Non-Euclidean distances in **secr** 2.9

Murray Efford

2014-11-19

Contents

Introduction

Spatially explicit capture–recapture (SECR) entails a distance-dependent observation model: the expected number of detections (λ) or the probability of detection (g) declines with increasing distance between a detector and the home-range centre of a focal animal. 'Distance' here, usually and by default, means the Euclidean distance $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^)}$. The observation model can be customised by replacing the Euclidean distance with one that 'warps' space in some ecologically meaningful way. There are innumerable ways to do this. One is the a non-Euclidean 'ecological distance' envisioned by Royle et al. (2013).

This document shows how to define and use non-Euclidean distances in **secr** 2.9. An appendix gives example **secr** code for the non-Euclidean SECR analysis of Sutherland et al. (2014).

Basics

Non-Euclidean distances are defined in **secr** by setting the 'userdist' component of the 'details' argument of `secr.fit`. The options are to (i) provide a static $K \times M$ matrix containing the distances between the K detectors and each of the M mask points, or (ii) to provide a function that computes the distances dynamically. We focus on the second option because it is more flexible and allows the estimation of a parameter for the distance model.

The userdist function

The userdist function takes three arguments. The first two are simply 2-column matrices with the coordinates of the detectors and animal locations (mask points) respectively. The third is a habitat mask (this may be the same as xy2). The function has this form:

```
mydistfn <- function (xy1, xy2, mask) {
  if (missing(xy1)) return(charactervector)
  ...
  distmat  ## nrow(xy1) x nrow(xy2) matrix
}
```

Computation of the distances is entirely under the control of the user – here we indicate that by '. . . '. The calculations may use cell-specific values of two 'real' parameters 'D' and 'noneuc' that can be provided by `secr.fit` as covariates of the mask. 'D' is the usual cell-specific expected density in animals per hectare. 'noneuc' is a special cell-specific 'real' parameter used only here: it means whatever the user wants it to mean.

Whether 'noneuc', 'D' or other mask covariates are needed by mydistfn is indicated by the character vector returned by mydistfn when it is called with no arguments. Thus, charactervector may be either a zero-length character vector or a vector of one or more parameter names ("noneuc", "D", c("noneuc", "D")).

'noneuc' has its own link scale (default 'log') on which it may be modelled as a linear function of any of the predictors available for density (x, y, x2, y2, xy, session, Session, g, or any mask covariate – see secr-densitysurfaces.pdf). It may also, in principle, be modelled using regression splines (Borchers and Kidney in prep.), but this is untested. When the model is fitted by `secr.fit`, the beta parameters for the 'noneuc' submodel are estimated along with all the others. To make noneuc available to userdist, ensure that it appears in the 'model' argument. Use the formula noneuc ~ 1 if noneuc is constant.

The function may compute least-cost paths via intervening mask cells using the powerful **igraph** package (Csardi and Nepusz 2006). This is most easily accessed with Jacob van Etten's package **gdistance**, which in turn uses the RasterLayer S4 object class from the package **raster**. To facilitate this we include code in **secr** to treat the 'mask' S3 class as a virtual S4 class, and provide a method for the function 'raster' to convert a mask to a RasterLayer.

If the function generates any bad distances (negative, infinite or missing) these will be replaced by 1e10, with a warning.

Examples

We use annotated examples to show how the userdist function may be used to define different models. For illustration we use the Orongorongo Valley brushtail possum dataset from February 1996 (see OVpossum in secr-manual.pdf). The data are captures of possums over 5 nights in single-catch traps at 30-m spacing. We start by extracting the data, defining a habitat mask, and fitting a null model:

```
library(secr)
library(maptools)
options (digits = 4)
setwd(system.file("extdata", package = "secr"))
ovforest <- readShapeSpatial("OVforest")          ## forest polygons
leftbank <- read.table("leftbank.txt")[21:195,]  ## for plotting only
ovposs <- OVpossumCH[[1]]                         ## February 1996
ovmask <- make.mask(traps(ovposs), buffer = 120, type = "trapbuffer",
                    poly = ovforest[1:2,], spacing = 7.5, keep.poly = FALSE)
```

```
fit0 <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE)
```

The distance functions below are not specific to a particular study: each may be applied to other datasets.

1. Scale of movement σ depends on location of home-range centre

In this simple case we use the non-Euclidean distance function to model continuous spatial variation in σ. This cannot be done directly in **secr** because sigma is treated as part of the detection model, which does not allow for continuous spatial variation in its parameters. Instead we model spatial variation in 'noneuc' as a stand-in for 'sigma'

```
fn1 <- function (xy1, xy2, mask) {
  if (missing(xy1)) return("noneuc")
  sig <- covariates(mask)$noneuc   ## sigma(x,y) at mask points
  sig <- matrix(sig, byrow = TRUE, nrow = nrow(xy1), ncol = nrow(xy2))
  euc <- edist(xy1, xy2)
  euc / sig
}
fit1 <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn1), model = noneuc ~ x + y + x2 + y2 + xy,
                 fixed = list(sigma = 1))
```

```
predict(fit1)
```

```
##          link estimate SE.estimate      lcl     ucl
## D         log  14.6821    1.091461 12.69393 16.9816
## lambda0   log   0.1085    0.009626  0.09123  0.1291
## noneuc    log  25.9275    1.302266 23.49821 28.6080
```

We can take the values of noneuc directly from the mask covariates because we know xy2 and mask are the same points. We may sometimes want to use fn1 in context where this does not hold, e.g., when simulating data.

```
fn1a <- function (xy1, xy2, mask) {
  if(missing(xy1)) return("noneuc")
  xy1 <- addCovariates(xy1, mask)
  sig <- covariates(xy1)$noneuc   ## sigma(x,y) at detectors
  sig <- matrix(sig, nrow = nrow(xy1), ncol = nrow(xy2))
  euc <- edist(xy1, xy2)
  euc / sig
}
fit1a <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn1a), model = noneuc ~ x + y + x2 + y2 + xy,
                 fixed = list(sigma = 1))
```

```
predict(fit1a)
```

```
##          link estimate SE.estimate      lcl     ucl
## D         log  14.4597    1.029559 12.57845 16.6222
## lambda0   log   0.1076    0.009525  0.09047  0.1279
## noneuc    log  26.1291    1.420082 23.49071 29.0637
```

We can verify the use of 'noneuc' in fn1 by using it to re-fit the null model:

```
fit0a <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                  details = list(userdist = fn1), model = noneuc ~ 1,
                  fixed = list(sigma = 1))
```

```
predict(fit0)
```

```
##          link estimate SE.estimate     lcl     ucl
## D         log  14.3775    1.002926 12.5423 16.4812
## lambda0   log   0.1015    0.008947  0.0854  0.1206
## sigma     log  27.3772    0.973034 25.5356 29.3517
```

```
predict(fit0a)
```

```
##          link estimate SE.estimate     lcl     ucl
## D         log  14.3775    1.002926 12.5423 16.4812
## lambda0   log   0.1015    0.008947  0.0854  0.1206
## noneuc    log  27.3772    0.973031 25.5356 29.3517
```

Here, fitting noneuc as a constant while holding sigma fixed is exactly the same as fitting sigma alone.

2. Scale of movement σ depends on locations of both home-range centre and detector

Hypothetically, detections at xy1 of an animal centred at xy2 may depend on both locations (this may also be seen as a approximation to the following case of continuous variation along the path between xy1 and xy2). To model this we need to retrieve the value of noneuc for both locations. Within fn2 we use `addCovariates` to extract the covariates of the mask (and hence noneuc) for each point in xy1 and xy2. The call to `secr.fit` is identical except that it uses fn2 instead of fn1:

```
fn2 <- function (xy1, xy2, mask) {
  if (missing(xy1)) return("noneuc")
  xy1 <- addCovariates(xy1, mask)
  xy2 <- addCovariates(xy2, mask)
  sig1 <- covariates(xy1)$noneuc   ## sigma(x,y) at detectors
  sig2 <- covariates(xy2)$noneuc   ## sigma(x,y) at mask points
  euc <- edist(xy1, xy2)
  sig <- outer (sig1, sig2, FUN = function(s1, s2) (s1 + s2)/2)
  euc / sig
}
fit2 <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn2), model = noneuc ~ x + y + x2 + y2 + xy,
                 fixed = list(sigma = 1))
```

```
predict(fit2)
```

```
##          link estimate SE.estimate      lcl     ucl
## D         log  14.5442    1.057939 12.61409 16.7697
## lambda0   log   0.1078    0.009549  0.09066  0.1282
## noneuc    log  26.0265    1.351356 23.50980 28.8126
```

Tip: the value of noneuc reported by predict.secr is the predicted value at the centroid of the mask, because the model uses standardised mask coordinates.

3. Continuously varying σ using gdistance

A more elegant but slower approach is to find the least-cost path across the network of cells between xy1 and xy2, using noneuc (i.e. sigma) as the cell-specific cost weighting (large cell-specific sigma equates with greater 'conductance', the inverse of friction or cost). For this we use functions from the package **gdistance**, which in turn uses **igraph**.

```r
fn3 <- function (xy1, xy2, mask) {
  if (missing(xy1)) return("noneuc")
  ## warp distances to be proportional to \int_along path sigma(x,y) dp
  ## where p is path distance
  if (!require(gdistance))
    stop ('install package gdistance to use this function')
  ## make raster from mask
  Sraster <- raster(mask, values = covariates(mask)$noneuc)
  ## Assume animals can traverse gaps: bridge gaps using global mean
  Sraster[is.na(Sraster[])] <- mean(Sraster[], na.rm = TRUE)
  ## TransitionLayer
  tr <- transition(Sraster, transitionFunction = mean, directions = 16)
  tr <- geoCorrection(tr, type = "c", multpl = FALSE)
  ## costDistance
  costDistance(tr, as.matrix(xy1), as.matrix(xy2))
}
fit3 <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn3), model = noneuc ~ x + y + x2 + y2 + xy,
                 fixed = list(sigma = 1))
```

```r
predict(fit3)
```

```
##         link estimate SE.estimate      lcl     ucl
## D        log  14.4047    1.050992 12.48768 16.6160
## lambda0  log   0.1076    0.009505  0.09052  0.1279
## noneuc   log  26.4373    1.358816 23.90539 29.2373
```

The **gdistance** function `costDistance` uses a TransitionLayer object that essentially describes the connections between cells in a RasterLayer. In `transition` adjacent cells are assigned a positive value for 'conductance' and all other cells a zero value. Adjacency is defined by the directions argument as 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves) and possibly other values (see ?adjacent in **gdistance**). Values < 16 can considerably distort distances even if conductance is homogeneous. `geoCorrection` is needed to allow for the greater separation ($\times\sqrt{2}$) of cell centres measured along diagonals.

In `ovmask` there are two forest blocks separated by a shingle stream bed and low scrub that is easily crossed by possums but does not count as 'habitat'. Habitat gaps are assumed in **secr** to be traversible. The opposite is assumed by **gdistance**. To coerce **gdistance** to behave like **secr** we here temporarily fill in the gaps.

The argument 'transitionFunction' determines how the conductance values of adjacent cells are combined to weight travel between them. Here we simply average them, but any other single-valued function of 2 inputs can be used.

Integrating along the path (fn3) takes about 3.6 times as long as the approximation (fn2) and gives quite similar results.

4. Density-dependent σ

A more interesting variation makes sigma a function of the cell-specific density, which may vary independently across space. Specifically, $\sigma(x, y) = k/\sqrt{D(x, y)}$, where k is the fitted parameter (noneuc).

```
fn4 <- function (xy1, xy2, mask) {
  if(missing(xy1)) return(c("D", "noneuc"))
  if (!require(gdistance))
    stop ('install package gdistance to use this function')
  ## make raster from mask
  D <- covariates(mask)$D
  k <- covariates(mask)$noneuc
  Sraster <- raster(mask, values = k / D^0.5)
  ## Assume animals can traverse gaps: bridge gaps using global mean
  Sraster[is.na(Sraster[])] <- mean(Sraster[], na.rm = TRUE)
  ## TransitionLayer
  tr <- transition(Sraster, transitionFunction = mean, directions = 16)
  tr <- geoCorrection(tr, type = "c", multpl = FALSE)
  ## costDistance
  costDistance(tr, as.matrix(xy1), as.matrix(xy2))
}
fit4 <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn4), fixed = list(sigma = 1),
                 model = list(noneuc ~ 1, D ~ x + y + x2 + y2 + xy))
```

```
predict(fit4)
```

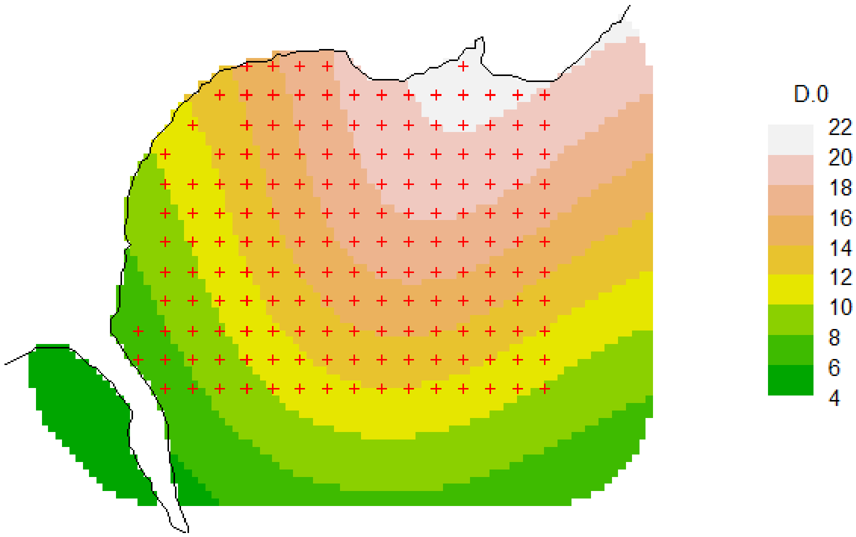
```
##          link estimate SE.estimate      lcl      ucl
## D         log  15.4579    1.626193 12.58486  18.9868
## lambda0   log   0.1067    0.009408  0.08978   0.1268
## noneuc    log 103.2258    5.031658 93.82566 113.5677
```

```
## or using regression splines with same df
fit4a <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn4), fixed = list(sigma = 1),
                 model = list(noneuc~1, D ~ s(x,y, k = 6)))
```

```
predict(fit4a)
```

```
##          link estimate SE.estimate      lcl      ucl
## D         log  15.6903    1.786561 12.56093  19.5993
## lambda0   log   0.1068    0.009412  0.08985   0.1269
## noneuc    log 103.0735    5.015270 93.70318 113.3808
```

```
plot(predictDsurface(fit4a))
plot(traps(ovposs), add=T)
lines(leftbank)
```



5. Habitat model for connectivity

Yet another possibility, in the spirit of Royle et al. (2013), is to model conductance as a function of habitat covariates. As usual in **secr** these are stored as one or more mask covariates. It is easy to add a covariate for forest type (*Nothofagus*-dominant 'beech' vs 'nonbeech') to our mask:

```
ovmask <- addCovariates(ovmask, ovforest[1:2,])
fit5 <- secr.fit(ovposs, mask = ovmask, detectfn = "HHN", trace = FALSE,
                 details = list(userdist = fn2), model = list(D ~ forest, noneuc ~ forest),
                 fixed = list(sigma = 1))
```

```
predict(fit5, newdata = data.frame(forest=c('beech', 'nonbeech')))
```

```
## $`forest = beech`
##          link estimate SE.estimate      lcl     ucl
## D         log   9.4275    2.653192  5.48772 16.1957
## lambda0   log   0.1011    0.008933  0.08506  0.1202
## noneuc    log  29.5556    3.405277 23.59890 37.0160
##
## $`forest = nonbeech`
##          link estimate SE.estimate      lcl     ucl
## D         log  15.6702    1.267629 13.37613 18.3578
## lambda0   log   0.1011    0.008933  0.08506  0.1202
## noneuc    log  27.2356    1.032005 25.28684 29.3345
```

Note that we have re-used the userdist function fn2, and allowed both density and noneuc (sigma) to vary by forest type. Strictly, we should have identified "forest" as a required covariate in the (re)definition of fn2, but this is obviously not critical.

A full analysis should also consider models with variation in lambda0. There is no simple way in **secr** to model continuous spatial variation in lambda0 as a function of home-range location (cf sigma in Example 1 above). However, variation in lambda0 at the point of detection may be modelled with detector-level covariates(secr-overview.pdf).

And the winner is...

Now that we have a bunch of fitted models, let's see which does the best:

```
AIC(fit0, fit0a, fit1, fit1a, fit2, fit3, fit4, fit4a, fit5, criterion = 'AIC')[, -c(2,4,6)]
```

```
##                                            model npar  AIC   dAIC  AICwt
## fit4a       D~s(x, y, k = 6) lambda0~1 noneuc~1    8 3098  0.000 0.4259
## fit4  D~x + y + x2 + y2 + xy lambda0~1 noneuc~1    8 3099  0.144 0.3963
## fit1  D~1 lambda0~1 noneuc~x + y + x2 + y2 + xy    8 3102  3.477 0.0749
## fit3  D~1 lambda0~1 noneuc~x + y + x2 + y2 + xy    8 3102  4.071 0.0556
## fit2  D~1 lambda0~1 noneuc~x + y + x2 + y2 + xy    8 3104  5.183 0.0319
## fit1a D~1 lambda0~1 noneuc~x + y + x2 + y2 + xy    8 3105  6.633 0.0155
## fit0                     D~1 lambda0~1 sigma~1    3 3118 19.752 0.0000
## fit0a                   D~1 lambda0~1 noneuc~1    3 3118 19.752 0.0000
## fit5           D~forest lambda0~1 noneuc~forest    5 3118 20.059 0.0000
```

...the model with a quadratic or spline trend in density and density-dependent sigma.

Notes

The 'real' parameter for spatial scale (σ) is lurking in the background as part of the detection model. User-defined non-Euclidean distances are used in the detection function just like ordinary Euclidean distances. This means in practice that they are (almost) always divided by (σ). Formally: the distance d_{ij} between an animal i and a detector j appears in all commonly used detection functions as the ratio $r_{ij} = d_{ij}/\sigma$ (e.g., halfnormal $\lambda = \lambda_0 \exp(-0.5 r_{ij})$ and exponential $\lambda = \lambda_0 \exp(-r_{ij})$).

What if I want non-Euclidean distances, but do not want to estimate noneuc? This is a perfectly reasonable request if sigma is constant across space and the distance computation is determined entirely by the habitat geometry, with no need for an additional parameter. If 'noneuc' is not included in the character vector returned by your userdist function when it is called with no arguments then noneuc is not modelled at all. (This is the default in **secrlinear**).

The initial value of 'noneuc' can be a problem. From **secr** 2.9.1 the argument 'start' of `secr.fit` may be a named and possibly incomplete list of real parameter values, so a call such as this is valid:

```
secr.fit (captdata, model = noneuc~1, details = list(userdist=fn2), start = list(noneuc = 25),
          fixed = list(sigma = 1))
```

Barriers to movement may be modelled with **gdistance**, at least in principle (I haven't tried).

We have ignored the parameter λ_0. This is almost certainly a mistake, as large variation in σ without compensatory or normalising variation in λ_0 is biologically implausible and can lead to improbable results (Efford and Mowat 2014, Efford 2014).

It is intended that non-Euclidean distances should work with all relevant functions in **secr**. However, not all possible combinations have been tested, and not all make sense. Please report any problems.

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.

Csardi, G. and Nepusz, T. (2006) The igraph software package for complex network research. *InterJournal* **1695**. http://igraph.org.

Efford, M. G. (2014) Bias from heterogeneous usage of space in spatially explicit capture–recapture analyses. *Methods in Ecology and Evolution* **5**, 599–602.

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

Royle, J. A., Chandler, R. B., Gazenski, K. D. and Graves, T. A. (2013) Spatial capture–recapture models for jointly estimating population density and landscape connectivity. *Ecology* **94** 287–294.

Sutherland, C., Fuller, A. K. and Royle, J. A. (2014) Modelling non-Euclidean movement and landscape connectivity in highly structured ecological networks. *Methods in Ecology and Evolution* (in press).

van Etten, J. (2014) gdistance: distances and routes on geographical grids. R package version 1.1-5. http://CRAN.R-project.org/package=gdistance

Appendix. Implementation in secr of Sutherland et al. (2014) non-Euclidean simulation.

Sutherland et al. (2014) simulated SECR data from a population of animals whose movement was channeled to varying extents along a dendritic network (river system). Their model treated the habitat as 2-dimensional and shrank distances for pixels close to water and expanded them for pixels further away. Chris has kindly provided data for the network map and detector layout which we use here to emulate their simulations in **secr**. We assume an existing SpatialLinesDataFrame `sample.water` for the network, and a matrix of x-y coordinates for detector locations `gridTrapsXY`. `rivers` is a version of `sample.water` clipped to the habitat mask and used only for plotting.

```
## use package secrlinear to create a discretised version of the network,
## as a handy way to get distance to water (secrlinear should be published
## by the time you read this - it is not used later in the analysis)
## loading this package also loads secr
library(secrlinear)
library(gdistance)
```

```
swlinearmask <- read.linearmask(data = sample.water, spacing = 100)
```

```
## generate secr traps object from detector locations
tr <- data.frame(gridTrapsXY*1000)  ## convert to metres
names(tr) <- c('x','y')
tr <- read.traps(data=tr, detector = 'count')

## generate 2-D habitat mask
sw2Dmask <- make.mask(tr, buffer = 3950, spacing = 100)
d2w <- distancetotrap(sw2Dmask, swlinearmask)
covariates(sw2Dmask) <- data.frame(d2w = d2w/1000)   ## km to water
```

```
## plot distance to water
par(mar = c(1,6,1,6))
plot(sw2Dmask, covariate = 'd2w', dots = FALSE)
plot(tr, add = TRUE)
plot(rivers, add = TRUE, col = 'blue')
```

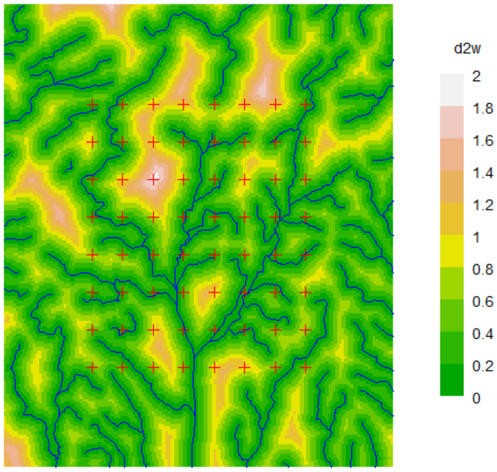


Fig. A1. Shaded plot of distance to water (d2w in km) with detector sites (red crosses) and rivers superimposed. Detector spacing 1.5 km N-S.

The distance function requires a value of the friction parameter 'noneuc' for each mask pixel. Distances are approximated using **gdistance** functions as before, except that we interpret the distance-to-water scale as 'friction' and invert that for **gdistance**.

```
userdfn1 <- function (xy1, xy2, mask) {
    if (missing(xy1)) return('noneuc')
    require(gdistance)
    Sraster <- raster(mask, values = covariates(mask)$noneuc)
    ## conductance is inverse of friction
    trans <- transition(Sraster, transitionFunction = function(x) 1/mean(x),
                     directions = 16)
    trans <- geoCorrection(trans)
    costDistance(trans, as.matrix(xy1), as.matrix(xy2))
}
```

The Royle et al. (2013) and Sutherland et al. (2014) models use an (α_0, α_1) parameterisation instead of (λ_0, σ). Their α_2 translates directly to a coefficient in the **secr** model, as we'll see. We consider just one realisation of one scenario (the package **secrdesign** manages replicated simulations of multiple scenarios).

```
## parameter values from Sutherland et al. 2014
alpha0 <- -1    ## implies lambda0 = invlogit(-1) = 0.2689414
sigma <- 1400
alpha1 <- 1 / (2 * sigma^2)
alpha2 <- 5     ## just one scenario from the range 0..10
K  <- 10        ## sampling over 10 occasions, collapsed to 1 occasion
```

Now we are ready to build a simulated dataset.

```r
## simulate fixed population of 200 animals in masked area
pop <- sim.popn (D = 200/nrow(sw2Dmask), core = tr, buffer = 3950, Ndist = 'fixed')
## in order to simulate non-Euclidean detection we attach a mask with
## the pixel-specific friction to the simulated popn object
covariates(sw2Dmask)$noneuc <- exp(alpha2 * covariates(sw2Dmask)$d2w)
attr(pop, 'mask') <- sw2Dmask
## now simulate detections, specifying our non-Euclidean distance function
CH <- sim.capthist(tr, pop = pop, userdist = userdfn1, noccasions = 1, binomN = K,
          detectpar = list(lambda0 = invlogit(alpha0), sigma = sigma), detectfn = 'HHN')
```

```r
summary(CH)
```

```
## Object class      capthist
## Detector type     count
## Detector number   64
## Average spacing   1386 m
## x-range           1698699 1708399 m
## y-range           2387891 2398391 m
## Counts by occasion
##                     1 Total
## n                  41    41
## u                  41    41
## f                  41    41
## M(t+1)             41    41
## losses              0     0
## detections        114   114
## detectors visited  35    35
## detectors used     64    64
```

Model fitting is simple, but slow (38 minutes on an aging PC). This is partly because the mask is large (32384 pixels) in order to maintain resolution in relation to the stream network. The default starting value for noneuc is not suitable and is overridden.

```r
fitne1 <- secr.fit (CH, mask = sw2Dmask, detectfn = 'HHN', binomN = 10,
        model = noneuc ~ d2w -1, details = list(userdist = userdfn1),
        start = list(noneuc = 1))
```

```r
coef(fitne1)
```

```
##               beta SE.beta    lcl     ucl
## D           -5.108 0.17620 -5.454 -4.7630
## lambda0     -1.291 0.16925 -1.623 -0.9593
## sigma        7.312 0.09644  7.123  7.5011
## noneuc.d2w   5.641 0.67081  4.327  6.9563
```

```r
predict(fitne1)
```

```
##          link  estimate SE.estimate       lcl       ucl
## D         log 6.046e-03   1.074e-03 4.280e-03 8.540e-03
## lambda0   log 2.750e-01   4.688e-02 1.974e-01 3.832e-01
## sigma     log 1.498e+03   1.448e+02 1.240e+03 1.810e+03
## noneuc    log 5.976e+02   5.285e+02 1.347e+02 2.651e+03
```

```r
region.N(fitne1)
```

```
##     estimate SE.estimate   lcl   ucl  n
## E.N    195.8       34.77 138.6 276.6 41
## R.N    195.8       31.83 144.9 271.7 41
```

The coefficient noneuc.d2w corresponds to alpha2. Estimates of predicted ('real') parameters D and lambda0, and the coefficient noneuc.d2w, and are comfortably close to the true values, and all true values are covered by the 95% CI.

We fit the 'noneuc' (friction) parameter through the origin (zero intercept; -1 in formula). The predicted value of 'noneuc' relates to the covariate value for the first pixel in the mask (d2w = 1.133 km), but in this zero-intercept model the meaning of 'noneuc' itself is obscure. In effect, the parameter alpha1 (or sigma) serves as the intercept; the same model may be fitted by fixing sigma (`fixed = list(sigma = 1)`) and estimating an intercept for noneuc (`model = noneuc ~ d2w`). In this case, 'noneuc' may be interpreted as the site-specific sigma (see also examples in the main text).

It is interesting to plot the predicted detection probability under the simulated model. For plotting we add the pdot value as an extra covariate of the mask. Note that pdot here uses the 'noneuc' value previously added as a covariate to `sw2Dmask`.

```r
covariates(sw2Dmask)$predicted.pdot <- pdot(sw2Dmask, tr, noccasions = 1, binomN = 10,
        detectfn = 'HHN', detectpar = list(lambda0 = invlogit(-1), sigma = sigma),
        userdist = userdfn1)
```

```r
par(mar = c(1,6,1,6))
plot(sw2Dmask, covariate = 'predicted.pdot', dots = FALSE)
plot(tr, add = TRUE)
plot(rivers, add = TRUE, col = 'blue')
```

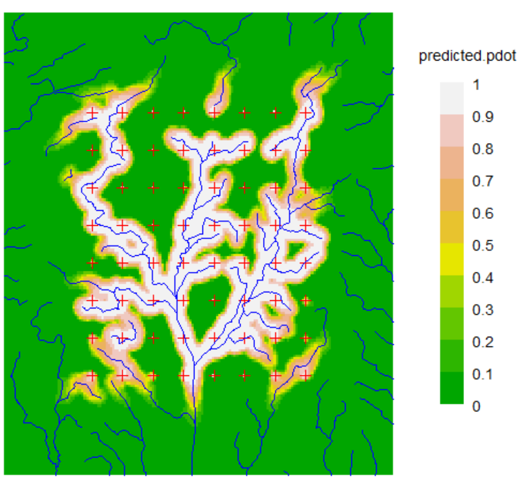


Fig. A2. Shaded plot of $p.(x, y)$ (probability animal is detected at least once). Detector sites and rivers as in Fig. A1. Animals living within the detector array and away from a river (about half the population within the array) stand very little chance of being detected because the model confines them to a small home range and λ_0 is constant.