Changes in **secr** 4.0

Murray Efford

2021-05-19

Contents

Introduction

Version 4 has been substantially re-written. Most code from earlier versions will still work. However, there are major changes to some features, and results may differ. This note describes the major changes in version 4.0 and additions in 4.1 and 4.2. Version 4.1 fixed some bugs in **secr** 4.0 (covariate models were prone to error). Version 4.2 introduced a new mechanism for setting the number of threads in multi-threaded functions. The default number of threads has been capped at two; to take full advantage of multi-threading it is now usually necessary to specify 'ncores' at least once.

Table 1. Key changes in **secr** 4.0 (see also see Appendix 1)

Category	Feature	Status
General	Multi-threading of likelihood computation	New
General	Fast evaluation for proximity and count detectors	New, optional
General	Individual mask subsets	New, optional
Mark-resight	Conditional formulation of likelihood for unidentified, marked	Discontinued
Data	Fitted models	Most fitted models have been omitted to save space (see Appendix 2)

Multi-threading

Multi-threading is a way to use multiple processor cores to speed computation. **secr** 4.0 performs multi-threading with `parallelFor` in **RcppParallel** (Allaire et al. 2019). Multi-threading is used in computationally intensive functions, particularly likelihood evaluation in `secr.fit` (Appendix 3).

Multi-threading was not applied in **secr** 4.0 for polygon and transect detector types because these used integration code from the R API (Rdqags). Multi-threading is enabled for these detector types in **secr** 4.1 and later, using the integration interface in **RcppNumerical** (Qiu et al. 2019).

Multi-threading in `secr.fit` is at the level of entire detection histories: histories are distributed over cores (in general the number of histories is much greater than the number of cores). In a typical application on a quad-core desktop running Windows (8 virtual cores) this can speed up model fitting by a factor of 3–4.

By default, the code will use only two threads (cores). The number of threads may be controlled directly with the `ncores` argument of `secr.fit` etc. or with the function `setNumThreads` (new in **secr** 4.2; see `?setNumThreads` for details). If `ncores = NULL` the number of threads is taken from the environment variable RCPP_PARALLEL_NUM_THREADS that is initialised to 2 when the package is loaded. The variable is updated by `setNumThreads` and whenever a function is called with specified `ncores`.

```
# For example,
setNumThreads(8)
secr.fit(captdata)
# is equivalent to
secr.fit(captdata, ncores = 8)
```

Parallel processing in **secr** previously relied on the **parallel** package, with mixed success (setting ncores>1 could even slow down computation). The old function `par.secr.fit` no longer speeds up computation when ncores>1, but it is retained.

Faster fitting of 'proximity' and 'count' data

Data from binary proximity ('proximity') and count proximity ('count') detectors often take the form of large, sparse 3-D arrays. For one session the array has dimensions n (number of detected individuals), s (number of occasions) and k (number of detectors), and most cells will be zero.

The code in **secr** 3.2 always iterated over all cells of the array; this does not constrain the models that can be fitted. For many models it is faster to operate on a compressed version of the data, and this is the default in **secr** 4.0 and later versions. Fast processing (and the associated compression) can be bypassed by setting the details argument `fastproximity = FALSE`. This may be needed to avoid restrictions.

Data compression

Two types of compression are applied automatically by default (`fastproximity = TRUE`):

1. Binary proximity data as binomial count

Binary data for a particular animal and detector over s occasions are summed and modelled as a binomial count of size s on a single occasion. This option has been available with manual preprocessing using code such as

```
CH1 <- reduce(CH, outputdetector = 'count', by = 'all')
secr.fit(CH1, binomN = 1)
```

The capthist object from the `reduce` step includes detectors with a usage attribute that records the number of occasions s pooled for each detector; `binomN = 1` directs `secr.fit` to find the binomial size in the usage attribute.

2. Pass only non-zero counts

Even with compression over occasions, the number of detections at most detectors is usually zero. It is efficient to preprocess the data for each animal into a list of those detectors with non-zero counts, and the associated counts. The likelihood code in **secr** automatically applies this compression under the fastproximity option.

Operation

Compressed counts may be modelled as Poisson or binomial. A Poisson model is used for the compressed counts if detector type `count` appears anywhere in the input data and a Poisson model is specified via the argument `binomN = 0`.

Otherwise, a binomial model is used for the compressed counts. The size of the binomial model is specific to each detector. It is set internally by manipulating the `usage` attribute. Most simply (`binomN = NULL`) this is merely the number of occasions or the sum over occasions of the input `usage` if that exists. For specified `binomN` > 1 the `usage` is multiplied by `binomN`.

Restrictions

`fastproximity = TRUE` is applied by `secr.fit` under these conditions:

- all detectors are of type 'proximity', 'count' or 'capped'
- no learned response in model (~b, ~bk etc. prohibited)
- no temporal variation in model (~t, ~T, ~tcov etc. prohibited)
- `groups = NULL`
- no variation over detectors in model [potential, not implemented]

If any condition is not met then processing continues as if `fastproximity = FALSE`.

Individual mask subsets

The base algorithm of `secr` treats all points on the habitat mask as potential locations for the activity centre of each individual i.e. integration is over the area of the mask. When the mask is large relative to the home range, most points will contribute almost nothing to the integral, and time is wasted including them in the summation.

A more efficient approach is to consider only mask points in the vicinity of the known detections of an individual. In **secr** 4.0 and later versions the user may specify a maximum mask radius (`details$maxdistance`) with respect to the centroid of detections, computed separately for each animal. The centroid is the geometric mean of detection locations for each individual. If maxdistance is not specified then summation is over all mask points. The entire mask is always used for the likelihood component due to undetected individuals.

A simple example:

```
library(secr)
fit0 <- secr.fit (captdata, buffer = 100, trace = FALSE)
fitims <- secr.fit (captdata, buffer = 100, details = list(maxdistance = 100), trace = FALSE)
fits <- secrlist(fit0, fitims)
sapply(fits, '[[', 'proctime')
```

```
##   fit0.elapsed fitims.elapsed
##           4.41           3.48
```

```
collate(fits)[1,,,'D']
```

```
##        estimate SE.estimate      lcl      ucl
## fit0   5.479803   0.6467406 4.351622 6.900471
## fitims 5.500117   0.6481336 4.369301 6.923598
```

It is up to the user to select a limit that is large enough not to affect the likelihood. This should be larger than the expected home range radius, as the centroid is only an approximation to the activity centre. The value 5σ is suggested for a half-normal detection function.

Using individual mask subsets makes fitting much faster, particularly when activity areas are small relative to the overall extent of the mask (examples in Table 2).

Table 2. Relative timings for model fitting in `secr.fit` with two scenarios for halfnormal σ in relation to spacing s between detectors on a 10×10 square grid. Using individual, local mask achieves >5-fold speed improvement with the usual approach to likelihood evaluation ('Basic'), but only minor improvement when other tools are used to speed up execution ('Fast'). $D = (0.8/\sigma)^2$; $\lambda_0 = 0.1/\sigma^{0.5}$.

```
##                          Method
## Scenario                  Basic Fast
##   sigma = s, full mask     303    7
##   sigma = 2s, full mask    112    6
##   sigma = s, local mask     51    4
##   sigma = 2s, local mask    24    3
```

Backward compatibility

Code that worked in previous versions should work in **secr** 4.2, but this is not guaranteed. Problems are likely with models fitted in earlier versions, as these will not include the new components 'designD', 'designNE' and 'learnedresponse'. The solution is to refit the models from scratch in the new **secr**, or continue using the earlier version (**secr** $\leq$ 3.2).

The default behaviour of **secr** $\geq$ 4.0 is to use the new 'fastproximity' algorithms where possible (see above). This forces the detector type to 'count' and the number of occasions to 1. To suppress this behaviour and return to something like **secr** $\leq$ 3.2, set `details = list(fastproximity = FALSE)` in the call to `secr.fit`.

Fewer fitted models are included with the datasets. If you rely on a missing model for one of the built-in datasets, fit it yourself with the code in Appendix 2.

References

Allaire, J. J., Francois, R., Ushey, K., Vandenbrouck, G., Geelnard, M. and Intel (2019) RcppParallel: Parallel Programming Tools for 'Rcpp'. R package version 4.4.4. https://CRAN.R-project.org/package=RcppParallel.

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

Efford, M. G. (2019a) secr: Spatially explicit capture-recapture models. R package version 3.2.0. https://CRAN.R-project.org/package=secr.

Efford, M. G. (2019b) secr: Spatially explicit capture-recapture models. R package version 4.0.1. https://CRAN.R-project.org/package=secr.

Qiu, Y., Balan, S., Beall, M., Sauder, M., Okazaki, N. and Hahn, T. (2019) RcppNumerical: 'Rcpp' Integration for Numerical Computing Libraries. R package version 0.3-3. https://CRAN.R-project.org/package=RcppNumerical

Appendix 1. Specific changes in secr 4.0

Function	Change
`secr.fit` etc.	Argument `ncores` now refers to the number of cores used for multi-threading by **RcppParallel**, not the number of clusters used by **parallel**. In **secr** 4.2 the default NULL is to use the value of the RCPP_PARALLEL_NUM_THREADS environment variable, initially 2
`secr.fit`	autoini = 'all' for start values from multi-session data (experimental)
`secr.fit`	New components designD, designNE and learnedresponse in output `secr` object
`secr.fit`	LLonly = TRUE uses automatic parameter values when start not provided
`secr.fit`	LLonly = TRUE output has new attributes 'npar', 'preptime' and 'LLtime'
`sim.capthist`	Detection function for polygons and transects must be of hazard type (14:19) as required by `secr.fit`
`make.systematic`	New argument `order` controls sequence of clusters in generated traps object
`par.secr.fit`	No longer automatically sets ncores=1 in each call to `secr.fit`
`RSE`	New function to extract precision of 'real' parameter estimates
`summary.capthist`	Includes summary of individual covariates
`summary.secr`	Component 'versiontime' includes elapsed time
`autoini`, `region.N`, `suggest.buffer`	Acquire `ncores` argument
`fx.secr`, `fxi.contour`	Argument `normal` dropped
`mask.check`, `pmixProfileLL`, `simulate.secr`	Argument `ncores` dropped
`read.SPACECAP`, `write.SPACECAP`	Removed

The new components in 'secr' objects returned by `secr.fit` are described on the help page for the function.

Appendix 2. Code for fitting models omitted from secr $\geq$ 4.0

Earlier versions of **secr** included pre-fitted models to demonstrate various features. Most of these have been dropped in **secr** $\geq$ 4.0 to save space and to reduce maintenance costs. The omitted models may be regenerated with the code in this appendix.

House mouse

```r
morning <- subset(housemouse, occ = c(1,3,5,7,9))
afternoon <- subset(housemouse, occ = c(2,4,6,8,10))

morning.0    <- secr.fit(morning, buffer = 20, trace = FALSE)
morning.h2   <- secr.fit(morning, buffer = 20, model=list(g0~h2), trace = FALSE)
morning.0h2  <- secr.fit(morning, buffer = 20, model=list(sigma~h2), trace = FALSE)
morning.h2h2 <- secr.fit(morning, buffer = 20, model=list(g0~h2, sigma~h2), trace = FALSE)
morning.t    <- secr.fit(morning, buffer = 20, model=g0~t, trace = FALSE)
morning.b    <- secr.fit(morning, buffer = 20, model=g0~b, trace = FALSE)

housemouse.0 <- secr.fit (housemouse, buffer = 20, trace = FALSE)
housemouse.ampm <- secr.fit (housemouse, model = g0~tcov, timecov =
    c(0,1,0,1,0,1,0,1,0,1,0,1), buffer = 20, trace = FALSE)
housemouse.ampmh2h2 <- secr.fit (housemouse, model=list(g0~tcov+h2, sigma~tcov+h2),
    timecov = c(0,1,0,1,0,1,0,1,0,1,0,1), buffer = 20, trace = FALSE)
```

Stoat

```
stoat.model.HN <- secr.fit(stoatCH, buffer = 1000, detectfn = 0, trace = FALSE)
stoat.model.HZ <- secr.fit(stoatCH, buffer = 1000, detectfn = 1, trace = FALSE,
                           biasLimit = NA)
stoat.model.EX <- secr.fit(stoatCH, buffer = 1000, detectfn = 2, trace = FALSE)
```

Ovenbird

```
ovenmask <- make.mask(traps(ovenCH), type = 'pdot', buffer = 400,
    spacing = 15, detectpar = list(g0 = 0.03, sigma = 90), nocc = 10)
ovenbird.model.1 <- secr.fit(ovenCH, mask = ovenmask, trace = FALSE)
ovenbird.model.1b <- secr.fit(ovenCH, mask = ovenmask, model = list(g0 ~ b), trace = FALSE)
ovenbird.model.1T <- secr.fit(ovenCH, mask = ovenmask, model = list(g0 ~ T), trace = FALSE)
ovenbird.model.h2 <- secr.fit(ovenCH, mask = ovenmask, model = list(g0~h2), trace = FALSE)
ovenbird.model.D <- secr.fit(ovenCH, mask = ovenmask, model = list(D ~ Session),
                             trace = FALSE)
```

Brushtail possum

```
possum.model.0 <- secr.fit(possumCH, mask = possummask, trace = FALSE)
possum.model.b <- secr.fit(possumCH, mask = possummask, model = g0~b,
                           trace = FALSE)
possum.model.Ds <- secr.fit(possumCH, mask = possummask, model =
    list(D ~ d.to.shore), link = list(D = 'identity'), trace = FALSE,
    method = 'Nelder-Mead')
```

Deer mouse

```
ESG.0 <- secr.fit(deermouse.ESG, trace = FALSE)
ESG.b <- secr.fit(deermouse.ESG, model=g0~b, trace = FALSE)
ESG.t <- secr.fit(deermouse.ESG, model=g0~t, trace = FALSE)
ESG.h2 <- secr.fit(deermouse.ESG, model=g0~h2, trace = FALSE)
WSG.0 <- secr.fit(deermouse.WSG, model=g0~1, trace = FALSE)
WSG.b <- secr.fit(deermouse.WSG, model=g0~b, trace = FALSE)
WSG.t <- secr.fit(deermouse.WSG, model=g0~t, trace = FALSE)
WSG.h2 <- secr.fit(deermouse.WSG, model=g0~h2, trace = FALSE)
```

DENSITY demo data

```
secrdemo.0 <- secr.fit (captdata, trace = FALSE)
secrdemo.CL <- secr.fit (captdata, CL = TRUE, trace = FALSE)
secrdemo.b <- secr.fit (captdata, model = list(g0 = ~b), trace = FALSE)
```

Ovenbird song

```
signalCH.525 <- subset(signalCH, cutval = 52.6)
omask <- make.mask(traps(signalCH), buffer=200)
ostart <- c(log(20), 80, log(0.1), log(2))
ovensong.model.1 <- secr.fit( signalCH.525, mask = omask, start = ostart, detectfn = 11,
                            trace = FALSE )
ovensong.model.2 <- secr.fit( signalCH.525, mask = omask, start = ostart, trace = FALSE )
```

Appendix 3. Functions calling multi-threaded C++ in secr $\geq$ 4.1

These internal C++ functions use multi-threading. Set 'ncores = 1' or the `secr.fit` details argument 'grain = 0' to suppress multi-threading. 'Level' refers to the units that are distributed over parallel threads. `makegkPointcpp` precomputes probability and hazard values for each combination of parameter values, detector, and mask point.

Function	Level	Called by user-facing functions…
makegkPointcpp	cell of habitat mask	autoini, esa, expected.n, fxi, region.N, secr.fit, sim.secr
makegkPolygoncpp	cell of habitat mask	esa, expected.n, fxi, region.N, secr.fit
fasthistoriescpp	detection history	secr.fit
polygonhistoriescpp	detection history	secr.fit
pdotpointcpp	point (of mask)	pdot (used in many functions)
hdotpolycpp	point (of mask)	pdot (used in many functions)
sightingchatcpp	detection history	secr.fit
signalhistoriescpp	detection history	secr.fit
simplehistoriescpp	detection history	secr.fit
simplehistoriesfxicpp	detection history	fxi