Simulations of Ecological dataset using R

Emy Guilbault & Ian Renner

March 2021

In this tutorial, I present some functions for generate presence-only and occupancy data sets that arise from a number of different biological and observation processes, including dynamic processes. The goal is to provide functions to produce simulated data sets that more realistically resemble real data sets. We focus here on Presence-only data (PO) and Occupancy data. Some example follows the "OptimLassoDemo" document developped by Ian Renner, Olivier Gimenez and Julie Louvrier in Renner et al. (2019).

Getting started

We first display a study window of [0, 100]x[0, 100] square. We also generate a quadrature scheme for that area.

```
library(lattice)
library(spatstat)
library(raster)
library(rasterVis)
library(latticeExtra)
library(NLMR)
library(viridis)
library(RColorBrewer)
library(scales)
library(sp)
library(data.table)

source("Simulation Functions.R")
quad = expand.grid(seq(0, 100, 1), seq(0, 100, 1))
names(quad) = c("X", "Y")

win   = owin(xrange = c(-0.5, 100.5), yrange = c(-0.5, 100.5))
```

The simulation function document contains three functions from Renner et al. (2019) (clogloginv, newenv.var and their dependencies), as well as all the new functions presented below.

Setting up covariates

We start setting up covariates using the NLMR package. We use different functions that allow to simulate different processes: fractional brownian motion, gaussian field and midpoint deplacement (Approximation of brownian motion). These processes have been used in some studies to model patterns in hydrology, altitude, rainfall, temperature and terrain surfaces.

- Fractional brownian motion (`nlm_fbm`): For this function the important parameters are the dimension of your window (`ncol` and `nrow`) and the `fract_dim` parameter (value betwen 0 and 2) that allow to

play with smoothness. A higher value will create a smoother raster image. For simulating the same pattern, there is a user.seed parameter.

- Gaussian field: This function presents different parameters that allow the user to modify the raster image. The range of the spatial autocorrelation (`autorr_range`), the magnitude of variation over the window (`mag_var`) and in the scale of the `autocorr_range` (`nug`). For this last argument, a smaller value gives more homogeneous landscape and less patchiness. A seed can also be added for the simulation.
- Midpoint deplacement: This raster image is an approximation of fractional brownian motion. The `roughness` argument controls the level of spatial autocorrelation and `rand_dev` the initial standard deviation for displacement step in the diamond square algorithm.

Here, I will use these functions to generate 4 meaningful covariates for defining species intensity.

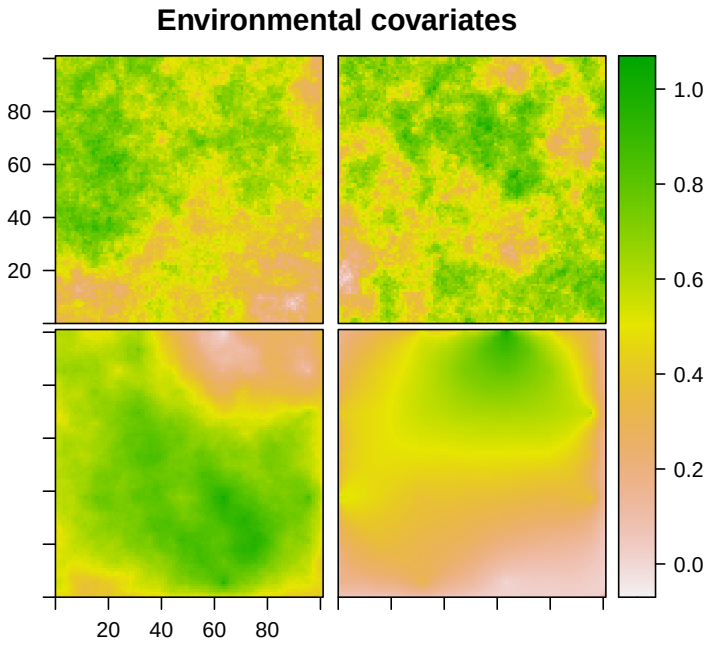
```
# generate environmental covariate using NLMR
set.seed(2)
v1 <- nlm_fbm(ncol = 101, nrow = 101, fract_dim = 0.8, user_seed = 1)
```

```
## NOTE: simulation is performed with fixed random seed 1.
## Set 'RFoptions(seed=NA)' to make the seed arbitrary.
```

```
v2 <- nlm_gaussianfield(ncol = 101, nrow = 101,
                        autocorr_range = 10,
                        mag_var = 80,
                        nug = 1, user_seed = 1)
```

```
## NOTE: simulation is performed with fixed random seed 1.
```

```
v3 <- nlm_mpd(ncol = 102, nrow = 102, roughness = 0.5)

v4 <- nlm_mpd(ncol = 102, nrow = 102, roughness = 0.2)

# Display the plot of the covariates
Lv1 = levelplot(v1, col.regions = rev(terrain.colors(255)),
                cuts=254, margin=FALSE)
Lv2 = levelplot(v2, col.regions = rev(terrain.colors(255)),
                cuts=254, margin=FALSE, colorkey=FALSE)
Lv3 = levelplot(v3, col.regions = rev(terrain.colors(255)),
                cuts=254, margin=FALSE, colorkey=FALSE)
Lv4 = levelplot(v4, col.regions = rev(terrain.colors(255)),
                cuts=254, margin=FALSE, colorkey=FALSE)
comb_levObj <- c(Lv1, Lv2, Lv3, Lv4, layout = c(2, 2), merge.legends = T)
update(comb_levObj, main="Environmental covariates")
```




```
# from raster to dataframe for the landscape variables
Datav1 = as.data.frame(flip(v1,2))
Datav2 = as.data.frame(flip(v2,2))
Datav3 = as.data.frame(flip(v3,2))
Datav4 = as.data.frame(flip(v4,2))

vmat = as.matrix(data.frame(1, Datav1, Datav2, Datav3, Datav4))
```

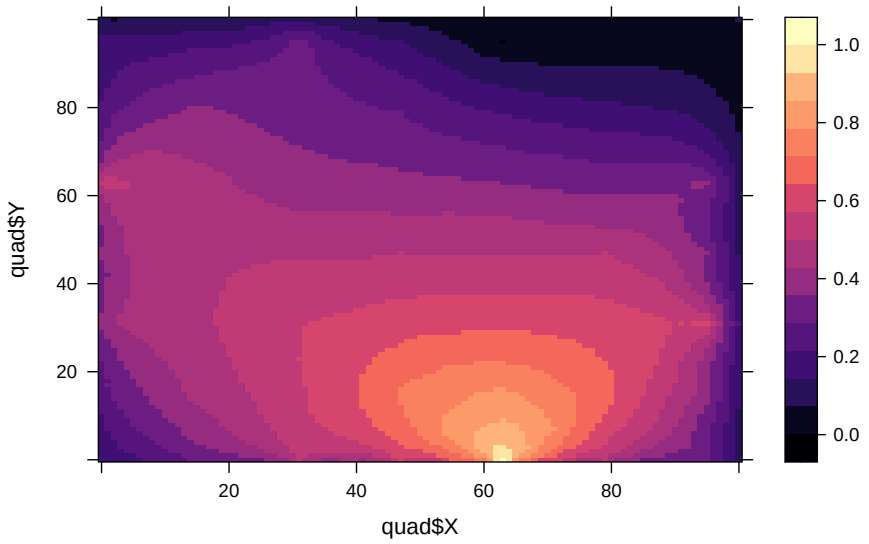
The meaningful covariates are used to generate the point patterns and site history for the occupancy model. We can generate meaningless covariates in the same way to test the performance of shrinkage procedures like the lasso.

Variation in environmental covariates

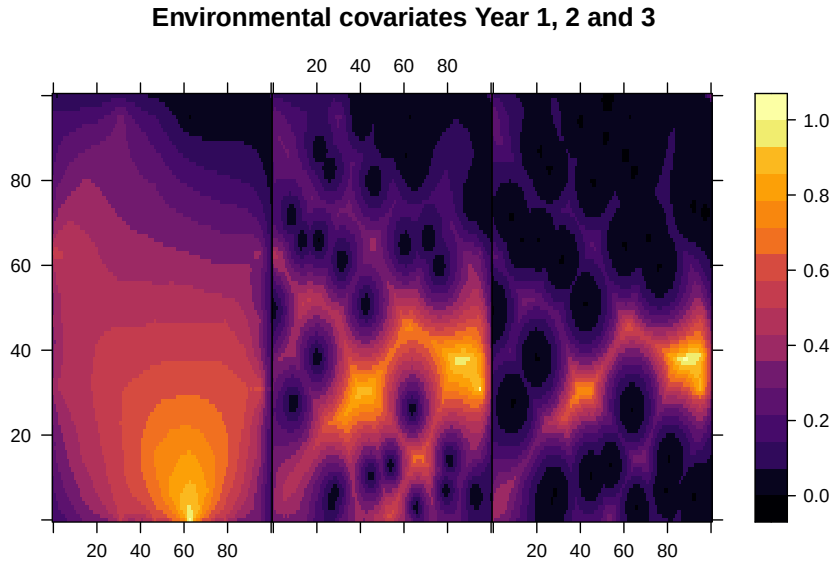
Disturbances or changes in the values of environmental variables can happen from year to year or throughout some period of time. A way to simulate such changes can be through some centers of disturbance (points in space) that will influence the changes in variable intensity depending on the distance to these centers.

First, I randomly create 30 centers of disturbance within the window. I convert them into spatial points and calculate the distance from these centers to the quadrature points throughout the window. Here, I first degrade the covariates based on these distances for the second time period, and then degrade further based on 10 times these distances for the third time period.

```
# For variable v4
v4.df = as.data.frame(v4)
levelplot(v4.df$layer~ quad$X + quad$Y, col.regions=magma(20))
```

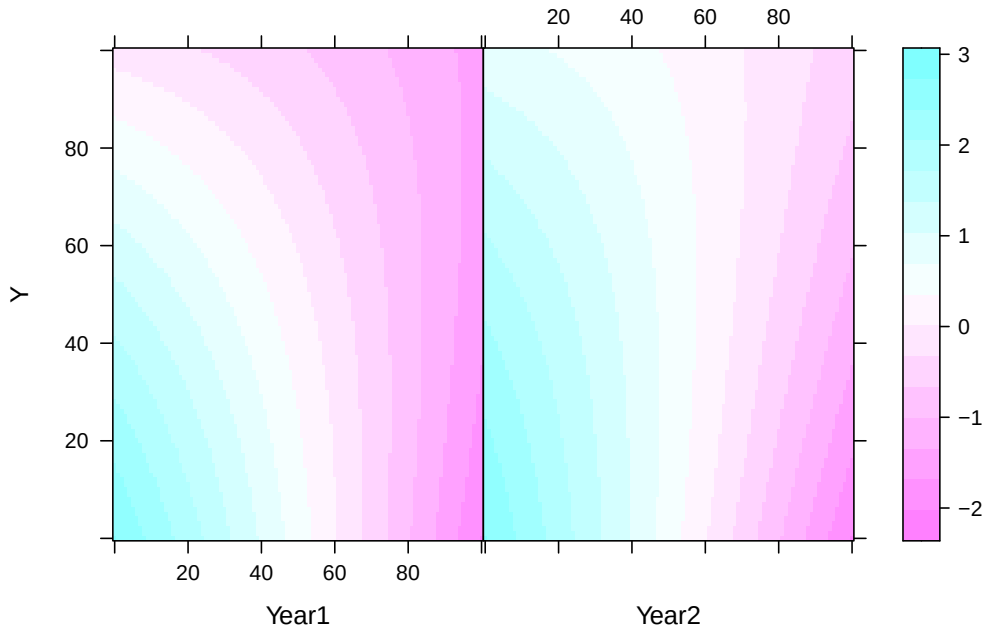



```
L4Y1.test = levelplot(v4.df$layer~ quad$X + quad$Y, col.regions=inferno(20), ylab="", xlab="")

## test with multiple centers randomly created
set.seed(8)
c1 = runif(30, min=0, max=100)
c2 = runif(30, min=0, max=100)
xy2 <- matrix(cbind(c1, c2),ncol=2, byrow=F)
p2 <- SpatialPoints(xy2)

dfp2 = distanceFromPoints(v4, p2)
distfP2 = as.data.frame(dfp2)

v4.df = as.data.frame(v4)
v4tre.Y2 = v4.df*distfP2$layer
v4tre.Y2.Lrsc = (v4tre.Y2$layer-min(v4tre.Y2$layer))/(max(v4tre.Y2$layer)-min(v4tre.Y2$layer))
L4tre.Y2.rsc = levelplot(v4tre.Y2.Lrsc~quad$X+quad$Y, col.regions=inferno(20), ylab="", xlab="")


v4tre.Y3 = v4tre.Y2*distfP2$layer*10
v4tre.Y3.Lrsc = (v4tre.Y3$layer-min(v4tre.Y3$layer))/(max(v4tre.Y3$layer)-min(v4tre.Y3$layer))
L4tre.Y3.rsc = levelplot(v4tre.Y3.Lrsc~quad$X+quad$Y, col.regions=inferno(20), ylab="", xlab="")

update(c(L4Y1.test, L4tre.Y2.rsc, L4tre.Y3.rsc), layout = c(3, 1),
       main="Environmental covariates Year 1, 2 and 3")
```



The idea behind these changing covariates is to induce changes in the resulting patterns of observed points, whether using degraded covariates as above or shifting the covariates through new functional definitions as below.

```
# generate environmental covariate
XY = expand.grid(seq(0, 100, 1), seq(0, 100, 1))
X = XY[,1]
Y = XY[,2]

# Generate new covariates for population density
vd = (X + 300)^2 + (Y + 160)^2 - 5*X*Y

vd = -1*scale(vd)
Lvd = levelplot(vd ~ X + Y)

vd.Y2 = (X + 300)^2 + (Y + 90)^2 - 5*X*Y

vd.Y2 = -1*scale(vd.Y2)
Lvd.Y2 = levelplot(vd.Y2 ~ X + Y)

update(c(Lvd, Lvd.Y2, x.same = TRUE), layout = c(2, 1),
       xlab = list(c("Year1", "Year2"),
                   par.settings = list(layout.heights = list(panel = c(1, 1)))))
```



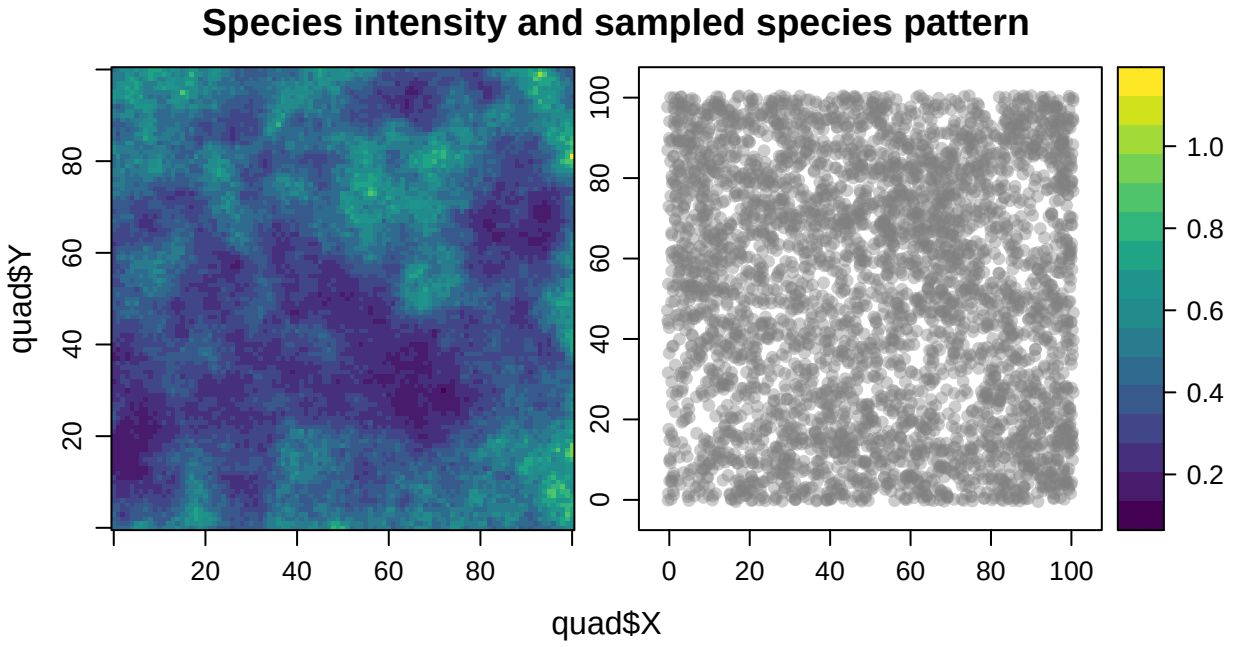
Sample data from a species true intensity:

Aspect 1: variation in recording due to the landscape

True species intensity and pattern

Here, I define the true coefficients used to define the species intensity using linear and quadratic terms of the 4 covariates created earlier. I use the `rpoispp` function of the `spatstat` package to generate points from this defined intensity.

```
sp1_coef = c(-1.5, 0.6, 2, -1, -0.8)
sp1_int = exp(vmat %*% sp1_coef)
sp1_int_im = as.im(data.frame(x = quad$X, y = quad$Y, z = sp1_int))
LV1 = levelplot(sp1_int ~ quad$X + quad$Y, asp = "iso", main = "True intensity",
                col.regions=viridis(30))

sp1_sim = rpoispp(sp1_int_im)

xypo1 = xyplot(sp1_sim$y~sp1_sim$x, ylab="", xlab="", pch=16, col=rgb(0.5, 0.5, 0.5, alpha=0.4))

comb_levObj <- c(LV1, xypo1, layout = c(2, 1), merge.legends = T)
update(comb_levObj, main="Species intensity and sampled species pattern")
```

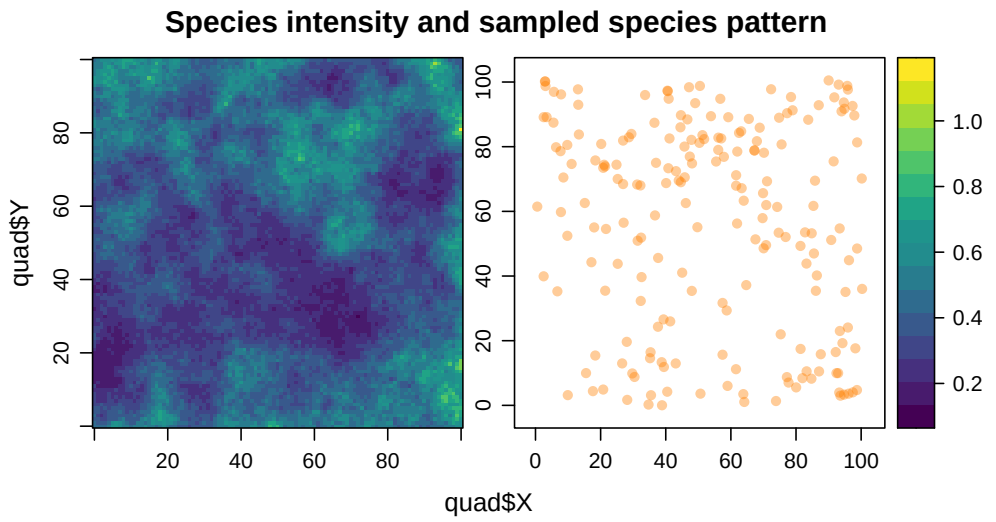


PO data

I obtain covariate information at quadrature points and species locations using functions from Renner et al. (2019). According to the environmental information at the species locations, the quadrature points and the sampled species coefficients (a new intercept control the number of points and the environmental covariates coefficients), I can sample `npoint_samp` to create a sampled point pattern according to its intensity `po_intensity`. These lines are based on Renner et al. (2019) codes.

```
quads_env = data.frame(X = quad$X, Y = quad$Y, V1 = Datav1, V2 = Datav2,
    V3= Datav3, V4 = Datav4)

sp_xy = data.frame(X = sp1_sim$x, Y = sp1_sim$y)
sp_env = newenv.var(sp.xy = sp_xy, env.grid = quads_env,
    env.scale = 1, coord = c("X", "Y"), file.name = NA)

intsamp=-1.5
npoint_samp = 200
po_X = data.frame(Intercept = 1, sp_env[,c(3:dim(quads_env)[2])])
po_beta = c(intsamp, sp1_coef[-1])
po_intensity = exp(as.matrix(po_X) %*% po_beta)
PO_rows = sample(1:sp1_sim$n, npoint_samp, prob = po_intensity)
PO = sp1_sim[PO_rows]

xypo = xyplot(PO$y~PO$x, ylab="", xlab="", pch=16, col=rgb(1, 0.5, 0, alpha=0.4))

comb_levObj <- c(LV1, xypo, layout = c(2, 1), merge.legends = T)
```

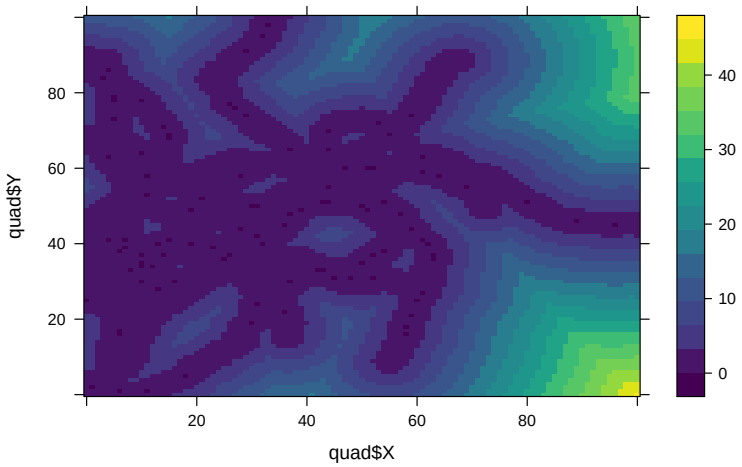
```
update(comb_levObj, main="Species intensity and sampled species pattern")
```



Generate observer bias through a road network

The method of biased sampling comes from Renner et al. (2019).

```
load("road.RDATA")

levelplot(d_rd~quad$X+quad$Y, col.regions=viridis(20))
```

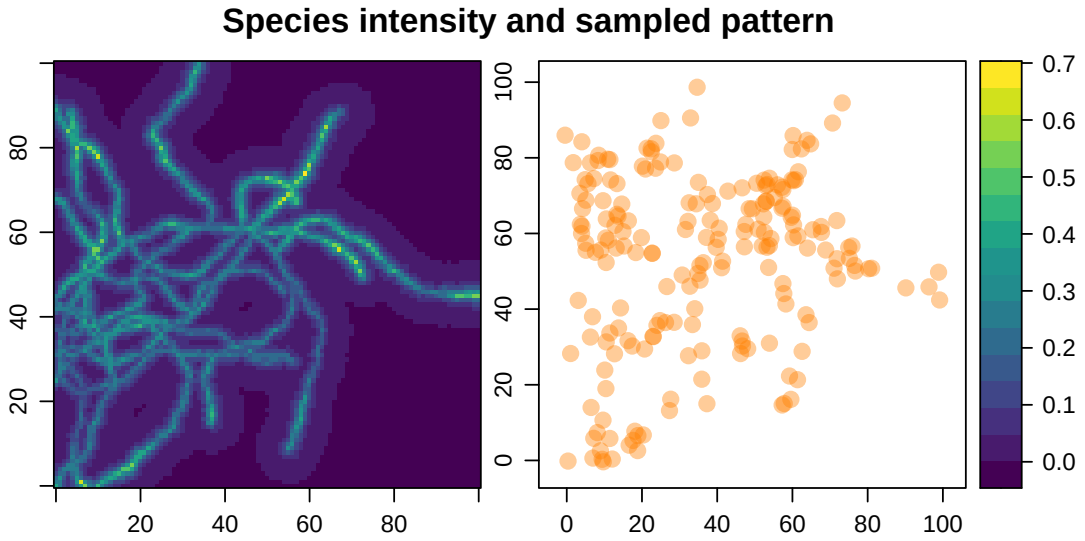


Generating PO data with observer bias from a species true intensity

The `simpo` function allows to simulate biased presence-only patterns from an underlying species intensity and take the following arguments:

- `quadsenv`: a dataframe that contains the environmental information at the quadrature points
- `sp_coef`: the coefficients associated to each environmental covariate
- `intsamp`: the intersept value for the sampled pattern
- `npoint_samp`: the number of point to sample from the true point pattern
- `samp=TRUE`: decision on whether to sampled observation or only show the true pattern
- `plot`: decision on whether to display the plots or not
- `compPlot`: decision on whether to display comparative plots of the intensity and the corresponding pattern.

```
env.mat = as.matrix(data.frame(quad$X, quad$Y, Datav1, Datav2, Datav3, Datav4, d_rd))
colnames(env.mat)=c("X", "Y", "v1", "v2", "v3", "v4", "d_rd")

sp_coef = c(-1.5, 0.6, 2, -1, -0.8, -0.8)

Test1 = simpo(quadsenv = as.data.frame(env.mat), sp_coef=sp_coef,
    intsamp=-1.5, npoint_samp=200, samp=TRUE, plot=TRUE, compPlot = TRUE)

Test1
```



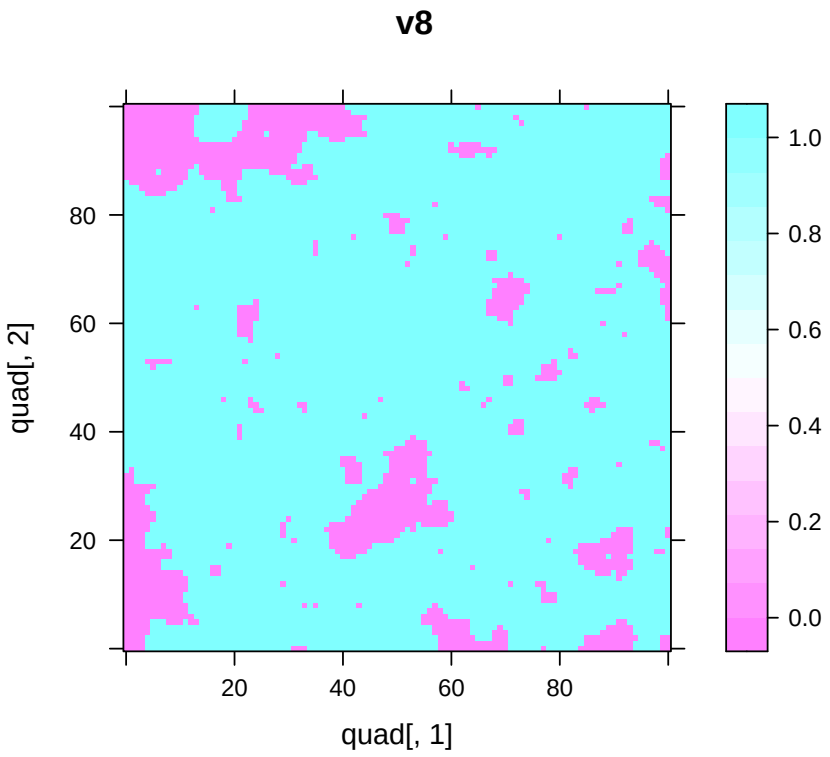
Habitat facilitate the detection

In this context, we consider a categorical variable that would represents patches of habitat where the detection is easier.

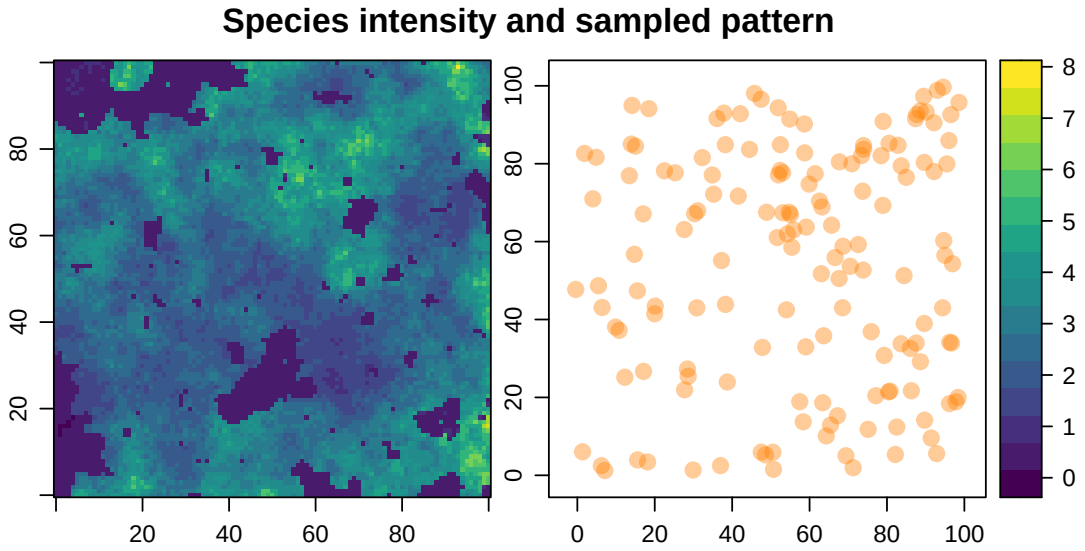
For PO data

Here, I generate this covariate using a random cluster function from the `NLMR` package. The function `nlm_randomcluster` simulates clusters of nearest neighbours. The parameter p is the proportion of elements randomly selected from the cluster, `ai` is a vector of cluster types, i.e. a vector of percentages of occupancy.

```
set.seed(13)
v7 = nlm_randomcluster(ncol = 101, nrow = 101, p = 0.6,
                       ai = c(0.25, 0.25, 0.5))
Datav7=as.data.frame(v7)
v8 = rep(0, length(v7)) # categorical for PO bias

v8[Datav7 >= quantile(v7, 0.75)] = 1
levelplot(v8 ~ quad[,1] + quad[,2], main = "v8", asp = "iso")
```



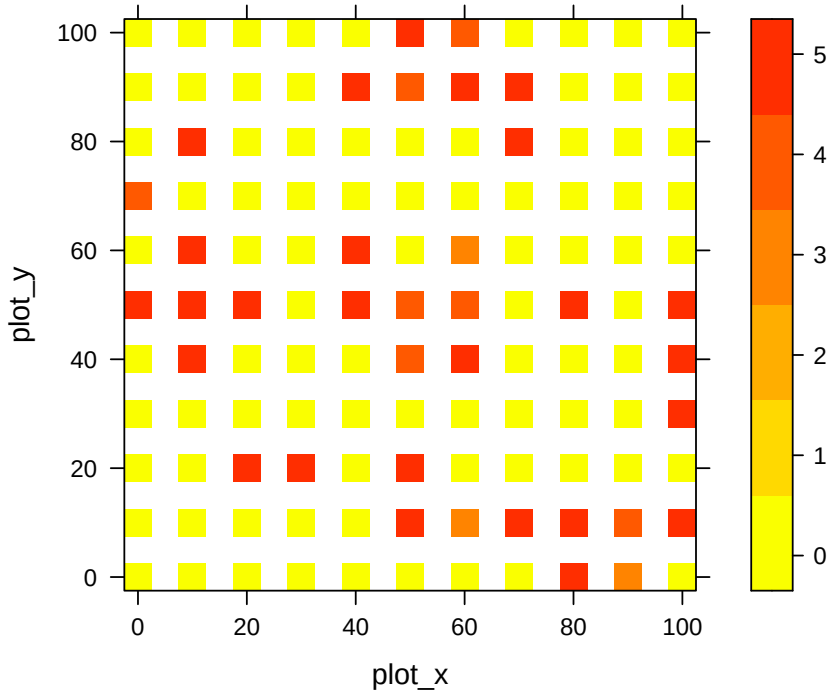

```
env.mat2 = as.matrix(data.frame(quad$X, quad$Y, Datav1, Datav2, Datav3, Datav4, v8))
colnames(env.mat2)=c("X", "Y", "v1", "v2", "v3", "v4", "v8")

sp_coef2 = c(-1.5, 0.6, 2, -1, -0.8, 2)

Test2 = simpo(quadsenv = as.data.frame(env.mat2), sp_coef=sp_coef2, intsamp=-1.5,
    npoint_samp=150, samp=TRUE, plot=TRUE, compPlot = TRUE)
Test2
```



For occupancy data

Another way that species observations may be recorded from the true distribution is through occupancy data, which is demonstrated in Renner et al. (2019). I have created a function `simocc` to extend this code to display multiple occupancy data simulations. This function uses two parameters define for `simpo`: `sp_coef` and `quadsenv`. For occupancy data, we can consider a probability of detection according to the same categorical variables `v8` created previously which is the 7th environmental covariates in our `quadsenv` dataframe, hence we set `vardetect=7`. Here, we can set the number of visits `n_visits` to generate a site history. The `upres` argument allows us to set the maximum distance from the centre of the occupancy site within which a species must be for the site to be considered occupied. Here, we set it to 0.2 units.

```
Test3 = simocc(sp_coef=sp_coef2, quadsenv=as.data.frame(env.mat2),
    vardetect=7, n_visits=5, upres=0.2, occ_plot=TRUE)

Test3
```

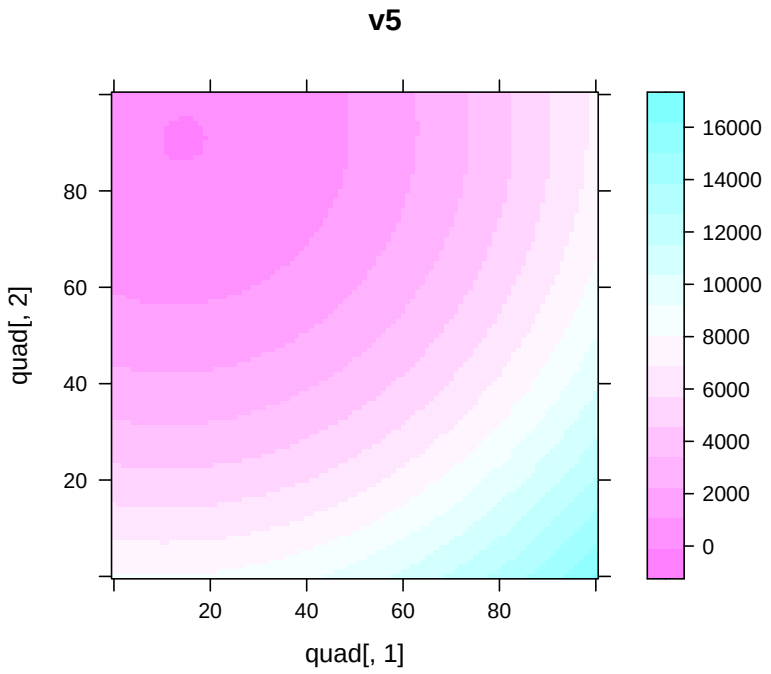


Uneven spatial coverage

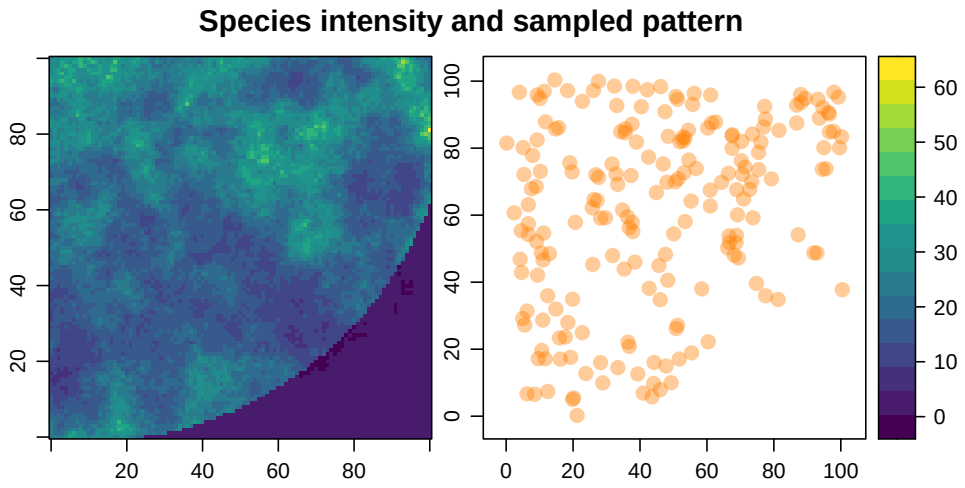
for PO data

We first create a categorical variable that represents an area such as a mountain or pond that serves as an ecological barrier, making observations difficult or impossible.

```
set.seed(13)

# Creating a categorical variable
v5 = (X - 10)^2 + (Y - 90)^2 - 0.1*X*Y
v6 = rep(0, length(v5))
levelplot(v5 ~ quad[,1] + quad[,2], main = "v5", asp = "iso")
```



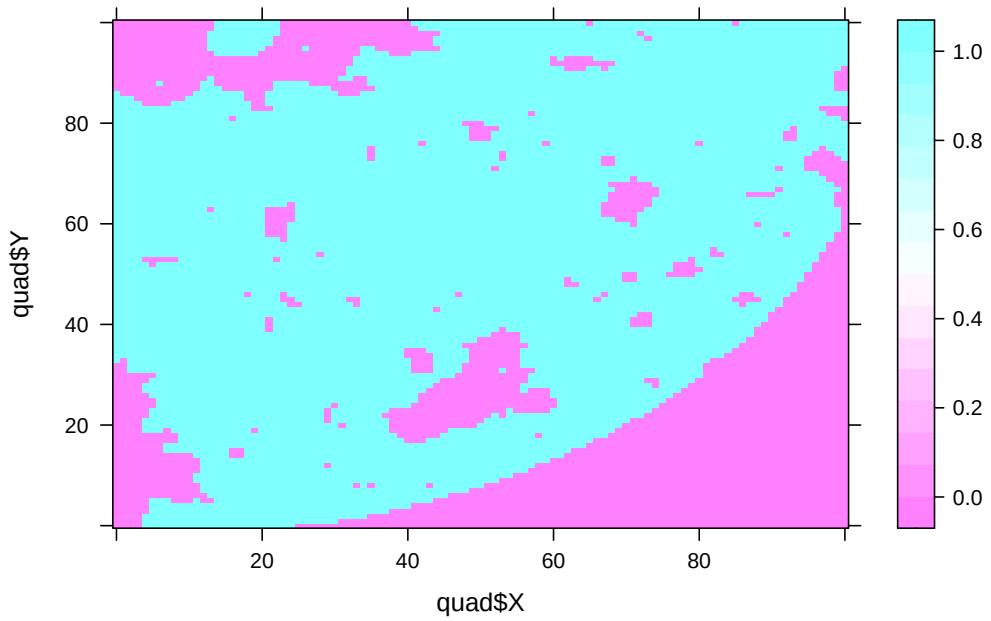

```
v6[v5 <= quantile(v5, 0.85)] = 1

env.mat3 = as.matrix(data.frame(quad$X, quad$Y, Datav1, Datav2, Datav3, Datav4, v6))
colnames(env.mat3)=c("X", "Y", "v1", "v2", "v3", "v4", "v6")
sp_coef3 = c(-1.5, 0.6, 2, -1, -0.8, 4)

Test4 = simpo(quadsenv = as.data.frame(env.mat3), sp_coef=sp_coef3, intsamp=-1.5,
    npoint_samp=200, samp=TRUE, plot=TRUE, compPlot = TRUE)
Test4
```



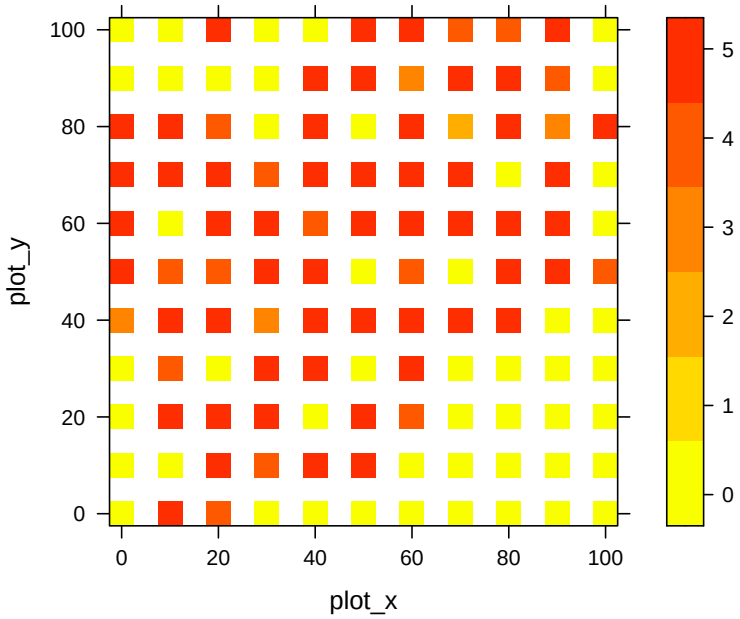
For occupancy data

For occupancy data, ecological barriers result in some survey sites not being visited, and hence no detections can be made there.

```
v6.rast = rast1 <- raster(ncol=101, nrow=101, xmn=0, ymn=0,
                          xmx=100, ymx=100)
ncell(rast1)
values(v6.rast) = v6

# No detection in a certain area from v6
v8.uneven = v8
v8.uneven[values(v6.rast)==0] = 0
levelplot(v8.uneven~quad$X+quad$Y)
```




```
env.mat4 = as.matrix(data.frame(quad$X, quad$Y, Datav1, Datav2, Datav3,
    Datav4, v8.uneven))
colnames(env.mat4)=c("X", "Y", "v1", "v2", "v3", "v4", "v8")

Test5 = simocc(sp_coef=sp_coef3, quadsenv=as.data.frame(env.mat4),
    upres=0.2, vardetect=7, n_visits=5, occ_plot=TRUE)
Test5
```



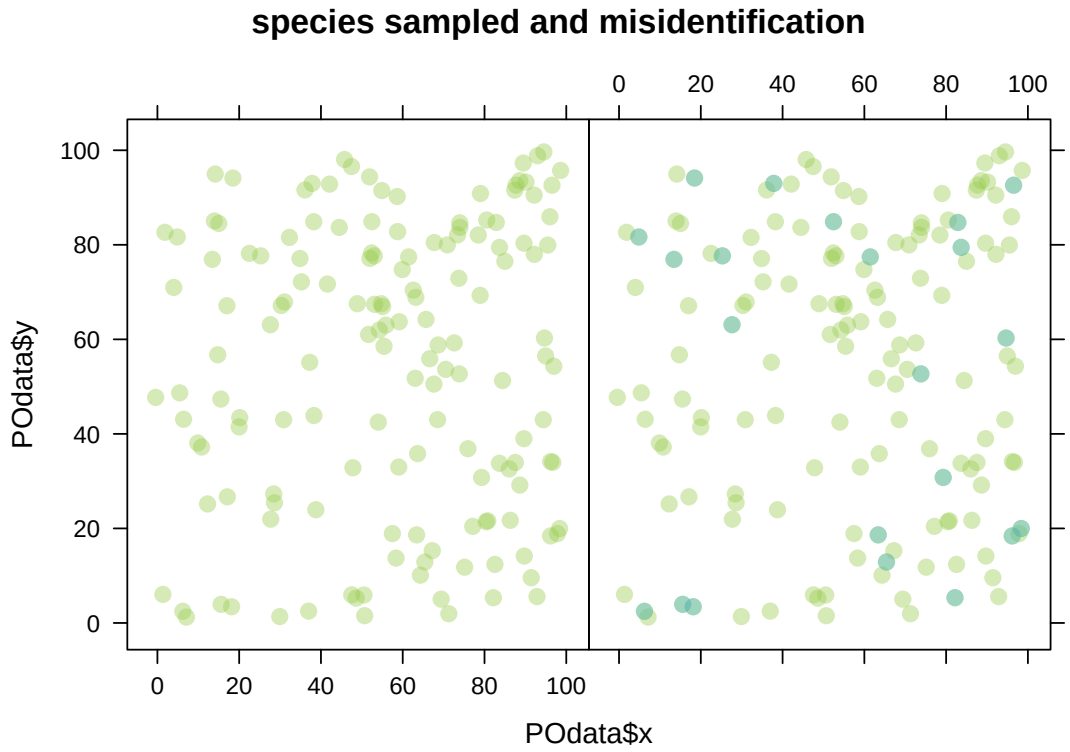
Aspect 2 : Observer recording and accuracy

Misidentificiation

Observers can make wrong observations either by considering some other species observation as the species of interest or by wrongly associating to another species the observation that should have been the species of interest.

- Case 1 (False negative): Observations are not correctly attributed to the species of interest. I have written the function `misidpoints` to simulate misidentification from a supplied point pattern coming from `simpo` output (`simpoRes`). We choose the proportion of points to misidentify with the `perc_mis` argument. It can be chosen to display sampled plots (`compPlot`).

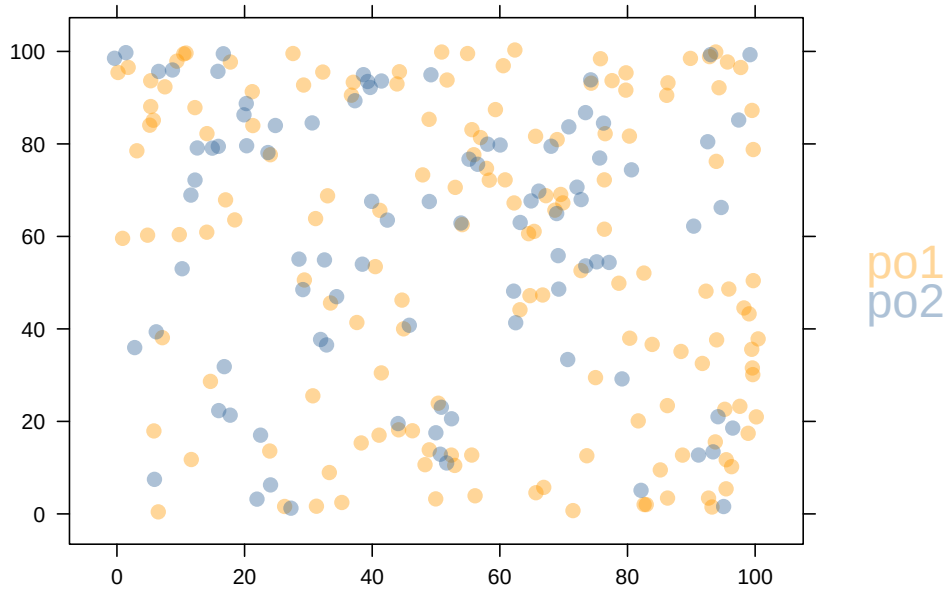
```
misidpoints(simpoRes = Test2, perc_mis=0.15, compPlot=TRUE)
```



- Case 2 (False positive): Observations are wrongly attributed to the species of interest. It can happen that several species have very close distribution because they have similar habitat preferences. In this context, some observations can be wrongly attributed to one or the other species.

I have written the function `multiplepo` to generate multiple species patterns with related distributions through supplied species-level coefficients `sp_coefmult` for the same set of covariates `quadsenv`. The user can specify the number of points to be generated for each species with the `npoint_sampmult` argument and specify the intercept for both sampled point pattern with the argument `intsampmult`. The arguments `sp_coefmult`, `npoint_sampmult` and `intsampmult` are lists of vector or numerical values.

```
sp1_coef = c(-1.5, 0.6, 2, -1, -0.8)
spbis_coef = c(-1.5, 0.3, 1.8, -0.7, 0.4)

multiplepo(sp_coefmult=list(sp1_coef, spbis_coef), quadsenv=as.data.frame(quads_env),
           intsampmult=list(-1.5, -1.5), npoint_sampmult=list(150, 90), plot=TRUE,
           compPlot=TRUE)
```



Data-dynamic

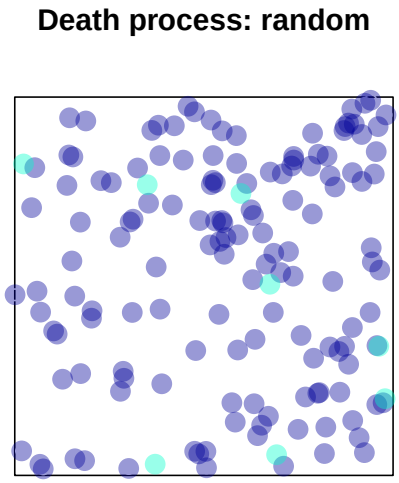
Any species data set provides a time-constrained picture of the underlying species distribution over the time the data were collected. In reality, individuals in the pattern may be born, die, or move for a number of reasons. Here, I present functions to simulate dynamics of point patterns.

Death

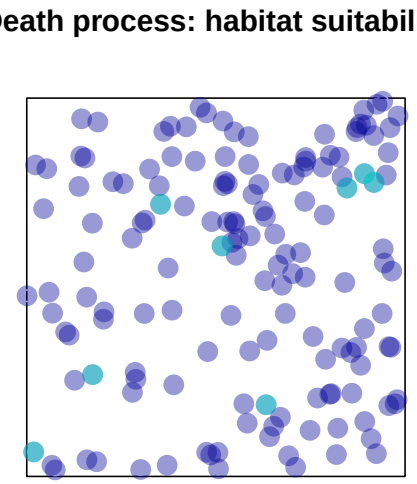
For the death process, we can randomly take out some points according to a probability of death that we choose. The point pattern is then thinned by keeping only the points that did not die.

We could also have higher death rate in less suitable habitat. Thus we need to use the species intensity distribution and use it as a proxy for the habitat suitability. I create a function `text` which allows to simulate the death of point randomly (`random = TRUE`) or using habitat suitability (`random = TRUE`) from a supplied output from `simpo`, `simpoRes`. We define the probability of points that will die with the argument `prob_death` and the standard deviation `sd` when sampling the points in the case of `(random = TRUE)`.

```
deathpoints(simpoRes=Test2, prob_death=0.05, sd=0.01, random = TRUE)
```




```
deathpoints(simpoRes=Test2, prob_death=0.05)
```



Colonization

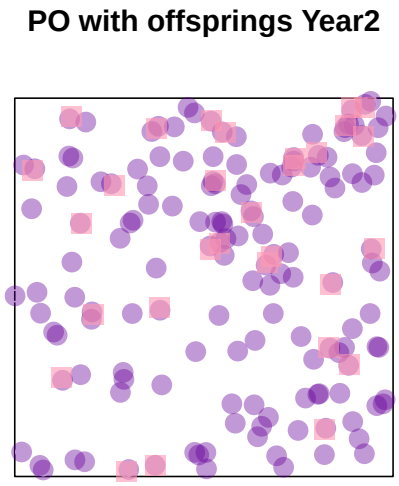
I created a function to simulate birth and colonization processes `birthpoints`. The two types of emerging points simulation are decided using the parameter `birth`. In the case of the birth process (`birth=TRUE`) we define the radius around which the offspring are born (`radius`). For both processes we need to define the proportion of the total points that will be created using `prop_offsp`.

birth

For the birth process, we consider buffer areas around each points and inside these buffers some offsrpings are randomly generated. Not all existing points in the pattern are likely to produce offspring - we might expect that points located in more suitable habitats will be more likely to produce offspring.

We first sample the points that will produce offspring, sampled in such a way that points in more suitable habitats are more likely to be selected as parents.

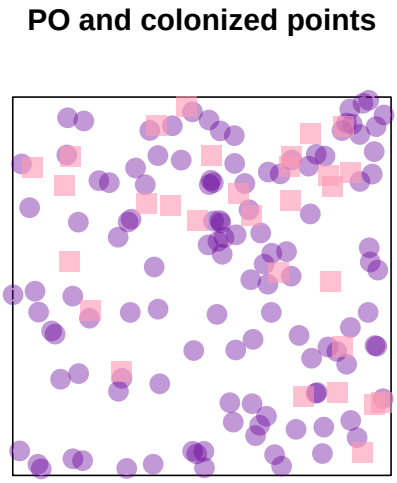
```
Test10 = birthpoints(simpoRes=Test2, prop_offsp=0.2, radius=1,
    birth=TRUE)
```



Colonization from outside the window area

For the external colonization process, we simulate new points from the same species intensity as was used to generate the existing point pattern. In this code, as we simulate the new observations from the exact same habitat as species 1 and using the same species intensity, the new points happen in more suitable habitat.

```
Test11 = birthpoints(simpoRes=Test2, prop_offsp=0.2, birth=FALSE)
```



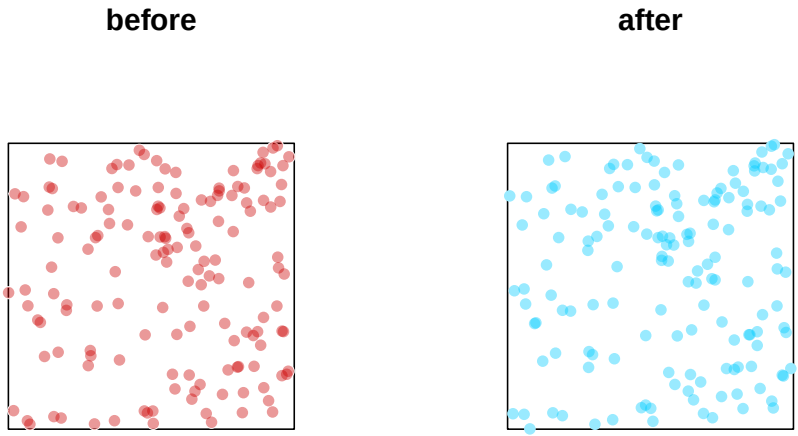

```
Test11
```

Movement inside the window area

Random movements

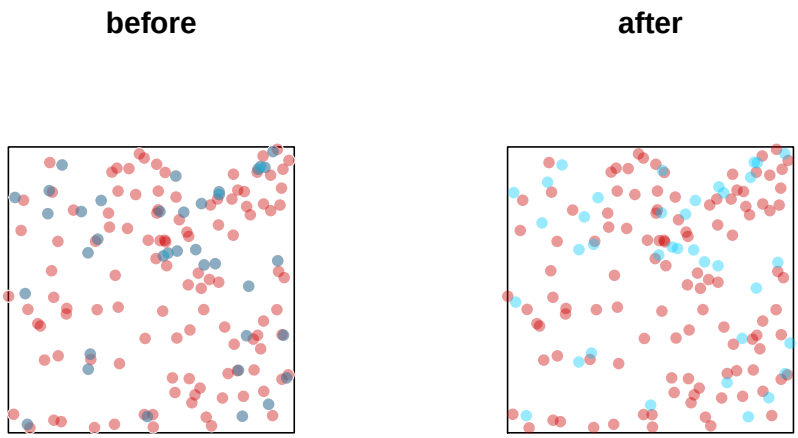
For the possible movements inside the window area, different cases can be done. First we can consider random movements of the points using the function `movepoints`. This function also uses the output `simpoRes` from the function `simpo`. We can control the sampling using `mean` and `sd`.

```
set.seed(11)
Test12 = movepoints(simpoRes=Test2, mov_type = "random", mean=0, sd=1,
    Allpt = TRUE, p_move=NULL)
```



If we decide that not the whole point pattern will move (`Allpt=TRUE`), we need to specify the proportion of points that will move with `p_move`.

```
Test12b = movepoints(simpoRes=Test2, mov_type = "random", mean=0,
    sd=2, Allpt = FALSE, p_move = 0.25)
```

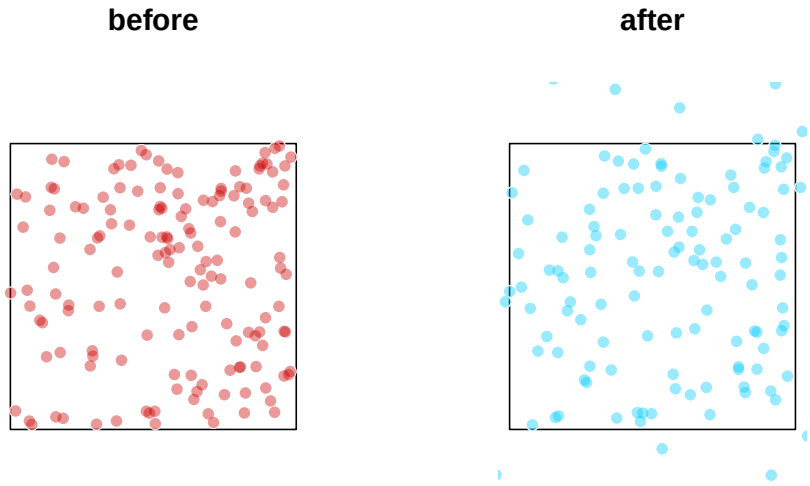


Habitat suitability movements

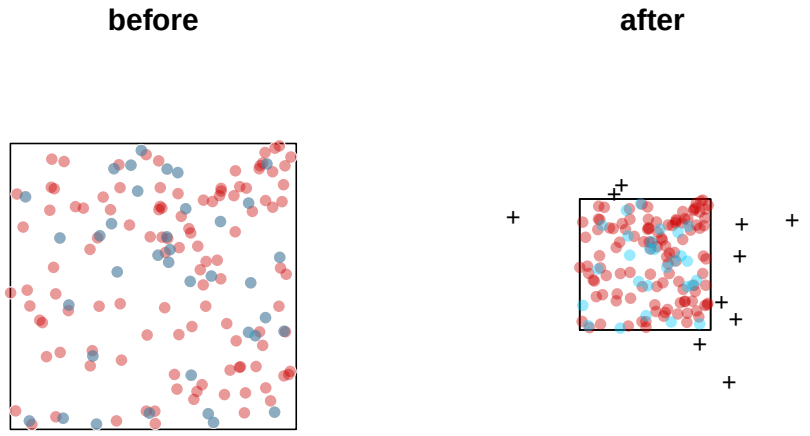
We could also consider that we will get higher random movements in less suitable environment. Thus, with the same function as above we chose `mov_type = "intensity"`. We can also control if all points or a proportion

of points are moving using `Allpt` and `p_move`.

```
Test13 = movepoints(simpoRes=Test2, mov_type = "intensity", mean=0,
    sd=2, Allpt = TRUE, p_move=NULL)
```




```
Test13b = movepoints(simpoRes=Test2, mov_type = "intensity", mean=0,
    sd=2, Allpt = FALSE, p_move = 0.25)
```

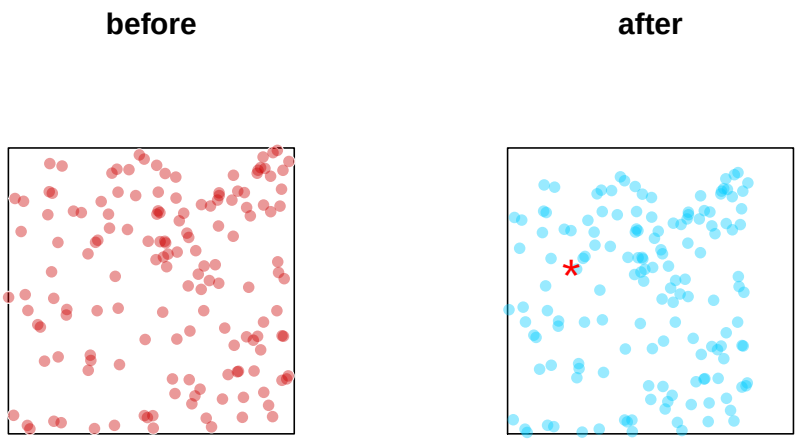


Covariate suitability movements

A second way of moving the points could be according to an environmental covariate. In the following example, I move the points toward the center of a covariate by rescaling their coordinates towards the center.

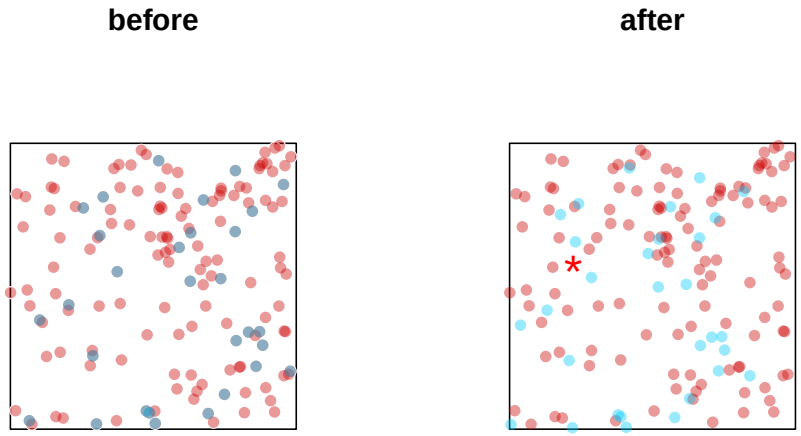
With the option: `mov_type = "center"` of the `movepoints` function, we decide to move points towards the center of the variable defined by `n_var` in the dataframe `quadsenv` which is part of the `simpoRes` output. We can also choose how close of the center of this covariate the points will move the movement distances as the proportion of the distances to the center using the parameter `p_scale`.

```
Test14 = movepoints(simpoRes=Test2, mov_type = "center", mean=0,
    sd=2, Allpt = T, n_var=3, p_move=NULL, p_scale=0.2)
```



Similarly to the previous example we can decide to only move certain points using `Allpt` and `p_move`.

```
Test14b = movepoints(simpoRes=Test2, mov_type = "center", mean=0,
    sd=2, Allpt = F, n_var=3, p_move=0.2, p_scale=0.2)
```



Using multiple functions presented in this tutorial, biased PO and occupancy data can be generated through various environmental covariates (`simpo` and `simocc`). Two functions allow to simulate recording errors. From the biased point patterns, the function `misidpoints` allow to misidentify some observations. `multiplepo` simulate multiple species at the same time. Three dynamic aspects are also shown: death `deathpoints`, birth and colonisation `birthpoints`, and various movements of either the whole point pattern of a proportion of it `movepoints`. Other types of simulations (data, processes) could be built from these functions and more extensions are going to be considered to complete this work.

Renner, IW, Louvrier, J, Gimenez, O. Combining multiple data sources in species distribution models while accounting for spatial dependence and overfitting with combined penalized likelihood maximization. Methods Ecol Evol. 2019; 10: 2118– 2128. https://doi.org/10.1111/2041-210X.13297