secrdesign - sampling design for spatially explicit capture–recapture

Murray Efford

2015-01-11

Contents

Introduction **2**

A simple example . . . 3

Defining scenarios **4**

Detector layouts . . . 4

The scenarios dataframe . . . 4

Running simulations **6**

Arguments of `run.scenarios()` . . . 6

Customising run.scenarios() . . . 7

More on masks . . . 8

Summarising simulation output **9**

Extracting estimate tables from fitted models . . . 9

Choosing the statistics to summarise . . . 10

Disposing of rogue values . . . 12

Summary method . . . 12

Plot method . . . 14

Additional topics **15**

Parallel processing . . . 15

Shortcut evaluation of precision . . . 16

Non-uniform populations . . . 17

Linear habitat . . . 17

Splitting data generation and model fitting . . . 18

Populations with sub-classes or multiple sessions . . . 18

Limitations, tips and troubleshooting **19**

References **20**

Appendix. Examples **21**

Multiple grids, varying number of occasions 21

Learned trap response 23

Non-uniform possums 27

Code for linear habitat 30

Grouped populations 32

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). Other tools may be added in future.

This document is a technical guide to **secrdesign**. It assumes an understanding of estimator properties such as bias, precision, and confidence interval coverage, and the use of Monte Carlo simulation to predict the frequentist 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.

Introduction

When designing a study we assume you 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 can proceed to use Monte Carlo simulation in **secrdesign** to evaluate how alternative sampling scenarios perform with respect to your chosen criterion.

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. Simulations are specified by 'scenarios' stored in a dataframe. The scenario dataframe will usually be constructed with `make.scenarios`. You may construct it manually, but there is a rigid list of required columns (see The scenarios dataframe).

Typically, you will (i) construct detector array(s), (ii) construct a dataframe of scenarios, (iii) use `run.scenarios()` to generate data and fit SECR models, (iv) select some statistics with `select.stats()`, and (v) summarise and plot the results.

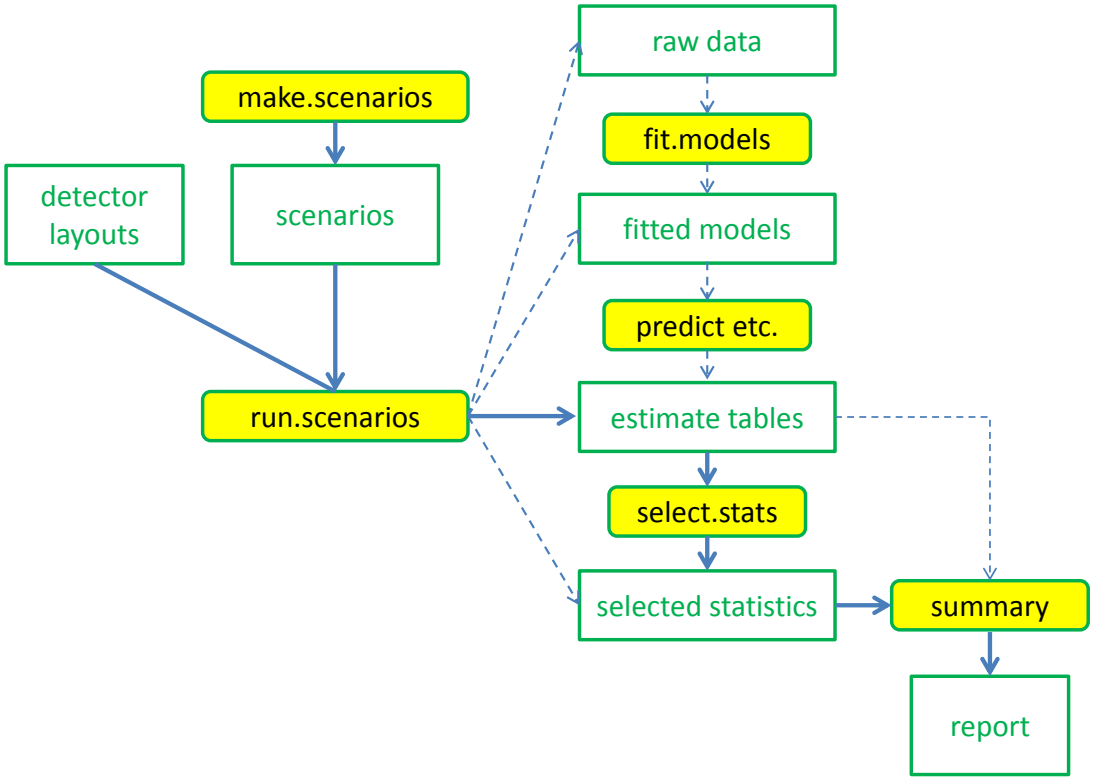

Fig. 1. Core functions in **secrdesign** (yellow) and their main inputs and outputs. Output from the simulation function `run.scenarios()` may be saved as whole fitted models, predicted values (parameter estimates), or selected statistics. Each form of output requires different subsequent handling. The 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 g0 are fixed.

```
library(secrdesign)
scen1 <- make.scenarios(D = c(5,10), sigma = 25, g0 = 0.2)
traps1 <- make.grid()
sims1 <- run.scenarios(nrepl = 50, trapset = traps1, scenarios =
     scen1, seed = 345, fit = TRUE)
```

The 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
```

```
## $`D = 5`
```

```
##              n     mean      se
## estimate    50  5.23767 0.24994
## SE.estimate 50  1.84409 0.05958
## lcl         50  2.70152 0.16476
## ucl         50 10.28534 0.37589
## RB          50  0.04753 0.04999
## RSE         50  0.36890 0.00962
## COV         50  0.92000 0.03876
##
## $`D = 10`
##              n     mean      se
## estimate    50  9.58795 0.36972
## SE.estimate 50  2.45915 0.05721
## lcl         50  5.86410 0.27453
## ucl         50 15.75842 0.47775
## RB          50 -0.04121 0.03697
## RSE         50  0.26727 0.00668
## COV         50  0.92000 0.03876
```

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)
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, det.args, fit = FALSE, fit.args, extractfn =
    NULL, multisession = FALSE, ncores = 1, seed = 123)
```

`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` × `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.

`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` and `ncores` is discussed under Additional topics.

Customising run.scenarios()

The output from `run.scenarios()` is controlled by its argument `extractfn`. This is a short function that is applied either (i) to each simulated raw dataset (capthist object) (`fit = FALSE`), or (ii) to each fitted model (`fit = TRUE`).

The default (builtin) `extractfn` (below) behaves appropriately for either data type. It is mostly concerned with summarising raw counts when fit = FALSE. A dataframe with no rows is returned when a model fails to fit.

```
extractfn  <- function(x) {
    if (inherits(x, 'capthist')) {
        ## summarised raw data
        counts <- function(CH) {
            ## for single-session CH
            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))
            c(n=nrow(CH), ndet=sum(abs(CH)>0), nmov=nmoves, dpa = dpa)
        }
        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
        predict(x)
    else
        ## null output: dataframe of 0 rows and 0 columns
        data.frame()
}
```

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). Nearly the same is achieved by setting `extractfn = predict` for the 'beta' coefficients set `extractfn = coef`. For a conditional likelihood fit it may be appropriate to set `extractfn = derived`. To focus on population size in the masked region, set `extractfn = region.N`.

The user may also 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). However, the full model is needed for `derived.SL` or `regionN.SL` (see Extracting estimate tables from fitted models).

`run.scenarios()` sets the class[1] of its output to distinguish among fitted models ("fittedmodels"), estimate tables from predict, coef etc. ("estimatetables"), and numeric values ready for summarisation ("selectd-statistics"). 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", "capthist", or "numeric"). Output from `extractfn = derived` is treated as "predicted".

When `fit = TRUE`, analyses are performed with `secr.fit`. 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

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

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:

```
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

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

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

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 `regionN.SL`) and columns to the respective estimates, standard errors, and confidence limits (with some variations).

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

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

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

Choosing the statistics to summarise

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

```
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

```
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.

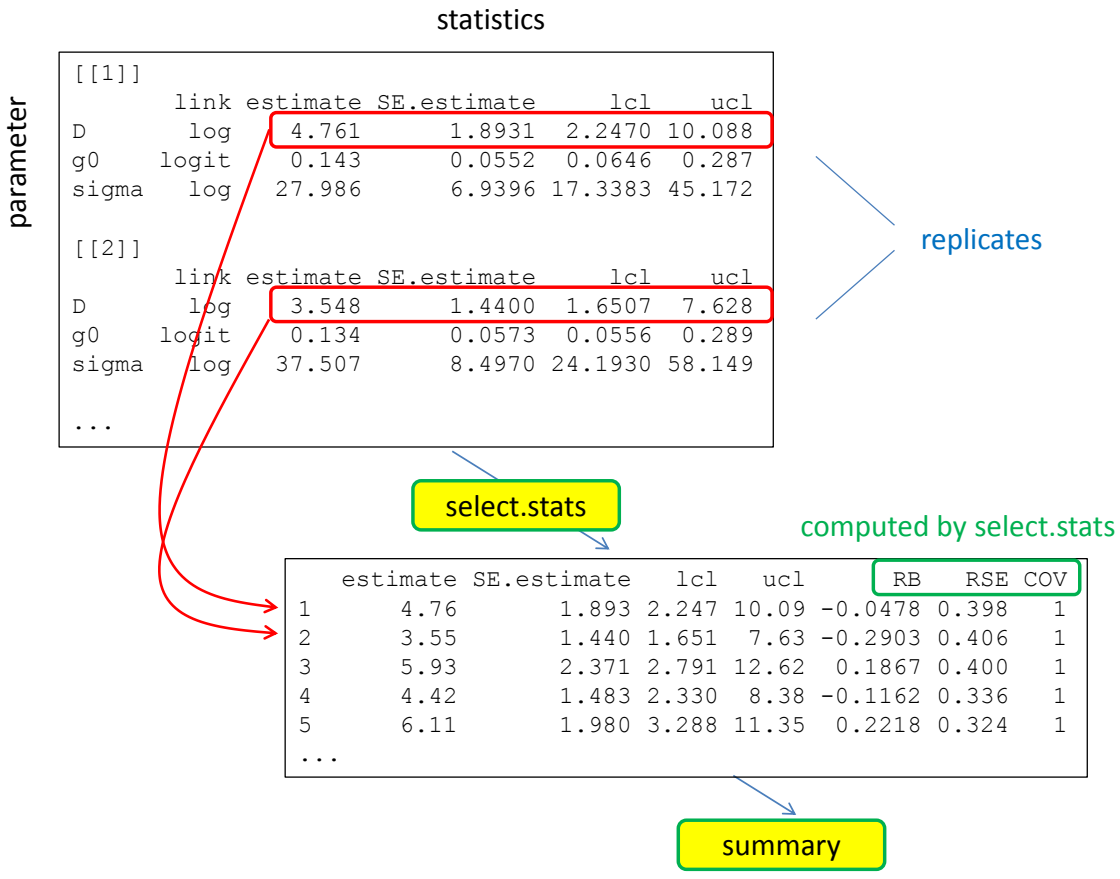

Fig. 2. Operation of `select.stats` for one scenario. Each replicate contributes one row to a replicates × statistics matrix.

[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`.

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 1). 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 1. 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    4.563 2.320 8.976 -0.0873 0.3557   1
## 2    3.626 1.767 7.439 -0.2749 0.3794   1
## 3    3.861 1.838 8.109 -0.2279 0.3926   1
## 4    4.455 2.396 8.285 -0.1090 0.3246   1
##
## $`2`
##   estimate    lcl   ucl       RB    RSE COV
## 1   16.330 11.211 23.79  0.63297 0.1937   0
## 2    9.601  5.695 16.19 -0.03992 0.2713   1
## 3   10.306  6.417 16.55  0.03060 0.2453   1
## 4   12.629  8.371 19.05  0.26288 0.2122   1
```

The two scenarios yield two replicates × 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 2.

Table 2. 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:

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

```
## run.scenarios(nrepl = 50, scenarios = scen1, trapset = traps1,
##     fit = TRUE, seed = 345)
##
## Replicates    50
## Started       19:07:47 04 Dec 2014
## Run time      1.795  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
## $`D = 5`
##              n     mean      se   median
## estimate 50  5.23767 0.24994  5.04535
## lcl      50  2.70152 0.16476  2.50031
## ucl      50 10.28534 0.37589 10.01498
## RB       50  0.04753 0.04999  0.00907
## RSE      50  0.36890 0.00962  0.36154
## COV      50  0.92000 0.03876  1.00000
##
## $`D = 10`
##              n     mean      se   median
## estimate 50  9.58795 0.36972  9.86434
## lcl      50  5.86410 0.27453  5.91509
## ucl      50 15.75842 0.47775 16.14298
## RB       50 -0.04121 0.03697 -0.01357
## RSE      50  0.26727 0.00668  0.25726
## COV      50  0.92000 0.03876  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"`).

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

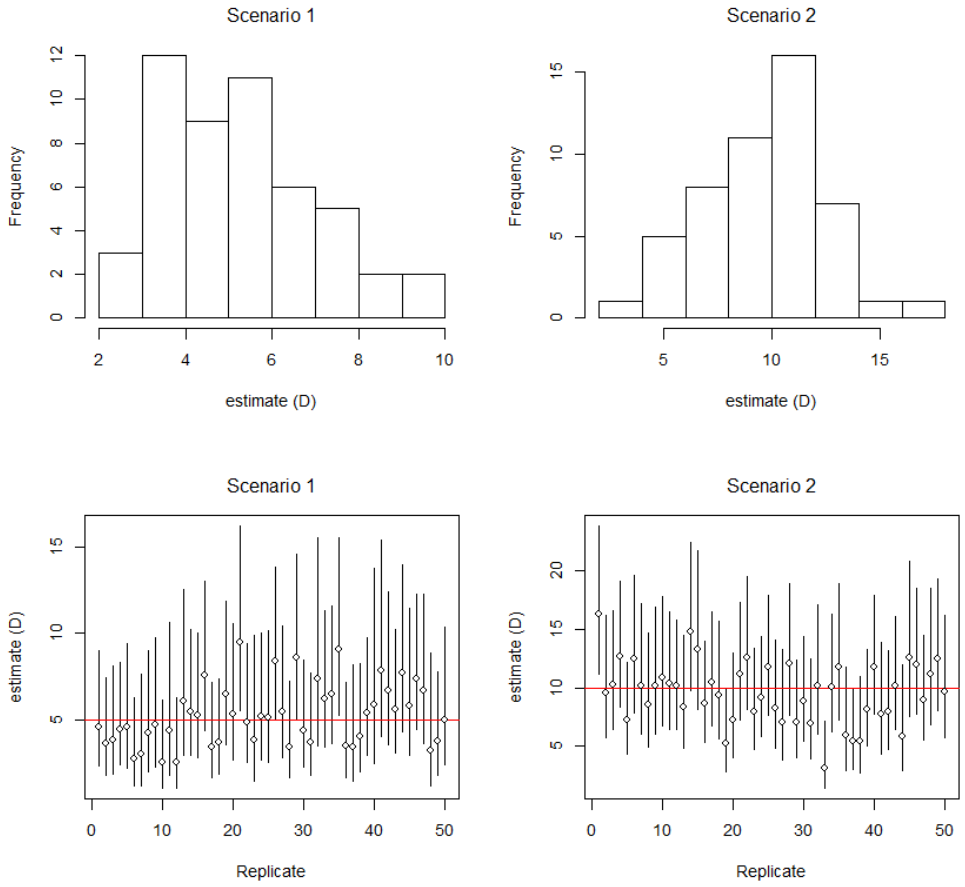

Fig. 3. Plot method applied to a 2-scenario 'selectedstatistics' object with `type = "hist"` (top) and `type = "CI"` (bottom)

Additional topics

Parallel processing

Setting `ncores > 1` causes `run.scenarios()` to run separate scenarios on separate cores. 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. There is no way around this limitation in **secrdesign**.

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"))
```

```
summary(sims2)
```

```
## run.scenarios(nrepl = 50, scenarios = scen1, trapset = traps1,
##     fit = TRUE, fit.args = list(method = "none"))
##
## Replicates    50
## Started       19:09:35 04 Dec 2014
## Run time      0.378  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
```

```
## $`D = 5`
##                n    mean      se
## estimate      50  5.0000 0.00000
## SE.estimate   49  1.8888 0.04280
## lcl           49  2.4603 0.03457
## ucl           49 10.2701 0.16263
## RB             0      NA      NA
## RSE           49  0.3777 0.00856
## COV            0      NA      NA
##
## $`D = 10`
##                n    mean      se
## estimate      50 10.0000 0.00000
## SE.estimate   50  2.6670 0.05948
## lcl           50  6.0032 0.06291
## ucl           50 16.7560 0.19230
## RB             0      NA      NA
## RSE           50  0.2667 0.00595
## 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 the Appendix (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 `masksset` argument.

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 `secrlinear::networkdistance` for the distance function (`secr.fit` argument `details$userdist`).

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.

```
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)
summary(raw3)
## fit and summarise models
sims3 <- fit.models(raw3, fit.args = list(list(model = g0~1),
    list(model = g0~T)), fit = TRUE, ncores = 4)
summary(sims3)
```

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`.

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")`).

Populations with sub-classes or multiple sessions

Simulation of structured populations is 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 the Appendix (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 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.

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

Efford, M. G., Borchers D. L. and Byrom, A. E. (2009) Density estimation by spatially explicit capture–recapture: likelihood-based methods. In: D. L. Thomson, E. G. Cooch, 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. 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, ncores = 3)
```

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

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

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

```
## [1] "n"    "ndet" "nmov" "dpa"
```

```
summary(sims4)
```

```
## run.scenarios(nrepl = 500, scenarios = scen4, trapset = traps4,
##     fit = FALSE, ncores = 3)
##
## Replicates    500
## Started       19:15:01 04 Dec 2014
## Run time      0.34  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
## $`trapsindex = 1, noccasions = 8, sigma = 20, maskindex = 1`
##        n   mean      se
## n    500 14.378 0.17151
## ndet 500 50.546 0.68825
## nmov 500 30.782 0.48020
## dpa  500  2.785 0.01952
##
## $`trapsindex = 2, noccasions = 4, sigma = 20, maskindex = 2`
##        n   mean      se
## n    500 21.342 0.20958
## ndet 500 48.200 0.52306
## nmov 500 23.542 0.30967
## dpa  500  2.045 0.00915
##
## $`trapsindex = 3, noccasions = 2, sigma = 20, maskindex = 3`
##        n   mean      se
## n    500 32.404 0.25352
## ndet 500 46.904 0.38394
## nmov 500 12.860 0.15695
## dpa  500  1.397 0.00382
##
## $`trapsindex = 1, noccasions = 8, sigma = 30, maskindex = 4`
##        n   mean      se
## n    500 23.190 0.20912
## ndet 500 92.086 0.97648
## nmov 500 62.306 0.75562
## dpa  500  3.327 0.01948
##
## $`trapsindex = 2, noccasions = 4, sigma = 30, maskindex = 5`
##        n   mean      se
## n    500 31.648 0.23747
## ndet 500 82.594 0.67822
## nmov 500 47.164 0.44615
## dpa  500  2.422 0.00874
##
## $`trapsindex = 3, noccasions = 2, sigma = 30, maskindex = 6`
##        n  mean      se
## n    500 45.41 0.29922
## ndet 500 73.90 0.50501
## nmov 500 26.77 0.23381
## dpa  500  1.59 0.00347
```

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

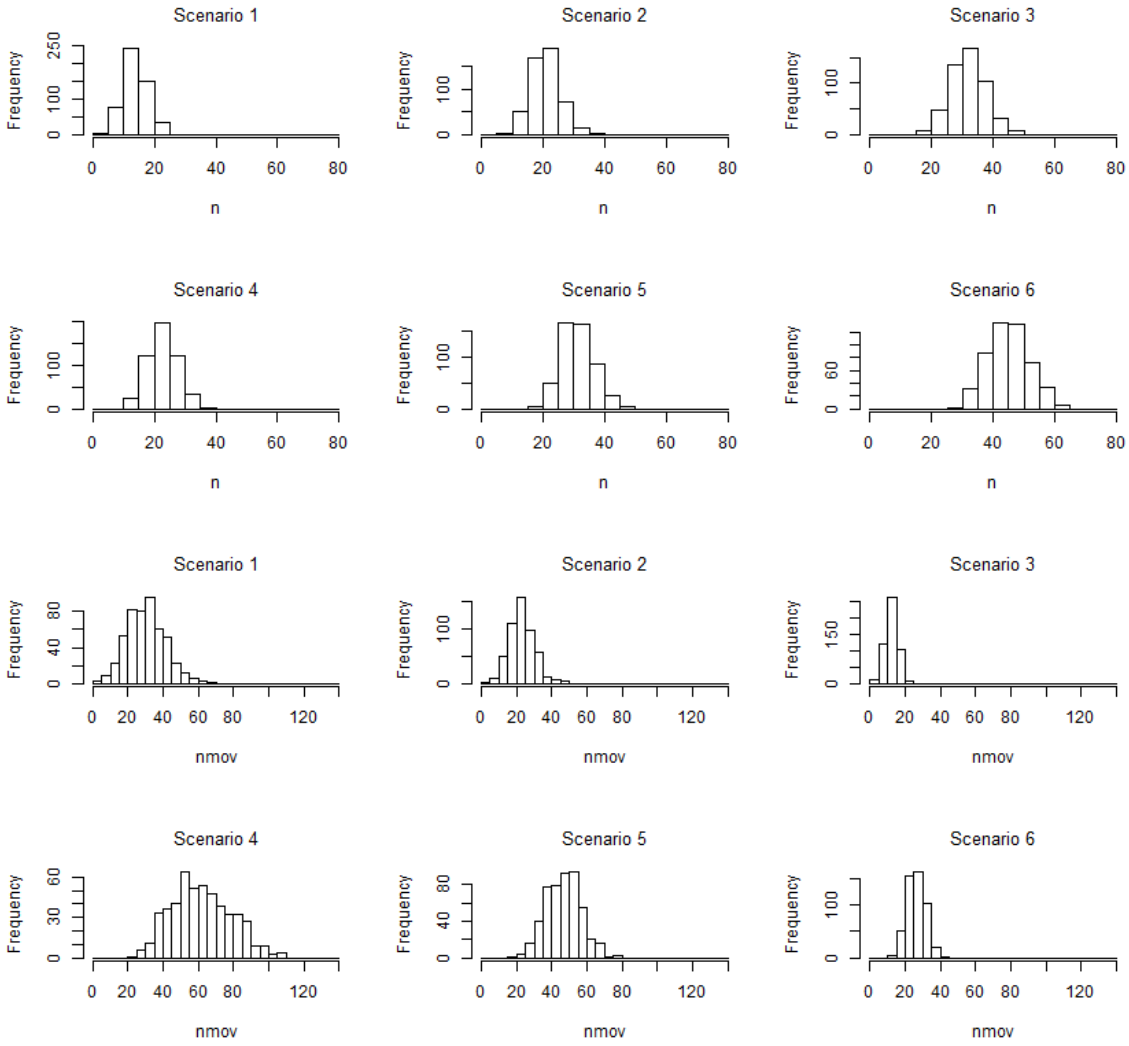

Fig. 4. Numbers of individuals (n) and movements (nmov) from six scenarios differing in trap number, number of sampling occasions and scale of movement.

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)), ncores = 6)
```

```
## select statistics and throw out any replicates with SE > 100
## (there is one -- see reduced n in output for scenario 11)
stats5 <- select.stats(sims5)
stats5 <- validate(stats5, "SE.estimate", c(0,100), "all")
sum5 <- summary(stats5, fields = c("n","mean","se","lcl","ucl", "median"))
```

```
## plot
plot(c(0.5,6.5), c(-0.2,0.4), type = "n", xlab = "Scenario", ylab = "RB(D-hat)")
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)
}
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 = " = "))
```

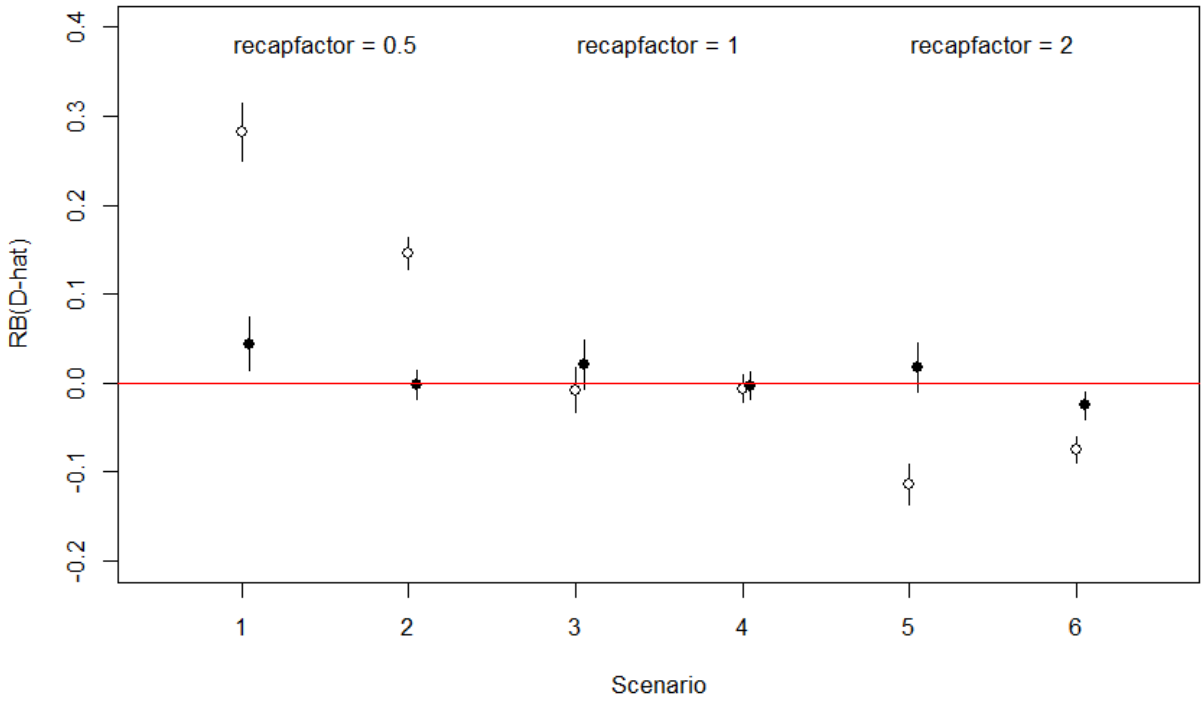

Fig. 5 Relative bias of SECR density estimate from null model (filled circles) and g0∼b model (open circles) when data were generated with negative, zero, or positive learned response.

```
## 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)), ncores = 6)
##
## Replicates    500
## Started       19:15:21 04 Dec 2014
## Run time      104.8  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
## $`trapsindex = 1, recapfactor = 0.5, fitindex = 1, maskindex = 1`
##                n    mean      se     lcl     ucl  median
## estimate     500  6.4126 0.08204  6.2518  6.5733  6.2543
## SE.estimate  500  1.9965 0.02054  1.9562  2.0368  1.9330
## lcl          500  3.5507 0.05517  3.4426  3.6589  3.3630
## ucl          500 11.6784 0.12692 11.4296 11.9271 11.3254
## RB           500  0.2825 0.01641  0.2504  0.3147  0.2509
## RSE          500  0.3214 0.00251  0.3165  0.3263  0.3132
## COV          500  0.8600 0.01553  0.8296  0.8904  1.0000
##
## $`trapsindex = 2, recapfactor = 0.5, fitindex = 1, maskindex = 2`
##                n   mean      se    lcl    ucl median
## estimate     500 5.7277 0.04459 5.6403 5.8151 5.6665
## SE.estimate  500 1.0221 0.00456 1.0132 1.0311 1.0175
## lcl          500 4.0501 0.03642 3.9787 4.1215 4.0027
## ucl          500 8.1067 0.05348 8.0019 8.2115 8.0313
```

```
## RB          500 0.1455 0.00892 0.1280 0.1630 0.1333
## RSE         500 0.1809 0.00069 0.1796 0.1823 0.1794
## COV         500 0.8800 0.01455 0.8515 0.9085 1.0000
##
## $`trapsindex = 1, recapfactor = 1, fitindex = 1, maskindex = 1`
##                n     mean      se      lcl     ucl   median
## estimate    500  4.96161 0.06381  4.83654 5.08669  4.98843
## SE.estimate 500  1.47525 0.01175  1.45223 1.49828  1.47518
## lcl         500  2.81512 0.04459  2.72772 2.90251  2.78540
## ucl         500  8.79952 0.08750  8.62803 8.97102  8.84495
## RB          500 -0.00768 0.01276 -0.03269 0.01734 -0.00231
## RSE         500  0.30992 0.00240  0.30521 0.31463  0.29930
## COV         500  0.95000 0.00976  0.93088 0.96912  1.00000
##
## $`trapsindex = 2, recapfactor = 1, fitindex = 1, maskindex = 2`
##                n    mean      se      lcl     ucl   median
## estimate    500  4.9675 0.03917  4.89071 5.04426  4.92346
## SE.estimate 500  0.8810 0.00375  0.87364 0.88832  0.88014
## lcl         500  3.5202 0.03224  3.45698 3.58334  3.47659
## ucl         500  7.0161 0.04641  6.92515 7.10708  6.97002
## RB          500 -0.0065 0.00783 -0.02186 0.00885 -0.01531
## RSE         500  0.1801 0.00073  0.17866 0.18151  0.17812
## COV         500  0.9600 0.00877  0.94281 0.97719  1.00000
##
## $`trapsindex = 1, recapfactor = 2, fitindex = 1, maskindex = 1`
##                n    mean      se     lcl      ucl  median
## estimate    500  4.4284 0.05842  4.3139  4.54287  4.4158
## SE.estimate 500  1.2996 0.00981  1.2803  1.31878  1.3012
## lcl         500  2.5323 0.04180  2.4504  2.61425  2.4967
## ucl         500  7.8019 0.07721  7.6506  7.95325  7.8078
## RB          500 -0.1143 0.01168 -0.1372 -0.09143 -0.1168
## RSE         500  0.3070 0.00243  0.3023  0.31178  0.2972
## COV         500  0.9440 0.01029  0.9238  0.96417  1.0000
##
## $`trapsindex = 2, recapfactor = 2, fitindex = 1, maskindex = 2`
##                n     mean      se      lcl      ucl   median
## estimate    500  4.62434 0.03824  4.54939  4.69929  4.59226
## SE.estimate 500  0.81600 0.00362  0.80891  0.82309  0.81839
## lcl         500  3.28267 0.03149  3.22094  3.34440  3.23926
## ucl         500  6.52046 0.04528  6.43172  6.60920  6.49095
## RB          500 -0.07513 0.00765 -0.09012 -0.06014 -0.08155
## RSE         500  0.17943 0.00076  0.17795  0.18092  0.17822
## COV         500  0.93200 0.01127  0.90991  0.95409  1.00000
##
## $`trapsindex = 1, recapfactor = 0.5, fitindex = 2, maskindex = 1`
##                n    mean      se     lcl      ucl  median
## estimate    500 5.22167 0.07578 5.07315  5.37019 5.18323
## SE.estimate 500 1.74583 0.02123 1.70423  1.78744 1.71095
## lcl         500 2.78488 0.05062 2.68566  2.88410 2.70792
## ucl         500 9.94459 0.12380 9.70195 10.18723 9.75850
## RB          500 0.04433 0.01516 0.01463  0.07404 0.03665
## RSE         500 0.34976 0.00350 0.34290  0.35661 0.33759
## COV         500 0.94200 0.01046 0.92149  0.96251 1.00000
##
```

```
## $`trapsindex = 2, recapfactor = 0.5, fitindex = 2, maskindex = 2`
##                n     mean      se     lcl     ucl   median
## estimate    500  4.98864 0.04090  4.9085 5.06880  4.97724
## SE.estimate 500  0.92341 0.00434  0.9149 0.93192  0.92743
## lcl         500  3.48357 0.03322  3.4184 3.54868  3.47396
## ucl         500  7.15203 0.04916  7.0557 7.24838  7.15163
## RB          500 -0.00227 0.00818 -0.0183 0.01376 -0.00455
## RSE         500  0.18824 0.00084  0.1866 0.18989  0.18550
## COV         500  0.94600 0.01012  0.9262 0.96583  1.00000
##
## $`trapsindex = 1, recapfactor = 1, fitindex = 2, maskindex = 1`
##                n     mean      se      lcl      ucl  median
## estimate    500  5.10488 0.06876  4.97011  5.23966  4.9510
## SE.estimate 500  1.72420 0.08918  1.54942  1.89899  1.5818
## lcl         500  2.78061 0.04719  2.68812  2.87311  2.6784
## ucl         500 10.08840 0.61405  8.88488 11.29193  9.1970
## RB          500  0.02098 0.01375 -0.00598  0.04793 -0.0098
## RSE         500  0.34471 0.01301  0.31922  0.37021  0.3212
## COV         500  0.95800 0.00898  0.94040  0.97560  1.0000
##
## $`trapsindex = 2, recapfactor = 1, fitindex = 2, maskindex = 2`
##                n    mean      se     lcl     ucl   median
## estimate    500  4.9840 0.03952  4.9065 5.06145  4.95613
## SE.estimate 500  0.9066 0.00427  0.8983 0.91502  0.90778
## lcl         500  3.5012 0.03202  3.4385 3.56397  3.48533
## ucl         500  7.1013 0.04781  7.0076 7.19497  7.08731
## RB          500 -0.0032 0.00790 -0.0187 0.01229 -0.00877
## RSE         500  0.1846 0.00076  0.1831 0.18609  0.18262
## COV         500  0.9500 0.00976  0.9309 0.96912  1.00000
##
## $`trapsindex = 1, recapfactor = 2, fitindex = 2, maskindex = 1`
##                n    mean      se      lcl     ucl  median
## estimate    499 5.08846 0.06975  4.95175 5.22517 5.08003
## SE.estimate 499 1.58825 0.01682  1.55528 1.62122 1.57657
## lcl         499 2.81611 0.04749  2.72302 2.90920 2.80196
## ucl         499 9.28585 0.10583  9.07842 9.49327 9.22943
## RB          499 0.01769 0.01395 -0.00965 0.04503 0.01601
## RSE         499 0.32504 0.00269  0.31976 0.33031 0.31165
## COV         499 0.93587 0.01098  0.91436 0.95739 1.00000
##
## $`trapsindex = 2, recapfactor = 2, fitindex = 2, maskindex = 2`
##                n     mean      se      lcl      ucl   median
## estimate    500  4.87325 0.04075  4.79337  4.95313  4.82339
## SE.estimate 500  0.89038 0.00427  0.88201  0.89876  0.88879
## lcl         500  3.41845 0.03312  3.35353  3.48337  3.39162
## ucl         500  6.95444 0.04902  6.85836  7.05052  6.91494
## RB          500 -0.02535 0.00815 -0.04133 -0.00937 -0.03532
## RSE         500  0.18580 0.00080  0.18422  0.18737  0.18290
## COV         500  0.94800 0.00994  0.92852  0.96748  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"),
                list(model2D = "IHP", D = "D2"))

## 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)
```

```
summary(sims6)
```

```
## run.scenarios(nrepl = 50, scenarios = scen6, trapset = traps(possumCH),
##     maskset = possummask, pop.args = poplist)
##
## Replicates    50
## Started       21:00:10 04 Dec 2014
## Run time      0.081  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
##         1     IHP D1
##         2     IHP D2
##
## OUTPUT
## $`D = 2, popindex = 1`
##       n    mean      se
## n    50 109.320 1.33445
## ndet 50 267.680 3.17894
## nmov 50 141.740 1.89797
## dpa  50   2.199 0.01287
##
## $`D = 2.5, popindex = 2`
##       n    mean      se
## n    50 135.140 1.54999
## ndet 50 325.940 3.66555
## nmov 50 170.100 2.23830
## dpa  50   2.172 0.01254
```

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)
```

```
## 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")
```

```
pop1 <- attr(sims6a$output[[1]][[1]], "popn")
pop2 <- attr(sims6a$output[[2]][[1]], "popn")
png(file='d:/density secr 2.9/secrdesign/vignettes/secrdesign-fig6.png',
    width=850, height=400)
par(mfrow = c(1,2), mar=c(1,1,1,6), cex=1.25)
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)
dev.off()
```

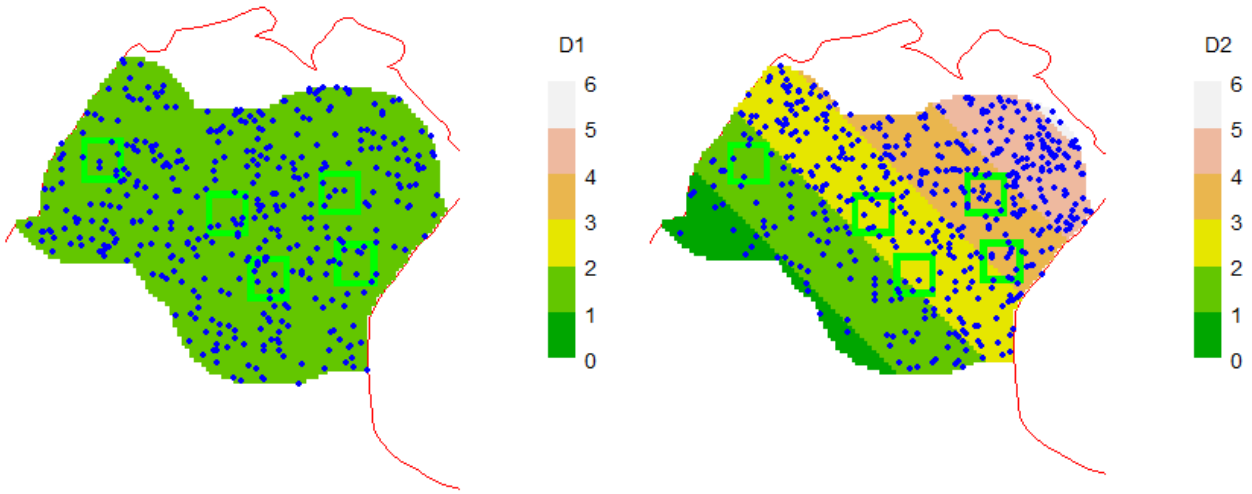

Fig. 6. Simulated homogeneous (left) and inhomogeneous (right) distributions of brushtail possums at Waitarere, New Zealand. Traps in green (each hollow grid 180 m square).

Code for linear habitat

Code to illustrate the use of linear habitat models. This assumes you have the package **secrlinear**, which is not yet on CRAN.

```
library(secrlinear)
library(secrdesign)

## 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)
```

```
summary(sims7)
```

```
## run.scenarios(nrepl = 50, scenarios = scen7, trapset = trplist,
##     maskset = linmask, det.args = list(det.arg), fit = FALSE,
##     seed = 345)
##
## Replicates    50
## Started       21:00:16 04 Dec 2014
## Run time      0.222  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
## $`trapsindex = 1, D = 50`
##       n    mean      se
## n    50  93.040 1.26973
## ndet 50 157.280 2.65069
## nmov 50  42.600 1.22724
## dpa  50   1.457 0.01058
##
## $`trapsindex = 2, D = 50`
##       n  mean      se
## n    50 57.74 1.03285
## ndet 50 73.90 1.53443
## nmov 50  5.84 0.38371
## dpa  50  1.10 0.00604
##
## $`trapsindex = 1, D = 200`
##       n    mean      se
## n    50 358.180 2.40487
## ndet 50 604.640 5.02010
## nmov 50 165.080 2.38188
## dpa  50   1.461 0.00549
##
## $`trapsindex = 2, D = 200`
##       n    mean      se
## n    50 234.340 2.01780
## ndet 50 298.920 2.88230
## nmov 50  24.140 0.61545
## dpa  50   1.103 0.00244
```

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 = list(grid), fit = FALSE, maskset = list(mask))
summary(raw8)
```

```
## run.scenarios(nrepl = 20, scenarios = scen8, trapset = list(grid),
##     maskset = list(mask), fit = FALSE)
##
## Replicates    20
## Started       22:17:42 10 Jan 2015
## Run time      0.032  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
## $`group = F, noccasions = 4, D = 8, g0 = 0.3, sigma = 30 +
##   group = M, noccasions = 4, D = 4, g0 = 0.2, sigma = 40`
```

```
##         n    mean      se
## F.n    20  75.200 1.36034
## F.ndet 20 190.250 3.88037
## F.nmov 20  99.450 2.54277
## F.dpa  20   2.223 0.02309
## M.n    20  46.850 1.36358
## M.ndet 20 118.200 3.97995
## M.nmov 20  65.800 2.65924
## M.dpa  20   2.351 0.03109
##
## $`group = F, noccasions = 8, D = 8, g0 = 0.3, sigma = 30 +
##   group = M, noccasions = 8, D = 4, g0 = 0.2, sigma = 40`
##         n    mean       se
## F.n    20  87.450  2.67392
## F.ndet 20 401.200 12.42379
## F.nmov 20 270.850  9.20505
## F.dpa  20   3.482  0.04598
## M.n    20  53.250  1.38007
## M.ndet 20 226.800  7.18646
## M.nmov 20 158.300  5.61253
## M.dpa  20   3.557  0.05052
```

Now fit the models and check the summary output for density ('D') and sex ratio (proportion female 'pmix') without repeating the header information.

```
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))
```

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

```
## $`group = F, noccasions = 4, D = 8, g0 = 0.3, sigma = 30 +
##   group = M, noccasions = 4, D = 4, g0 = 0.2, sigma = 40`
##              n     mean      se
## estimate    20 11.92890 0.21816
## SE.estimate 20  1.16438 0.01198
## lcl         20  9.85650 0.19577
## ucl         20 14.43775 0.24149
## RB          20 -0.00593 0.01818
## RSE         20  0.09789 0.00083
## COV         20  1.00000 0.00000
##
## $`group = F, noccasions = 8, D = 8, g0 = 0.3, sigma = 30 +
##   group = M, noccasions = 8, D = 4, g0 = 0.2, sigma = 40`
##              n     mean      se
## estimate    20 12.31550 0.20505
## SE.estimate 20  1.07385 0.00994
## lcl         20 10.38437 0.18631
## ucl         20 14.60624 0.22443
## RB          20  0.02629 0.01709
## RSE         20  0.08740 0.00067
## COV         20  1.00000 0.00000
```

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

```
## Warning: assuming first scenario row corresponds to requested pmix
## Warning: assuming first scenario row corresponds to requested pmix

## $`group = F, noccasions = 4, D = 8, g0 = 0.3, sigma = 30 +
##   group = M, noccasions = 4, D = 4, g0 = 0.2, sigma = 40`
##              n    mean      se
## estimate    20 0.67768 0.00994
## SE.estimate 20 0.04492 0.00047
## lcl         20 0.58408 0.01010
## ucl         20 0.75874 0.00908
## RB          20 0.01652 0.01491
## RSE         20 0.06669 0.00155
## COV         20 1.00000 0.00000
##
## $`group = F, noccasions = 8, D = 8, g0 = 0.3, sigma = 30 +
##   group = M, noccasions = 8, D = 4, g0 = 0.2, sigma = 40`
##              n     mean      se
## estimate    20  0.65850 0.00730
## SE.estimate 20  0.04035 0.00041
## lcl         20  0.57562 0.00765
## ucl         20  0.73268 0.00662
## RB          20 -0.01225 0.01095
## RSE         20  0.06151 0.00120
## COV         20  1.00000 0.00000
```