Species distribution modelling with Poisson point processes

Ian Flint and Roozbeh Valavi

2020-02-19

Species distribution modelling

Species distribution modelling is...

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

Envrionmental data

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

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

# read the raster layers as a raster stack
nsw_stack <- stack(vars)
# set the coordinate system
crs(nsw_stack) <- CRS("+init=epsg:4283")
# you can plot them by: plot(nsw_stack)
```

Species data

The species data comes form Fithian and others (2015).

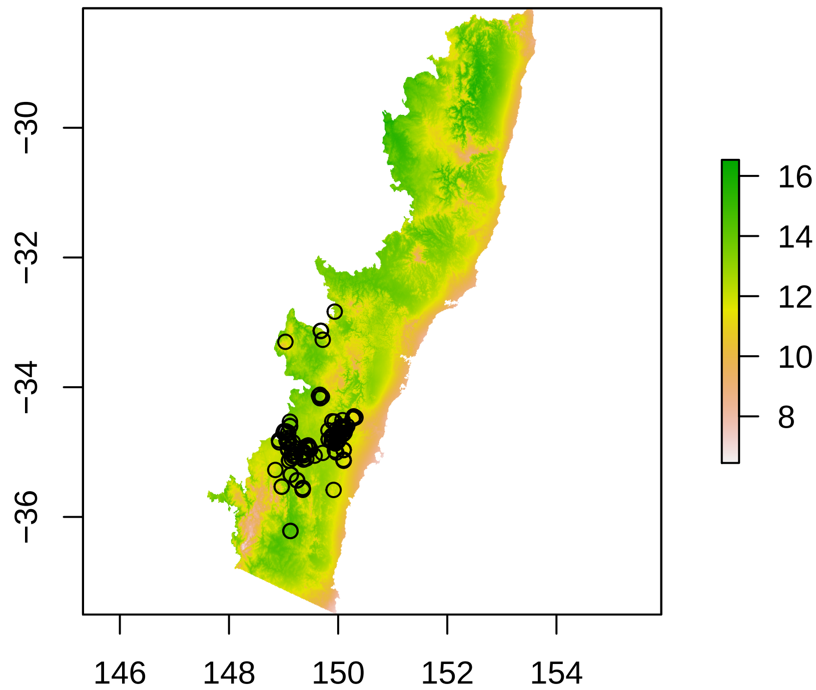
All the species starts with `euca` ...

```
# load Fithian et al (2015) species data
load("data/moddat.RData")

# show few rows and column in the species dataset
eucacine_moddat[1:5, 1:8]
```

```
##   id     long       lat year accuracy  longlat   cells     bc01
## 1  1 150.0862 -34.60553 2000      100 15008622 6279811 13.63983
## 2  2 150.2854 -34.44967 2000      100 15028535 6126927 13.24499
## 3  3 150.0125 -34.80654 2000      100 15001250 6474021 13.71813
## 4  4 150.0997 -34.96775 2000      100 15009972 6631876 13.09869
## 5  5 149.8980 -34.85114 2007      100 14989800 6517680 13.29461
```

```
# plot the sapecies on the raster map
plot(nsw_stack[[1]])
points(eucacine_moddat$long, eucacine_moddat$lat)
```



Modelling with Maxent

Here we use `maxnet` package, the open-source version of `Maxent` in R.

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

Sampling random background from the landscape…

For the purpose of this tutorial, we random

```
# generate some random samples
rnd <- randomPoints(nsw_stack, n = 10000)

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

```
##           bc02     bc04     bc05      bc12      bc14     bc21      bc32
## [1,] 11.176427 1.498268 26.36980 1049.9161 12.296509 25.40611 0.9889812
## [2,] 13.190133 1.873963 29.42034  648.8047  9.575684 26.51981 0.9201095
## [3,]  9.611765 1.307230 26.66947 1130.4515 12.088196 25.04248 0.9988450
## [4,] 13.572127 1.565184 28.47215  928.3860  5.858215 23.73382 0.7965168
## [5,] 11.328123 1.244282 28.40651 1259.3962  7.274658 23.55162 1.0000000
## [6,] 12.124986 1.595333 25.29785  857.4468  7.839417 24.26004 0.8301125
##           bc33 mvbf     rjja        rsea       rugg twmd  twmx
## [1,] 0.4369141    0 233.4139  0.15485157  50.817688 6817 14676
## [2,] 0.2314613    3 177.0858 -0.07545944   8.477350 7550  9683
## [3,] 0.5069848    0 240.6921  0.15615730  22.085060 9325 12267
## [4,] 0.4714794    0 108.3237  1.27347505 133.038635 8102 10859
## [5,] 0.5486870    4 177.1310  0.53952032   1.624972 7611  8519
## [6,] 0.5685839    0 123.1467  0.95896024  29.266521 8911 13467
```

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

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

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

As the values in the presence data is already extracted, we just merge the data to make it ready for model fitting. Here, I make a `daata.frame` with only the covariates that is going to be used in model fitting.

```r
# subset the covariate in species data by raster
prcov <- eucacine_moddat[, names(nsw_stack)]

# now add them together
covariates <- rbind(prcov, background)
```

Now we can use this for model fitting... the parameters...

Here, we need to make a voctor of 1s and 0s (for presence and background points respectively). This shoudl 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(eucacine_moddat)),
               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 15.20242 secs
```

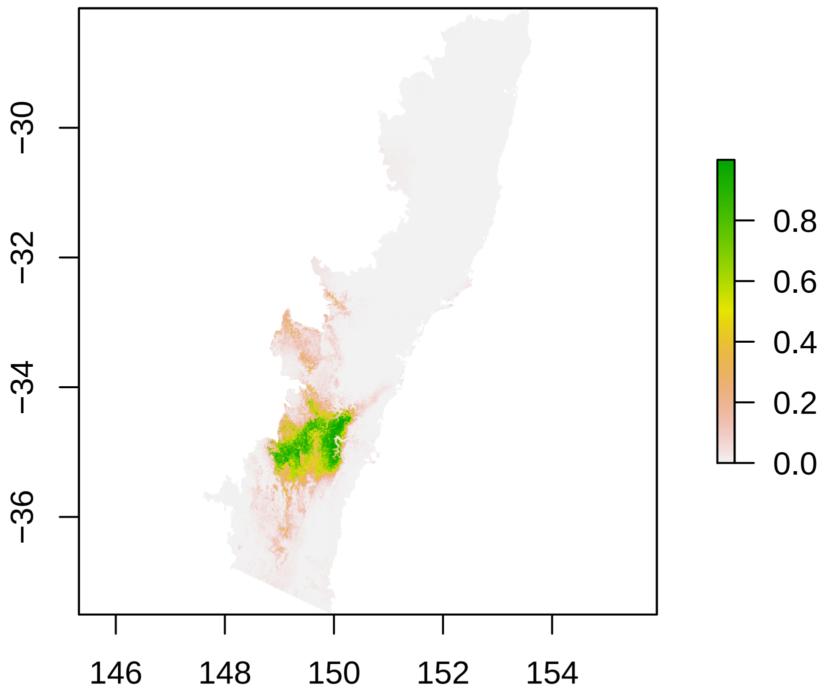
```r
# plot the fitted function
# plot(mxnet, type = "cloglog")
```

Pridicting 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 `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).

```r
max_pred <- raster::predict(object = nsw_stack, model = mxnet, type = "cloglog")
# plot the prediction
plot(max_pred)
```



Bias Correction

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

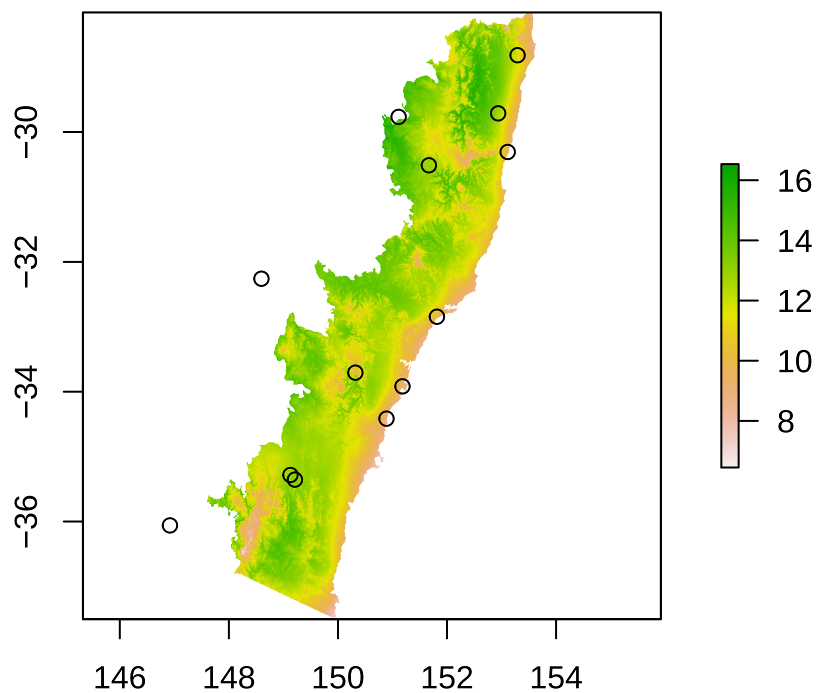
```r
library(sf)

# the border shapefile
border <- st_read("data/ibraone.shp", crs = 4283, quiet = TRUE)
# reading the town point data
towns <- st_read("data/towns/ecologist_towns.shp")
```

```
## Reading layer `ecologist_towns' from data source `/Users/rvalavi/Dropbox/MyProjects/SDM_with_PPM/data
## Simple feature collection with 15 features and 92 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 144.9731 ymin: -37.81809 xmax: 153.2931 ymax: -27.45309
## epsg (SRID):    NA
## proj4string:    NA
```

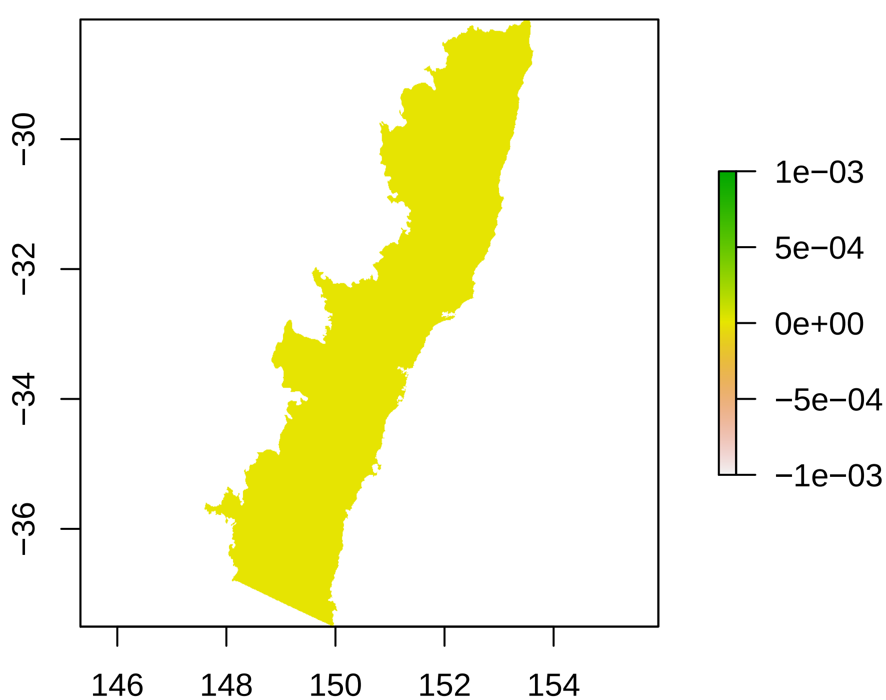
```r
plot(nsw_stack[[1]], main = "Towns in the region")
plot(st_geometry(towns), add = TRUE)
```

Towns in the region




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

plot(distmap)
```



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.

```
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.

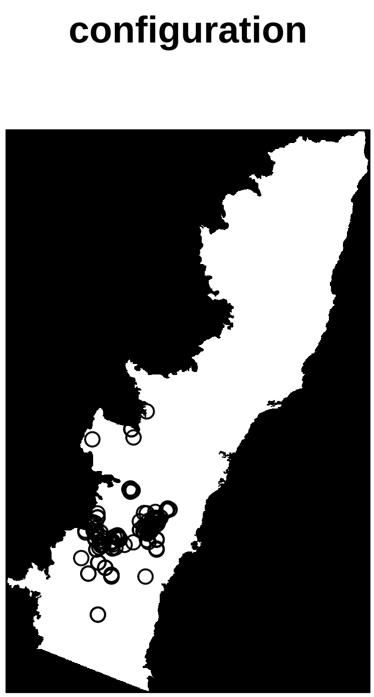
```
window <- spatstat::as.owin(as.mask(covariates[[1]], eps = 0.01))
```

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 = eucacine_moddat$long, y = eucacine_moddat$lat,
    window = window)
```

Point patterns can easily be plotted.

```
plot(configuration)
```



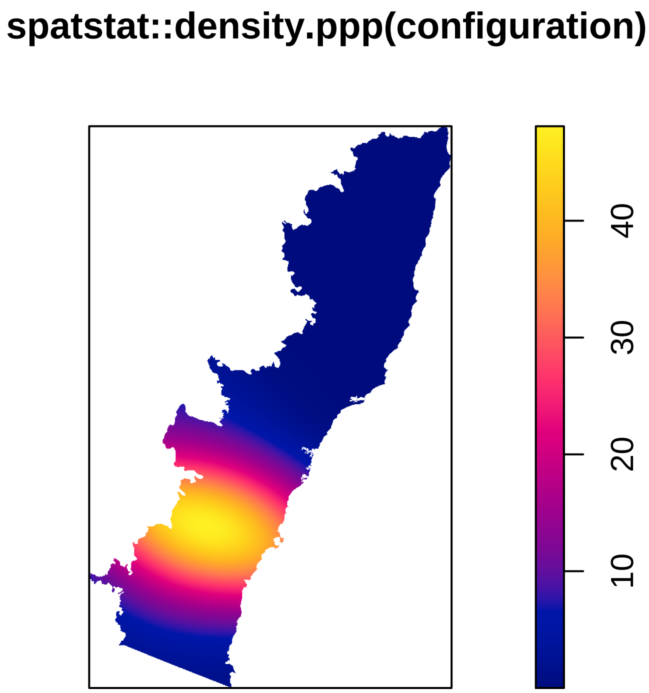
It is usually a good idea to start by a "static" analysis of the point pattern, without yet involving covariates.

```
summary(configuration)
```

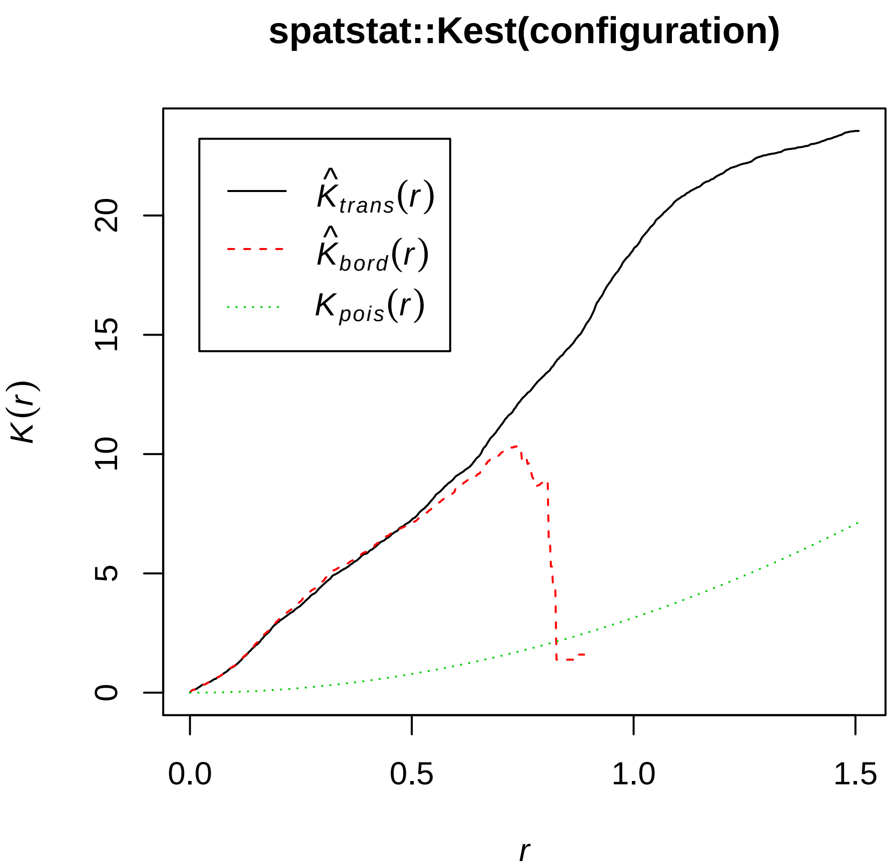
```
## Planar point pattern:  171 points
```

```
## Average intensity 9.396613 points per square unit
##
## Coordinates are given to 6 decimal places
##
## binary image mask
## 935 x 604 pixel array (ny, nx)
## pixel size: 0.00998 by 0.01 units
## enclosing rectangle: [147.6075, 153.6375] x [-37.505, -28.1575] units
## Window area = 18.198 square units
## Fraction of frame area: 0.323
```

```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:  171 points
## Average intensity 9.4 points per square unit
## binary image mask
## 935 x 604 pixel array (ny, nx)
## pixel size: 0.00998 by 0.01 units
## enclosing rectangle: [147.6075, 153.6375] x [-37.505, -28.1575] units
## Window area = 18.198 square units
## Fraction of frame area: 0.323
##
## Dummy quadrature points:
##      151 x 55 grid of dummy points, plus 4 corner points
##      dummy spacing: 0.03993377 x 0.16995455 units
##
## Original dummy parameters: =
## Planar point pattern:  2968 points
## Average intensity 163 points per square unit
## binary image mask
## 935 x 604 pixel array (ny, nx)
## pixel size: 0.00998 by 0.01 units
## enclosing rectangle: [147.6075, 153.6375] x [-37.505, -28.1575] units
## Window area = 18.198 square units
## Fraction of frame area: 0.323
## Quadrature weights:
##      (counting weights based on 151 x 55 array of rectangular tiles)
## All weights:
##  range: [9.98e-05, 0.00679]  total: 18.2
## Weights on data points:
##  range: [0.000566, 0.00339]  total: 0.279
## Weights on dummy points:
##  range: [9.98e-05, 0.00679]  total: 17.9
## -------------------------------------------------------------------------
## 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
## -7.007317e+01 -6.964933e-01  9.263520e+00 -7.473635e-01  2.870190e-02
##          bc14          bc21          bc32          bc33          mvbf
##  2.375400e-02  3.512655e+00 -6.009921e+00 -2.504986e+01 -1.294599e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.620218e-02 -4.544442e+00 -7.297494e-03 -1.635421e-04  3.656648e-05
##
##                  Estimate         S.E.       CI95.lo       CI95.hi Ztest
## (Intercept) -7.007317e+01 1.983616e+01 -1.089513e+02 -3.119500e+01   ***
## bc02        -6.964933e-01 2.587332e-01 -1.203601e+00 -1.893855e-01    **
## bc04         9.263520e+00 2.064876e+00  5.216438e+00  1.331060e+01   ***
## bc05        -7.473635e-01 1.179860e-01 -9.786119e-01 -5.161152e-01   ***
## bc12         2.870190e-02 3.646108e-03  2.155566e-02  3.584814e-02   ***
## bc14         2.375400e-02 1.162362e-01 -2.040648e-01  2.515729e-01
## bc21         3.512655e+00 6.292925e-01  2.279264e+00  4.746045e+00   ***
## bc32        -6.009921e+00 2.629596e+00 -1.116383e+01 -8.560080e-01     *
## bc33        -2.504986e+01 8.425420e+00 -4.156338e+01 -8.536344e+00    **
## mvbf        -1.294599e-01 5.720662e-02 -2.415828e-01 -1.733694e-02     *
## rjja        -6.620218e-02 1.318730e-02 -9.204881e-02 -4.035556e-02   ***
## rsea        -4.544442e+00 3.312762e+00 -1.103734e+01  1.948453e+00
## rugg        -7.297494e-03 6.019354e-03 -1.909521e-02  4.500224e-03
## twmd        -1.635421e-04 7.866809e-05 -3.177287e-04 -9.355480e-06     *
## twmx         3.656648e-05 2.975702e-05 -2.175621e-05  9.488917e-05
##                   Zval
## (Intercept) -3.5325971
## bc02        -2.6919362
## bc04         4.4862362
## bc05        -6.3343397
## bc12         7.8719275
## bc14         0.2043597
## bc21         5.5819106
## bc32        -2.2854923
## bc33        -2.9731295
## mvbf        -2.2630223
## rjja        -5.0201481
## rsea        -1.3717983
## rugg        -1.2123383
## twmd        -2.0788874
## twmx         1.2288353
##
## ----------- gory details -----
##
## Fitted regular parameters (theta):
##   (Intercept)          bc02          bc04          bc05          bc12
## -7.007317e+01 -6.964933e-01  9.263520e+00 -7.473635e-01  2.870190e-02
##          bc14          bc21          bc32          bc33          mvbf
##  2.375400e-02  3.512655e+00 -6.009921e+00 -2.504986e+01 -1.294599e-01
```

```
##          rjja          rsea          rugg          twmd          twmx
## -6.620218e-02 -4.544442e+00 -7.297494e-03 -1.635421e-04  3.656648e-05
##
## Fitted exp(theta):
##  (Intercept)          bc02          bc04          bc05          bc12
## 3.694962e-31 4.983297e-01 1.054619e+04 4.736136e-01 1.029118e+00
##         bc14          bc21          bc32          bc33          mvbf
## 1.024038e+00 3.353718e+01 2.454282e-03 1.321242e-11 8.785699e-01
##         rjja          rsea          rugg          twmd          twmx
## 9.359416e-01 1.062610e-02 9.927291e-01 9.998365e-01 1.000037e+00
## Problem:
##  Values of the covariates 'twmd', 'twmx' were NA or undefined at 0.19% (6 out of 3139) of the quadra
```

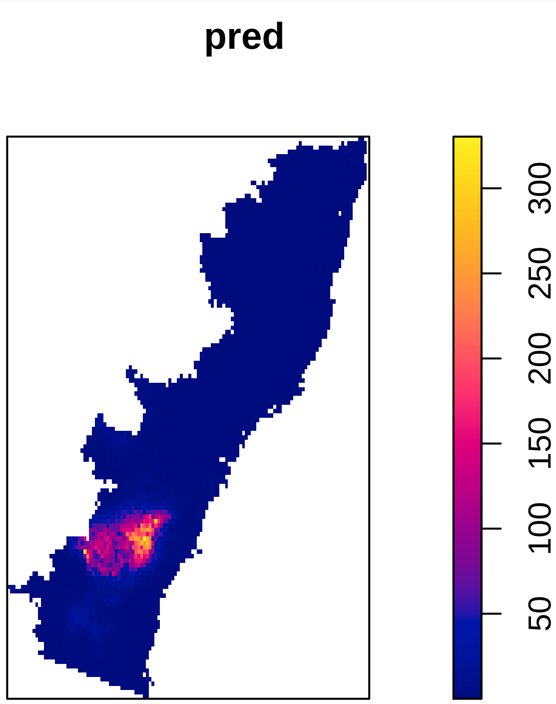
or do an ANOVA.

```
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  -18.955
```

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.

```r
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
## 6.56350580 0.03719749
##
## Fitted covariance parameters:
##        var      scale
## 6.56350580 0.03719749
## Fitted mean of log of random intensity:  [pixel image]
##
## Converged successfully after 125 function evaluations
##
## Starting values of parameters:
##     sigma2      alpha
## 1.00000000 0.05169264
## Domain of integration: [ 0 , 1.508 ]
## 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:  171 points
## Average intensity 9.4 points per square unit
## binary image mask
## 935 x 604 pixel array (ny, nx)
## pixel size: 0.00998 by 0.01 units
## enclosing rectangle: [147.6075, 153.6375] x [-37.505, -28.1575] units
## Window area = 18.198 square units
```

```
## Fraction of frame area: 0.323
##
## Dummy quadrature points:
##      151 x 55 grid of dummy points, plus 4 corner points
##      dummy spacing: 0.03993377 x 0.16995455 units
##
## Original dummy parameters: =
## Planar point pattern:  2968 points
## Average intensity 163 points per square unit
## binary image mask
## 935 x 604 pixel array (ny, nx)
## pixel size: 0.00998 by 0.01 units
## enclosing rectangle: [147.6075, 153.6375] x [-37.505, -28.1575] units
## Window area = 18.198 square units
## Fraction of frame area: 0.323
## Quadrature weights:
##      (counting weights based on 151 x 55 array of rectangular tiles)
## All weights:
##  range: [9.98e-05, 0.00679]  total: 18.2
## Weights on data points:
##  range: [0.000566, 0.00339]  total: 0.279
## Weights on dummy points:
##  range: [9.98e-05, 0.00679]  total: 17.9
## ----------------------------------------------------------------------------
## 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
## -7.007317e+01 -6.964933e-01  9.263520e+00 -7.473635e-01  2.870190e-02
```

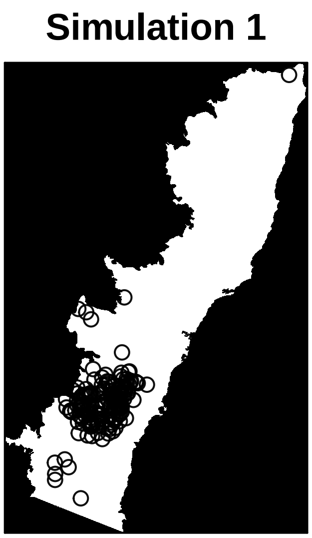
```
##          bc14          bc21          bc32          bc33          mvbf
##  2.375400e-02  3.512655e+00 -6.009921e+00 -2.504986e+01 -1.294599e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.620218e-02 -4.544442e+00 -7.297494e-03 -1.635421e-04  3.656648e-05
##
##                  Estimate         S.E.       CI95.lo       CI95.hi Ztest
## (Intercept) -7.007317e+01 1.983616e+01 -1.089513e+02 -3.119500e+01   ***
## bc02        -6.964933e-01 2.587332e-01 -1.203601e+00 -1.893855e-01    **
## bc04         9.263520e+00 2.064876e+00  5.216438e+00  1.331060e+01   ***
## bc05        -7.473635e-01 1.179860e-01 -9.786119e-01 -5.161152e-01   ***
## bc12         2.870190e-02 3.646108e-03  2.155566e-02  3.584814e-02   ***
## bc14         2.375400e-02 1.162362e-01 -2.040648e-01  2.515729e-01
## bc21         3.512655e+00 6.292925e-01  2.279264e+00  4.746045e+00   ***
## bc32        -6.009921e+00 2.629596e+00 -1.116383e+01 -8.560080e-01     *
## bc33        -2.504986e+01 8.425420e+00 -4.156338e+01 -8.536344e+00    **
## mvbf        -1.294599e-01 5.720662e-02 -2.415828e-01 -1.733694e-02     *
## rjja        -6.620218e-02 1.318730e-02 -9.204881e-02 -4.035556e-02   ***
## rsea        -4.544442e+00 3.312762e+00 -1.103734e+01  1.948453e+00
## rugg        -7.297494e-03 6.019354e-03 -1.909521e-02  4.500224e-03
## twmd        -1.635421e-04 7.866809e-05 -3.177287e-04 -9.355480e-06     *
## twmx         3.656648e-05 2.975702e-05 -2.175621e-05  9.488917e-05
##                   Zval
## (Intercept) -3.5325971
## bc02        -2.6919362
## bc04         4.4862362
## bc05        -6.3343397
## bc12         7.8719275
## bc14         0.2043597
## bc21         5.5819106
## bc32        -2.2854923
## bc33        -2.9731295
## mvbf        -2.2630223
## rjja        -5.0201481
## rsea        -1.3717983
## rugg        -1.2123383
## twmd        -2.0788874
## twmx         1.2288353
##
## ----------- gory details -----
##
## Fitted regular parameters (theta):
##   (Intercept)          bc02          bc04          bc05          bc12
## -7.007317e+01 -6.964933e-01  9.263520e+00 -7.473635e-01  2.870190e-02
##          bc14          bc21          bc32          bc33          mvbf
##  2.375400e-02  3.512655e+00 -6.009921e+00 -2.504986e+01 -1.294599e-01
##          rjja          rsea          rugg          twmd          twmx
## -6.620218e-02 -4.544442e+00 -7.297494e-03 -1.635421e-04  3.656648e-05
##
## Fitted exp(theta):
##  (Intercept)         bc02         bc04         bc05         bc12
## 3.694962e-31 4.983297e-01 1.054619e+04 4.736136e-01 1.029118e+00
##         bc14         bc21         bc32         bc33         mvbf
## 1.024038e+00 3.353718e+01 2.454282e-03 1.321242e-11 8.785699e-01
##         rjja         rsea         rugg         twmd         twmx
```

```
## 9.359416e-01 1.062610e-02 9.927291e-01 9.998365e-01 1.000037e+00
## Problem:
##  Values of the covariates 'twmd', 'twmx' were NA or undefined at 0.19% (6 out of 3139) of the quadra
##
##
## ----------- COX MODEL -----------
## Model: log-Gaussian Cox process
##
##  Covariance model: exponential
## Fitted covariance parameters:
##        var      scale
## 6.56350580 0.03719749
## 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) -7.007317e+01   NA      NA      NA  <NA>   NA
## bc02        -6.964933e-01   NA      NA      NA  <NA>   NA
## bc04         9.263520e+00   NA      NA      NA  <NA>   NA
## bc05        -7.473635e-01   NA      NA      NA  <NA>   NA
## bc12         2.870190e-02   NA      NA      NA  <NA>   NA
## bc14         2.375400e-02   NA      NA      NA  <NA>   NA
## bc21         3.512655e+00   NA      NA      NA  <NA>   NA
## bc32        -6.009921e+00   NA      NA      NA  <NA>   NA
## bc33        -2.504986e+01   NA      NA      NA  <NA>   NA
## mvbf        -1.294599e-01   NA      NA      NA  <NA>   NA
## rjja        -6.620218e-02   NA      NA      NA  <NA>   NA
## rsea        -4.544442e+00   NA      NA      NA  <NA>   NA
## rugg        -7.297494e-03   NA      NA      NA  <NA>   NA
## twmd        -1.635421e-04   NA      NA      NA  <NA>   NA
## twmx         3.656648e-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)
```

draw_ppp

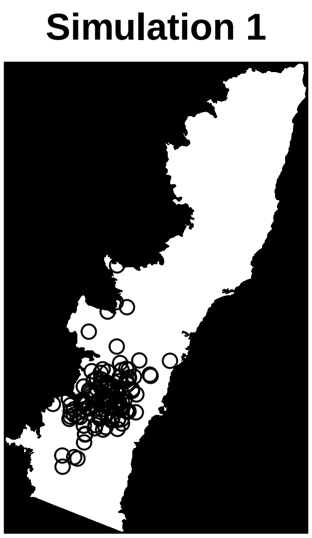


Drawing from a Cox point process requires you to use another library, but it essentially works in the same way.

```
library(RandomFields)
library(RandomFieldsUtils)

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

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,  [etd 3:24] 3,  [etd 3:15] 4,
##  [etd 3:15] 5,  [etd 3:11] 6,  [etd 3:09] 7,  [etd 3:06] 8,
##  [etd 3:03] 9,  [etd 3:02] 10,  [etd 3:05] 11,  [etd 3:02] 12,
##  [etd 2:59] 13,  [etd 2:57] 14,  [etd 2:55] 15,  [etd 2:53] 16,
##  [etd 2:50] 17,  [etd 2:48] 18,  [etd 2:45] 19,  [etd 2:43] 20,
##  [etd 2:41] 21,  [etd 2:38] 22,  [etd 2:36] 23,  [etd 2:34] 24,
##  [etd 2:31] 25,  [etd 2:29] 26,  [etd 2:27] 27,  [etd 2:24] 28,
##  [etd 2:22] 29,  [etd 2:20] 30,  [etd 2:18] 31,  [etd 2:16] 32,
##  [etd 2:13] 33,  [etd 2:11] 34,  [etd 2:09] 35,  [etd 2:07] 36,
##  [etd 2:05] 37,  [etd 2:03] 38,  [etd 2:01] 39,  [etd 1:59] 40,
##  [etd 1:57] 41,  [etd 1:55] 42,  [etd 1:53] 43,  [etd 1:50] 44,
##  [etd 1:48] 45,  [etd 1:46] 46,  [etd 1:44] 47,  [etd 1:42] 48,
##  [etd 1:40] 49,  [etd 1:38] 50,  [etd 1:36] 51,  [etd 1:34] 52,
##  [etd 1:32] 53,  [etd 1:30] 54,  [etd 1:28] 55,  [etd 1:26] 56,
##  [etd 1:24] 57,  [etd 1:22] 58,  [etd 1:20] 59,  [etd 1:18] 60,
##  [etd 1:16] 61,  [etd 1:14] 62,  [etd 1:12] 63,  [etd 1:10] 64,
##  [etd 1:08] 65,  [etd 1:06] 66,  [etd 1:05] 67,  [etd 1:03] 68,
##  [etd 1:01] 69,  [etd 59 sec] 70,  [etd 57 sec] 71,  [etd 55 sec] 72,
##  [etd 53 sec] 73,  [etd 51 sec] 74,  [etd 49 sec] 75,  [etd 47 sec] 76,
##  [etd 45 sec] 77,  [etd 43 sec] 78,  [etd 41 sec] 79,  [etd 39 sec] 80,
##  [etd 37 sec] 81,  [etd 35 sec] 82,  [etd 34 sec] 83,  [etd 32 sec] 84,
##  [etd 30 sec] 85,  [etd 28 sec] 86,  [etd 27 sec] 87,  [etd 25 sec] 88,
##  [etd 23 sec] 89,  [etd 21 sec] 90,  [etd 19 sec] 91,  [etd 17 sec] 92,
##  [etd 14 sec] 93,  [etd 12 sec] 94,  [etd 10 sec] 95,  [etd 8 sec] 96,
##  [etd 6 sec] 97,  [etd 4 sec] 98,  [etd 2 sec]  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, 1.5075]
##  Test statistic: Integral of squared absolute deviation
##  Deviation = leave-one-out
##
## data:  fit
## u = 2.2088, rank = 8, p-value = 0.08
```

Comparing the results