secrdesign - sampling design for spatially explicit capture–recapture

Murray Efford

2024-10-24

Contents

The R package **secrdesign** is a set of tools to assist the design of studies using spatially explicit capture–recapture (SECR). It provides convenient wrappers for simulation and model fitting functions in package **secr** to emulate many features of the 'Simulator' module of Density 5.0 (Efford 2012).

This document is a technical guide to the simulation aspects of **secrdesign**. Functions that do not rely on simulation are described in secrdesign-tools.pdf.

We assume an understanding of estimator properties such as bias, precision, and confidence interval coverage, and the use of Monte Carlo simulation to predict the performance of different sampling designs. Using **secrdesign** can be daunting because it allows for many different combinations of data generation, model fitting and summary statistics. Several examples are given to indicate the range of possibilities.

The Shiny application secrdesignapp is recommended as a convenient interface. It displays the **secrdesign** code for running simulations of a single scenario, among other outputs.

Introduction

When designing a study you should have in mind –

(i) a population parameter you want to measure (probably density or population size),

(ii) one or more design variables over which you have some control (number and spacing of detectors, number of sampling occasions etc.),

(iii) some pilot data, or parameter estimates from published studies, and

(iv) a criterion by which to evaluate different designs. This is most likely the precision of the estimates, as this in turn determines your ability (power) to recognise changes. Cost or effort may be an explicit criterion, or the designs may be constructed to allocate constant effort in different ways.

We use 'relative standard error' (RSE) for the relative precision of an estimate. This is sometimes called the coefficient of variation (CV) of the estimate, but RSE is more appropriate. We use 'accuracy' in the sense of Williams et al. (2002 p.45) and other authors from the United States. Accuracy combines both systematic error (bias) and precision: one measure is the square root of the mean squared difference between the true value and the estimate (RMSE).

Once you have sorted out (i)–(iv) you are ready to define and compare sampling scenarios. Scenarios are specified in a dataframe that is usually constructed with `make.scenarios` (see The scenarios dataframe). Each scenario has an integer 'trapsindex' code that identifies a particular detector array; a list of detector arrays is constructed separately, possibly using functions such as `make.grid`.

Scenarios may be investigated in two ways –

- tabulation of expected counts etc. with `scenarioSummary`
- Monte Carlo simulation with `run.scenarios`.

Monte Carlo simulation is ultimately the more reliable and comprehensive route, but it is much slower.

Fig. 1 shows the sequence of steps taken in **secrdesign** to conduct simulations and summarise the results. Each step is described in detail in a later section. Typically, you will use `run.scenarios()` to generate data and fit SECR models, select some statistics with `select.stats()`, and summarise and plot the results.

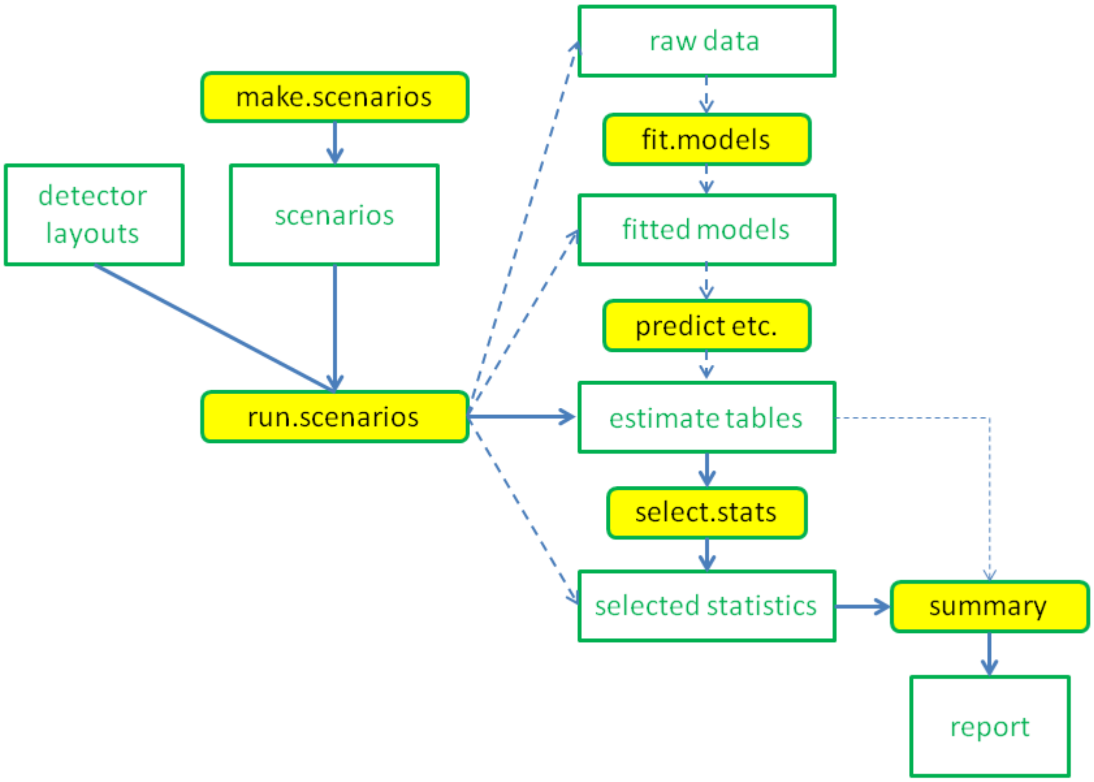


Fig. 1. Core functions in **secrdesign** (yellow) and their main inputs and outputs. Output from the simulation function `run.scenarios()` may be saved as whole fitted models, predicted values (parameter estimates), or selected statistics. Each form of output requires different subsequent handling. The default path is shown by solid blue arrows.

A simple example

For an introductory example we construct a simple set of scenarios and perform some simulations. The trap layout is a default 6 x 6 grid of multi-catch traps at 20-m spacing. Density takes one of two levels (5/ha or 10/ha) and detection parameters sigma and lambda0 are fixed.

```
library(secrdesign)
```

```
## Loading required package: secr
## This is secr 5.0.0. For overview type ?secr
## This is secrdesign 2.9.2 . For overview type ?secrdesign
```

```
setNumThreads(18)    # adjust as appropriate (max 8 on quadcore Intel)
```

```
## [1] 18
```

```
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid(8, 8, spacing = 25)
scenarioSummary(scen1, traps1)
```

```
##   scenario trapsindex noccasions nrepeats D  g0 sigma detectfn     En     Er     Em
## 1        1          1          3        1 5 0.2    25        0 27.652 22.453 19.601
```

```
## 2        2          1          3        1 10 0.2    25        0 55.303 44.905 39.203
##      En2      esa CF rotRSE rotRSEB arrayN arrayspace arrayspan saturation
## 1 14.176 5.530333  1 0.2110  0.1714     64          1    9.8995  0.2369281
## 2 28.351 5.530333  1 0.1492  0.1212     64          1    9.8995  0.4172652
##   detperHR.median         k
## 1               8 0.5590170
## 2               8 0.7905694
```

The initial summary includes the expected number of individuals (En) and recaptures (Er), and an approximate rule-of-thumb relative standard error for estimated density (rotRSE). `scenarioSummary` approximates the default detection function (HN) by a hazard-based detection function (HHN) with a warning that is not shown here. See `?scenarioSummary` and secrdesign-tools.pdf for further details.

```
sims1 <- run.scenarios(nrepl = 50, trapset = traps1, scenarios =
     scen1, seed = 345, fit = TRUE, fit.args = list(detectfn = 'HHN'))
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed in 0.372 minutes
```

The simulation output is an object of class `c( "estimatetables", "secrdesign", "list")`. We use the summary method for estimatetables to view results, and here display only the summary output (omitting a header that describes the simulations).

```
summary(sims1)$OUTPUT
```

```
## $`1`
##              n    mean      se
## estimate    50 5.02808 0.15602
## SE.estimate 50 1.05707 0.01865
## lcl         50 3.35008 0.12476
## ucl         50 7.56696 0.18881
## RB          50 0.00562 0.03120
## RSE         50 0.21583 0.00426
## COV         50 0.98000 0.02000
##
## $`2`
##              n     mean      se
## estimate    50  9.86995 0.20303
## SE.estimate 50  1.47010 0.01856
## lcl         50  7.38442 0.17085
## ucl         50 13.19702 0.23887
## RB          50 -0.01301 0.02030
## RSE         50  0.15023 0.00142
## COV         50  0.96000 0.02799
```

In this example there is close agreement between the fast rule-of-thumb RSE and the simulation results.

Later sections show how to customize the summary and plot results.

Defining scenarios

Detector layouts

Detector layouts are specified as **secr** 'traps' objects. These may be input from text files using `read.traps` or constructed according to a particular geometry and spacing with functions such as `make.grid`, `make.circle`, make.systematic or trap.builder. See the help files for these **secr** functions for further details. The detector type (multi-catch trap, proximity detector etc.) is stored as an attribute of each 'traps' object, which may also include detector-level covariates and detector 'usage' by occasion.

Multiple layouts are combined in a single list object; component names ('grid6x6' etc.) will be used to annotate the output.

```
library(secrdesign)
```

```
## Warning: package 'secrdesign' was built under R version 4.4.1
## Loading required package: secr
## This is secr 5.0.1 pre-release. For overview type ?secr
## This is secrdesign 2.9.2 . For overview type ?secrdesign
```

```
mydetectors <- list(grid6x6 = make.grid(6,6),
                    grid8x9 = make.grid(8,9),
                    grid12x12 = make.grid(12,12))
```

This creates square grids with the default detector type 'multi' and default spacing 20 m. See ?secr::make.grid for other options.

The scenarios dataframe

The function make.scenarios() constructs a dataframe in which each row defines a simulation scenario. Its arguments (with defaults) are:

```
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)
```

Each argument except for 'groups' and 'crosstraps' may be used to specify a range of values for a parameter. Four ('trapsindex', 'popindex', 'detindex', 'fitindex') are actually surrogate numerical indices; the index is used to select one component from a list of possibilities later provided as input to run.scenarios().

By default, a scenario is formed from each unique combination of the input values (trapsindex, noccasions, nrepeats, D, g0, sigma, lambda0, detectfn, recapfactor, popindex, and fitindex) using expand.grid. For example,

```
make.scenarios (trapsindex = 1:3, noccasions = 4, D = 5, g0 = 0.2, sigma = c(20,30))
```

```
##   scenario trapsindex noccasions nrepeats D  g0 sigma detectfn recapfactor popindex
## 1        1          1          4        1 5 0.2    20        0           1        1
## 2        2          2          4        1 5 0.2    20        0           1        1
## 3        3          3          4        1 5 0.2    20        0           1        1
## 4        4          1          4        1 5 0.2    30        0           1        1
## 5        5          2          4        1 5 0.2    30        0           1        1
## 6        6          3          4        1 5 0.2    30        0           1        1
##   detindex fitindex
## 1        1        1
## 2        1        1
## 3        1        1
## 4        1        1
## 5        1        1
## 6        1        1
```

In this case three different grids (possibly differing in number of traps) are trapped for the same number of occasions. A more interesting possibility is to vary the number of occasions inversely with the number of traps. However, if we naively set e.g., `noccasions = c(8, 4, 2)`, this would generate all combinations of grid and number of occasions (18 different scenarios).

The alternative is to set `crosstraps = FALSE`. Then the vectors 'trapsindex', 'noccasions', and 'nrepeats' are locked together (if fewer values are provided in one of the vectors then it is re-used as required), and only the combination is crossed with the remaining parameter scenarios:

```
make.scenarios (trapsindex = 1:3, noccasions = c(8,4,2), D = 5, g0 = 0.2,
                sigma = c(20,30), crosstraps = FALSE)
```

```
##   scenario trapsindex noccasions nrepeats D  g0 sigma detectfn recapfactor popindex
## 1        1          1          8        1 5 0.2    20        0           1        1
## 2        2          2          4        1 5 0.2    20        0           1        1
## 3        3          3          2        1 5 0.2    20        0           1        1
## 4        4          1          8        1 5 0.2    30        0           1        1
## 5        5          2          4        1 5 0.2    30        0           1        1
## 6        6          3          2        1 5 0.2    30        0           1        1
##   detindex fitindex
## 1        1        1
## 2        1        1
## 3        1        1
## 4        1        1
## 5        1        1
## 6        1        1
```

All arguments except 'fitindex' control the generation of data. Note that 'g0' and 'lambda0' are alternatives: use the one appropriate to the detection function specified with `detectfn' (see`?secr::detectfn' for codes). 'D' is omitted if an inhomogeneous Poisson distribution is specified using a mask covariate (see Non-uniform populations). D is in animals / hectare (1 ha = 0.01km^2) and sigma is in metres, as in **secr**.

The 'nrepeats' column refers to the number of notional independent replicates of the particular detector layout. Notional replicates are simulated by (invisibly) multiplying density (D) by this factor, and ultimately dividing it into the estimate. Think of 5 grids of automatic cameras so widely separated that no animal moves between the grids. As detections of different animals are ordinarily modelled as independent, the entire design is equivalent to 5 times the density of animals interacting with one grid. This breaks down, of course, if animals compete for traps (as with single-catch traps), and should not be used in that case except as a rough approximation.

Just as `trapsindex` serves as a placeholder for entire detector arrays, `popindex`, `detindex` and `fitindex` tell `run.scenarios()` which set of arguments to select from `pop.args`, `det.args` and `fit.args` for `sim.popn`, `sim.capthist` and `secr.fit` respectively. These are for more advanced use: you may not need them.

When a vector of group identifiers is provided in 'groups', the population in each scenario is a set of independently sampled groups, each defined on a separate row. Groups are initially assigned the same parameter values and other settings: it is up to the user to insert group-specific values (example at Grouped populations).

Running simulations

The function `run.scenarios()` generates multiple datasets and, if requested, fits an SECR model to each one. In this section we describe its main arguments, with additional detail on habitat masks and customizing the output.

Arguments of `run.scenarios()`

```
run.scenarios (nrepl, scenarios, trapset, maskset, xsigma = 4,
    nx = 32, pop.args, CH.function = c("sim.capthist", "simCH"),
    det.args, fit = FALSE, fit.function = c("secr.fit", "ipsecr.fit"),
    fit.args, chatnsim = 0, extractfn = NULL, multisession = FALSE,
    joinsessions = FALSE, ncores = NULL, byscenario = FALSE,
    seed = 123, trap.args = NULL, prefix = NULL, ...)
```

`nrepl` determines the number of replicate simulations. Make this large enough that the summary statistics have enough precision to answer your question. This is usually a matter for experimentation, remembering that precision (SE) is proportional to the square root of `nrepl`.

`scenarios` is the dataframe constructed with `make.scenarios()` as described in the last section.

`trapset` is a single 'traps' object or a list of traps objects, as described in 'Detector layouts' above.

`maskset` is an optional set of habitat masks, usually one per detector layout. If not specified, then masks will be constructed 'on the fly' using `secr::make.mask` with a 'buffer' of width `xsigma` $\times$ `scenarios$sigma` and `nx` cells in the x dimension.

`pop.args` provides additional control over `sim.popn` (see `?secr::sim.popn` for more). You may wish, for example, to set `pop.args = list(Ndist = "fixed")` to override the default of Poisson variation in the total number of simulated animals.

`det.args` provides additional control over `sim.capthist` (see `?secr::sim.capthist` for more). One use is to retain the simulated population as an attribute of the capthist object by setting `det.args = list(savepopn = TRUE)`; another is to set the `binomN` argument for count detectors. The `sim.capthist` arguments `traps`, `popn`, `detectpar`, `detectfn` and `noccasions` are defined in the scenario or `pop.args` and cannot be overridden by setting `det.args`.

Use `fit = FALSE` to generate and summarise detection data without fitting SECR models. This lets you check that your scenarios result in reasonable numbers of detected individuals (n), detections (ndet), and movements (nmov), before launching a full-blown simulation.

When `fit = TRUE` a SECR model will be fitted using `secr.fit`. Before **secrdesign** 2.6.0 some models could also be fitted with `openCR.fit`, but this has been discontinued.

`fit.args` lets you specify how SECR models will be fitted to the simulated data. Most default arguments of `secr.fit` may be overridden by including them in `fit.args`. For example, to specify a negative exponential detection function use `fit.args = list(detectfn = "EX")`. If you wish to compare nspec different model specifications then `fit.args` should be a list of lists, one per specification, with `fitindex` taking values in the range 1:nspec.

The use of `multisession`, `ncores` and `byscenario` is discussed under Additional topics.

Customising run.scenarios()

The 'output' component of the value returned by `run.scenarios` includes the result of `extractfn` for each replicate in each scenario. `extractfn` is a (usually short) function that is applied either (i) to each simulated raw dataset (capthist object) (`fit = FALSE`), or (ii) to each fitted model (`fit = TRUE`; fitted models are of class 'secr').

Default `extractfn`

The default (builtin) `extractfn` (Appendix 1) behaves appropriately for either value of `fit`.

When `fit = FALSE`, the default output from each replicate is a vector of 4 summary statistics:

- n - number of different individuals
- ndet - total number of detections ('captures' and 'recaptures')
- nmov - total number of detections at a detector other than the one where an animal was last detected

- dpa - detectors per animal (average number of detectors at which each animal was recorded)

When `fit = TRUE`, the default output from each replicate is the result of applying `predict` to the fitted model, i.e. a dataframe of 'real' parameter estimates and their standard errors etc. (an empty dataframe is returned if model fitting fails).

Other possible `extractfn`

Table 1. Possible `extractfn` when `fit = FALSE`

Function	Note
`summary`	good alternative to the default `extractfn`
`identity`	save entire dataset

Arguments of `extractfn` may be included in the call to `run.scenarios`. In particular, the `summary` method for capthist objects works well with `terse = TRUE, moves = TRUE`.

Table 2. Possible `extractfn` when `fit = TRUE`

Function	`secr.fit`	Note
`summary`	✓	good alternative to the default `extractfn` now that there are summary methods for fitted models
`predict`	✓	nearly same as default `extractfn`
`coef`	✓	'beta' coefficients
`derived`	✓	derived density estimate; useful when `CL = TRUE`
`region.N`	✓	population size in the masked region
`identity`	✓	save entire fitted model
`trim`	✓	save pared-down version of model

The user may choose to save the entire dataset (`fit = FALSE`) or the entire fitted model (secr object; `fit = TRUE`) for each replicate by setting `extractfn = identity`. For a large analysis there is a risk of exceeding memory limits in R, and saving everything is generally not a good idea. For most purposes it is sufficient to save a trimmed version of the fitted model `extractfn = trim` (note **secr** defines the function trim.secr). However, the full model is needed for `derived` or `region.N` (see Extracting estimate tables from fitted models).

Value returned by `run.scenarios()`

`run.scenarios()` sets the class[1] of its output to distinguish among fitted models ("fittedmodels"), estimate tables from predict, coef etc. ("estimatetables"), summaries ("summary"), and numeric values ready for summarisation ("selectedstatistics"). Simulated data saved with `fit = FALSE, extractfn = identity` have class c("rawdata", "secrdesign", "list"). An attribute 'outputtype' is used to make finer distinctions among these types of output ("secrfit", "predicted", "coef", "secrsummary", "capthist", or "numeric"). Output from `extractfn = derived` is treated as "predicted".

Alternative analyses

When `fit = TRUE`, analyses are performed with `fit.function`. Other analyses may be specified by setting `fit = FALSE` and providing the analysis function as the value for `extractfn`. The function should accept a 'capthist' object as input. For example, conventional closed-population estimates may be obtained with

[1] the full class is actually c(x, "secrdesign", "list") where x is as described.

```r
closedNsim <- run.scenarios (nrepl = 10, scenarios = scen1, trapset = traps1,
     extractfn = closedN, estimator = c("null", "chao", "chaomod"))
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed in 0.002 minutes
```

This applies various *nonspatial* estimators to simulated *spatial* samples. Named arguments of `extractfn` may be included (here, 'estimator'); these are used for all scenarios, unlike `fit.args` which may vary among scenarios. Summarisation of alternative analyses will usually require careful selection of 'parameter' and 'statistics' in `select.stats` (see Choosing the statistics to summarise).

An alternative is to write your own code along these lines:

```r
sum1 <- function(out) {
    require(abind)
    ## collapse replicates to an array, omitting non-numeric column
    out <- do.call(abind, c(out, along = 3))[,-1,,drop = FALSE]
    ## convert array from character to numeric
    mode(out) <- "numeric"
    ## take the average over replicates (meaningless for some fields)
    apply(out, 1:2, mean, na.rm = TRUE)
    }
lapply(closedNsim$output, sum1)
```

More on masks

A habitat mask in **secr** is a raster representation of the region near the detectors in which the centres of detected animals may lie. This excludes both nearby non-habitat, and habitat that is so distant that it is implausible any animal centred there would reach a detector. It is often convenient to define a mask to include all cells whose centre is within a certain distance of a detector - the buffer radius.

Within **secrdesign**, a mask is used both when generating populations of animals with `sim.popn` and when fitting SECR models with `secr.fit`. The extent of the mask used to generate populations is important if you are concerned with population size (for example, if you set `extractfn = region.N`). Then the size of the region determines the true value of the parameter of interest (N), and influences its sampling variance. The extent of the mask is less critical when density is the parameter of interest.

The default behaviour of `run.scenarios()` is to use a concave buffer of width `xsigma` $\times$ sigma around the particular detector layout. The 'coarseness' of the mask is determined by `nx`; note that the default for `run.scenarios` (`nx = 32`) is coarser than the default for `secr::make.mask` (`nx = 64`). This makes for speed, and is fairly safe when the buffer width is well matched to the scale of movement (we know sigma, so the default buffer width *is* well-matched). The same mask is used both for generating populations and fitting models.

Users may specify their own masks in the 'maskset' list argument. If the number of masks in maskset is one or a number equal to the number of detector layouts, then a column 'maskindex' is added automatically to the scenarios dataframe (all 1, or equal to trapsindex, in the two cases). Otherwise, the user must have manually added a maskindex column to scenarios to clarify which mask should be used with which scenario.

Summarising simulation output

`run.scenarios` usually takes a long time to run, but having saved its output you can quickly extract and summarise the results in many different ways.

We saw in the previous section and Fig. 1 that the output from `run.scenarios` for each replicate may be a fitted model, a table of parameter estimates, or a numeric vector. Summarisation across replicates (the `summary` method) requires output in 'selected statistics' form, so each of the other forms must be processed first[2]

Look again at Fig. 1: you will see that the primary input to the `summary` function is in the form of selected statistics. A secondary route is to automatically extract statistics from estimates tables, as shown by the dashed line in Fig. 1 – we used this in A simple example. We now address how other forms of output from `run.scenarios` can be processed into 'selected statistics' form.

Extracting estimate tables from fitted models

The methods `predict`, `coef`, `derived` and `region.N` for the 'fittedmodels' class are provided to extract estimates of 'real' parameters from each fitted model. These are direct analogues of `predict`, `coef`, `derived` and `region.N` in **secr**. Here, they apply across all replicates of all scenarios and return an object of class `estimatetables`.

In each case, the result is a dataframe or list of dataframes for each replicate. Rows correspond to estimated parameters (or 'R.N' and 'E.N' for `region.N`) and columns to the respective estimates, standard errors, and confidence limits (with some variations).

The ... argument of the functions `predict`, `coef`, `derived` and `region.N` lets you pass arguments such as `alpha` to the corresponding **secr** function (e.g., `predict.secr` or `derived.secr`).

We can skip this step for the output from our simple example as it is already in 'estimatetables' form.

Choosing the statistics to summarise

Given tabular output from `predict()` or `derived()`, we must select replicate-specific numerical quantities for further summarisation.[3] This is the role of `select.stats()`, which has arguments –

```r
select.stats(object, parameter = "D", statistics)
```

The parameter of interest defaults to density ("D"); others such as "g0" or "sigma" may be substituted,so long as they appear in the input object. To check which parameters are available use

```r
find.param(object)
```

The task of `select.stats()` is to reduce each replicate to a vector of numeric values - we can think of the result as a replicate × values matrix for each scenario (Fig. 2). A later step (see Summary method) computes statistics ('fields') such as mean and SE for each column in the matrix.

[2] Processing happens silently using default settings of `select.stats()` when `summary` is applied directly to 'estimate tables' output.

[3] If `run.scenarios()` has been used with `fit = FALSE`, then the output from each replicate is probably already in the form of selected statistics (the default raw data summaries 'n', 'ndet','nmov' and 'dpa') and `select.stats()` is not relevant. The same may also apply with a user-provided `extractfn` when `fit = TRUE`.

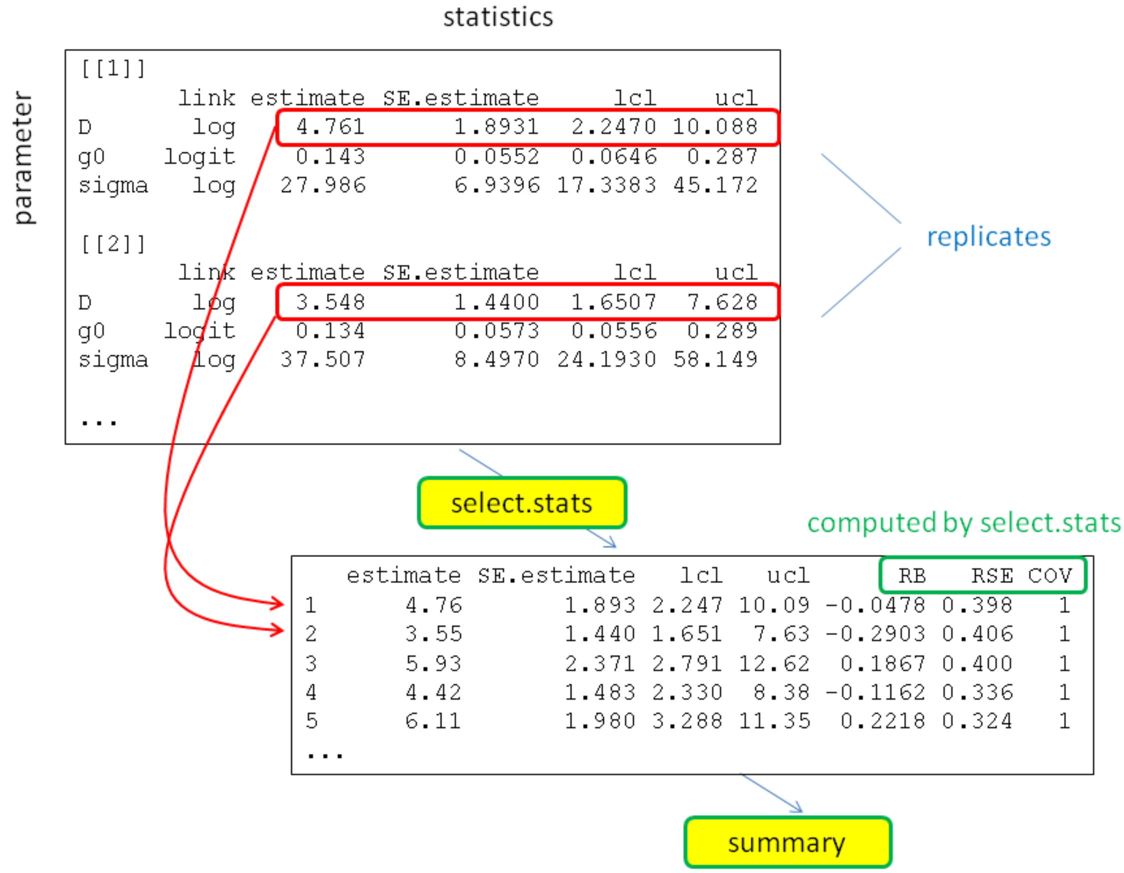


Fig. 2. Operation of `select.stats` for one scenario. Each replicate contributes one row to a replicates × statistics matrix.

Here we describe the replicate-specific statistics that form the numeric vector. These may be simply 'estimate', 'SE.estimate', 'lcl' and 'ucl' as output from `predict.secr`. Additionally, when `fit = TRUE`, we can include statistics derived from the estimates of a parameter (Table 3). To describe these we use 'true' to stand for the known value of a real parameter, and 'estimate' for the estimate from a particular replicate.

Table 3. Computed statistics available in `select.stats()`

Statistic	Short name	Value
Relative bias[1]	RB	(estimate – true) / true
Relative SE[2]	RSE	SE.estimate / estimate
Absolute deviation	ERR	abs(estimate – true)
Coverage indicator	COV	(estimate $>$ lcl) & (estimate $<$ ucl)

1. Also called 'normalized bias'
2. Also called 'coefficient of variation'

We use ('lcl', 'ucl') to represent a confidence interval for 'estimate'. Usually these are 95% intervals, but the level may be varied by setting the argument 'alpha' in `predict` (e.g., alpha = 0.1 for a 90% interval). Intervals from `predict.secr` are symmetrical on the link scale, and hence asymmetrical on the natural scale. Note also the argument `loginterval` in `derived`; the default `loginterval = TRUE` gives an asymmetrical interval on the natural scale.

The coverage indicator COV is a binary value (0/1); this becomes interesting later when averaged over a large number of replicates to give a coverage proportion. The absolute deviation ERR also comes into its own later as the basis for RMSE. In a sense the same is true of replicate-specific RB: RB should be reported only as an average over a large number of replicates.

Returning to our simple example, we apply `select.stats()` to focus on the density parameter "D".

```
stats1 <- select.stats(sims1, parameter = "D", statistics = c("estimate",
   "lcl", "ucl", "RB", "RSE", "COV"))
lapply(stats1$output, head, 4)
```

```
## $`1`
##   estimate      lcl      ucl          RB       RSE COV
## 1 5.151809 3.368615 7.878947  0.03036185 0.2193336   1
## 2 5.855271 4.016080 8.536732  0.17105423 0.1941623   1
## 3 4.482259 2.949923 6.810565 -0.10354827 0.2159011   1
## 4 4.883663 3.270896 7.291630 -0.02326737 0.2066668   1
##
## $`2`
##   estimate      lcl      ucl          RB       RSE COV
## 1 9.409476 7.045668 12.56634 -0.05905243 0.1484146   1
## 2 9.166819 6.787848 12.37956 -0.08331806 0.1542019   1
## 3 9.134215 6.781993 12.30227 -0.08657849 0.1528000   1
## 4 9.165342 6.812591 12.33062 -0.08346581 0.1522293   1
```

The two scenarios yield two replicates $\times$ statistic matrices, from which we display the first 4 rows.

Disposing of rogue values

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. The median (chosen as a field value in `summary`) is recommended as a robust alternative to the mean when there are some extreme estimates. Another way to deal with the problem is to set statistics to NA when a simulation fails.

The function `validate` sets selected 'target' statistics to NA for replicates in which another test statistic is out-of-range or NA:

```
x <- validate (x, test, validrange = c(0, Inf), targets = test)
```

The permissable bounds are usually arbitrary, and the method should be used with care. The keyword "all" may be used for `targets` to indicate all columns.

`validate` accepts a selectedstatistics object (x) as input and returns a modified selectedstatistics object as output. See Learned trap response for an application.

Summary method

The `summary` method for 'selectedstatistics' objects reports both header information on the simulation scenarios and user-selected summaries of the pre-selected statistics.

```
summary (object, dec = 5, fields = c("n", "mean", "se"), alpha = 0.05,
    type = c("list", "dataframe", "array"), ...)
```

Here the summary statistics are called 'fields' to distinguish them from the 'statistics' in each column of the numeric replicate × value matrix for each scenario (see Choosing the statistics to summarise). The task of the summary method is to compute the 'field' value for each 'statistic', summarising across replicates to give a 'statistic' × 'field' matrix for each scenario. The choice of 'fields' is shown in Table 4.

Table 4. Statistic fields available in the `summary` method for selectedstatistics objects.

Field	Description
n	number of non-missing values
mean	mean
se	standard error
sd	sample standard deviation
min	minimum
max	maximum
lcl	lower $100(1-\alpha)$ % confidence limit
ucl	upper $100(1-\alpha)$ % confidence limit
rms	root mean square
median	median
qxxx	xxx/1000 quantile
qyyy	yyy/1000 quantile

The summary fields 'lcl' and 'ucl' are for a simple Wald interval $(\hat\mu + z_{\alpha/2}\widehat{\mathrm{SE}}(\hat\mu), \hat\mu + z_{1-\alpha/2}\widehat{\mathrm{SE}}(\hat\mu))$ where z_α is the 100α-percentile of a standard normal distribution (e.g., $z_{0.975} = 1.96$). [Do not confuse these with the confidence limit statistics of the same name that are symmetrical only on the link scale].

Quantiles are specified as 'qxxx' and 'qyyy' where xxx and yyy are integers between 1 and 999 corresponding to quantiles 0.001 to 0.999. For example, 'q025' refers to the 2.5% quantile.

Applying the 'rms' field to the absolute deviation of an estimate (ERR) provides the root-mean-square-error 'RMSE'.

To recap – a summary value is reported for each combination of a selected statistic, computed for each replicate, and a 'field' that summarises the statistic across replicates, potentially resulting in a table with this structure:

```
              Fields
Statistics     n mean se sd min max lcl ucl rms median q025 q975
  estimate     * *    *  *  *   *   *   *       *      *    *
  SE.estimate  * *    *  *  *   *   *   *       *      *    *
  lcl          * *    *  *  *   *                *      *    *
  ucl          * *    *  *  *   *                *      *    *
  RB           * *    *  *  *   *                *      *    *
  RSE          * *    *  *  *   *   *   *       *      *    *
  ERR          * *    *  *  *   *   *   *   *   *      *    *
  COV          * *    *  *  *   *
```

Cells are left blank for combinations that are unlikely to be meaningful. 'rms' is useful with ERR (i.e. RMSE), but not when applied to other statistics. 'n', 'mean' and 'se' summarise the COV indicator, but other potential summaries are (almost) meaningless.

Apply this to the selected statistics from our simple example:

```r
summary(stats1, c('n', 'mean', 'se', 'median'))
```

```
## run.scenarios(nrepl = 50, scenarios = scen1, trapset = traps1,
##     fit = TRUE, fit.args = list(detectfn = "HHN"), seed = 345)
##
## Replicates    50
## Started       08:24:17 28 Sep 2024
## Run time      0.372  minutes
## Output class  selectedstatistics
##
## $constant
##              value
## trapsindex      1
## noccasions      3
## nrepeats        1
## g0            0.2
## sigma          25
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## fitindex        1
## maskindex       1
##
## $varying
##  scenario  D
##         1  5
##         2 10
##
## $detectors
##  trapsindex trapsname
##           1    traps1
##
## $fit.args
##  fitindex detectfn
##         1      HHN
##
## OUTPUT
##
```

```
## $1
##  1
##             n    mean      se  median
## estimate 50 5.02808 0.15602 5.04640
## lcl      50 3.35008 0.12476 3.29861
## ucl      50 7.56696 0.18881 7.68860
## RB       50 0.00562 0.03120 0.00928
## RSE      50 0.21583 0.00426 0.21136
## COV      50 0.98000 0.02000 1.00000
##
## $2
##  2
##             n     mean      se   median
## estimate 50  9.86995 0.20303  9.91979
## lcl      50  7.38442 0.17085  7.37450
## ucl      50 13.19702 0.23887 13.34369
## RB       50 -0.01301 0.02030 -0.00802
## RSE      50  0.15023 0.00142  0.14992
## COV      50  0.96000 0.02799  1.00000
```

Plot method

Use the plot method to visualize the distributions of 'selectedstatistics' that you have simulated. You may plot either (i) histograms of the selected statistics (`type = "hist"`) or (ii) the estimate and confidence interval for each replicate (`type = "CI"`). One histogram is plotted for each combination of scenario and statistic – you may want to select a subset of scenarios and statistics, and use the graphics options mfcol or mfrow to control the layout. For `type = "CI"` the statistics must include 'estimate', 'lcl' and 'ucl' (or 'beta', 'lcl' and 'ucl' if `outputtype = "coef"`).

```r
par(mfrow = c(2,2))
plot(stats1, type = "hist", statistic = "estimate")
plot(stats1, type = "CI")
```

```
## pdf
##   2
```

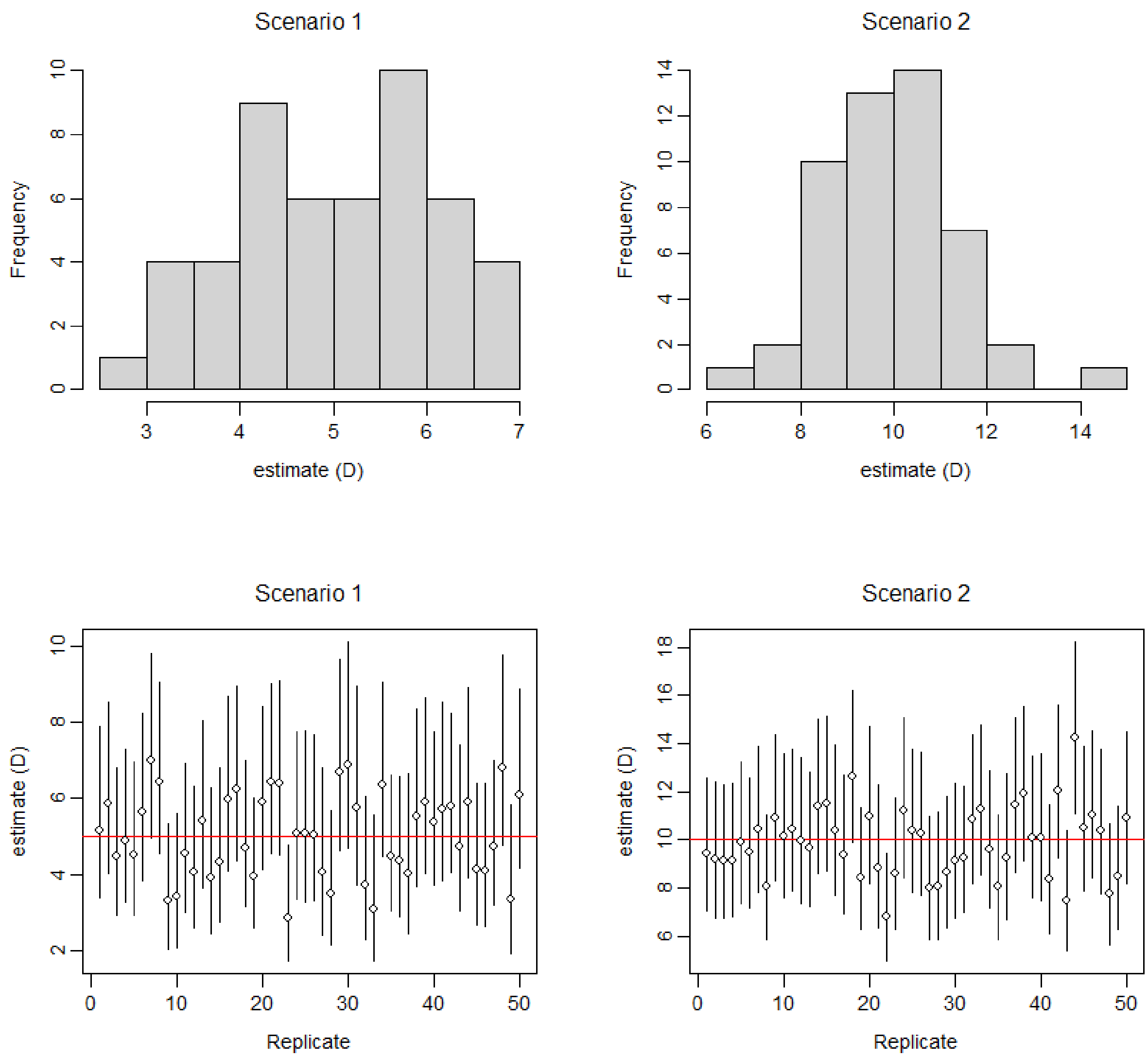


Fig. 3. Plot method applied to a 2-scenario 'selectedstatistics' object with `type = "hist"` (top) and `type = "CI"` (bottom)

Additional topics

Parallel processing

Parallel processing happens in two different ways depending on the argument `byscenarios`.

By default (`byscenario = FALSE`) each call to `secr.fit` uses multi-threaded C++ with the number of threads determined as usual by the environment variable RCPP_PARALLEL_NUM_THREADS (see `secr::setNumThreads`).

Otherwise (`byscenario = TRUE`) each scenario is run on a separate core. This uses the R package **parallel**. Technically, it relies on Rscript, and communication between the master and worker processes is via sockets. As stated in the R **parallel** documentation "Users of Windows and Mac OS X may expect pop-up dialog boxes from the firewall asking if an R process should accept incoming connections". It appears to be safe to accept these.

Use `parallel::detectCores()` to get an idea of how many cores are available on your machine; this may (in Windows) include virtual cores over and above the number of physical cores. If you use the maximum available cores for `run.scenarios()` then expect any other processes on the machine to slow down!

Running one scenario per core is suboptimal if scenarios differ widely in how long they take to run: the system waits for the slowest. Setting `byscenario = FALSE` provides a way around this: replicates within a scenario are divided among cores, instead of all replicates going to one core.

Random number generation for multiple cores uses the "L'Ecuyer-CMRG"" random number generator as described in `?RNG`.

Shortcut evaluation of precision

The asymptotic variance (and hence RSE) of a maximum likelihood estimate is typically obtained from the curvature of the likelihood computed numerically at the fitted value of the parameter(s) (i.e., at the MLE). Fitting SECR models is slow. An alternative estimate of the RSE that is sufficient for most purposes may be got from the curvature of the likelihood computed at the known 'true' value(s) of the parameter(s). This is much faster as it does not require the model to be fitted.

`secr.fit` may be 'tricked' into providing this variance estimate by setting `method = "none"` and providing the true values as the `start` vector. `run.scenarios()` makes this easy by assuming that if you specify `method = "none"` you wish to use `start = "true"`. However, this works only when there is a 1:1 relationship between 'beta' and 'real' parameters; it does not work when 'recapfactor' is specified.

```
sims2 <- run.scenarios(nrepl = 50, trapset = traps1, scenarios = scen1,
    fit = TRUE, fit.args = list(method = "none"))
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed in 0.142 minutes
```

```
summary(sims2)
```

```
## run.scenarios(nrepl = 50, scenarios = scen1, trapset = traps1,
##     fit = TRUE, fit.args = list(method = "none"))
##
## Replicates    50
## Started       08:24:40 28 Sep 2024
## Run time      0.142  minutes
## Output class  selectedstatistics
##
## $constant
##              value
## trapsindex      1
## noccasions      3
## nrepeats        1
## g0            0.2
## sigma          25
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## fitindex        1
## maskindex       1
##
## $varying
```

```
##  scenario  D
##         1  5
##         2 10
##
## $detectors
##  trapsindex trapsname
##           1    traps1
##
## $fit.args
##  fitindex method
##         1   none
##
## OUTPUT
##
## $1
##  1
##                n    mean      se
## estimate    50 5.00000 0.00000
## SE.estimate 50 1.06702 0.00572
## lcl         50 3.30672 0.00705
## ucl         50 7.56210 0.01658
## RB           0      NA      NA
## RSE         50 0.21340 0.00114
## COV          0      NA      NA
##
## $2
##  2
##                n     mean      se
## estimate    50 10.00000 0.00000
## SE.estimate 50  1.48331 0.00397
## lcl         50  7.48916 0.00573
## ucl         50 13.35301 0.01022
## RB           0       NA      NA
## RSE         50  0.14833 0.00040
## COV          0       NA      NA
```

Each estimate of RSE is essentially the same as before (see `summary(stats1)` in Summary method), but the run time is reduced by nearly 80%. Note the true value of density appears as a constant 'estimate' in the summary. Care is needed with this method as its performance in extreme cases has not been investigated fully.

Non-uniform populations

The simulated population by default has a uniform (homogeneous) Poisson distribution. To generate and sample from a spatially inhomogeneous population we use the 'IHP' option for argument `model2D` in `secr::sim.popn`. This involves three steps:

1. Create a habitat mask object with the desired extent.
2. Add to the mask a covariates dataframe with one or more columns defining pixel-specific densities.
3. In `run.scenarios()` specify a list of pop.args including `model2D = "IHP"` and `D = "XX"` where XX is the name of the particular mask covariate you wish to use for density, and name your mask in the 'maskset' argument.

A full demonstration is given in Appendix 2 (Non-uniform possums).

To visualize simulated populations you should set `savepopn = TRUE` in det.args and later extract the popn attribute from the capthist object (for example, with a custom extractfn).

To compare several inhomogeneous distributions, specify several pop.args lists and use the popindex argument in `make.scenarios()`. The distribution may be varied simply by using the `sim.popn()` argument `D` to select different covariates of one mask.

The columns 'nrepeats' and 'D' in the scenarios argument of `run.scenarios` are ignored when `model2D = "IHP"`. 'D' is replaced by the average density over the mask, which is used as the 'true' value of density in computing RB, RSE etc. in summaries. For stratified analyses you will have to define your own extractfn.

Linear habitat

secrdesign may be used to simulate sampling of populations in linear habitats as implemented in R package **secrlinear**. The procedure is similar to that for non-uniform (inhomogeneous Poisson) populations as described in the previous section: one or more masks must be provided, but in this case they will be of type 'linearmask'.

The steps are:

1. Create linear habitat mask objects with the desired extent.
2. Create detector layouts and a scenario dataframe as usual.
3. Add a 'maskindex' column to the scenarios dataframe identifying which mask is to be used in each scenario (may be omitted for a single mask).
4. In `run.scenarios()` specify your mask(s) in the `maskset` argument and optionally override the default Euclidean distances (below).

Density may be specified in the scenario dataframe as a constant number of animals per km, and in this case the 'nrepeats' column is respected.

Density also may be modelled as inhomogeneous, i.e. varying along the length of the linear mask. The mechanism for this is like that for two dimensions: use a list of pop.args including `model2D = "linear"` and `D = "XX"` where XX is the name of the particular mask covariate you wish to use for density. In this case, the columns 'nrepeats' and 'D' in the scenarios dataframe are ignored, as for the 'IHP' option.

With a linear mask, `run.scenarios` defaults to using Euclidean distances[4]. A function to compute non-euclidean distances - usually the **secrlinear** function `networkdistance` - may be specified for data generation (`sim.capthist` argument 'userdist') and model fitting (`secr.fit` argument 'details$userdist'). See example in 'Code for linear habitat'.

Splitting data generation and model fitting

Each new detector layout or new model specification (in a `fit.args` list) defines a new scenario. The default procedure is to generate new data (both animal locations and simulated detection histories) for each scenario. To compare different models applied to the same dataset, save raw data from an initial call to `run.scenarios()` with `fit = FALSE, extractfn = identity`, and separately fit a list of models with `fit.models`. You can also peek at the raw data with the `summary` method.

```r
scen3 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps3 <- make.grid()
raw3 <- run.scenarios(nrepl = 50, trapset = traps3, scenarios =
    scen3, fit = FALSE, extractfn = identity)
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed in 0.005 minutes
```

[4] The default was wrongly stated to be `networkdistance` in previous versions of this document.

```r
summary(raw3)
```

```
## Completed scenario 1

## Completed scenario 2

## Completed in 0.003 minutes

## fit.models(rawdata = object)
##
## Replicates    50
## Started       08:24:49 28 Sep 2024
## Run time      0.003  minutes
## Output class  selectedstatistics
##
## $constant
##             value
## trapsindex      1
## noccasions      3
## nrepeats        1
## g0            0.2
## sigma          25
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## fitindex        1
## maskindex       1
##
## $varying
##  scenario  D
##         1  5
##         2 10
##
## $detectors
##  trapsindex trapsname
##           1    traps1
##
## OUTPUT
##
## $1
##  1
##         n     mean      se
## n    50 15.36000 0.60391
## r    50 13.44000 0.65538
## nmov 50 12.30000 0.63391
## dpa  50  1.78397 0.02972
## rse  50  0.29093 0.00724
## rpsv 50 19.23713 0.34950
##
## $2
##  2
##         n     mean      se
## n    50 27.80000 0.68512
## r    50 24.04000 0.85617
```

```
## nmov 50 21.74000 0.78601
## dpa  50  1.75619 0.02021
## rse  50  0.21059 0.00406
## rpsv 50 19.68918 0.34341
```

```
## fit and summarise models
sims3 <- fit.models(raw3, fit.args = list(list(model = g0~1),
    list(model = g0~T)), fit = TRUE, byscenario = FALSE)
```

```
## Completed scenario 1.1

## Completed scenario 1.2

## Completed scenario 2.1

## Completed scenario 2.2

## Completed in 0.926 minutes
```

```
summary(sims3)
```

```
## fit.models(rawdata = raw3, fit = TRUE, fit.args = list(list(model = g0 ~
##     1), list(model = g0 ~ T)), byscenario = FALSE)
##
## Replicates    50
## Started       08:24:49 28 Sep 2024
## Run time      0.926  minutes
## Output class  selectedstatistics
##
## $constant
##              value
## trapsindex      1
## noccasions      3
## nrepeats        1
## g0            0.2
## sigma          25
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## maskindex       1
##
## $varying
##  scenario  D fitindex
##       1.1  5        1
##       1.2  5        2
##       2.1 10        1
##       2.2 10        2
##
## $detectors
##  trapsindex trapsname
##           1    traps1
##
## $fit.args
##  fitindex  model
##         1 g0 ~ 1
##         2 g0 ~ T
```

```
##
## OUTPUT
##
## $1.1
##  1.1
##               n     mean      se
## estimate    50  5.46452 0.29218
## SE.estimate 50  1.86874 0.05836
## lcl         50  2.87787 0.19784
## ucl         50 10.55931 0.40310
## RB          50  0.09290 0.05844
## RSE         50  0.36939 0.01385
## COV         50  0.90000 0.04286
##
## $1.2
##  1.2
##               n     mean      se
## estimate    50  5.34778 0.29288
## SE.estimate 50  1.83693 0.06191
## lcl         50  2.80468 0.19622
## ucl         50 10.34912 0.41909
## RB          50  0.06956 0.05858
## RSE         50  0.36773 0.01074
## COV         50  0.88000 0.04642
##
## $2.1
##  2.1
##               n     mean      se
## estimate    50  9.58835 0.32383
## SE.estimate 50  2.43988 0.05416
## lcl         50  5.88488 0.24301
## ucl         50 15.69643 0.42323
## RB          50 -0.04117 0.03238
## RSE         50  0.26214 0.00586
## COV         50  0.96000 0.02799
##
## $2.2
##  2.2
##               n     mean      se
## estimate    50  9.52179 0.32507
## SE.estimate 50  2.43182 0.05463
## lcl         50  5.83418 0.24359
## ucl         50 15.61520 0.42566
## RB          50 -0.04782 0.03251
## RSE         50  0.26320 0.00589
## COV         50  0.96000 0.02799
```

Here, `scen3` describes two scenarios, and in the call to `fit.models` each of these is split into two new scenarios, one for each component of `fit.args`.

The arguments 'scen' and 'repl' of `fit.models` let you select particular scenarios and replicates for fitting (**secrdesign** $\geq$ 2.3.0).

It is not possible within **secrdesign** precisely to evaluate the application to the same animal distribution (population) of differing detector layouts or specifications for the fitted model (cf Fewster and Buckland

2004). Comparisons inevitably include variance from the varying number and placement of animals, and the sampling process; this variance may be reduced by fixing the number of individuals (`pop.args = list(Ndist = "fixed")`).

Multi-model inference

You may want to fit different models to the same simulated dataset and compare or combine the results. **secrdesign** provides for this by allowing nested argument lists in 'fit.args', i.e. for each fitindex there may be a list of component lists. `run.scenarios(..., fit = "multifit")` will fit each model separately and return the result as a single 'secrlist' object.

The challenge is to specify a meaningful 'extractfn'. This should accept an 'secrlist' object as input, with other arguments as needed. Obvious possibilities are –

1. Parameter estimates from the model with lowest AIC
2. Model-averaged parameter estimates
3. The entire secrlist, possibly trimmed to save space. Estimates are extracted from each replicate post hoc.

This extractfn computes estimates for (1) (AIC-best model):

```
extractfn1 <- function (scrlist, criterion = 'AIC', ...) {
    aic <- AIC(scrlist, sort = FALSE)
    i <- which.min(aic[,criterion]) # find 'best'
    pred <- predict(scrlist[[i]], ...)
    pred[,'model'] <- names(scrlist)[i]  # add column with name of best model
    pred
}
```

For option (2) use modelAverage(), and for option (3) use identity() or trim() (examples below).

The built-in summary method for estimate tables and the `estimateSummary` function may be used directly on the output from options 1 and 2. These functions expect that the output for each replicate is a dataframe or matrix with one row per parameter and columns 'estimate', 'SE.estimate', 'lcl' and 'ucl' (and possibly other columns).

For demonstration, we simulate data with no learned response and fit both a constant model and one with a learned response in g0.

```
scen1 <- make.scenarios(D = 10, sigma = 25, g0 = 0.2, noccasions = 5)
traps1 <- make.grid(8,8)   ## 8 x 8 grid of multi-catch traps
fitargs1 <- list(
    list(
        list(model = g0~1, buffer = 100),
        list(model = g0~b, buffer = 100)
    )
)
aicbest <- run.scenarios(nrepl = 20, trapset = traps1, scenarios = scen1,
    fit = "multifit", fit.args = fitargs1, extractfn = extractfn1)
```

```
## Completed scenario 1
```

```
## Completed in 0.627 minutes
```

```
estimateSummary(aicbest)
```

```
##   scenario true.D nvalid      EST     seEST           RB       seRB       RSE
## 1        1     10     20 9.979296 0.3493413 -0.002070433 0.03493413 0.1551515
##       RMSE     rRMSE  COV
## 1 1.522884 0.1522884 0.95
```

To show the frequency with which each model was selected we tabulate the model names saved in ancillary column 'model' of each output table:

```
lapply(aicbest$output, function(x) table(sapply(x, '[',1,'model')))
```

```
## $`1`
##
## g0 ~ 1 g0 ~ b
##     15      5
```

Model-averaged estimates are easy. Pass other `modelAverage` arguments as required.

```
# criterion is argument of modelAverage passed in ...
modav <- run.scenarios(nrepl = 20, trapset = traps1, scenarios = scen1,
    fit = "multifit", fit.args = fitargs1, extractfn = modelAverage, criterion = 'AIC')
```

```
## Completed scenario 1
```

```
## Completed in 0.612 minutes
```

```
estimateSummary(modav)
```

```
##   scenario true.D nvalid      EST     seEST           RB       seRB       RSE
## 1        1     10     20 9.973927 0.3273585 -0.002607279 0.03273585 0.1570175
##       RMSE     rRMSE  COV
## 1 1.427161 0.1427161 0.95
```

Post-hoc extraction

Option (3) - saving the fitted model object - gives maximum flexibility. The function `transformOutput` applies an extraction function post hoc, replacing the output of each replicate with the 'extracted' version. To demonstrate, we conduct a set of simulations and try different post-hoc manipulations. There is a `trim` method for 'secrlist' objects from **secr** 4.5.11 onwards, and we use this instead of `identity` to save space.

```
trimmed <- run.scenarios(nrepl = 20, trapset = traps1, scenarios = scen1,
     fit = "multifit", fit.args = fitargs1, extractfn = trim)
```

```
## Completed scenario 1
```

```
## Completed in 0.623 minutes
```

Post-hoc extraction: AIC table for each replicate

```
phoc1 <- transformOutput(trimmed, AIC, criterion = 'AIC', sort = FALSE)
# show a few replicates
phoc1$output[[1]][1:3]
```

```
## [[1]]
##                      model   detectfn npar    logLik      AIC     AICc  dAIC  AICwt
## g0 ~ 1 D~1 g0~1 sigma~1 halfnormal    3 -552.7727 1111.545 1112.045 0.000 0.6891
## g0 ~ b D~1 g0~b sigma~1 halfnormal    4 -552.5687 1113.137 1113.988 1.592 0.3109
##
## [[2]]
##                      model   detectfn npar    logLik     AIC    AICc  dAIC  AICwt
## g0 ~ 1 D~1 g0~1 sigma~1 halfnormal    3 -483.3022 972.604 973.162 0.000 0.6673
## g0 ~ b D~1 g0~b sigma~1 halfnormal    4 -482.9981 973.996 974.948 1.392 0.3327
##
## [[3]]
##                      model   detectfn npar    logLik     AIC    AICc  dAIC AICwt
## g0 ~ 1 D~1 g0~1 sigma~1 halfnormal    3 -390.4459 786.892 787.507 2.165 0.253
## g0 ~ b D~1 g0~b sigma~1 halfnormal    4 -388.3637 784.727 785.780 0.000 0.747
```

```
# which model 'best'?
besti <- function(x) match(0, x$dAIC)
phoc2 <- transformOutput(phoc1, besti)
# frequencies
table(unlist(phoc2$output[[1]]))
```

```
##
##  1  2
## 15  5
```

Post-hoc extraction: compare models

The **secr** function `collate` combines output from multiple models in a single array. We can couple this with the **secrdesign** function `estimateArray` to make a compact table of average simulated estimates of a chosen parameter (`estimateArray` here treats each model as a 'parameter').

```
extractfn2 <- function (x, parameter = 'D') collate(x)[1,,,parameter]
apply(estimateArray(transformOutput(trimmed, extractfn2)), 1:2, mean)
```

```
##
## Parameter  estimate SE.estimate      lcl      ucl
##    g0 ~ 1 10.033452    1.529444 7.456757 13.50653
##    g0 ~ b  9.954981    1.568692 7.325681 13.53449
```

```
# or
apply(estimateArray(transformOutput(trimmed, extractfn2, parameter = 'sigma')), 1:2, mean)
```

```
##
## Parameter estimate SE.estimate      lcl      ucl
##    g0 ~ 1 24.46153    1.591270 21.53786 27.78607
##    g0 ~ b 24.47991    1.593697 21.55195 27.80970
```

Post-hoc extraction: model-averaged densities

```
estimateSummary(transformOutput(trimmed, modelAverage, criterion = 'AIC'))
```

```
##   scenario true.D nvalid      EST     seEST           RB       seRB       RSE
## 1        1     10     20 9.973927 0.3273585 -0.002607279 0.03273585 0.1570175
##       RMSE     rRMSE  COV
## 1 1.427161 0.1427161 0.95
```

Note that the estimates are identical to those obtained directly.

Post-hoc extraction: estimates from 'best' model

We can use the extraction function defined already.

```
estimateSummary(transformOutput(trimmed, extractfn1))
```

```
##   scenario true.D nvalid      EST     seEST           RB       seRB       RSE
## 1        1     10     20 9.979296 0.3493413 -0.002070433 0.03493413 0.1551515
##       RMSE     rRMSE  COV
## 1 1.522884 0.1522884 0.95
```

Post-hoc extraction: coefficients of a specified model

```r
extractfn2 <- function (scrlist, model) coef(scrlist[[model]])
# specify checkfields and validrange to dodge warnings
estimateSummary(transformOutput(trimmed, extractfn2, model = 2),
    true = 0, checkfields = "beta", validrange = c(-10,10),
    parameter = "g0.bTRUE")
```

```
##   scenario true.g0.bTRUE nvalid         EST      seEST  RB seRB      RSE      RMSE
## 1        1             0     20 -0.06843486 0.06953955 NaN  NaN 0.810412 0.3107452
##   rRMSE COV
## 1   Inf   1
```

The method does not work to extract coefficients of the 'best' model because the number of coefficients differs between models.

Populations with sub-classes or multiple sessions

Simulation of structured populations was introduced in **secrdesign** 2.2.0 and is still experimental.

The 'groups' argument of `make.scenarios` replicates rows so that within a scenario there is one row for each group. Group-specific parameter values are inserted by the user.

Rows sharing the same scenario number are recognised by `run.scenarios` as subclasses (groups). Each subclass is generated as a separate capthist object. The argument `multisession` determines whether the capthist objects corresponding to subclasses are pooled for analysis (using `secr::rbind.capthist`) or treated as multiple distinct sessions (using `secr::MS.capthist`).

The original sub-class of each individual is recorded as an individual covariate named "group". This is a factor. It may ignored in the fitted model, or used in such `secr.fit` arguments as 'groups' and 'hcov', or included in models directly as an individual covariate when `CL = TRUE`. (If the output from `predict.secr` is not a single dataframe then you will have to write a custom `extractfn`).

An example is given in Appendix 2 (Grouped populations).

Limitations, tips and troubleshooting

secrdesign has some limitations (Surprise!).

1. A progress message is output only on the completion of each scenario, which can be annoying, and when using multiple cores even this message is lost. It is strongly recommended that you start by generating summaries of raw data only (`run.scenarios()` with `fit = FALSE`), and confirm that your scenarios are realistic by reviewing the simulated number of detected individuals, total number of detections, etc. If these are inadequate or unrealistically large then there's no point going on. Then, try fitting with just a few replicates on one core to be sure you have specified the model you intended and to assess the likely run time. Only then submit a run with a large number of replicates on multiple cores.

2. Only 2-parameter detection functions are allowed for data generation. This excludes the hazard-rate function, the cumulative gamma, and some others.

3. The default `extractfn` does does not handle models that produce more than one estimates table per replicate (e.g., finite mixture models). A custom `extractfn` is needed; it should either produce a numeric vector of 'selected statistics' or mimic single-dataframe output from `predict()`.

4. The function `secr::sim.capthist` that generates detection histories for **secrdesign** has limited capacity for simulating temporal, behavioural or other heterogeneity in detection probability. Heterogeneity may be simulated as discrete subclasses (see preceding section). Only a simple permanent learned response is allowed in `run.scenarios` ('recapfactor').

5. As noted before, the same mask is used for generating populations and fitting models. It would be possible to replace the maskset component of a ‘rawdata’ object before running `fit.models`, but this is not recommended.

6. It is easy to forget the random number seed. Consider replacing the default value.

7. The method for fitting a fixed-N model (`distribution =     binomial`) is somewhat fragile: it can fail when given a start value for D that is less than the minimum density observed (i.e. the number of distinct individuals divided by the mask area). This can easily happen when a population is simulated with `pop.args = list(Ndist = "poisson")` (the default) and sampled with high detection probability, but `secr.fit` is called with (`distribution = "binomial"`). The solution is to use `pop.args = list(Ndist = "fixed")`.

8. If your summaries do not include enough significant digits, increase the ‘dec’ argument of `summary.selectedstatistics`!

References

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

Cooch, E. and White, G. (eds) (2014) *Program MARK: A Gentle Introduction*. 13th edition. Available online at http://www.phidot.org/software/mark/docs/book/.

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

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

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

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

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.

Williams, B. K., Nichols, J. D. and Conroy, M. J. (2002) *Analysis and management of animal populations*. Academic Press, San Diego

Appendix 1. Default extractfn for `run.scenarios`.

```
defaultextractfn <- function(x) {
    if (inherits(x, 'try-error')) {
        ## null output: dataframe of 0 rows and 0 columns
        data.frame()
    }
    else if (inherits(x, 'capthist')) {
        ## summarised raw data
        counts <- function(CH) {
            ## for single-session CH
            if (nrow(CH)==0) { ## 2015-01-24
                if (sighting(traps(CH)))
                    c(n = 0, ndet = 0, nmov = 0, dpa = 0,
                      unmarked=0, nonID = 0, nzero = 0)
                else
                    c(n=0, ndet=0, nmov=0, dpa = NA)
            }
            else {
                n <- nrow(CH)
                ndet <- sum(abs(CH)>0)
                r <- ndet - n
                nmoves <- sum(unlist(sapply(moves(CH), function(y) y>0)))
                ## detectors per animal
                dpa <- if (length(dim(CH)) == 2)
                    mean(apply(abs(CH), 1, function(y) length(unique(y[y>0]))))
                else
                    mean(apply(apply(abs(CH), c(1,3), sum)>0, 1, sum))
                if (sighting(traps(CH))) {
                    unmarked <- if (is.null(Tu <- Tu(CH))) NA else sum(Tu)
                    nonID <- if (is.null(Tm <- Tm(CH))) NA else sum(Tm)
                    nzero <- sum(apply(abs(CH),1,sum) == 0)
                    c(n = n, ndet = ndet, nmov = nmoves, dpa = dpa,
                      unmarked=unmarked, nonID = nonID, nzero = nzero)
                }
                else
                    c(n=n, ndet=ndet, nmov=nmoves, dpa = dpa, rse = sqrt(1/n + 1/r))
            }
        }
        if (ms(x))
            unlist(lapply(x, counts))
        else {
            gp <- covariates(x)$group
            if (is.null(gp))
                counts(x)
            else
                unlist(lapply(split(x,gp,dropnullocc=TRUE), counts))
        }
    }
    else if (inherits(x,'secr') & (!is.null(x$fit))) {
        ## fitted model:
        ## default predictions of 'real' parameters
        out <- predict(x)
```

```
        if (is.data.frame(out))
            out
        else {
            warning ("summarising only first session, group or mixture class")
            out[[1]]
        }
    }
    else
        ## null output: dataframe of 0 rows and 0 columns
        data.frame()
}
```

Appendix 2. Examples

Here we give some annotated examples of simulation code and selected output. Running this code with reduced `nrepl`, and viewing the output, will give you an idea of how `secrdesign` works.

Multiple grids, varying number of occasions

This is the example from the main text, slightly extended

```
traps4 <- list(grid6x6 = make.grid(6,6),
               grid8x9 = make.grid(8,9),
               grid12x12 = make.grid(12,12))
scen4 <- make.scenarios (trapsindex = 1:3, noccasions = c(8,4,2), D = 5,
    g0 = 0.2, sigma = c(20,30), crosstraps = FALSE)

sims4 <- run.scenarios(nrepl = 500, trapset = traps4, scenarios =
     scen4, fit = FALSE)
```

```
## Completed scenario 1

## Completed scenario 2

## Completed scenario 3

## Completed scenario 4

## Completed scenario 5

## Completed scenario 6

## Completed in 0.547 minutes
```

```
class(sims4)        ## just peeking
```

```
## [1] "selectedstatistics" "secrdesign"         "list"
```

```
find.stats(sims4)   ## just peeking
```

```
## [1] "n"    "r"    "nmov" "dpa"  "rse"  "rpsv"
```

```
summary(sims4)
```

```
## run.scenarios(nrepl = 500, scenarios = scen4, trapset = traps4,
##     fit = FALSE)
##
## Replicates    500
## Started       08:26:53 28 Sep 2024
```

```
## Run time      0.547  minutes
## Output class  selectedstatistics
##
## $constant
##              value
## nrepeats        1
## D               5
## g0            0.2
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## fitindex        1
##
## $varying
##  scenario trapsindex noccasions sigma maskindex
##        1          1          8    20         1
##        2          2          4    20         2
##        3          3          2    20         3
##        4          1          8    30         4
##        5          2          4    30         5
##        6          3          2    30         6
##
## $detectors
##  trapsindex trapsname
##          1   grid6x6
##          2   grid8x9
##          3 grid12x12
##
## OUTPUT
##
## $1
##  1
##         n     mean      se
## n    500 14.60800 0.17246
## r    500 36.57400 0.53100
## nmov 500 31.18600 0.47196
## dpa  500  2.79234 0.01970
## rse  500  0.26948 0.00186
## rpsv 500 16.21581 0.07241
##
## $2
##  2
##         n     mean      se
## n    500 21.38400 0.20372
## r    500 27.28200 0.33938
## nmov 500 24.00400 0.30224
## dpa  500  2.06428 0.00937
## rse  500  0.22197 0.00124
## rpsv 500 17.44654 0.08765
##
## $3
##  3
##         n     mean      se
```

```
## n    500 32.47600 0.27147
## r    500 14.38600 0.17440
## nmov 500 12.85200 0.16246
## dpa  500  1.39627 0.00388
## rse  500  0.27160 0.00181
## rpsv 500 18.05341 0.11265
##
## $4
##  4
##         n     mean      se
## n    500 23.06800 0.21610
## r    500 68.55400 0.80957
## nmov 500 62.25200 0.74922
## dpa  500  3.32153 0.01872
## rse  500  0.21202 0.00111
## rpsv 500 21.38099 0.07097
##
## $5
##  5
##         n     mean      se
## n    500 31.58000 0.24203
## r    500 50.34800 0.47648
## nmov 500 46.69600 0.45834
## dpa  500  2.41175 0.00875
## rse  500  0.18003 0.00073
## rpsv 500 23.86144 0.08773
##
## $6
##  6
##         n     mean      se
## n    500 46.07000 0.32065
## r    500 28.63200 0.24505
## nmov 500 26.99600 0.23639
## dpa  500  1.58694 0.00347
## rse  500  0.18955 0.00084
## rpsv 500 25.82948 0.11664
```

```
par(mfrow = c(4,3), cex=0.8, mgp=c(2.2,0.6,0))
plot(sims4, statistic = "n", breaks = seq(0,80,5))      ## animals
plot(sims4, statistic = "nmov", breaks = seq(0,140,5))  ## movements
```

```
## pdf
##   2
```

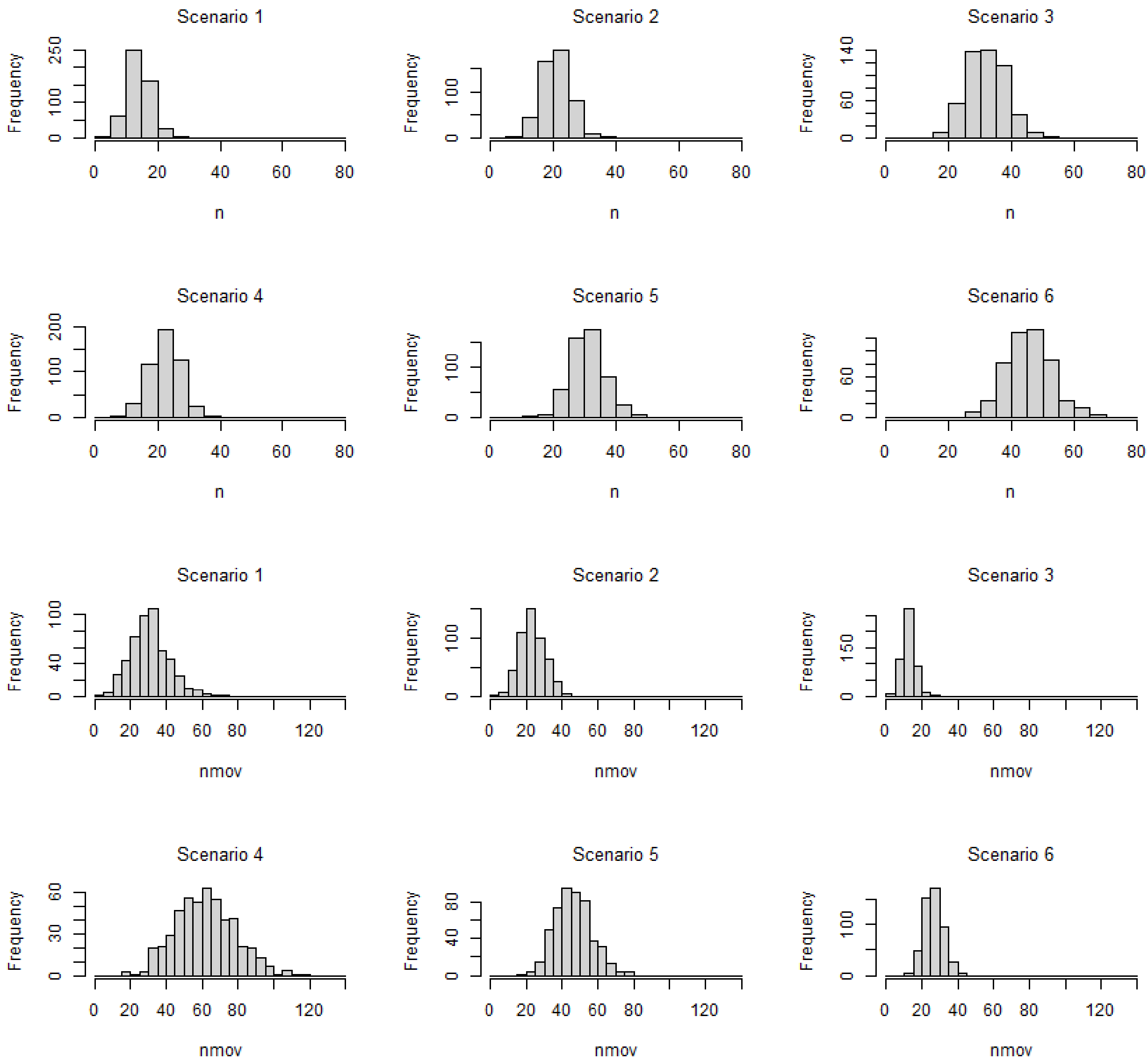


Fig. 4. Numbers of individuals (n) and movements (nmov) from six scenarios differing in trap number, number of sampling occasions and scale of movement.

Learned trap response

Here we assess the bias in $\hat{D}$ caused by ignoring a learned trap response.

```
## set up and run simulations
traps5 <- list(grid6x6 = make.grid(6,6),
               grid10x10 = make.grid(10,10))
scen5 <- make.scenarios (trapsindex = 1:2, noccasions = 5, D = 5,
    g0 = 0.2, sigma = 25, recapfactor = c(0.5, 1, 2), fitindex = 1:2)
sims5 <- run.scenarios(nrepl = 500, trapset = traps5, scenarios =
    scen5, fit = TRUE, fit.args = list(list(model = g0 ~ 1),
    list(model = g0 ~ b)))
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed scenario 3
## Completed scenario 4
## Completed scenario 5
## Completed scenario 6
## Completed scenario 7
## Completed scenario 8
## Completed scenario 9
## Completed scenario 10
## Completed scenario 11
## Completed scenario 12
## Completed in 36.206 minutes
```

```r
## select statistics and throw out any replicates with SE > 100, if any
stats5 <- select.stats(sims5)
stats5 <- validate(stats5, "SE.estimate", c(0,100), "all", quietly = TRUE)
sum5 <- summary(stats5, fields = c("n", "mean", "se", "lcl", "ucl", "median"))
```

```r
## plot
par(mar = c(6,5,6,4), mgp = c(2.2,0.6,0), cex = 0.9)
plot(0,0, xlim = c(0.5,6.5), ylim = c(-0.2,0.4), type = "n",
    xlab = "Scenario", ylab = "RB(D-hat)")
abline(h = 0, col = "red")
text(c(1.5,3.5,5.5), rep(0.38,3), paste("recapfactor", c(0.5,1,2), sep = " = "))
for (i in 1:12) {
    xv <- if (i<=6) i else (i-6)+0.05
    segments (xv, sum5$OUTPUT[[i]]["RB","lcl"], xv, sum5$OUTPUT[[i]]["RB","ucl"])
    ptcol <- if (i<=6) "white" else "black"
    points(xv, sum5$OUTPUT[[i]]["RB","mean"], pch = 21, bg = ptcol)
}
```

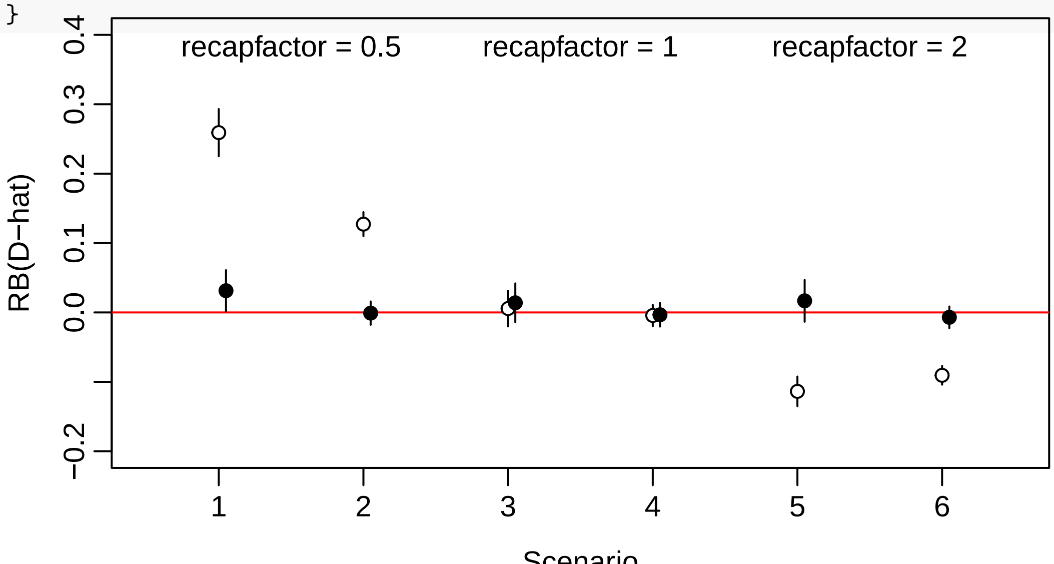


Fig. 5 Relative bias of SECR density estimate from null model (filled circles) and g0∼b model (open circles) when data were generated with negative, zero, or positive learned response.

```r
## look at extended output
sum5
```

```
## run.scenarios(nrepl = 500, scenarios = scen5, trapset = traps5,
##     fit = TRUE, fit.args = list(list(model = g0 ~ 1), list(model = g0 ~
##         b)))
##
## Replicates    500
## Started       08:27:26 28 Sep 2024
## Run time      36.206  minutes
## Output class  selectedstatistics
##
## $constant
##            value
## noccasions     5
## nrepeats       1
## D              5
## g0           0.2
## sigma         25
## detectfn       0
## popindex       1
## detindex       1
##
## $varying
##  scenario trapsindex recapfactor fitindex maskindex
##         1          1         0.5        1         1
##         2          2         0.5        1         2
##         3          1         1.0        1         1
##         4          2         1.0        1         2
##         5          1         2.0        1         1
##         6          2         2.0        1         2
##         7          1         0.5        2         1
##         8          2         0.5        2         2
##         9          1         1.0        2         1
##        10          2         1.0        2         2
##        11          1         2.0        2         1
##        12          2         2.0        2         2
##
## $detectors
##  trapsindex trapsname
##           1   grid6x6
##           2 grid10x10
##
## $fit.args
##  fitindex  model
##         1 g0 ~ 1
##         2 g0 ~ b
##
## OUTPUT
##
## $1
##  1
##                n     mean      se      lcl      ucl   median
## estimate    500  6.29552 0.08679  6.12542  6.46562  6.25012
```

```
## SE.estimate 500  1.97898 0.02515  1.92968  2.02828  1.94886
## lcl         500  3.47292 0.05819  3.35887  3.58697  3.34943
## ucl         500 11.54389 0.14756 11.25467 11.83310 11.30584
## RB          500  0.25910 0.01736  0.22508  0.29312  0.25002
## RSE         500  0.32630 0.00285  0.32072  0.33188  0.31243
## COV         500  0.88200 0.01444  0.85369  0.91031  1.00000
##
## $2
##  2
##               n    mean      se     lcl     ucl  median
## estimate    500 5.63608 0.04443 5.54900 5.72316 5.60457
## SE.estimate 500 1.01264 0.00452 1.00379 1.02150 1.00875
## lcl         500 3.97628 0.03631 3.90510 4.04745 3.92673
## ucl         500 7.99597 0.05315 7.89180 8.10014 7.95666
## RB          500 0.12722 0.00889 0.10980 0.14463 0.12091
## RSE         500 0.18236 0.00074 0.18091 0.18381 0.17994
## COV         500 0.89400 0.01378 0.86699 0.92101 1.00000
##
## $3
##  3
##               n    mean      se      lcl     ucl   median
## estimate    500 5.02798 0.06576  4.89910 5.15685  4.88804
## SE.estimate 500 1.49104 0.01186  1.46780 1.51427  1.46925
## lcl         500 2.85735 0.04644  2.76634 2.94837  2.71620
## ucl         500 8.90484 0.08942  8.72958 9.08011  8.72796
## RB          500 0.00560 0.01315 -0.02018 0.03137 -0.02239
## RSE         500 0.30864 0.00218  0.30436 0.31292  0.30151
## COV         500 0.95600 0.00918  0.93800 0.97400  1.00000
##
## $4
##  4
##               n     mean      se      lcl     ucl   median
## estimate    500  4.97804 0.03955  4.90052 5.05556  4.97705
## SE.estimate 500  0.87977 0.00382  0.87229 0.88725  0.88413
## lcl         500  3.53184 0.03249  3.46816 3.59552  3.51179
## ucl         500  7.02275 0.04694  6.93075 7.11476  7.04853
## RB          500 -0.00439 0.00791 -0.01990 0.01111 -0.00459
## RSE         500  0.17957 0.00076  0.17809 0.18106  0.17747
## COV         500  0.94400 0.01029  0.92383 0.96417  1.00000
##
## $5
##  5
##               n     mean      se      lcl      ucl   median
## estimate    500  4.43140 0.05457  4.32444  4.53836  4.37744
## SE.estimate 500  1.30146 0.00926  1.28332  1.31961  1.30623
## lcl         500  2.53278 0.03943  2.45549  2.61006  2.47702
## ucl         500  7.81033 0.07204  7.66912  7.95153  7.83426
## RB          500 -0.11372 0.01091 -0.13511 -0.09233 -0.12451
## RSE         500  0.30594 0.00236  0.30131  0.31057  0.29766
## COV         500  0.95400 0.00938  0.93562  0.97238  1.00000
##
## $6
##  6
##               n     mean      se      lcl      ucl   median
```

```
## estimate    500  4.54690 0.03447  4.47933  4.61447  4.52938
## SE.estimate 500  0.80922 0.00330  0.80274  0.81569  0.81119
## lcl         500  3.21817 0.02835  3.16262  3.27373  3.19419
## ucl         500  6.42956 0.04086  6.34947  6.50965  6.40852
## RB          500 -0.09062 0.00689 -0.10413 -0.07711 -0.09412
## RSE         500  0.18054 0.00071  0.17914  0.18193  0.17898
## COV         500  0.94000 0.01063  0.91916  0.96084  1.00000
##
## $7
##  7
##               n     mean      se     lcl      ucl  median
## estimate    500  5.15669 0.07556 5.00859  5.30479 5.07499
## SE.estimate 500  1.80534 0.03354 1.73961  1.87107 1.69799
## lcl         500  2.69541 0.04933 2.59873  2.79209 2.61434
## ucl         500 10.18060 0.18971 9.80877 10.55243 9.68346
## RB          500  0.03134 0.01511 0.00172  0.06096 0.01500
## RSE         500  0.36383 0.00513 0.35378  0.37388 0.34096
## COV         500  0.95800 0.00898 0.94040  0.97560 1.00000
##
## $8
##  8
##               n     mean      se      lcl     ucl   median
## estimate    500  4.99510 0.04293  4.91096 5.07923  4.94743
## SE.estimate 500  0.92337 0.00462  0.91431 0.93243  0.91972
## lcl         500  3.48967 0.03476  3.42154 3.55780  3.44574
## ucl         500  7.15777 0.05190  7.05604 7.25950  7.10779
## RB          500 -0.00098 0.00859 -0.01781 0.01585 -0.01051
## RSE         500  0.18803 0.00081  0.18644 0.18961  0.18634
## COV         500  0.94200 0.01046  0.92149 0.96251  1.00000
##
## $9
##  9
##               n    mean      se      lcl     ucl   median
## estimate    500 5.06940 0.07138  4.92949 5.20930  4.91299
## SE.estimate 500 1.63968 0.02210  1.59636 1.68300  1.56140
## lcl         500 2.75671 0.04775  2.66311 2.85030  2.63157
## ucl         500 9.46797 0.12635  9.22033 9.71561  9.10009
## RB          500 0.01388 0.01428 -0.01410 0.04186 -0.01740
## RSE         500 0.33521 0.00330  0.32875 0.34168  0.32095
## COV         500 0.94800 0.00994  0.92852 0.96748  1.00000
##
## $10
##  10
##               n     mean      se      lcl     ucl  median
## estimate    500  4.98327 0.04350  4.89802 5.06852 5.00048
## SE.estimate 500  0.90419 0.00458  0.89523 0.91316 0.90603
## lcl         500  3.50415 0.03531  3.43494 3.57336 3.50390
## ucl         500  7.09441 0.05242  6.99167 7.19716 7.14905
## RB          500 -0.00335 0.00870 -0.02040 0.01370 0.00010
## RSE         500  0.18474 0.00082  0.18314 0.18634 0.18173
## COV         500  0.94000 0.01063  0.91916 0.96084 1.00000
##
## $11
##  11
```

```
##                n    mean      se      lcl      ucl   median
## estimate    500 5.08372 0.07698  4.93284  5.23461  4.92336
## SE.estimate 500 1.67649 0.06734  1.54449  1.80848  1.52631
## lcl         500 2.77419 0.04827  2.67958  2.86880  2.65485
## ucl         500 9.75618 0.39492  8.98215 10.53020  8.90894
## RB          500 0.01674 0.01540 -0.01343  0.04692 -0.01533
## RSE         500 0.33189 0.00456  0.32295  0.34082  0.31478
## COV         500 0.94400 0.01029  0.92383  0.96417  1.00000
##
## $12
##  12
##                n     mean      se      lcl     ucl   median
## estimate    500  4.96505 0.03988  4.88688 5.04322  4.95642
## SE.estimate 500  0.89888 0.00409  0.89087 0.90690  0.90371
## lcl         500  3.49381 0.03259  3.42994 3.55769  3.46651
## ucl         500  7.06266 0.04773  6.96911 7.15621  7.07775
## RB          500 -0.00699 0.00798 -0.02262 0.00864 -0.00872
## RSE         500  0.18387 0.00076  0.18238 0.18536  0.18290
## COV         500  0.94000 0.01063  0.91916 0.96084  1.00000
```

Non-uniform possums

Code to illustrate the use of homogeneous and inhomogeneous density models.

```
## add covariates to builtin secr object possummask
## D1 is homogeneous density
## D2 is artificial SW - NE gradient in density

xy <- apply(possummask,1,sum) / 500
covariates(possummask)[, "D1"] <- 2
covariates(possummask)[, "D2"] <- xy - mean(xy) + 2.5

## Note that this object already had a covariates dataframe
## -- if it didn't we would use
## covariates(possummask) <- data.frame ( D1 = ..., D2 = ...)

## specify scenarios
## anticipate two different sets of arguments for sim.popn
## with popindex = 1:2

scen6 <- make.scenarios (g0 = 0.2, sigma = 45, noccasions = 5,
    popindex = 1:2)

## specify alternate models for distribution of animals

poplist <- list(list(model2D = "IHP", D = "D1", core = possummask),
                list(model2D = "IHP", D = "D2", core = possummask))

## run scenarios and summarise
## we use the trap layout from the builtin secr object possumCH

sims6 <- run.scenarios (500, scen6, traps(possumCH), possummask,
    pop.args = poplist)
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed in 1.231 minutes
```

```r
summary(sims6)
```

```
## run.scenarios(nrepl = 500, scenarios = scen6, trapset = traps(possumCH),
##     maskset = possummask, pop.args = poplist)
##
## Replicates    500
## Started       09:06:24 28 Sep 2024
## Run time      1.231  minutes
## Output class  selectedstatistics
##
## $constant
##              value
## trapsindex      1
## noccasions      5
## nrepeats        1
## g0            0.2
## sigma          45
## detectfn        0
## recapfactor     1
## detindex        1
## fitindex        1
## maskindex       1
##
## $varying
##  scenario   D popindex
##         1 2.0        1
##         2 2.5        2
##
## $detectors
##  trapsindex trapsname
##           1    traps1
##
## $pop.args
##  popindex model2D  D core
##         1     IHP D1 core
##         2     IHP D2 core
##
## OUTPUT
##
## $1
##  1
##         n      mean      se
## n    500 111.86600 0.46103
## r    500 161.02600 0.75242
## nmov 500 143.05000 0.67248
## dpa  500   2.18628 0.00429
## rse  500   0.09485 0.00020
## rpsv 500  34.92702 0.09193
##
## $2
##  2
```

```
##         n      mean      se
## n    500 139.70400 0.48521
## r    500 190.70000 0.73257
## nmov 500 169.37400 0.68597
## dpa  500   2.12850 0.00358
## rse  500   0.08480 0.00015
## rpsv 500  35.11009 0.08984
```

To visualize individual realisations of the distribution of animals, use `fit = FALSE` (the default), `det.args = list(savepopn = TRUE)`, and save the entire capthist object (`extractfn = identity`). Here we create a single replicate.

```
sims6a <- run.scenarios (1, scen6, traps(possumCH), possummask,
    pop.args = poplist, det.args = list(savepopn = TRUE),
    extractfn = identity)
```

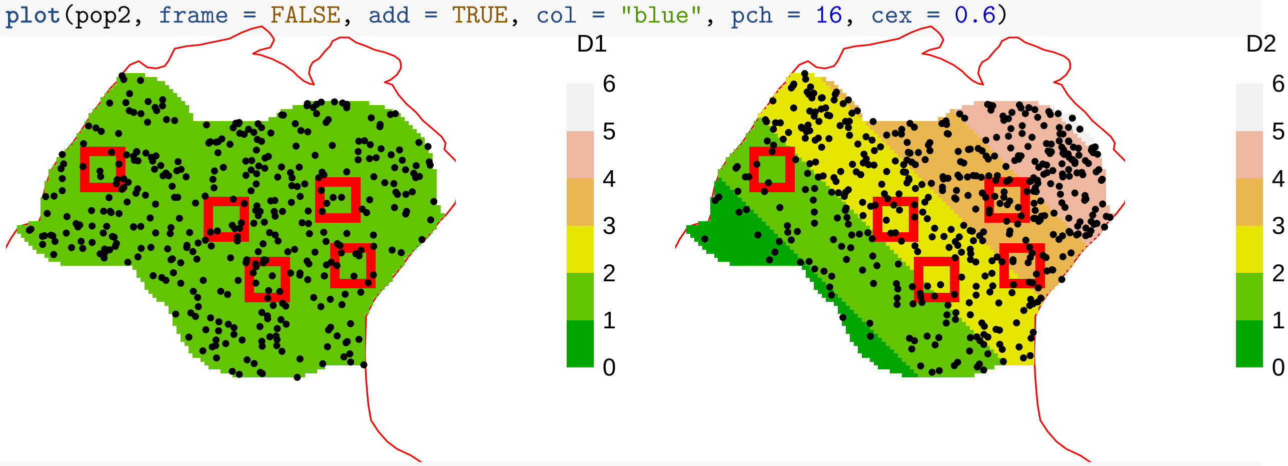
```
## Completed scenario 1

## Completed scenario 2

## Completed in 0.002 minutes
```

```
## sims6a$output is now a list (one component per scenario) of lists
## (one component per replicate) of simulated capthist objects, each
## with its 'popn' object embedded as an attribute

pop1 <- attr(sims6a$output[[1]][[1]], "popn")
pop2 <- attr(sims6a$output[[2]][[1]], "popn")
par(mfrow = c(1,2), mar=c(1,1,1,6))
plot(possummask, covariate = "D1", dots = FALSE, breaks = 0:6)
plot(traps(possumCH), detpar = list(col = 'green', pch = 15), add = TRUE)
plot(pop1, frame = FALSE, add = TRUE, col = "blue", pch = 16, cex = 0.6)
plot(possummask, covariate = 'D2', dots = FALSE, breaks = 0:6)
plot(traps(possumCH), detpar = list(col = 'green', pch = 15), add = TRUE)
plot(pop2, frame = FALSE, add = TRUE, col = "blue", pch = 16, cex = 0.6)
```




```
## click on map to display height; Esc to exit
spotHeight(possummask, prefix = "D2")
```

Fig. 6. Simulated homogeneous (left) and inhomogeneous (right) distributions of brushtail possums at Waitarere, New Zealand. Traps in green (each hollow grid 180 m square).

Code for linear habitat

Code to illustrate the use of linear habitat models. This assumes you have the package **secrlinear**. Some warnings and messages are suppressed.

```
library(secrlinear)
```

```
## create a habitat geometry
x <- seq(0, 4*pi, length = 200)
xy <- data.frame(x = x*100, y = sin(x)*300)
linmask <- read.linearmask(data = xy, spacing = 5)

## define two possible detector layouts
trp1 <- make.line(linmask, detector = 'proximity', n = 80,
                  startbuffer = 200, endbuffer = 200, by = 30)
trp2 <- make.line(linmask, detector = 'proximity', n = 40,
                  startbuffer = 200, endbuffer = 200, by = 60)
trplist <- list(spacing30 = trp1, spacing60 = trp2)

## create a scenarios dataframe
scen7 <- make.scenarios(D = c(50,200), trapsindex = 1:2,
                        sigma = 25, g0 = 0.2)

## we specify a mask, rather than construct it 'on the fly',
## and must manually add column 'maskindex' to the scenarios
scen7$maskindex <- c(1,1)

## we will use a non-Euclidean distance function...
det.arg <- list(userdist = networkdistance)

## run the scenarios and summarise results
sims7 <- run.scenarios(nrepl = 500, trapset = trplist,
    maskset = linmask, det.args = list(det.arg),
    scenarios = scen7, seed = 345, fit = FALSE)
```

```
## Completed scenario 1
```

```
## Completed scenario 2
```

```
## Completed scenario 3
```

```
## Completed scenario 4
```

```
## Completed in 0.939 minutes
```

```
summary(sims7)
```

```
## run.scenarios(nrepl = 500, scenarios = scen7, trapset = trplist,
##     maskset = linmask, det.args = list(det.arg), fit = FALSE,
##     seed = 345)
##
## Replicates    500
## Started       09:07:40 28 Sep 2024
## Run time      0.939  minutes
## Output class  selectedstatistics
##
## $constant
##              value
```

```
## noccasions      3
## nrepeats        1
## g0            0.2
## sigma          25
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## fitindex        1
## maskindex       1
##
## $varying
##  scenario trapsindex   D
##         1          1  50
##         2          2  50
##         3          1 200
##         4          2 200
##
## $detectors
##  trapsindex trapsname
##           1 spacing30
##           2 spacing60
##
## $det.args
##  detindex   userdist
##         1 userdistfn
##
## OUTPUT
##
## $1
##  1
##         n     mean      se
## n    500 90.89600 0.40609
## r    500 61.10800 0.45823
## nmov 500 46.61200 0.38376
## dpa  500  1.44977 0.00294
## rse  500  0.12930 0.00050
## rpsv 500 18.30608 0.06394
##
## $2
##  2
##         n     mean      se
## n    500 59.17200 0.34544
## r    500 15.94000 0.19381
## nmov 500  6.58000 0.12055
## dpa  500  1.10159 0.00174
## rse  500  0.25805 0.00173
## rpsv 500 18.95382 0.14568
##
## $3
##  3
##         n      mean      se
## n    500 357.76000 0.84627
## r    500 242.29000 0.90899
```

```
## nmov 500 185.14600 0.76737
## dpa  500   1.45224 0.00144
## rse  500   0.06441 0.00012
## rpsv 500  18.34929 0.03318
##
## $4
##  4
##         n      mean      se
## n    500 236.12400 0.67737
## r    500  64.23400 0.40979
## nmov 500  26.63600 0.25583
## dpa  500   1.10240 0.00090
## rse  500   0.12576 0.00042
## rpsv 500  19.16660 0.07179
```

Now use a non-Euclidean distance function for fitting as well -

```r
fit.arg <- list(details = list(userdist = networkdistance))

## run the scenarios and summarise results
sims7a <- run.scenarios(nrepl = 500, trapset = trplist,
    maskset = linmask, det.args = list(det.arg), fit = TRUE,
    fit.args = list(fit.arg), scenarios = scen7, seed = 345)
```

```
## Completed scenario 1

## Completed scenario 2

## Completed scenario 3

## Completed scenario 4

## Completed in 58.806 minutes
```

```r
summary(sims7a)
```

```
## run.scenarios(nrepl = 500, scenarios = scen7, trapset = trplist,
##     maskset = linmask, det.args = list(det.arg), fit = TRUE,
##     fit.args = list(fit.arg), seed = 345)
##
## Replicates    500
## Started       09:24:13 24 Oct 2024
## Run time      58.806  minutes
## Output class  selectedstatistics
##
## $constant
##              value
## noccasions       3
## nrepeats         1
## g0             0.2
## sigma           25
## detectfn         0
## recapfactor      1
## popindex         1
## detindex         1
## fitindex         1
## maskindex        1
##
```

```
## $varying
##   scenario trapsindex   D
##          1          1  50
##          2          2  50
##          3          1 200
##          4          2 200
##
## $detectors
##   trapsindex trapsname
##            1 spacing30
##            2 spacing60
##
## $det.args
##   detindex   userdist
##          1 userdistfn
##
## $fit.args
##   fitindex             details
##          1 userdist=userdistfn
##
## OUTPUT
##
## $1
##   1
##                n     mean      se
## estimate    500 51.35351 0.25709
## SE.estimate 500  6.19211 0.03040
## lcl         500 40.58483 0.21484
## ucl         500 64.99592 0.31272
## RB          500  0.02707 0.00514
## RSE         500  0.12091 0.00037
## COV         500  0.96800 0.00788
##
## $2
##   2
##                n     mean      se
## estimate    500 53.05974 0.55032
## SE.estimate 500 12.10066 0.20717
## lcl         500 34.24590 0.30499
## ucl         500 82.70737 1.08049
## RB          500  0.06119 0.01101
## RSE         500  0.22340 0.00173
## COV         500  0.95800 0.00898
##
## $3
##   3
##                n      mean      se
## estimate    500 200.26871 0.54062
## SE.estimate 500  12.03829 0.02966
## lcl         500 178.03124 0.49438
## ucl         500 225.28710 0.59275
## RB          500   0.00134 0.00270
## RSE         500   0.06016 0.00009
## COV         500   0.95600 0.00918
```

```
##
## $4
##  4
##                  n      mean      se
## estimate    500 203.12270 1.02810
## SE.estimate 500  21.86946 0.17145
## lcl         500 164.60725 0.77073
## ucl         500 250.72739 1.38872
## RB          500   0.01561 0.00514
## RSE         500   0.10714 0.00040
## COV         500   0.94000 0.01063
```

Grouped populations

This example demonstrates the simulation of a structured population - nominally females and males with a 2:1 sex ratio.

First we form a scenarios dataframe with 2 groups and two levels of 'noccasions', and manually adjust the 'male' parameter values:

```
scen8 <- make.scenarios (D = 8, g0 = 0.3, sigma = 30, noccasions = c(4,8),
                         groups = c('F','M'))
male <- scen8$group == 'M'
scen8$D[male] <- 4
scen8$g0[male] <- 0.2
scen8$sigma[male] <- 40
scen8[,1:8]
```

```
##   scenario group trapsindex noccasions nrepeats D  g0 sigma
## 1        1     F          1          4        1 8 0.3    30
## 2        1     M          1          4        1 4 0.2    40
## 3        2     F          1          8        1 8 0.3    30
## 4        2     M          1          8        1 4 0.2    40
```

Next we set up a trapping grid, a habitat mask, and a customized extract function for multi-class output from a hybrid mixture model:

```
grid <- make.grid(8, 8, spacing = 30)
mask <- make.mask(grid, buffer = 160, type = 'trapbuffer')
## extracts total density and proportion from output for the first group (F)
exfn <- function(x) {
    if (inherits(x, 'secr') & !is.null(x$fit)) {
        pred <- predict(x)
        pred[[1]][c('D','pmix'),]
    }
    else data.frame()
}
```

It is desirable to check the raw simulations. We specify the mask, rather than relying on one constructed automatically, to ensure the same mask is used for both females and males.

```
raw8 <- run.scenarios(20, scen8, trapset = grid, fit = FALSE, maskset = mask)
```

```
## Completed scenario 1

## Completed scenario 2

## Completed in 0.029 minutes
```

```r
summary(raw8)
```

```
## run.scenarios(nrepl = 20, scenarios = scen8, trapset = grid,
##     maskset = mask, fit = FALSE)
##
## Replicates    20
## Started       09:08:37 28 Sep 2024
## Run time      0.029  minutes
## Output class  selectedstatistics
##
## $constant
##             value
## trapsindex      1
## nrepeats        1
## detectfn        0
## recapfactor     1
## popindex        1
## detindex        1
## fitindex        1
## maskindex       1
##
## $varying
##  scenario group noccasions D  g0 sigma
##         1     F          4 8 0.3    30
##         1     M          4 4 0.2    40
##         2     F          8 8 0.3    30
##         2     M          8 4 0.2    40
##
## $detectors
##  trapsindex trapsname
##           1    traps1
##
## OUTPUT
##
## $1
##  1
##          n      mean      se
## F.n    20  75.20000 1.36034
## F.r    20 115.05000 2.97487
## F.nmov 20  99.45000 2.54277
## F.dpa  20   2.22321 0.02309
## F.rse  20   0.11560 0.00111
## F.rpsv 20  25.42728 0.23559
## M.n    20  46.85000 1.36358
## M.r    20  71.35000 2.75516
## M.nmov 20  65.80000 2.65924
## M.dpa  20   2.35099 0.03109
## M.rse  20   0.14698 0.00216
## M.rpsv 20  32.86123 0.47589
##
## $2
##  2
##          n      mean      se
## F.n    20  87.45000 2.67392
```

```
## F.r    20 313.75000 9.95249
## F.nmov 20 270.85000 9.20505
## F.dpa  20   3.48190 0.04598
## F.rse  20   0.10772 0.00179
## F.rpsv 20  25.64669 0.18815
## M.n    20  53.25000 1.38007
## M.r    20 173.55000 5.99626
## M.nmov 20 158.30000 5.61253
## M.dpa  20   3.55734 0.05052
## M.rse  20   0.13769 0.00178
## M.rpsv 20  32.24355 0.36042
```

Now fit the models…

```
sims8 <- run.scenarios(20, scen8, trapset = list(grid), fit = TRUE, extractfn = exfn,
                       fit.args = list(model = list(g0~h2, sigma~h2), hcov = 'group'),
                       maskset = list(mask), byscenario = FALSE)
```

```
## Warning in log(pdpmix/sum(pdpmix)): NaNs produced

## Warning in log(pdpmix/sum(pdpmix)): NaNs produced
## Completed scenario 1
## Warning in log(pdpmix/sum(pdpmix)): NaNs produced
## Completed scenario 2
## Completed in 18.632 minutes
```

… and check the summary output for density ('D') and sex ratio (proportion female 'pmix') without repeating the header information:

```
summary(select.stats(sims8,'D'))$OUTPUT
```

```
## $`1`
##              n     mean      se
## estimate    20 11.89786 0.19096
## SE.estimate 20  1.16466 0.00947
## lcl         20  9.82556 0.17356
## ucl         20 14.40801 0.20827
## RB          20 -0.00851 0.01591
## RSE         20  0.09814 0.00087
## COV         20  1.00000 0.00000
##
## $`2`
##              n     mean      se
## estimate    20 12.02259 0.26420
## SE.estimate 20  1.05365 0.01200
## lcl         20 10.12878 0.24109
## ucl         20 14.27142 0.28754
## RB          20  0.00188 0.02202
## RSE         20  0.08803 0.00096
## COV         20  0.95000 0.05000
```

```
summary(select.stats(sims8,'pmix'))$OUTPUT
```

```
## $`1`
##              n    mean      se
```

```
## estimate    20 0.66262 0.00782
## SE.estimate 20 0.04568 0.00051
## lcl         20 0.56816 0.00820
## ucl         20 0.74563 0.00698
## RB           0      NA      NA
## RSE         20 0.06924 0.00146
## COV          0      NA      NA
##
## $`2`
##              n    mean      se
## estimate    20 0.66541 0.01074
## SE.estimate 20 0.04047 0.00069
## lcl         20 0.58209 0.01161
## ucl         20 0.73960 0.00939
## RB           0      NA      NA
## RSE         20 0.06142 0.00208
## COV          0      NA      NA
```