Modelling Geostatistical Data

1 Point processes

In ecology many of the ecological processes of interest can be view as an aggregation of a spatial point process. For example, Figure 1 shows how species occupancy (presence or absence of a species in a given area), and abundance (the number of individual of a species that occur in an area) are related quantities to a spatial point process.

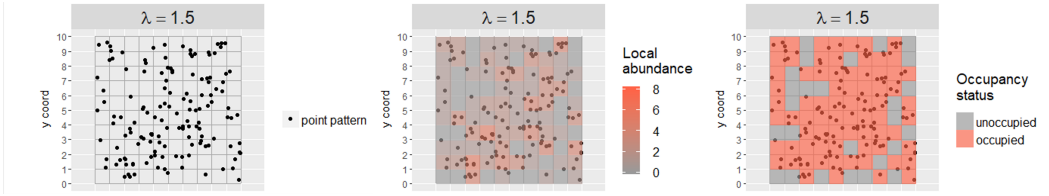


Figure 1: Illustration of the unidirectional information exchange structure of different spatial processes in ecology.

Many contemporary data sources (e.g., Citizen science, GPS tracking, camera traps, biologging devices, etc) collect georeferenced information about the location where species occur. This point-based information provides valuable insights into species distributions and ecosystem dynamics.

A key focus of spatial point pattern analysis is to quantify spatial arrangements, whether aggregated, uniform, or random. Recognizing these patterns is fundamental to interpreting ecological interactions, such as territoriality, competition, and social behavior. Moreover, spatial point patterns can reveal mechanisms of species coexistence and the factors influencing biodiversity.

From a conservation perspective, the clustering of habitat patches—conceptualized as points across a landscape—can have profound effects on extinction risk and population persistence. Understanding the spatial scale of these patterns helps ecologists assess species invasion dynamics and long-term population viability.

In the following sections, we introduce spatial point pattern analysis and its application to ecological and conservation-related questions, emphasizing how species dispersion patterns shape and inform biodiversity management.

A spatial point process is a set of locations…presumed to have been generated by some form of stochastic (random) mechanism".

In other words, the point process is a random variable operating in continuous space, and we observe realisations of this variable as point patterns across space (and/or time). Spatial point pattern analysis focuses on examining patterns of points to establish whether there are regularities in the process they represent.

Consider a fixed geographical region A. The set of locations at which events occur are denoted $\mathbf{s} = s_1, \ldots, s_n$. We let $N(A)$ be the random variable which represents the number of events in region A.

Our primary interest is in measuring where events occur, so the **locations are our data**. We typically assume that a spatial point pattern is generated by an unique point process

over the whole study area. This means that the delimitation of the study area will affect the observed point patters.

The observed distribution of points can be described based on the intensity of points within a delimited region. Thereof, we want to know whether there is any particular spatial pattern associated with the process.

There are three broad types of spatial structures which can be explored, each representing a different type of spatial dependence.

- **Complete spatial randomness** (CSR) - events occur at random, and independently of each other.
- **Clustered process** - events occur close to existing events.
- **Regular process** - events occur away from existing events.

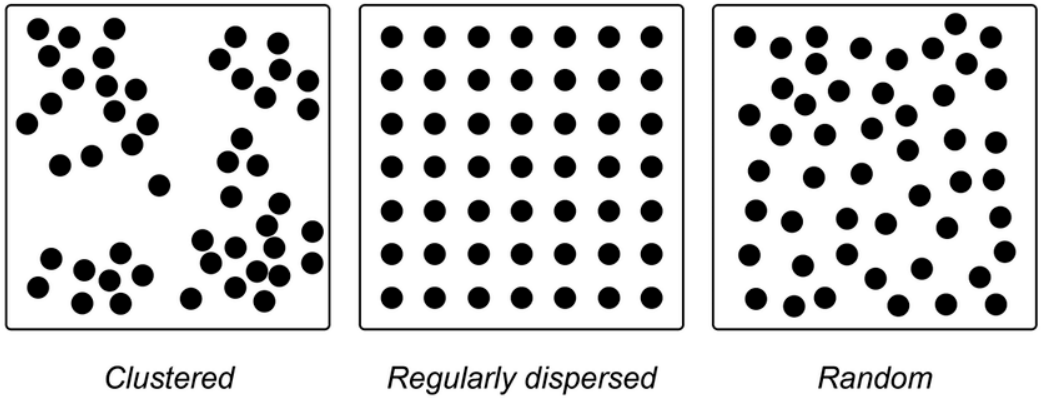


Figure 2: Examples of spatial point patterns

Unfortunately, it still isn't straightforward to determine by eye which of these categories a set of points falls into. So we need a more systematic approach.

1.1 Summaries of Point processes

We can define the (first order) **intensity** of a point process as the **expected number of events per unit area.** This can also be thought of as a measure of the density of our points. In some cases, the intensity will be constant over space (homogeneous), while in other cases it can vary by location (inhomogeneous or heterogenous). If our intensity is homogeneous, we can define it as

$$\lambda(s) = \frac{N(A)}{|A|} = \lambda$$

We can use the concept of intensity to help us define **complete spatial randomness** (CSR).

Def.

For any spatial region A, CSR requires that:

1. *Uniformity and Independent scattering* : Given the number of events $N(A) = n$ in a region, the n events are independently and uniformly distributed over space (i.e., each event has an equal probability of occurring anywhere in the study area).

2. *Poisson distribution of point counts*: The number of points in any set A_i follows a Poisson distribution with mean $\lambda|A_i|$, that is

$$N(A_i) \sim \text{Poisson}(\lambda|A_i|).$$

3. If these conditions are satisfied, we can describe our process as a **homogeneous Poisson process**.

The likelihood of a point pattern $\mathbf{y} = [\mathbf{s}_1, \dots, \mathbf{s}_n]^\top$ distributed as a HPP with intensity λ and observation window Ω is

$$p(\mathbf{y}|\lambda) \propto \lambda^n e^{(-|\Omega|\lambda)},$$

- $|\Omega|$ is the size of the observation window.

```
- $\lambda$ is the expected number of points per unit area.

- $|\Omega|\lambda$ the total expected number of points in the observation window.
```

A key property of a Poisson process is that the number of points within any subset A_i of region A is Poisson distributed with constant rate $|A_i|\lambda$.

While CSR rarely occurs in nature, it is the simplest null model that we use can to a determine whether an homogeneous Poisson process is appropriate for our data or not. We do so by contrasting the observed point pattern with a point pattern generated from the CRS model. One of the most common approaches for identifying point patterns is the **Ripley's K** function.

1.2 Ripley's K function

Ripley's K calculates the degree of spatial aggregation of points within a circle (buffer) of radius r and contrasts the observed pattern to that expected under CSR. Ripley's K is defined as:

$$K(r) = \frac{E[N(s_0, r)]}{\lambda},$$

Here, $N(s_0, r)$ denotes the number of events that occur within distance r of an event s_0; clearly, as r increases, so too will $K(r)$.

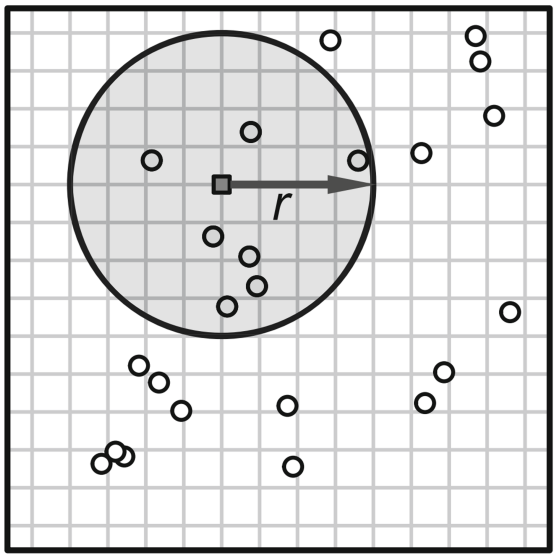


Figure 3: Buffer with radius r centred at the event s_0.

Ripley's K can be estimated as:

$$\hat{K}(r) = \frac{1}{n} \sum_{i=1}^{n} \sum_{i \neq j} I(d_{ij} < r) \times \lambda^{-1}$$

This first part of the equation correspond to the expected number of events that occur within a buffer of radius r , i.e. for each point $s_i \in i, \dots, n$ we count the number of points other than i that fall within the buffer of radius r and then we sum up the number of neighbors for all points and weight it by the total number of points n in the whole region A. The second part of the equation correspond to the density of events estimated as $\lambda = n/|A|$.

The idea is to compare $\hat{K}(r)$ against the expected $K(r)$ under CSR. If we assume an *homogeneous Poisson process* and given the area is that of a circle, we would expect that under CSR the expected $K(r)$ is:

$$K_{CRS}(r) = \frac{\lambda \pi r^2}{\lambda} = \pi \times r^2$$

That is, under CSR we would expect that the K function is equal to the area of the circle with radius r. Then we compare $\hat{K}(r)$ and $K_{CSR}(r)$:

- If $\hat{K}(r) > K_{CSR}(r)$ it means that more points are found within a radius r than what would be expected under complete randomness, suggesting a clustering pattern. $\hat{K}(r)$ will be relatively large for small values of r, since events are likely to be surrounded by further members of the same cluster. E.g., tree seedlings often cluster near parent trees due to seed dispersal limitations.
- If $\hat{K}(r) < K_{CSR}(r)$, it indicates that the pattern is more regular since we observe fewer neighboring points within a distance r than expected under CSR. For regular patterns, $K(r)$ will be relatively small for small values of r, since there is likely to be more empty space around events. E.g., territorial animals (e.g., nesting birds) often exhibit regular spacing due to competition for space.

When working with real data, some natural variation is to be expected even when CSR holds. We therefore need an approach which accounts for this when assessing for CSR to determine whether or not the observed pattern is non-random.

We can estimate $\hat{K}(r)$ across a set of distances r for our set of observed events. Then, $\hat{K}(r)$ can be compared to the theoretical function for CSR, $K(r) = \pi \times r^2$. If the two functions are similar, then CSR is reasonable. See the example below,

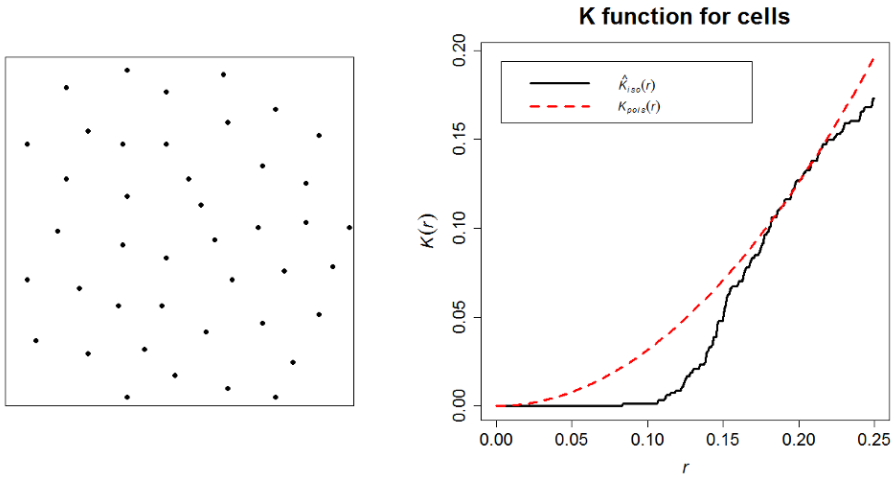


Figure 4: Simulated regular point process data and associate Ripley's K function. Red line indicates theoretical $K_{CSR}(r)$ and back line $\hat{K}(r)$.

In this example we can notice that for most distances $\hat{K}(r) < K_{CSR}(r)$. However we cannot be completely sure if the observed point pattern is best described by a CSR process or if the pattern is more regular. Thus, we might wish to be more precise in our comparison of the two lines.

We can calculate what is known as a *simulation envelope* which is produced by simulating multiple sets of data under CSR. These are not theoretically the same as confidence intervals, but can be described as significance bands. If our observed line falls outside of the envelope, this implies that CSR is not reasonable. Constructing a simulation envelope in our example produces:

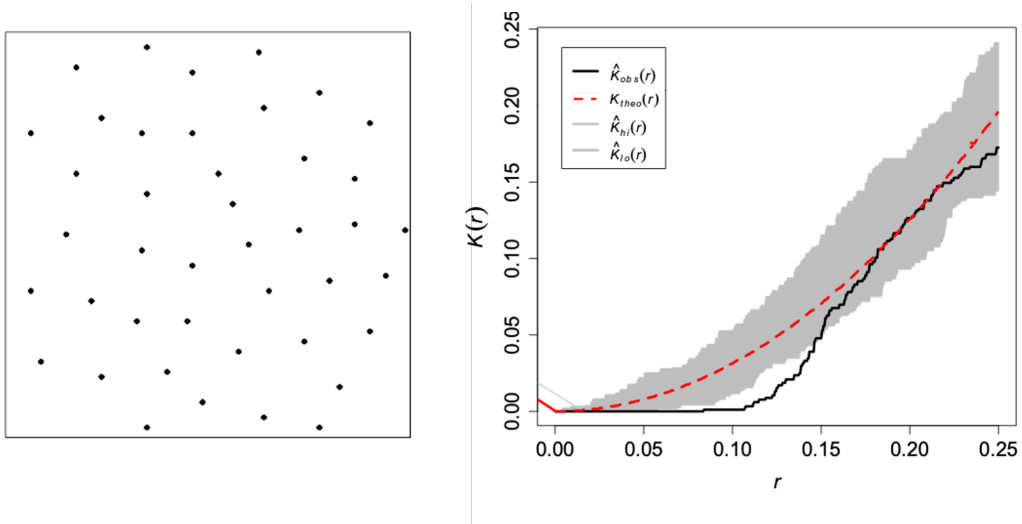


Figure 5: Simulated regular point process data and associate Ripley's K function. Red line dotted indicates theoretical $K_{CSR}(r)$ with grey areas denoting the *simulation envelope* and black solid line $\hat{K}(r)$.

In this example the black line lies outside the envelope between $r = 0$ and $r = 0.15$ therefore we still conclude that the data are regularly spaced. Figure 6 shows examples of simulated cluster and regular patterns with their associated Ripley's K function.

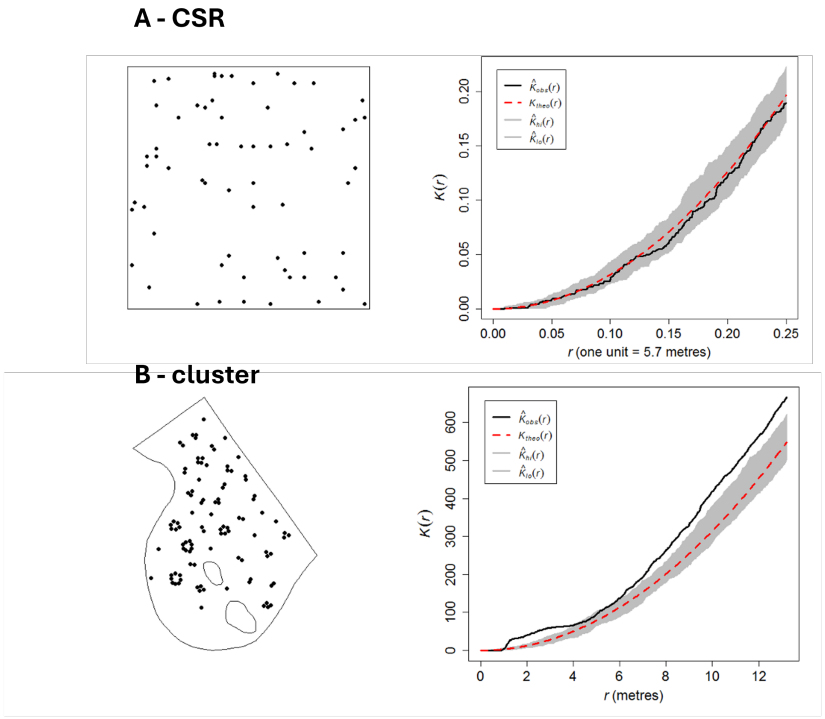


Figure 6: Simulated random (A) and clusterred (B) point processes data and associated Ripley's K function. Red line dotted indicates theoretical $K_{CSR}(r)$ with grey areas denoting the *simulation envelope* and black solid line $\hat{K}(r)$.

While Ripley's K function is widely used in environmental and ecological studies it has some caveats. For example, $K(s)$ is a cumulative function, where all points less than r are also used. So, if there is a strong clustering patter at 5m but no pattern at larger distances, then Ripley's K could still indicate a strong clustering at larger scales due to the data $<$ 5m still being used.

Another issues with this method is its sensitivity to *edge effects*. This occurs because points near the boundaries of the study area have fewer neighboring points within distance r, leading to underestimation of $K(r)$ (when considering a radius r from a point near a boundary, the number of observed points is likely lower than the true number of points if points could occur outside of study area).

The $K(r)$ function can be adjusted for edge effects by including some weights w. There are several weights to specify these weights. For example, if the ith buffer lies completely within the study area then it receives a weight of $w = 1$ but if it lies outside it receives a weight inverse of the fraction of circumference lying inside the area, (e.g., if half of a point's search area lies within the study area then the point will receive a weight $w = 2$, meaning it contributes twice as much to compensate for the missing area).

Lastly, the above analyses assume that the point process is *stationary* (homogeneous over space) and isotropic (point process does not change with direction). These assumption rarely hold true in real-data, thus, *inhomogeneous point process* (IPP) *models* are often used for inference prediction and mapping spatial patterns.

2 Inhomogeneous Point Process Models

So far we have assumed that the point process is stationary and isotropic. However, these assumption rarely hold true in real-data. Thus, *inhomogeneous Poisson process* (IPP) models are often used for inference prediction and mapping spatial patterns

The IPP has a spatially varying intensity $\lambda(\mathbf{s})$ defined in terms of spatially varying covariates that are available across the whole study area:

$$\lambda(s) = \exp(\alpha + \beta x(s) + ...)$$

where $\beta = \{\beta_1, ..., \beta_p\}$ is a vector of parameters corresponding to the p environmental covariates $\mathbf{x}(s)$. Let $\mathbf{y} = s_1, ..., s_n$ the n number of observed events/points in an observation window Ω

For an IPP with an intensity $\lambda(s)$, the likelihood is given by:

$$p(\mathbf{y}|\lambda) \propto \exp\left(-\int_\Omega \lambda(\mathbf{s})d\mathbf{s}\right)\prod_{i=1}^{n}\lambda(\mathbf{s}_i).$$

Note that inf the case of an **HPP** the integral in the likelihood can easily be computed as $\int_\Omega \lambda(\mathbf{s})d\mathbf{s} = |\Omega|\lambda$. For an **HPP** with an intensity λ, the log-likelihood is given by:

$$l(\beta; \mathbf{y}) = n\log(\lambda) - \lambda|\Omega|,$$

Thus, the maximum likelihood estimators is $\hat{\lambda} = n/|\Omega|$. However, for an **IPP**, the integral in the likelihood has to be approximateda as a weighted sum.

2.1 Fitting an IPP

This integral is approximated as $\int_\Omega \lambda(\mathbf{s})d\mathbf{s} \approx \sum_{j=1}^{J} w_j\lambda(\mathbf{s}_j)$.

- w_j are the integration weights
- $\mathbf{s}_j$ are the quadrature locations.

This serves two purposes:

1. Approximating the integral
2. re-writing the inhomogeneous Poisson process likelihood as a regular Poisson likelihood.

The idea behind this trick is to rewrite the approximate likelihood by introducing a dummy vector **z** and an integration weights vector **w** of length $J + n$ such that

- $\mathbf{z} = \left[\underbrace{0_1, ..., 0_J}_{\text{quadrature locations}}, \underbrace{1_1, ..., 1_n}_{\text{data points}}\right]^\top$
- $\mathbf{w} = \left[\underbrace{w_1, ..., w_J}_{\text{quadrature locations}}, \underbrace{0_1, ..., 0_n}_{\text{data points}}\right]^\top$

Then the approximate likelihood can be written as

$$p(\mathbf{z}|\lambda) \propto \prod_{i=1}^{J+n} \eta_i^{z_i} \exp\left(-w_i \eta_i\right)$$
$$\eta_i = \log \lambda(\mathbf{s}_i) = \mathbf{x}(s)'\beta$$

This is similar to a product of Poisson distributions with means η_i, exposures w_i and observations z_i.

3 Log Gaussian Cox processes

IPP models assume that data points are conditionally independent given the covariates, meaning that any spatial clustering is fully explained by environmental and sampling factors. However, this assumption often fails in practice due to biological processes like dispersal, social aggregation, or unmeasured environmental variables that create spatial dependence. Ignoring such dependencies can lead to misleading conclusions. A more flexible approach to account for spatial dependence is the **log-Gaussian Cox process (LGCP)**, a type of **Cox process** where the intensity function itself follows a Gaussian stochastic process $\log(\lambda(s) = Z(s)$. This allows for a varying intensity across space, leading to an **inhomogeneous Poisson process** with spatially structured variability.

An LGCP extends the Poisson process by allowing the intensity function to vary spatially as:

$$\log(\lambda(s)) = \mathbf{x}(s)'\beta + Z(s) \tag{1}$$

Where $Z(s)$ is a spatial Gaussian field with mean zero and a covariance function that depends on the distance between observations. The events are then assumed to be independent given the Gaussian field. Moreover, conditional on a realisation of the random field, a log-Gaussian Cox process is an inhomogeneous Poisson process.

An computationally efficient method to perform estimate LGCP is presented in Simpson et al. (2016) This method represents the Gaussian random field using a finite-dimensional continuous formulation with basis functions:

$$Z(s) = \sum_{i=1}^{n} z_i \phi_i(s), \tag{2}$$

$z = (z_1, \ldots, z_n)'$ is a multivariate Gaussian random vector and $\{\phi_i(s)_{i=1}^n\}$ is a set of linearly independent deterministic basis functions. Unlike lattice-based approaches that bin observations into cells, it models them at their exact locations and approximates the Gaussian random field using a triangulated mesh via the so-called SPDE approach.

Recall that the stochastic partial differential equation (SPDE) approach proposed by Lindgren et al. (2011), represents the continuous spatial process (e.g., the $Z(\cdot)$ GRF in Equation 1) as a discretely indexed spatial random process such as a Gaussian Markov Random Field (GMRF). The general idea is construct an appropriate lower-resolution approximation of the surface by sampling it in a set of well designed points (integration points) and constructing a piecewise linear interpolant (e.g., the piecewise linear basis functions defined in Equation 2) .

4 Example: Forest fires in Castilla-La Mancha

This dataset is a record of forest fires in the Castilla-La Mancha region of Spain between 1998 and 2007. This region covers approximately 400 by 400 kilometres. The coordinates are recorded in kilometres.

> **Note**
>
> The code and details of this section will be discussed in greater detail on the lecture and on the lab session

For more info about the data you can type:

```
?clmfires
```

```
# Load some libraries
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
library(spatstat)
library(sf)
library(scico)
library(spatstat)
library(lubridate)
library(terra)
library(tidyterra)
```

We first read the data and transform them into an `sf` object. We also create a polygon that represents the border of the Castilla-La Mancha region. We select the data for year 2004 and only those fires caused by lightning.

```
data("clmfires")
pp = st_as_sf(as.data.frame(clmfires) %>%
                mutate(x = x,
                       y = y),
              coords = c("x","y"),
              crs = NA) %>%
  filter(cause == "lightning",
         year(date) == 2004)

poly = as.data.frame(clmfires$window$bdry[[1]]) %>%
  mutate(ID = 1)

region = poly %>%
  st_as_sf(coords = c("x", "y"), crs = NA) %>%
  dplyr::group_by(ID) %>%
  summarise(geometry = st_combine(geometry)) %>%
  st_cast("POLYGON")

ggplot() + geom_sf(data = region, alpha = 0) + geom_sf(data = pp)
```

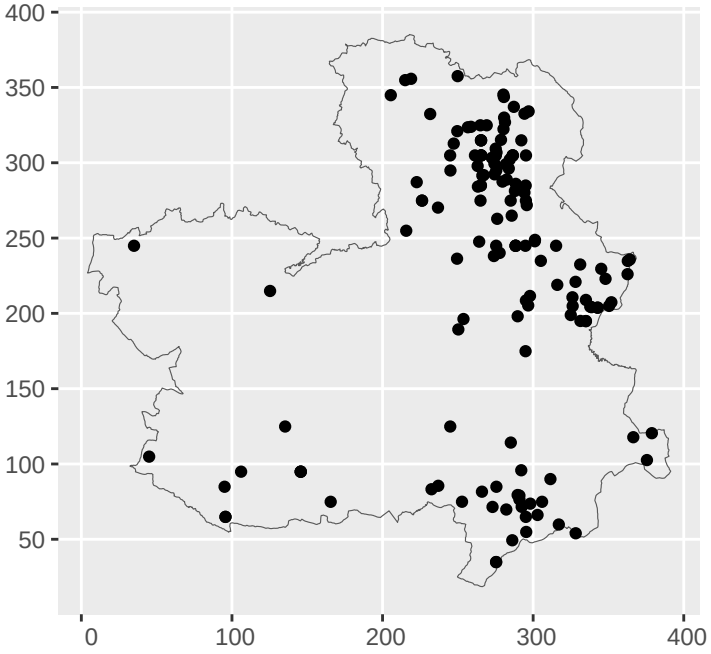


Figure 7: Distribution of the observed forest fires caused by lightning in Castilla-La Mancha in 2004

4.1 Fit a homogeneous Poisson Process

As a first exercise we are going to fit a homogeneous Poisson process (HPP) to the data. This is a model that assume constant intensity over the whole space so our linear predictor is then:

$$\eta(s) = \log \lambda(s) = \beta_0, \ \mathbf{s} \in \Omega$$

so the likelihood can be written as:

$$\begin{aligned} p(\mathbf{y}|\lambda) &\propto \exp\left(-\int_\Omega \lambda(\mathbf{s})d\mathbf{s}\right) \prod_{i=1}^{n} \lambda(\mathbf{s}_i) \\ &= \exp\left(-\int_\Omega \exp(\beta_0)ds\right) \prod_{i=1}^{n} \lambda(\mathbf{s}_i) \end{aligned}$$

where $|\Omega|$ is the area of the domain of interest.

We need to approximate the integral using a numerical integration scheme as:

$$\approx \exp\left(-\sum_{k=1}^{N_k} w_k \lambda(s_k)\right) \prod_{i=1}^{n} \lambda(\mathbf{s}_i)$$

Where N_k is the number of integration points $s_1, \dots, s_{N_k}$ and $w_1, \dots, w_{N_k}$ are the integration weights.

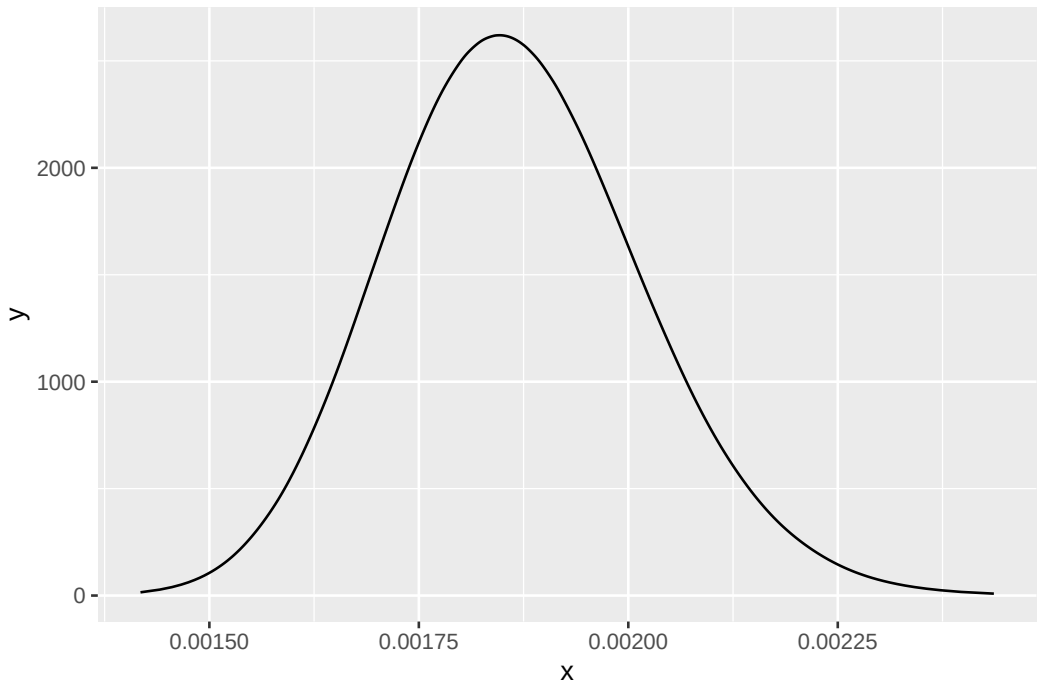
In this case, since the intensity is constant, the integration scheme is really simple: it is enough to consider one random point inside the domain with weight equal to the area of the domain.

```
# define integration scheme

ips <- fm_int_object(
  st_sample(region, 1), # some random location inside the domain
  weight = st_area(region), # integration weight is the area of the domain
  name = "geometry"
)


cmp = ~ 0 + beta_0(1)

formula = geometry ~ beta_0

lik = bru_obs(data = pp,
              family = "cp",
              formula = formula,
              ips = ips)
fit1 = bru(cmp, lik)
```

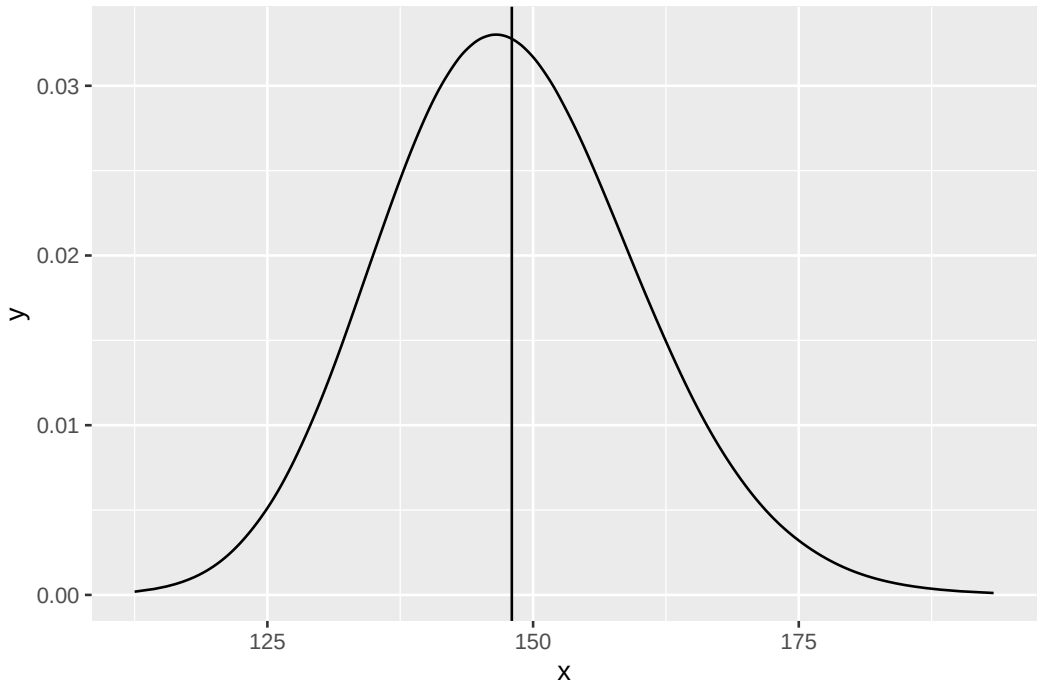
We can then:

1. Plot the estimated posterior distribution of the intensity
2. Compare the estimated expected number of fires on the whole domain with the observed ones.

```
# 1) The estimated posterior distribution of the  intensity is

post_int = inla.tmarginal(function(x) exp(x), fit1$marginals.fixed$beta_0)
post_int %>% ggplot() + geom_line(aes(x,y))
```



```
# 2) To compute the expected number of points in the area we need to multiply the
# estimated intensity by the area of the domain.
# In the same plot we also show the number of observed fires as a vertical line.

post_int = inla.tmarginal(function(x) st_area(region)* exp(x), fit1$marginals.fixed$beta_0)
post_int %>% ggplot() + geom_line(aes(x,y)) +
  geom_vline(xintercept = dim(pp)[1])
```



4.2 Fit an Inhomogeneous Poisson Process

The model above has the clear disadvantages that assumes a constant intensity and from Figure 7 we clearly see that this is not the case.

The library `spatstat` contains also some covariates that can help explain the fires distribution. Figure (**fit-altitude?**) shows the location of fires together with the (scaled) altitude.

```
elev_raster = rast(clmfires.extra[[2]]$elevation)
elev_raster = scale(elev_raster)
ggplot() +
  geom_spatraster(data = elev_raster) +
  geom_sf(data = pp) +
  scale_fill_scico()
```

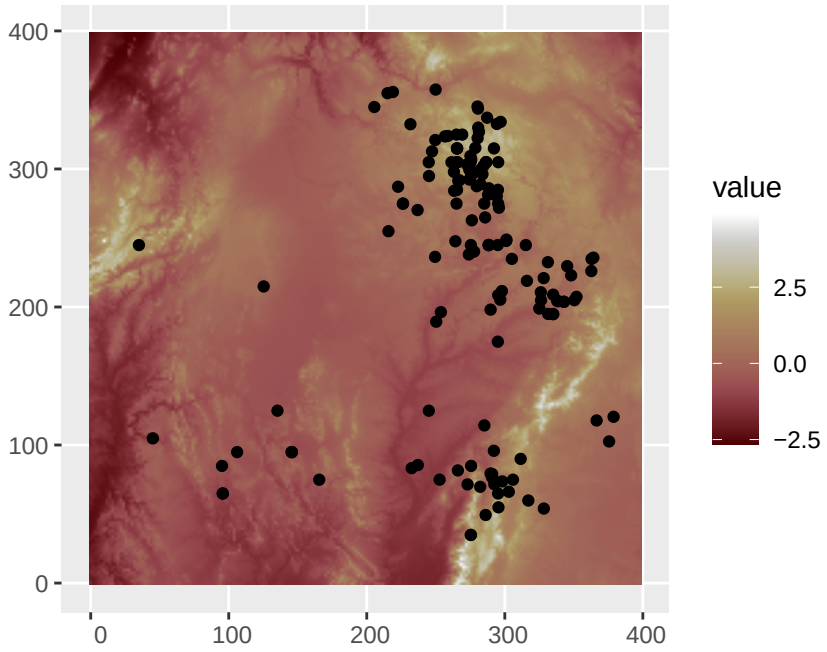


Figure 8: Distribution of the observed forest fires and scaled altitude

We are now going to use the altitude as a covariate to explain the variability of the intensity $\lambda(s)$ over the domain of interest.

Our model is

$$\log \lambda(s) = \beta_0 + \beta_1 x(s)$$

where $x(s)$ is the altitude at location s.

The likelihood becomes:

$$\begin{aligned} p(\mathbf{y}|\lambda) &\propto \exp\left(-\int_\Omega \lambda(\mathbf{s})d\mathbf{s}\right) \prod_{i=1}^{n} \lambda(\mathbf{s}_i) \\ &= \exp\left(-\int_\Omega \exp(\beta_0 + \beta_1 x(s))d\mathbf{s}\right) \prod_{i=1}^{n} \lambda(\mathbf{s}_i) \end{aligned}$$

Now we need to choose an integration scheme to solve the integral.

In this case we will take a simple grid based approach where each quadrature location has an equal weight. Our grid consists of $N_k = 1000$ points and the weights are all equal to $|\Omega|/N_k$.

```
n.int = 1000
ips <- st_sample(region, size = n.int, type = "regular") # May not be exactly n.int points
ips <- fm_int_object(
  ips,
  weight = st_area(region) / length(ips),
  name = "geometry"
)
ggplot() + geom_sf(data = ips, aes(color = weight)) + geom_sf(data= region, alpha = 0)
```

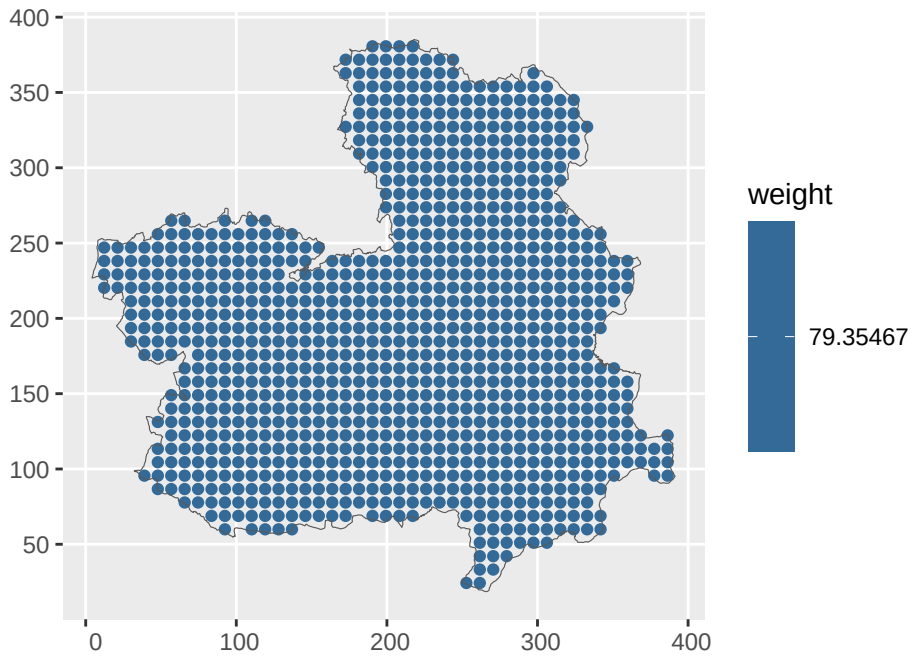


Figure 9: Integration scheme.

OBS: The implicit assumption here is that the intensity is constant inside each grid box, *and so is the covariate*!!

We can now fit the model:

```
cmp = ~ Intercept(1) + elev(elev_raster, model = "linear")
formula = geometry ~ Intercept + elev
lik = bru_obs(data = pp,
              family = "cp",
              formula = formula,
              ips = ips)
fit2 = bru(cmp, lik)
```

Task

What is the effect of the altitude on the (log) intensity of the process?
Take hint
You can look at the summary for the fixed effects
Click here to see the solution

```
fit2$summary.fixed
```

```
                mean         sd 0.025quant   0.5quant 0.975quant       mode kld
Intercept -6.6426089 0.10465414 -6.8477273 -6.6426089 -6.4374906 -6.6426089   0
elev       0.7383368 0.07450872  0.5923024  0.7383368  0.8843712  0.7383368   0
```

Warning

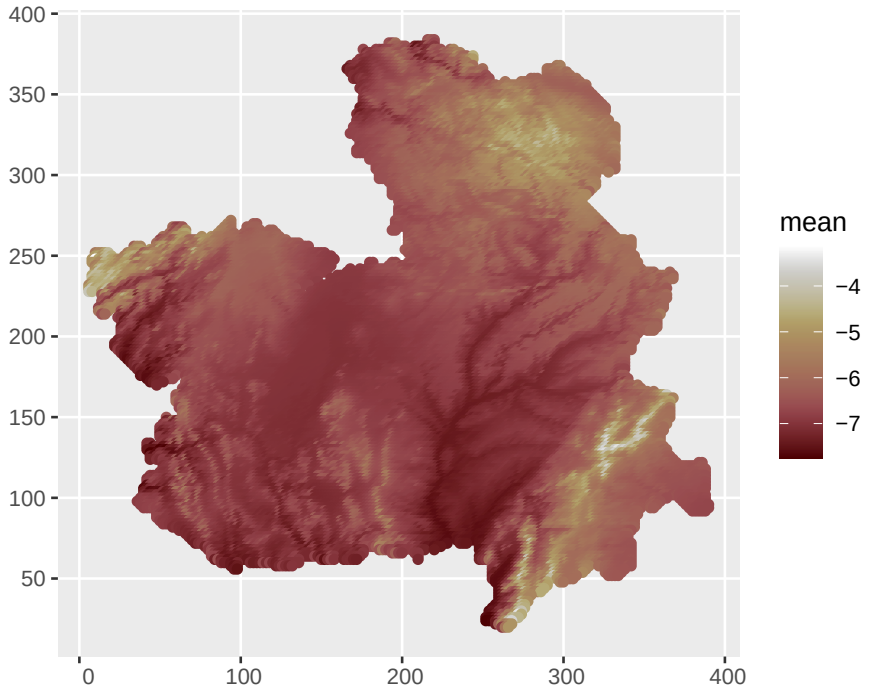
 WARNING!! When fitting a Point process, the integration scheme has to be fine enough to capture the spatial variability of the covariate!!

Now we want to predict the log-intensity over the whole domain. Use the grid from the

elevation raster to predict the intensity over the domain.

```
est_grid = st_as_sf(data.frame(crds(elev_raster)), coords = c("x","y"))
est_grid  = est_grid[region,]
```

```
preds2 = predict(fit2, est_grid, ~ data.frame(log_scale = Intercept + elev,

                                              lin_scale = exp(Intercept + elev)))
# then visualize it like
preds2$log_scale %>%
  ggplot() +
  geom_sf(aes(color = mean)) +
  scale_color_scico()
```



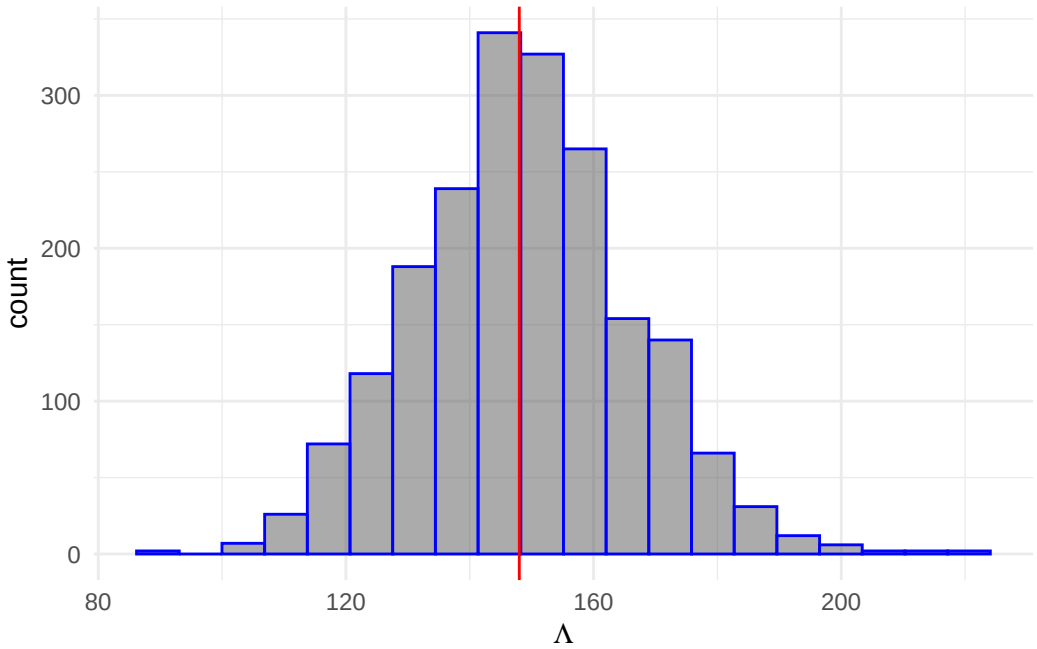
Finally, we want to use the fitted model to estimate the total number of fires over the whole region. To do this we first have to fine the expected number of fires as:

$$E(N_\Omega) = \int_\Omega \exp(\lambda(s))\, ds$$

Then simulate possible realizations of N_Ω to include also the likelihood variability in our estimate:

```
N_fires = generate(fit2, ips,
                       formula = ~ {
                         lambda = sum(weight * exp(elev + Intercept))
                         rpois(1, lambda)},
                     n.samples = 2000)

ggplot(data = data.frame(N = as.vector(N_fires))) +
```

```
  geom_histogram(aes(x = N),
                 colour = "blue",
                 alpha = 0.5,
                 bins = 20) +
  geom_vline(xintercept = nrow(pp),
             colour = "red") +
  theme_minimal() +
  xlab(expression(Lambda))
```



4.3 Fit a Log-Gaussian Cox Process

Finally we want to fit a LGCP with log intensity:

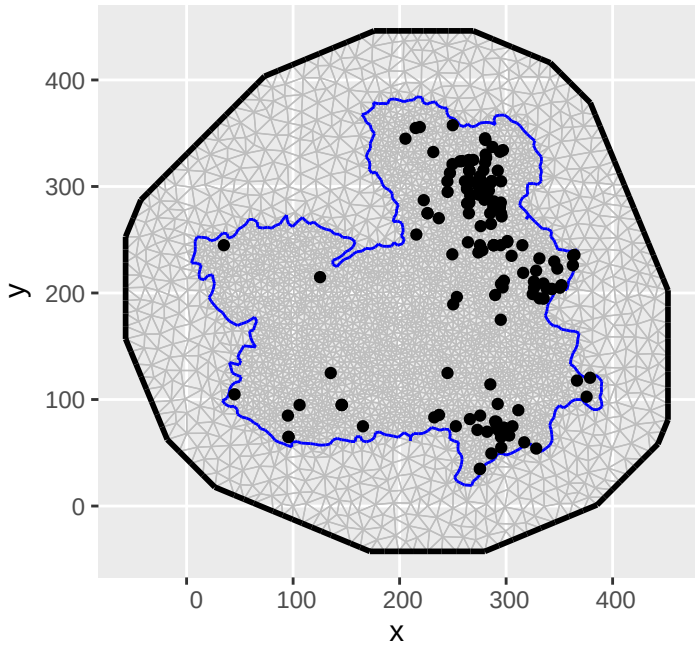
$$\log(s) = \beta_0 + \beta_1 x + u(s)$$

where β_0 is the intercept, β_1 the effect of (standardized) altitude $x(s)$ as before and $u(s)$ is a Gaussian Random field defined through the SPDE approach.

4.3.1 Define the mesh

The first step, as any time we use the SPDE approach is to define the mesh and the priors for the marginal variance and range:

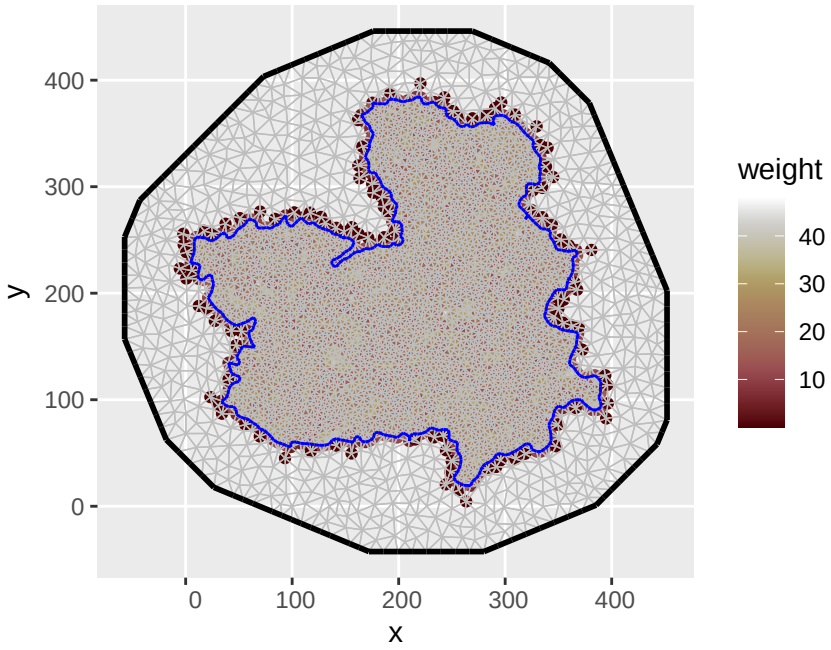
```
mesh = fm_mesh_2d(boundary = region,
                  max.edge = c(8, 20),
                  cutoff = 4, crs = NA)

ggplot() + gg(mesh) + geom_sf(data = pp)
```




```
spde_model =  inla.spde2.pcmatern(mesh,
                                  prior.sigma = c(1, 0.5),
                                  prior.range = c(100, 0.5))
```

We can then define the integration weight. Here we use the same points to `define the SPDE approximation` and to `approximate the integral` in the likelihood. We will see later that this does not have to be like this, BUT integration weight and SPDE weights have to be consistent with each other!

```
ips = fm_int(mesh, samplers = region)

ggplot() + geom_sf(data = ips, aes(color = weight)) +
  gg(mesh) +
   scale_color_scico()
```



4.3.2 Run the model

```
cmp = ~ Intercept(1) + space(geometry, model = spde_model) + elev(elev_raster, model = "line

formula = geometry ~ Intercept + space + elev

lik = bru_obs("cp",
              formula = formula,
              data = pp,
              ips = ips)

fit3 = bru(cmp, lik)
```

Note when running the model above you will get a warning:

```
Warning in bru_log_warn(msg): Model input 'elev_raster' for 'elev' returned some NA values.
Attempting to fill in spatially by nearest available value.
To avoid this basic covariate imputation, supply complete data.
```

It means that the `bru()` function cannot find the covariate values for some of the mesh nodes. This is a common situation. As the warning says, the `bru()` function automatically imputes the value of the covarite using the nearest nodes. This increases the running time of the`bru()` function, so one solution is to impute the values of the covariate over the whole mesh '*before*' running the `bru()` function.

Here, we notice that there are points for which elevation values are missing (see Figure 10 the red points that lies outside the raster extension).

To solve this, we can increase the raster extension so it covers all both data-points and quadrature locations as well. Then, we can use the `bru_fill_missing()` function to input the missing values with the nearest-available-value. We can achieve this using the following code:

```
# Extend raster ext by 30 % of the original raster so it covers the whole mesh
re <- extend(elev_raster, ext(elev_raster)*1.3)
# Convert to an sf spatial object
re_df <- re %>% stars::st_as_stars() %>%  st_as_sf(na.rm=F)
# fill in missing values using the original raster
re_df$lyr.1 <- bru_fill_missing(elev_raster,re_df,re_df$lyr.1)
# rasterize
elev_rast_p <- stars::st_rasterize(re_df) %>% rast()
ggplot() + geom_spatraster(data = elev_rast_p)+
  geom_sf(data=ips,alpha=0.25,col=1)
```

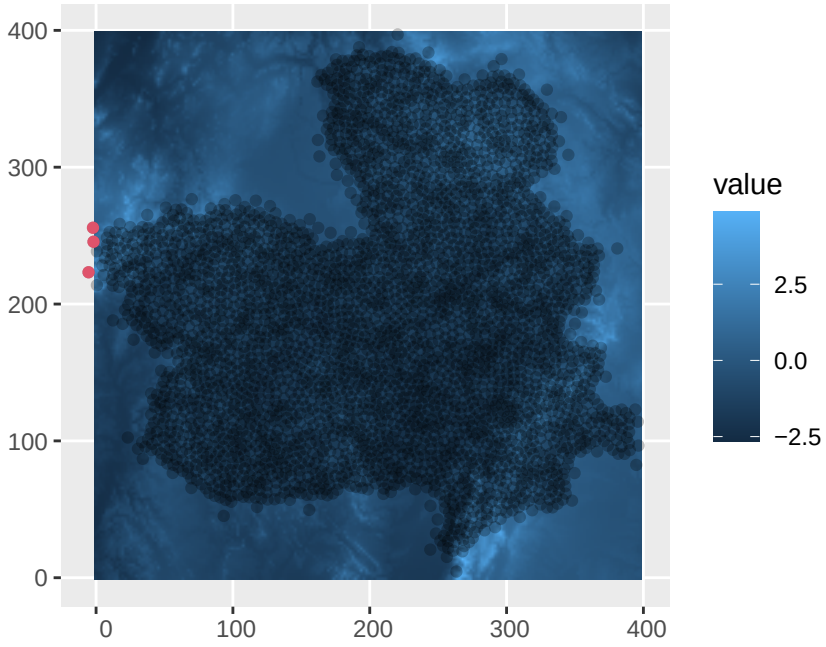
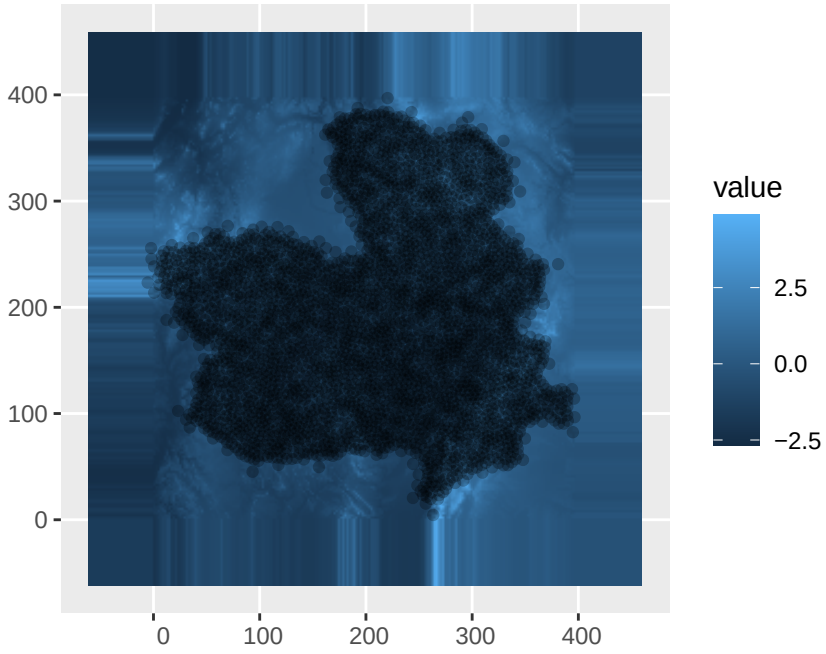


Figure 10: Integration scheme for numerical approximation of the stochastic integral in La Mancha Region

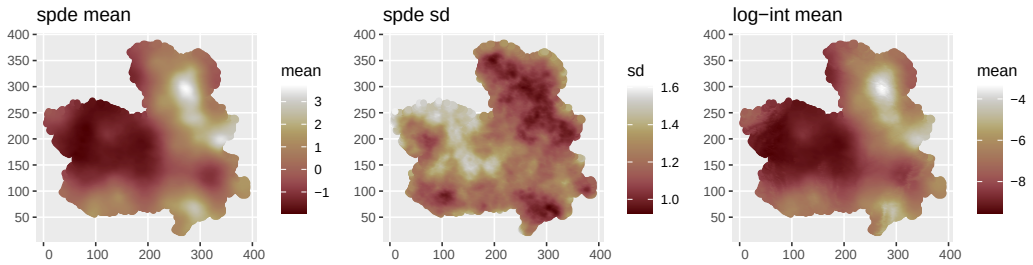


> **Note**
>
> The `bru_fill_missing()` function was added mainly to handle very local infilling on domain boundaries. For properly missing data, one should consider doing a proper model of the spatial field instead.

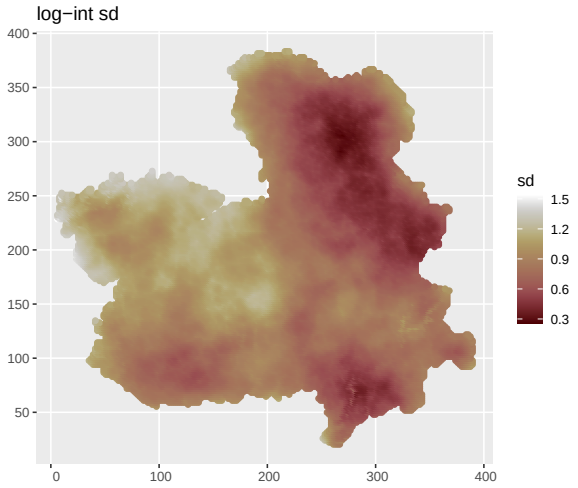
4.3.3 Results

We can plot the estimated mean and standard deviation of the spatial GF and the log-intensity over the domain of interest

```
pxl = fm_pixels(mesh, mask= region, dims = c(200,200))
preds = predict(fit3, pxl, ~data.frame(spde = space,
                                       log_int = Intercept + space + elev))

#and plot as
library(scico)
library(patchwork)

ggplot(data= preds$spde) +
  geom_sf(aes(color = mean)) +
  scale_color_scico() +
 ggtitle("spde mean") +
ggplot(data=preds$spde ) +
  geom_sf(aes(color = sd)) +
  scale_color_scico() +
 ggtitle("spde sd") +

ggplot(data=preds$log_int) +
  geom_sf(aes(color = mean)) +
  scale_color_scico() +
 ggtitle("log-int mean")
```




```
ggplot(data=preds$log_int) +
  geom_sf(aes(color = sd)) +
  scale_color_scico() +
 ggtitle("log-int sd") +
  plot_layout(ncol=2)
```



5 Distance Sampling

Distance sampling encompasses a family of related methods for estimating the abundance and spatial distribution of wild populations. The fundamental insight underlying these methods is that animals located further from observers are harder to detect than those that are nearer. This intuitive concept is formalized through a detection function that models the probability of detecting an animal as a function of its distance from the observer. As distance increases, the detection probability correspondingly declines, allowing us to account for animals that are present in the area but remain undetected during surveys.

5.1 Density Surface Models

Coupling distance sampling data with spatial modelling enables the production of maps showing spatially varying population density. This represents a significant advance over global abundance estimates, as it reveals how animals distribute themselves across the landscape in relation to environmental features. Historically, density surface modelling has been implemented through a two-stage process. The first stage involves estimating detectability to create an offset vector that accounts for imperfect detection. This offset is then incorporated into a second-stage model—typically a Generalized Linear Model (GLM) or Generalized Additive Model (GAM)—that analyzes count response data. Implementing this approach requires discretizing the continuous survey area into segments and aggregating the detected animals into counts within each spatial unit.

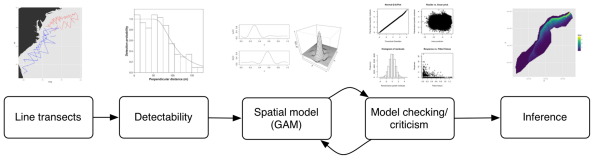


Figure 11: Schematic of the two-stage density surface modelling approach

The discretization process necessary for this approach is illustrated below, showing how continuous survey transects are divided into segments for analysis:

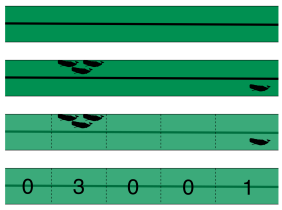


Figure 12: Discretization of survey data for two-stage modelling

Despite its widespread use, the two-stage approach suffers from a critical limitation: it fails to properly propagate uncertainty from the first-stage detection model to the second-stage spatial model. The detection function parameters are estimated with associated uncertainty, but this uncertainty is effectively fixed when creating the offset, leading to potentially overconfident predictions and underestimated standard errors in the final spatial maps.

The solution

The goal of contemporary methodological development has been to create a one-stage distance sampling model that simultaneously estimates detectability and the spatial distribution of animals. This unified approach is achieved through a point process framework, which naturally accommodates both the ecological process generating animal locations and the observation process determining which animals are detected.

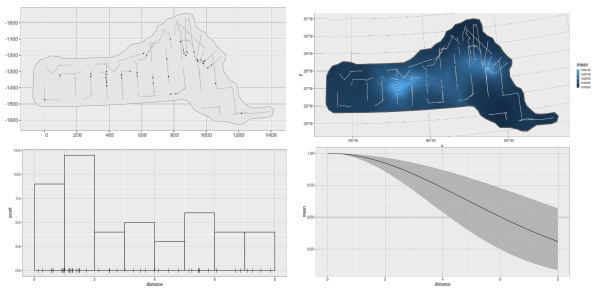


Figure 13: The one-stage approach using a point process framework

This integrated framework ensures that all sources of uncertainty are accounted for in a single, coherent analysis, producing more reliable estimates and predictions.

5.2 Thinned Point Processes

The Log-Gaussian Cox Process (LGCP) provides a flexible and powerful framework for modelling spatial point patterns. It models the intensity of points—representing animal

locations—by incorporating the effects of spatial covariates and including a mean-zero spatially structured random effect. This random effect accounts for unexplained heterogeneity in the distribution that is not captured by the measured covariates, making the LGCP particularly suitable for ecological data where many influential factors may be unobserved.

Ecological survey data never represents a complete census of all animals present. To account for the imperfect detection inherent in distance sampling, we specify a thinning probability function:

$$g(s) = \mathbb{P}(\text{a point at s is detected} | \text{a point is at s})$$

This function represents the probability that an animal at location s is detected by the observer, given that an animal is actually present at that location.

A key and remarkably useful property of the Log-Gaussian Cox Process is that a realization of a point process with intensity $\lambda(s)$, when thinned by the probability function $g(s)$, itself follows a LGCP. The intensity of this observed, thinned process is simply the product of the true intensity and the thinning probability:

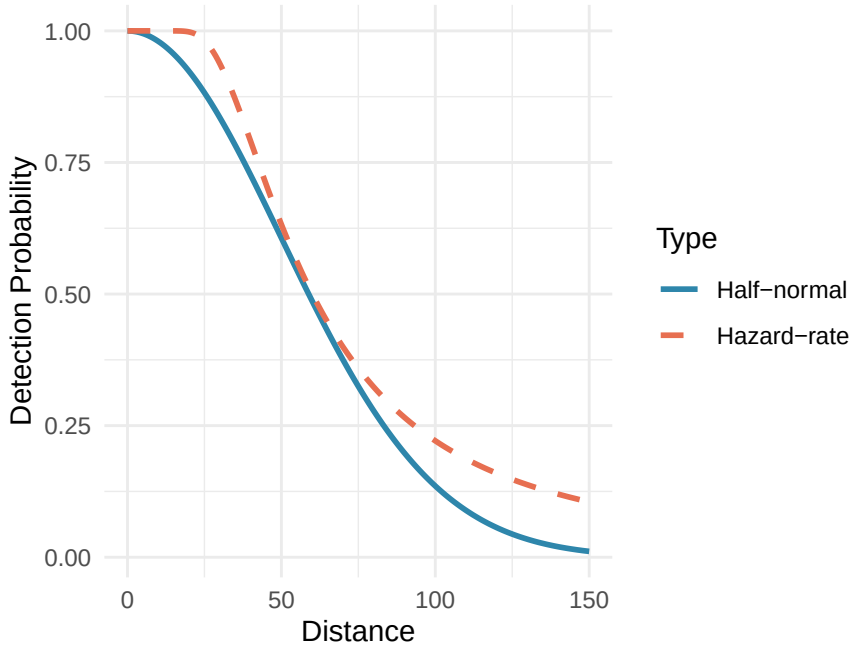
$$\underbrace{\tilde{\lambda}(s)}_{\text{observed process}} = \underbrace{\lambda(s)}_{\text{true process}} \times \underbrace{g(s)}_{\text{thinning probability}}$$

This mathematical formulation allow us to build unified models that simultaneously estimate both the true density of animals and the detection process. The thinning framework then separates the ecological process of interest from the observation process, while maintaining them within a single, coherent statistical model.

Standard distance sampling approaches specify $g(s)$ as a function that declines with increasing distance: - horizontal distance to the observer for *point transects* - perpendicular distance to the transect line for *line transects*

In this course we will focus on perpendicular distances only. The thinning probability function is specified as a parametric family of functions:

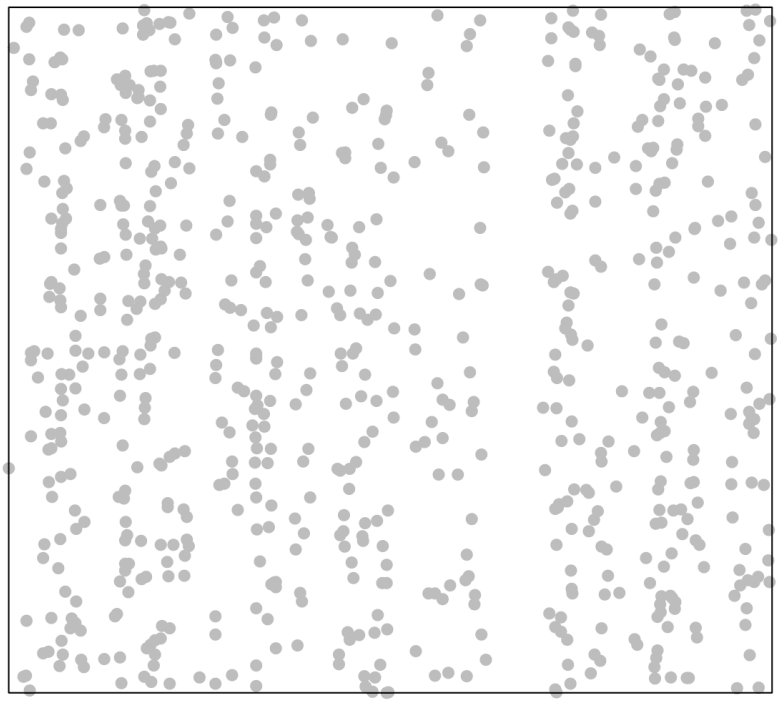
- **Half-normal**: $g(\mathbf{s}|\sigma) = \exp(-0.5(d(\mathbf{s})/\sigma)^2)$
- **Hazard-rate** :$g(\mathbf{s}|\sigma) = 1 - \exp(-(d(\mathbf{s})/\sigma)^{-1})$



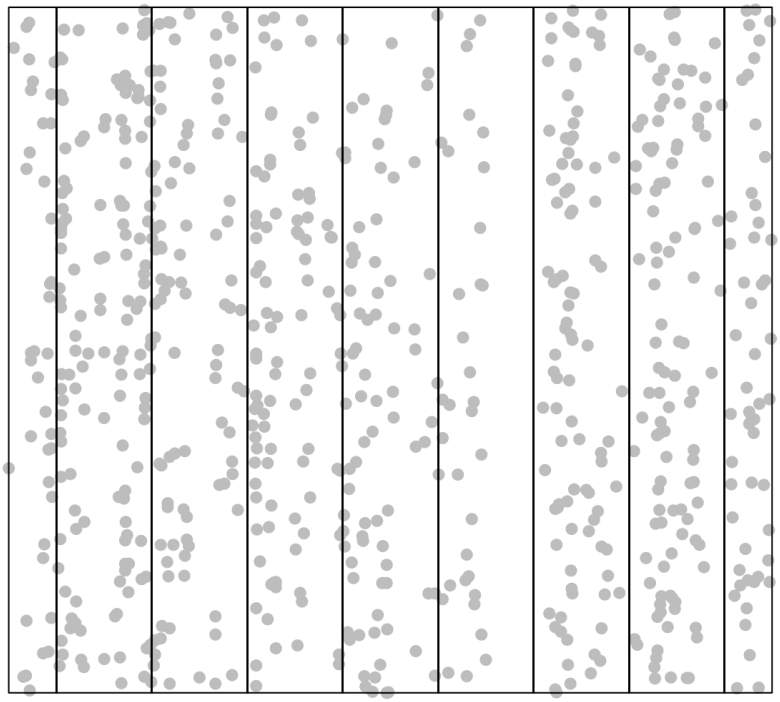
To make $g(s)$ and $\lambda(s)$ identifiable, we assume intensity is constant with respect to distance from the observer. In practice this means we assume animals are uniformly distributed with respect to distance from the line

5.3 Summary of Distance sampling PP approach

1. The true point pattern $Y = \mathbf{s}_1, \ldots, \mathbf{s}_n$ are a realization of a Point process with intensity $\lambda(s)$

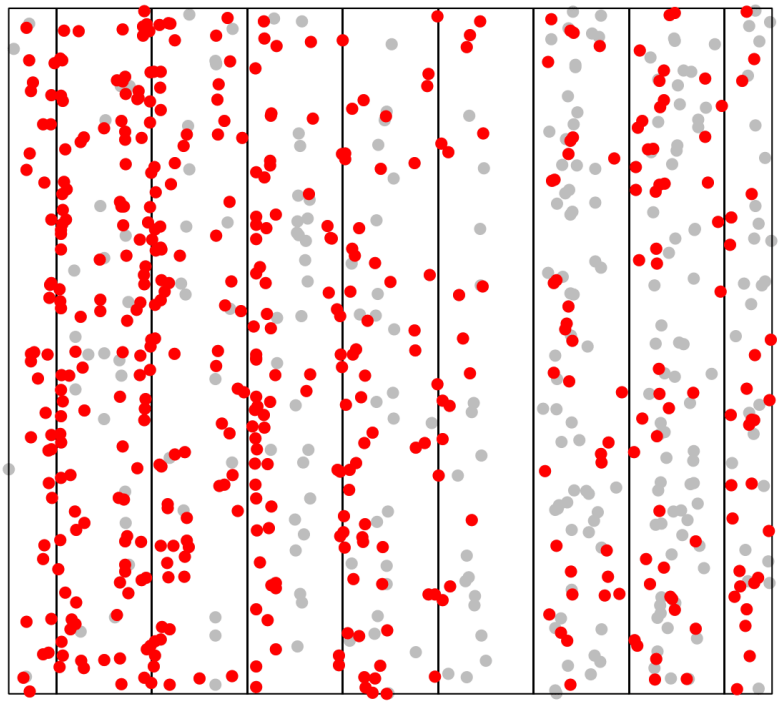


2. We design a sampling survey to collect the data along transects



3. detected points are generated from the thinned PP with intensity $\tilde{\lambda}(s) = \lambda(s)g(d(s))$

$$\therefore \log \tilde{\lambda}(s) = \overbrace{\log \lambda(s)}^{\mathbf{x}'\beta+\xi(s)} + \overbrace{\log g(d(s))}^{-0.5\, d(\mathbf{s})^2\sigma^{-2}}$$



4. The **encounter rate**, i.e. the number of observed animals within a distance W follows $m \sim \text{Poisson}\left(\int_0^W \tilde{\lambda}(d)\text{d}d\right)$

5. The pdf of detected *distances* is $\pi(d_1, \dots, d_m|m) \propto \prod_{i=1}^m \frac{\tilde{\lambda}(d_i)}{\int_0^W \tilde{\lambda}(d)\text{d}d}$ where $\tilde{\lambda}(d_i) = \lambda g(d_i)$ if we assume constant intensity with respect the distance to the observer.

If the strips width ($2W$) is narrow compared to study region (Ω) we can treat them as lines. This mean we need to define the Poisson process likelihood along the kronecker spaces (line $\times$ distance). Accounting for imperfect detection the thinned Poisson process model on (space, distance) along the transects becomes:

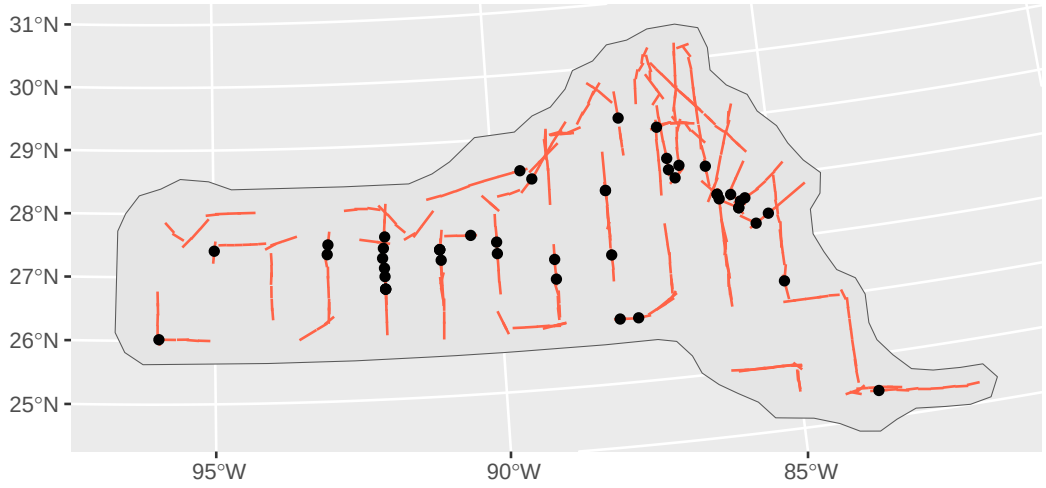
$$\log \tilde{\lambda}(s, \text{distance}) = \overbrace{\mathbf{x}'\beta + \xi(s)}^{\log \lambda(s)} + \log \mathbb{P}(\text{detection at } s|\text{distance}, \sigma) + \log(2)$$
$$\mathbb{P}(\text{detection}) = 1 - \exp\left(-\frac{\sigma}{\text{distance}}\right)$$

- Here $\log 2$ accounts for the two-sided detection.
- Typically $\mathbb{P}(distance)$ is a non-linear function, that is where `inlabru` can help via a **Fixed point iteration scheme** (further details available in this vignette)
- we define $\log(\sigma)$ as a latent Gaussian variable and iteratively linearise it.

6 Example: Dolphins in the Gulf of Mexico

In the next example, we will explore data from a combination of several NOAA shipboard surveys conducted on pan-tropical spotted dolphins in the Gulf of Mexico.

- A total of 47 observations of groups of dolphins were detected. The group size was recorded, as well as the Beaufort sea state at the time of the observation.
- Transect width is 16 km, i.e. maximal detection distance 8 km (transect half-width 8 km).

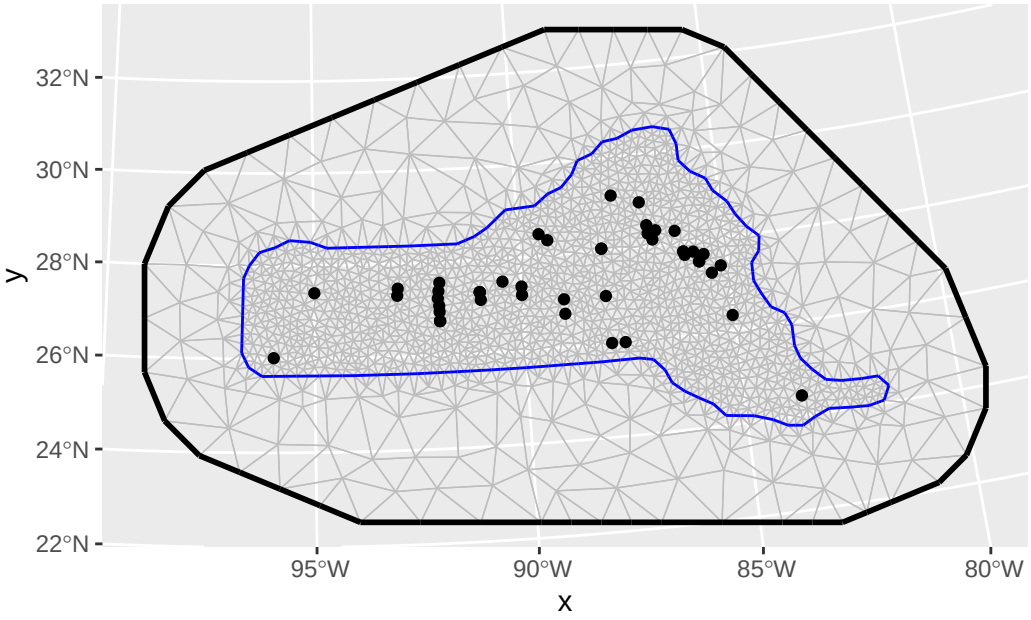


> **Note**
>
> The code and details of the following section will be discussed in greater detail in the lecture and in the lab session

First, we need to create the mesh used to approximate the random field. We can use a pre-define `sf` boundary and specify this directly into the mesh construction via the `fm_mesh_2d` function.

```
library(fmesher)

mesh_1 = fm_mesh_2d(boundary = mexdolphin$ppoly,
```

```
                    max.edge = c(30, 150),
                    cutoff = 15,
                    crs = fm_crs(mexdolphin$points))
```

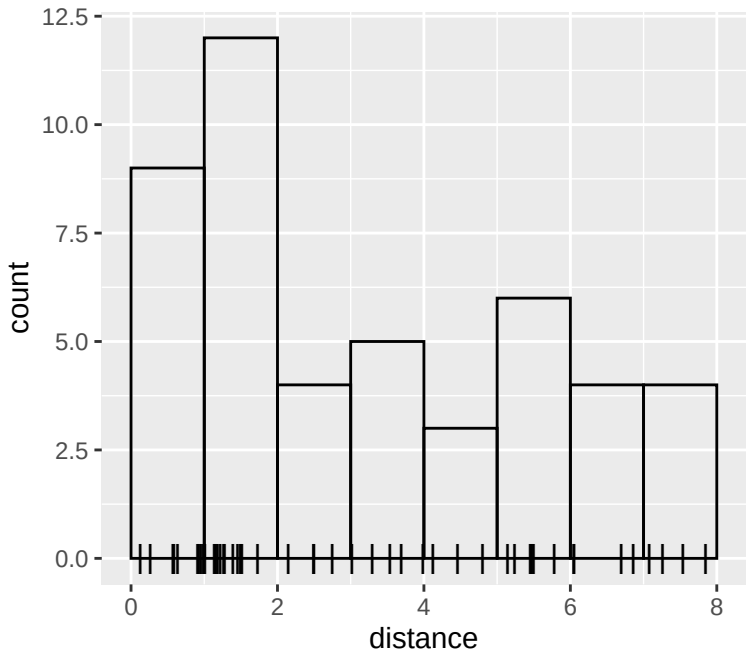


We use the `inla.spde2.pcmatern` to define the SPDE model using PC priors through the following probability statements

- $P(\rho < 50) = 0.1$
- $P(\sigma > 2) = 0.1$

```
spde_model =  inla.spde2.pcmatern(
  mexdolphin$mesh,
  prior.sigma = c(2, 0.1),
  prior.range = c(50, 0.1)
)
```

Nowe we need to define the Detection function. We start by plotting the distances and histogram of frequencies in distance intervals.



Then, we need to define a half-normal detection probability function. This must take distance as its first argument and the linear predictor of the sigma parameter as its second:

```
# define detection function
hn <- function(distance, sigma) {
  exp(-0.5 * (distance / sigma)^2)
}
```

6.1 The LGCP Model

$$p(\mathbf{y}|\lambda) \propto \exp\left(-\int_{\Omega} \lambda(\mathbf{s})p(\mathbf{s})\mathrm{d}\mathbf{s}\right) \prod_{i=1}^{n} \lambda(\mathbf{s}_i)p(\mathbf{s}_i))$$
$$\eta(s) = \beta_0 + \omega(s) + \log p(s)$$

6.2 The code

```
# build integration scheme
distance_domain <-  fm_mesh_1d(seq(0, 8,
                               length.out = 30))
ips = fm_int(list(geometry = mesh,
                  distance = distance_domain),
             samplers = mexdolphin$samplers)

# define model component
cmp = ~ Intercept(1) +
  space(main = geometry, model = spde_model) +
  sigma(1,
    prec.linear = 1,
    marginal = bru_mapper_marginal(qexp, pexp, dexp, rate = 1 / 8)
```

	mean	0.025quant	0.975quant
Intercept	-8.41	-9.47	-7.62
sigma	-0.05	-0.46	0.36
Range for space	131.74	41.79	320.28
Stdev for space	1.17	0.72	1.78

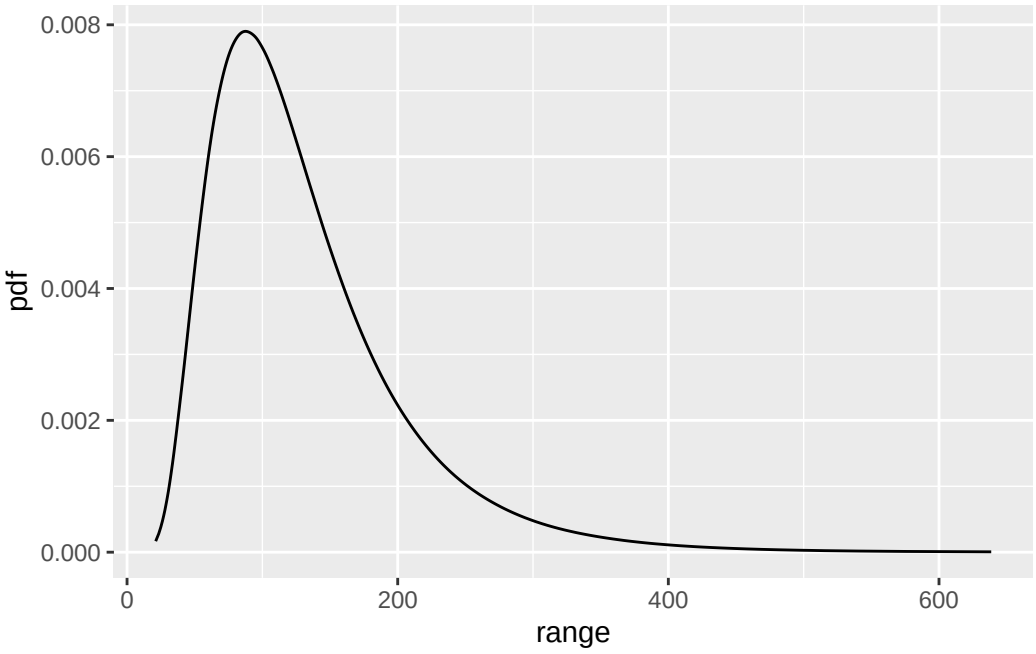
```
  )

# define model predictor
eta  = geometry + distance ~ space +
  log(hn(distance, sigma)) +
  Intercept + log(2)

# build the observation model
lik = bru_obs("cp",
              formula = eta,
              data = mexdolphin$points,
              ips = ips)
# fit the model
fit = bru(cmp, lik)
```

6.3 Results

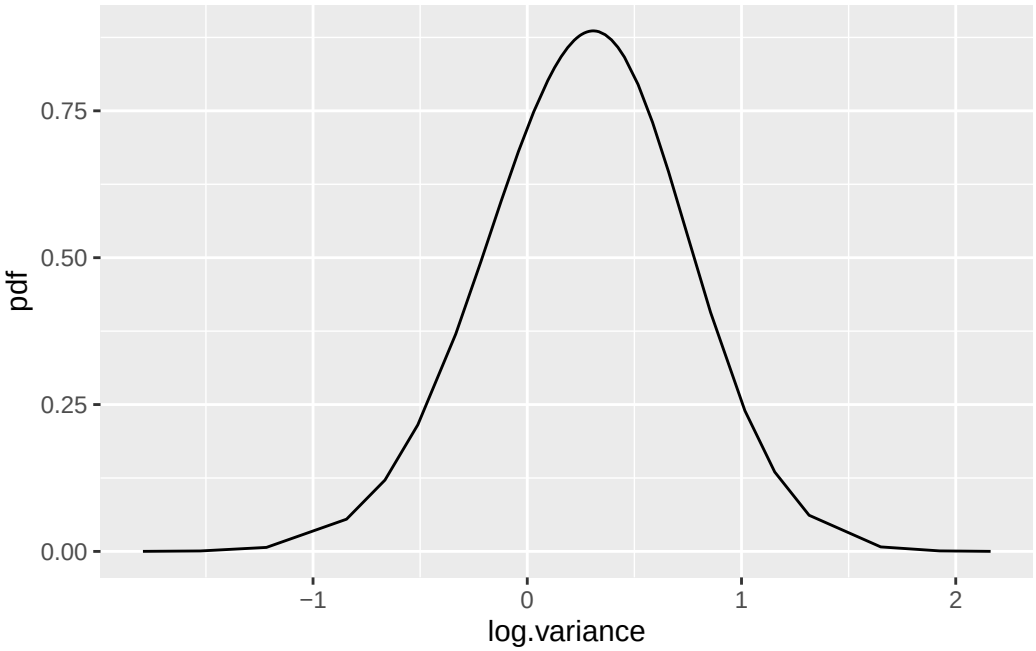
posterior summaries

We can also to plot the posterior density of the Matérn field parameters

```
spde.posterior(fit, "space", what = "range") %>% plot()
```




```
spde.posterior(fit, "space", what = "log.variance") %>% plot()
```

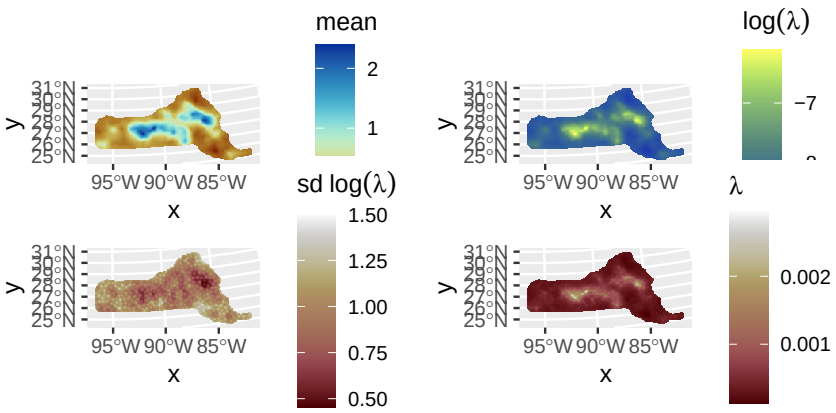


To map the spatial intensity we first need to define a grid of points where we want to predict.

- We do this using the function `fm_pixel()` which creates a regular grid of points covering the mesh
- Then, we use the `predict` function which takes as input
  - the fitted model (`fit`)
  - the prediction points (`pxl`)
  - the model components we want to predict (e.g., $e^{\beta_0+\xi(s)}$)
- To plot this you can use `ggplot` and add a `gg()` layer with your output of interest (E.g., `pr.int$spatial`)

```
library(patchwork)
pxl <- fm_pixels(mesh, dims = c(200, 100), mask = mexdolphin$ppoly)
pr.int <- predict(fit, pxl, ~ data.frame(spatial = space,
                                         loglambda = Intercept + space,
                                         lambda = exp(Intercept + space)))
```

```
ggplot() +
  gg(pr.int$spatial, geom = "tile")
```

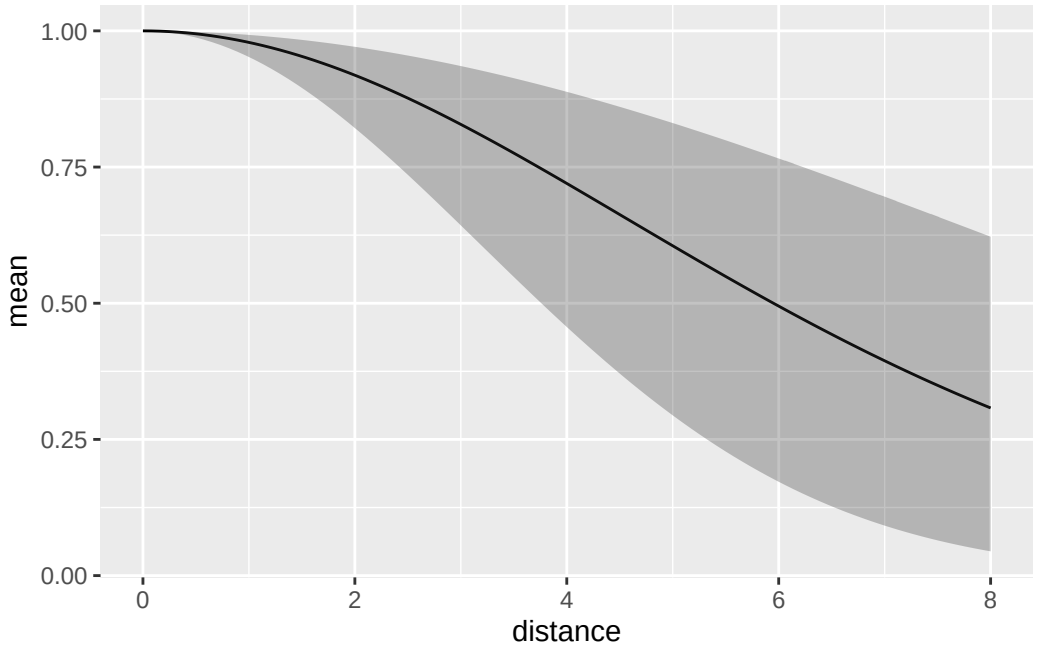

mean
2
1
sd log(λ)
1.50
1.25
1.00
0.75
0.50
log(λ)
−7
λ
0.002
0.001
y
x
31°N
30°N
29°N
28°N
27°N
26°N
25°N
95°W
90°W
85°W

mean	sd	q0.025	q0.5	q0.975	median	sd.mc_std_err	mean.mc_std_err
93.34	27.23	51.08	87.78	153.90	87.78	3.69	3.46

mean	sd	q0.025	q0.5	q0.975	median	sd.mc_std_err	mean.mc_std_err
330.57	106.82	187.42	312.41	583.07	312.41	8.57	12.40

We can also use the `predict` function to predict the detection probabilities:

```
distdf <- data.frame(distance = seq(0, 8, length.out = 100))
dfun <- predict(fit, distdf, ~ hn(distance, sigma))
plot(dfun)
```



Data level prediction* {.smaller}

47 groups were seen. How many would be seen along the transects under perfect detection?

```
predpts_transect <- fm_int(mexdolphin$mesh, mexdolphin$samplers)
Lambda_transect <- predict(fit,
                           predpts_transect,~ 16 * sum(weight * exp(space + Intercept)))
```

How many would be seen under perfect detection across the whole study area (i.e., the **mean** expected number of dolphins)?

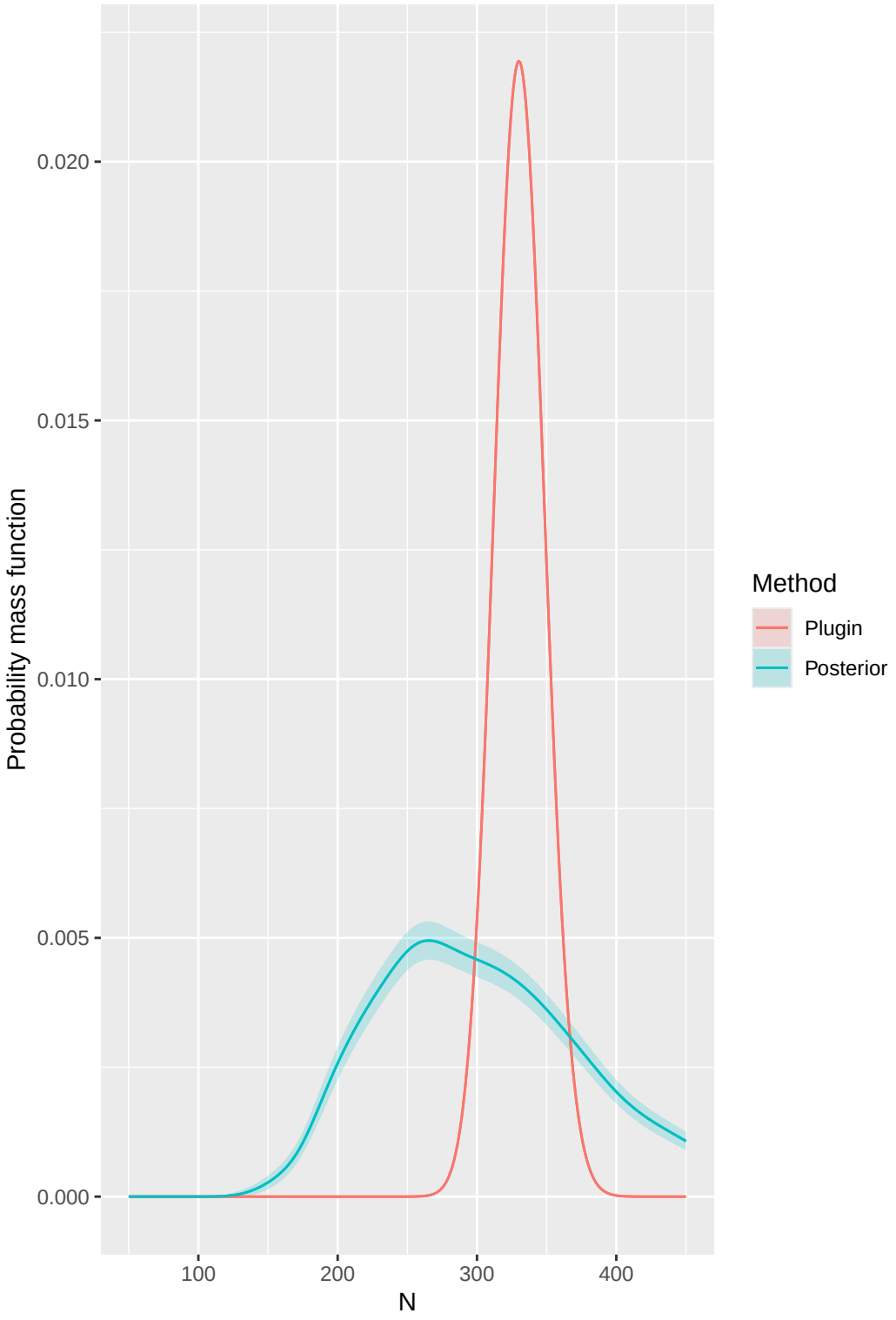
```
predpts <- fm_int(mexdolphin$mesh, mexdolphin$ppoly)
Lambda <- predict(fit, predpts, ~ sum(weight * exp(space + Intercept)))
```

expected counts

What's the predictive distribution of group counts?

We can get Monte Carlo samples for the expected number of dolphins as follows:

```
Ns <- seq(50, 450, by = 1)

Nest <- predict(fit, predpts,
  ~ data.frame(
    N = Ns,
    density = dpois(
      Ns,
      lambda = sum(weight * exp(space + Intercept))
    )
  ),
  n.samples = 2000
)

Nest <- dplyr::bind_rows(
  cbind(Nest, Method = "Posterior"),
  data.frame(
    N = Nest$N,
    mean = dpois(Nest$N, lambda = Lambda$mean),
    mean.mc_std_err = 0,
    Method = "Plugin"
  )
)
```



7 Summary of points

- Point process are a stochastic processes that describe the locations where events occur
- Unlike geostatistical data where the locations are fixed, here the locations have a stochastic nature *the locations are our data*!
- CSR as a realisation of an HPP that describe events that occur independently and uniformly at random across space, such that the number of events in any region follows

a Poisson distribution with mean $\lambda \times$ area.

- K functions can be used to distinguish between CSR, spatial clustering or regular point patterns.
- IPP allows the intensity of the point process to vary across space through spatially varying covariates.
- Numerical integration schemes are required to estimate the parameters of an IPP
- LGCP are a double stochastic process that extend IPP models by allowing the intensity function to vary spatially according to a structured spatial random effect
- Thinned Point Processes offer improved accuracy by accounting the observational process of how individuals are detected