Density surfaces in **secr** 4.6

Murray Efford

2023-12-20

Contents

Introduction

The formulation of spatially explicit capture–recapture (SECR) by Borchers and Efford (2008) allows for population density to vary over space. Models of density may include spatial covariates (e.g., habitat class), or spatial trend. This document describes the fitting of density models in **secr** 4.6 by means of a worked example.

Density surfaces - some background

SECR fits a state model and an observation model to data from incomplete spatial detections of individuals. The observation model is based on a distance-dependent detection function with parameters g_0 (or λ_0)

and σ, as described in secr-overview.pdf. The state model, with which we are concerned here, is a spatial Poisson process for animal range centres. The expected value of the process (centres per unit area) is either homogeneous (constant over space) or inhomogeneous (varying over space). The notation $D(\mathbf{x};\phi)$ covers both possibilities: density (D) is a function of location (the vector $\mathbf{x}$ represents a pair of x- and y-coordinates). If density is homogeneous its expected value is a flat surface and the parameter ϕ is one number, the density. In most other cases, ϕ is a vector of several parameters.

To model variation in a density surface we need to maximise the full likelihood. Maximising the conditional likelihood (conditional on n, the number of observed individuals) is a way to estimate the observation model; to go from there to a Horvitz-Thompson estimate of density we assume that density is homogeneous. Here we are concerned with inhomogeneous models that are all fitted with `CL = FALSE` in `secr.fit`.

Although $D(\mathbf{x};\phi)$ may be a smooth function, in **secr** we evaluate it only at the fixed points of a habitat 'mask' (think of these as the cell centres of a pixellated or raster representation). A mask defines the region of habitat relevant to a particular study: in the simplest case it is a buffered zone inclusive of the detector locations, but it may exclude interior areas of non-habitat or have an irregular outline.

A density model $D(\mathbf{x};\phi)$ is specified in the 'model' argument of `secr.fit`[1]. Spatial covariates, if any, are needed for each mask point; they are stored in the 'covariates' attribute of the mask. Results from fitting the model (including the estimated coefficients ϕ) are saved in an object of class 'secr'. To visualise a fitted density model we first evaluate it at each point on a mask with the function `predictDsurface` to create an object of class 'Dsurface'. A Dsurface is a mask with added density data, and plotting a Dsurface is like plotting a mask covariate.

Brushtail possum example

For illustration we use a brushtail possum (*Trichosurus vulpecula*) dataset from the Orongorongo Valley, New Zealand. Possums were live-trapped in mixed evergreen forest near Wellington for nearly 40 years (Efford and Cowan 2004). Single-catch traps were set for 5 consecutive nights, three times a year. The dataset 'OVpossumCH' has data from the years 1996 and 1997. The study grid was bounded by a shingle riverbed to the north and west. See ?OVpossum in **secr** for more details.

First we import data for the habitat mask from a polygon shapefile included with the package:

```
library(secr)
setNumThreads(18)    # depends on machine
```

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

```
datadir <- system.file("extdata", package = "secr")
OVforest <- sf::st_read (paste0(datadir, "/OVforest.shp"), quiet = TRUE)
leftbank <- read.table(paste0(datadir,"/leftbank.txt"))[21:195,]  ## drop some we don't need
options(digits = 6, width = 95)
```

OVforest is a simple features (sf) object defined in package **sf**. Now we can build a habitat mask object, selecting the first two polygons in OVforest and discarding the third that lies across the river. The attribute table of the shapefile (and hence OVforest) includes a categorical variable 'forest' that is either 'beech' or 'nonbeech'; 'addCovariates' attaches these data to each cell in the mask.

```
ovtrap <- traps(OVpossumCH[[1]])
ovmask <- make.mask(ovtrap, buffer = 120, type = "trapbuffer",
    poly = OVforest[1:2,], spacing = 7.5, keep.poly = FALSE)
ovmask <- addCovariates(ovmask, OVforest[1:2,])
```

```
## Warning: attribute variables are assumed to be spatially constant throughout all geometries
```

[1] Technically, it may also be specified in a user-written function supplied to `secr.fit` (see Appendix 1), but you are unlikely to need this.

Plotting is easy:

```
par(mar = c(1,6,2,8))
forestcol <- terrain.colors(6)[c(4,2)]
plot(ovmask, cov="forest", dots = FALSE, col = forestcol)
plot(ovtrap, add = TRUE)
par(cex = 0.8)
terra::sbar(d = 200, xy = c(2674670, 5982930), type = 'line', divs = 2,
            below = "metres", labels = c("0","100","200"), ticks = 10)
terra::north(xy = c(2674670, 5982830), label = "N")
```

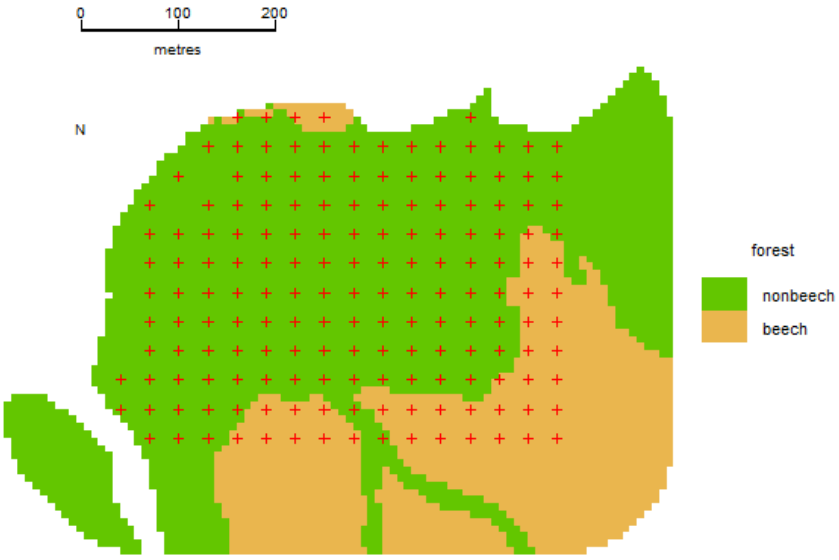


Figure 1: OVmask

We fit some simple models to data from February 1996 (session 49). Some warnings are suppressed for clarity.

```
base.args <- list(capthist = OVpossumCH[[1]], mask = ovmask, trace = FALSE)
args.0 <- c(base.args, model = D ~ 1)
args.Dxy <- c(base.args, model = D ~ x + y)
args.Dxy2 <- c(base.args, model = D ~ x + y + x2 + y2 + xy)
args.Dforest <- c(base.args, model = D ~ forest)
arglist <- list(null = args.0, Dxy = args.Dxy, Dxy2 = args.Dxy2, Dforest = args.Dforest)
fits <- par.secr.fit(arglist, ncores = 4)
```

```
## Completed in 1.42 minutes at 13:27:34 21 May 2023
```

```
AIC(fits, criterion = "AIC")[,-2]
```

```
##                                          model npar   logLik     AIC    AICc  dAIC  AICwt
## fit.Dxy2    D~x + y + x2 + y2 + xy g0~1 sigma~1    8 -1549.32 3114.64 3115.31 0.000 0.4429
## fit.Dxy                   D~x + y g0~1 sigma~1    5 -1552.86 3115.72 3116.00 1.086 0.2573
## fit.Dforest               D~forest g0~1 sigma~1    4 -1554.15 3116.29 3116.47 1.653 0.1938
## fit.null                      D~1 g0~1 sigma~1    3 -1555.75 3117.50 3117.61 2.860 0.1060
```

Each of the inhomogeneous models seems marginally better than the null model, but there is little to choose among them.

To visualise the entire surface we compute predicted density at each mask point. For example, we can plot

the quadratic surface like this:

```
par(mar = c(1,6,1,8))
surfaceDxy2 <- predictDsurface(fits$fit.Dxy2)
plot(surfaceDxy2, plottype = "shaded", poly = FALSE, breaks = seq(0,22,2),
     title = "Density / ha", text.cex = 1)
# Various graphical elements may be added, including contours of the Dsurface:
plot(ovtrap, add = TRUE)
plot(surfaceDxy2, plottype = "contour", poly = FALSE, breaks = seq(0,22,2), add = TRUE)
lines(leftbank)
```

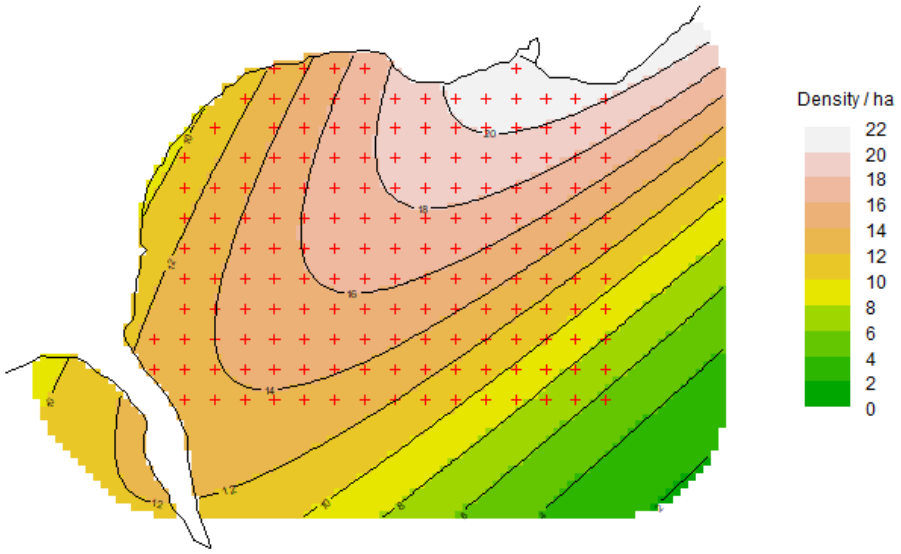


Figure 2: Quadratic surface

Following sections expand on the options for specifying and displaying density models.

Using the 'model' argument in secr.fit

The model argument of `secr.fit` is a list of formulae, one for each 'real' parameter[2] in both the state model (usually just D for density) and the observation model (typically g0 or lambda0, and sigma). A model formula defines variation in each parameter as a function of covariates (including geographic coordinates and their polynomial terms) that is linear on the 'link' scale, as in a generalized linear model.

The options differ between the state and observation models. D may vary with respect to group, session or point in space; g0, lambda0, and sigma may vary by group, session, occasion or latent class (finite mixture), but not with respect to continuous space. This was a choice made in the software design, aiming to tame the complexity that would result if g0 and sigma were allowed to vary continuously in space.

The predictors 'group' and 'session' behave for D as they do for other real parameters. They determine variation in the expected density for each group or session that is (by default) uniform across space, leading to a homogeneous Poisson model and a flat surface. No further explanation is therefore needed.

[2]Null formulae such as `D ~ 1` may be omitted, and if a single formula is used, it may be presented on its own rather than in list() form.

Link function

The default link for D is 'log'. It is equally feasible in most cases to choose 'identity' as the link (see the `secr.fit` argument 'link'), and for the null model D ~ 1 the estimate will be the same to numerical accuracy, as will estimates involving only categorical variables (e.g., session). However, with an 'identity' link the usual (asymptotic) confidence limits will be symmetrical (unless truncated at zero) rather than asymmetrical. In models with continuous predictors, including spatial trend surfaces, the link function will affect the result, although the difference may be small when the amplitude of variation on the surface is small. Otherwise, serious thought is needed regarding which model is biologically more appropriate: logarithmic or linear.

The 'identity' link may cause problems when density is very small or very large because, by default, the maximisation method assumes all parameters have similar scale (e.g., `typsize = c(1,1,1)` for default constant models). Setting `typsize` manually in a call to `secr.fit` can fix the problem and speed up fitting. For example, if density is around 0.001/ha (10 per 100 km^2) then call `secr.fit(..., typsize = c(0.001,1,1))` (`typsize` has one element for each beta parameter). See Appendix 2 for more on link functions.

You may wonder why `secr.fit` is ambivalent regarding the link function: link functions have seemed a necessary part of the machinery for capture–recapture modelling since Lebreton et al. (1992). Their key role is to keep the 'real' parameter within feasible bounds (e.g., 0-1 for probabilities). In `secr.fit` any modelled value of D that falls below zero is truncated at zero (of course this condition will not arise with a log link).

Built-in variables

`secr.fit` automatically recognises the spatial variables x, y, x2, y2 and xy if they appear in the formula for D. These refer to the x-coordinate, y-coordinate, x-coordinate^2 etc. for each mask point, and will be constructed automatically as needed.

The formula for D may also include the non-spatial variables g (group), session (categorical), and Session (continuous), defined as for modelling g0 and sigma (see secr-overview.pdf).

The built-in variables offer limited model possibilities:

Formula	Interpretation
D ~ 1	flat surface (default)
D ~ x + y	linear trend surface (planar)
D ~ x + x2	quadratic trend in east-west direction only
D ~ x + y + x2 + y2 + xy	quadratic trend surface

User-provided variables

More interesting models can be made with variables provided by the user. These are stored in a data frame as the 'covariates' attribute of a mask object. Covariates must be defined for every point on a mask.

Variables may be categorical (a factor or character value that can be coerced to a factor) or continuous (a numeric vector). The habitat variable 'habclass' constructed in the Examples section of the `skink` help is an example of a two-class categorical covariate. Remember that categorical variables entail one additional parameter for each extra level.

There are several ways to create or input mask covariates.

1. Read columns of covariates along with the x- and y-coordinates when creating a mask from a dataframe or external file (`read.mask`)
2. Read the covariates dataframe separately from an external file (`read.table`)
3. Infer covariate values by computation on in existing mask (see below).

4. Infer values for points on an existing mask from a GIS data source, such as a polygon shapefile or other spatial data source (see secr-spatialdata.pdf).

Use the function `addCovariates` for the third and fourth options.

Covariates computed from coordinates

Higher-order polynomial terms may be added as covariates if required. For example,

```
covariates(ovmask)[,"x3"] <- covariates(ovmask)$x^3
```

allows a model like D ~ x + x2 + x3.

If you have a strong prior reason to suspect a particular 'grain' to the landscape then this may be also be computed as a new, artificial covariate. This code gives a covariate representing a northwest – southeast trend:

```
covariates(ovmask)[,"NWSE"] <- ovmask$y - ovmask$x - mean(ovmask$y - ovmask$x)
```

Another trick is to compute distances to a mapped landscape feature. For example, possum density in our Orongorongo example may relate to distance from the river; this corresponds roughly to elevation, which we do not have to hand. The `distancetotrap` function of **secr** computes distances from mask cells to the nearest vertex on the riverbank, which are precise enough for our purpose.

```
covariates(ovmask)[,"DTR"] <- distancetotrap(ovmask, leftbank)
```

```
par(mar = c(1,6,1,8))
plot(ovmask, covariate = "DTR", breaks = seq(0,500,50),
     title = "Distance to river m", dots = FALSE, inset= 0.07)
```

Figure 3: Distance to river

Pre-computed resource selection functions

A resource selection function (RSF) was defined by Boyce et al. (2002) as "any model that yields values proportional to the probability of use of a resource unit". An RSF combines habitat information from multiple

sources in a single variable. Typically the function is estimated from telemetry data on marked individuals, and primarily describes individual-level behaviour (3rd-order habitat selection of Johnson (1980)).

However, the individual-level RSF is also a plausible hypothesis for 2nd-order habitat selection i.e. for modelling the relationship between habitat and population density. Then we interpret the RSF as a single variable that is believed to be proportional to the expected population density in each cell of a habitat mask.

Suppose, for example, in folder datadir we have a polygon shapefile (RSF.shp, RSF.dbf etc.) with the attribute "rsf" defined for each polygon. Given mask and capthist objects "habmask" and "myCH", this code fits a SECR model that calibrates the RSF in terms of population density:

```
rsfshape <- sf::st_read(paste0(datadir, "/RSF.shp"))
habmask <- addCovariates(habmask, rsfshape, columns = "rsf")
secr.fit (myCH, mask = habmask, model = D ~ rsf - 1)
```

- "rsf" must be known for every pixel in the habitat mask
- Usually it make sense to fit the density model through the origin (rsf = 0 implies D = 0). This is not true of habitat suitability indices in general.

This is a quite different approach to fitting multiple habitat covariates within **secr**, and one that should be considered. There are usually too few individuals in a SECR study to usefully fit models with multiple covariates of density, even given a large dataset such as our possum example. However, 3rd-order and 2nd-order habitat selection are conceptually distinct, and their relationship is an interesting research topic.

Regression splines

Regression splines are a flexible alternative to polynomials for spatial trend analysis. Regression splines are familiar as the smooth terms in 'generalized additive models' (gams) implemented (differently) in the base R package **gam** and in Simon Wood's package **mgcv**.

Some of the possible smooth terms from **mgcv** can be used in model formulae for `secr.fit` – see the help page for 'smooths' in **secr**. Smooths are specified with terms that look like calls to the functions `s` and `te`. Smoothness is determined by the number of knots which is set by the user via the argument 'k'. The number of knots cannot be determined automatically by the penalty algorithms of **mgcv**.

Here we fit a regression spline with the same number of parameters as a quadratic polynomial, a linear effect of the 'distance to river' covariate on log(D), and a nonlinear smooth.

```
base.args <- list(capthist = OVpossumCH[[1]], mask = ovmask, trace = FALSE)
args.D6 <- c(base.args, model = D ~ s(x,y, k = 6))
args.DDTR <- c(base.args, model = D ~ DTR)
args.DDTR3 <- c(base.args, model = D ~ s(DTR, k = 3))
arglist <- list(D6 = args.D6, DDTR = args.DDTR, DDTR3 = args.DDTR3)
fits2 <- par.secr.fit(arglist, ncores = 3)
```

```
## Completed in 1.385 minutes at 13:28:58 21 May 2023
```

Now add these to the AIC table and plot the 'AIC-best' model:

```
AIC(c(fits, fits2), criterion = "AIC")[,-2]
```

```
##                                          model npar   logLik     AIC    AICc  dAIC  AICwt
## fit.DDTR3          D~s(DTR, k = 3) g0~1 sigma~1    5 -1552.00 3114.01 3114.29 0.000 0.2667
## fit.Dxy2    D~x + y + x2 + y2 + xy g0~1 sigma~1    8 -1549.32 3114.64 3115.31 0.628 0.1948
## fit.DDTR                      D~DTR g0~1 sigma~1    4 -1553.36 3114.72 3114.90 0.705 0.1875
## fit.Dxy                     D~x + y g0~1 sigma~1    5 -1552.86 3115.72 3116.00 1.714 0.1132
## fit.D6           D~s(x, y, k = 6) g0~1 sigma~1    8 -1549.93 3115.86 3116.53 1.847 0.1059
## fit.Dforest               D~forest g0~1 sigma~1    4 -1554.15 3116.29 3116.47 2.281 0.0853
## fit.null                      D~1 g0~1 sigma~1    3 -1555.75 3117.50 3117.61 3.488 0.0466
```

```
tmp <- predict(fits2$fit.DDTR3, newdata = data.frame(DTR = seq(0,400,5)))
par(mar=c(5,8,2,4), pty = "s")
plot(seq(0,400,5), sapply(tmp, "[", "D","estimate"), ylim = c(0,20),
     xlab = "Distance from river (m)", ylab = "Density / ha", type = "l")
```

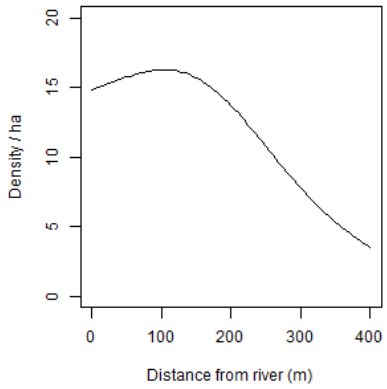
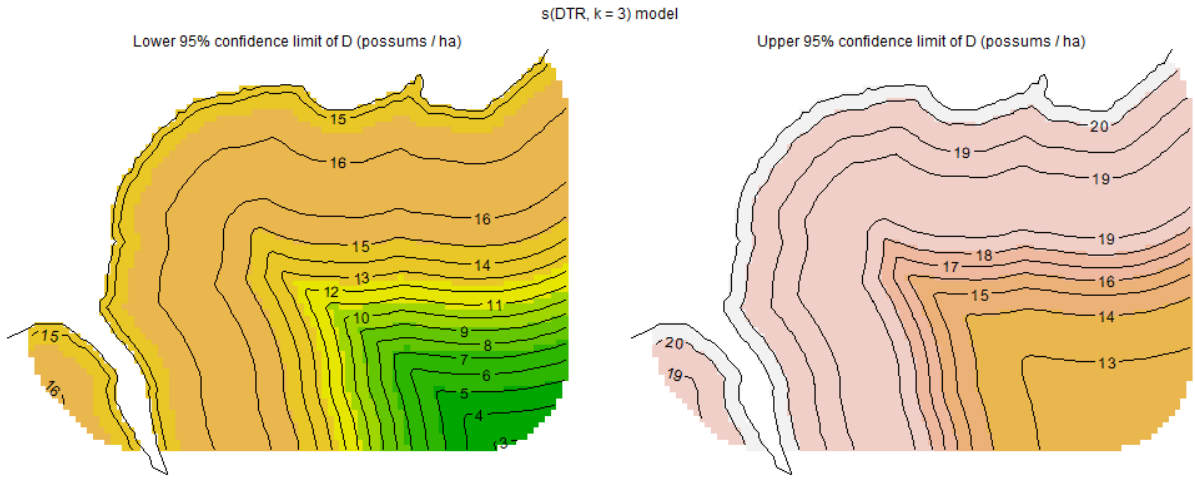


Figure 4: Predicted D from DTR3 model

Confidence intervals are computed in `predictDsurface` by back-transforming ±2SE from the link (log) scale:

```
par(mar = c(1,1,1,1), mfrow = c(1,2), xpd = FALSE)
surfaceDDTR3 <- predictDsurface(fits2$fit.DDTR3, cl.D = TRUE)
plot(surfaceDDTR3, covariate= "lcl", breaks = seq(0,22,2), legend = FALSE)
mtext(side=3,line=-1.5,"Lower 95% confidence limit of D (possums / ha)")
plot(surfaceDDTR3, plottype = "contour", breaks = seq(0,22,2), add = TRUE)
lines(leftbank)
plot(surfaceDDTR3, covariate= "ucl", breaks = seq(0,22,2), legend = FALSE)
mtext(side=3,line=-1.5,"Upper 95% confidence limit of D (possums / ha)")
plot(surfaceDDTR3, covariate= "ucl", plottype = "contour", breaks = seq(0,22,2),
     add = TRUE)
lines(leftbank)
mtext(side=3, line=-1, outer=TRUE, "s(DTR, k = 3) model")
```



Multiple predictors may be included in one 's' smooth term, implying interaction. This assumes isotropy – equality of scales on the different predictors – which is appropriate for geographic coordinates such as x and y in this example. In other cases, predictors may be measured on different scales (e.g., structural complexity of vegetation and elevation) and isotropy cannot be assumed. In these cases a tensor-product smooth (`te`) is appropriate because it is scale-invariant. For `te`, 'k' represents the order of the smooth on each axis, and we must fix the number of knots with 'fx = TRUE' to override automatic selection.

For more on the use of regression splines see the documentation for **mgcv**, the **secr** help page '?smooths', Wood (2006), and Borchers and Kidney (submitted).

Prediction and plotting

Fitting a model provides estimates of its coefficients or 'beta parameters'; use the `coef` method to extract these from an secr object. The coefficients are usually of little use in themselves, but we can use them to make predictions. In order to plot a fitted model we first predict the height of the density surface at each point on a mask. As we have seen, this is done with `predictDsurface`, which has arguments `(object, mask = NULL, se.D = FALSE, cl.D = FALSE, alpha = 0.05)`. By default, prediction is at the mask points used when fitting the model (i.e. object$mask); specify the mask argument to extrapolate the model to a different area.

The output from `predictDsurface` is a specialised mask object called a Dsurface (class "c('Dsurface', 'mask', 'data.frame')"). The covariate dataframe of a Dsurface has columns for the predicted density of each group (D.0 if there is only one). Usually when you print a mask you see only the x- and y-coordinates. The `print` method for Dsurface objects displays both the coordinates and the density values as one dataframe, as also do the `head` and `tail` methods.

Use the arguments 'se.D' and 'cl.D' to request computation of the estimated standard error and/or upper and lower confidence limits for each mask point[3]. If requested, values are saved as additional covariates of the output Dsurface (SE.0, lcl.0, and ucl.0 if there is only one group).

The plot method for a Dsurface object has arguments `(x, covariate = "D", group = NULL, plottype = "shaded", scale = 1, ...)`. `covariate` may either be a prefix (one of "D", "SE", "lcl", "ucl") or any full covariate name. 'plottype' may be one of "shaded", "dots", "persp", or "contour". A coloured legend is displayed centre-right (see ?plot.mask and ?strip.legend for options).

For details on how to specify colours, levels etc. read the help pages for `plot.mask`, `contour` and `persp` (these functions may be controlled by extra arguments to `plot.Dsurface`, using the 'dots' convention).

[3]Option available only for models specified in generalized linear model form with the 'model' argument of secr.fit, not for user-defined functions.

A plot may be enhanced by the addition of contours. This is a challenge, because the `contour` function in R requires a rectangular matrix of values, and our mask is not rectangular. We could make it so with the **secr** function `rectangularMask`, which makes a rectangular Dsurface with missing (NA) values of density at all the external points. `plot.Dsurface` recognises an irregular mask and attempts to fix this with an internal call to `rectangularMask`.

Scaling

So far we have ignored the scaling of covariates, including geographic coordinates.

`secr.fit` scales the x- and y-coordinates of mask points to mean = 0, SD = 1 before using the coordinates in a model. Remember this when you come to use the coefficients of a density model. Functions such as `predictDsurface` take care of scaling automatically. `predict.secr` uses the scaled values ('newdata' x = 0, y = 0), which provides the predicted density at the mask centroid. The mean and SD used in scaling are those saved as the 'meanSD' attribute of a mask (dataframe with columns 'x' and 'y' and rows 'mean' and 'SD').

Scaling of covariates other than x and y is up to the user. It is not usually needed.

The numerical algorithms for maximising the likelihood work best when the absolute expected values are roughly similar for all parameters on their respective 'link' scales (i.e. all beta parameters) rather than varying by orders of magnitude. The default link function for D and sigma (log) places the values of these parameters on a scale that is not wildly different to the variation in g0 or lambda0, so this is seldom an issue. In extreme cases you may want to make allowance by setting the `typsize` argument of `nlm` or the `parscale` control argument of `optim` (via the ... argument of `secr.fit`).

Scaling is not performed routinely by `secr.fit` for distance calculations. Sometimes, large numeric values in coordinates can cause loss of precision in distance calculations (there are a lot of them at each likelihood evaluation). The problem is serious in datasets that combine large coordinates with small detector spacing, such as the Lake Station `skink` dataset. Set `details = list(centred = TRUE)` to force scaling; this may become the default setting in a future version of **secr**.

Potential problems

Modelling density surfaces can be tricky. Recognise when model fitting has failed. If there is no asymptotic variance-covariance matrix, the estimates cannot be trusted. Some forensic work may be needed. If in doubt, try repeating the fit, perhaps starting from the previously fitted values (you can use `secr.fit(..., start = last.model)` where `last.model` is a previously fitted secr object) or from new arbitrary values. Problems may result when the discretization is too coarse, so try with smaller mask cells.

You can try another optimization method; `method = "Nelder-Mead"` is generally more robust than the default gradient-based method. Any method may fail to find the true maximum from a given starting point. We have no experience with simulated annealing ('SANN' in `optim`); it is reputedly effective, but slow. In the `optim` help it is stated ominously that "the 'SANN' method depends critically on the settings of the control parameters. It is not a general-purpose method".

Avoid using '[' to extract subsets from mask, capthist and other secr objects. Use the provided `subset` methods. It is generally safe to use '[[' to extract one session from a multi-session object, as in our possum example, but this is not guaranteed. With care, it is also possible to replace selected elements in situ, but note that any change in coordinates will require the attribute 'meanSD' to be recalculated (see `?getMeanSD`).

Do you really want to model density on the log scale? If not, change the link.

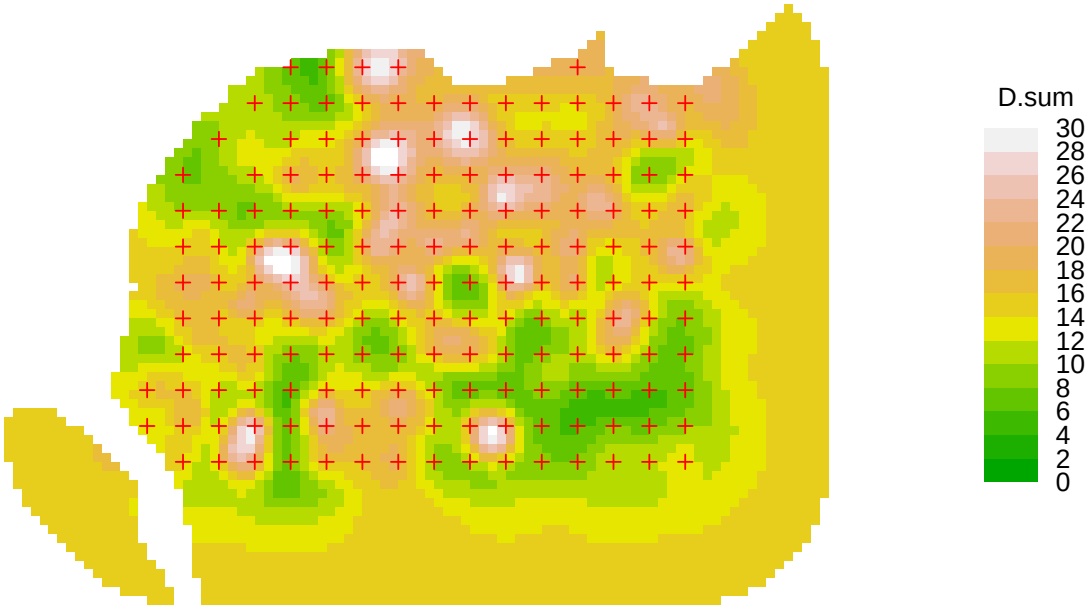
This is not a density surface[4]

The surfaces we have fitted involve inhomogeneous Poisson models for the distribution of animal home range centres. The models have parameters that determine the relationship of expected density to location or to habitat covariates.

Another type of plot is sometimes presented and described as a 'density surface' – the summed posterior distribution of estimated range centres from a Bayesian fit of a homogeneous Poisson model. A directly analogous plot may be obtained from the **secr** function `fx.total` (see also Borchers and Efford 2008 Section 4.3). The contours associated with the home range centre of each detected individual essentially represent 2-D confidence intervals for its home range centre, given the fitted observation model. Summing these gives a summed probability density surface for the centres of the observed individuals ('D.fx'), and to this we can add an equivalent scaled probability density surface for the individuals that escaped detection ('D.nc'). Both components are reported by `fx.total`, along with their sum ('D.sum') which we plot here for the flat possum model:

```
fxsurface <- fx.total(fits$fit.null)
```

```
par(mar = c(1,6,1,8))
plot(fxsurface, covariate = "D.sum", breaks = seq(0,30,2), poly = FALSE)
plot(ovtrap, add = TRUE)
```



The plot concerns only one realisation from the underlying Poisson model. It visually invites us to interpret patterns in that realisation that we have not modelled. There are serious problems with the interpretation of such plots as 'density surfaces':

- attention is focussed on the individuals that were detected; others that were present but not detected are represented by a smoothly varying base level that dominates in the outer region of the plot (contrast this figure with the previous quadratic and DTR3 models).
- the surface depends on sampling intensity, and as more data are added it will change shape systematically. Ultimately, the surface near the centre of a detector array becomes a set of spikes on a barren plain
- the 'summed confidence interval' plot is easily confused with the 2-D surface obtained by summing utilisation distributions across animals
- confidence intervals are not available for the height of the probability density surface.

[4] http://en.wikipedia.org/wiki/The_Treachery_of_Images

The plots are also prone to artefacts. In some examples we see concentric clustering of estimated centres around the trapping grids, apparently 'repelled' from the traps themselves (e.g., plot below for a null model of the Waitarere 'possumCH' dataset in **secr**). This phenomenon appears to relate to lack of model fit (unpubl. results).

```
fxsurfaceW <- fx.total(possum.model.0)
```

```
par(mar = c(1,5,1,8))
plot(fxsurfaceW, covariate = "D.sum", breaks = seq(0,5,0.5), poly = FALSE)
plot(traps(possumCH), add = TRUE)
```

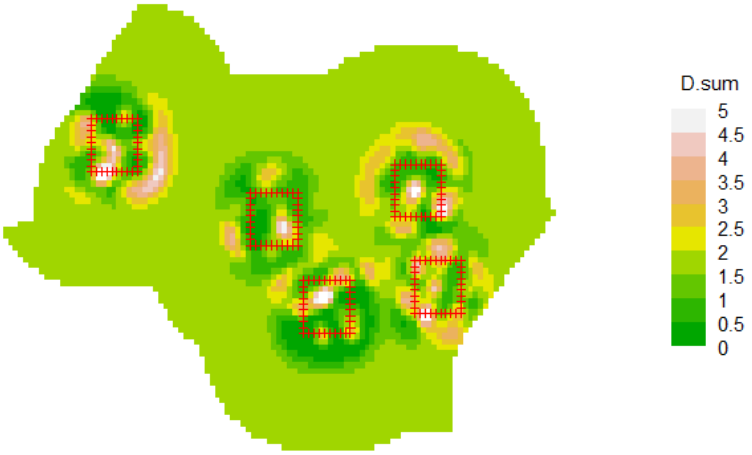


Figure 5: fxsurfaceW

Relative density

In rare cases it is useful to model the relative density of tagged animals. This is the best that can be done with data for which the tagged sample was not collected in a way that allows the initial detections to be modelled spatially. One scenario involves acoustic telemetry or other automated detection for which the only animals at risk of detection are those previously marked (cf resighting data, in which unmarked animals are detected and counted, but not identified).

The relative density model is fitted by maximising the likelihood for the density (ϕ) and detection (θ) parameters, conditional on the number of detected animals n. Other notation here follows Borchers and Efford (2008).

$$\mathrm{L}(\phi,\theta|\boldsymbol{\omega}) = \binom{n}{n_1,\ldots,n_C} \prod_{i=1}^{n} \int_{R^2} \Pr(\boldsymbol{\omega}_i|\theta,\mathbf{x})\, f(\mathbf{x}|\phi)\, d\mathbf{x}. \tag{1}$$

The factor $f(\mathbf{x}|\phi)$ describes the distribution of *tagged* animals under the model with no allowance for how they came to be tagged. The factor $\Pr(\boldsymbol{\omega}_i|\theta,\mathbf{x})$ is the probability of the observed detection history of animal i given that its activity centre was at $\mathbf{x}$.

A spatial model for relative density is fitted in **secr** by setting `details = list(relativeD = TRUE)` in the call to `secr.fit` (**secr** $\geq$ 4.6.5). For example

```
# routine fit with DTR as density covariate
fitrd0 <- secr.fit(capthist = OVpossumCH[[1]], mask = ovmask, trace = FALSE,
                   model = D ~ DTR, details = list(relativeD = FALSE))
```

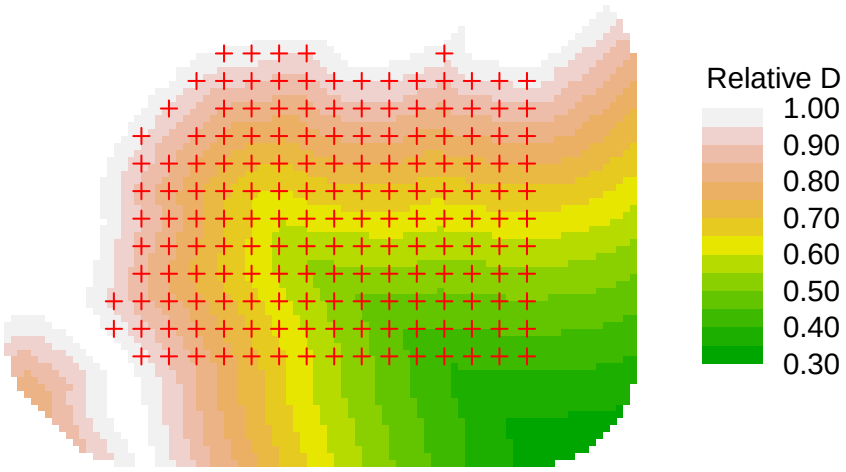
```
# relative density fit, assuming uniform probability of tagging
fitrd1 <- secr.fit(capthist = OVpossumCH[[1]], mask = ovmask, trace = FALSE,
         model = D ~ DTR, details = list(relativeD = TRUE))
coef(fitrd0)
```

```
##              beta     SE.beta         lcl          ucl
## D      2.88882711 0.119977099  2.65367632  3.123977908
## D.DTR -0.00177525 0.000811153 -0.00336508 -0.000185418
## g0    -2.22622833 0.094368595 -2.41118738 -2.041269281
## sigma  3.31910105 0.035575304  3.24937474  3.388827363
```

```
coef(fitrd1)
```

```
##             beta     SE.beta         lcl         ucl
## D.DTR -0.0027647 0.000713459 -0.00416306 -0.00136635
## g0    -2.1189469 0.090419964 -2.29616675 -1.94172700
## sigma  3.3677965 0.038319139  3.29269240  3.44290066
```

```
plot(predictDsurface(fitrd1), title = 'Relative D')
plot(traps(OVpossumCH[[1]]), add = TRUE)
```



This surface combines the population density and the probability that an animal was captured at least once, which probably varies spatially and is therefore confounded with relative population density. Densities are given relative to the intercept of the density model (assigned the arbitrary value 1.0). Absolute densities are not available because the model is fitted by maximising the likelihood conditional on n. The relative model has one fewer coefficients than the absolute density model.

A better predictor of the relative density of tagged animals is an approximation to the probability of detection, computed as a mask covariate. We invert the covariate so that the intercept corresponds to the (approximate) maximum relative density.

```
# construct a detection covariate using ballpark detection parameters
covariates(ovmask)$pd <- 1-pdot(ovmask, traps(OVpossumCH[[1]]),
    detectpar = list(g0 = 0.0974198, sigma = 27.6354961), noccasions = 5)
# fit relative density model with new predictor...
fitrd2 <- secr.fit(capthist = OVpossumCH[[1]], mask = ovmask, trace = FALSE,
         model = D ~ pd, details = list(relativeD = TRUE))
coef(fitrd2)
```

```
##           beta   SE.beta      lcl      ucl
## D.pd  -5.26727 0.9159966 -7.06259 -3.47195
## g0    -2.09150 0.0882073 -2.26438 -1.91862
```

```
## sigma  3.30745 0.0348788  3.23909  3.37581
```

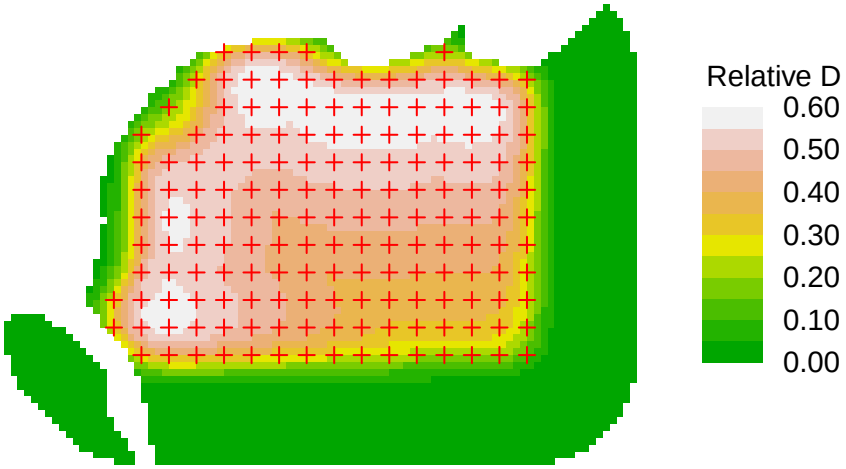
```
# ...and with both predictors
fitrd3 <- secr.fit(capthist = OVpossumCH[[1]], mask = ovmask, trace = FALSE,
         model = D ~ pd + DTR, details = list(relativeD = TRUE))
coef(fitrd3)
```

```
##              beta   SE.beta         lcl          ucl
## D.pd  -5.50099830 1.0280375 -7.51591476 -3.486081846
## D.DTR -0.00231103 0.0009217 -0.00411753 -0.000504532
## g0    -2.09228470 0.0882078 -2.26516875 -1.919400653
## sigma  3.30793316 0.0349507  3.23943110  3.376435225
```

```
# compare and plot
AIC(fitrd1, fitrd2, fitrd3)[,-2]
```

```
##                          model npar   logLik     AIC    AICc   dAICc AICcwt
## fitrd3 D~pd + DTR g0~1 sigma~1    4 -1576.27 3160.54 3160.72   0.000 0.9041
## fitrd2       D~pd g0~1 sigma~1    3 -1579.55 3165.10 3165.21   4.488 0.0959
## fitrd1      D~DTR g0~1 sigma~1    3 -1654.48 3314.96 3315.07 154.353 0.0000
```

```
plot(predictDsurface(fitrd3), title = 'Relative D')
plot(traps(OVpossumCH[[1]]), add = TRUE)
```



References

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

Borchers, D. L. and Kidney, D. J. (2014) Flexible density surface estimation using regression splines with spatially explicit capture-recapture data. In prep.

Boyce, M. S., Vernier, P. R., Nielsen, S. E. and Schmiegelow, F. K. A. (2002) Evaluating resource selection functions. *Ecological modelling* **157**, 281–300.

Efford, M. G. and Mowat, G. (2014) Compensatory heterogeneity in spatially explicit capture–recapture data. *Ecology* **95**, 1341–1348.

Johnson, D. H. (1980) The comparison of usage and availability measurements for evaluating resource preference. *Ecology* **61**, 65–71.

Lebreton, J.-D., Burnham, K. P., Clobert, J. and Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case studies. *Ecological Monographs* **62**,

67–118.

Pledger, S. (2000) Unified maximum likelihood estimates for closed capture–recapture models using mixtures. *Biometrics* **56**, 434–442.

Wood, S. N. (2006) *Generalized Additive Models: An Introduction with R.* Chapman and Hall/CRC.

Appendix 1. User-provided model functions

Some density models cannot be coded in the generalized linear model form of the model argument. To alleviate this problem, a model may be specified as an R function that is passed to `secr.fit`, specifically as the component 'userDfn' of the list argument 'details'. We document this feature here, although you may never use it.

The userDfn function must follow some rules.

- It should accept four arguments, the first a vector of parameter values or a character value (below), and the second a 'mask' object, a data frame of x and y coordinates for points at which density must be predicted.

Argument	Description
Dbeta	coefficients of density model, or one of c('name', 'parameters')
mask	habitat mask object
ngroup	number of groups
nsession	number of sessions

- When called with `Dbeta = "name"`, the function should return a character string to identify the density model in output. (This should not depend on the values of other arguments).
- When called with `Dbeta = 'parameters'`, the function should return a character vector naming each parameter. (When used this way, the call always includes the `mask` argument, so information regarding the model may be retrieved from any attributes of `mask` that have been set by the user).
- Otherwise, the function should return a numeric array with `dim = c(nmask, ngroup, nsession)` where nmask is the number of points (rows in mask). Each element in the array is the predicted density (natural scale, in animals / hectare) for each point, group and session. This is simpler than it sounds, as usually there will be a single session and single group.

The coefficients form the density part of the full vector of beta coefficients used by the likelihood maximization function (`nlm` or `optim`). Ideally, the first one should correspond to an intercept or overall density, as this is what appears in the output of `predict.secr`. If transformation of density to the 'link' scale is required then it should be hard-coded in userDfn.

Covariates are available to user-provided functions, but within the function they must be extracted 'manually' (e.g., `covariates(mask)$habclass` rather than just 'habclass'). To pass other arguments (e.g., a basis for splines), add attribute(s) to the mask.

It will usually be necessary to specify starting values for optimisation manually with the start argument of `secr.fit`.

If the parameter values in `Dbeta` are invalid the function should return an array of all zero values.

Here is a 'null' userDfn that emulates D ~ 1 with log link

```
userDfn0 <- function (Dbeta, mask, ngroup, nsession) {
    if (Dbeta[1] == "name") return ("0")
    if (Dbeta[1] == "parameters") return ("intercept")
    D <- exp(Dbeta[1])    # constant for all points
```

```
    tempD <- array(D, dim = c(nrow(mask), ngroup, nsession))
    return(tempD)
}
```

We can compare the result using userDfn0 to a fit of the same model using the 'model' argument. Note how the model description combines 'user.' and the name '0'.

```
model.0 <- secr.fit(captdata, model = D ~ 1, trace = FALSE)
userDfn.0 <- secr.fit(captdata, details = list(userDfn = userDfn0), trace = FALSE)
AIC(model.0, userDfn.0)
```

```
##                              model   detectfn npar   logLik     AIC    AICc dAICc AICcwt
## model.0         D~1 g0~1 sigma~1 halfnormal    3 -759.026 1524.05 1524.38     0    0.5
## userDfn.0 D~userD.0 g0~1 sigma~1 halfnormal    3 -759.026 1524.05 1524.38     0    0.5
```

```
predict(model.0)
```

```
##        link estimate SE.estimate       lcl       ucl
## D       log  5.47982   0.6467417  4.351636  6.900490
## g0    logit  0.27319   0.0270513  0.223477  0.329274
## sigma   log 29.36583   1.3049376 26.917572 32.036771
```

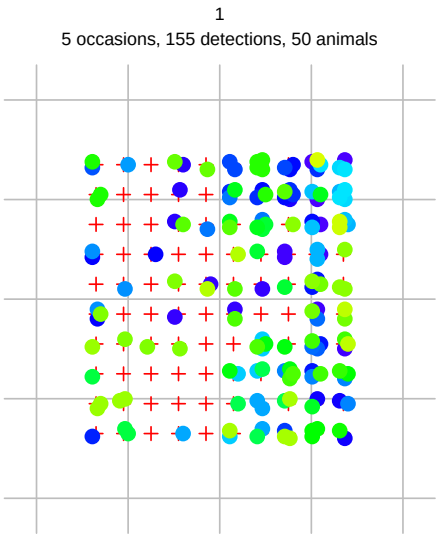
```
predict(userDfn.0)
```

```
##        link  estimate SE.estimate       lcl       ucl
## D       log  5.479811   0.6467412  4.351629  6.900480
## g0    logit  0.273191   0.0270513  0.223477  0.329274
## sigma   log 29.365826   1.3049374 26.917567 32.036765
```

Not very exciting, maybe, but reassuring!

Now let's try a more complex example. First create a test dataset with an east-west density step (this could be done more precisely with `sim.popn` + `sim.capthist`):

```
set.seed(123)
ch <- subset(captdata, centroids(captdata)[,1]>500 | runif(76)>0.75)
plot(ch)
```




```
# also make a mask and assign the x coordinate to covariate 'X'
msk <- make.mask(traps(ch), buffer = 100, type = 'trapbuffer')
covariates(msk)$X <- msk$x
```

Now define a sigmoid function of covariate X:

```
sigmoidfn <- function (Dbeta, mask, ngroup, nsession) {
    scale <- 7.5   # arbitrary 'width' of step
    if (Dbeta[1] == "name") return ("sig")
    if (Dbeta[1] == "parameters") return (c("D1", "threshold", "D2"))
    X2 <- (covariates(mask)$X - Dbeta[2]) / scale
    D <- Dbeta[1] + 1 / (1+exp(-X2)) * (Dbeta[3] - Dbeta[1])
    tempD <- array(D, dim = c(nrow(mask), ngroup, nsession))
    return(tempD)
}
```

Fit null model and sigmoid model:

```
fit.0 <- secr.fit(ch, mask = msk, link = list(D = "identity"), trace = FALSE)
```

```
## Warning in secr.fit(ch, mask = msk, link = list(D = "identity"), trace = FALSE): multi-catch
## likelihood used for single-catch traps
```

```
fit.sigmoid <- secr.fit(ch, mask = msk, details = list(userDfn = sigmoidfn),
    start=c(2.7, 500, 5.8, -1.2117, 3.4260), link = list(D = "identity"),
    trace = FALSE)
```

```
## Warning in secr.fit(ch, mask = msk, details = list(userDfn = sigmoidfn), : multi-catch
## likelihood used for single-catch traps
```

```
coef(fit.0)
```

```
##           beta   SE.beta      lcl       ucl
## D      3.61369 0.5244741  2.58574  4.641642
## g0    -1.08751 0.1617387 -1.40451 -0.770508
## sigma  3.39532 0.0556487  3.28625  3.504391
```

```
coef(fit.sigmoid)
```

```
##                  beta    SE.beta       lcl        ucl
## D.D1          1.68391  0.5138505   0.67678   2.691037
## D.threshold 514.26458 16.6528934 481.62551 546.903652
## D.D2          5.88101  1.0474658   3.82802   7.934009
## g0           -1.08605  0.1616989  -1.40297  -0.769123
## sigma         3.39315  0.0554869   3.28439   3.501898
```

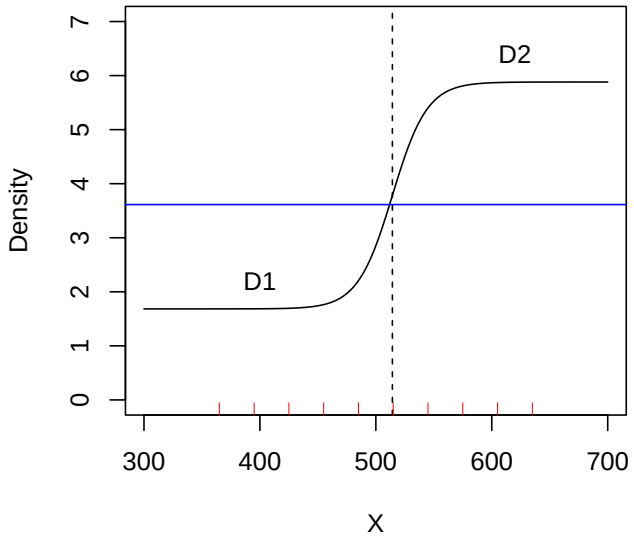
```
AIC(fit.0, fit.sigmoid)
```

```
##                               model   detectfn npar   logLik     AIC    AICc dAICc AICcwt
## fit.sigmoid D~userD.sig g0~1 sigma~1 halfnormal    5 -520.643 1051.29 1052.65   0.0      1
## fit.0               D~1 g0~1 sigma~1 halfnormal    3 -528.114 1062.23 1062.75  10.1      0
```

The sigmoid model has improved fit, but there is a lot of uncertainty in the two density levels. The average of the fitted levels D1 and D2 (3.7825) is not far from the fitted homogeneous level (3.6137).

```
beta <- coef(fit.sigmoid)[1:3,'beta']
X2 <- (300:700 - beta[2]) / 15
D <- beta[1] + 1 / (1+exp(-X2)) * (beta[3] - beta[1])
plot (300:700, D, type = 'l', xlab='X', ylab = 'Density', ylim=c(0,7))
abline(v=beta[2], lty=2)
abline(h=coef(fit.0)[1,1], lty=1, col='blue')
rug(unique(traps(ch)$x), col = 'red')
```

```
text(400, 2.2, 'D1')
text(620, 6.4, 'D2')
```



Appendix 2. More on link functions

From **secr** 4.5.0 there is a scaled identity link 'i1000' that multiplies each real parameter value by 1000. Then `secr.fit(..., link = list(D = 'i1000'))` is a fast alternative to specifying `typsize` for low absolute density.

Going further, you can even define your own *ad hoc* link function. To do this, provide the following functions in your workspace (your name 'xxx' combined with standard prefixes) and use your name to specify the link:

Name	Purpose	Example
xxx	transform to link scale	i100 <- function(x) x * 100
invxxx	transform from link scale	invi100 <- function(x) x / 100
se.invxxx	transform SE from link scale	se.invi100 <- function (beta, sebeta) sebeta / 100
se.xxx	transform SE to link scale	se.i100 <- function (real, sereal) sereal * 100

Following this example, you would call `secr.fit(..., link = list(D = 'i100'))`. To see the internal transformations for the standard link functions, type `secr:::transform`, `secr:::untransform`, `secr:::se.untransform` or `secr:::se.transform`.