

# PRACTICAL 8 - DISTANCE SAMPLING

## Aim of this practical:

In this practical we are going to look at distance sampling models.

## 1 Distance Sampling

In this practical we will:

- Fit a spatial distance sampling model
- Estimate animal abundance

Libraries to load:

```
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
library(sf)
# load some libraries to generate nice map plots
library(scico)
```

### 1 The data

---

In the next exercise, we will explore data from a combination of several NOAA shipboard surveys conducted on pan-tropical spotted dolphins in the Gulf of Mexico. The data set is available in `inlabru` (originally obtained from the `dsm` R package) and contains the following information:

- 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).

We can load and visualize the data as follows:

```
mexdolphins <- mexdolphins_sf
mexdolphins$depth <- mexdolphins$depth %>% mutate(depth=scale(depth)%>%c())
ggplot() + geom_sf(data = mexdolphins$points, color = "red" ) +
  geom_sf(data = mexdolphins$samplers) +
  geom_sf(data = mexdolphins$ppoly, alpha = 0)
```



## 1 The workflow

To model the density of spotted dolphins we take a thinned point process model of the form:

When fitting a distance sampling model we need to fulfill the following tasks:

1. Build the mesh
2. Define the SPDE representation of the spatial GF. This includes defining the priors for the range and sd of the spatial GF
3. Define the *components* of the linear predictor. This includes the spatial GF and all eventual covariates
4. Define the observation model using the `bru_obs()` function
5. Run the model using the `bru()` function

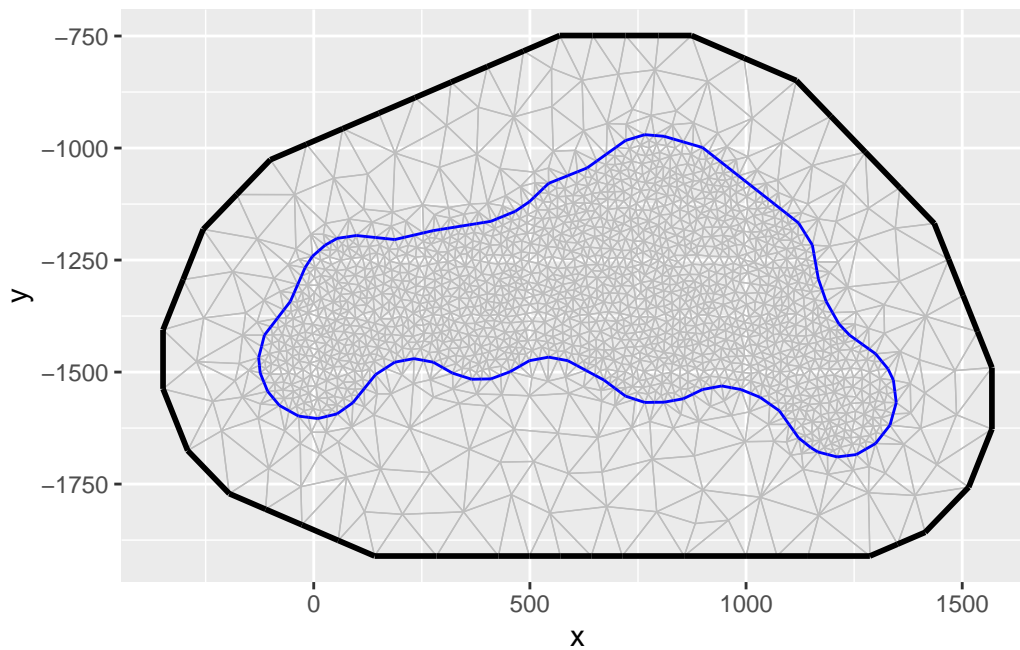
### 1.2.1 Building the mesh

The first task is to build the mesh that covers the area of interest. For this purpose we use the function `fm_mesh_2d`. To do so, we need to define the area of interest. We can either use a predefined boundary or create a non convex hull surrounding the location of the specie sightings

## 1 non-convex hull

```
boundary0 = fm_nonconvex_hull(mexdolphins$points, convex = -0.1)
mesh_0 = fm_mesh_2d(boundary = boundary0,
                    max.edge = c(30, 150), # The largest allowed triangle edge length.
                    cutoff = 15,
                    crs = fm_crs(mexdolphins$points))

ggplot() + gg(mesh_0)
```

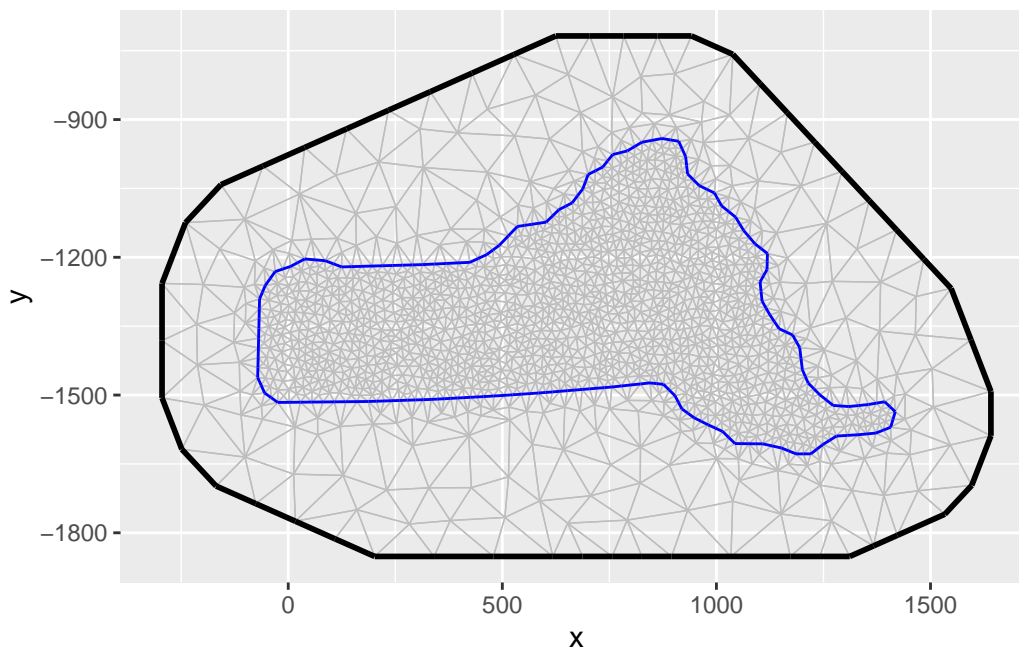


## 1 domain boundary

The `mexdolphins` object contains a predefined region of interest which can be accessed through `mexdolphins$ppoly`

```
mesh_1 = fm_mesh_2d(boundary = mexdolphins$ppoly,
  max.edge = c(30, 150),
  cutoff = 15,
  crs = fm_crs(mexdolphins$points))

ggplot() + gg(mesh_1)
```



## Task

Look at the documentation for the `fm_mesh_2d` function typing

```
?fm_mesh_2d
```

play around with the different options and create different meshes. You can compare these against a pre-computed mesh available by typing `plot(mexdolphins$mesh)`

The *rule of thumb* is that your mesh should be:

- fine enough to well represent the spatial variability of your process, but not too fine in order to avoid computation burden
- the triangles should be regular, avoid long and thin triangles.
- The mesh should contain a buffer around your area of interest (this is what is defined in the `offset` option) in order to avoid boundary artefact in the estimated variance.

### 1.4.1 Define the SPDE representation of the spatial GF

To define the SPDE representation of the spatial GF we use the function `inla.spde2.pcmatern`. This takes as input the mesh we have defined and the PC-priors definition for  $\rho$  and  $\sigma$  (the range and the marginal standard deviation of the field).

PC priors Gaussian Random field are defined in (Fuglstad et al. 2018). From a practical perspective for the range  $\rho$  you need to define two parameters  $\rho_0$  and  $p_\rho$  such that you believe it is reasonable that

$$P(\rho < \rho_0) = p_\rho$$

while for the marginal variance  $\sigma$  you need to define two parameters  $\sigma_0$  and  $p_\sigma$  such that you believe it is reasonable that

$$P(\sigma > \sigma_0) = p_\sigma$$

## Question

Take a look at the code below and select which of the following statements about the specified Matern PC priors are true.

```
spde_model <- inla.spde2.pcmatern(mexdolphins$mesh,
  prior.sigma = c(2, 0.01),
  prior.range = c(50, 0.01)
)
```

- (A) there is probability of 0.01 that the spatial range is greater or equal than 50
- (B) the probability that the spatial range is smaller than 50 is very small
- (C) the probability that the marginal standard deviation is smaller than 2 is very small

- (D) there is probability of 0.99 that the marginal standard deviation is less or equal than 2

### 1.4.2 Define the components of the linear predictor

We have now defined a mesh and a SPDE representation of the spatial GF. We now need to define the model components.

First, we need to define the detection function. Here, we will 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:

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

We need to now separately define the components of the model including the SPDE model, the Intercept, the effect of depth and the detection function parameter sigma.

```
cmp <- ~ space(main = geometry, model = spde_model) +
  sigma(1,
    prec.linear = 1,
    marginal = bm_marginal(qexp, pexp, dexp, rate = 1 / 8)
  ) +
  Intercept(1)
```

#### **i** Note

To control the prior distribution for the sigma parameter, we use a transformation mapper that converts a latent variable into an exponentially distributed variable with expectation 8 (this is a somewhat arbitrary value, but motivated by the maximum observation distance W)

The marginal argument in the sigma component specifies the transformation function taking  $N(0,1)$  to  $\text{Exponential}(1/8)$ .

The formula, which describes how these components are combined to form the linear predictor

$$\log \tilde{\lambda}(s) = \overbrace{\log \lambda(s)}^{\beta_0 + \xi(s)} + \overbrace{\log g(d(s))}^{-0.5 d(s)^2 \sigma^{-2}}$$

#### Task

Complete the code below to define the formula

```
eta <- ... + log(2)
```

[Click here to see the solution](#)

```
eta <- geometry + distance ~ space +
  log(hn(distance, sigma)) +
  Intercept + log(2)
```

Here, the  $\log(2)$  offset in the predictor takes care of the two-sided detections

### 1.4.3 Define the observation model

`inlabru` has support for latent Gaussian Cox processes through the `cp` likelihood family. To fit a point process model recall that we need to approximate the integral in using a numerical integration scheme as:

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

Thus, we first create our integration scheme using the `fm_int` function by specifying integration domains for the spatial and distance dimensions.

Here we use the same points to define the SPDE approximation and to approximate the integral in `?@eq-thinned_pp`, so that the integration weight and SPDE weights are consistent with each other. We also need to explicitly integrate over the distance dimension so we use the `fm_mesh_1d()` to create mesh over the samplers (which are the transect lines in this dataset, so we need to tell `inlabru` about the strip half-width).

```
# build integration scheme

distance_domain <- fm_mesh_1d(seq(0, 8,
                                length.out = 30))

ips = fm_int(list(geometry = mexdolphins$mesh,
                  distance = distance_domain),
             samplers = mexdolphins$samplers)
```

Now, we just need to supply the `sf` object as our data and the integration scheme `ips`:

```
lik = bru_obs("cp",
             formula = eta,
             data = mexdolphins$points,
             ips = ips)
```

Then we fit the model, passing both the components and the observational model

```
fit = bru(cmp, lik)
```

#### **i** Note

`inlabru` supports a shortcut for defining the integration points using the `domain` and `samplers` argument of `bru_obs()`. This `domain` argument expects a list of

named domains with inputs that are then internally passed to `fm_int()` to build the integration scheme. The `samplers` argument is used to define subsets of the domain over which the integral should be computed. An equivalent way to define the same model as above is:

```
lik = bru_obs(formula = eta,
              data = mexdolphins$points,
              family = "cp",
              domain = list(
                geometry = mesh,
                distance = fm_mesh_1d(seq(0, 8, length.out = 30))),
              samplers = mexdolphins$samplers)
```

## 1 Visualize model Results

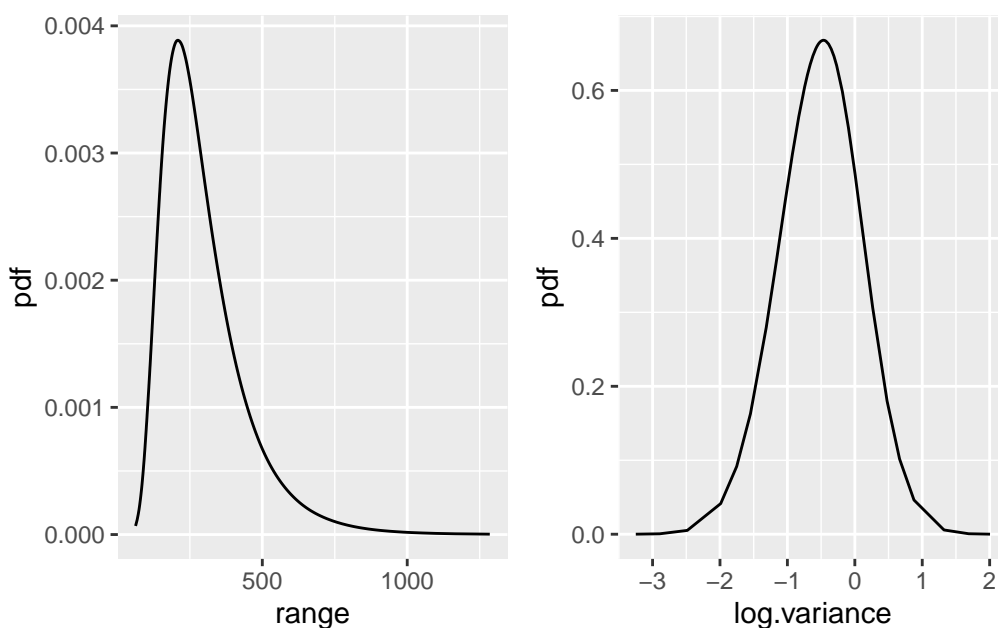
### 1.5.1 Posterior summaries

We can use the `fit$summary.fixed` and `summary.hyperpar` to obtain posterior summaries of the model parameters.

	mean	0.025quant	0.975quant
sigma	-0.05	-0.46	0.36
Intercept	-8.16	-9.29	-7.34
Range for space	295.48	110.54	673.68
Stdev for space	0.81	0.42	1.39

Look at the SPDE parameter posteriors as follows:

```
plot( spde.posterior(fit, "space", what = "range")) +
plot( spde.posterior(fit, "space", what = "log.variance"))
```



### 1.5.2 Model predictions

We now want to extract the estimated posterior mean and sd of spatial GF.

#### Task

Define a prediction grid using function `fm_pixel()`. Then compute the prediction for both the spatial GF and the linear predictor (spatial GF + intercept)

[Click here to see the solution](#)

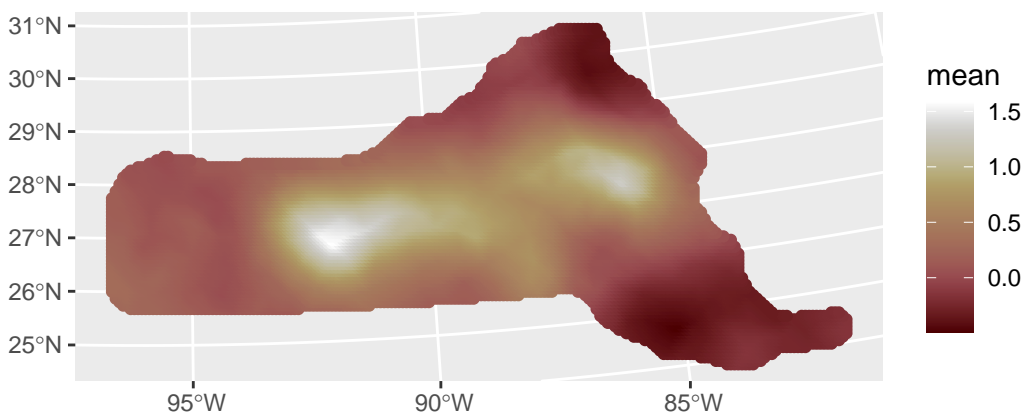
```
pxl <- fm_pixels(mexdolphins$mesh, dims = c(200, 100), mask = mexdolphins$ppoly)

pr.int = predict(fit, pxl, ~data.frame(spatial = space,
                                       lambda = exp(Intercept + space)))
```

Finally, we can plot the maps of the spatial effect

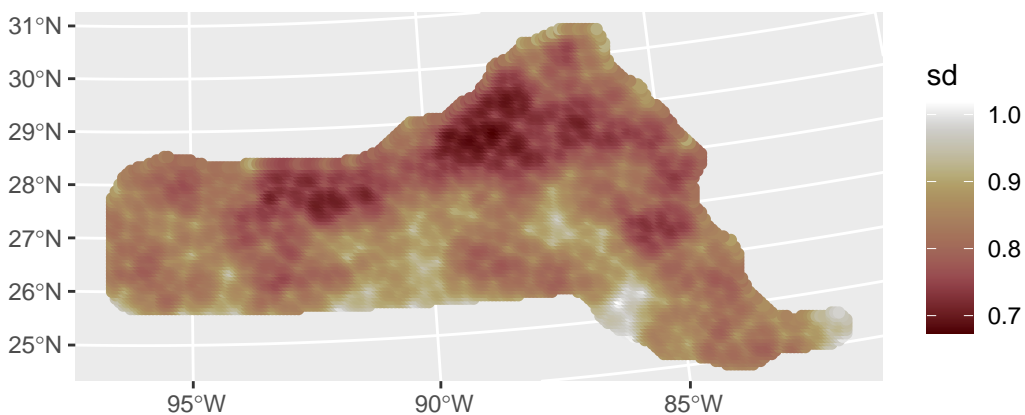
```
ggplot() + geom_sf(data = pr.int$spatial, aes(color = mean)) + scale_color_scico() + ggtitle("Posterior mean")
```

Posterior mean



```
ggplot() + geom_sf(data = pr.int$spatial, aes(color = sd)) + scale_color_scico() + ggtitle("Posterior sd")
```

Posterior sd



**Note** The posterior sd is lowest at the observation points. Note how the posterior sd is inflated around the border, this is the “border effect” due to the SPDE representation.



## Task

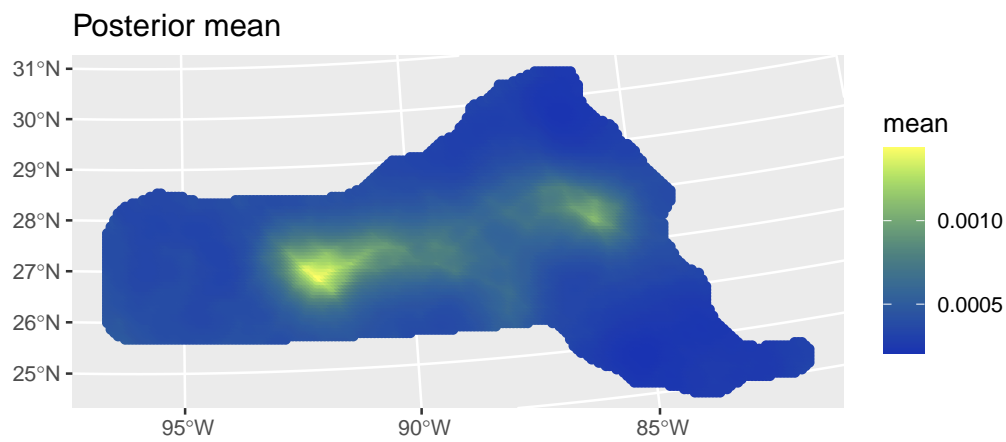
Using the predictions stored in `pr.int`, produce a map of the posterior mean intensity.

Take hint

Recall that the predicted intensity is given by  $\lambda(s) = \exp(\beta_0 + \xi(s))$

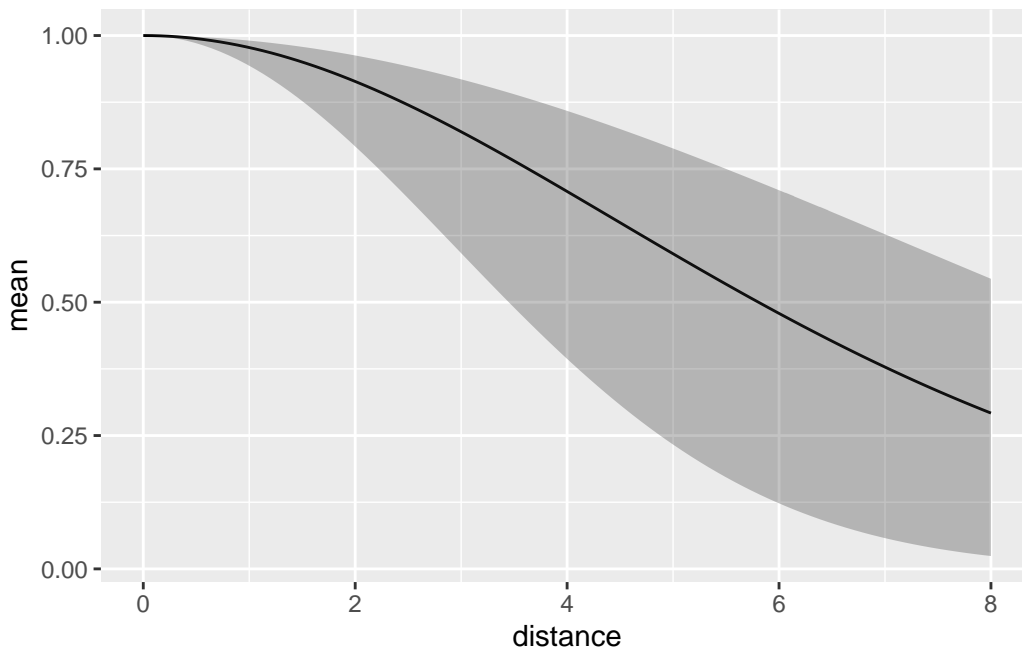
[Click here to see the solution](#)

```
ggplot() +  
  geom_sf(data = pr.int$lambda, aes(color = mean)) +  
  scale_color_scico(palette = "imola") +  
  ggtitle("Posterior mean")
```



We can predict the detection function in a similar fashion. Here, we should make sure that it doesn't try to evaluate the effects of components that can't be evaluated using the given input data.

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



## 1 Abundance estimates

The mean expected number of animals can be computed by integrating the intensity over the region of interest as follows:

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

```
      mean      sd  q0.025   q0.5   q0.975  median sd.mc_std_err
1 248.1913 42.06557 166.6362 245.355 327.5681 245.355      3.178905
mean.mc_std_err
1      4.842338
```

To fully propagate the uncertainty on the expected number animals we can draw Monte Carlo samples from the fitted model as follows (this could take a couple of minutes):

```
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
)
```

We can compare this with a simpler “plug-in” approximation:

```
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"
  )
)
```

Then, we can visualize the result as follows:

```
ggplot(data = Nest) +
  geom_line(aes(x = N, y = mean, colour = Method)) +
  geom_ribbon(
    aes(
      x = N,
      ymin = mean - 2 * mean.mc_std_err,
      ymax = mean + 2 * mean.mc_std_err,
      fill = Method,
    ),
    alpha = 0.2
  ) +
  geom_line(aes(x = N, y = mean, colour = Method)) +
  ylab("Probability mass function")
```

