Tutorial 1: simplest point process model

Ian Flint and Roozbeh Valavi

2023-03-08

In this document, we show how to fit the simplest point process model namely, Poisson process model.

Setup and R libraries

```
library(spatstat) # fitting and exploring ppm models
library(terra) # working with raster data
```

```
source("R/helper_functions.R")
```

Simplest point process model

Species data

Reading species data....

Note that the spatial coordinate system is a metric CRS...

```
occurrences <- read.csv("data/species/occurrences.csv")
head(occurrences)
```

```
##           X        Y
## 1 1852257.4 -3260237
## 2 1851901.4 -3259418
## 3 1051853.6 -4090644
## 4  667895.4 -4333839
## 5 1853470.1 -3261289
## 6 1873356.5 -3343068
```

Environmental data

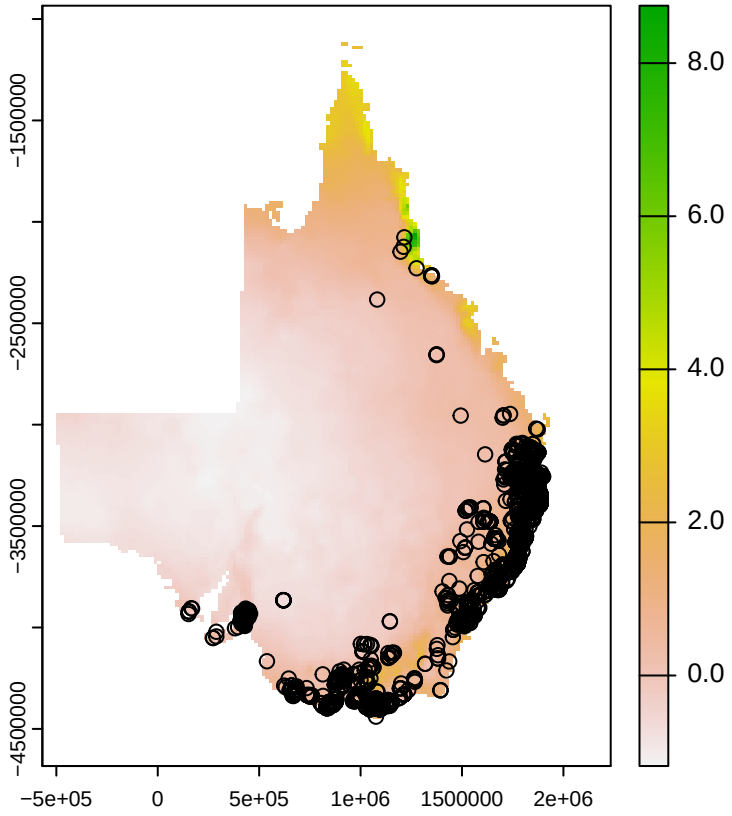
We read raster covariate here....

We need to centre and scale covariate to ...

```
# get the 19 bio-climatic variables
rasters <- list.files("data/rasters/", pattern = ".tif$", full.names = TRUE) |>
  terra::rast() |>
  terra::scale(center = TRUE, scale = TRUE)

# plot a few of them
```

```
plot(rasters[[1]])
points(occurrences)
```

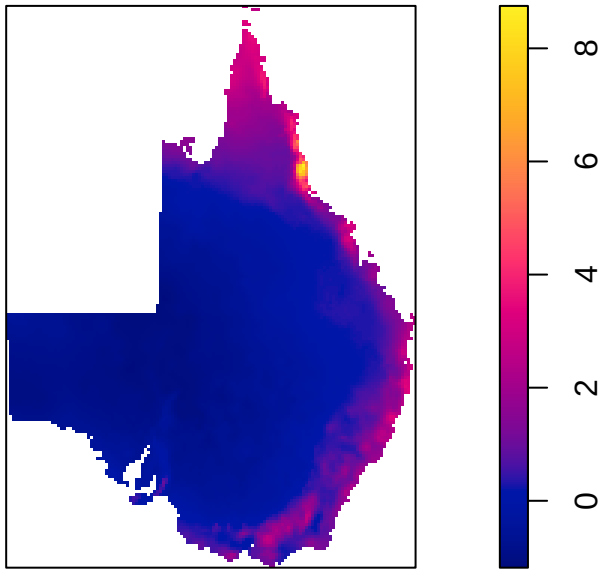


Converting everything to spatstat objects

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. In order to convert to the proper format, the `maptools` helper function is useful.

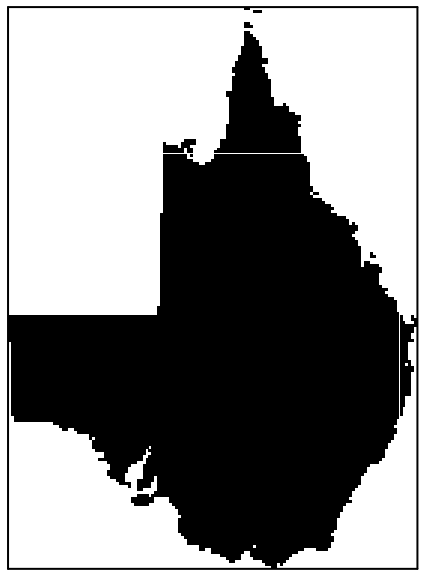
```
covariates <- lapply(rasters, spatstat.geom::as.im)
names(covariates) <- names(rasters)

plot(covariates[[1]], main = names(covariates)[1])
```



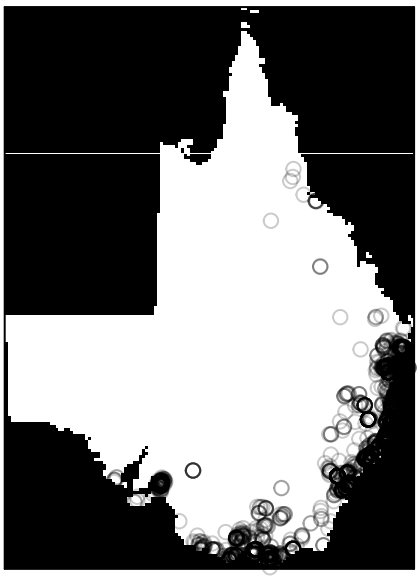
Next, when working with `spatstat` you need an observation window (region). It is a region in which the points are assumed to be drawn from. The probability of finding locations outside of this region is assumed to be zero. Common ways to construct an `spatstat::owin` is to either take a fixed rectangle, i.e. `window <- owin(c(0, 100), c(0, 100))`, or to use an existing covariate or raster to construct the window.

```
window <- spatstat.geom::as.owin(covariates[[1]])
plot(window)
```



Occurrences (or presence points) are `spatstat::ppp` objects, and basically you need a vector of x-values, a vector of y-values, and the observation window.

```
configuration <- spatstat.geom::ppp(x = occurrences$X,
                                    y = occurrences$Y,
                                    window = window)
plot(configuration)
```



Model fitting

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`. In the formula, we use `polynom` to allow the user to easily specify whether to use higher order polynomials in the model.

```
formula <- configuration ~ 1 + bio_4 + bio_5 + bio_12 + bio_15
```

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

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

Checking model

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 = formula, covariates = covariates)
## Edge correction: "border"
```

```
##  [border correction distance r = 0 ]
## --------------------------------------------------------------------------------
## Quadrature scheme (Berman-Turner) = data + dummy + weights
##
## Data pattern:
## Planar point pattern:  4958 points
## Average intensity 1.3e-09 points per square unit
## binary image mask
## 188 x 137 pixel array (ny, nx)
## pixel size: 17700 by 17700 units
## enclosing rectangle: [-497930.9, 1931687.7] x [-4449072, -1114996] units
##                      (2430000 x 3334000 units)
## Window area = 3.82539e+12 square units
## Fraction of frame area: 0.472
##
## Dummy quadrature points:
##      150 x 150 grid of dummy points, plus 4 corner points
##      dummy spacing: 16197.46 x 22227.17 units
##
## Original dummy parameters: =
## Planar point pattern:  10731 points
## Average intensity 2.81e-09 points per square unit
## binary image mask
## 188 x 137 pixel array (ny, nx)
## pixel size: 17700 by 17700 units
## enclosing rectangle: [-497930.9, 1931687.7] x [-4449072, -1114996] units
##                      (2430000 x 3334000 units)
## Window area = 3.82539e+12 square units
## Fraction of frame area: 0.472
## Quadrature weights:
##      (counting weights based on 150 x 150 array of rectangular tiles)
## All weights:
##  range: [564000, 3.6e+08]    total: 3.83e+12
## Weights on data points:
##  range: [564000, 1.8e+08]    total: 9.56e+10
## Weights on dummy points:
##  range: [564000, 3.6e+08]    total: 3.73e+12
## --------------------------------------------------------------------------------
## FITTED MODEL:
##
## Nonstationary Poisson process
##
## ---- Intensity: ----
##
## Log intensity: ~1 + bio_4 + bio_5 + bio_12 + bio_15
## Model depends on external covariates 'bio_4', 'bio_5', 'bio_12' and 'bio_15'
## Covariates provided:
##  bio_12: im
##  bio_15: im
##  bio_4: im
##  bio_5: im
##
## Fitted trend coefficients:
## (Intercept)       bio_4       bio_5      bio_12      bio_15
```

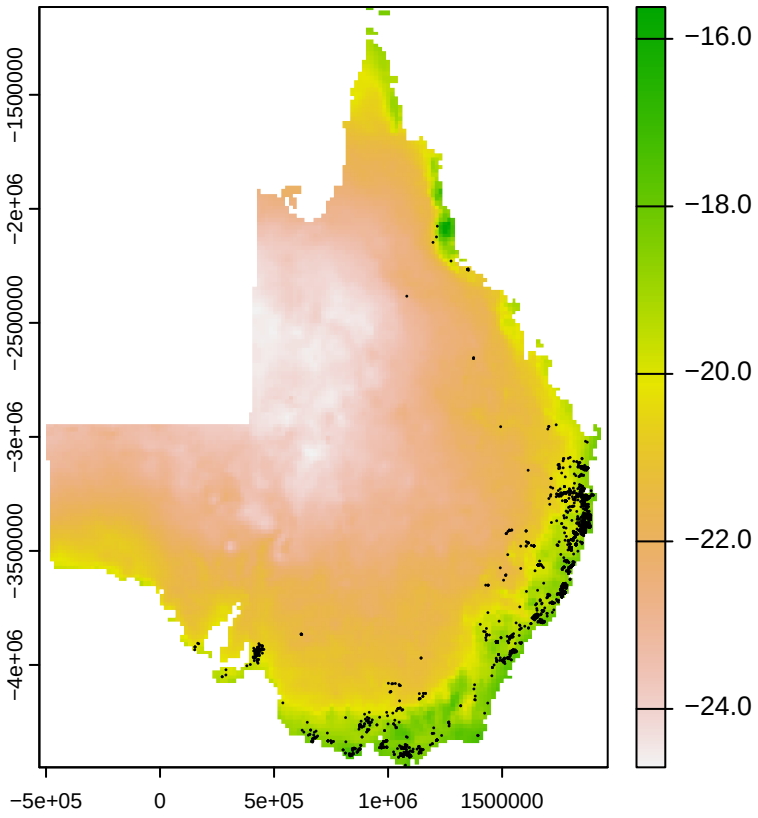
```
## -21.9410757  -0.7754387  -0.5465318   0.5613597  -1.0172469
##
##                Estimate       S.E.     CI95.lo     CI95.hi Ztest       Zval
## (Intercept) -21.9410757 0.03183788 -22.0034768 -21.8786746   *** -689.14996
## bio_4        -0.7754387 0.02883283  -0.8319500  -0.7189273   ***  -26.89429
## bio_5        -0.5465318 0.02200568  -0.5896622  -0.5034015   ***  -24.83594
## bio_12        0.5613597 0.00839050   0.5449146   0.5778048   ***   66.90420
## bio_15       -1.0172469 0.03624721  -1.0882901  -0.9462037   ***  -28.06414
##
## ----------- gory details -----
##
## Fitted regular parameters (theta):
## (Intercept)       bio_4       bio_5      bio_12      bio_15
## -21.9410757  -0.7754387  -0.5465318   0.5613597  -1.0172469
##
## Fitted exp(theta):
##  (Intercept)        bio_4        bio_5       bio_12       bio_15
## 2.958775e-10 4.605017e-01 5.789543e-01 1.753055e+00 3.615891e-01
```

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

Prediction maps

```
pred <- spatstat.core::predict.ppm(fit, covariates = covariates, dimyx = c(1024, 1024))
```

```
plot(log(rast(pred)))
points(configuration, pch = 16, cex = 0.2)
```



PPM advanced topics

Post prediction checks

Here, we use `K` function to assess the clustering on the residual of the model. There are other functions such as `L` function and `g` function.

```
k_func <- spatstat.core::envelope(Y = fit,
                                  fun = spatstat.core::Kinhom,
                                  correction = "translate",
                                  normpower = 2)

plot(k_func)
```

Cox process

Model evaluation