Modelling population trend in **secr** 5.2

Murray Efford

2025-01-26

Contents

Introduction

secr is primarily for estimating closed population density (density at one point in time), but multi-session data may also be modelled to describe population trend over time. The fundamentals of multi-session analysis are described separately in secr-multisession.pdf. This document describes methods specifically for population trend, defined as change in density between sessions and measured by the finite rate of increase $\lambda_t = D_{t+1}/D_t$.

Simple trend analysis may be achieved with a particular coding of the session factor as described in earlier versions of secr-multisession.pdf and repeated here in Appendix 1. The more flexible methods described here allow the direct estimation of λ_t, possibly including covariate effects. The newer methods require **secr** $\geq$ 4.6.2 2023-09-30.

Direct estimation of λ

A satisfying and flexible approach is to parameterize the density model in terms of the initial density D_1 and the finite rates of increase λ_t for the remaining sessions (λ_1 refers to the density increase between Session 1 and Session 2, etc.).

The Dlambda parameterization

Reparameterization of the density model is achieved internally in `secr.fit` by manipulating the density design matrix to provide a new array of mask-cell- and session-specific densities at each evaluation of the full likelihood. This happens when the details argument 'Dlambda' is set to TRUE. The density model (D~) and the fitted beta parameters take on a new meaning determined by the internal function `Dfn2`. More explanation is given later.

Now, fitting the ovenbird model with D~1 results in two density parameters (density in session 1, constant finite rate of increase across remaining sessions):

```
library(secr)
msk <- make.mask(traps(ovenCH[[1]]), buffer = 300, nx = 32)
fit1  <- secr.fit(ovenCH, model = D~1, mask = msk, trace = FALSE,
                  details = list(Dlambda = TRUE))
coef(fit1)
```

```
##              beta    SE.beta        lcl         ucl
## D.D1   0.03202408 0.19132459 -0.3429652  0.40701338
## D.D2  -0.06385772 0.07015152 -0.2013522  0.07363672
## g0    -3.56191929 0.15065352 -3.8571948 -3.26664381
## sigma  4.36396832 0.08105209  4.2051092  4.52282749
```

Density-relevant beta parameters have names starting with 'D.'[1]. The first is the log initial density; others relate to the λ parameters.

The summary function `predictDlambda`

To make the most of the reparameterization we need the special function `predictDlambda` to extract the lambda estimates (the simple `predict` method does not work).

```
predictDlambda (fit1)
```

```
##          estimate SE.estimate       lcl      ucl
## D1      1.0325424  0.19937244 0.7096629 1.502324
## lambda1 0.9381385  0.06589289 0.8176244 1.076416
## lambda2 0.9381385  0.06589289 0.8176244 1.076416
## lambda3 0.9381385  0.06589289 0.8176244 1.076416
## lambda4 0.9381385  0.06589289 0.8176244 1.076416
```

This is an advance on the earlier approach using sdif contrasts, as we have constrained λ to a constant.

Covariate and other models

The method allows many covariate models for λ. We can fit a time trend in λ using:

```
fit2  <- secr.fit(ovenCH, model = D~Session, mask = msk, trace = FALSE,
                  details = list(Dlambda = TRUE))
predictDlambda (fit2)
```

```
##          estimate SE.estimate       lcl      ucl
## D1      0.9013175  0.21201769 0.5719246 1.420420
## lambda1 1.1473738  0.22123502 0.7889927 1.668541
## lambda2 0.9997690  0.09247379 0.8343250 1.198020
## lambda3 0.8711529  0.08598817 0.7182558 1.056598
## lambda4 0.7590828  0.15341569 0.5128314 1.123579
```

[1]Their indices are listed in component 'D' of the 'parindx' component of the fitted model (e.g. `fit1$parindx$D`), but you are unlikely to need this.

Session-specific λ (lower-case 'session') provide a direct comparison with the original analysis:

```
fit3  <- secr.fit(ovenCH, model = D~session, mask = msk, trace = FALSE,
                  details = list(Dlambda = TRUE))
predictDlambda (fit3)
```

```
##          estimate SE.estimate       lcl      ucl
## D1      0.9202572   0.2276249 0.5707988 1.483663
## lambda1 1.0469025   0.3313263 0.5713714 1.918200
## lambda2 1.1818196   0.3496447 0.6698608 2.085056
## lambda3 0.7307856   0.2256698 0.4044803 1.320330
## lambda4 0.8420810   0.2941256 0.4330684 1.637387
```

Model selection procedures apply as usual:

```
AIC(fit1, fit2, fit3, criterion = 'AIC')[,-6]
```

```
##                       model   detectfn npar    logLik      AIC  dAIC  AICwt
## fit1       D~1 g0~1 sigma~1 halfnormal    4 -930.7267 1869.453 0.000 0.5692
## fit2 D~Session g0~1 sigma~1 halfnormal    5 -930.0667 1870.133 0.680 0.4051
## fit3 D~session g0~1 sigma~1 halfnormal    8 -929.8226 1875.645 6.192 0.0257
```

Session covariates are readily applied. The covariate for the second session predicts $\lambda_1 = D_2/D_1$, for the third session predicts $\lambda_2 = D_3/D_2$, etc. The covariate for the first session is discarded (remember D_1 is constant). This all may be confusing, but you can work it out, and it saves extra coding.

```
covs <- data.frame(acov = c(0,2,1,1,2))   # a fabricated covariate
fit4  <- secr.fit(ovenCH, model = D~acov, mask = msk, trace = FALSE,
                  details = list(Dlambda = TRUE), sessioncov = covs)
predictDlambda (fit4)
```

```
##          estimate SE.estimate       lcl      ucl
## D1      1.0280435   0.2124420 0.6885713 1.534879
## lambda1 0.9501129   0.2120216 0.6167783 1.463596
## lambda2 0.9302939   0.1452784 0.6862669 1.261094
## lambda3 0.9302939   0.1452784 0.6862669 1.261094
## lambda4 0.9501129   0.2120216 0.6167783 1.463596
```

Fixing coefficients

Another possibility is to fit the model with fixed trend (the second beta coefficient corresponds to lambda, before).

```
fit5 <- secr.fit(ovenCH, model = D~1, mask = msk, trace = FALSE,
    details = list(Dlambda = TRUE, fixedbeta = c(NA, log(0.9), NA, NA)))
predictDlambda(fit5)
```

```
##         estimate SE.estimate       lcl     ucl
## D1      1.115395   0.1538136 0.8523147 1.45968
## lambda1 0.900000   0.0000000 0.9000000 0.90000
## lambda2 0.900000   0.0000000 0.9000000 0.90000
## lambda3 0.900000   0.0000000 0.9000000 0.90000
## lambda4 0.900000   0.0000000 0.9000000 0.90000
```

For comparison, this can be achieved more conventionally by fixing the beta coefficient in a model with log-linear trend over sessions (D~Session):

```
fit6 <- secr.fit(ovenCH, model = D~Session, mask = msk, trace = FALSE,
    details = list(Dlambda = FALSE, fixedbeta = c(NA, log(0.9), NA, NA)))
```

```
t(sapply(predict(fit6), '[', 'D', ))
```

```
##                              link  estimate  SE.estimate lcl       ucl
## session = 2005, Session = 0 "log" 1.115389  0.1538133   0.8523095 1.459673
## session = 2006, Session = 1 "log" 1.00385   0.138432    0.7670785 1.313706
## session = 2007, Session = 2 "log" 0.9034654 0.1245888   0.6903707 1.182335
## session = 2008, Session = 3 "log" 0.8131188 0.1121299   0.6213336 1.064102
## session = 2009, Session = 4 "log" 0.7318069 0.1009169   0.5592003 0.9576916
```

Technical notes and tips

`Dfn2` performs some tricky manipulations. You can see the code by typing `secr:::Dfn2`. A column is pre-pended to the density design matrix specifically to model the initial density; this takes the value one in Session 1 and is otherwise zero. Other columns in the design matrix are set to zero for the first session. Session-specific density on the link (log) scale is computed as the cumulative sum across sessions of the initial log density and the modelled log-lambda values.

Note –

- The model allows detector locations and habitat masks to vary between sessions.
- The coding of `Dfn2` relies on a log link function for density.
- Dlambda is ignored for single-session data and conditional-likelihood (CL) models.
- The method is not (yet) suitable for group models.
- The default start values for D in `secr.fit` work well: all lambda are initially 1.0 ($\log(\lambda_t) = 0$ for all t).
- If session covariates are used in any model, AICcompatible() expects the argument 'sessioncov' to be included in all models.

Caveats

Spatial variation in density

D for session 1 is constant over space. It is not possible in the present version of **secr** to model simultaneous spatial variation in density or λ, and using Dlambda with a density model that includes spatial covariates will cause an error.

Underestimation of sampling variance

Underestimation of sampling variance is expected when a trend model is fitted to temporal samples with incomplete population turnover between sessions. The product likelihood assumes a new realisation of the underlying population process for each session. If in actuality much of the sampled population remains the same (the same individuals in the same home ranges) then the precision of the trend coefficient will be overstated.

The effect is often small. Possible solutions are to fit an open population model (e.g., in **openCR** Efford and Schofield 2020) or to apply some form of bootstrapping.

References

Efford, M. G. and Schofield, M. R. (2020) A spatial open-population capture–recapture model. *Biometrics* **76**, 392–402.

Appendix 1. Older approach

From an earlier version of secr-multisession.pdf:

A model with initial uppercase 'Session' fits a *trend* across sessions using the session number as the predictor. A trend model for density may be interesting if the sessions fall in some natural sequence, such as a series of annual samples (as in the ovenbird dataset ovenCH). The fitted trend is linear on the link scale; using the default link function for density ('log') this corresponds to exponential growth or decline if samples are equally spaced in time.

The pre-fitted model `ovenbird.model.D` provides an example. The coefficient 'D.Session' is the rate of change in log(D):

```
library(secr)
coef(ovenbird.model.D)
```

```
##                  beta    SE.beta        lcl        ucl
## D          0.03171125 0.19145984 -0.3435431  0.4069656
## D.Session -0.06385900 0.07015149 -0.2013534  0.0736354
## g0        -3.56193337 0.15062263 -3.8571483 -3.2667184
## sigma      4.36411056 0.08114866  4.2050621  4.5231590
```

The overall finite rate of increase λ (equivalent to Pradel's lambda) is given by

```
beta <- coef(ovenbird.model.D)['D.Session','beta']
sebeta <- coef(ovenbird.model.D)['D.Session','SE.beta']
exp(beta)
```

```
## [1] 0.9381373
```

Confidence intervals may also be back-transformed with `exp`. To back-transform the SE use the delta-method approximation `exp(beta) * sqrt(exp(sebeta^2)-1)` = 0.0658928.

This is fine for a single overall lambda. However, if you are interested in successive estimates (session 1 to session 2, session 2 to session 3 etc.) the solution is slightly more complicated. One trick is to use 'backward difference' coding of the levels of the factor `session`, specified with the new details argument 'contrasts'. This coding is provided by the function `contr.sdif` in the **MASS** package (e.g., Venables and Ripley 1999 Section 6.2).

```
msk <- make.mask(traps(ovenCH[[1]]), buffer = 300, nx = 32)
fit  <- secr.fit(ovenCH, model = D~session, mask = msk, trace = FALSE,
                 details = list(contrasts = list(session = MASS::contr.sdif)))
coef(fit)
```

```
##                            beta    SE.beta        lcl        ucl
## D                   -0.10602952 0.13809584 -0.3766924  0.1646334
## D.session2006-2005   0.04583767 0.30898483 -0.5597615  0.6514368
## D.session2007-2006   0.16705460 0.28968575 -0.4007190  0.7348282
## D.session2008-2007  -0.31365718 0.30181343 -0.9052006  0.2778863
## D.session2009-2008  -0.17185180 0.33930068 -0.8368689  0.4931653
## g0                  -3.56085953 0.15063281 -3.8560944 -3.2656247
## sigma                4.36392176 0.08104712  4.2050723  4.5227712
```

This estimates each session-specific log(lambda) directly - just back-transform as before:

```
beta <- coef(fit)[2:5, 'beta']
sebeta <- coef(fit)[2:5, 'SE.beta']
exp(beta)   # lambda-hat
```

```
## [1] 1.0469045 1.1818188 0.7307695 0.8421040
```

```
exp(beta) * sqrt(exp(sebeta^2)-1)    # SE(lambda-hat)
```

```
## [1] 0.3313541 0.3496657 0.2256754 0.2941507
```

The lambda estimates match the density ratios computed directly (a warning is suppressed):

```
D <- collate(fit, realnames='D')[,1,1,]
D[2:5]/D[1:4]
```

```
## session=2006 session=2007 session=2008 session=2009
##    1.0469045    1.1818188    0.7307695    0.8421040
```

The ovenbird population appeared to increase in density for two years and then decline for two years. The effects are far from significant.

Appendix 2. Simulated example with known λ

In this example the true λ_t are (1, 2, 1, 1, 1.5) for sessions 1–5.

```
# traps and mask
grid144 <- make.grid(12,12, detector='proximity', spacing = 2)
grid144mask <- make.mask(grid144, spacing = 0.5, buffer = 4)
# simulate some data
set.seed(123)
# some warnings are suppressed
pop <- sim.popn(D = 3000, core = grid144, buffer = 4, nsessions = 6,
                details = list(lambda = c(1, 2, 1, 1, 1.5)))
ch <- sim.capthist(grid144, popn = pop, detectfn = 'HHN', renumber = FALSE,
                   detectpar = list(lambda0 = 0.5, sigma = 1))
summary(ch, terse = TRUE)
```

```
##              1   2    3    4    5    6
## Occasions    5   5    5    5    5    5
## Detections 627 671 1228 1188 1163 1833
## Animals    192 211  388  377  364  569
## Detectors  144 144  144  144  144  144
```

$\hat{\lambda}$ from sequence of density estimates

```
setNumThreads(18)   # adjust as required
```

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

```
fit0 <- secr.fit(ch, mask = grid144mask, detectfn = 'HHN', model = D ~ session,
    trace = FALSE)
D <- sapply(predict(fit0), '[[', 'D', 'estimate')
D[2:6]/D[1:5]
```

```
## session = 2 session = 3 session = 4 session = 5 session = 6
##   1.0989973   1.8388559   0.9716593   0.9655128   1.5631850
```

Old style $\hat{\lambda}$ using contr.sdif

```
fit1 <- secr.fit(ch, mask = grid144mask, detectfn = 'HHN', model = D ~ session,
    trace = FALSE, details = list(contrasts = list(session = MASS::contr.sdif)))
```

```
# lambda estimates
exp(coef(fit1)[2:6,1])
```

```
## [1] 1.0989573 1.8388750 0.9716453 0.9655213 1.5631831
```

Session-specific $\hat{\lambda}$

```
fit2 <- secr.fit(ch, mask = grid144mask, detectfn = 'HHN', model = D~session,
    trace = FALSE, details = list(Dlambda = TRUE))
predictDlambda(fit2)
```

```
##             estimate  SE.estimate          lcl         ucl
## D1      3024.3167722 218.59153305 2625.3322097 3483.936968
## lambda1    1.0989925   0.10987997    0.9038598    1.336252
## lambda2    1.8388655   0.15759569    1.5550098    2.174537
## lambda3    0.9716466   0.07041738    0.8431418    1.119737
## lambda4    0.9655201   0.07117930    0.8357853    1.115393
## lambda5    1.5631800   0.10545086    1.3697862    1.783878
```

Using session covariate

The covariate used here is a perfect match to the original λ - completely unrealistic, but a test of the idea.

```
covs <- data.frame(sesscov = log(c(1, 1, 2, 1, 1, 1.5)))
fit3 <- secr.fit(ch, mask = grid144mask, detectfn = 'HHN', model = D ~ sesscov,
    sessioncov = covs, trace = FALSE, details = list(Dlambda = TRUE))
predictDlambda(fit3)
```

```
##             estimate  SE.estimate          lcl         ucl
## D1      3138.7003752 147.14963649 2863.2906572 3440.600772
## lambda1    0.9901158   0.03255250    0.9283424    1.056000
## lambda2    1.9563585   0.14237976    1.6966081    2.255877
## lambda3    0.9901158   0.03255250    0.9283424    1.056000
## lambda4    0.9901158   0.03255250    0.9283424    1.056000
## lambda5    1.4746737   0.04756057    1.3843645    1.570874
```