secrdesign miscellaneous tools

Murray Efford

2017-10-09

Contents

Introduction

This vignette focuses on functions in **secrdesign** that aid the evaluation of study designs without the more time-consuming step of simulation (see secrdesign-vignette.pdf for simulation).

For each scenario we would like to determine –

1. Expected sample size for a given design (`Enrm`)

2. Predicted precision of density estimates, given expected sample size (`minnrRSE`)
3. Costing (`costing`)
4. Detector spacing that maximises precision (`optimalSpacing`)

Components 1–3 are wrapped together in the function `scenarioSummary`, which accepts a dataframe of scenarios constructed with `make.scenarios` (described in secrdesign-vignette.pdf).

Expected sample size

Expected sample sizes (component 1 above) are covered in detail in secrdesign-Enrm.pdf. Formulae are provided for hazard detection functions ('HHN', 'HEX' etc. with parameters 'lambda0' and 'sigma') rather than the more familiar probability-based functions ('HN', 'EX' etc. with parameters 'g0' and 'sigma'). There is no exact conversion between corresponding functions (e.g. 'HN' and 'HHN') because they differ in shape. However, an approximate conversion is built into `Enrm` and functions that draw on it. This sets $\lambda_0 = -\log(1 - g_0)$ and scales σ so that the probability-of-detection curves cross at $\sigma' = \sigma$.

Predicted precision of density estimates

Our measure of precision (strictly, its inverse) is the relative standard error (RSE)[1]. Precision depends on sample size. We characterise the size of a SECR sample by the number of distinct individuals n and the number of recaptures r, where the total number of detections is $C = n + r$. Expected values of n and r can be computed directly for detectors of type[2] "multi", "proximity" or "count", given an SECR model (secrdesign-Enrm.pdf).

A rule of thumb for precision

There is an approximate relationship between expected sample size (n and r) and the precision of density estimates. This 'rule of thumb' suggests that $\mathrm{RSE}(\hat{D}) \approx 1/\sqrt{\min(n, r)}$ (Appendix 1). The method is faster and easier than simulating many scenarios, which is the usual approach in the literature and elsewhere in **secrdesign**. However, the result is approximate and disregards possible bias: it is wise to check with a few simulations (the `optimalSpacing` function includes optional simulations).

Poisson N vs fixed N

The distinction between Poisson n and binomial n (Poisson vs fixed N) can have a substantial effect on $\mathrm{RSE}(\hat{D})$ as detailed in Appendix 2. Output from `scenarioSummary` includes both the rule-of-thumb RSE for Poisson n ('rotRSE') and this value adjusted for binomial n ('rotRSEB').

Geometry correction factor

For some geometries, $\mathrm{RSE}(\hat{D})$ is underestimated by the plain rule-of-thumb. This is covered by including a simple multiplier $\mathrm{RSE}(\hat{D}) \approx \mathrm{CF}/\sqrt{\min(n, r)}$ where $\mathrm{CF} > 1$. The value of CF should be determined by simulation; typical values are $\mathrm{CF} \approx 1.2$ for a hollow grid and $\mathrm{CF} \approx 1.4$ for a line of detectors.

[1] $\mathrm{RSE}(\hat{D}) = \widehat{SE}(\hat{D})/\hat{D}$; 'coefficient of variation' (CV) is also sometimes used for $\mathrm{RSE}(\hat{D})$.

[2] See secr-overview.pdf for more on detector types.

Characterisation of detector arrays

Metrics not reported by `summary.traps` may be useful for understanding the performance of a detector layout. `scenarioSummary` currently reports –

Metric	Description
arrayN	Number of detectors per array
arrayspace	Mean distance to nearest detector in sigma units
arrayspan	Maximum dimension of array in sigma units

Costing

The current arguments of the `costing` function are

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

Unit costs are provided in the `unitcost` argument. The costing algorithm applies up to 5 unit costs as in the following table.

Component	Unit cost	Costing
arrays	perarray	perarray × nrepeats
detectors	perdetector	perdetector × nrow(traps) × nrepeats
travel	perkm	perkm × routelength × (noccasions+1) × nrepeats
visits	pervisit	$\sum$(pervisit × trapcost) × (noccasions+1) × nrepeats
detections	perdetection	perdetection × (E(n) + E(r))

'arrays' and 'detectors' represent one-off costs. 'nrepeats' is the number of replicate detector arrays (default 1): it is a multiplier in all costings except for 'detections', where it is implicit in E(n) and E(r).

'travel' and 'visits' are alternative ways to cost field time. The variable 'routelength' represents the length of a path followed to visit all detectors; if not specified it is approximated by the sum of the nearest-trap distances (nrow(traps)-1) × spacing(traps). The variable 'trapcost' is a vector of length equal to the number of detectors. By default it is a vector of 1's, but detector-specific values may be provided as trap covariate 'costpervisit'. In the latter case the value of 'pervisit' should be 1.0 for clarity.

The total cost is the sum of the 5 components, some of which may be zero.

Example

Suppose each detector check costs $5 and each detection costs $15 to process –

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

```
##     travel     arrays  detectors     visits detections  totalcost
##     0.0000     0.0000     0.0000  1920.0000   895.3985  2815.3985
```

If detectors differ in their cost of access we record detector-specific costs in the trap covariate 'costpervisit'. Here we mock up an example in which cost increases with distance from the bottom left detector –

```
covariates(tr) <- data.frame(costpervisit = 5 + edist(tr, tr[1,])/20)
```

You might like to visualise the gradient in per-detector cost with `plot(as.mask(tr), cov = 'costpervisit')`. Cost of access varies from \$5 to \$17.37, and it costs \$757.76 to visit all 64 sites.

The covariate 'costpervisit' is detected automatically and multiplied by pervisit during costing. Hence

```
costing (tr, nrm, 5, unitcost = list(pervisit = 1, perdetection = 15))
```

```
##     travel     arrays  detectors     visits detections  totalcost
##     0.0000     0.0000     0.0000  4546.5651   895.3985  5441.9636
```

Demonstration of `scenarioSummary`

Here is a simple demonstration comparing two scenarios in the dataframe `scen1` constructed with `make.scenarios`. The default probability-based halfnormal detection function (paramaters g0, sigma) is converted to a hazard detection function with a warning (not shown here).

```
scen1 <- make.scenarios(D = c(5,10), g0 = 0.2, sigma = 25, noccasions = 3)
scenarioSummary(scen1, tr)
```

```
##   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
##        esa CF rotRSE rotRSEB arrayN arrayspace arrayspan
## 1 5.530333  1 0.2110  0.1714     64          1    9.8995
## 2 5.530333  1 0.1492  0.1212     64          1    9.8995
```

Focusing on costings, if each detector check costs \$5 and each detection costs \$15 to process –

```
summ <- scenarioSummary(scen1, tr, costing = TRUE,
                        unitcost = list(pervisit = 5, perdetection = 15))
summ[,c(1,22:24)]  # selected columns
```

```
##   scenario   visits detections totalcost
## 1        1 15155.22   751.5651  15906.78
## 2        2 15155.22  1503.1301  16658.35
```

Optimal detector spacing

Our goal is to find the detector spacing that maximises the precision of SECR estimates of density.

The function `optimalSpacing` starts with a preferred array geometry (e.g., a square grid of 64 traps)[3]. We assume you have pilot estimates of density D and the detection parameters λ_0 and σ (see Making do without pilot values if you lack these).

What does `optimalSpacing` do?

The function performs four tasks for a given geometry and SECR model:

1. Find the detector spacing that minimises RSE($\hat{D}$), using the rule of thumb.

[3] `optimalSpacing` is also useful for comparing different candidate geometries, but we do not demonstrate that.

2. Compute rule-of-thumb RSE($\hat{D}$) for a range of spacings
3. For selected detector spacings, determine RSE($\hat{D}$) and the relative root-mean-square-error of $\hat{D}$ by simulation[4].
4. Plot the results.

The current argument list is

```
str(optimalSpacing)
```

```
## function (D, traps, detectpar, noccasions, nrepeats = 1, detectfn = c("HHN",
##     "HHR", "HEX", "HAN", "HCG", "HN", "HR", "EX"), fittedmodel = NULL, xsigma = 4,
##     R = seq(0.2, 4, 0.2), CF = 1, simulationR = seq(0.4, 4, 0.4), nrepl = 0,
##     plt = FALSE, ...)
```

Steps 3 and 4 are optional, depending respectively on 'nrepl' and 'plt'. Setting CF does not change the optimal spacing determined by `optimalSpacing`.

Parameter values

Pilot values are required for the population density D and detection parameters λ_0 and σ. It is assumed that D has units animals per hectare (0.01 km^2) and σ is in metres. For these units the product $\sigma\sqrt{D}$ usually falls in the range 20–110.

Parameters λ_0 and σ define a detection function. The supported detection functions are a subset of those available in **secr** - those that directly model the hazard of detection ('HHN', 'HEX' etc.; see ?detectfn in **secr**). Values of λ_0 and σ obtained by fitting a hazard function are numerically similar to the corresponding values of g_0 and σ from fitting a probability function ('HN', 'EX' etc.), but they are not identical. See also Making do without pilot values.

Rule-of-thumb optimisation

```
grid <- make.grid(8, 8, detector = "proximity")  # 64 binary proximity detectors
par(mar = c(4,5,2,2), mgp = c(2.4,0.6,0))
out <- optimalSpacing(D = 5, traps = grid, detectpar = list(lambda0 = 0.2, sigma = 20),
               noccasions = 5, plt = TRUE)
```

[4]Estimated as rRMSE($\hat{D}$) $= \frac{1}{D}\sqrt{\sum_{i=1}^{m}(\hat{D}_i - D)^2/m}$ from m replicate simulations.

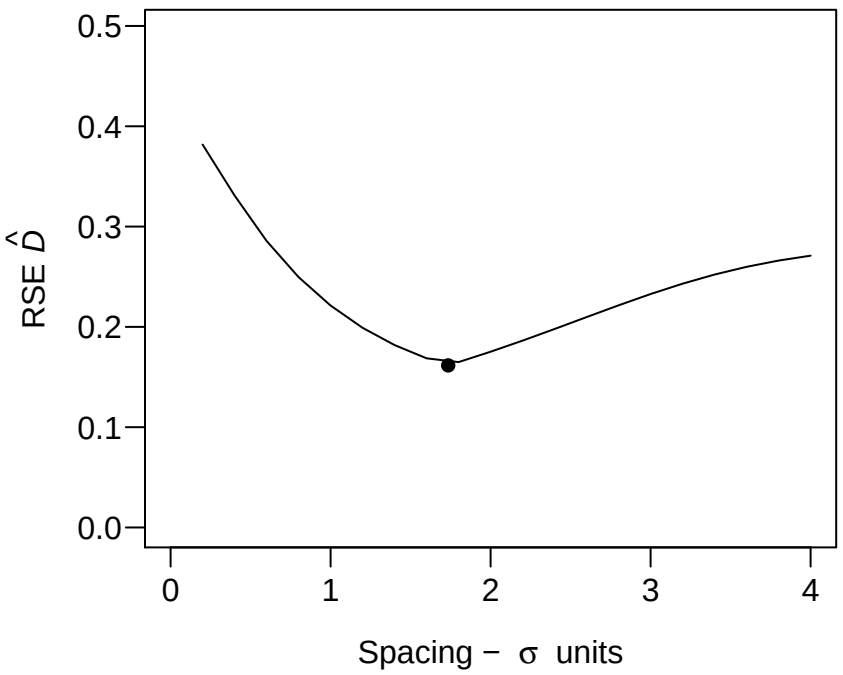

Fig. 1. Precision of density estimates predicted by rule-of-thumb for various spacings. Dot shows optimal spacing. The x axis corresponds to argument `R`.

Now examine the numeric rule-of-thumb output, dropping the first component which is a lengthy dataframe:

```
out$rotRSE[-1]
```

```
## $optimum.spacing
## [1] 34.70044
##
## $optimum.R
## [1] 1.735022
##
## $minimum.RSE
## [1] 0.1616114
```

The optimum spacing of about 35 metres is predicted to yield estimates with RSE($\hat{D}$) $\approx$ 16%.

Simulations

Simulations are more reliable for predicting RSE($\hat{D}$) than the rule of thumb, and additionally allow the assessment of bias, but they are much slower. Simulations may be performed for a selection of relative detector spacings merely by specifying nrepl > 0 in `optimalSpacing`. Only simple models are allowed, with the same restrictions as for the rule-of-thumb calculations (single session, constant detector type, homogeneous density, constant detection parameters etc.).

Simulation to predict RSE does not require many replicates – nrepl = 20 is often enough. RB and rRMSE are more variable than RSE and hence require larger values of nrepl.

Here is a simple example[5]. Results are overplotted.

[5]In Windows you may need to respond to a firewall request to allow parallel processing (ncores>1).

```
par(mar = c(4,5,2,2), mgp = c(2.4,0.6,0))
out2 <- optimalSpacing(D = 5, traps = grid, detectpar = list(lambda0 = 0.2, sigma = 20),
                       noccasions = 5, plt = TRUE, ylim = c(0,0.6),
                       nrepl = 20, ncores = 2, seed = 237)
```

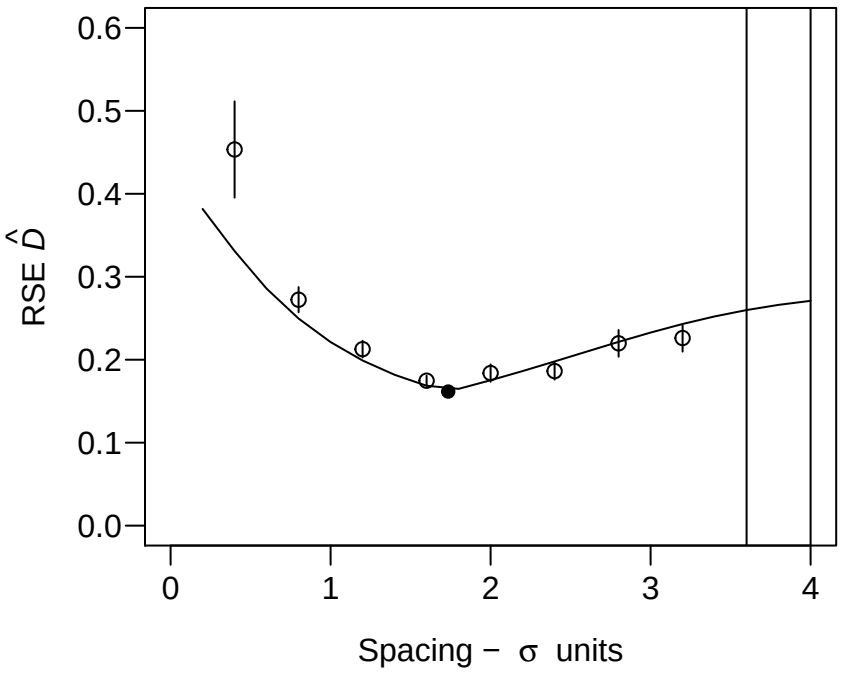

Fig. 2. Simulated RSE(D-hat) (open circles) corresponding to **Fig. 1**. Vertical lines indicate a ±2SE confidence interval.

Predicted RSE was extreme (off the plot, or nearly so) for the smallest and largest spacings simulated.

More plotting

Plots may be overlaid to better illustrate relationships. This example shows the effect of varying λ_0.

```
par(mar = c(4,5,2,2), mgp = c(2.4,0.6,0), mfrow = c(1,1))
cols <- c("blue","orange","red")
# set up plot
plot(0, 0, type = "n", xlim = c(0,4), ylim = c(0,0.6),
     xlab = expression(paste("Spacing -  ", sigma, "  units")),
     ylab = expression(paste("RSE ", hat(italic(D)))))
# for three values of lambda0...
lam0 <- c(0.1, 0.2, 0.5)

for (i in 1:3) {
    out <- optimalSpacing(D = 5, traps = grid, noccasions = 5,
                          detectpar = list(lambda0 = lam0[i], sigma = 20))
    plot(out, add = TRUE, col = cols[i], lwd = 1.5)
}
legend ("top", col = cols, lwd = 1.5, pch = 16, legend = lam0,
```

```
        bty = "n", horiz = TRUE)
```

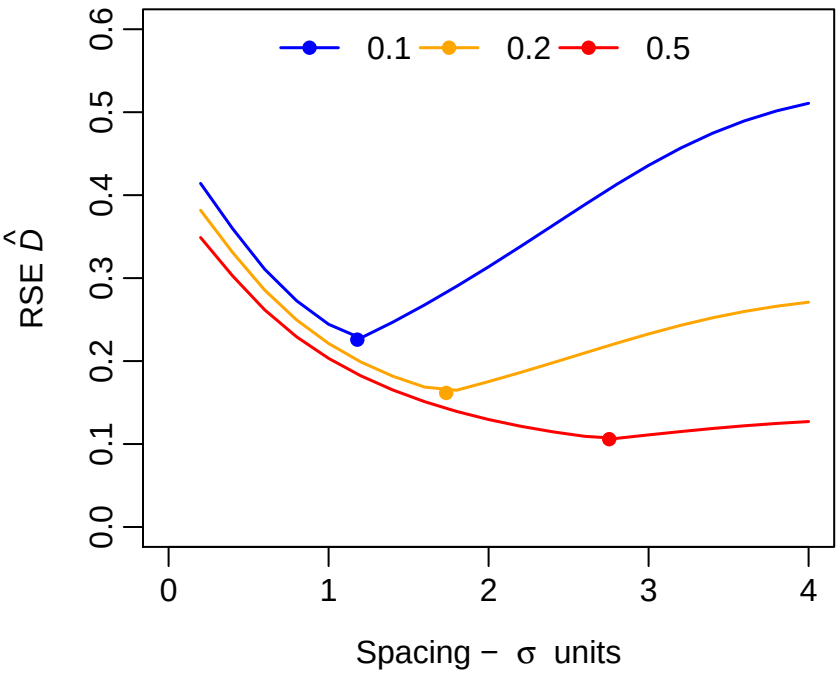

Fig. 3. Rule-of-thumb RSE($\hat{D}$) and optimal spacing for three levels of λ_0. Other parameters as in **Figs. 1 & 2**.

Pathological designs

If we examine the simulation results from `optimalSpacing` more closely we see that the large simulated RSE are associated with large bias and RMSE:

```
out2$simRSE[-1]  # do not show each simulation
```

```
## $summary
##       R  n  RSE.mean    RSE.se  RB.mean   RB.se     rRMSE
## 1   0.4 20   0.45341   0.02901 -0.07823 0.09463 0.4198500
## 2   0.8 20   0.27234   0.00760  0.01113 0.05862 0.2557469
## 3   1.2 20   0.21274   0.00504 -0.01222 0.04445 0.1941167
## 4   1.6 20   0.17470   0.00304  0.09740 0.03449 0.1791289
## 5   2.0 20   0.18390   0.00524  0.01661 0.04363 0.1909130
## 6   2.4 20   0.18628   0.00511  0.06975 0.05294 0.2410639
## 7   2.8 20   0.21976   0.00803  0.04261 0.05109 0.2267497
## 8   3.2 20   0.22613   0.00817  0.11583 0.06699 0.3141348
## 9   3.6 20   2.32601   1.96537  0.45628 0.33361 1.5240866
## 10 4.0 20 254.58228 252.35949  0.69471 0.45808 2.1141319
```

Clearly SECR is unreliable for these spacing scenarios – we can describe them as 'pathological'. The rule-of-thumb RSE is unreliable for pathological scenarios, so we have an incentive to first reject such scenarios. However, objective criteria for pathological scenarios are elusive.

At one extreme, the SECR model is unidentifiable when recaptures provide no information about the scale of detection - the case when all recaptures are at zero distance from the first capture. This happens when detectors are too far apart ($R >> 1$).

Recaptures also provide inadequate information about the scale of detection when the entire array spans only a few of the sites at which an individual may be detected - the case when the array is much smaller than one home-range.

Making do without pilot values: function `getdetectpar`

So far we have required plausible estimates of the model parameters D, λ_0 and σ. These may be based on a pilot study or literature search.

If no prior information is available for λ_0 and σ then ballpark values may be inferred from the likely density D and the expected total number of detections C.

Function `getdetectpar` first infers σ from D using the relationship $\sigma = k/\sqrt{D}$ (Efford et al. 2016). This requires a value for k; unpublished results suggest k is commonly in the range 0.3–1.1, and the default of $k = 0.5$ is somewhat conservative. If σ is known then it may be provided to override this step.

A numerical search is then conducted for the value of λ_0 that results in C expected detections with the given design.

Using the 64-detector grid from before:

```
msk <- make.mask(grid, buffer = 100, type = 'trapbuffer')
getdetectpar(D = 5, C = 200, traps = grid, mask = msk, noccasions = 5)
```

```
## $lambda0
## [1] 0.2458863
##
## $sigma
## [1] 22.36068
```

Beyond seeking a single λ_0, we can examine the effect of varying C and k:

```
Cval <- seq(20, 400, 40)
kval <- c(0.4, 0.6, 0.8)
plot (0, 0, type = 'n', xlim = c(0,400), ylim = c(0,1.1), las = 1,
      xlab = expression(paste('Total detections  ',  italic(C))),
      ylab = expression(paste('Inferred  ',  lambda[0])))
for (i in 1:3) {
    lam0 <- sapply(Cval, getdetectpar, D = 5, sigma = NULL, k = kval[i],
                   traps = grid, mask = msk, noccasions = 5)[1,]
    lines (Cval, lam0, col = cols[i])
}
legend ("top", col = cols, lwd = 1.5, pch = 16, legend = kval,
        bty = "n", horiz = TRUE)
```

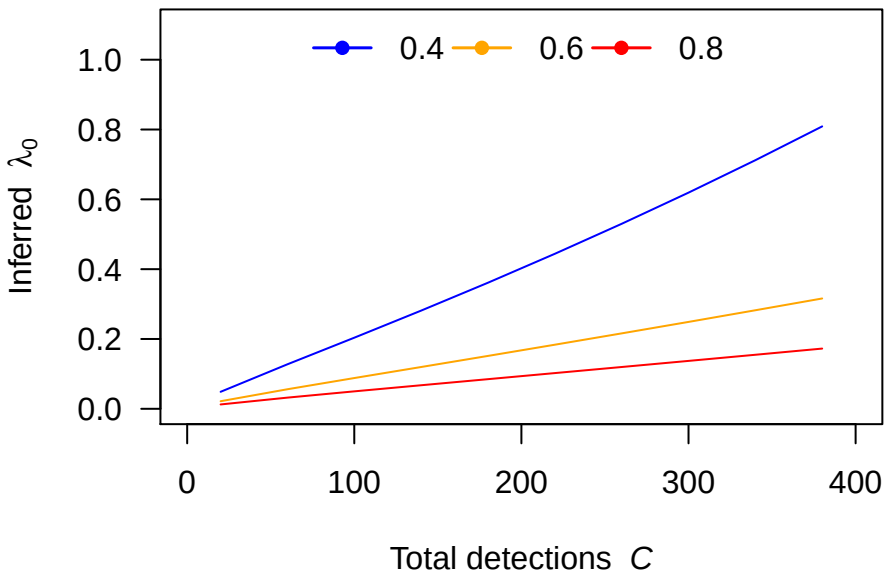

Fig. 4. Values of the parameter λ_0 (lambda0) corresponding to varying total number of detections for a particular density and sampling design. Curves correspond to different levels of the crowding (overlap) parameter k as shown. The levels of k in this case correspond to sigma = 17.9, 26.8, 35.8m.

Limitations

The scope of 'optimalSpacing' is limited to single-session capture–recapture models with 'multi', 'proximity' or 'count' detectors (this excludes single-catch traps). It is assumed that 'detectpar' and 'detector' do not differ among occasions.

References

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

Efford, M. G., Dawson, D. K., Jhala, Y. V. and Qureshi, Q. (2016) Density-dependent home-range size revealed by spatially explicit capture–recapture. *Ecography* **39**, 676–688.

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

Seber, G. A. F. (1982) The estimation of animal abundance and related parameters. 2nd Ed. Griffin, London.

Appendix 1. A rule of thumb for RSE($\hat{D}$)

Recaptures are the lifeblood of capture–recapture estimators. It has long been known that the variance of a simple Lincoln-Petersen 2-sample estimate of population size N depends directly on the number of captures of marked animals r ($\text{var}(\hat{N}) \approx 1/r$; Seber 1982 p. 61).

Perhaps surprisingly, the relationship also holds approximately for the variance of density estimated by SECR. However, in practice there is a clear divergence from the variance of simulated estimates for large r when n remains small (Fig. 5a). Empirically, this may be corrected by relying on n rather than r as the measure of sample size when $n < r$, hence the rule-of-thumb $\text{RSE}(\hat{D}) \approx 1/\sqrt{\min(n, r)}$ (Fig. 5b). The approximation overstates the precision of $\hat{D}$ for more linear detector arrays and a correction factor may be needed (M. Efford unpubl. results).

The simulations in Fig. 5 are for a scenario in which a square 8 × 8 grid of "count" detectors was operated for 5 occasions. Code follows.

```
nk <- function (x, k = 0.5, target= 20, grid, mask, noccasions = 5) {
    # x is lambda0; k is sigma in units of l_D
    # D = 1 because in lD units
    nrm <- Enrm(D = 10000, traps = grid, mask = mask, detectpar =
                    list(lambda0 = x, sigma = k), noccasions = noccasions)
    nrm['Er'] - target
}
tryit <- function (r= 20, k = 0.5, grid, mask, noccasions = 5, nrepl = 5) {
    # find occasion-specific lambda0 that provides r recaptures given sigma = k
    ur <- try(uniroot(nk, interval = c(0.001, 100), target = r, k = k,
                      grid = grid, mask = mask, noccasions = noccasions))
    if (inherits(ur, 'try-error'))
        lam0 <- NA
    else
        lam0 <- ur$root * noccasions
    output <- matrix(nrow = nrepl, ncol = 12,
                     dimnames=list(NULL,c('Er','k','lam0','n','r','pMove',
                                          'D','SE','sig','SEsig','CV', 'CVsig')))
    output[,1] <- r
    output[,2] <- k
    output[,3] <- lam0
    if (is.na(lam0)) return(output)
    for (i in 1:nrepl) {
        CH <- sim.capthist(grid, popn = list(D = 10000, buffer = 4), noccasions = 1,
                           detectfn = 14, detectpar = list(lambda0 = lam0, sigma = k))
        output[i,4] <- nrow(CH)          # n
        output[i,5] <- sum(CH)-nrow(CH)  # r
        output[i,6] <- sum(unlist(moves(CH))>0, na.rm=TRUE) / output[i,5]
        if (nrow(CH)>4) {   # autoini limit n>4 in secr.fit
            fit <- secr.fit(CH, mask=mask, detectfn=14, trace=F)
            if (inherits(fit, 'secr')) {
                out <- c(unlist(predict(fit)['D',2:3]),
                         unlist(predict(fit)['sigma',2:3]))
                out[5] <- out[2]/out[1]
                out[6] <- out[4]/out[3]
                output[i,7:12] <- out
            }
        }
    }
```

```
    output
}
runtrials <- function(grid, mask, Er = c(5,10,20,50,100,200), kval = c(0.5,1)) {
    rk <- expand.grid(Er, kval)
    mapply(tryit, rk[,1], rk[,2], MoreArgs = list(grid = grid, mask = mask),
           SIMPLIFY = FALSE)
}
grid64 <- make.grid(nx = 8, ny = 8, detector = 'count', spacing = 1)
mask64 <- make.mask(grid64, nx = 32, type = 'trapbuffer', buffer = 4)
trials2 <- runtrials(grid64, mask64)
```

```
plottrials <- function (trials, title = '', hline = c(0.1, 0.2, 0.5, 1),
                        xmax, type = 'r', xl, label = '') {
    plot(0,0,type = 'n',xlim = c(0,xmax), ylim = c(0.05,2), log = 'y',
         yaxs = 'i', xaxs = 'i',
         ylab = expression(paste('RSE(', hat(italic(D)), ')')),
         xlab = '', axes = FALSE)
    axis(1)
    axis(2, at = c(0.05,0.1,0.2,0.5,1,2), las = 1,
         labels = c('0.05','0.1','0.2','0.5','1','2'))
    box()
    lines(1:xmax, 1/sqrt(1:xmax))
    abline(h = hline, lty = 2, xpd = FALSE)
    tmp <- as.data.frame(do.call(rbind,trials))
    mtext(side = 1, line = 2.4, xl, cex = 0.9, xpd = TRUE)
    xval <- if (type == 'r') tmp$r else pmin(tmp$n,tmp$r)
    points(xval, tmp$CV, pch=21, bg = "red", cex=1.4, xpd=TRUE)
    usr <- par()$usr
    text ((usr[2]-usr[1])*-0.2, (usr[4]-usr[3])*1.9, label, cex = 1.3, xpd = TRUE)
    invisible()
}

par(mfrow = c(1,2), mar = c(4,5,3,1), mgp = c(2.4,0.8,0))
plottrials(trials2, '', xmax = 270, xl = expression(italic(r)), type = 'r', label = 'a.')
plottrials(trials2, '', xmax = 120, xl = expression(paste("min(",italic(n), "," ,
    italic(r),")")), type = 'nr', label = 'b.')
```

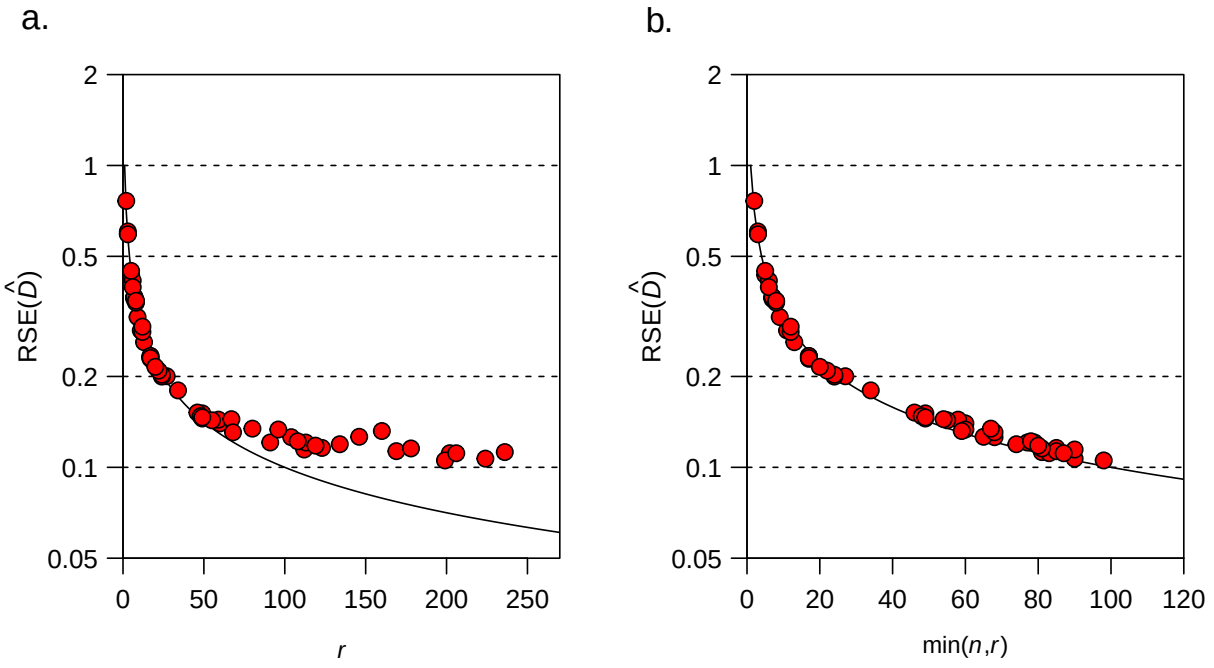

Fig. 5. Precision of simulated SECR density estimates as function of two measures of sample size, number of recaptures of marked animals r and number of individuals n. (a) r alone, and (b) minimum of n and r. Each point represents one of 5 replicates for each of 12 scenarios (see text).

Appendix 2. Extending rule-of-thumb RSE to fixed N, binomial n

Appendix 1 ignored an important distinction that is addressed here. The sampling variance (precision) of density estimates depends on the nature of the sample:

- By default **secr**, treats the study population as a cookie-cutter sample from a larger universe, and the variance includes survey-to-survey variation in N, the number of individuals in the (realised) local population. The variation in N usually Poisson, and the number of individuals actually detected n is also Poisson-distributed. This approach has the advantage that the estimated sampling variance does not depend on the (possibly arbitrary) extent of the local population (the size of the cookie cutter).
- Another scenario is that we are concerned only to estimate the unique, immediate, realised population size. The local population is our target, and N is a fixed quantity that depends on the size of the cookie cutter A. The number of individuals actually detected is then binomial rather than Poisson.

In the first case our estimate is of the expected density across multiple surveys of a population with the same density parameters. In the second case our estimate is of the realised density in a particular survey. The difference lies in the inclusion or exclusion of between-survey variance. Excluding that component of uncertainty results in lower variance and narrower confidence intervals. The magnitude of the difference decreases as A increases, and in the limit as $A \to \infty$ it disappears.

The rule of thumb as presented in the previous section concerns the Poisson-N case that is the default in **secr**. Here we develop an adjustment for the rule of thumb when the target population occupies a known region of area A. Consider the RSE($\hat{D}$) for a homogeneous population estimated by the Horvitz-Thompson approach (see Huggins (1989) and Borchers and Efford (2007)):

$$\mathrm{var}(\hat{D}) = s^2 + V_\theta.$$

The term V_θ concerns uncertainty in the detection parameters and is unaffected by fixed vs Poisson N. The term s^2 represents uncertainty due to n, which may be Poisson ($s^2 = n/a^2$) or binomial ($s^2 = (1-a/A).n/a^2$), where a is the effective sampling area[6]. If rse is the expected RSE($\hat{D}$) for the Poisson-N case, then by manipulating these expressions we get the expected RSE($\hat{D}$) for fixed N in area A ($N = DA$):

$$\mathrm{rse}_B = \sqrt{\mathrm{rse}^2 - \frac{1}{N}}.$$

Appendix 3. Technical issues.

`optimalSpacing` considers detector spacing relative to the spatial scale parameter σ, given the shape of detection function specified by the argument 'detectfn'. It does this by holding the detector array fixed and varying σ. However, in order to emulate varying spacing for fixed σ, the real aim of the exercise, it is necessary also to vary other spatial parameters, specifically density D and attributes of the habitat mask (spacing and buffer width). Consider the relative spacing $R = s/\sigma$. Starting from a baseline density D_1 in which $R = 1$ ($\sigma = s$), density must be scaled as $D_R = D_1 \times R^2$. Linear mask attributes are scaled by $1/R$.

$D(\mathbf{x})$ is assumed constant for all $\mathbf{x}$ in the implementation of `optimalSpacing`.

The spacing corresponding to minimum RSE($\hat{D}$) is determined by linear interpolation in `optimalSpacing`. The function `minnrRSE` typically (perhaps always) has a minimum where $E(n) = E(r)$. This suggests that the optimal spacing could be found by solving for this point. However, both $E(n)$ and $E(r)$ require extensive computation, so there is probably nothing to be gained.

[6] $a = \int p.(\mathbf{x})\, d\mathbf{x}$ where $p.(\mathbf{x})$ is the probability an animal centred at $\mathbf{x}$ is detected at least once (Borchers and Efford 2008).