oSCR vignettes

Chris Sutherland, Dan Linden & Gates Dupont

2020-07-22

Contents

Using this book **5**

0.1 Why should you read this book? . . . 5

0.2 Why should you use `oSCR` . . . 5

0.3 Getting set up . . . 5

1 Integrated RSF-SCR models in `oSCR` **7**

1.1 Simulating capture & telemetry data . . . 8

1.2 Model fitting in `oSCR` . . . 14

1.3 Application to NY black bears . . . 18

1.4 Summary . . . 21

2 Design criteria **23**

3 Preparing design objects **25**

3.1 Background . . . 25

3.2 Creating `statespace` from a polygon . . . 26

3.3 Creating `alltraps` from a polygon . . . 29

3.4 Inspecting the objects . . . 34

4 Generating designs **37**

Using this book

0.1 Why should you read this book?

The goal of this bookdown is to provide a complete overview of the theory and methodology of study design for spatial capture-recapture, and to empower our users to confidently generate optimized sampling design for their own use.

0.2 Why should you use `oSCR`

The main function in `oSCR` does likelihood analysis of several classes of spatial capture-recapture (SCR) models. There are also a suite of helper functions for formatting and processing data objects. Here are a few of the things that motivated our development of the package:

1. 100% native `R` code, making it (reasonably) accessible to people who know `R` and presumably extensible by ordinary `R` programmers.
2. Because it's written in `R`, you can look at the code to figure out exactly what's going on.
3. It's a bit slower compared to `secr`, but we think it's quite robust to massive-sized problems.
4. The data structure is relatively simple, just as ordinary `R` lists (for the most part).
5. The models accommodate least-cost path models and models that include telemetry data and resource selection functions.
6. `oSCR` forces you to define the state-space of the point process which we think is important to understanding an analysis.

0.3 Getting set up

So, using this book of course requires that the `oSCR` package is loaded:

```r
#remotes::install_github("jaroyle/oSCR")
library(oSCR)
```

But you will also need a few others:

```r
library(ggplot2)
library(raster)
library(sf)
library(viridis)
```

If you have any issues or questions, we have a very responsive, and friendly user group.

Chapter 1

Integrated RSF-SCR models in `oSCR`

Dan Linden

Statistical power in a spatial capture-recapture (SCR) analysis is dictated by the collection of captures obtained, particularly the number individuals captured (n) and the number of spatial locations at which each individual was captured. This latter quantity (i.e., spatial recaptures) requires a balance in trap design and configuration to ensure that a representative sample of individuals is observed in multiple locations to provide information on space use and movement.

Collecting enough spatial recaptures to allow for relevant inferences from an SCR model can be a challenge. Applications of SCR involving rare and/or wide-ranging species often result in sparse encounters and problems with fitting even simple models. Complex space use requires a large and diverse sample of spatial recaptures across the landscape that can rarely be achieved with most sampling designs. In either case, the integration of telemetry data from a subset of individuals in the population of interest may allow for better estimation of the relevant space use and movement parameters.

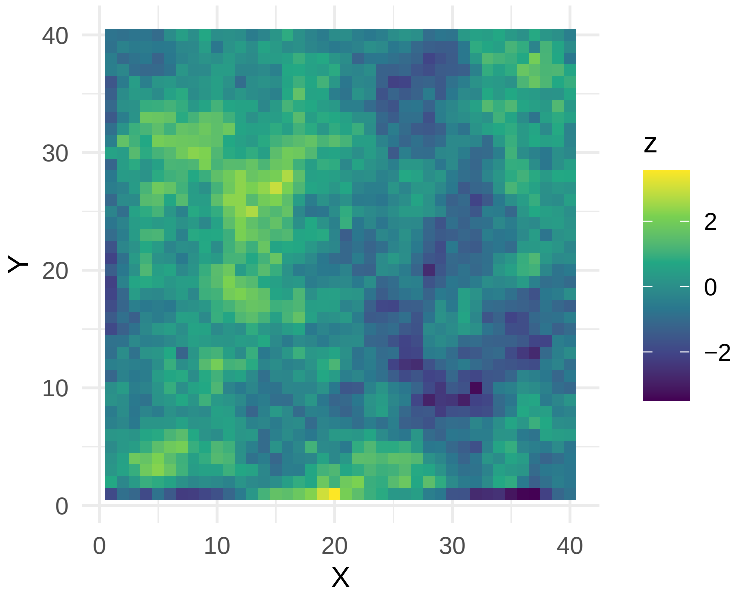
Here, we illustrate the approach described by Royle et al. (2013) as implemented in `oSCR`. We simulate a small data set and fit some SCR models that vary with regards to telemetry integration. Importantly, we distinguish the inferences made by models that assume independent or dependent samples (i.e., collared animals being captured) as described in Linden et al. (2018). We then fit a model to the New York bear data from Royle et al. (2013).

1.1 Simulating capture & telemetry data

We mostly replicate the supplementary R script from Royle et al. (2013) with a few tweaks to simulate some encounter data from a trap array and a collection of telemetry fixes from 4 collared individuals. Aside from some small details, the big change is that the collared individuals were also captured during the trapping survey.

First, we simulate a state-space with a spatially-correlated landscape covariate:

```
# drink the tidyverse Kool-aid for superior data visualization
library(tidyverse)

## the following block of code makes up a covariate as a spatially correlated noise fi
## with an exponential spatial correlation function
set.seed(1234)
gr <- expand.grid(X = 1:40, Y = 1:40)
Dmat <- as.matrix(dist(gr))
V <- exp(-Dmat / 5)
# below change to matrix multiplication
z <- as.vector(crossprod(t(chol(V)), rnorm(1600)))

# create the state-space dataframe and plot it
ssDF <- data.frame(gr,z=z)
ss.plot <- ggplot(ssDF,aes(x=X,y=Y)) + coord_equal() + theme_minimal()
ss.plot + geom_tile(aes(fill=z)) + scale_fill_viridis_c()
```



We set some values for sample sizes and model parameters:

```r
alpha0 <- -1.5   # log(encounter rate)
sigma  <- 2      # movement scale
alpha2 <- 1      # effect of covariate on resource selection
Ntel   <- 4      # number of individuals with telemetry devices
Nfixes <- 40     # number of telemetry fixes per individual
N      <- 50     # population size
```

We set up a 7×7 trap array and identify the raster cells to which they correspond for obtaining the spatial covariate value:

```r
X <- expand.grid(X=seq(5,35,5),Y=seq(5,35,5))
ntraps <- nrow(X)
raster.point <- rep(NA, nrow(X))
for (j in 1:nrow(X)) {
  raster.point[j] <- (1:1600)[(X[j, 1] == gr[, 1]) &
                              (X[j, 2] == gr[, 2])]}
```

Now we simulate the capture data. We set another seed here to make our example more compelling – a larger simulation would be needed to truly illustrate the superior performance of the integrated likelihood. The activity centers (**s**) of all

N individuals in the population are randomly distributed across the landscape and the pixel id and coordinates are identified for each. The distance matrix between individual activity centers and trap locations is calculated (with help from `e2dist`) and used to determine the encounter rate for each individual i at trap j. Finally, the hazard rate is converted to a probability and used to simulate binary capture data (y_{ij}).

```r
set.seed(9933)
# Simulate activity centers of all N individuals in the population
Sid <- sample(1:1600,N,replace=TRUE)
# and coordinates
S <- gr[Sid,]

# Hazard model is used. This seems the most sensible.
D <- oSCR::e2dist(S,X) ## N by ntraps distances
Zmat <- matrix(z[raster.point],nrow=N,ncol=ntraps,byrow=TRUE) # note make dims the sam
loglam <- alpha0 -(1/(2*sigma*sigma))*D*D + alpha2*Zmat
p <- 1-exp(-exp(loglam))
# Now simulate SCR data
K <- 3
y <-array(NA,dim=c(N,ntraps))
for(i in 1:N){ y[i,] <- rbinom(ntraps,K,p[i,]) }

# Subset data to captured individuals
cap <- apply(y,1,sum)>0
y <- y[cap,]
```

We now move to the telemetry data, where 4 individuals are chosen to be collared. The individuals need to have “interior” activity centers so that their fix locations can be observed fully within the state space. In a real application, the state space would simply need to encompass all traps and fixes with an appropriate buffer.

So we identify some possible activity centers for telemetry individuals and, critically, they have to be individuals that were captured. The `cap.tel` vector identifies the rows in the capture history for N_{tel} individuals, which is needed by `oSCR` for fitting the dependent integrated models.

```r
poss.tel<- S[,1]>5 & S[,1]<35 & S[,2]>5 & S[,2]<35

# Need to account for capture so that dependent models can be explored
tel.guys.id <- sort(sample(which(cap & poss.tel),Ntel))
tel.guys <- Sid[tel.guys.id]                #which s for each Ntel
cap.tel <- match(tel.guys.id,which(cap))    #which row in capture history for each Ntel
sid <- tel.guys
stel <- gr[sid,]
```

Instead of directly simulating the fix locations (XY coordinates) of the telemetry data, Royle et al. (2013) simulated the pixel frequencies. These frequencies are simply tallies of fixes within each pixel (x in $1, 2, ..., nG$) of the state space, with pixel-level probabilities, $\pi(\mathbf{x}|\mathbf{s})$, determined by a multinomial model of resource selection:

$$\pi(\mathbf{x}|\mathbf{s}) = \frac{\lambda(\mathbf{x}|\mathbf{s})}{\sum_x \lambda(\mathbf{x}|\mathbf{s})}$$

where,

$$\log(\lambda(\mathbf{x}|\mathbf{s})) = \alpha_0 - \alpha_1 d(\mathbf{x}, \mathbf{s})^2 + \alpha_2 \mathrm{z}(\mathbf{x})$$

The primary argument by Royle et al. (2013) is that this model of $\lambda(\mathbf{x}|\mathbf{s})$ can be assumed equivalent between the capture and telemetry data (aside from the intercept, α_0), allowing for a joint estimation of α_1 and α_2 from the likelihoods of each data set. Note that $\alpha_1 = 1/2\sigma^2$.

So, we simulate fixes according to our specified model parameters and then double-check that the averages of the simulated fix locations match the selected activity centers.

```
# Make a matrix to store RSF data (telemetry fixes)
nfix<-matrix(NA,nrow=Ntel,ncol=1600)

# for each telemetered guy simulate a number of fixes.
# note that nfix = 0 for most of the landscape pixels
lammat<-matrix(NA,nrow=Ntel,ncol=1600)
for(i in 1:Ntel){
  d<- Dmat[sid[i],]
  lam<- exp(1 - (1/(2*sigma*sigma))*d*d + alpha2* z)
  pi_i <- lam/sum(lam)
  nfix[i,]<-rmultinom(1,Nfixes,pi_i)
  lammat[i,]<-lam
}
# Average fix location (matches activity centers)
(sbar <- (nfix%*%as.matrix(gr))/as.vector(nfix%*%rep(1,nrow(gr))))
```

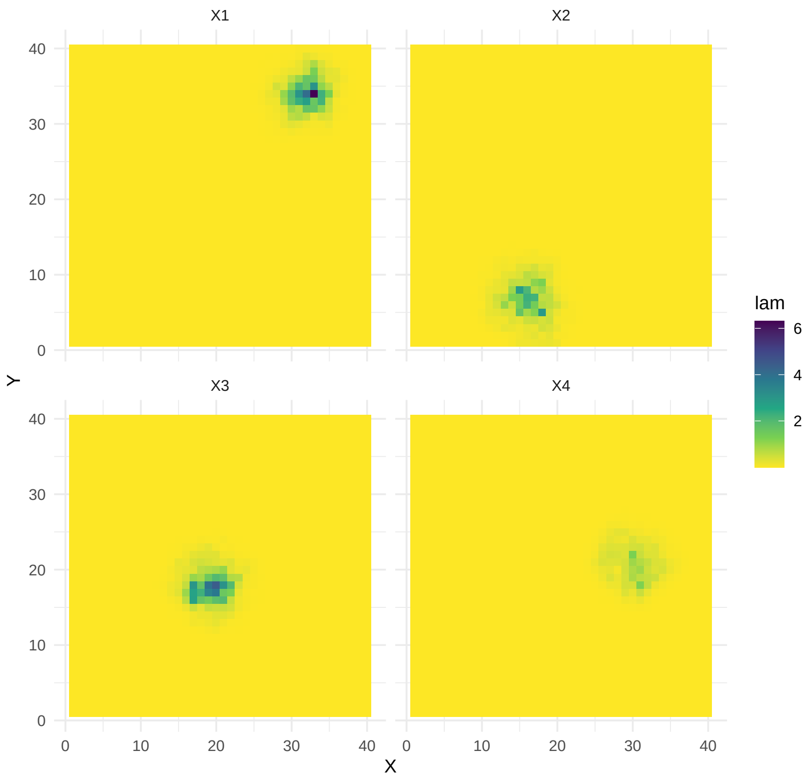
```
##           X      Y
## [1,] 31.850 33.825
## [2,] 16.550  6.325
## [3,] 19.550 17.625
## [4,] 30.475 20.900
```

```
stel
```

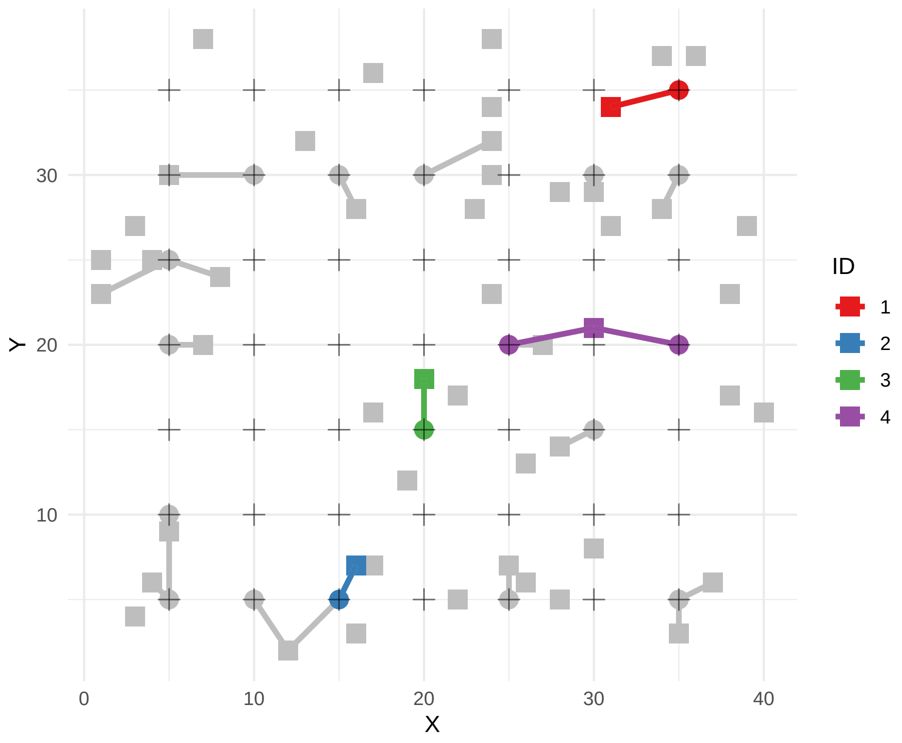
```
##       X  Y
## 1351 31 34
## 256  16  7
## 700  20 18
## 830  30 21
```

Finally, we can visualize the predicted space usage patterns and then plot the captures of the N_{tel} individuals.

```
# expected encounter rates (lambda) for collared guys
lamDF <- pivot_longer(
  data.frame(gr,t(lammat)),-c(1:2),names_to="ind",values_to="lam")
# plot lambdas by collared individual
ss.plot + geom_tile(data=lamDF,aes(fill=lam)) +
  scale_fill_viridis_c(direction = -1) + facet_wrap(~ind)
```



```r
# collection of capture locations and s for each n
caps.X <- data.frame(which(apply(y,c(1,2),sum)>0,arr.ind=T))
caps.X[,c("X","Y")] <- X[caps.X$col,]; caps.X <- caps.X[order(caps.X$row),]
caps.X <- data.frame(caps.X,s=S[cap,][caps.X$row,])
caps.X$tel <- match(caps.X$row,cap.tel)

# plot the capture data, emphasis on collared guys
ss.plot +
  # activity centers of all N, then Ntel
  geom_point(data=S,pch=15,col="gray",size=4) +
  geom_point(data=stel,aes(col=factor(1:4)),size=4,pch=15) +
  # trap of captures for n, then Ntel
  geom_point(data=caps.X,col="gray",size=4,pch=16) +
  geom_segment(data=caps.X,aes(x=X,y=Y,xend=s.X,yend=s.Y),col="gray",lwd=1.25) +
  geom_point(data=caps.X,aes(color=factor(tel)),size=4,pch=16) +
  geom_segment(data=caps.X,aes(x=X,y=Y,xend=s.X,yend=s.Y,col=factor(tel)),lwd=1.25) +
  # trap locations
  geom_point(data=X,pch=3,size=3,alpha=0.55) +
  scale_color_brewer(palette="Set1") + labs(color="ID")
```



The solid squares represent activity centers, crosses are trap locations, and circles are captures with segments indicating the individual. The 4 collared individuals are uniquely identified.

1.2 Model fitting in oSCR

We now prepare the data for fitting models in oSCR. Since we already have an $i \times j$ matrix of encounters by individuals at traps, we do not need to use the `data2oscr` function to format the data. But, we *do* need the binary encounters to be distributed across occasions to use the complimentary log-log encounter model in oSCR (the likelihood code in Royle et al. (2013) was less general). In the absence of occasion-specific effects, we can randomly distribute each encounter to one of the $k = 1, 2, 3$ occasions.

```
y.arr <- array(0,dim=c(dim(y)[1],ntraps,K))
for(i in 1:nrow(y)){
  for(j in 1:ntraps){
    which.K <- sample(1:K,y[i,j])
    y.arr[i,j,which.K] <- 1
}}
```

We load the oSCR package and define the RSF dataframe, which can simply match the state-space dataframe (or potentially be defined at a finer resolution).

```
# Set up the rsfDF (matches ssDF)
rsfDF <- ssDF
```

The telemetry data are then packaged into a list that only requires one object, fixfreq, which contains the $N_{tel} \times nG$ matrix of fix frequencies. Other optional objects in the list are cap.tel, a vector identifying the row of capture for each collared individual, and indCovs, a vector of individual attributes (e.g., sex). One thing to note is that if cap.tel is used, any collared individuals that were *not* captured need to be sorted to the end of the fixfreq matrix and indCovs vector (obviously they are simply absent from cap.tel). As always, each object must be packaged within a list to support multi-session functionality in oSCR.

```
# list of lists (lol) format
telemetry <- list(fixfreq=list(nfix),
                  cap.tel=list(cap.tel))
```

The last data preparation step is to make the SCR dataframe for oSCR. This could be done automatically by data2oscr if we were working with a dataframe of encounters but here we manually created our y_{ijk} capture array. In either approach, the entry of telemetry and rsfDF objects signifies the presence of telemetry data and interest in an RSF surface for the SCR model. Importantly, adding an rsfDF object even without any telemetry data will prompt the generation of trap-specific covariates for each column of the rsfDF that indicates a potential landscape covariate.

```
# Create the scrFrame
sftel <- make.scrFrame(caphist = list(y.arr),
                       traps = list(X),
                       telemetry = telemetry,
                       rsfDF = list(rsfDF)
)
```

Finally, we can fit some models. To integrate telemetry data, we need to set the option to either "ind" or "dep" (default is "none"), indicating whether our collared individuals are an independent or dependent sample. We'll first compare results between a typical SCR model and one with telemetry integration that assumes independence.

```
fit1 <- oSCR.fit(scrFrame=sftel,ssDF=list(ssDF),DorN="D",encmod="CLOG",
                 #rsfDF=list(rsfDF),RSF=TRUE,
                 trimS=sftel$mdm,
                 model=list(D~1,p0~z,sigma~1,path~1))

fit2 <- oSCR.fit(scrFrame=sftel,ssDF=list(ssDF),DorN="D",encmod="CLOG",
                 rsfDF=list(rsfDF),RSF=TRUE,telemetry="ind",
                 trimS=sftel$mdm,
                 model=list(D~1,p0~z,sigma~1,path~1))
```

```
data.frame(fit1=fit1$outStats,fit2=fit2$outStats[,2:3],
           truth = c(alpha0,log(sigma),alpha2,log(N/1600)))
```

```
##   fit1.parameters  fit1.mle fit1.std.er   fit2.mle fit2.std.er      truth
## 1  p0.(Intercept) -2.005576   0.4521090 -1.5074864  0.33108019 -1.5000000
## 2 sig.(Intercept)  0.889889   0.1648810  0.6483914  0.03608053  0.6931472
## 3        t.beta.z  1.287438   0.3229795  1.0670170  0.13865802  1.0000000
## 4  d0.(Intercept) -3.576972   0.2858384 -3.4862682  0.28572999 -3.4657359
```

The parameter estimates of the integrated model are all closer to the data-generating values and the standard errors are much lower, at least for the parameters relating to the encounter model.

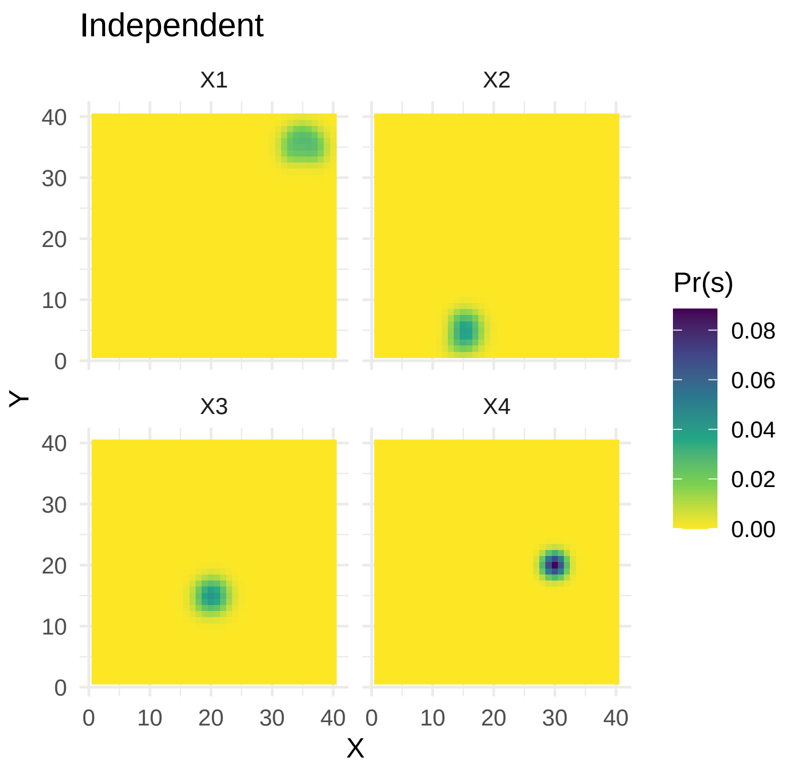
We can then fit an integrated model that admits dependence between the samples, since our collared individuals were also captured.

```
fit3 <- oSCR.fit(scrFrame=sftel,ssDF=list(ssDF),DorN="D",encmod="CLOG",
                rsfDF=list(rsfDF),RSF=TRUE,telemetry="dep",
                trimS=sftel$mdm,
                model=list(D~1,p0~z,sigma~1,path~1))
```

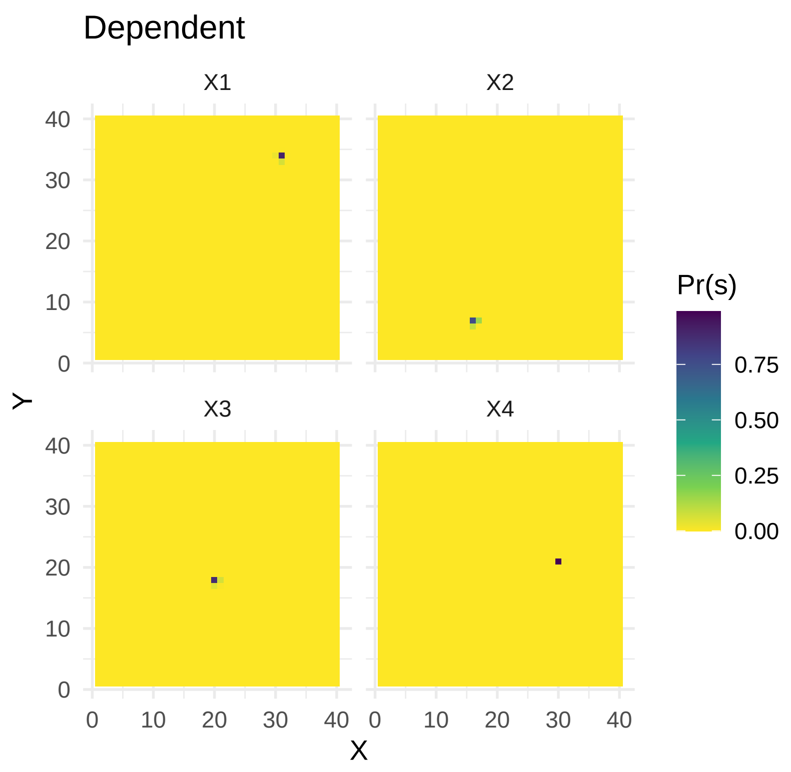
The parameter estimates (not shown here) are largely the same but this model is more appropriate because our inferences on space use came from a sample of 21 individuals, not 25 (21 captured + 4 collared). The other benefit of this model is that the activity center estimates for the 4 collared individuals are going to be far more accurate and precise. We can illustrate the difference by calculating and mapping the posterior distributions of **s** for the N_{tel} individuals.

```
# telemetry = independent
pred2 <- predict.oSCR(fit2)
pred.s2 <- pivot_longer(
  data.frame(gr,t(pred2$preds[[1]][cap.tel,])),-c(1:2),names_to="ind",values_to="Pr(s)"

# plot Pr(s) by collared individual
```

```r
ss.plot + geom_tile(data=pred.s2,aes(fill=`Pr(s)`)) +
  scale_fill_viridis_c(direction = -1) + facet_wrap(~ind) + labs(title="Independent")
```



```r
# telemetry = dependent
pred3 <- predict.oSCR(fit3)
pred.s3 <- pivot_longer(
  data.frame(gr,t(pred3$preds[[1]][cap.tel,])),-c(1:2),names_to="ind",values_to="Pr(s)")

# plot Pr(s) by collared individual
ss.plot + geom_tile(data=pred.s3,aes(fill=`Pr(s)`)) +
  scale_fill_viridis_c(direction = -1) + facet_wrap(~ind) + labs(title="Dependent")
```



The increased accuracy and precision in estimating the latent activity centers should improve other parameter estimates, most notably if the point process model is inhomogeneous and variation in density is being explored.

1.3 Application to NY black bears

The application in Royle et al. (2013) comes from a study on black bears in New York (Sun, 2014). The data object (`nybears`) can be loaded and used to replicate their analysis. In this case, we will use `data2oscr` to create our SCR dataframe. Also, the telemetry data are raw fix locations that require some manipulation for use in the integrated model.

First, the data are loaded and the UTM coordinates are rescaled so that 1 distance unit = 10 km.

```
data("nybears")
ls(nybears)
```

```
## [1] "edf"       "elevation" "K"         "ss"        "tdf"       "teldata"
## [7] "traplocs"  "y2d"
```

```
# capture data
edf <- nybears$edf
K <- nybears$K
# rescale spatial coordinates so each unit = 10km
tdf <- nybears$tdf
tdf[,2:3] <- tdf[,2:3]/1e4
colnames(tdf)[2:3] <- c("X","Y")
ntraps <- nrow(tdf)

ss <- nybears$ss/1e4
colnames(ss) <- c("X","Y")

# telemetry data
fixes <- nybears$teldata[,c("animalid","X_UTM","Y_UTM")]
colnames(fixes)<- c("ind","X","Y")
fixes[,c("X","Y")] <- fixes[,c("X","Y")]/1e4

# create the state space and RSF surfaces with elevation covariate
ssDF <- rsfDF <- data.frame(ss,z=nybears$elevation)
```

An important consideration now is that the telemetry data are from GPS collars with an hourly fix rate. The RSF model assumes independent observations of space use and short time intervals between observations should lead to serial autocorrelation that violates assumptions about independence. Royle et al. (2013) spend their penultimate paragraph discussing this issue and argue it is less of a problem than it might seem. A comprehensive solution would be to model the correlation (e.g., Johnson et al. (2008)). A potentially sufficient solution, one used by Royle et al. (2013), is to thin the telemetry fixes.

```
kp<- seq(1,nrow(fixes),length=0.10*nrow(fixes)) #keep x% of fixes
fixes.thin <- fixes[kp,]
```

The next step is to take these telemetry fixes and convert them into the $N_{tel} \times nG$ matrix of fix frequencies. This is achieved by the `telemetry.processor` function, which requires dataframes of the RSF and fixes (input as session-specific lists) and creates several objects including the `nfreq` that is needed here. Note we include a vector of indicators for sex to the `indCovs` object just for illustration (the values are random and we do not fix any sex-specific models).

```
# create the individual by pixel frequencies of fixes
nfix <- telemetry.processor(list(rsfDF),list(fixes.thin))$nfreq
```

```
# create the telemetry object for oSCR (note, nfix already a list)
telemetry <- list(fixfreq=nfix, indCovs=list(data.frame(sex=c(0,1,0))))
```

Finally, we package the data with the `data2oscr` function in order to create our SCR dataframe and fit the model. Because there were no collared animals captured, we fit the independent version.

```
# create the scrFrame from data2oscr
oSCR.dat <- data2oscr(edf = edf,
                      sess.col = 1,
                      id.col   = 2,
                      occ.col  = 3,
                      trap.col = 4,
                      sex.col  = 5,
                      tdf = list(tdf),
                      K = K,
                      ntraps = ntraps,
                      rsfDF = list(rsfDF),
                      telemetry = telemetry
                      )
sftel <- oSCR.dat$scrFrame

# fit the model "SCR+RSF" from Royle et al. 2013
fit <- oSCR.fit(scrFrame=sftel,ssDF=list(ssDF),DorN="D",encmod="CLOG",
                rsfDF=list(rsfDF),RSF=TRUE,telemetry="ind",
                trimS=sftel$mdm,
                model=list(D~1,p0~z,sigma~1,path~1))
```

```
##  Model:  D ~ 1 p0 ~ z sigma ~ 1 path ~ 1
##  Run time:  28.06683  minutes
##  AIC:  2746.351
##
## Summary table:
##                  Estimate    SE       z P(>|z|)
## p0.(Intercept)     -3.100 0.273 -11.341   0.000
## t.beta.z           -0.263 0.120  -2.184   0.029
## sig.(Intercept)    -0.812 0.036 -22.255   0.000
## d0.(Intercept)     -3.968 0.243 -16.343   0.000
## psi.constant       -0.305 0.352  -0.867   0.386
## *Density intercept is log(individuals per pixel)
##   Nhat(state-space) = exp(d0.)*nrow(ssDF)
##   (caution is warranted when model contains density covariates)
```

The estimates largely match those of model "SCR+RSF" in Table 1 in Royle et al. (2013), with small differences due to a slightly modified state-space (note the SEs are identical). All other model versions in Table 1 could be fit here by changing whether the telemetry integration occurs or which model components use elevation as a covariate (e.g., `D~z`). The code used here can be accessed as an `oSCR` help file with `?telemetry`.

1.4 Summary

Here we have illustrated how to use `oSCR` to integrate telemetry data with spatial captures into a joint SCR model following the approach of Royle et al. (2013). The resulting effort can improve parameter estimation and potentially facilitate sparse data situations and/or variation in resource selection that cannot be addressed with the capture data alone.

Chapter 2

Design criteria

A description and properties of design criteria.

Chapter 3

Preparing design objects

Chris Sutherland & Gates Dupont

3.1 Background

This document provides an example of generating the input objects for the optimal design function, `scrdesignGA()`, which is available in the spatial capture-recapture `R` package, `oSCR`. From the package help file (`?scrdesignGA()`):

> `scrdesignGA()` implements a genetic algorithm to solve this k-of-n design problem. In the context of SCR, and for a given objective function and landscape, the function finds the best subset of sampling locations (k) from all possible sampling locations (n).

We developed this framework with camera traps in mind, but this method can easily be applied to determine the general location of other non-invasive surveys, assuring that effort is standard across all sampling locations.

The functions requires the following input objects:

- `statespace`: A discretized representation of the study area (state-space), defined as a convex hull around the trapping extent with an additional buffer of approximately $3 \times \sigma$ to ensure the incorporation of the activity centers of all individuals available for captured.
- `alltraps`: A two-column dataframe, labelled `X` and `Y` accordingly, comprised of all possible trapping locations. These locations should be user-defined after accounting for logistic constraints, primarily accessibility. Design generation then relies on subsetting from these locations.

The function also requires input values of *beta0* and *sigma* which will come from literature, pilot studies, or best guesses, and the number of traps which will be defined by the user. We do not discuss these inputs any further except to say that in this example we use values of `beta0` = 0.025, `sigma` =3.25, and `ntraps` = 50. Here the focus is on generating the *statespace* and *alltraps* objects.

Before getting started, a few packages should be loaded:

```r
library(oSCR)       # Design-generating function
library(sf)         # Spatial data manipulation
library(raster)     # Raster data manipulation
library(ggplot2)    # Improved plotting
library(viridis)    # Color palette
```

3.2 Creating statespace from a polygon

Consider the motivating example in Dupont *et al.* (in review, preprint on bioRxiv), where the interest is designing an SCR study in an area defined by a watershed that can be represented as a polygon (`pak_poly`):

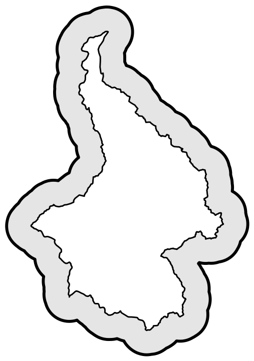
```r
# Plot the study area
plot(st_geometry(pak_poly), main = NULL, col = "grey90")
```

This polygon represents the general area within which traps can be deployed. The help file suggests that the statespace should be "a convex hull around the trapping extent with an additional buffer of approximately $3 \times \sigma$". Creating the statespce object therefore requres first a buffering of the study area, then the creation of a regular grid of points withing that buffer.

3.2.1 Buffer the trappping area

The first step is to buffer `pak_poly` using a buffer width of $3 \times \sigma$. In this case, σ is 3.25, and three times that is 9.75. The projection of `pak_poly` is in UTM (i.e., meters), so we also need to make sure to convert σ, which is in km, to m.

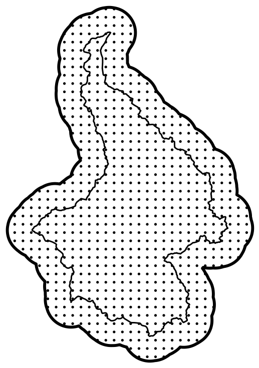
```r
# Assign buffer distance
buff <- 3*sigma*1000  #change to m

# Create state-space from study area and buffer
ss_polygon <- st_buffer(pak_poly, dist = buff)

# Plot the state-space and study area
plot(st_geometry(ss_polygon), lwd = 2, col = "grey90")
plot(st_geometry(pak_poly), lwd = 1, col = "white", add = TRUE)
```



3.2.2 Generating a regular grid

The second step is to generate a regular grid of points within this area that represents the possible activity centers of all individuals that could be captured by any trap placed in `pak_poly`. This process is straightforward using the the spatial sampling function in the `sf` package (`sf::st_sample()`) and specifying `type = "regular"`. It does require that we approximate the number of points (the `size` argument), which requires that we also define the resolution (spacing) of this grid. We use a resolution equal to σ, and therefore, the number of points is the area divided by the area of a pixel with sides equal to sigma:

```r
# Assign the resolution of the state-space
ssgrid_size <- as.numeric(round((st_area(ss_polygon)/1000^2) / sigma^2))

# Sample points within the state-space at that resolution
ss_points <- st_sample(x = ss_polygon, size = ssgrid_size, type = "regular")

# Plot the state-space as pixels, the study area polygon, and the state-space polygon
plot(st_geometry(ss_points), pch = 16, cex = 0.25)
plot(st_geometry(pak_poly), lwd = 1, add = TRUE)
plot(st_geometry(ss_polygon), lwd = 2, add = TRUE)
```



3.2.3 Extracting the coordinates

The `statespace` object should be a 2-column matrix or dataframe countaining the `X` and `Y` coordinates. These can be extrated directly from the spatial object `ss_points` using the `coordinates()` function. For convenience, here all coordinates are converted to the km scale (for UTM, divide by 1000).

```
# Generate state-space df from pixels
statespace <- data.frame(X = st_coordinates(ss_points)[,1]/1000,
                         Y = st_coordinates(ss_points)[,2]/1000)
```

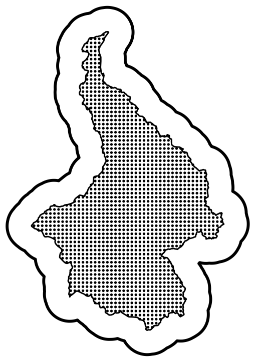
3.3 Creating `alltraps` from a polygon

3.3.1 Create all technically possible points

Using the same principles and functions as the previous section, the first step for creating the `alltraps` object is to generate a regular grid of points within this area that represents all possible trapping locations. We use $0.5 \times \sigma$ here, but as long as the spacing is not larger than `sigma` the end result should not be effected†. Again, this process is straightforward using the the spatial sampling function in the `sf` package (`sf::st_sample()`) and specifying `type = "regular"`.

† this has not been formally evaluated, but should be easy to test.

```
# Determine possible trap spacing
traps_size <- round(as.numeric((st_area(pak_poly)/1000^2) / #m -> km
                         (0.5*sigma)^2))

# Generate traps using that spacing
alltraps <- st_sample(x = pak_poly, size = traps_size, type = "regular")

# Plot the state-space, study area, and possible trap locations
plot(st_geometry(ss_polygon), lwd = 2)
plot(st_geometry(pak_poly), lwd = 1, add = TRUE)
plot(st_geometry(alltraps), pch = 16, cex = 0.25, add = TRUE)
```



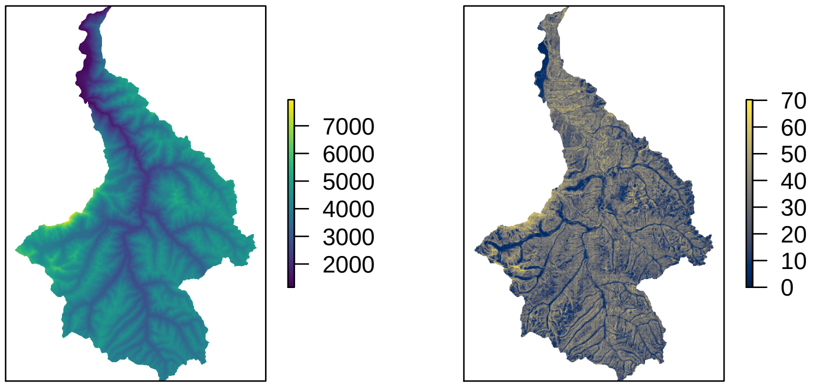
3.3.2 Create all practically possible points

Part of the motivation for developing this design generation framework was to accomodate sampling restrictions and logistical constraints in the design phase (as opposed to adjusting *in situ*). To demonstrate this in practice, sampling in these mountainous areas of the snow leopard range in Pakistan has two obvious challenges:

1. areas too steep cannot be sampled (we use a slope cut-off of 45°)
2. areas too high (in elevation) cannot be sampled (we use an elevation cut-off of 4500m)

Luckily, elevation layers are available so we can use that (`elev`), and slope can be derived using the `terrain()` function from the `raster` package.

```
#the layer 'elev' has been loaded

# Generate the slope raster from the elevation raster
slop <- terrain(elev, opt = 'slope', unit = 'degrees', neighbors = 8)

# Plot the elevation and slope side-by-side
par(mfrow = c(1,2), new = FALSE)
plot(elev, axes = FALSE, col = viridis(100, option = "D"))
plot(slop, axes = FALSE, col = viridis(100, option = "E"))
```

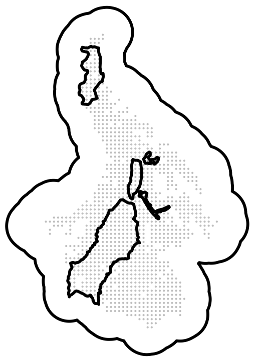


Using the `extract()` function from the `raster` package, we can extract the elevation and slope values for each potential trap location. Then, using the cut-off values, we can identify and select only those points that are 'sampleable'. As with the `statespce` object, the `alltraps` object should be 2-column matrix or dataframe countaining the `X` and `Y` coordinates of the sample points.

```
# Possible trap locations to 2-column object (dataframe or matrix)
allpoints <- st_coordinates(alltraps)

# Extract elevation and slope for the traps
allpoints_elev <- extract(elev, allpoints)
allpoints_slop <- extract(slop, allpoints)

# Subset possible trap locations according to logistic constraints
alltraps <- allpoints[allpoints_elev < 4500  &
                      allpoints_slop < 45    &
                      !is.na(allpoints_elev) &
                      !is.na(allpoints_slop) ,]/1000

# Plot all of the traps below the now subsetted (usable) traps (now called "alltraps")
plot(allpoints/1000, pch = 16, cex = 0.5, col = "grey",
     asp = 1, axes  =FALSE, xlab = "", ylab = "")
points(alltraps, pch = 16, cex = 0.5, col = "blue")
```

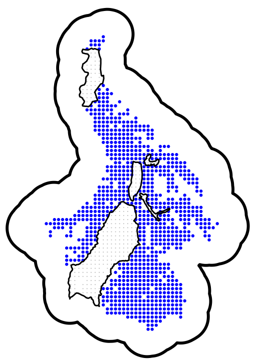
The removal of points can also be done using a polygon that delineated unsampleable areas (e.g., cities or areas that are impossible to access). In the following example, we remove potential trapping points using a polygon called `nogo_areas` that represent inaccessible parts of the landscape.

```
#the shapefile 'nogo_areas' has been loaded
plot(st_geometry(ss_polygon), lwd=2)
points(alltraps*1000, pch=16, cex=0.2, col="grey")
plot(st_geometry(nogo_areas), lwd=2, add=TRUE)
```



Its *much* easier to use spatial objects to identify and remove overlapping regions:

```
alltraps_df <- data.frame(X=alltraps[,"X"]*1000,
                          Y=alltraps[,"Y"]*1000)
alltraps_sp <- st_as_sf(alltraps_df,coords=c("X","Y"))
st_crs(alltraps_sp) <- st_crs(nogo_areas)

#subset the spatial object
alltraps_sp$nogo <- lengths(st_within(alltraps_sp, nogo_areas))
cliptraps_sp <- alltraps_sp[alltraps_sp$nogo == 0,]

plot(st_geometry(ss_polygon), lwd=2)
plot(alltraps_sp, pch=16, cex=0.1, col="grey", add=TRUE)
plot(cliptraps_sp, pch=16, cex=0.3, col="blue", add=TRUE)
plot(st_geometry(nogo_areas), lwd=1, add=TRUE)
```

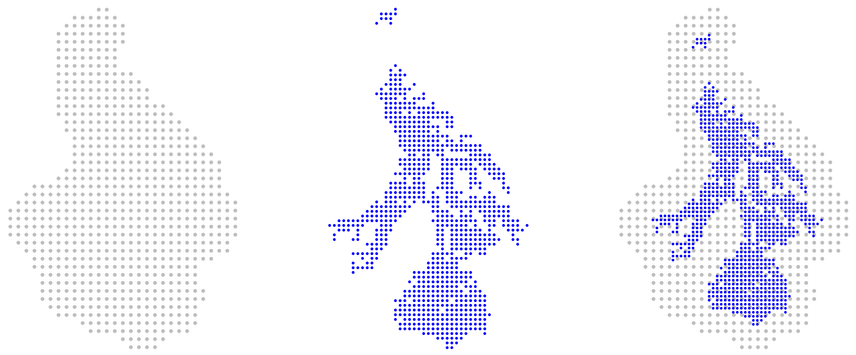


```
alltraps <- st_coordinates(cliptraps_sp)/1000
```

3.4 Inspecting the objects

With both objects created (`statespace` and `alltraps`), it is worth plotiing the objects to confirm that they all line up in space.

```
# Plot the generated objects
par(mfrow = c(1,3))
plot(statespace, pch = 16, cex = 1,    col = "grey", asp = 1, axes = FALSE, xlab = "",
plot(alltraps,   pch = 16, cex = 0.75, col = "blue", asp = 1, axes = FALSE, xlab = "",
plot(statespace, pch = 16, cex = 1,    col = "grey", asp = 1, axes = FALSE, xlab = "",
points(alltraps, pch = 16, cex = 0.7,  col = "blue")
```



Great! Both objects are created and spatially aligned, and the units appear to be on the km scale. These are the landscape (`statespace`) and trap (`alltraps`) inputs for `scrdesignGA()`.

As previously mentioned: if you have any issues or questions, we have a very responsive, and friendly user group.

Chapter 4

Generating designs

Now with data, we can generate some designs.

Bibliography

Johnson, D. S., Thomas, D. L., Ver Hoef, J. M., and Christ, A. (2008). A general framework for the analysis of animal resource selection from telemetry data. *Biometrics*, 64(3):968–976.

Linden, D. W., Sirén, A. P., and Pekins, P. J. (2018). Integrating telemetry data into spatial capture–recapture modifies inferences on multi-scale resource selection. *Ecosphere*, 9(4):e02203.

Royle, J. A., Chandler, R. B., Sun, C. C., and Fuller, A. K. (2013). Integrating resource selection information with spatial capture–recapture. *Methods in Ecology and Evolution*, 4(6):520–530.

Sun, C. (2014). Estimating black bear population density in the southern black bear range of new york with a non-invasive, genetic, spatial capture-recapture study.