

Package ‘secrdesign’

March 10, 2023

Type Package

Title Sampling Design for Spatially Explicit Capture-Recapture

Version 2.8.2

Date 2023-03-11

Description Tools for designing spatially explicit capture-recapture studies of animal populations. This is primarily a simulation manager for package 'secr'. Extensions in version 2.5.0 include costing and evaluation of detector spacing.

Depends R (>= 3.5.0), secr (>= 4.2.0)

Imports abind, kofnGA, parallel, sf, Rcpp (>= 0.12.14)

LinkingTo BH, Rcpp, RcppArmadillo

Suggests secrlinear, ipsecc (>= 1.4.0), testthat (>= 0.11.0)

License GPL (>= 2)

URL <https://www.otago.ac.nz/density/>,
<https://github.com/MurrayEfford/secrdesign/>

NeedsCompilation yes

Author Murray Efford [aut, cre] (<<https://orcid.org/0000-0001-5231-5184>>),
Ian Durbach [ctb] (<<https://orcid.org/0000-0003-0769-2153>>)

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

Repository CRAN

Date/Publication 2023-03-10 19:30:02 UTC

R topics documented:

secrdesign-package	2
costing	4
count	6
estimateSummary	7
GAoptim	11
getdetectpar	14
Internal	15

Lambda	16
make.array	19
make.scenarios	20
optimalSpacing	22
plot.optimalSpacing	24
predict.fittedmodels	25
rbind.estimatetables	27
run.scenarios	28
saturation	35
scenariosFromStatistics	36
scenarioSummary	38
select.stats	40
summary.secrdesign	41
validate	44

Index	46
--------------	-----------

secrdesign-package *Spatially Explicit Capture–Recapture Study Design*

Description

Tools to assist the design of spatially explicit capture–recapture studies of animal populations.

Details

Package: secr
 Type: Package
 Version: 2.8.2
 Date: 2023-03-11
 License: GNU General Public License Version 2 or later

The primary use of **secrdesign** is to predict by Monte Carlo simulation the precision or bias of density estimates from different detector layouts, given pilot values for density and the detection parameters λ_0/g_0 and σ .

Tools are also provided for predicting the performance of detector layouts without simulation, and for optimising layouts to meet various criteria, particularly expected counts.

The simulation functions in **secrdesign** are:

<code>make.scenarios</code>	generate dataframe of parameter values etc.
<code>run.scenarios</code>	perform simulations, with or without model fitting
<code>fit.models</code>	fit SECR model(s) to rawdata output from <code>run.scenarios</code>
<code>predict.fittedmodels</code>	infer ‘real’ parameter estimates from fitted models
<code>select.stats</code>	collect output for a particular parameter
<code>summary.selectedstatistics</code>	numerical summary of results

`plot.selectedstatistics` histogram or CI plot for each scenario

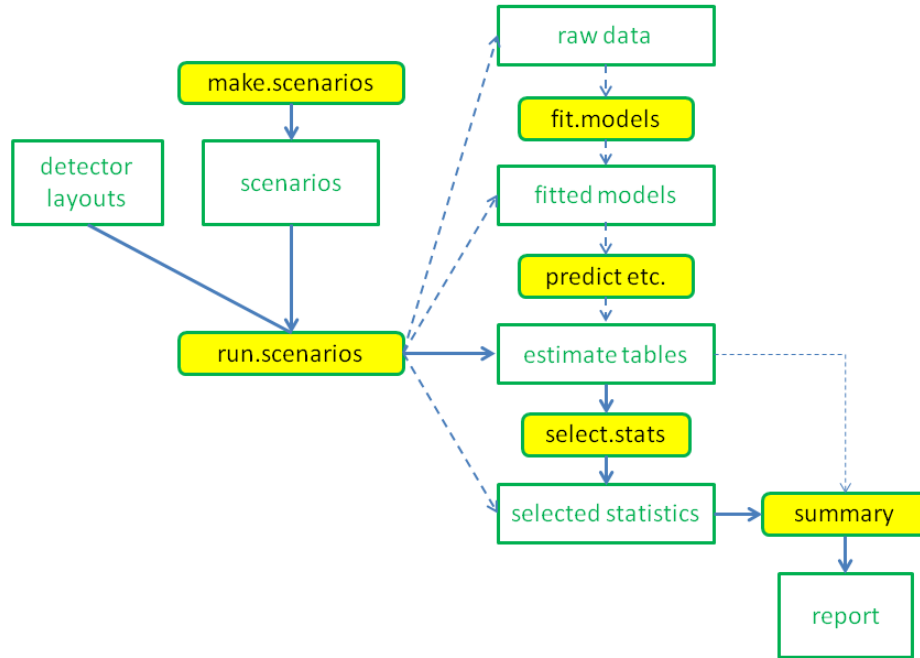


Fig. Core simulation functions in **secrdesign** (yellow) and their main inputs and outputs. Output from the simulation function `run.scenarios()` may be saved as whole fitted models, predicted values (parameter estimates), or selected statistics. Each form of output requires different subsequent handling. The default path is shown by solid blue arrows.

Other functions not used exclusively for simulation are:

<code>Enrm</code>	expected numbers of individuals n , re-detections r and movements m
<code>En2</code>	expected number of individuals detected at two or more detectors
<code>minnrRSE</code>	approximate RSE(D-hat) given sample size (n, r) (Efford and Boulanger 2019)
<code>GAoptim</code>	optimization of detector placement using genetic algorithm (Durbach et al. 2021)
<code>costing</code>	various cost components
<code>saturation</code>	expected detector saturation (trap success)
<code>scenarioSummary</code>	applies <code>Enrm</code> , <code>minnrRSE</code> , and other summaries to each scenario in a dataframe
<code>optimalSpacing</code>	optimal detector spacing by rule-of-thumb and simulation RSE(D-hat)

A vignette documenting the simulation functions is available at [secrdesign-vignette.pdf](#). An Appendix in that vignette has code for various examples that should help get you started.

Documentation for expected counts is in [secrdesign-Enrm.pdf](#). Another vignette [secrdesign-tools.pdf](#) demonstrates other tools. These include the `optimalSpacing` function, for finding the detector spacing that yields the greatest precision for a given detector geometry, number of sampling occasions, density and detection parameters.

Help pages are also available as [../doc/secrdesign-manual.pdf](#).

Author(s)

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

References

Durbach, I., Borchers, D., Sutherland, C. and Sharma, K. (2021) Fast, flexible alternatives to regular grid designs for spatial capture–recapture. *Methods in Ecology and Evolution* **12**, 298–310. DOI 10.1111/2041-210X.13517

Efford, M. G., and Boulanger, J. (2019) Fast evaluation of study designs for spatially explicit capture–recapture. *Methods in Ecology and Evolution*, **10**, 1529–1535. DOI: 10.1111/2041-210X.13239

See Also

[make.grid](#), [sim.popn](#), [sim.caphist](#), [secr.fit](#)

costing

Cost of SECR design

Description

The cost of implementing a spatially explicit capture–recapture design depends on the detector layout, the number of detections and the various unit costs.

Usage

```
costing(traps, nr, noccasions, unitcost = list(), nrepeats = 1, routelength = NULL,
        setupoccasion = TRUE)
```

Arguments

<code>traps</code>	traps object for detector array
<code>nr</code>	numeric vector with $E(n)$ and $E(r)$ as first two elements
<code>noccasions</code>	integer number of sampling occasions
<code>unitcost</code>	list with unit costs (see Details)
<code>nrepeats</code>	integer number of repeated arrays
<code>routelength</code>	numeric route length (km)
<code>setupoccasion</code>	logical; if TRUE then the cost of a setup visit is included (<code>noccasions+1</code>)

Details

`nr` is a vector with the expected sample sizes (numbers of individuals and recaptures), usually the output from [Enrm](#).

`unitcost` should be a list with at least one of the components ‘perkm’, ‘perarray’, ‘perdetector’, ‘pervisit’ and ‘perdetection’.

The number of occasions (`noccasions`) is incremented by 1 if `setupoccasion` is TRUE.

Component	Unit cost	Costing
Arrays	perarray	perarray x nrepeats
Detectors	perdetector	perdetector x nrow(traps) x nrepeats
Travel	perkm	perkm x routelength x noccasions x nrepeats
Visits	pervisit	sum(pervisit x trapcost) x noccasions x nrepeats
Detections	perdetection	perdetection x total detections ($E(n) + E(r)$)

‘Travel’ and ‘Visits’ are alternative ways to cost field time. The variable ‘routelength’ represents the length of a path followed to visit all detectors; if not specified it is approximated by the sum of the nearest-trap distances. The variable ‘trapcost’ is a vector of length equal to the number of detectors. By default it is a vector of 1’s, but detector- specific values may be provided as trap covariate ‘costpervisit’. In the latter case the value of ‘pervisit’ should probably be 1.0.

‘Arrays’ and ‘Detectors’ represent one-off costs.

‘Detections’ includes costs such as handling time and laboratory DNA analysis.

See [../doc/secrdesign-tools.pdf](#) for more.

Value

A named numeric vector

See Also

[Enrm](#), [scenarioSummary](#)

Examples

```
tr <- make.grid(8, 8, spacing = 25)
msk <- make.mask(tr, buffer = 100, type = 'trapbuffer')
nrm <- Enrm(D = 5, tr, msk, list(lambda0 = 0.2, sigma = 20), 5)
costing (tr, nrm, 5, unitcost = list(pervisit = 5, perdetection = 15))
```

count

*Extract Summaries***Description**

Reshape results from `run.scenarios(..., extractfn = summary)` so that they may be passed to the usual summary functions of **secrdesign**.

Usage

```
count(object, ...)

## S3 method for class 'summary'
predict(object, ...)
## S3 method for class 'summary'
coef(object, ...)
## S3 method for class 'summary'
count(object, ...)
```

Arguments

`object` summary simulation output from `run.scenarios`
`...` other arguments (not used)

Details

The aim is to extract numerical results from simulations performed using `run.scenarios(..., extractfn = summary)`. The results may then be passed to the summary method for ‘secrdesign’ objects, possibly via `select.stats` (see Examples).

Value

An object of class `c("estimatetables", "secrdesign", "list")` in which the output component for each scenario is a list of dataframes, one per replicate. The structure of each dataframe is indicated in the following table (parameters may vary with model); ‘parameters’ and ‘statistics’ correspond to arguments of `select.stats`.

Function	Row(s) (parameters)	Columns (statistics)
<code>count</code>	Number	Animals, Detections, Moves
<code>coef</code>	D, g0, sigma	estimate, SE.estimate, lcl, ucl
<code>predict</code>	D, g0, sigma	estimate, SE.estimate, lcl, ucl

See Also

[predict.secr](#), [coef.secr](#),

Examples

```
## generate some simulations
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid(6, 6, spacing = 25)
sims1 <- run.scenarios(nrepl = 2, trapset = traps1, scenarios =
  scen1, seed = 345, fit = TRUE, extractfn = summary)

## view the results
count(sims1)$output
predict(sims1)$output

summary(sims1) ## header only

summary(count(sims1)) # equivalent to following
summary(select.stats(count(sims1), parameter = 'Number'))

summary(predict(sims1)) # default select.stats parameter = 'D'
summary(select.stats(predict(sims1), parameter = 'sigma') )
```

estimateSummary

Direct summary of estimate tables

Description

An alternative approach to summarising output from `run.scenarios` (cf [summary.estimateTables](#)). `estimateSummary` is especially useful when `extractfn = predict` or `extractfn = coef`, and all scenarios have group structure.

Usage

```
estimateArray(object)
```

```
estimateSummary(object, parameter = "D", statistics = c("true", "nvalid",
  "EST", "seEST", "RB", "seRB", "RSE", "RMSE", "rRMSE", "COV"), true,
  validrange = c(0, Inf), checkfields = c('estimate', 'SE.estimate'),
  format = c('list', 'data.frame'), cols = NULL)
```

Arguments

object	secrdesign object of class "estimatetables"
parameter	character name of parameter (row in estimate table)
statistics	character choice of outputs
true	numeric vector of true values, one per scenario and group
validrange	numeric allowed for estimates or other checkfields
checkfields	character choice of columns in each estimate table that will be checked against validrange
format	character choice of output
cols	indices of scenario columns to include when format = "data.frame"

Details

When 'predict(fittedmodel)' in `run.scenarios` generates more than one estimate table (i.e. when the model uses groups, mixture classes or multiple sessions), the default extract function retains only the first. This is often OK, but it can be frustrating if you care about group- or session-specific estimates.

The alternative is to use 'predict' as the `run.scenarios` extractfn, which retains all estimate tables. This requires a different function for summarisation; `estimateSummary` will suffice for many purposes.

`estimateSummary` internally calls `estimateArray` to pre-process the output from `run.scenarios`.

The code should be examined for the precise definition of each statistic.

True parameter values are required for RB, RMSE and COV, and these are computed even if later dropped from the output. If provided, the argument `true` should have length equal to the number of parameter tables in each replicate, i.e. (number of scenarios) * (number of groups), ordered by scenario. Otherwise, true values will be taken from rows of the data frame `object$scenarios`.

Replicates are rejected (set to NA) if any `checkfields` falls outside `validrange`.

Output statistics 'EST', 'RB', and 'RSE' are means across replicates, and 'seEST', 'seRB' the corresponding standard errors.

The output list may optionally be formatted as a `data.frame` with pre-pended columns from `object$scenarios`. Set `cols` to 0 or NULL for no scenario columns.

`cols` defaults to `c("scenario", "group")` if groups are present and "scenario" otherwise.

Value

`estimateArray` – array with dimensions (Parameter, statistic, Group, Scenario, Replicate)

`estimateSummary` –

If groups present and `format = "list"` - a list of matrices (group x scenario), one for each statistic:

<code>true.X</code>	true value of parameter (X)
<code>nvalid</code>	number of valid replicates used in later summaries
<code>EST</code>	mean of parameter estimates

seEST	standard error of estimates (across replicates)
RB	relative bias
seRB	standard error of replicate-specific RB (across replicates)
RSE	relative standard error (SE.estimate/estimate)
RMSE	root mean squared error
rRMSE	RMSE/true.X
COV	coverage of confidence intervals (usually 95% intervals).

If groups absent and format = "list" - a list of vectors (one element per scenario) with statistics as above.

If format = "data.frame" - a data frame with rows corresponding to group x scenario (or session x scenario) combinations and columns corresponding to statistics as above.

Note

These functions were introduced in version 2.8.1. They may change in later versions.

Results may be confusing when scenarios have group structure and groups are not used in the fitted model.

See Also

[run.scenarios](#), [header](#), [summary.estimateables](#)

Examples

```
# 2-scenario, 2-group simulation
scen8 <- make.scenarios (D = 8, g0 = 0.3, sigma = 30,
  noccasions = c(4,8), groups = c('F','M'))

# replace density and sigma values of males to make it interesting
male <- scen8$group == 'M'
scen8$D[male] <- 4
scen8$sigma[male] <- 40

grid <- make.grid(8, 8, spacing = 30)
mask <- make.mask(grid, buffer = 160, type = 'trapbuffer')

old <- options(digits = 3)
setNumThreads(2)

#-----
# run a few simulations

# model groups
sims <- run.scenarios(10, scen8, trapset = grid, fit = TRUE,
  fit.args = list(model = list(D~g, g0~1, sigma~g), groups = 'group'),
```

```

    extractfn = predict, maskset = mask)

# format as list, selecting statistics
# default summary uses true = c(8,4,8,4)
estimateSummary(sims, 'D', c("true", "nvalid", "EST", "RB", "seRB"))

# format as data.frame by scenario and group, all statistics
estimateSummary(sims, 'D', format = 'data.frame')

#-----
# try with default extractfn (single table per replicate, despite groups)
sims2 <- run.scenarios(10, scen8, trapset = grid, fit = TRUE,
    maskset = mask)

# Fails with "Error in estimateSummary(sims2, "D") : incongruent 'true'"
# estimateSummary(sims2, 'D')

# OK if manually provide scenario-specific true density
estimateSummary(sims2, 'D', true = c(12,12))

# reformat by scenario
estimateSummary(sims2, 'D', true = c(12,12), format = 'data.frame')

# compare standard summary
summary(sims2)$OUTPUT

#-----

# multiple estimate tables also arise from multi-session simulations
# argument 'true' must be specified manually
# interpret with care: sessions are (probably) not independent
# this example uses the previous grid and mask

scen9 <- make.scenarios (D = 8, g0 = 0.3, sigma = 30, noccasions = 5)
poparg <- list(nsessions = 3, details = list(lambda = 1.2)) # for sim.popn
detarg <- list(renumber = FALSE) # for sim.caphist
fitarg <- list(model = D~Session) # for secr.fit

sims3 <- run.scenarios(5, scen9, trapset = grid, fit = TRUE,
    maskset = mask, pop.args = poparg, det.args = detarg,
    fit.args = fitarg, extractfn = predict)

estimateSummary(sims3, parameter = 'D', format = 'data.frame',
    true = 8 * 1.2^(0:2))
#-----

# extractfn = coef results in a single estimate table per replicate,
# so the usual summary method is sufficient. For completeness we show
# that estimateSummary can also be used. Coefficients are often negative,
# so relative values (e.g., RB, RSE) may be meaningless.

sims4 <- run.scenarios(5, scen9, trapset = grid, fit = TRUE,
    maskset = mask, pop.args = poparg, det.args = detarg,

```

```

fit.args = fitarg, extractfn = coef)

estimateSummary(sims4, parameter = 'D', c("nvalid", "EST", "seEST", "RMSE", "COV"),
  format = 'data.frame', true = log(8), checkfields = 'beta',
  validrange = log(c(2,20)))

estimateSummary(sims4, parameter = 'D.Session', c("nvalid", "EST", "seEST",
  "RMSE", "COV"), format = "data.frame", true = log(1.2), checkfields = "beta",
  validrange = log(c(0.5,2)))
#-----

options(old)

```

GAoptim

SECR detector placement by maximisation of a simple criterion

Description

Implements the approach of Durbach et al. (2021) for optimization of detector placement using a genetic algorithm to maximize the lesser of $E(n)$ and $E(r)$, where n is the number of distinct individuals and r is the total number of recaptures. This criterion predicts the relative standard error of the density estimate (Efford and Boulanger 2019).

Users may choose the criterion to be maximised. The number of individuals detected at two or more detectors is an alternative of particular interest (Dupont et al. 2021).

Usage

```

GAoptim(mask, alltraps, ntraps, detectpar, noccasions,
  detectfn = c("HHN", "HHR", "HEX", "HAN", "HCG"), D = NULL,
  criterion = 4, penalty = NULL, seed = NULL, ...)

```

Arguments

mask	mask object
alltraps	traps object with all possible trap locations
ntraps	number of required trap locations
detectpar	list values of detection parameters λ_0 , σ etc.
detectfn	integer code or character string for shape of detection function - see detectfn
noccasions	integer number of sampling occasions
D	numeric density animals per hectare (0.01 km^2)
criterion	integer code for criterion to maximise, or function (see Details)

penalty	list defining penalty for layout in relation to reference grid (optional)
seed	set a random seed for reproducibility of GA
...	other arguments passed to kofnGA

Details

detectpar is a named list with values of the detection parameters for the chosen detectfn. Usually this will be just lambda0 (baseline hazard of detection) and sigma (spatial scale of detection).

The genetic algorithm is provided by function kofnGA from package **kofnGA** (Wolters 2015). The first three arguments of kofnGA (i.e., n, k, OF) are set by GAoptim. Others may be adjusted by the user via the ... argument. Specifically,

Argument	Default	Description
ngen	500	number of generations to run
popsize	200	size of the population; equivalently, the number of offspring produced each generation
mutprob	0.01	mutation rate
verbose	0	integer controlling the display of progress during search. If a positive value, then the iteration number a
cluster	NULL	number of parallel cores or a prebuilt parallel cluster

The default for ngen may (or may not) be larger than is needed for routine use. Durbach et al. (2021) used ngen = 50, popsize = 1000 and mutprob = 0.01.

Density D may be a scalar or a vector of length equal to the number of mask cells. No value need be specified if the sole aim is to optimize trap placement, but D is required for predictions of $E(n)$ and $E(r)$.

Pathological detector layouts (sensu Efford and Boulanger 2019) may be avoided by adding a penalty to the objective. No penalty is applied by default. To apply a penalty, penalty should be a list with named components pen_wt>0 and pen_gridsigma). If a penalty is applied, the default compares the number of trap pairs with close spacing (2.5-3.5 sigma, 3.5-4.5 sigma) to the number in a compact sample from a regular grid with spacing sigma * pen_gridsigma (see internal functions [GApfn](#) and [compactSample](#) and the vignette). An alternative penalty function may be supplied as component 'pen_fn' of penalty.

The default criterion is the minimum of $E(n)$ and $E(r)$ as used by Durbach et al. (2021). The full list of builtin possibilities is:

Code	Description	Note
1	$E(n)$	number of distinct individuals
2	$E(r)$	number of recaptures
3	$E(m)$	number of movement recaptures
4	$\min(E(n), E(r))$	minimum $E(n)$, $E(r)$
5	$E(n2)$	expected number of animals detected at 2 or more sites (cf Qpm Dupont et al. 2021)
6	$E(n) + E(n2)$	(1) + (5) (cf Qpb Dupont et al. 2021)

Criteria 1–4 are computed with function [Enrm](#) (see also Efford and Boulanger 2019). Criteria 5–6 are computed with function [En2](#). Any penalty is applied only when criterion = 4.

The criterion may also be a function that returns a single numeric value to be maximised. Arguments of the function should match those of [En2](#), although ... may suffice for some or all (see Examples).

Value

An object of class "GAoptim" that is a list with components

mask	saved input
alltraps	saved input
detectpar	saved input
noccasions	saved input
detectfn	saved input
D	saved input
penalty	saved input
criterion	saved input
des	kofnGA() output object
optimaltraps	traps object with optimized layout
optimalenrms	E(n), E(r), E(m) evaluated with optimized layout

Warnings

Spatial representativeness is not considered, so designs 'optimised' with GAoptim are not robust to unmodelled variation in density or detection parameters.

Author(s)

Ian Durbach and Murray Efford.

References

- Dupont, G., Royle, J. A., Nawaz, M. A. and Sutherland, C. (2021) Optimal sampling design for spatial capture–recapture. *Ecology* **102** e03262.
- Durbach, I., Borchers, D., Sutherland, C. and Sharma, K. (2021) Fast, flexible alternatives to regular grid designs for spatial capture–recapture. *Methods in Ecology and Evolution* **12**, 298–310. DOI 10.1111/2041-210X.13517
- Efford, M. G., and Boulanger, J. (2019) Fast evaluation of study designs for spatially explicit capture–recapture. *Methods in Ecology and Evolution*, **10**, 1529–1535. DOI: 10.1111/2041-210X.13239
- Wolters, M. A. (2015) A genetic algorithm for selection of fixed-size subsets with application to design problems. *Journal of Statistical Software, Code Snippets*, **68**, 1–18. DOI 10.18637/jss.v068.c01

See Also

[Enrm](#), [En2](#), [minnrRSE](#), [GApnfn](#), [compactSample](#)

Examples

```
# an artificial example
msk <- make.mask(type = 'rectangular', spacing = 10, nx = 30, ny = 20, buffer = 0)
alltrps <- make.grid(nx = 29, ny = 19, origin = c(10,10), spacing = 10)
set.seed(123)

# 50 generations for demonstration, use more in practice
opt <- GAOptim(msk, alltrps, ntraps = 20, detectpar = list(lambda0 = 0.5, sigma = 20),
  detectfn = 'HHN', D = 10, noccasions = 5, ngen = 50, verbose = 1)

plot(msk)
plot(opt$optimaltraps, add = TRUE)
minnrRSE(opt, distribution = 'binomial')

# Using a criterion function
# En2 is unsuitable as a criterion function as it returns 2 values
# This function selects the second as the (unique) criterion
fn <- function(...) En2(...)[2]
opt2 <- GAOptim(msk, alltrps, ntraps = 20, detectpar = list(lambda0 = 0.5, sigma = 20),
  detectfn = 'HHN', D = 10, noccasions = 5, ngen = 50, verbose = 1, criterion = fn)
```

getdetectpar

Ballpark Detection Parameters

Description

Detection parameters for an animal population may be guessed from some basic inputs (population density, a coefficient of home-range overlap, and the expected number of detections on a given detector array). These values are useful as a starting point for study design. They are not 'estimates'.

Usage

```
getdetectpar(D, C, sigma = NULL, k = 0.5, ...)
```

Arguments

D	population density animals / hectare; may be scalar or vector of length nrow(mask)
C	integer expected total number of detections
sigma	numeric spatial scale parameter of chosen detection function, in metres (optional)

k coefficient of overlap - typically in range 0.3 to 1.1
 ... named arguments passed to [Enrm](#) and [Lambda](#) (traps, mask, noccasions, detectfn)

Details

If sigma is missing and detectfn = 'HHN' then sigma is first inferred from the relationship $\sigma = 100k\sqrt{D}$ (D in animals per hectare and σ in metres). Other detectfn give an error.

A numerical search is then conducted for the value of lambda0 that results in C expected detections for the given density and design. The calculation takes account of the detector array, the habitat mask and the number of sampling occasions (all specified in the ... argument - see example).

Only hazard detection functions are supported ('HHN', 'HHR', 'HEX', 'HAN', 'HCG'). The default is 'HHN'.

Value

A list with one component for each detection parameter.

See Also

[Enrm](#), [Lambda](#)

Examples

```
tr <- traps(captdata)
detector(tr) <- "multi"
msk <- make.mask(tr, buffer = 100, type = 'trapbuffer')
getdetectpar(D = 5.48, C = 235, traps = tr, mask = msk, noccasions = 5)
```

Internal

Internal Functions

Description

Functions that are called internally by **secrdesign**. These are exported and may be called separately for testing.

Usage

```
compactSample (traps, n)
```

```
GOpenfn(traps, sigma)
```

Arguments

traps secr trapsobject
n integer number in sample ($0 < n \leq ntraps$)
sigma numeric sparial scale parameter

Details

compactSample selects a detector at random and returns the a compact subset of surrounding detectors.

GApennfn is the default pen_fn used by [GAoptim](#) When called with a non-null penalty argument.

Value

GApennfn – a numeric vector with the number of trap pairs separated by 2.5-3.5 sigma and 3.5-4.5 sigma.

compactSample – an object like traps, but with only n rows.

References

Durbach, I., Borchers, D., Sutherland, C. and Sharma, K. (2021) Fast, flexible alternatives to regular grid designs for spatial capture–recapture. *Methods in Ecology and Evolution* **12**, 298–310. DOI 10.1111/2041-210X.13517

See Also

[GAoptim](#),

Examples

```
CStraps <- compactSample(traps(captdata), n = 20)

plot(traps(captdata))
plot(CStraps, add = TRUE, detpar = list(fg = 'blue', pch = 16))

GApennfn(CStraps, sigma = 25)
```

Lambda

Expected Detections

Description

Compute the expected number of detections as a function of location (Lambda), and the expected total numbers of individuals n , recaptures r and movements m for a population sampled with an array of detectors (Enm) or the number of individuals detected at two or more detectors (En2).

Usage

```
Lambda(traps, mask, detectpar, noccasions, detectfn = c("HHN", "HHR", "HEX",
  "HAN", "HCG", 'HN', 'HR', 'EX'))
```

```
Enrm(D, ...)
```

```
minnrRSE(D, ..., CF = 1.0, distribution = c("poisson", "binomial"))
```

```
En2(D, traps, mask, detectpar, noccasions, detectfn = c("HHN", "HHR", "HEX",
  "HAN", "HCG", "HN", "HR", "EX"))
```

```
Qpm(D, traps, mask, detectpar, noccasions, detectfn = c("HHN", "HHR", "HEX",
  "HAN", "HCG", "HN", "HR", "EX"))
```

Arguments

traps	traps object
mask	mask object
detectpar	a named list giving a value for each parameter of detection function
noccasions	integer number of sampling occasions
detectfn	integer code or character string for shape of detection function – see detectfn
D	population density animals / hectare; may be scalar or vector of length nrow(mask)
...	arguments passed to Lambda
CF	numeric correction factor
distribution	character distribution of n

Details

The detector attribute of traps may be ‘multi’, ‘proximity’ or ‘count’. It is assumed that detectpar and detector type do not differ among occasions.

The calculation is based on an additive hazard model. If detectfn is not a hazard function (‘HHN’, ‘HEX’, ‘HHR’, ‘HAN’ and ‘HCG’) then an attempt is made to approximate one of the hazard functions (HN -> HHN, HR -> HHR, EX -> HEX). The default is ‘HHN’.

For hazard function $\lambda(d)$ and S occasions, we define $\Lambda(x) = \sum_s \sum_k \lambda(d_k(x))$.

Formulae for expected counts are given in [secrdesign-Enrm.pdf](#).

minnrRSE has mostly the same inputs as Enrm but returns $\sqrt{CF/\min(n,r)}$. The correction factor CF may be used to adjust for systematic bias (e.g., for a line of detectors CF = 1.4 may be appropriate). The default distribution = ‘poisson’ is for Poisson-distributed N and n . To adjust the prediction for fixed N (binomial n) use distribution = ‘binomial’ (see [../doc/secrdesign-tools.pdf](#) Appendix 2).

From 2.7.0, the first argument of minnrRSE may also be the output from [GAoptim](#).

En2 is defined for detectors ‘multi’, ‘proximity’ and ‘count’.

Qpm returns the optimisation criteria Q_p and Q_{p_m} of Dupont et al. (2021), defined only for ‘proximity’ and ‘count’ detectors. The criteria are mask-dependent, and En2 is generally preferred. For ‘proximity’ and ‘count’ detectors the following expressions give the same result:

En2(D, trp, msk, dp)

Qpm(D, trp, msk, dp) * maskarea(msk) * D

given constant density ‘D’, detectors ‘trp’, mask ‘msk’ and detection parameters ‘dp’.

Value

Lambda – [mask](#) object with covariates ‘Lambda’ ($\Lambda(x)$), ‘sumpk’ and ‘sumq2’ (intermediate values for computation of expected counts - see [../doc/expectedcounts.pdf](#))

Enrm – numeric vector of length 3, the values of $E(n)$, $E(r)$ and $E(m)$

minnrRSE – rule-of-thumb RSE(D-hat) Efford and Boulanger (2019)

En2 – numeric vector comprising the values $E(n)$ and E (number of animals detected at 2 or more sites)

Qpm – numeric vector comprising the criteria Q_p and Q_{p_m} of Dupont et al. (2021)

References

Dupont, G., Royle, J. A., Nawaz, M. A. and Sutherland, C. (2021) Optimal sampling design for spatial capture–recapture. *Ecology* **102** e03262.

Efford, M. G., and Boulanger, J. (2019) Fast evaluation of study designs for spatially explicit capture–recapture. *Methods in Ecology and Evolution*, **10**, 1529–1535. DOI: 10.1111/2041-210X.13239

See Also

[getdetectpar](#), [optimalSpacing](#), [scenarioSummary](#), [GAoptim](#)

Examples

```
tr <- traps(captdata)
detector(tr) <- "multi"
msk <- make.mask(tr, buffer = 100, type = 'trapbuffer')

L <- Lambda(tr, msk, list(lambda0 = 0.2, sigma = 20), 5)
nrm <- Enrm(D = 5, tr, msk, list(lambda0 = 0.2, sigma = 20), 5)
nrm

En2(D = 5, tr, msk, list(lambda0 = 0.2, sigma = 20), 5)

plot(L, cov = "Lambda", dots = FALSE)
plot(tr, add = TRUE)
mtext(side = 3, paste(paste(names(nrm), round(nrm,1)), collapse = ", "))
```

make.array	<i>Re-cast Simulated Statistical Output as Array</i>
------------	--

Description

This function is used internally by [summary.secrdesign](#), and may occasionally be of general use.

Usage

```
make.array(object)
```

Arguments

object	secrdesign object containing numerical values for a particular parameter (i.e. output from select.stats inheriting from 'selectedstatistics')
--------	---

Details

`make.array` converts a particular simulated numerical output into an array with one dimension for each varying input.

Value

A numeric array with dimensions corresponding to the varying inputs.

See Also

[run.scenarios](#)

Examples

```
## collect raw counts
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid()
tmp1 <- run.scenarios(nrepl = 50, trapset = traps1, scenarios = scen1,
  fit = FALSE)
make.array(tmp1)
```

make.scenarios	<i>Construct Scenario Data Frame</i>
----------------	--------------------------------------

Description

This function prepares a dataframe in which each row specifies a simulation scenario. The dataframe is used as input to [run.scenarios](#).

Usage

```
make.scenarios(trapsindex = 1, noccasions = 3, nrepeats = 1, D, g0, sigma, lambda0,
detectfn = 0, recapfactor = 1, popindex = 1, detindex = 1, fitindex = 1, groups,
crosstraps = TRUE)
```

Arguments

trapsindex	integer vector determining the traps object to use
noccasions	integer vector for the number of sampling occasions
nrepeats	integer vector of multipliers for D (see Details)
D	numeric vector of values for the density parameter (animals / hectare)
g0	numeric vector of values for the g0 parameter
sigma	numeric vector of values for the sigma parameter (m)
lambda0	numeric vector of values for the lambda0 parameter
detectfn	vector of valid detection function codes (numeric or character)
recapfactor	numeric vector of values for recapfactor (sim.caphist)
popindex	integer vector determining which population model is used
detindex	integer vector determining which detection options are used
fitindex	integer vector determining which model is fitted
groups	character vector of group labels (optional)
crosstraps	logical; if TRUE the output includes all combinations of trapsindex, noccasions and nrepeats

Details

The index in trapsindex is used in [run.scenarios](#) to select particular detector arrays from the list of arrays provided as an argument to that function.

The function generates all combinations of the given parameter values using [expand.grid](#). By default, it also generates all combinations of the parameters with trapsindex and the number of sampling occasions. If crosstraps is FALSE then trapsindex, noccasions, and nrepeats are merely used to fill in these columns in the output dataframe.

The argument lambda0 replaces g0 for the hazard detection functions 14–18 ([detectfn](#)).

Designs may use multiple detector arrays with the same internal geometry (e.g., number and spacing of traps). The number of such arrays is varied with the `nrepeats` argument. For example, you may compare designs with many small arrays or a few large ones. In practice, `run.scenarios` simulates a single layout with density $D * nrepeats$. This shortcut is not appropriate when animals compete for traps (`detector = 'single'`).

`fitindex` allows a choice of different models when the argument `fit.args` of `run.scenarios` is a compound list.

If `groups` is provided each scenario is replicated to the length of `groups` and a column 'group' is added.

Value

Dataframe with one row per scenario (or sub-scenario) and the columns

<code>scenario</code>	a number identifying the scenario
<code>group</code>	(optional)
<code>trapsindex</code>	
<code>noccasions</code>	
<code>nrepeats</code>	
<code>D</code>	
<code>g0</code>	or <code>lambda0</code>
<code>sigma</code>	
<code>detectfn</code>	see <code>detectfn</code> ; always numeric
<code>recapfactor</code>	
<code>popindex</code>	
<code>detindex</code>	
<code>fitindex</code>	

An attribute 'inputs' is saved for possible use in `make.array`.

See Also

[run.scenarios](#), [scenarioSummary](#), [sim.caphist](#)

Examples

```
make.scenarios(trapsindex = 1, nrepeats = 1, D = c(5,10), sigma = 25,
g0 = 0.2)
```

optimalSpacing *Optimal Detector Spacing*

Description

Estimate the detector spacing that yields the greatest precision for a given detector geometry, number of sampling occasions, density and detection parameters.

Usage

```
optimalSpacing (D, traps, detectpar, noccasions, nrepeats = 1,
  detectfn = c('HHN', 'HHR', 'HEX', 'HAN', 'HCG', 'HN', 'HR', 'EX'),
  fittedmodel = NULL, xsigma = 4, R = seq(0.2, 4, 0.2), CF = 1.0,
  distribution = c("poisson", "binomial"),
  fit.function = c("none", "secr.fit"),
  simulationR = seq(0.4, 4, 0.4), nrepl = 10,
  plt = FALSE, ...)
```

Arguments

D	population density animals / hectare (constant)
traps	traps object
detectpar	named list giving a value for each parameter of detection function (sigma not needed)
noccasions	integer number of sampling occasions
nrepeats	integer number of replicate arrays (not yet used)
detectfn	integer code or character string for shape of detection function – see detectfn
fittedmodel	secr fitted model (instead of preceding arguments)
xsigma	numeric buffer width as multiple of sigma
R	numeric vector of relative spacings at which to plot rule-of-thumb RSE(D-hat)
CF	numeric correction factor for rule-of-thumb RSE
distribution	character distribution of number of individuals detected
fit.function	character function to use for model fitting
simulationR	numeric vector of relative spacings at which to simulate
nrepl	integer number of replicate simulations
plt	logical; if TRUE then results are plotted
...	other arguments passed to various functions (see Details)

Details

A numerical search over possible spacings uses the rule-of-thumb RSE(D-hat) given by `minnrRSE` as the objective function.

`traps` provides the geometry of the detector layout and the initial spacing s . Function `optimize` is used to search for a solution (minimum RSE) in the range of $R \times s$.

The computation emulates variation in detector spacing by inverse variation in sigma ($\text{sigma}' = \text{sigma} / R$) with compensating variation in density. Mask buffer width and spacing are also scaled by R .

If `fit.function` is not "none" then simulations are also performed for the relative spacings in `simulationR`. Density, sigma and mask attributes are scaled as for the rule-of-thumb calculations. Using `'method = "none"'` gives fast prediction of RSE (from the Hessian evaluated at the known parameter values), but does not estimate bias.

The `...` argument may be used to set the values of these arguments:

Function	Arguments
<code>make.mask</code>	<code>'nx'</code> , <code>'type'</code> , <code>'poly'</code> , <code>'poly.habitat'</code>
<code>run.scenarios</code>	<code>'seed'</code> , <code>'ncores'</code> , <code>'method'</code>
<code>plot.optimalSpacing</code>	<code>'add'</code> , ...

The argument `CF` may be set to `NA` to suppress rule-of-thumb RSE, including optimisation. `range(R)` specifies the search interval for optimisation.

A plot method is provided, with options for plotting different components.

Value

List of two components, one for the rule-of-thumb optimisation (`rotRSE`) and the other for simulation results, if requested (`simRSE`).

The optimisation results are

<code>values</code>	dataframe with $E(n)$, $E(r)$ and the rule-of-thumb RSE for each requested R
<code>optimum.spacing</code>	the absolute spacing that yields maximum precision (minimum rule-of-thumb RSE(D-hat))
<code>optimum.R</code>	spacing relative to sigma
<code>minimum.RSE</code>	final value of the objective function (minimum rule-of-thumb RSE(D-hat))

The simulation results in the dataframe `simRSE` are the mean and SE of the simulated RSE(D-hat) for each level of `simulationR`, with added columns for the relative bias (RB) and relative root-mean-square-error (rRMSE) of D-hat.

Results are returned invisibly if `plt = TRUE`.

Warnings

For single-catch traps, use of a maximum likelihood estimate of `lambda0` from a fitted multi-catch model results in negative bias.

Only hazard-based detection functions are supported. The meaning of the ‘sigma’ parameter depends on the function, and so will the optimal spacing in sigma units.

Note

fit.function = ‘openCR.fit’ was deprecated from 2.5.8 and has been removed as an option

See Also

[minnrRSE](#), [plot.optimalSpacing](#)

Examples

```
grid <- make.grid(7, 7) # default multi-catch detector
optimalSpacing(D = 5, traps = grid, detectpar = list(lambda0 = 0.2, sigma = 20),
  nooccasions = 5, plt = TRUE)

## Not run:

optimalSpacing(D = 5, traps = grid, detectpar = list(lambda0 = 0.4, sigma = 20),
  detectfn = 'HEX', R = seq(1,6,0.4), nooccasions = 10, plt = TRUE, col = "blue")

## with simulations
grid <- make.grid(8, 8, spacing = 20, detector = 'proximity')
optimalSpacing(D = 5, traps = grid, detectfn = "HHN", detectpar =
  list(lambda0 = 0.2, sigma = 20), nooccasions = 5, nrepl = 20, nx = 32,
  ncores = 4, plt = TRUE, col = "blue")

## manual check
grid <- make.grid(8, 8, spacing = 60, detector = 'proximity')
scen <- make.scenarios(D = 5, detectfn = 14, lambda0 = 0.2, sigma = 20,
  nooccasions = 5)
sim1 <- run.scenarios(nrepl = 20, scen, trapset = list(grid), fit = TRUE,
  fit.args = list(detectfn = 14), ncores = 4, byscenario = FALSE)
summary(sim1)

## End(Not run)
```

plot.optimalSpacing *Plot and print methods for optimalSpacing object*

Description

Plots or print results from optimalSpacing.

Usage

```
## S3 method for class 'optimalSpacing'
plot(x, add = FALSE, plottype = c("RSE", "nrm"), ...)
## S3 method for class 'optimalSpacing'
print(x, ...)
```

Arguments

x object from [optimalSpacing](#)
 add logical; if TRUE will add to existing plot
 plottype character code
 ... other arguments for plot, lines or points

Details

If type = "RSE" then RSE(D-hat) is plotted against R (relative detector spacing), otherwise the expected numbers of individuals, recaptures and movements are plotted against R.

The ... argument may be used to pass other plotting arguments to override defaults:

Function	Arguments	Note
plot	'xlab', 'ylab', 'xlim', 'ylim', 'las', 'xaxs', 'yaxs'	add = FALSE
points	'col', 'cex', 'pch'	optimum and simulated RSE
lines	'col', 'lwd', 'lty'	rule-of-thumb RSE

The print method removes attributes before printing.

Value

None

See Also

[optimalSpacing](#)

predict.fittedmodels *Extract Estimates From Fitted Models*

Description

If simulations have been saved from run.scenarios as fitted secr models it is necessary to use one of these functions to extract estimates for later summarization.

Usage

```
## S3 method for class 'fittedmodels'
predict(object, ...)

## S3 method for class 'fittedmodels'
coef(object, ...)

## S3 method for class 'fittedmodels'
derived(object, ...)

## S3 method for class 'fittedmodels'
region.N(object, ...)
```

Arguments

object	fitted model simulation output from run.scenarios
...	other arguments passed to predict, coef, derived or region.N

Details

These functions are used when output from [run.scenarios](#) has been saved as fitted models. `derived` and `region.N` require a full fit (including the mask and design0 objects) whereas a trimmed model is sufficient for `predict` and `coef`.

`derived` is used to compute the Horvitz-Thompson-like estimate of density when [secre.fit](#) has been used with `CL = TRUE`; it is roughly equivalent to `predict`.

`region.N` predicts the realised number (R.N) or expected number (E.N) in a masked area. When detector layouts and/or sigma vary, the masked area will also vary (arbitrarily, depending on the buffer argument 'xsigma') unless a mask is provided by the user; this may be done either in [run.scenarios](#) or in `region.N`.

Value

An object with class ('estimatables', 'secredesign', 'list') with appropriate outputtype ('predicted', 'coef', 'derived', 'regionN'; see also [run.scenarios](#)).

Note

From [secredesign](#) 2.5.3 the methods described here replace the functions `derived.SL` and `regionN.SL`. This is for compatibility with **secre**.

See Also

[run.scenarios](#) [coef.secre](#) [predict.secre](#) [derived.secre](#) [region.N.secre](#)

Examples

```
## Not run:
scen1 <- make.scenarios(D = c(3,6), sigma = 25, g0 = 0.2)
traps1 <- make.grid() ## default 6 x 6 grid of multi-catch traps
tmp1 <- run.scenarios(nrepl = 10, trapset = traps1, scenarios = scen1,
  fit = TRUE, extractfn = trim)
tmp2 <- predict(tmp1)
tmp3 <- select.stats(tmp2, 'D', c('estimate','RB','RSE'))
summary(tmp3)

## for derived and region.N need more than just 'trimmed' secr object
## use argument 'keep' to save mask and design0 usually discarded by trim
tmp4 <- run.scenarios(nrepl = 10, trapset = traps1, scenarios = scen1,
  fit = TRUE, extractfn = trim, keep = c('mask','design0'))

summary(derived(tmp4))

## for region.N we must specify the parameter for which we want statistics
## (default 'D' not relevant)
tmp5 <- select.stats(region.N(tmp4), parameter = 'E.N')
summary(tmp5)

## End(Not run)
```

rbind.estimatetables *Combine Simulation Output*

Description

Methods to combine output from separate executions of [run.scenarios](#).

Usage

```
## S3 method for class 'estimatetables'
rbind(..., deparse.level = 1)

## S3 method for class 'selectedstatistics'
rbind(..., deparse.level = 1)

## S3 method for class 'estimatetables'
c(...)

## S3 method for class 'selectedstatistics'
c(...)
```

Arguments

... estimatetables or selectedstatistics output from [run.scenarios](#)
 deparse.level not used (required by generic method [rbind](#))

Details

[rbind](#) assumes all inputs used exactly the same scenarios. Replicate estimate tables are combined across executions for each scenario in turn. This is useful to increase the number of replicates by combining two batches of simulations with different random seeds. The ‘scenarios’ component remains unchanged.

[c](#) combines outputs from [run.scenarios](#) that may differ in their scenarios. The ‘output’ component of the result is a concatenation of the output lists in the input. The ‘scenarios’ component of the result comprises the input scenarios stacked with [rbind.data.frame](#).

The compatibility of the inputs is checked, but the checks are not exhaustive. Users should be wary.

Value

‘[estimatetables](#)’ or ‘[selectedstatistics](#)’ object combining the inputs

See Also

[make.scenarios](#) [run.scenarios](#)

Examples

```
## Simple example: generate and summarise trapping data at two densities
## result inherits from 'selectedstatistics'

scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2, noccasions = 5)
traps1 <- make.grid() ## default 6 x 6 trap grid
tmp1 <- run.scenarios(nrepl = 5, trapset = traps1, scenarios = scen1,
  fit = FALSE, seed = 123)
tmp2 <- run.scenarios(nrepl = 15, trapset = traps1, scenarios = scen1,
  fit = FALSE, seed = 127)

summary(rbind(tmp1,tmp2))
summary(c(tmp1,tmp2))
```

Description

This function performs simulations to predict the precision of density and other estimates from simple 1-session SECR designs. Scenarios are specified via an input dataframe that will usually be constructed with `make.scenarios`. Each scenario comprises an index to a detector layout, the number of sampling occasions, and specified density (D) and detection parameters (usually g_0 and σ).

Detector layouts are provided in a separate list `trapset`. This may comprise an actual field design input with `read.traps` or ‘traps’ objects constructed with `make.grid` etc., as in the Examples. Even a single layout must be presented as a component of a list (e.g., `list(make.grid())`).

Alternative approaches are offered for predicting precision. Both start by generating a pseudorandom dataset under the design using the parameter values for a particular scenario. The first estimates the parameter values and their standard errors from each dataset by maximizing the full likelihood, as usual in `secr.fit`. The second takes the short cut of computing variances and SE from the Hessian estimated numerically at the known expected values of the parameters, without maximizing the likelihood. Set `method = "none"` in `fit.args` for this shortcut.

Usage

```
run.scenarios(nrepl, scenarios, trapset, maskset, xsigma = 4, nx = 32,
  pop.args, CH.function = c("sim.caphist", "simCH"), det.args,
  fit = FALSE, fit.function = c("secr.fit", "ipsecr.fit"),
  fit.args, chatnsim, extractfn = NULL, multisession = FALSE,
  ncores = NULL, byscenario = FALSE, seed = 123, trap.args, ...)

fit.models(rawdata, fit = FALSE, fit.function = c("secr.fit", "ipsecr.fit"),
  fit.args, chatnsim, extractfn = NULL, ncores = NULL, byscenario = FALSE,
  scen, repl, ...)
```

Arguments

<code>nrepl</code>	integer number of replicate simulations
<code>scenarios</code>	dataframe of simulation scenarios
<code>trapset</code>	secr traps object or a list of traps objects or functions
<code>maskset</code>	secr mask object or a list of mask objects (optional)
<code>xsigma</code>	numeric buffer width as multiple of sigma (alternative to <code>maskset</code>)
<code>nx</code>	integer number of cells in mask in x direction (alternative to <code>maskset</code>)
<code>pop.args</code>	list of named arguments to <code>sim.popn</code> (optional)
<code>CH.function</code>	character name of function to simulate caphist
<code>det.args</code>	list of named arguments to <code>sim.caphist</code> (optional)
<code>fit</code>	logical; if TRUE a model is fitted with <code>fit.function</code> , otherwise data are generated but no model is fitted (see also Design-only statistics in Details)
<code>fit.function</code>	character name of function to use for model fitting

fit.args	list of named arguments to <code>secr.fit</code> or <code>ipsecr.fit</code> (optional)
chatnsim	integer number of simulations for overdispersion of mark-resight models
extractfn	function to extract a vector of statistics from secr model
multisession	logical; if TRUE groups are treated as additional sessions
ncores	integer number of cores for parallel processing or NULL
byscenario	logical; if TRUE then each scenario is sent to a different core
seed	integer pseudorandom number seed
trap.args	list of arguments for trapset components if using function option
...	other arguments passed to extractfn
rawdata	'rawdata' object from previous call to <code>run.scenarios</code>
scen	integer vector of scenario subscripts
repl	integer vector of subscripts in range 1:nrepl

Details

Designs are constructed from the trap layouts in `trapset`, the numbers of grids in `ngrid`, and the numbers of sampling occasions (secondary sessions) in `noccasions`. These are *not* crossed: the number of designs is the maximum length of any of these arguments. Any of these arguments whose length is less than the maximum will be replicated to match.

`pop.args` is used to customize the simulated population distribution. It will usually comprise a single list, but may be a list of lists (one per `popindex` value in scenarios).

`det.args` may be used to customize some aspects of the detection modelling in `sim.caphist`, but not traps, `popn`, `detectpar`, `detectfn`, and `noccasions`, which are controlled directly by the scenarios. It will usually comprise a single list, but may be a list of lists (one per `detindex` value in scenarios).

`fit.args` is used to customize the fitted model; it will usually comprise a single list. If you are interested in precision alone, use `fit.args=list(method = 'none')` to obtain variance estimates from the hessian evaluated at the parameter estimates. This is much faster than a complete model fit, and usually accurate enough.

If no `extractfn` is supplied then a default is used - see Examples. Replacement functions should follow this pattern i.e. test for whether the single argument is an `secr` object, and if not supply a named vector of NA values of the correct length.

Using `extractfn = summary` has the advantage of allowing both model fits and raw statistics to be extracted from one set of simulations. However, this approach requires an additional step to retrieve the desired numeric results from each replicate (see [count.summary](#) and [predict.summary](#)).

Parallel processing:

If `byscenario = TRUE` then by default each scenario will be run in a separate worker process using `parLapply` from **parallel** (see also [Parallel](#)). The number of scenarios should not exceed the available number of cores (set by the '`ncores`' argument or a prior call to '`setNumThreads`').

If `byscenario = FALSE` then from **secrdesign** 2.6.0 onwards the usual multithreading of **secr** 4.5 is applied. The number of cores should usually be preset with '`setNumThreads`'. If `ncores` is provided then the environment variable `RCPPE_PARALLEL_NUM_THREADS` is reset. The

default behaviour of the fitting functions (`secr.fit`, `ipsecr.fit`) is to use this value (unless specified in `fit.args`).

When `'byscenario = TRUE'` the L'Ecuyer pseudorandom generator is used with a separate random number stream for each core (see `clusterSetRNGStream`).

For `ncores > 1` it pays to keep an eye on the processes from the Performance page of Windows Task Manager (`<ctrl><alt>`), or 'top' in linux OS. If you interrupt `run.scenarios` (`<Esc>` from Windows) you may occasionally find some processes do not terminate and have to be manually terminated from the Task Manager - they appear as `Rscript.exe` on the Processes page.

Alternate functions for simulation and fitting:

The default is to use functions `sim.caphist` and `secr.fit` from `secr`. Either may be substituted by the corresponding function (`simCH` or `ipsecr.fit`) from package `ipsecr` if that has been installed.

Design-only statistics:

Designs for distance sampling were evaluated by Fewster and Buckland (2004) by computing statistics from simulated detections without fitting a model to estimate the detection parameters. An analogous procedure for SECR is implemented by setting `fit = 'design'`. A new default `extractfn` (`designextractfn`) computes the effective sampling area with the `secr` function `pdot` and returns a vector of values -

<code>n</code>	number of individuals detected
<code>r</code>	number of recaptures
<code>esa</code>	effective sampling area, given the known detection parameters
<code>D</code>	$D = n/esa$

The resulting simulation object is of type 'selectedstatistics' for which the summary method works as usual.

A similar effect may be achieved by providing a custom `extractfn` and passing arguments to it via the `dots` argument of `run.scenarios`.

Miscellaneous:

From 2.2.0, two or more rows in `scenarios` may share the same scenario number. This is used to generate multiple population subclasses (e.g. sexes) differing in density and/or detection parameters. If `multisession = TRUE` the subclasses become separate sessions in a multi-session `caphist` object (this may require a custom `extractfn`). `multisession` is ignored with a warning if each scenario row has a unique number.

From 2.7.0, each component of 'trapset' may be a function that constructs a detector layout. This allows layouts to be constructed dynamically at the time each `caphist` is generated; arguments of each function are provided in the 'trap.args' list. The primary purpose is to allow systematic grids, laceworks etc. to be constructed with a unique random origin for each replicate. The 'maskset' argument must be provided - it should cover all potential layouts, regardless of origins.

In `fit.models` the arguments `scen` and `repl` may be used to select a subset of datasets for model fitting.

Mark-resight: `chatnsim` controls an additional quasi-likelihood model step to adjust for overdispersion of sighting counts. No adjustment happens when `chatnsim = 0`; otherwise `abs(chatnsim)`

gives the number of simulations to perform to estimate overdispersion. If `chatnsim < 0` then the quasilielihood is used only to re-estimate the variance at the previous MLE (method = "none").

Further processing: A summary method is provided (see [summary.secrdesign](#)). It is usually necessary to process the simulation results further with [predict.fittedmodels](#) and/or [select.stats](#) before summarization.

Value

An object of class (x, 'secrdesign', 'list'), where x is one of 'fittedmodels', 'estimatetables', 'selectedstatistics' or 'rawdata', with components

call	function call
version	character string including the software version number
starttime	character string for date and time of run
proctime	processor time for simulations, in seconds
scenarios	dataframe as input
trapset	list of trap layouts as input
maskset	list of habitat masks (input or generated)
xsigma	from input
nx	from input
pop.args	from input
CH.function	from input
det.args	from input
fit	from input
fit.function	from input
fit.args	from input
extractfn	function used to extract statistics from each simulation
seed	from input
nrepl	from input
output	list with one component per scenario
outputtype	character code - see vignette

If `fit = FALSE` and `extractfn = identity` the result is of class ('rawdata', 'secrdesign', 'list'). This may be used as input to `fit.models`, which interprets each model specification in `fit.args` as a new 'sub-scenario' of each input scenario (i.e. all models are fitted to every dataset). The output possibilities are the same as for `run.scenarios`.

If subclasses have been defined (i.e. `scenarios` has multiple rows with the same scenario ID), each simulated capthist object has covariates with a character-valued column named "group" ("1", "2" etc.) (there is also a column "sex" generated automatically by `sim.popn`).

Note

100 ha = 1 km².

`fit.function = 'openCR.fit'` was deprecated from 2.5.8 and has been removed.

Author(s)

Murray Efford

References

Fewster, R. M. and Buckland, S. T. 2004. Assessment of distance sampling estimators. In: S. T. Buckland, D. R. Anderson, K. P. Burnham, J. L. Laake, D. L. Borchers and L. Thomas (eds) *Advanced distance sampling*. Oxford University Press, Oxford, U. K. Pp. 281–306.

See Also

[select.stats](#),
[summary.secrdesign](#),
[summary.estimateables](#),
[summary.selectedstatistics](#),
[estimateSummary](#)
 Miscellaneous –
[predict.fittedmodels](#),
[scenarioSummary](#),
[count.summary](#),
[predict.summary](#)
secr functions used internally –
[sim.popn](#),
[sim.caphist](#),
[secr.fit](#)
 To combine output –
[rbind.estimateables](#),
[rbind.selectedstatistics](#),
[c.estimateables](#),
[c.selectedstatistics](#)

Examples

```
## Simple example: generate and summarise trapping data
## at two densities and for two levels of sampling frequency
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2, nooccasions =
  c(5,10))
traps1 <- make.grid() ## default 6 x 6 trap grid
tmp1 <- run.scenarios(nrepl = 20, trapset = traps1, scenarios = scen1,
  fit = FALSE)
summary(tmp1)

## Not run:
```

```

setNumThreads(7)

#####
# new summary method (secrdesign >= 2.8.1)
# assumes fit = TRUE, extractfn = predict

tmp2 <- run.scenarios(nrepl = 10, trapset = traps1, scenarios = scen1,
  fit = TRUE, extractfn = predict)
estimateSummary(tmp2, format = "data.frame",
  cols = c('scenario', 'noccasions'))

#####
## 2-phase example
## first make and save rawdata
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid() ## default 6 x 6 trap grid
tmp1 <- run.scenarios(nrepl = 20, trapset = traps1, scenarios = scen1,
  fit = FALSE, extractfn = identity)

## review rawdata
summary(tmp1)

## then fit and summarise models
tmp2 <- fit.models(tmp1, fit.args = list(list(model = g0~1),
  list(model = g0~T)), fit = TRUE)
summary(tmp2)
#####

## Construct a list of detector arrays
## Each is a set of 5 parallel lines with variable between-line spacing;
## the argument that we want to vary (spacey) follows nx, ny and spacex
## in the argument list of make.grid().

spacey <- seq(2000,5000,500)
names(spacey) <- paste('line', spacey, sep = '.')
trapset <- lapply(spacey, make.grid, nx = 101, ny = 5, spacex = 1000,
  detector = 'proximity')

## Make corresponding set of masks with constant spacing (1 km)
maskset <- lapply(trapset, make.mask, buffer = 8000, spacing = 1000,
  type = 'trapbuffer')

## Generate scenarios
scen <- make.scenarios (trapsindex = 1:length(spacey), nrepeats = 8,
  noccasions = 2, D = 0.0002, g0 = c(0.05, 0.1), sigma = 1600, cross = TRUE)

## RSE without fitting model
sim <- run.scenarios (50, scenarios = scen, trapset = trapset, maskset = maskset,
  fit = TRUE, fit.args = list(method = 'none'), seed = 123)

## Extract statistics for predicted density
sim <- select.stats(sim, parameter = 'D')

```

```

## Plot to compare line spacing
summ <- summary (sim, type='array', fields = c('mean','lcl','ucl'))$OUTPUT
plot(0,0,type='n', xlim=c(1.500,5.500), ylim = c(0,0.36), yaxs = 'i',
     xaxs = 'i', xlab = 'Line spacing km', ylab = 'RSE (D)')
xv <- seq(2,5,0.5)
points(xv, summ$mean[,1,'RSE'], type='b', pch=1)
points(xv, summ$mean[,2,'RSE'], type='b', pch=16)
segments(xv, summ$lcl[,1,'RSE'], xv, summ$ucl[,1,'RSE'])
segments(xv, summ$lcl[,2,'RSE'], xv, summ$ucl[,2,'RSE'])
legend(4,0.345, pch=c(1,16), title = 'Baseline detection',
      legend = c('g0 = 0.05', 'g0 = 0.1'))

## End(Not run)

```

saturation

Detector saturation

Description

Computes the expected proportion of successful detectors (i.e., ‘trap success’). The calculation does not allow for local variation in realised density (number of animals centred near each detector) and the predictions are therefore slightly higher than simulations with Poisson local density. The discrepancy is typically less than 1%.

Usage

```

saturation(traps, mask, detectpar, detectfn =
  c("HHN", "HHR", "HEX", "HAN", "HCG", "HN", "HR", "EX"),
  D, plt = FALSE, add = FALSE, ...)

```

Arguments

traps	secr traps object
mask	secr mask object
detectpar	a named list giving a value for each parameter of detection function
detectfn	integer code or character string for shape of detection function – see detectfn
D	population density animals / hectare; may be scalar or vector of length nrow(mask)
plt	logical; if TRUE then a colour plot is produced
add	logical; if TRUE any plot is added to the existing plot
...	other arguments passed to plot.mask when plt = TRUE

Details

The calculation is based on an additive hazard model. If `detectfn` is not a hazard function ('HHN', 'HEX', 'HHR', 'HAN' and 'HCG') then an attempt is made to approximate one of the hazard functions (HN -> HHN, HR -> HHR, EX -> HEX). The default is 'HHN'.

Computation is not possible for single-catch traps.

An empirical estimate of saturation is the total number of detectors visited divided by the total number of detectors used. These are outputs from the summary method for `capthist` objects. See Examples.

Value

A list with components

<code>bydetector</code>	expected saturation for each detector
<code>mean</code>	average over detectors

The list is returned invisibly if `plt = TRUE`.

See Also

[Enrm](#)

Examples

```
tr <- traps(captdata)
detector(tr) <- 'multi'
mask <- make.mask(tr, buffer = 100)
saturation(tr, mask, detectpar = list(lambda0 = 0.27, sigma = 29),
  detectfn = 'HHN', D = 5.5, plt = TRUE)
plotMaskEdge(as.mask(tr), add = TRUE) ## boundary line

# empirical - useful for extractfn argument of secrdesign::run.scenarios
satfn <- function(CH) {
  sumCH <- summary(CH)$counts
  sumCH['detectors visited', 'Total'] / sumCH['detectors used', 'Total']
}
satfn(captdata)
```

Description

The `make.scenarios` function requires prior knowledge of population density and the intercept of the detection function (g_0). This function provides an alternative mechanism for generating scenarios from a value of σ and target values for the numbers of individuals n and recaptures r . Only a halfnormal detection function is supported (probability, not hazard), and many options in `make.scenarios` have yet to be implemented. Only a single detector layout and single mask may be specified.

Usage

```
scenariosFromStatistics(sigma, nooccasions, traps, mask, nval, rval,  
  g0.int = c(0.001, 0.999))
```

Arguments

<code>sigma</code>	numeric vector of one or more values for σ
<code>nooccasions</code>	integer vector of number of sampling occasions
<code>traps</code>	traps object
<code>mask</code>	mask object
<code>nval</code>	integer vector of values of n
<code>rval</code>	integer vector of values of r
<code>g0.int</code>	numeric vector defining the interval to be searched for g_0

Details

The algorithm is based on R code in Appendix B of Efford, Dawson and Borchers (2009).

Value

A scenario dataframe with one row for each combination of `sigma`, `nooccasions`, `nval` and `rval`.

References

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

See Also

[make.scenarios](#)

Examples

```
grid36 <- make.grid(nx = 6, ny = 6, spacing = 200)
mask <- make.mask(grid36, buffer = 2000)
scen <- scenariosFromStatistics (sigma = c(200,400), noccasions = 44,
  traps = grid36, mask = mask, nval = 14, rval = 34)
sim <- run.scenarios(scen, nrepl = 5, traps = grid36, mask = mask)
summary(sim)
```

scenarioSummary *Summary of Scenarios*

Description

Compute various deterministic summaries for scenarios generated by `make.scenarios`

Usage

```
scenarioSummary(scenarios, trapset, maskset, xsigma = 4, nx = 64, CF = 1.0,
  costing = FALSE, ..., ncores = 1)
```

Arguments

scenarios	dataframe of simulation scenarios
trapset	secr traps object or a list of traps objects
maskset	secr mask object or a list of mask objects (optional)
xsigma	numeric buffer width as multiple of sigma (alternative to maskset)
nx	integer number of cells in mask in x direction (alternative to maskset)
CF	numeric correction factor for rule-of-thumb RSE (see minnrRSE)
costing	logical; if TRUE then costings will be appended
...	arguments passed to costing
ncores	integer number of cores for parallel processing

Details

Not all scenarios from `make.scenarios()` are suitable. Grouped (multi-line) scenarios are excluded. Hazard detection functions are preferred ('HHN', 'HHR', 'HEX', 'HAN', 'HCG'). 'HN', 'HR' and 'EX' are converted approximately to 'HHN', 'HHR' and 'HEX' respectively, with a warning; other functions are rejected.

CF may be a vector of values that is recycled across the components of trapset. The correction factor is a multiplier applied after all other calculations.

The approximate $RSE(\hat{D})$ is $rotRSE = CF / \sqrt{\min(E(n), E(r))}$. This assumes n is Poisson-distributed. For binomial n an ad hoc adjustment is $rotRSEB = \sqrt{rotRSE^2 - 1 / (D \times A)}$ where A is the mask area.

The default `ncores = 1` (new in 2.7.0) is usually faster than setting `ncores > 1` because of the overheads in setting up a parallel cluster.

The `...` argument is for inputs to [costing](#), including `unitcost` (required) and `routelength` (optional).

Value

A dataframe including the first 8 columns from scenarios and the computed columns –

En	expected number of individuals
Er	expected number of recaptures
Em	expected number of movement recaptures
En2	expected number of individuals detected at two or more sites
esa	effective sampling area (ha)
CF	rule-of-thumb correction factor
rotRSE	rule-of-thumb relative standard error of density estimate
rotRSEB	rotRSE with adjustment for fixed N in region defined by mask (i.e. Binomial n rather than Poisson n)
arrayN	number of detectors in each array
arrayspace	array spacing in sigma units
arrayspan	largest dimension of array in sigma units
saturation	expected proportion of detectors at which detection occurs (trap success)
travel	travel cost
arrays	cost of each repeated array
detectors	fixed cost per detector
visits	cost per detector per visit
detections	cost per detection
totalcost	summed costs
detperHR	median number of detectors per 95% home range

Costings (the last 6 columns) are omitted if `costing = FALSE`.

See Also

[make.scenarios](#), [Enrm](#), [costing](#), [minnrRSE](#)

Examples

```
scen <- make.scenarios(D = c(5,10), sigma = 25, lambda0 = 0.2, detectfn = 'HHN')
grid <- make.grid(6,6, detector = 'multi')
scenarioSummary(scen, list(grid), costing = TRUE, unitcost = list(perkm = 10))
```

select.stats *Select Statistics to Summarize*

Description

When the results of each simulation with `run.scenarios` are saved as a dataframe (e.g. from `predict()`) it is necessary to select estimates of just one parameter for numerical summarization. This does the job. `find.param` is a helper function to quickly display the parameters available for summarisation.

Usage

```
select.stats(object, parameter = "D", statistics, true)
find.param(object)
find.stats(object)
```

Arguments

<code>object</code>	'estimatetables' object from run.scenarios
<code>parameter</code>	character name of parameter to extract
<code>statistics</code>	character vector of statistic names
<code>true</code>	numeric vector of 'true' values of parameter, one per scenario

Details

`select.stats` is used to select a particular vector of numeric values for summarization. The 'parameter' argument indexes a row in the data.frame for one replicate (i.e., one 'real' parameter). Each 'statistic' is either a column in that data.frame or a statistic derived from a column.

If `statistics` is not specified, the default is to use all numeric columns in the input (i.e., `c('estimate', 'SE.estimate', 'lcl', 'ucl')` for `predict` and `c('beta', 'SE.beta', 'lcl', 'ucl')` for `coef`).

`statistics` may include any of 'estimate', 'SE.estimate', 'lcl', 'ucl', 'true', 'RB', 'RSE', 'COV' and 'ERR' (for outputtype 'coef' use 'beta' and 'SE.beta' instead of 'estimate' and 'SE.estimate'). 'true' refers to the known parameter value used to generate the data.

The computed statistics are:

Statistic	Name	Value
RB	Relative bias	$(\text{estimate} - \text{true}) / \text{true}$
RSE	Relative SE	$\text{SE.estimate} / \text{estimate}$
ERR	Absolute deviation	$\text{abs}(\text{estimate} - \text{true})$
COV	Coverage	$(\text{estimate} > \text{lcl}) \ \& \ (\text{estimate} < \text{ucl})$

'RB', 'COV' and 'ERR' relate an estimate to the known (true) value of the parameter in `object$scenarios`.

They are computed only when a model has been fitted without `method = 'none'`.

'COV' remains binary (0/1) in the output from `select.stats`; the result of interest is the mean of this statistic across replicates (see [summary.secrdesign](#)). Similarly, 'ERR' is used with field 'rms' in [summary.secrdesign](#) to compute the root-mean-squared-error RMSE.

`find.param` and `find.stats` may be used to 'peek' at objects of class 'estimatetables' and 'selectedstatistics' respectively to recall the available parameter estimates or 'statistics'.

An attempt is made to extract `true` automatically if it is not provided. This does not always work (e.g. with `extractfn.region.N`, region differing from the mask, and a heterogeneous density model). Check this by including "true" as a statistic to summarise (see Examples).

Value

For `select.stats`, an object with class `c('selectedstatistics','secrdesign','list')` suitable for numerical summarization with [summary.selectedstatistics](#). The value of 'parameter' is stored as an attribute.

For `find.param`, a character vector of the names of parameters with estimates in object.

See Also

[run.scenarios](#), [validate](#)

Examples

```
## using nrepl = 2 just for checking
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid()
tmp1 <- run.scenarios(nrepl = 2, trapset = traps1, scenarios = scen1,
  fit = TRUE, extractfn = secr::trim)
tmp2 <- predict(tmp1)
tmp3 <- select.stats(tmp2, 'D', c('estimate','true','RB','RSE','COV'))
summary(tmp3)
```

summary.secrdesign *Generic Methods for secrdesign Objects*

Description

Methods to summarize simulated datasets.

Usage

```
## S3 method for class 'secrdesign'
summary(object, ...)

## S3 method for class 'rawdata'
summary(object, ...)
```

```
## S3 method for class 'estimatetables'
summary(object, ...)

## S3 method for class 'selectedstatistics'
summary(object, fields = c('n', 'mean',
'se'), dec = 5, alpha = 0.05, type = c('list','dataframe','array'), ...)

## S3 method for class 'selectedstatistics'
plot(x, scenarios, statistic, type =
c('hist', 'CI'), refline, xlab = NULL, ...)

header(object)
```

Arguments

object	object of class simulations from run.scenarios
dec	number of decimal places in output
fields	character vector; names of required summary statistics (see Details)
alpha	alpha level for confidence intervals and quantiles
type	character code for type of output (see Details)
...	other arguments – not currently used by summary but passed to hist by the plot method
x	object of class 'selectedstatistics' from run.scenarios
scenarios	integer indices of scenarios to plot (all plotted if not specified)
statistic	integer or character indices of the statistics in x for which histograms are requested
refline	logical; if TRUE a reference line is plotted at the true value of a parameter
xlab	character; optional label for x-axis

Details

If object inherits from 'selectedstatistics' then the numeric results from replicate simulations are summarized using the chosen 'fields' (by default, the number of non-missing values, mean and standard error), along with header information describing the simulations. Otherwise the header alone is returned.

fields is a vector of any selection from c('n', 'mean', 'sd', 'se', 'min', 'max', 'lcl', 'ucl', 'median', 'q', 'rms', 'var'), or the character value 'all'.

Field 'q' provides 1000 alpha/2 and 1000[1 - alpha/2] quantiles qxxx and qyyy.

'lcl' and 'ucl' refer to the upper and lower limits of a 100(1 - alpha)% confidence interval for the statistic, across replicates.

'rms' gives the root-mean-square of the statistic - most useful for the statistic 'ERR' (see [select.stats](#)) when it represents the overall accuracy or RMSE.

The plot method plots either (i) histograms of the selected statistics (type = 'hist') or (ii) the estimate and confidence interval for each replicate (type = 'CI'). The default for type = 'hist' is to plot the first statistic - this is usually 'n' (number of detected animals) when fit = FALSE, and 'estimate' (parameter estimate) when fit = TRUE. If length(statistic) > 1 then more than one plot will be produced, so a multi-column or multi-row layout should be prepared with par arguments 'mfc' or 'mfrow'.

For type = 'CI' the statistics must include 'estimate', 'lcl' and 'ucl' (or 'beta', 'lcl' and 'ucl' if outputtype = 'coef').

[estimateSummary](#) is a simpler approach that provides full output for models with groups or multiple sessions simulated in [run.scenarios](#) with extractfn predict or coef).

Value

List with components 'header'

call	original function call
starttime	from object
proctime	from object
constants	small dataframe with values of non-varying inputs
varying	small dataframe with values of varying inputs
fit.args	small dataframe with values arguments for secr.fit, if specified

and 'OUTPUT', a list with one component for each field. Each component may be a list or an array.

See Also

[run.scenarios](#), [make.array](#), [select.stats](#) [validate estimateSummary](#)

Examples

```
## collect raw counts
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid()
tmp1 <- run.scenarios(nrepl = 50, trapset = traps1, scenarios = scen1,
  fit = FALSE)

opar <- par(mfrow=c(2,3))
plot(tmp1, statistic = 1:3)
par(opar)

summary(tmp1)

summary(tmp1, field=c('q025', 'median', 'q975'))
```

validate	<i>Reject Implausible Statistics</i>
----------	--------------------------------------

Description

Simulation output may contain rogue values due to idiosyncracies of model fitting. For example, nonidentifiability due to inadequate data can result in spurious extreme ‘estimates’ of the sampling variance. Undue influence of rogue replicates can be reduced by using the median as a summary field rather than the mean. This function is another way to deal with the problem, by setting to NA selected statistics from replicates for which some ‘test’ statistic is out-of-range.

Usage

```
validate(x, test, validrange = c(0, Inf), targets = test, quietly = FALSE)
```

Arguments

x	object that inherits from ‘selectedstatistics’
test	character; name of statistic to check
validrange	numeric vector comprising the minimum and maximum permitted values of ‘test’, or a matrix (see details)
targets	character vector with names of one or more statistics to set to missing (NA) when test is out-of-range
quietly	logical; if TRUE messages are suppressed

Details

Values of ‘test’ and ‘targets’ should be columns in each component ‘replicate x statistic’ matrix (i.e., scenario) of `x$output`. You can check for these with [find.stats](#).

If `validrange` is a matrix its first and second columns are interpreted as scenario-specific bounds (minima and maxima), and the number of rows must match the number of scenarios.

If all non-missing values of ‘test’ are in the valid range, the effect is to force the target statistics to NA wherever ‘test’ is NA.

The default is to change only the test field itself. If the value of ‘test’ does not appear in ‘targets’ then the test field is unchanged.

If `targets = "all"` then all columns are set to NA when the test fails.

Value

An object of class `c(‘selectedstatistics’, ‘secrdesign’, ‘list’)` with the same structure and header information as the input, but possibly with some values in the ‘output’ component converted to NA.

See Also

[select.stats](#), [find.stats](#)

Examples

```
## Not run:

## generate some data
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid()
tmp1 <- run.scenarios(nrepl = 5, trapset = traps1, scenarios = scen1,
  fit = TRUE, extractfn = trim)
tmp2 <- predict(tmp1)
tmp3 <- select.stats(tmp2, 'D', c('estimate','RB','RSE','COV'))

## just for demonstration --
## apply scenario-specific +/- 20% bounds for estimated density
## set RB, RSE and COV to NA when estimate is outside this range
permitted <- outer(tmp3$scenarios$D, c(0.8,1.2))
permitted ## a 2 x 2 matrix
tmp4 <- validate(tmp3, 'estimate', permitted, c('RB', 'RSE','COV'))

## what have we done?!
tmp4$output
summary(tmp4)

## End(Not run)
```

Index

- * **Datagen**
 - run.scenarios, 28
- * **Generic**
 - summary.secrdesign, 41
- * **datagen**
 - count, 6
 - getdetectpar, 14
 - scenariosFromStatistics, 36
- * **design**
 - optimalSpacing, 22
- * **hplot**
 - plot.optimalSpacing, 24
- * **manip**
 - Lambda, 16
 - make.array, 19
 - make.scenarios, 20
 - predict.fittedmodels, 25
 - saturation, 35
 - select.stats, 40
 - validate, 44
- * **package**
 - secrdesign-package, 2
- c.estimatetables, 33
- c.estimatetables
 - (rbind.estimatetables), 27
- c.selectedstatistics, 33
- c.selectedstatistics
 - (rbind.estimatetables), 27
- clusterSetRNGStream, 31
- coef(predict.fittedmodels), 25
- coef.secr, 7, 26
- coef.summary(count), 6
- compactSample, 12, 13
- compactSample(Internal), 15
- costing, 3, 4, 38, 39
- count, 6
- count.summary, 30, 33
- derived(predict.fittedmodels), 25
- derived.secr, 26
- detectfn, 11, 17, 20–22, 35
- En2, 3, 12, 13
- En2(Lambda), 16
- Enrm, 3, 5, 12, 13, 15, 36, 39
- Enrm(Lambda), 16
- estimateArray(estimateSummary), 7
- estimateSummary, 7, 33, 43
- expand.grid, 20
- find.param(select.stats), 40
- find.stats, 44
- find.stats(select.stats), 40
- fit.models, 2
- fit.models(run.scenarios), 28
- GAoptim, 3, 11, 16–18
- GApfn, 12, 13
- GApfn(Internal), 15
- getdetectpar, 14, 18
- header, 9
- header(summary.secrdesign), 41
- hist, 42
- Internal, 15
- ipsecr.fit, 30, 31
- kofnGA, 12
- Lambda, 15, 16
- make.array, 19, 21, 43
- make.grid, 4, 29
- make.scenarios, 2, 20, 28, 29, 37, 39
- mask, 17, 18
- minnrRSE, 3, 13, 23, 24, 38, 39
- minnrRSE(Lambda), 16
- optimalSpacing, 3, 18, 22, 25

- optimize, [23](#)
- Parallel, [30](#)
- pdot, [31](#)
- plot.optimalSpacing, [24, 24](#)
- plot.selectedstatistics, [3](#)
- plot.selectedstatistics
 - (summary.secrdesign), [41](#)
- predict (predict.fittedmodels), [25](#)
- predict.fittedmodels, [2, 25, 32, 33](#)
- predict.secr, [7, 26](#)
- predict.summary, [30, 33](#)
- predict.summary (count), [6](#)
- print.optimalSpacing
 - (plot.optimalSpacing), [24](#)
- Qpm (Lambda), [16](#)
- rbind, [28](#)
- rbind.data.frame, [28](#)
- rbind.estimatetables, [27, 33](#)
- rbind.selectedstatistics, [33](#)
- rbind.selectedstatistics
 - (rbind.estimatetables), [27](#)
- read.traps, [29](#)
- region.N (predict.fittedmodels), [25](#)
- region.N.secr, [26](#)
- run.scenarios, [2, 6, 8, 9, 19–21, 26–28, 28, 40, 41, 43](#)
- saturation, [3, 35](#)
- scenariosFromStatistics, [36](#)
- scenarioSummary, [3, 5, 18, 21, 33, 38](#)
- secr.fit, [4, 26, 30, 31, 33](#)
- secrdesign (secrdesign-package), [2](#)
- secrdesign-package, [2](#)
- select.stats, [2, 6, 19, 32, 33, 40, 42–44](#)
- sim.caphist, [4, 20, 21, 29, 31, 33](#)
- sim.popn, [4, 29, 33](#)
- simCH, [31](#)
- summary.estimatetables, [7, 9, 33](#)
- summary.estimatetables
 - (summary.secrdesign), [41](#)
- summary.rawdata (summary.secrdesign), [41](#)
- summary.secrdesign, [19, 32, 33, 41, 41](#)
- summary.selectedstatistics, [2, 33, 41](#)
- summary.selectedstatistics
 - (summary.secrdesign), [41](#)
- traps, [17, 22](#)
- validate, [41, 43, 44](#)