ipsecr 1.4 - spatially explicit capture–recapture by inverse prediction

Murray Efford

2025-06-10

Contents

This document provides an overview of **ipsecr** 1.4, an R package for spatially explicit capture–recapture analysis (SECR) that uses simulation and inverse prediction instead of maximum likelihood, as in **secr** (Efford 2022), or MCMC, as in various Bayesian approaches. The parent package **secr** included the function `ip.secr`, now deprecated; **ipsecr** extends that capability in its central function `ipsecr.fit` whose arguments are closely modelled on `secr.fit` from **secr**.

Simulation and inverse prediction allows some models to be fitted that strictly cannot be fitted by other methods. Single-catch traps are a common example.

There are limitations: **ipsecr** 1.4 allows variation in detection parameters (e.g., λ_0, σ) only with respect to individual and session. This excludes variation with respect to occasion, detector or previous detection (e.g., ~t, ~b or ~bk as allowed in **secr**). **ipsecr** does allow variation in λ_0 and σ that depends on the location of each animal's activity centre, which is not allowed in **secr**.

The package **plot3D** should be installed to display Fig. 2, and some results require secr >= 4.5.9.

Inverse prediction for capture–recapture estimation

The method (Efford 2004; Efford, Dawson and Robbins 2004; see also Carothers 1979 and Pledger and Efford 1998) uses a vector-valued function of the raw data that provides one or more proxies for each coefficient (beta parameter) in the capture–recapture model. Each proxy is assumed to be monotonically related to the corresponding coefficient.

We use $\mathbf{x}_p$ for the vector of known parameter values at which simulations are performed and $\mathbf{y}_p$ for the vectors of proxies computed from these simulated data. The method fits a multivariate multiple regression over a set of points in parameter space ('box') and inverts that regression to estimate parameter values $\mathbf{x}$ from the observed proxy vector $\mathbf{y}$.

The default proxy function `proxy.ms` works for simple models with two detection parameters. It uses a naive non-spatial estimate of population size (simply the number detected), the corresponding non-spatial detection probability p, and the 'root-pooled spatial variance' measure from the function `RPSV` in **secr** (argument CC = TRUE):

Parameter	Proxy	Proxy scale
Density D	number detected n	log
Detection intercept g_0 or λ_0*	$\hat{p}$	cloglog
Detection spatial scale σ	RPSV	log

* depends on detection function

The monotonic relationship is demonstrated by simulation in the following figure.

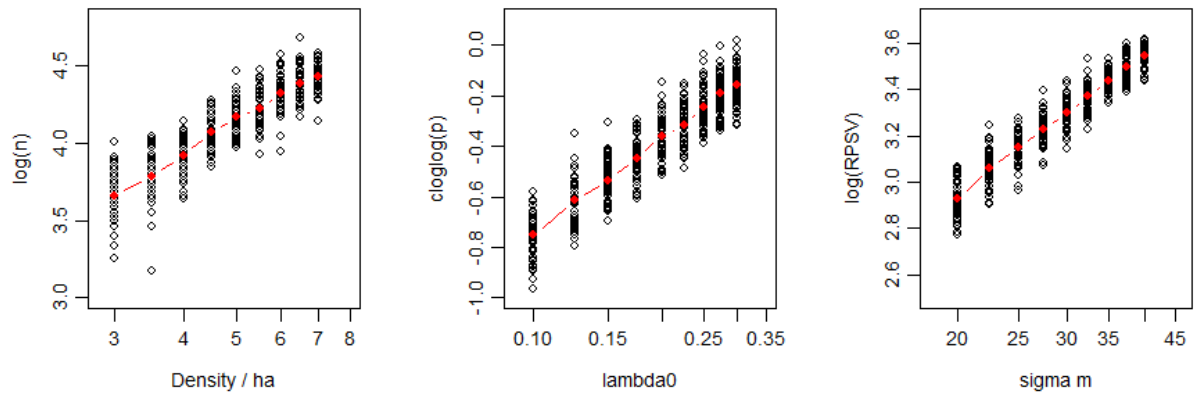


Figure 1. Monotonic relationships between parameters (x-axis, log scale) and their default proxies (y-axis). 100 single-catch traps at 20-m spacing on a square grid. 50 simulations for each level, with other parameters held at their central value. Red line follows mean.

The algorithm includes these steps

1. Compute proxy function from data
2. Simulate data for parameters at the vertices* of a box around plausible estimates
3. Compute proxy function for each simulation to generate predictor matrix
4. Fit a multivariate multiple regression model with proxies at each vertex as the dependent variables
5. Invert regression model to estimate vector of parameters from the data proxies (1)
6. If the estimated parameters do not all lie inside box, adjust the box and repeat from (2)
7. Simulate at the estimates to obtain variance-covariance matrix

* including some centre points

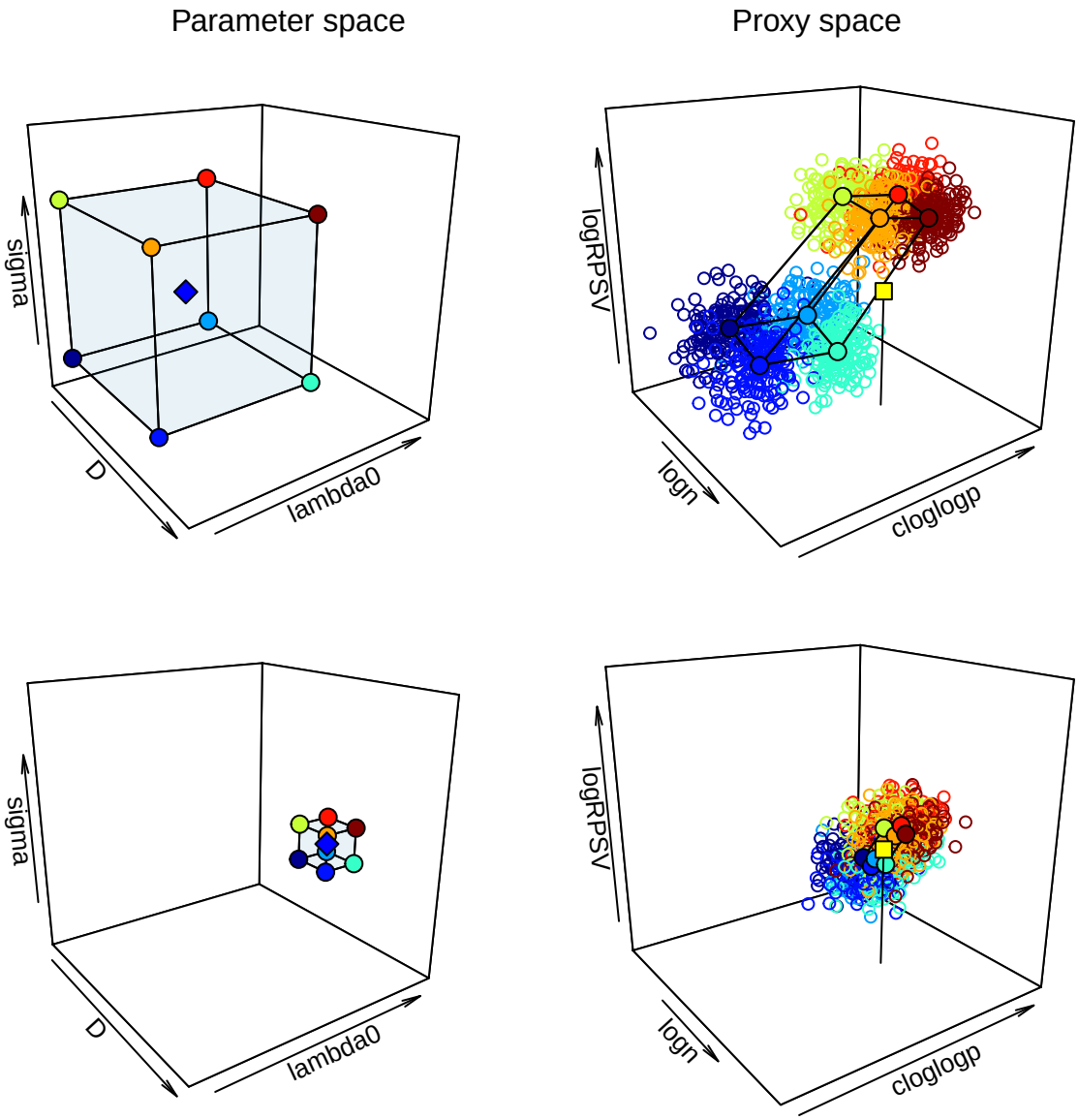


Figure 2. Schematic process of fitting in `ipsecr.fit` (requires package plot3D). Simulations are conducted at the vertices of a box in parameter space (top left; link scale) centred on an initial guess (blue diamond). The results in proxy space (top right; frame connects design point means, centre omitted for clarity) support a linear model for proxies as a function of parameters. The model is inverted and applied to the observed proxy vector (yellow square) giving the centre of a new, smaller box in parameter space (bottom left). The model is refined by further simulations (bottom right) from which the final estimate is inferred (white square, bottom left)

Simple example

This example uses a simulated single-catch trap dataset in **secr** that is loaded automatically when **ipsecr** is loaded. See here for instructions on reading actual data.

```
library(ipsecr)
if (!require("spatstat")) warning ("install spatstat to run vignette code")
```

```
## Warning: package 'spatstat.data' was built under R version 4.3.3
```

```
setNumThreads(2)   # adjust to number of available cores
```

```
ip.single <- ipsecr.fit(captdata, buffer = 100, detectfn = 'HHN')
```

The fitted model is of class 'ipsecr' for which there are the usual methods matching those in **secr** (`print`, `coef`, `predict`, `summary` etc. as detailed below).

```
predict(ip.single)
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log  5.6237165  0.67323230  4.4512887  7.1049509
## lambda0  log  0.4382384  0.06696152  0.3253861  0.5902307
## sigma    log 28.2457061  1.31161350 25.7897687 30.9355203
```

Proxy functions

The proxy function takes an **secr** capthist object as its first argument. It returns a vector of surrogate values, at least one for each coefficient (beta parameter) in the model. There does not strictly need to be a one-to-one relationship between proxies and coefficients, but multiple proxies have not so far been found useful and may muddy the water. The default function is `proxy.ms`. For example,

```
proxy.ms(captdata)
```

```
##        logn    cloglogp     logRPSV
##  4.33073334 -0.03724765  3.24371994
```

`proxy.ms` works for models with variation in parameters D and NT. Detection parameters may vary spatially and by session. Users are free to define their own proxyfn.

Changing the proxy function may have little effect on the estimates. Here we demonstrate an older builtin proxy function `proxyfn1`:

```
ip.single.1 <- ipsecr.fit(captdata, buffer = 100, detectfn = 'HHN', proxyfn = proxyfn1,
    N.estimator = "null", verbose = FALSE)
```

```
# secr function 'collate' works for both secr and ipsecr fits
collate(ip.single, ip.single.1)[1,,,]
```

```
## , , D
##
##             estimate SE.estimate      lcl      ucl
## ip.single   5.623717   0.6732323 4.451289 7.104951
## ip.single.1 5.650595   0.7244975 4.399477 7.257506
##
## , , lambda0
##
##              estimate SE.estimate       lcl       ucl
## ip.single   0.4382384  0.06696152 0.3253861 0.5902307
## ip.single.1 0.4409693  0.07895862 0.3113119 0.6246274
##
## , , sigma
##
##             estimate SE.estimate      lcl      ucl
## ip.single   28.24571    1.311614 25.78977 30.93552
## ip.single.1 28.26378    1.560606 25.36683 31.49156
```

Multi-session models

Data may take the form of independent samples. Models are constructed as in **secr** (secr-multisession.pdf). In **ipsecr 1.4**, density D, detection parameters, and non-target NT parameters may all depend on session or session covariates as in **secr**.

Fitting a density gradient

Distiller and Borchers (2015) simulated an example with a gradient in population density to demonstrate their method for data from single-catch traps when the time of each capture is known. Inverse prediction may be used to estimate density from these data, but assuming a constant density can result in bias. Here we illustrate a strategy for fitting the density gradient with inverse prediction.

This requires a proxy function that includes a proxy for the density gradient. As the function is a function of the capture histories alone we must use a gradient over detectors to stand for a gradient over points on the habitat mask. The default function `proxy.ms` does this automatically. A model in terms of mask coordinates and covariates is re-cast as a model in terms of detector coordinates and covariates. Proxies are coefficients of a glm for detector-specific counts as a function of the predictors, using a log link.

First simulate some data with an east-west gradient in density.

```
tr <- traps(captdata)
mask <- make.mask(tr)
covariates(mask) <- data.frame(D = (mask$x-265)/20)  # for sim.pop
set.seed(1237)
pop <- sim.popn(D = 'D', core = mask, model2D = 'IHP', buffer = 100)
ch <- sim.capthist(tr, popn = pop, detectfn = 'HHN', noccasions = 5,
  detectpar = list(lambda0 = 0.2, sigma = 25))
# show east-west trend
table(tr[trap(ch),'x'])
```

```
##
## 365 395 425 455 485 515 545 575 605 635
##  12  17  21  20  28  24  22  23  31  36
```

Note that the x- and y-coordinates of traps and mask are scaled internally and independently to zero mean and unit variance.

```
ipx <- ipsecr.fit(ch, mask = mask, detectfn = 'HHN', model = list(D~x))
```

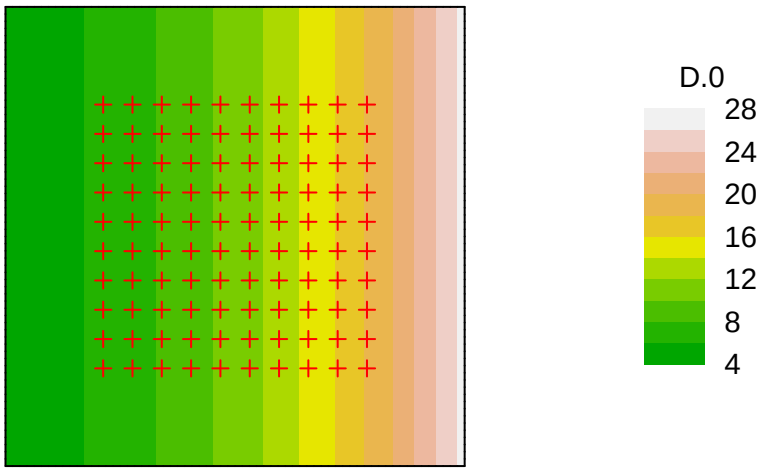
```
coef(ipx)
```

```
##                beta    SE.beta        lcl       ucl
## D         2.3856258 0.10242293  2.1848805  2.586371
## D.x       0.5154916 0.14363647  0.2339693  0.797014
## lambda0 -1.7330217 0.12028393 -1.9687738 -1.497269
## sigma     3.2556566 0.05067609  3.1563333  3.354980
```

```
predict(ipx)
```

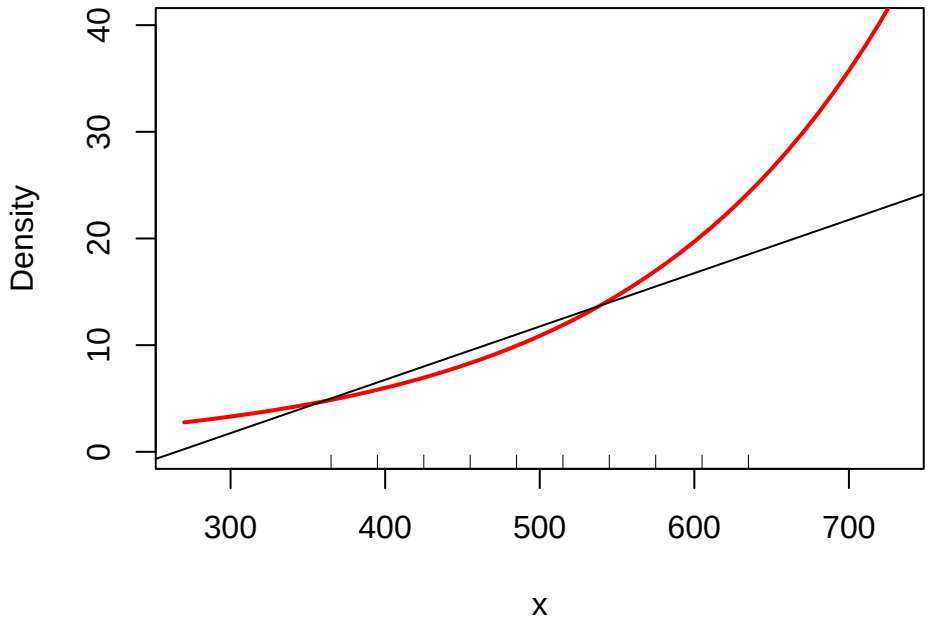
```
##          link   estimate SE.estimate       lcl        ucl
## D         log 10.8658603  1.11583837  8.889586 13.2814861
## lambda0   log  0.1767495  0.02133726  0.139628  0.2237403
## sigma     log 25.9366397  1.31521170 23.484329 28.6450294
```

```
plot(predictDsurface(ipx))
plot(tr, add = TRUE)
plotMaskEdge(ipx$mask, add=T)
```



The fitted relationship is linear on the log scale, whereas it was simulated as linear on the natural scale. To better match the original trend it is possible to use an identity link for D.

```
oldpar <- par(mar = c(4,6,4,4))
# model refers to scaled values of x, so repeat here
m <- mean(traps(ch)$x); s <- sd(traps(ch)$x)
xval <- seq(270,730,10)
xvals <- scale(xval, m, s)
pred <- predict(ipx, newdata = data.frame(x = xvals))
plot(0,0, type='n', xlim= c(270,730), ylim = c(0,40), xlab = 'x', ylab = 'Density')
lines(xval, sapply(pred, '[', 'D','estimate'), col = 'red', lwd=2)
abline(-265/20,0.05) # true linear trend
rug(unique(tr$x))    # trap locations
```




```
par(oldpar)
```

Figure 3. Estimated trend in density from log-linear model (red). Actual trend shown as black line.

Non-target interference

The capthist data object may include an attribute 'nontarget' that is a binary $K \times S$ matrix (K detectors, S occasions) indicating whether detector k was disturbed or occupied by non-target species on each occasion s.

Disturbance at single-catch and capped detectors is usually taken to be exclusive of captures: an occupied trap cannot later catch a non-target animal or be disturbed, and vice versa. Multi-catch traps or binary proximity detectors may be disturbed after having registered detections, and both detections and interference are recorded (disturbance is independent of prior detector status, and has the effect of truncating the sampling interval). Variations may be specified with the details argument 'nontargettype' that takes one of the values 'exclusive', 'truncated', 'erased' or 'independent'.

Type	Allowed detectors	Explanation
'exclusive'	single, capped	detector may be occupied or interfered with, but not both
'truncated'	all	sampling interval ends at time of interference (default for all detectors except single and capped)
'erased'	all	interference erases any previous detections at a site in the particular interval
'independent'	all	interference has no effect on detections, but is recorded and modelled

We illustrate the fitting of a model with non-target captures by simulating disturbance at about 50% of the 235 unoccupied traps in the 'captdata' dataset. (This is an ad hoc method, used only for generating demonstration data).

```
set.seed(123)
ch <- captdata
attr(ch, 'nontarget') <- (1-t(apply(ch,2:3,sum))) * (runif(500)>0.5)
summary(ch)$nontarget
```

```
##                      1  2  3  4  5 Total
## detectors nontarget 25 27 29 27 28   136
```

The attribute 'nontarget' is a matrix with one row per trap and one column per occasion. Entries are either 1 (nontarget or disturbed) or 0 (target species or undisturbed). For a dataset including non-target data `ipsecr.fit` automatically adds the parameter 'NT' for the hazard of a single ubiquitous disturbance process competing (in the 'exclusive' case) for detectors with each individual-specific capture process. The non-target model may be suppressed by setting the details argument 'ignorenontarget = TRUE'.

Information on the disturbance process is given by the frequency of nontarget events in the capthist input (attribute 'nontarget'). The last element of the vector returned by the proxy function `proxy.ms` transforms this to a hazard on the log scale (i.e. complementary log-log of proportion):

```
proxy.ms(ch)
```

```
##        logn    cloglogp     logRPSV   cloglogNT
##  4.33073334 -0.03724765  3.24371994 -1.14742163
```

```
ip.single.nontarget <- ipsecr.fit(ch, detectfn = 'HHN')
```

The estimate of lambda0 has risen compared to ip.single:

```
predict(ip.single.nontarget)
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log  5.6363589  0.75114333  4.3457030  7.3103344
## lambda0  log  0.6288584  0.10570868  0.4533825  0.8722500
## sigma    log 28.1845337  1.41140896 25.5512151 31.0892432
```

```
## NT       log  0.5509196  0.04872183  0.4634008  0.6549673
```

Note that 'NT' cannot be estimated if all traps are occupied. 'NT' may be modelled as a function of session and detector (trap), but not occasion.

Conditioning to minimise the effect of ghost individuals

Genotyping errors or photo misidentification commonly manifest in SECR datasets as ghost individuals with a single detection. Kodi et al. (2024) proposed to reduce the resulting bias by conditioning on a certain minimum number of detections K, specifically $K = 2$. The required modification to the likelihood has not yet been implemented in **secr** owing to the complexity of covering all **secr** options, but a model for $K \geq 2$ is readily implemented in **ipsecr** as we show here.

First we generate some data.

```
grid <- make.grid(9,9, spacing = 20, detector = 'count')
msk <- make.mask(grid, buffer = 100)
CH <- sim.capthist(grid, popn = list(D = 10), detectfn = 'HHN', seed = 123,
                   detectpar = list(lambda0 = 0.3, sigma = 25))
```

Next we corrupt the simulated detection histories in two ways. Ghosting increases the number of individuals without changing the number of detections; dropping singletons decreases both.

```
# add ghosts
pGhost <- 0.05
CHdf <- as.data.frame(CH)
gh <- runif(nrow(CHdf)) < pGhost
CHdf$ID[gh] <- paste('N', CHdf$ID[gh], CHdf$Occasion[gh], CHdf$TrapID[gh], sep = '.')
CHg <- make.capthist(CHdf, grid, noccasions = ncol(CH))

# drop histories with only one detection
CHg2 <- subset(CHg, apply(abs(CHg), 1, sum) >= 2)

# summarise
sapply(list(CH=CH, CHg=CHg, CHg2=CHg2), summary, terse = TRUE)
```

```
##             CH CHg CHg2
## Occasions    5   5    5
## Detections 532 532  483
## Animals     69 104   55
## Detectors   81  81   81
```

```
# conventional ML fit with and without ghosts
predict(secr.fit(CH, mask = msk, detectfn = 'HHN', trace = FALSE))
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log 10.3917847  1.27648639  8.1756648 13.2086126
## lambda0  log  0.2764994  0.01789651  0.2435889  0.3138563
## sigma    log 26.1872063  0.75520576 24.7483757 27.7096882
```

```
predict(secr.fit(CHg, mask = msk, detectfn = 'HHN', trace = FALSE))
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log 15.0776861  1.53583237 12.3553164 18.3999026
## lambda0  log  0.2048706  0.01405527  0.1791229  0.2343194
## sigma    log 28.9058118  0.92348473 27.1517524 30.7731871
```

Ghosting causes a spurious increase in the density estimate, as expected. Next we try **ipsecr** on these datasets, plus one from which singletons have been removed.

```
# fits using inverse prediction also show the effect of ghosting
fit0 <- ipsecr.fit(CH, mask = msk, detectfn = 'HHN', verbose = FALSE)
fitg <- ipsecr.fit(CHg, mask = msk, detectfn = 'HHN',  verbose = FALSE)
```

```
predict(fit0)
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log 10.2685989  1.28795597  8.0382995 13.1177150
## lambda0  log  0.2975771  0.03038013  0.2437385  0.3633079
## sigma    log 26.2391799  0.95069438 24.4410441 28.1696052
```

```
predict(fitg)
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log 16.2581729  1.91451390 12.9175955 20.4626461
## lambda0  log  0.1902281  0.01882617  0.1567615  0.2308393
## sigma    log 26.0893933  0.87752085 24.4254051 27.8667412
```

We fit a model that allows for excluded singletons by tweaking the simulation function.

```
# define a custom simulation function that drops singleton histories
simCHK <- function (trps, popn, detectfn, detparmat, noccasions, NT, details, K = 2) {
    ch <- secr::sim.capthist(traps = trps, popn  = popn, detectfn = detectfn,
                        detectpar = as.list(detparmat[1,]), noccasions = noccasions)
    ndet <- apply(abs(ch),1,sum)
    subset(ch, ndet >= K)
}

# fit with custom function
fitg2 <- ipsecr.fit(CHg2, mask = msk, detectfn = 'HHN',  verbose = FALSE,
                    details = list(CHmethod = simCHK))
```

```
predict(fitg2)
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log 10.2678802  1.53035364  7.6791211 13.7293530
## lambda0  log  0.2801537  0.03049268  0.2264759  0.3465539
## sigma    log 26.2604391  0.96719606 24.4321718 28.2255162
```

The effect of ghosting is removed.

Tuning the algorithm

The inverse prediction method assumes

1. linear relationships between proxies and parameters (jointly, a hyper-plane), and
2. negligible error in the predicted (mean) proxy for a given set of parameter values.

Performance of the algorithm with respect to these assumptions depends on several components of the 'details' argument of `ipsecr.fit` that may be seen as tuning parameters. The first assumption is more plausible for small regions of parameter space (hence a small 'boxsize'). The second assumption is eased by increasing the number of simulations. Actual settings are a compromise between these requirements and execution time. Users may choose a different compromise.

By default, the size of the box in parameter space is set to ± 'boxsize' units on the link scale. This may be changed to a multiple of the central value with `boxtype = 'relative'`.

It is usual to start with a wide box and to use a narrower box for subsequent simulations, on the assumption that the first box has selected a region of parameter space very close to the solution.

The stopping criterion 'dev.max' is used to exit the simulation loop early when sufficient precision has been achieved for all parameters. If `boxtype = 'absolute'` then the criterion is the standard error on the link scale. If `boxtype = 'relative'` then the criterion is the relative standard error (RSE or CV) on the link scale.

Tuning parameter	Default	Description
boxtype	'absolute'	'absolute' or 'relative'
boxsize1	0.2	size of first box
boxsize2	0.05	boxsize for boxes after the first
centre	3	number of centre points
dev.max	0.002	stopping criterion
min.nsim	20	minimum number of simulations per vertex
max.nsim	200	maximum number of simulations per vertex
max.nbox	5	maximum number of boxes
max.ntries	2	maximum number of attempts to achieve valid simulation
var.nsim	2000	number of simulations for variance-covariance matrix

Fractional designs

By default, **ipsecr** fits a full factorial design in the parameter space. For NP parameters, simulations are performed at 2^{NP} points, the corners of a hyperrectangle (box), plus possible centre points. The total number of simulations grows rapidly for large NP. Fractional factorial designs may omit some parameter combinations while retaining balance and other desirable characteristics.

The package **FrF2** may be used for fractional factorial designs (Groemping, 2014). Fractional designs are selected by setting `details = list(factorial = 'fractional')` in `ipsecr.fit`.

```
ip.Fr <- ipsecr.fit(captdata, detectfn = 'HHN', details = list(factorial = 'fractional'))
```

```
collate(ip.single, ip.Fr)[1,,,]
```

```
## , , D
##
##           estimate SE.estimate      lcl      ucl
## ip.single 5.623717   0.6732323 4.451289 7.104951
## ip.Fr     5.619563   0.6741916 4.445766 7.103272
##
## , , lambda0
##
##            estimate SE.estimate       lcl       ucl
## ip.single 0.4382384  0.06696152 0.3253861 0.5902307
## ip.Fr     0.4393181  0.07165968 0.3197755 0.6035495
##
## , , sigma
##
##           estimate SE.estimate      lcl      ucl
## ip.single 28.24571    1.311614 25.78977 30.93552
## ip.Fr     28.27523    1.324641 25.79591 30.99285
```

```
ip.single$proctime
```

```
## [1] 100.43
```

```
ip.Fr$proctime
```

```
## [1] 109.53
```

In this example the fractional design was actually slower than the full design because (i) there is little difference in the number of design points when NP = 3 (7 vs 11 with 3 centre points) and (ii) the fractional fit went to a third box. Conditions when fractional designs are faster have not been determined - they are probably useful only when parameters are numerous.

The default fractional design is a 1/2 factorial, illustrated by this code:

```
if (require('FrF2')) {
  NP <- 3
  boxsize <- rep(0.2,3)
  design <- FrF2(2^(NP-1),NP, factor.names = c('D','lambda0','sigma'), ncenter = 2)
  # recast factors as numeric
  design <- sapply(design, function(x) as.numeric(as.character(x)))
  design <- sweep(design, MAR=2, STATS = boxsize, FUN='*')
  # apply to beta
  beta <- log(c(5,0.2,25))
  designbeta <- sweep(design, MAR=2, STATS=beta, FUN='+')
  round(designbeta,3)
}
```

```
         D lambda0 sigma
[1,] 1.609  -1.609 3.219
[2,] 1.809  -1.409 3.419
[3,] 1.809  -1.809 3.019
[4,] 1.409  -1.809 3.419
[5,] 1.409  -1.409 3.019
[6,] 1.609  -1.609 3.219
```

The first and last rows are centre points.

For other designs you may specify the desired arguments of FrF2 as a list e.g., `details = list(factorial = 'fractional', FrF2args = list(nruns = 4, nfactors = 3, ncenter = 3))`.

Models with extra parameters

Extensions of **ipsecr** may involve the estimation of additional 'real' parameters (i.e. parameters other than D, g0, lambda0, sigma or NT). Extra parameters should be specified in the details argument 'extraparam', a named list with the numeric starting value of each parameter. Extra parameters are assumed to be constant: their corresponding models are set automatically to ~1. Extra parameters are passed internally to user-defined simulation functions via the 'details' argument.

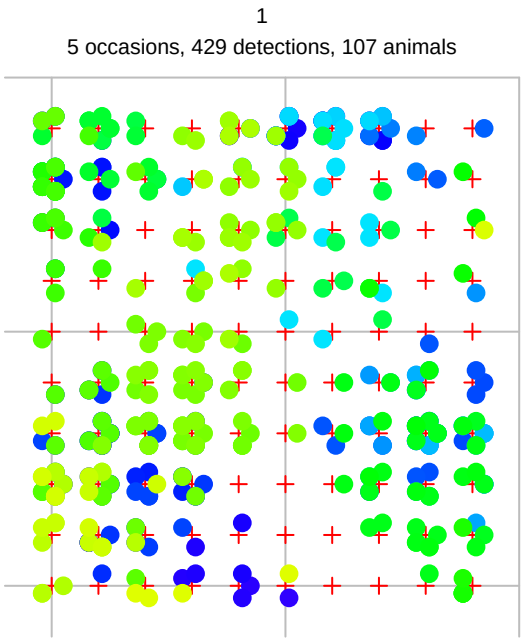
Our example fits a clumped distribution of activity centres instead of the usual Poisson distribution. This uses a form of Neyman-Scott clustering (a Thomas distribution). The Thomas distribution draws cluster 'parent' points from a uniform (Poisson) distribution and places a Poisson random number of centres in a bivariate normal pattern around each parent. Parameters to be estimated are the overall intensity (density D), the expected number of centres per parent (mu), and the spatial spread of each cluster (named hsigma, following the cluster option in `secr::sim.popn`). Experimentation suggests that SECR data are usually inadequate to estimate both mu and hsigma, but there are biologically interesting scenarios for which hsigma may be considered known: specifically, clumped groups of animals sharing an activity centre may be modelled with hsigma = ϵ where ϵ is a small positive distance (e.g. 1 metre).

We proceed to estimate mu and fix hsigma. This requires a custom function for simulation and a custom proxy function. Simulation is straightforward - just a wrapper around the function from secr. For a cluster

proxy we fit a Thomas distribution in **spatstat** (Baddeley and Turner 2005). This proxy is one of several possibilities, none obviously better than the rest. This requires that **spatstat** has been installed.

First simulate some test data (requires secr >= 4.5.9):

```
grid <- make.grid(nx = 10, ny = 10, spacing = 20, detector = 'proximity')
msk <- make.mask(grid, buffer = 100)
set.seed(123)
pop <- sim.popn(D = 20, core = grid, buffer = 100, model2D = 'cluster',
    details = list(mu = 5, hsigma = 1))
ch <- sim.capthist(grid, pop, detectfn = 14, detectpar =
        list(lambda0 = 0.2, sigma = 20), noccasions = 5)
plot(ch, border = 20)
```



Next, call `ipsecr.fit` with these customizations:

- user proxyfn : clusterproxyT
- user popmethod : simclusteredpop
- user parameters : mu, hsigma
- fixed parameter : hsigma = 1

```
# user function to simulate Thomas (Neyman-Scott) distribution of activity centres
# expect parameters mu and hsigma in list 'details$extraparam'
simclusteredpop <- function (mask, D, N, details) {
    secr::sim.popn(
        D = D[1],
        core = mask,
        buffer = 0,
        Ndist = 'poisson',    # necessary for N-S cluster process
        model2D = 'cluster',
        details = details$extraparam)
}
```

```
# extend the built-in proxy with clumping argument mu
# spatstat fits Thomas process parameters kappa and scale = hsigma^2
# mu is a model parameter derived from mu = D / kappa
```

```
clusterproxyT <- function (capthist, ...) {
    pr <- ipsecr::proxy.ms(capthist)
    pp <- spatstat.geom::as.ppp(secr::centroids(capthist),
        W = as.numeric(apply(secr::traps(capthist),2,range)))
    tfit <- spatstat.core::thomas.estK(pp)
    c(pr, logmu = log(tfit$modelpar['mu']))
}
clusterfitT <- ipsecr.fit(ch, proxyfn = clusterproxyT, mask = msk,
    detectfn = 'HHN', details = list(popmethod = simclusteredpop,
    extraparam = list(mu = 5, hsigma = NA)), fixed = list(hsigma = 1))
```

```
predict(clusterfitT)
```

```
##         link   estimate SE.estimate       lcl        ucl
## D        log 19.1320785  5.99039474 10.506335 34.8395933
## lambda0  log  0.2052792  0.03070909  0.153360  0.2747755
## sigma    log 19.4953885  0.88410258 17.838170 21.3065669
## mu       log  7.0044856  4.65039396  2.142833 22.8962419
```

The results are not very impressive (wide confidence intervals on both $\hat{D}$ and $\hat{\mu}$), but at least mu can be estimated. Let's compare with the MLE assuming a Poisson distribution of activity centres:

```
clusterfitML <- secr.fit(ch, mask = msk, detectfn = 'HHN', trace = FALSE)
```

```
predict(clusterfitML)
```

```
##         link   estimate SE.estimate        lcl        ucl
## D        log 17.9234488  1.76726799 14.7807093 21.7344114
## lambda0  log  0.2185081  0.01681006  0.1879664  0.2540124
## sigma    log 19.8730580  0.64043382 18.6569565 21.1684277
```

Relationship to package secr

Some functionality of **ipsecr** is shared with **secr**.

Methods for 'ipsecr' of generic functions defined in **secr**

Function	Description
makeNewData	generate dataframe suitable for `predict.ipsecr`

Methods for 'ipsecr' of generic functions from base R and stats

Function	Description
coef	coefficients (beta parameters) of fitted model
plot	plot detection function from fitted model
print	display 'ipsecr' object
predict	predict real parameters
summary	summarise fitted model
vcov	variance-covariance matrix of coefficients (beta parameters)

Functions from **secr** that work on fitted 'ipsecr' objects as-is

Function	Description
ellipse.secr	plot confidence ellipse for 2 parameters
predictDsurface	predict density for each cell of mask
makeStart	generate starting values from various inputs
collate	tabulate results from multiple models

Troubleshooting and limitations

ipsecr 1.4 is not intended for models with many sessions or many covariates.

This version does not allow for variation in detection parameters (g0, lambda0, sigma) among occasions (e.g., g0~t, g0~b, g0~bk).

Only 2-parameter detection functions are allowed in `ipsecr.fit` (HN, EX, UN, HHN, HEX) (see ?detectfn). 3-parameter functions are simulated by `simCH`, but proxies for the shape parameter have not been explored (maybe some sample measure of bivariate skewness or tail weight).

Sparse data

Simulation becomes unreliable with very sparse populations, or sparse sampling, because some simulated datasets will have no recaptures or even no captures. The code allows a failed simulation to be repeated (set the 'max.ntries' details argument > 1), but results probably should not be relied upon when there are warning messages regarding failed simulations.

"simulations for box 1 did not reach target for proxy SE 0.002"

This message may not be fatal. The target precision is arbitrary. Review the 'Variance bootstrap' table of verbose output.

"solution not found after 5 attempts"

It seems ipsecr.fit is not converging on the right part of the parameter space. Try specifying a wider boxsize1 for the first box, or set starting values.

References

Baddeley, A. and Turner, R. (2005) spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**, 1–42. DOI: 10.18637/jss.v012.i06

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

Carothers, A. D. (1979) The effects of unequal catchability on Jolly–Seber estimates. *Biometrics* **29**, 79–100.

Distiller, G. and Borchers, D. L. (2015) A spatially explicit capture–recapture estimator for single-catch traps. *Ecology and Evolution* **5**, 5075–5087.

Efford, M. G. (2004) Density estimation in live-trapping studies. *Oikos* **106**, 598–610.

Efford, M. G. (2022). secr: Spatially explicit capture–recapture models. R package version 4.5.6. https://CRAN.R-project.org/package=secr/

Efford, M. G. (2023) ipsecr: An R package for awkward spatial capture–recapture data. *Methods in Ecology and Evolution* **14**, 1182–1189.

Efford, M. G., Dawson, D. K. and Robbins C. S. (2004) DENSITY: software for analysing capture-recapture data from passive detector arrays. *Animal Biodiversity and Conservation* **27**, 217–228.

Groemping, U. (2014). R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs. *Journal of Statistical Software*, **56**, 1–56. https://www.jstatsoft.org/article/view/v056i01.

Kodi, A. R., Howard, J., Borchers, D. L., Worthington, H., Alexander, J. S., Lkhagvajav, P., Bayandonoi, G., Ochirjav, M., Erdenebaatar, S., Byambasuren, C., Battulga, N., Johansson, Ö., and Sharma, K. (2024) Ghostbusting - reducing bias due to identification errors in spatial capture–recapture histories. *Methods in Ecology and Evolution* **15**, 1060–1070.

Pledger, S. and Efford, M. G. (1998) Correction of bias due to heterogeneous capture probability in capture-recapture studies of open populations. *Biometrics* **54**, 888–898.