn = derived)
outg3 <- runsim.nonspatial(nrepl = 100, N = c(50,50), cores = cores, turnover = turnover,
p = c(0.3,0.3), recapfactor = 1, nsessions = 10, noccasions = 1,
fitargs = fitarg, extractfn = derived)
apply(sapply(outg2, function(x) x$estimates$lambda),1,mean)

plot(2:10, apply(sapply(outg2, function(x) x$estimates$lambda),1,mean)[-1],
type='o', xlab = c(1,10), ylab = c(0.9,1.1))

## RMark

extfn <- function(x) x$results$real$estimate[3:11]
MarkPath <- 'c:/mark' # customise as needed
turnover <- list(phi = 0.85, lambda = 1.0, recrmodel = 'discrete')
outrm <- runsim.RMark (nrepl = 100, model = 'Pradlambda', extractfn = extfn,
model.parameters = list(Lambda=list(formula=~time)),
N = c(200,200), turnover = turnover, p = c(0.3,0.9),
recapfactor = 1, nsessions = 10, noccasions = 1)
extout <- apply(do.call(rbind, outrm),1,mean)

## Spatial

ggrid <- make.grid()
msk <- make.mask(grid, type = 'trapbuffer', nx = 32)
turnover <- list(phi = 0.8, lambda = 1)
poparg <- list(D = 10, core = grid, buffer = 100, Ndist = 'fixed', nsessions = 6,
details = turnover)
detarg <- list(nooccasions = 5, detectfn = 'HHN', detectpar = list(lambda0 = 0.5, sigma = 20))
fittarg <- list(type = 'JSSAscrfCL', mask = msk, model = list(phi~1, f~1))
sims <- runsim.spatial (nrepl = 7, ncores = cores, pop = poparg, det = detarg, fit = fitarg)
sumsims(sims)

## extract the convergence code from nlm for each replicate in preceding simulation
sapply(lapply(sims, attr, 'fit'), '[', 'code')
## if method != 'Newton-Raphson then optim is used and the code is named 'convergence'
# sapply(lapply(sims, attr, 'fit'), '[', 'convergence')

## End(Not run)

---

**squeeze**

**Unique Capture Histories**

**Description**

Compresses or expands capthist objects.

**Usage**

```r
squeeze(x)
unsqueeze(x)
```

**Arguments**

- `x` secr capthist object

**Details**

Although squeeze may be applied to spatial capthist objects, the effect is often minimal as most spatial histories are unique.

The `freq` covariate is used by `openCR.fit` to weight summaries and likelihoods. It is currently ignored by `secr.fit`.

**Value**

Both functions return a capthist object with one row for each unique capture history (including covariates). The individual covariate `freq` records the number of instances of each unique history in the input.

**See Also**

- `openCR.fit`
Examples

squeeze(captdat)

<table>
<thead>
<tr>
<th>strata</th>
<th>Stratum names</th>
</tr>
</thead>
</table>

Description

Extract or replace the stratum names of a capthist object.

Usage

strata(object, ...)
strata(object) <- value

Arguments

object object with 'stratum' attribute e.g. capthist
value character vector or vector that may be coerced to character, one value per stratum
... 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

openCR uses the term ‘stratum’ for an independent set of samples, rather like a ‘session’ in secr. Strata offer flexibility in defining and evaluating between-stratum models. The log likelihood for a stratified model is the sum of the separate stratum log likelihoods. Although this assumes independence of sampling, parameters may be shared across strata, or stratum-specific parameter values may be functions of stratum-level covariates. The detector array and mask can be specified separately for each stratum.

For open population analyses, each stratum comprises both primary and secondary sessions of Pollock’s robust design ‘joined’ in a single-session capthist object.

The function stratify can be useful for manipulating data into multi-stratum form.

Models are stratified only if the argument stratified of openCR.fit() is set to TRUE. Strata will otherwise be treated as primary sessions and concatenated as usual with join().
**stratify**

**Stratify Capture-Recapture Data**

**Description**

Arrange existing capthist data in stratified form.

**Usage**

```r
stratify(..., intervals = NULL, MoreArgs = list(), covariate = NULL, bytraps = FALSE)
```

**Arguments**

- `...` one or more multi-session capthist objects, or a list of such objects
- `intervals` list of intervals vectors, one for each multi-session capthist in ...
- `MoreArgs` list of other arguments passed to `join`
- `covariate` character; name of individual or trap covariate to stratify by
- `bytraps` logical; if TRUE then covariate is interpreted as the name of a detector covariate

**Details**

The argument `...` may be

1. a list of single-session capthist, one for each stratum (sessions already joined)
2. a list of multi-session capthist, one for each stratum (sessions will be joined)
3. one single-session capthist, to split by `covariate` (sessions already joined)
4. one multi-session capthist, to be joined as one then split by `covariate`

Cases 1 and 2 result in one stratum for each component of the input list. Cases 3 and 4 result in one stratum for each level of `covariate`.

The result in Case 1 is identical to `MS.capthist(...)`.

The argument `intervals` refers to the intervals between primary sessions before joining (Cases 2, 4 only) (see Examples).

`MoreArgs` may include the arguments `remove.dupl.sites`, `tol`, `sites.by.name` or `drop.sites` of `join`; these otherwise take their default values.

**See Also**

`openCR.fit`, `session`, `stratify`

**Examples**

```r
# artificial example, treating years as strata
strata(ovenCH)
```

```r
stratify(..., intervals = NULL, MoreArgs = list(), covariate = NULL, bytraps = FALSE)
```
Value

Multi-stratum (multi-session) capthist object for which each component has been 'join'ed.

See Also

join, MS.capthist, openCR.fit.strata

Examples

# FebpossumCH comprises 9 annual February sessions.
# The individual covariate 'sex' takes values 'F' and 'M', resulting in two strata.
# 'intervals' can be omitted as the default does the same job.
ch <- stratify(FebpossumCH, covariate = 'sex', intervals = rep(list(rep(1,8)),2))
summary(ch, terse = TRUE)

ucare.cjs

Goodness-of-fit tests for the Cormack-Jolly-Seber model

Description

The package R2ucare (Gimenez et al. 2017, 2018) provides the standard tests for CJS models from Burnham et al. (1987) along with tests for multi-state models as described by Pradel et al. (2005). This function is a wrapper for the tests relevant to openCR (see Details). Original papers and the associated vignette for R2ucare should be consulted for interpretation.

Usage

ucare.cjs(CH, tests = "all", by = NULL, verbose = TRUE, rounding = 3, ...)

Arguments

CH capthist object suitable for openCR
tests character vector with the names of specific tests (see Details) or 'all'
by character name of covariate in CH used to split rows of CH into separate groups
verbose logical; if TRUE then additional details are tabulated
rounding integer number of decimal places in output
... other arguments passed to split.capthist if needed
Details

The possible tests are "test3sr", "test3sm", "test2ct", "test2cl", and "overall_CJS".

If CH is a multi-session object then it will first be collapsed to a single-session object with join as usual in openCR. If CH has an intervals attribute indicating that the data are from a robust design (some intervals zero) then it will first be collapsed to one secondary session per primary session, with a warning.

If by is specified it should point to a categorical variable (factor or character) in the covariates attribute of CH. Separate tests will be conducted for each group.

Value

A list of results, possibly nested by the grouping variable by. The verbose form includes both the overall result of each test and its breakdown into components ('details').

References


See Also

m.array

Examples

```r
if (requireNamespace("R2ucare"))
  ucare.cjs(dipperCH, verbose = FALSE, by = 'sex')
```
Index

* datagen
  simulation, 69
* datasets
  dipperCH, 14
  Field vole, 17
  gonodontisCH, 20
  Microtus, 35
  PPNpossums, 61
* hplot
  LLsurface, 27
  make.kernel, 29
  plot.derivedopenCR, 58
  plot.openCR, 59
* htest
  ucare.cjs, 76
* manip
  age.matrix, 4
  JS.counts, 24
  make.table, 32
  miscellaneous, 38
  openCR.design, 46
  read.inp, 67
  rev.capthist, 68
  squeeze, 73
  stratify, 75
* models
  AIC.openCR, 5
  classMembership, 8
  derived, 12
  makeNewData, 33
  modelAverage, 39
  strata, 74
* model
  cloned.fit, 9
  openCR.fit, 48
  openCRlist, 54
  par.openCR.fit, 55
* package
  openCR-package, 3

* print
  print.openCR, 64
  [.openCRlist(openCRlist), 54
  age.matrix, 4, 51
  AIC, 7
  AIC.openCR, 5, 40, 41, 55, 63, 66, 71
  AIC.openCRlist (AIC.openCR), 5
  bd.array (JS.counts), 24
  capthist, 4
  classMembership, 8, 53
  classMembership.openCR, 53
  cloned.fit, 9
  contour, 28
  cumMove, 10
  cyclic.fit (Internal), 22
  derived, 12
  derived.openCR, 53, 63, 64
  detectfn, 50
  detector, 49
  dipperCH, 14
  dkernel, 30, 32, 43
  dkernel (pkernel), 57
  expected.d, 16, 35
  extractFocal (moving.fit), 43
  FebpossumCH (PPNpossums), 61
  Field vole, 17
  fieldvoleCH (Field vole), 17
  gkernel, 43
  gkernel (pkernel), 57
  gonodontisCH, 20
  Internal, 22
  intervals, 25
  join, 25, 75, 76
INDEX

<table>
<thead>
<tr>
<th>Item</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>.m.array, 6, 77</td>
<td></td>
</tr>
<tr>
<td>.m.array(.JSCOUNTS), 24</td>
<td></td>
</tr>
<tr>
<td>make.kernel(.JSCOUNTS), 11, 16, 17, 29, 35, 43, 58</td>
<td></td>
</tr>
<tr>
<td>make.table, 32, 41, 55, 56</td>
<td></td>
</tr>
<tr>
<td>makeNewData, 13, 33, 63</td>
<td></td>
</tr>
<tr>
<td>mask, 32, 48</td>
<td></td>
</tr>
<tr>
<td>matchescale, 30, 32, 34, 58</td>
<td></td>
</tr>
<tr>
<td>Microtus, 35</td>
<td></td>
</tr>
<tr>
<td>microtusCH(Microtus), 35</td>
<td></td>
</tr>
<tr>
<td>microtusFCH(Microtus), 35</td>
<td></td>
</tr>
<tr>
<td>microtusFMCH(Microtus), 35</td>
<td></td>
</tr>
<tr>
<td>microtusMCH(Microtus), 35</td>
<td></td>
</tr>
<tr>
<td>microtusRDCH(Microtus), 35</td>
<td></td>
</tr>
<tr>
<td>miscellaneous, 38</td>
<td></td>
</tr>
<tr>
<td>model.matrix, 46, 51</td>
<td></td>
</tr>
<tr>
<td>modelAverage, 6, 39</td>
<td></td>
</tr>
<tr>
<td>Movement models, 16, 17, 30, 32, 34, 35, 41, 50, 57, 58</td>
<td></td>
</tr>
<tr>
<td>moving.fit, 43</td>
<td></td>
</tr>
<tr>
<td>MS.capthist, 76</td>
<td></td>
</tr>
<tr>
<td>nlm, 51, 71</td>
<td></td>
</tr>
<tr>
<td>openCR(openCR-Package), 3</td>
<td></td>
</tr>
<tr>
<td>openCR-deprecated, 45</td>
<td></td>
</tr>
<tr>
<td>openCR-package, 3</td>
<td></td>
</tr>
<tr>
<td>openCR.design, 5, 46, 52, 53</td>
<td></td>
</tr>
<tr>
<td>openCR.esa (derived), 12</td>
<td></td>
</tr>
<tr>
<td>openCR.fit, 3, 4, 6–8, 10, 14, 23, 24, 34, 41, 43, 44, 46, 47, 48, 53, 55, 56, 63, 66, 70, 73, 75, 76</td>
<td></td>
</tr>
<tr>
<td>openCR.pdot (derived), 12</td>
<td></td>
</tr>
<tr>
<td>openCRlist, 33, 41, 54, 56</td>
<td></td>
</tr>
<tr>
<td>optim, 51, 71</td>
<td></td>
</tr>
<tr>
<td>ovenCH, 4</td>
<td></td>
</tr>
<tr>
<td>par.openCR.fit, 33, 53, 55</td>
<td></td>
</tr>
<tr>
<td>Parallel, 56</td>
<td></td>
</tr>
<tr>
<td>PCH1(openCR-Package), 22</td>
<td></td>
</tr>
<tr>
<td>PCH1secr(openCR-Package), 22</td>
<td></td>
</tr>
<tr>
<td>pkernel, 17, 30, 32, 35, 43, 57</td>
<td></td>
</tr>
<tr>
<td>plot, 59, 60</td>
<td></td>
</tr>
<tr>
<td>plot.derivedopenCR, 58, 60</td>
<td></td>
</tr>
<tr>
<td>plot.kernel(make.kernel), 29</td>
<td></td>
</tr>
<tr>
<td>plot.mask, 30</td>
<td></td>
</tr>
<tr>
<td>plot.openCR, 59, 59</td>
<td></td>
</tr>
<tr>
<td>points, 59, 60</td>
<td></td>
</tr>
<tr>
<td>pointsInPolygon, 11</td>
<td></td>
</tr>
<tr>
<td>PPNpossums, 61</td>
<td></td>
</tr>
<tr>
<td>pradelloglik(openCR-Package), 22</td>
<td></td>
</tr>
<tr>
<td>predict, 60</td>
<td></td>
</tr>
<tr>
<td>predict.openCR, 30, 40, 53, 55, 62</td>
<td></td>
</tr>
<tr>
<td>predict.openCRlist, 32</td>
<td></td>
</tr>
<tr>
<td>predict.openCRlist(predict.openCR), 62</td>
<td></td>
</tr>
<tr>
<td>primarysessions(miscellaneous), 38</td>
<td></td>
</tr>
<tr>
<td>print.data.frame, 64</td>
<td></td>
</tr>
<tr>
<td>print.derivedopenCR, 13, 14, 63</td>
<td></td>
</tr>
<tr>
<td>print.openCR, 7, 64</td>
<td></td>
</tr>
<tr>
<td>proportionInPolygon(cumMove), 10</td>
<td></td>
</tr>
<tr>
<td>prwi(openCR-Package), 22</td>
<td></td>
</tr>
<tr>
<td>prwisecr(openCR-Package), 22</td>
<td></td>
</tr>
<tr>
<td>qkernel, 17, 30, 32, 43</td>
<td></td>
</tr>
<tr>
<td>qkernel(pkernel), 57</td>
<td></td>
</tr>
<tr>
<td>random numbers, 70</td>
<td></td>
</tr>
<tr>
<td>read.inp, 15, 67</td>
<td></td>
</tr>
<tr>
<td>rev.capthist, 68</td>
<td></td>
</tr>
<tr>
<td>RMarkInput, 68</td>
<td></td>
</tr>
<tr>
<td>runsim.nonspatial(simulation), 69</td>
<td></td>
</tr>
<tr>
<td>runsim.RMark(simulation), 69</td>
<td></td>
</tr>
<tr>
<td>runsim.spatial(simulation), 69</td>
<td></td>
</tr>
<tr>
<td>secondariesessions(miscellaneous), 38</td>
<td></td>
</tr>
<tr>
<td>secr.fit, 23, 48, 53</td>
<td></td>
</tr>
<tr>
<td>segments, 59, 60</td>
<td></td>
</tr>
<tr>
<td>session, 75</td>
<td></td>
</tr>
<tr>
<td>setNumThreads, 50</td>
<td></td>
</tr>
<tr>
<td>sim.capthist, 70, 71</td>
<td></td>
</tr>
<tr>
<td>sim.nonspatial, 70</td>
<td></td>
</tr>
<tr>
<td>sim.nonspatial(simulation), 69</td>
<td></td>
</tr>
<tr>
<td>sim.popn, 70, 71</td>
<td></td>
</tr>
<tr>
<td>simulation, 69</td>
<td></td>
</tr>
<tr>
<td>split.capthist, 76</td>
<td></td>
</tr>
<tr>
<td>squeeze, 8, 51, 73</td>
<td></td>
</tr>
<tr>
<td>strata, 74, 76</td>
<td></td>
</tr>
<tr>
<td>strata&lt;-strata, 74</td>
<td></td>
</tr>
<tr>
<td>stratify, 74, 75, 75</td>
<td></td>
</tr>
</tbody>
</table>
summary.kernel (make.kernel), 29
summary.openCR, 53
summary.openCR (print.openCR), 64
sumsims (simulation), 69

turnover, 70

care.cjs, 76
unirroot, 57
unRMarkInput, 68
unsqueeze (squeeze), 73