Species distribution modelling with Poisson point processes

The Quantitative & Applied Ecology Group (QAECO), The University of Melbourne

Ian Flint and Roozbeh Valavi

2020-03-04

Species distribution modelling

Species distribution modelling is…

The example data used in this tutorial comes form Fithian and others (2015).

Envrionmental data

The environmental data are a series of climatic and topographic variables. We already transferred the coordinate reference system (CRS) of all the input data from geographic CRS to a metric system (UTM zone 56, south). So, the unit of the maps is metre.

```r
# library(maptools)
library(raster)

grid_dir <- "data/grids_utm"
vars <- list.files(grid_dir, pattern = ".tif$", full.names = TRUE)

# read the raster layers as a raster stack
nsw_stack <- stack(vars)

# you can plot them by: plot(nsw_stack)
```

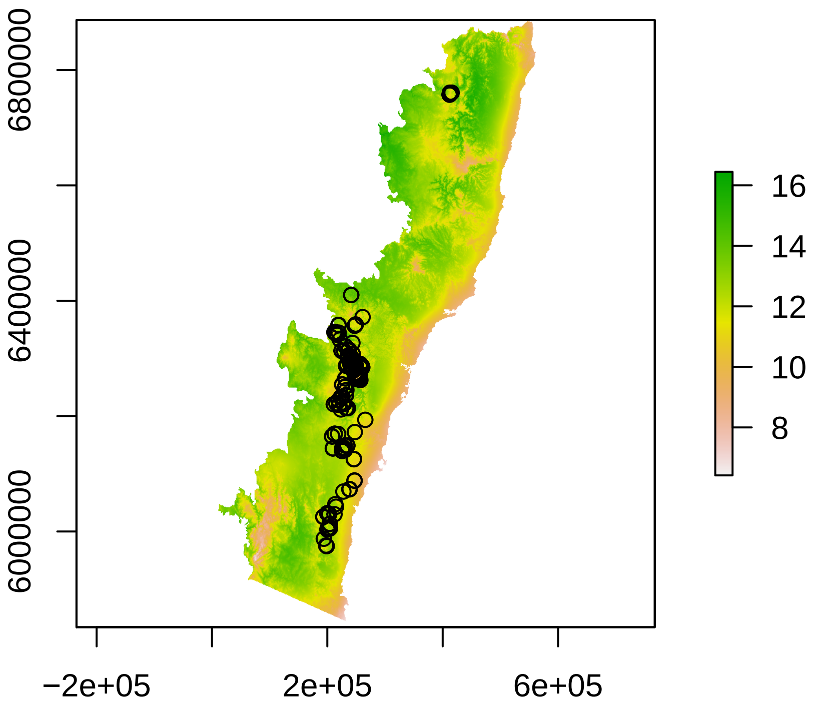
Species data

The species presence-only data consists of 36 eucalypt species in south-eastern Australia. We use an example species to fit MaxNet and PPM models. As documented in Fithian et al. (2015), the species data has a bias towards cities and towns, especially toward Sydney, the biggest city in the study area. The rest of the species is used to generate Target Group Background (TGB) samples to be used as one way to account for the sampling bias in the presence-only species data in MaxNet.

```r
# reading species occurrence data
occ <- read.csv("data/species/eucablax.csv")
str(occ)
```

```
## 'data.frame':    183 obs. of  3 variables:
##  $ X : num  221756 249588 251500 251460 248248 ...
##  $ Y : num  6332627 6287597 6268939 6267924 6270796 ...
##  $ po: int  1 1 1 1 1 1 1 1 1 1 ...
```

```r
# plot the sapecies on the raster map
plot(nsw_stack[[1]])
points(occ$X, occ$Y)
```



Modelling with Maxent

Maxent is a very popular tool for modelling species occurrence data. Maxent was first developed as a stand-alone Java program and later a wrapper function was implemented in the `dismo` package to fit SDMs with Maxent in R programming. Here we use the `maxnet` package, the open-source version of `Maxent` in R, newly developed by Phillips *et al.,* (2017) to model species distributions from occurrences and environmental variables, using `glmnet` for model fitting.

```r
# load the libraries
library(maxnet)
library(dismo)  # for generating random points
```

For model fitting with maxnet, we need a vector of presence and background data (with 0 indicating backgrounds and 1 indicating presences) and a `data.frame` of the environmental covariates.

First, we need to get the background samples. The background samples are typically selected randomly from the landscape. The number of samples should be high enough to represent the environmental variability in the landscape of interest. Based on the conventional wisdom, 10,000 random background is commonly used for model fitting with background, However, evidence were provided in the literature related to spatial point processes that this number is not enough. Here we have 3031210 number of non-NA pixels in our landscape, so clearly the 10,000 is not enough. Although, for the purpose of this tutorial, we limit the number of background points here to speeds up the computation.

After generating the random bacground samples, we need to extract the values of environmental variables. This will be a `data.frame` that later will be used in model fitting.

```
# generate some random samples
rnd <- randomPoints(nsw_stack, n = 25000)

# extracting the values of rasters for the background points
background <- raster::extract(nsw_stack, rnd)
head(background)
```

```
##          bc02     bc04     bc05      bc12      bc14     bc21      bc32
## [1,] 12.78696 1.452687 28.18386 1183.9574  6.057946 23.07162 0.9992301
## [2,] 11.47992 1.432350 28.80465  930.2715  8.558402 25.92738 0.8825114
## [3,] 11.57093 1.597248 25.96810  864.6246 11.300412 25.70141 0.9151739
## [4,] 13.19624 1.638541 30.01427  790.4206  8.802155 24.98329 0.7856771
## [5,] 11.03801 1.543735 25.38669 1291.4635 14.685533 23.40656 1.0000000
## [6,] 11.95732 1.505784 25.15636 1036.1412  7.258162 23.55991 0.9169670
##           bc33 mvbf     rjja      rsea      rugg     twmd     twmx
## [1,] 0.6446267    0 133.1010 1.2117937  84.80847 5881.826 10979.53
## [2,] 0.4559298    0 161.1522 0.4684293  13.03116 7657.112 12930.74
## [3,] 0.3703640    0 183.1257 0.2536727  33.23503 7930.766 13284.49
## [4,] 0.3968723    0 163.2192 0.4105503  53.22516 6153.022 11366.04
## [5,] 0.8134366    0 249.7320 0.4334998 125.78829 5877.465 10584.35
## [6,] 0.5658982    0 127.6450 1.0346737  51.20577 6863.161  9597.29
```

```
# check for any NA in the extracted values
anyNA(background)
```

```
## [1] TRUE
```

```
# remove the NA values
background <- na.omit(background)
```

The values of the environmental variable for the species occurrences also need to be extracted. Then the covariates of the presence and background should be combined in a single `data.frame`.

```
# subset the covariate in species data by raster
occ_env <- raster::extract(nsw_stack, occ[, c("X", "Y")])
# now add them together
covariates <- as.data.frame(rbind(occ_env, background))
head(covariates)
```

```
##        bc02     bc04     bc05      bc12     bc14     bc21      bc32      bc33
## 1 11.972410 1.717385 26.59623  783.9828 10.82395 25.43798 0.9177738 0.3925151
## 2  9.897316 1.540964 23.80956 1144.0826 14.38624 24.42523 1.0000000 0.6849323
## 3  9.896243 1.527091 24.24118 1220.1611 13.21331 24.32824 1.0000000 0.7116060
## 4  9.762543 1.523916 24.03707 1243.3524 13.43661 24.29864 1.0000000 0.7256904
## 5  9.715970 1.533431 23.88721 1198.0652 13.69844 24.44256 1.0000000 0.7012489
## 6 10.229693 1.527476 24.77595 1235.9242 12.68289 24.15882 1.0000000 0.7258136
##   mvbf     rjja      rsea     rugg     twmd      twmx
## 1    0 175.2417 0.2601534 50.00940 5972.308  7818.455
## 2    0 209.0410 0.4728162 20.49779 6058.351 10611.254
## 3    0 209.2487 0.5629562 16.08357 8089.800 10523.992
## 4    0 214.0122 0.5510899 20.37995 7510.101 13576.601
## 5    0 213.8317 0.5030060 10.48010 8000.462 12159.029
## 6    0 201.4558 0.6387870 38.48753 6572.515  9368.119
```

Here, we need to make a vector of 1s and 0s. This should be with the same order as the covariate data is provided.

```r
# make the presence and background points
# presnece (1s) and background (0s)
# the order should be the same as the order in the covariates
presences <- c(rep(1, nrow(occ)),
               rep(0, nrow(background)))

tmp <- Sys.time()
mxnet <- maxnet(p = presences,
                data = covariates,
                regmult = 1, # regularisation multiplier
                maxnet.formula(presences, covariates, classes = "lqph"))
Sys.time() - tmp
```

```
## Time difference of 33.24719 secs
```

```r
# plot the fitted function
# plot(mxnet, type = "cloglog")
```

Predicting on rasters

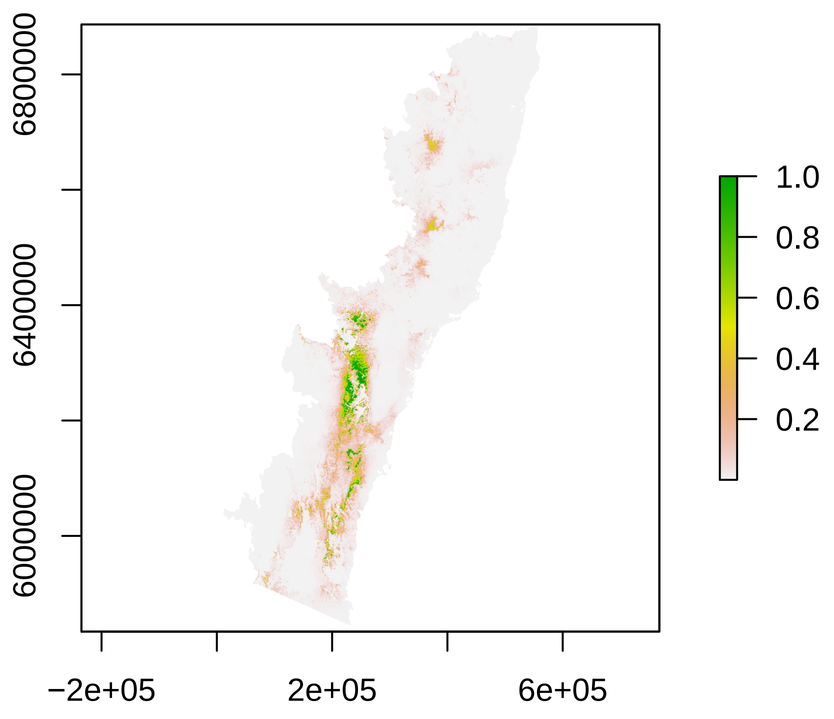
To predict this model on rasters, you need a `RasterStack` with all the covariates used in model fitting. The names of the covariates in the `RasterStack` must be exactly the same. This is done through the `raster` package (already loaded with `dismo` pcakage).

There are different options for scaling the raw output of `maxnet` including the **link**, **exponential**, **logistic** and recently introduced **cloglog**. For more information about these option please read Phillips et al. (2017). To show a progress bar use `progress = "text"`.

```r
tmp <- Sys.time()
max_pred <- raster::predict(object = nsw_stack, model = mxnet, type = "cloglog")
Sys.time() - tmp
```

```
## Time difference of 46.05229 secs
```

```r
# plot the prediction
plot(max_pred)
```



Bias Correction

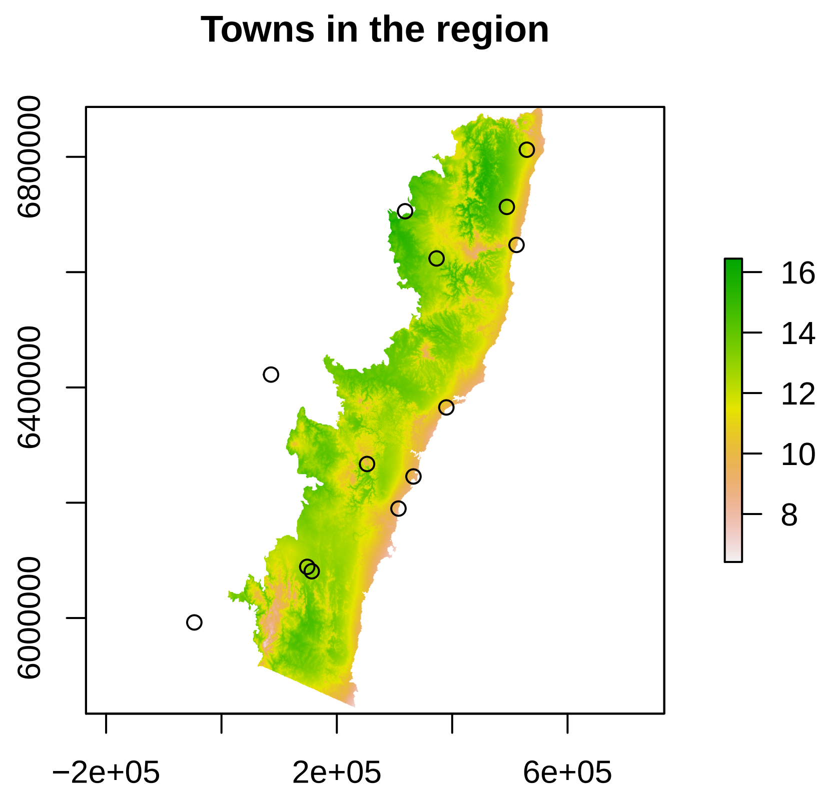
Creating a map of distance to the towns to correct for biases in the data.

```r
library(sf)

# the border shapefile
border <- st_read("data/border/ibraone.shp", quiet = TRUE)
# reading the town point data
towns <- st_read("data/towns/towns.shp")
```

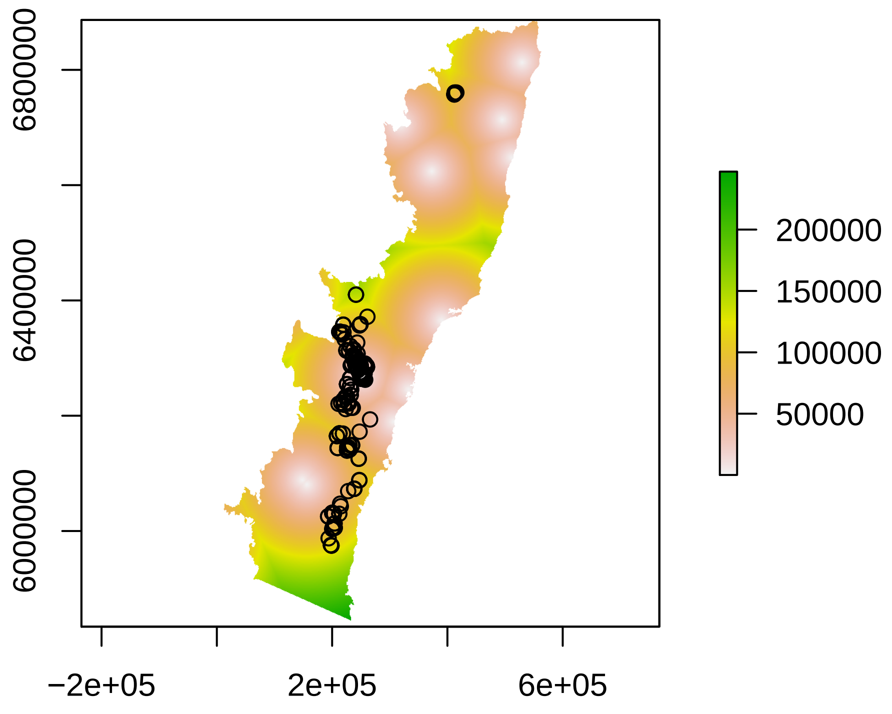
```
## Reading layer `towns' from data source `/home/unimelb.edu.au/iflint/R/libraries/SDM_with_PPM_tutorial
## Simple feature collection with 15 features and 92 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -206242.7 ymin: 5784013 xmax: 529376 ymax: 6963421
## epsg (SRID):    32756
## proj4string:    +proj=utm +zone=56 +south +datum=WGS84 +units=m +no_defs
```

```r
plot(nsw_stack[[1]], main = "Towns in the region")
plot(st_geometry(towns), add = TRUE)
```



```
distmap <- distanceFromPoints(nsw_stack[[1]], towns)  # this may take some time
distmap <- mask(distmap, mask = border)  # crop base on the region

plot(distmap)
points(occ$X, occ$Y)
```



Modelling with point processes

The `spatstat` package is the most widely used to work with point processes. Covariates are usually specified in their image objects `spatstat::im`. Internally, this is represented as a large pixel matrix, so conversion from rasters and other image objects is usually straightforward.

```r
library(spatstat)

covariates <- lapply(as.list(nsw_stack), function(element) maptools::as.im.RasterLayer(element))
names(covariates) <- names(nsw_stack)
```

Spatstat also needs to be told what the observation region is. The required object type is `spatstat::owin`. Common ways to construct an `owin` is to either take a fixed rectangle, i.e. `window <- owin(c(0, 100), c(0, 100))`, or to use an existing covariate or raster to construct the window. The latter technique is what we will use here.

Although it would be possible to do `window <- spatstat::as.owin(covariates[[1]])`, it will be easier to work on a window with a lower resolution, as shown next.

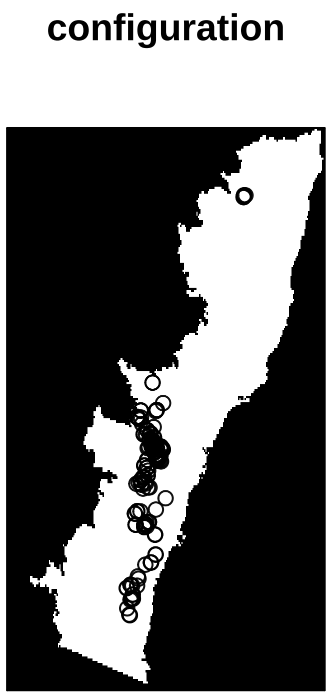
```r
window <- spatstat::as.owin(as.mask(covariates[[1]], dimyx = c(250, 250)))
```

Locations of individuals are represented via a point pattern object `spatstat::ppp`, and consist in coordinates along with a window in which the species has been observed.

```
configuration <- spatstat::ppp(x = occ$X, y = occ$Y, window = window)
```

Point patterns can easily be plotted.

```
plot(configuration)
```

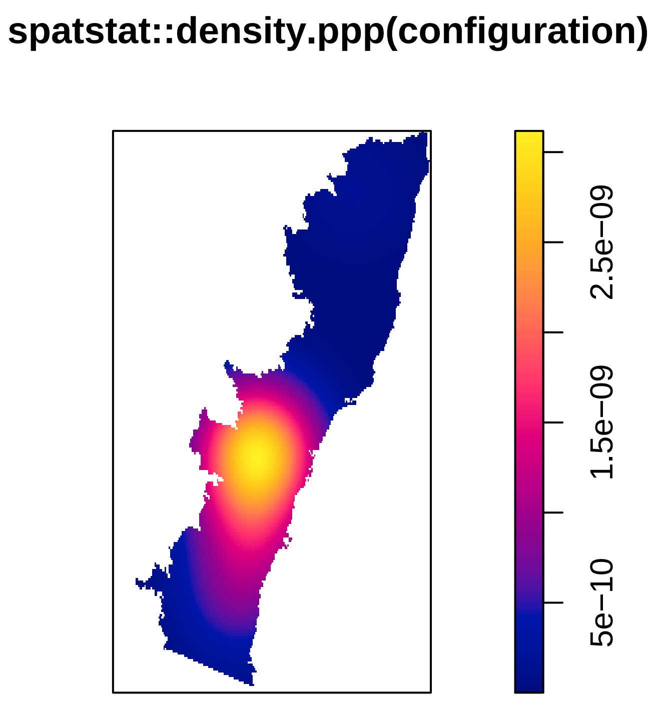


It is usually a good idea to start by a "static'' analysis of the point pattern, without yet involving covariates.

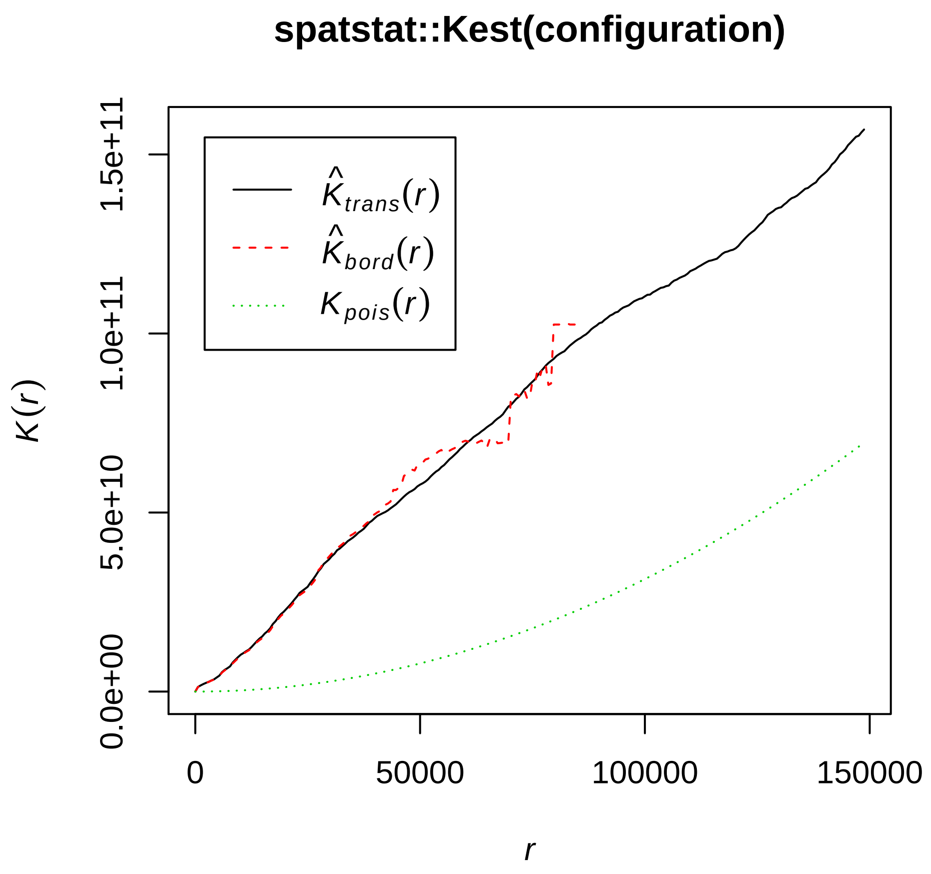
```
summary(configuration)
```

```
## Planar point pattern:  183 points
## Average intensity 9.658296e-10 points per square unit
##
## Coordinates are given to 1 decimal place
## i.e. rounded to the nearest multiple of 0.1 units
##
## binary image mask
## 250 x 250 pixel array (ny, nx)
## pixel size: 2380 by 4210 units
## enclosing rectangle: [-31118, 563882] x [5834100, 6886600] units
##                        (595000 x 1052000 units)
## Window area = 1.89474e+11 square units
## Fraction of frame area: 0.303
```

```r
plot(spatstat::density.ppp(configuration))
```



```r
plot(spatstat::Kest(configuration))
```



```
# The line below takes 3 min to execute and is not crucial to the analysis.
# plot(spatstat::envelope(configuration,Kest))
```

Doing inference on the point pattern is just as easy as setting up a `glm` regression. Start by writing the formula, essentially `formula <- "configuration ~ covariates`

```
formula <- paste0("configuration ~ ", paste0(names(covariates), collapse = " + "))
print(formula)
```

```
## [1] "configuration ~ bc02 + bc04 + bc05 + bc12 + bc14 + bc21 + bc32 + bc33 + mvbf + rjja + rsea + ru
```

The fitting function (analogue of `glm`) is `spatstat::ppm` and is used as follows.

```
fit <- spatstat::ppm(as.formula(formula), covariates = covariates)
```

The fitted regression is manipulated in the same way as a `glm` fit is, so for example you can have a look at the summary

```
summary(fit)
```

```
## Point process model
## Fitting method: maximum likelihood (Berman-Turner approximation)
## Model was fitted using glm()
## Algorithm converged
## Call:
## ppm.formula(Q = as.formula(formula), covariates = covariates)
## Edge correction: "border"
##  [border correction distance r = 0 ]
## -----------------------------------------------------------------------------
## Quadrature scheme (Berman-Turner) = data + dummy + weights
##
## Data pattern:
## Planar point pattern:  183 points
## Average intensity 9.66e-10 points per square unit
## binary image mask
## 250 x 250 pixel array (ny, nx)
## pixel size: 2380 by 4210 units
## enclosing rectangle: [-31118, 563882] x [5834100, 6886600] units
##                      (595000 x 1052000 units)
## Window area = 1.89474e+11 square units
## Fraction of frame area: 0.303
##
## Dummy quadrature points:
##      50 x 50 grid of dummy points, plus 4 corner points
##      dummy spacing: 11900 x 21050 units
##
## Original dummy parameters: =
## Planar point pattern:  873 points
## Average intensity 4.61e-09 points per square unit
## binary image mask
## 250 x 250 pixel array (ny, nx)
## pixel size: 2380 by 4210 units
## enclosing rectangle: [-31118, 563882] x [5834100, 6886600] units
##                      (595000 x 1052000 units)
## Window area = 1.89474e+11 square units
## Fraction of frame area: 0.303
## Quadrature weights:
##      (counting weights based on 50 x 50 array of rectangular tiles)
## All weights:
##  range: [1e+07, 2.5e+08] total: 1.89e+11
## Weights on data points:
##  range: [14700000, 1.25e+08] total: 7.58e+09
## Weights on dummy points:
##  range: [1e+07, 2.5e+08] total: 1.82e+11
## -----------------------------------------------------------------------------
## FITTED MODEL:
##
## Nonstationary Poisson process
##
## ---- Intensity: ----
##
## Log intensity: ~bc02 + bc04 + bc05 + bc12 + bc14 + bc21 + bc32 + bc33 + mvbf +
## rjja + rsea + rugg + twmd + twmx
## Model depends on external covariates 'bc02', 'bc04', 'bc05', 'bc12', 'bc14',
## 'bc21', 'bc32', 'bc33', 'mvbf', 'rjja', 'rsea', 'rugg', 'twmd' and 'twmx'
## Covariates provided:
##  bc02: im
##  bc04: im
##  bc05: im
##  bc12: im
##  bc14: im
##  bc21: im
##  bc32: im
##  bc33: im
##  mvbf: im
##  rjja: im
##  rsea: im
##  rugg: im
##  twmd: im
##  twmx: im
##
## Fitted trend coefficients:
##   (Intercept)          bc02          bc04          bc05          bc12
## -4.974334e+01 -3.826403e-01  1.755348e+00 -3.857281e-01  4.603596e-03
##          bc14          bc21          bc32          bc33          mvbf
##  6.170793e-01  1.633662e+00  6.139074e-01  6.231206e+00 -1.392892e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.124539e-02 -2.284542e+00  1.331625e-02 -3.812760e-04 -1.619788e-05
##
##                  Estimate         S.E.       CI95.lo       CI95.hi Ztest
## (Intercept) -4.974334e+01 9.827155e+00 -6.900421e+01 -3.048247e+01   ***
## bc02        -3.826403e-01 2.203367e-01 -8.144923e-01  4.921160e-02
## bc04         1.755348e+00 1.402199e+00 -9.929123e-01  4.503608e+00
## bc05        -3.857281e-01 8.821139e-02 -5.586192e-01 -2.128369e-01   ***
## bc12         4.603596e-03 1.833516e-03  1.009971e-03  8.197220e-03     *
## bc14         6.170793e-01 9.842179e-02  4.241761e-01  8.099824e-01   ***
## bc21         1.633662e+00 3.545120e-01  9.388310e-01  2.328493e+00   ***
## bc32         6.139074e-01 1.698929e+00 -2.715933e+00  3.943748e+00
## bc33         6.231206e+00 2.234334e+00  1.851991e+00  1.061042e+01    **
## mvbf        -1.392892e-01 1.526370e-01 -4.384523e-01  1.598739e-01
## rjja        -6.124539e-02 1.310312e-02 -8.692703e-02 -3.556375e-02   ***
## rsea        -2.284542e+00 1.674026e+00 -5.565572e+00  9.964880e-01
## rugg         1.331625e-02 2.017299e-03  9.362419e-03  1.727009e-02   ***
## twmd        -3.812760e-04 1.262400e-04 -6.287018e-04 -1.338501e-04    **
## twmx        -1.619788e-05 4.414832e-05 -1.027270e-04  7.033124e-05
##                   Zval
## (Intercept) -5.0618253
## bc02        -1.7366167
## bc04         1.2518533
## bc05        -4.3727690
## bc12         2.5108027
## bc14         6.2697424
## bc21         4.6081986
## bc32         0.3613496
## bc33         2.7888420
## mvbf        -0.9125519
## rjja        -4.6741087
## rsea        -1.3646995
## rugg         6.6010307
## twmd        -3.0202469
## twmx        -0.3668968
##
## ----------- gory details -----
##
```

```
## Fitted regular parameters (theta):
##   (Intercept)          bc02          bc04          bc05          bc12
## -4.974334e+01 -3.826403e-01  1.755348e+00 -3.857281e-01  4.603596e-03
##          bc14          bc21          bc32          bc33          mvbf
##  6.170793e-01  1.633662e+00  6.139074e-01  6.231206e+00 -1.392892e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.124539e-02 -2.284542e+00  1.331625e-02 -3.812760e-04 -1.619788e-05
##
## Fitted exp(theta):
##  (Intercept)         bc02         bc04         bc05         bc12         bc14
## 2.493102e-22 6.820582e-01 5.785460e+00 6.799554e-01 1.004614e+00 1.853507e+00
##         bc21         bc32         bc33         mvbf         rjja         rsea
## 5.122598e+00 1.847637e+00 5.083681e+02 8.699764e-01 9.405924e-01 1.018207e-01
##         rugg         twmd         twmx
## 1.013405e+00 9.996188e-01 9.999838e-01
## Problem:
##  Values of the covariates 'bc02', 'bc04', 'bc05', 'bc12', 'bc14', 'bc21', 'bc32', 'bc33', 'mvbf', 'r
```

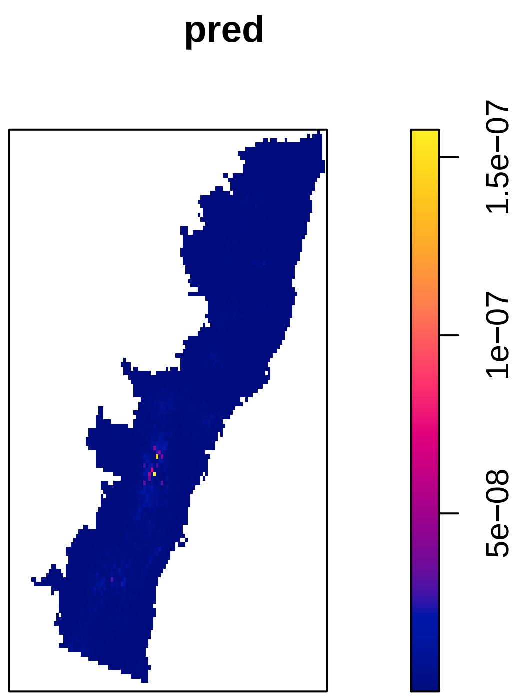
or do an ANOVA.

```
formula_without_bc04 <- paste0("configuration ~ ", paste0(names(covariates)[-2],
    collapse = " + "))
fit_without_bc04 <- spatstat::ppm(as.formula(formula_without_bc04), covariates = covariates)
anova(fit, fit_without_bc04)
```

```
## Analysis of Deviance Table
##
## Model 1: ~bc02 + bc04 + bc05 + bc12 + bc14 + bc21 + bc32 + bc33 + mvbf + rjja + rsea + rugg + twmd +
## Model 2: ~bc02 + bc05 + bc12 + bc14 + bc21 + bc32 + bc33 + mvbf + rjja + rsea + rugg + twmd + twmx
##   Npar Df Deviance
## 1   15
## 2   14 -1   -1.562
```

To look at the predicted intensity, you use the `spatstat::predict.ppm` function.

```
pred <- spatstat::predict.ppm(fit, covariates = covariates)
plot(pred)
```



Spatstat can handle many different types of correlation structures between individuals of the species. You would usually supply an `interaction` parameter to `spatstat::ppm`. However, initial analysis suggested attraction between the individuals, in which case a doubly-stochastic (Cox) point process is more appropriate. Fitting such point processes uses another function, as shown below.

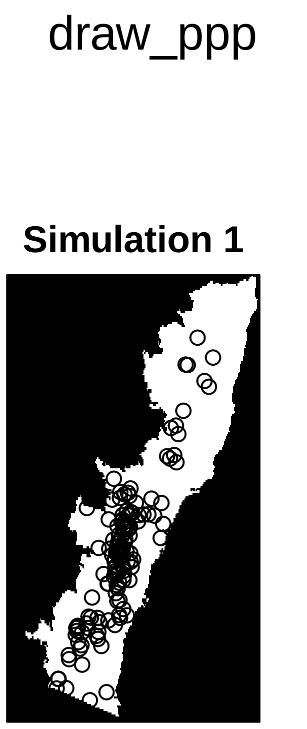
```
fit_cox <- spatstat::kppm(as.formula(formula), covariates = covariates, clusters = "LGCP")
summary(fit_cox)
```

```
## Inhomogeneous Cox point process model
## Fitted to point pattern dataset 'configuration'
## Fitted by minimum contrast
##  Summary statistic: inhomogeneous K-function
## Minimum contrast fit (object of class "minconfit")
## Model: Log-Gaussian Cox process
##   Covariance model: exponential
## Fitted by matching theoretical K function to configuration
##
## Internal parameters fitted by minimum contrast ($par):
##     sigma2      alpha
##   13.83409 1097.89958
##
## Fitted covariance parameters:
##        var      scale
##   13.83409 1097.89958
## Fitted mean of log of random intensity:  [pixel image]
##
## Converged successfully after 203 function evaluations
##
## Starting values of parameters:
##   sigma2    alpha
##    1.000 3358.596
## Domain of integration: [ 0 , 148800 ]
## Exponents: p= 2, q= 0.25
##
## ----------- TREND MODEL -----
## Point process model
## Fitting method: maximum likelihood (Berman-Turner approximation)
## Model was fitted using glm()
## Algorithm converged
## Call:
## ppm.ppp(Q = X, trend = trend, rename.intercept = FALSE, covariates = covariates,
##     covfunargs = covfunargs, use.gam = use.gam, forcefit = TRUE,
##     nd = nd, eps = eps)
## Edge correction: "border"
##  [border correction distance r = 0 ]
## --------------------------------------------------------------------------------
## Quadrature scheme (Berman-Turner) = data + dummy + weights
##
## Data pattern:
## Planar point pattern:  183 points
## Average intensity 9.66e-10 points per square unit
## binary image mask
## 250 x 250 pixel array (ny, nx)
## pixel size: 2380 by 4210 units
## enclosing rectangle: [-31118, 563882] x [5834100, 6886600] units
##                      (595000 x 1052000 units)
## Window area = 1.89474e+11 square units
## Fraction of frame area: 0.303
##
## Dummy quadrature points:
##      50 x 50 grid of dummy points, plus 4 corner points
##      dummy spacing: 11900 x 21050 units
##
## Original dummy parameters: =
## Planar point pattern:  873 points
## Average intensity 4.61e-09 points per square unit
## binary image mask
## 250 x 250 pixel array (ny, nx)
## pixel size: 2380 by 4210 units
## enclosing rectangle: [-31118, 563882] x [5834100, 6886600] units
##                      (595000 x 1052000 units)
## Window area = 1.89474e+11 square units
## Fraction of frame area: 0.303
## Quadrature weights:
##      (counting weights based on 50 x 50 array of rectangular tiles)
## All weights:
##  range: [1e+07, 2.5e+08] total: 1.89e+11
## Weights on data points:
```

```
##  range: [14700000, 1.25e+08] total: 7.58e+09
## Weights on dummy points:
##  range: [1e+07, 2.5e+08] total: 1.82e+11
## --------------------------------------------------------------------------------
## FITTED MODEL:
##
## Nonstationary Poisson process
##
## ---- Intensity: ----
##
## Log intensity: ~bc02 + bc04 + bc05 + bc12 + bc14 + bc21 + bc32 + bc33 + mvbf +
## rjja + rsea + rugg + twmd + twmx
## Model depends on external covariates 'bc02', 'bc04', 'bc05', 'bc12', 'bc14',
## 'bc21', 'bc32', 'bc33', 'mvbf', 'rjja', 'rsea', 'rugg', 'twmd' and 'twmx'
## Covariates provided:
##  bc02: im
##  bc04: im
##  bc05: im
##  bc12: im
##  bc14: im
##  bc21: im
##  bc32: im
##  bc33: im
##  mvbf: im
##  rjja: im
##  rsea: im
##  rugg: im
##  twmd: im
##  twmx: im
##
## Fitted trend coefficients:
##   (Intercept)          bc02          bc04          bc05          bc12
## -4.974334e+01 -3.826403e-01  1.755348e+00 -3.857281e-01  4.603596e-03
##          bc14          bc21          bc32          bc33          mvbf
##  6.170793e-01  1.633662e+00  6.139074e-01  6.231206e+00 -1.392892e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.124539e-02 -2.284542e+00  1.331625e-02 -3.812760e-04 -1.619788e-05
##
##                  Estimate         S.E.       CI95.lo       CI95.hi Ztest
## (Intercept) -4.974334e+01 9.827155e+00 -6.900421e+01 -3.048247e+01   ***
## bc02        -3.826403e-01 2.203367e-01 -8.144923e-01  4.921160e-02
## bc04         1.755348e+00 1.402199e+00 -9.929123e-01  4.503608e+00
## bc05        -3.857281e-01 8.821139e-02 -5.586192e-01 -2.128369e-01   ***
## bc12         4.603596e-03 1.833516e-03  1.009971e-03  8.197220e-03     *
## bc14         6.170793e-01 9.842179e-02  4.241761e-01  8.099824e-01   ***
## bc21         1.633662e+00 3.545120e-01  9.388310e-01  2.328493e+00   ***
## bc32         6.139074e-01 1.698929e+00 -2.715933e+00  3.943748e+00
## bc33         6.231206e+00 2.234334e+00  1.851991e+00  1.061042e+01    **
## mvbf        -1.392892e-01 1.526370e-01 -4.384523e-01  1.598739e-01
## rjja        -6.124539e-02 1.310312e-02 -8.692703e-02 -3.556375e-02   ***
## rsea        -2.284542e+00 1.674026e+00 -5.565572e+00  9.964880e-01
## rugg         1.331625e-02 2.017299e-03  9.362419e-03  1.727009e-02   ***
## twmd        -3.812760e-04 1.262400e-04 -6.287018e-04 -1.338501e-04    **
## twmx        -1.619788e-05 4.414832e-05 -1.027270e-04  7.033124e-05
##                   Zval
## (Intercept) -5.0618253
## bc02        -1.7366167
## bc04         1.2518533
## bc05        -4.3727690
## bc12         2.5108027
## bc14         6.2697424
## bc21         4.6081986
## bc32         0.3613496
## bc33         2.7888420
## mvbf        -0.9125519
## rjja        -4.6741087
## rsea        -1.3646995
## rugg         6.6010307
## twmd        -3.0202469
## twmx        -0.3668968
##
## ----------- gory details -----
##
## Fitted regular parameters (theta):
##   (Intercept)          bc02          bc04          bc05          bc12
## -4.974334e+01 -3.826403e-01  1.755348e+00 -3.857281e-01  4.603596e-03
##          bc14          bc21          bc32          bc33          mvbf
##  6.170793e-01  1.633662e+00  6.139074e-01  6.231206e+00 -1.392892e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.124539e-02 -2.284542e+00  1.331625e-02 -3.812760e-04 -1.619788e-05
##
## Fitted exp(theta):
##  (Intercept)         bc02         bc04         bc05         bc12         bc14
## 2.493102e-22 6.820582e-01 5.785460e+00 6.799554e-01 1.004614e+00 1.853507e+00
##         bc21         bc32         bc33         mvbf         rjja         rsea
## 5.122598e+00 1.847637e+00 5.083681e+02 8.699764e-01 9.405924e-01 1.018207e-01
##         rugg         twmd         twmx
## 1.013405e+00 9.996188e-01 9.999838e-01
## Problem:
##  Values of the covariates 'bc02', 'bc04', 'bc05', 'bc12', 'bc14', 'bc21', 'bc32', 'bc33', 'mvbf', 'r
##
##
## ----------- COX MODEL -----------
## Model: log-Gaussian Cox process
##
##  Covariance model: exponential
## Fitted covariance parameters:
##        var      scale
##   13.83409 1097.89958
## Fitted mean of log of random intensity: [pixel image]
##
## Final standard error and CI
## (allowing for correlation of Cox process):
##                  Estimate S.E. CI95.lo CI95.hi Ztest Zval
## (Intercept) -4.974334e+01   NA      NA      NA  <NA>   NA
## bc02        -3.826403e-01   NA      NA      NA  <NA>   NA
## bc04         1.755348e+00   NA      NA      NA  <NA>   NA
## bc05        -3.857281e-01   NA      NA      NA  <NA>   NA
## bc12          4.603596e-03   NA       NA       NA  <NA>   NA
## bc14          6.170793e-01   NA       NA       NA  <NA>   NA
## bc21          1.633662e+00   NA       NA       NA  <NA>   NA
## bc32          6.139074e-01   NA       NA       NA  <NA>   NA
## bc33          6.231206e+00   NA       NA       NA  <NA>   NA
## mvbf         -1.392892e-01   NA       NA       NA  <NA>   NA
## rjja         -6.124539e-02   NA       NA       NA  <NA>   NA
## rsea         -2.284542e+00   NA       NA       NA  <NA>   NA
## rugg          1.331625e-02   NA       NA       NA  <NA>   NA
## twmd         -3.812760e-04   NA       NA       NA  <NA>   NA
## twmx         -1.619788e-05   NA       NA       NA  <NA>   NA
```

A nice way to appreciate the difference in the underlying model is to draw from the fitted distribution. This can easily be done for the fitted Poisson point process.

```r
draw_ppp <- spatstat::simulate.ppm(fit)
plot(draw_ppp)
```

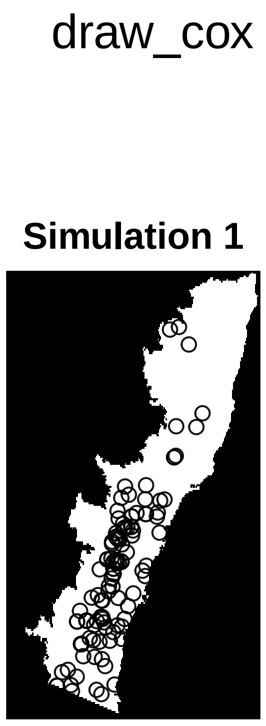


Drawing from a Cox point process requires you to use another library, but it essentially works in the same way.

```r
library(RandomFields)
library(RandomFieldsUtils)

draw_cox <- spatstat::simulate.kppm(fit_cox)
```

```r
plot(draw_cox)
```



Making goodness-of-fit tests is straightforward, we refer in particular to the functions `spatstat::quadrat.test`, `spatstat::cdf.test`, `spatstat::dclf.test` and `spatstat::mad.test`. A lot of these functions rely on multiple simulations of the point process, which is going to be exeedingly slow for the Cox process. Instead, we show what a goodness-of-fit test looks like with a simple fit with a Poisson point process.

```r
dclf.test(fit)
```

```
## Generating 99 simulated realisations of fitted Poisson model  ...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 2
## 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
## 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,  99.
##
## Done.
```

```
##
##  Diggle-Cressie-Loosmore-Ford test of fitted Poisson model
##  Monte Carlo test based on 99 simulations
##  Summary function: K(r)
##  Reference function: sample mean
##  Alternative: two.sided
##  Interval of distance values: [0, 148750]
##  Test statistic: Integral of squared absolute deviation
##  Deviation = leave-one-out
##
## data:  fit
## u = 1.149e+26, rank = 1, p-value = 0.01
```

Comparing the results