PRACTICAL

Aim of this practical:

In this first practical we are going to learn how fit a point process model to transect data using `inlabru`

1 Distance Sampling

In this practical we will:

- Fit a spatial distance sampling model
- Estimate animal abundance
- Compare models that use different detection functions

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)
library(mapview)
```

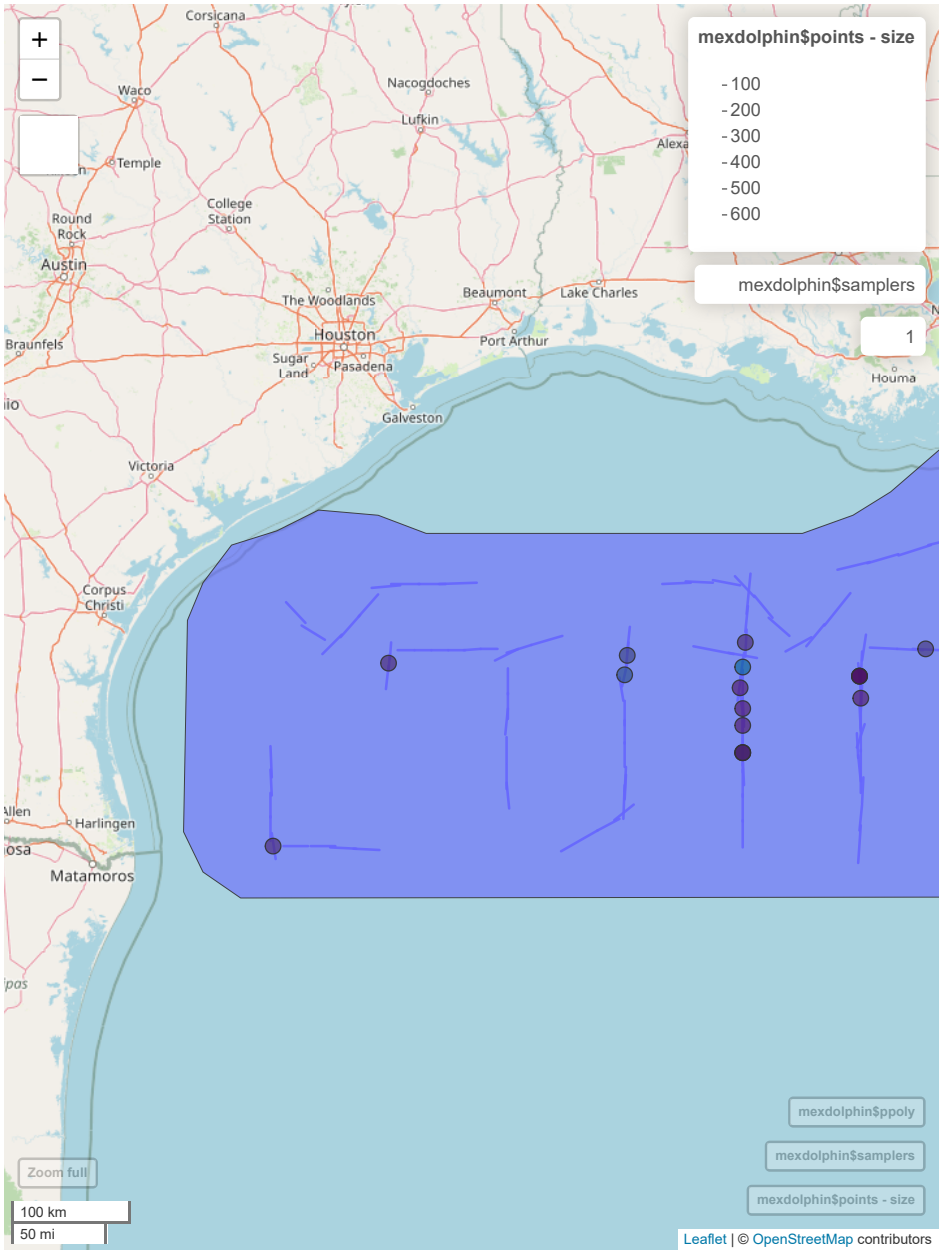
1 The data

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:

```
mexdolphin <- mexdolphin_sf
mexdolphin$depth <- mexdolphin$depth %>% mutate(depth=scale(depth)%>%c())
mapviewOptions(basemaps = c( "OpenStreetMap.DE"))

mapview(mexdolphin$points,zcol="size")+
  mapview(mexdolphin$samplers)+
 mapview(mexdolphin$ppoly )
```



1 The workflow

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

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

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

1. Build the mesh

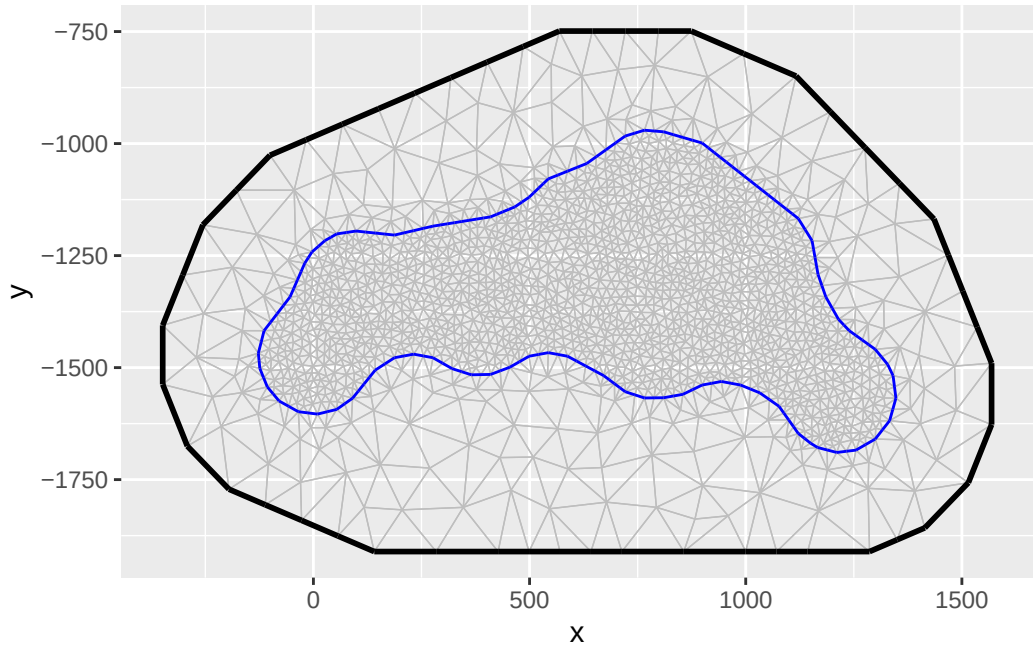
2. Define the SPDE representation of the spatial GF. This includes defining the priors for the range and sd of the spatial GF
3. Define the *components* of the linear predictor. This includes the spatial GF and all eventual covariates
4. Define the observation model using the `bru_obs()` function
5.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

1 non-covex hull

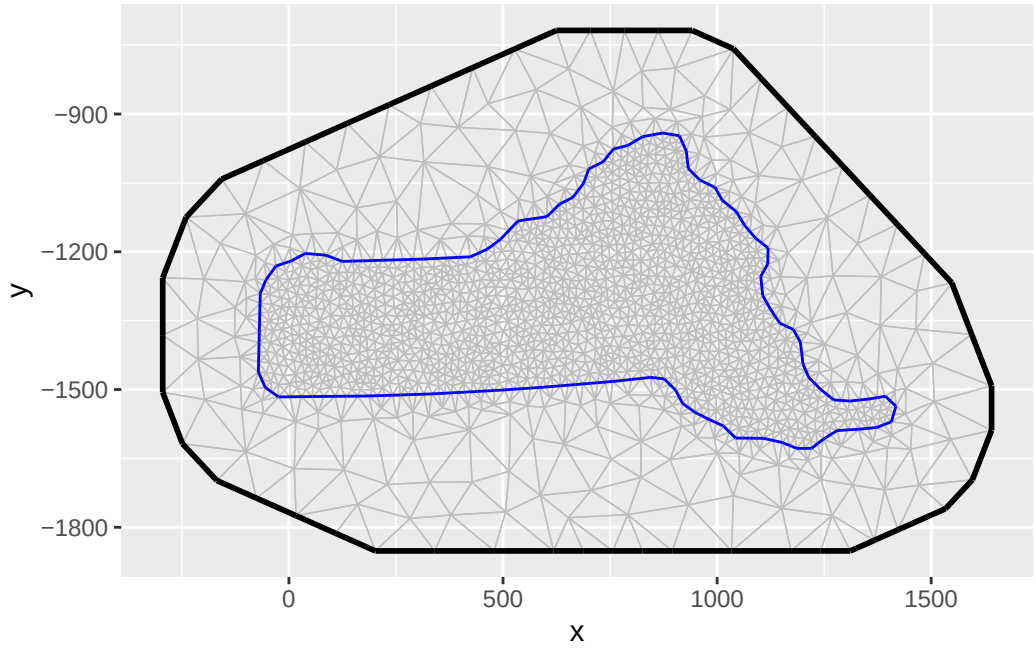
```
boundary0 = fm_nonconvex_hull(mexdolphin$points,convex = -0.1)

mesh_0 = fm_mesh_2d(boundary = boundary0,
                            max.edge = c(30, 150), # The largest allowed triangle edge length.
                            cutoff = 15,
                            crs = fm_crs(mexdolphin$points))
ggplot() + gg(mesh_0)
```



1 domain boundary

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

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



Key parameters in mesh construction include: `max.edge` for maximum triangle edge lengths, `offset` for inner and outer extensions (to prevent edge effects), and cutoff to avoid overly small triangles in clustered areas.

Note

General guidelines for creating the mesh

1. Create triangulation meshes with `fm_mesh_2d()`
2. Move undesired boundary effects away from the domain of interest by extending to a smooth external boundary
3. Use a coarser resolution in the extension to reduce computational cost (`max.edge=c(inner, outer)`)
4. Use a fine resolution (subject to available computational resources) for the domain of interest (inner correlation range) and filter out small input point clusters (0 < `cutoff` < inner)
5. Coastlines and similar can be added to the domain specification in `fm_mesh_2d()` through the `boundary` argument.

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(mexdolphin$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.

Projecting the covariate

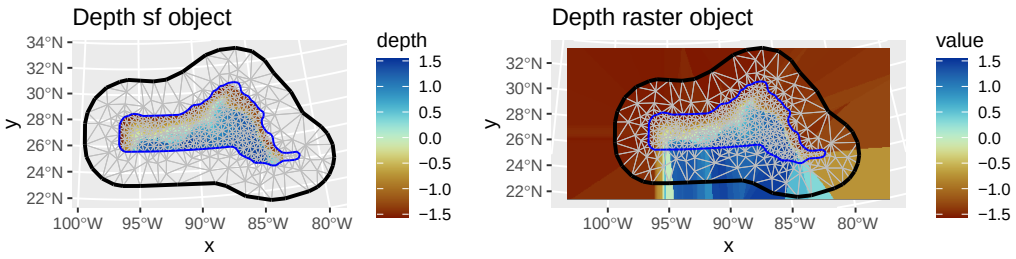
For point process models the spatial covariate has to cover the whole domain including the out mesh. To achieve this we can:

1. Convert the `sf` spatial object containing the covariate values (i.e., depth) to a raster object using the `st_rasterize` function form `stars` which can then be transformed into a `terra` raster as follows:

```
library(terra)
library(stars)
# Convert sf to stars raster
stars_raster <- st_rasterize(mexdolphin$depth[, "depth"])
# Convert stars to terra raster if needed
terra_raster <- rast(stars_raster)
```

2. Then, we can extend the raster resolution and use the `bru_fill_missing` function to fill-in the missing values with the nearest available value using the following code:

```
# Extend raster ext by 5 % of the original raster
re <- extend(terra_raster, ext(terra_raster)*2.1)
# 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$depth <- bru_fill_missing(terra_raster,re_df,re_df$depth)
# store the projectes values as a raster
depth_rast_p <- stars::st_rasterize(re_df) %>% rast()
```



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 ρ and σ (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 ρ you need to define two paramters ρ_0 and p_ρ such that you believe it is reasonable that

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

while for the marginal variance σ you need to define two parameters σ_0 and p_σ 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 Matérn PC priors are true.

```
spde_model <- inla.spde2.pcmatern(mexdolphin$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)
  ) +
  depth(depth_rast_p$depth,model="linear")+
  Intercept(1)
```

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 Exponential(1/8).

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

$$\log \tilde{\lambda}(s) = \overbrace{\log \lambda(s)}^{\beta_0+\beta_1 x(s)+\xi(s)} + \overbrace{\log g(d(s))}^{-0.5\, d(\mathbf{s})^2\sigma^{-2}}$$

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)) +
  depth +
  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 Equation 1, 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 = mexdolphin$mesh,
                  distance = distance_domain),
             samplers = mexdolphin$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 = mexdolphin$points,
              ips = ips)
```

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

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

> **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

	mean	0.025quant	0.975quant
sigma	-0.05	-0.46	0.36
depth	0.57	0.14	1.05
Intercept	-8.09	-9.28	-7.14
Range for space	344.76	122.58	813.61
Stdev for space	0.85	0.42	1.49

model as above is:

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

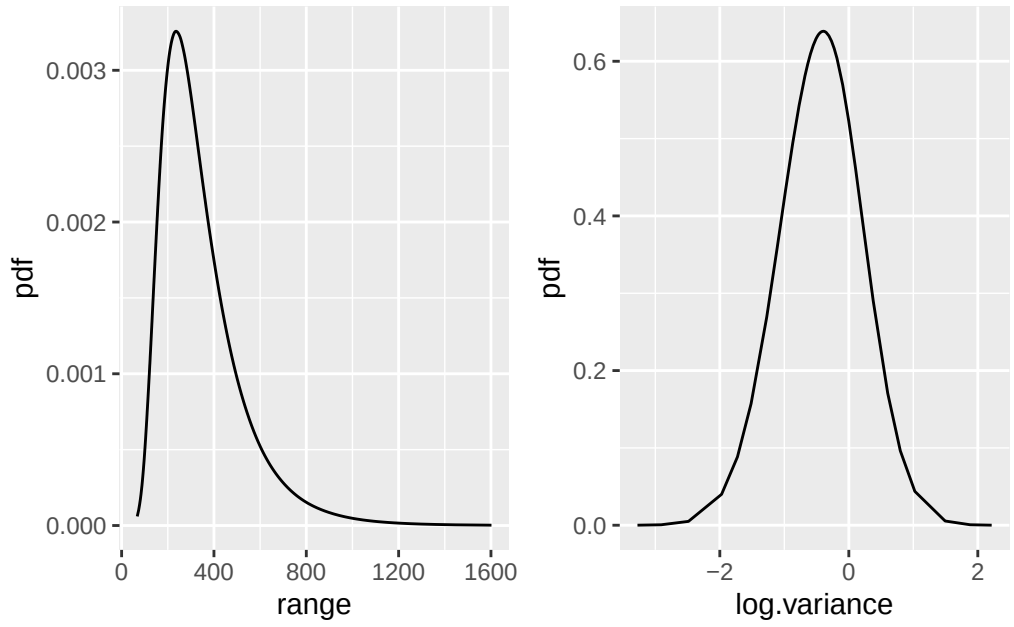
1 Visualize model Results

1.5.1 Posterior summaries

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

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. To do this 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

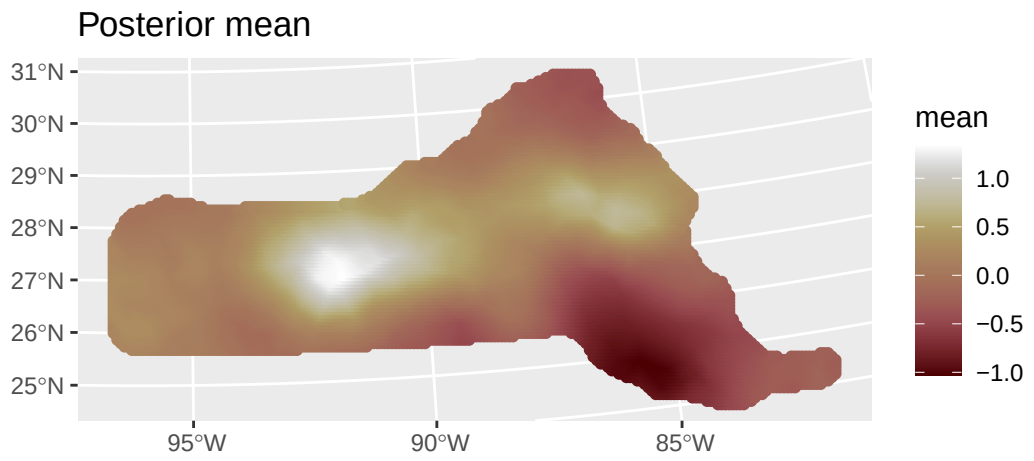
```
pxl <- fm_pixels(mexdolphin$mesh, dims = c(200, 100), mask = mexdolphin$ppoly)
```

then compute the prediction for both the spatial GF and the linear predictor (spatial GF + intercept + depth covariate)

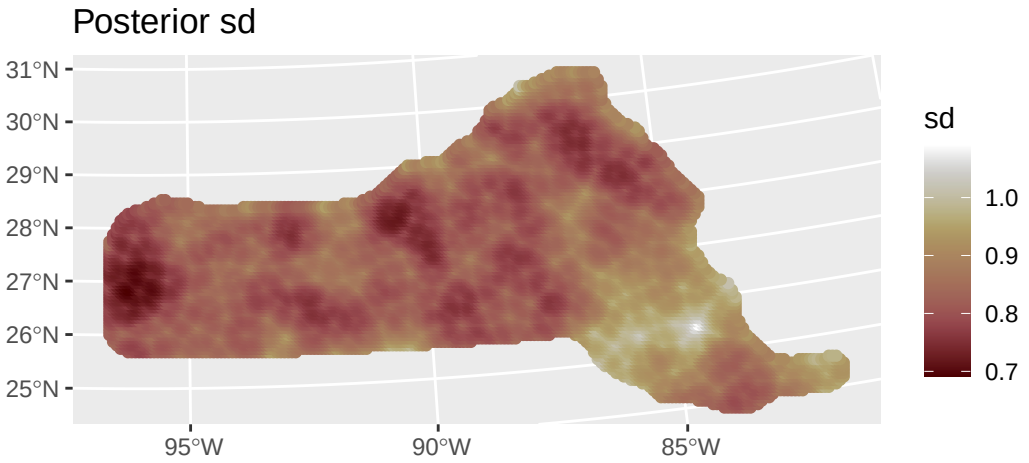
```
pr.int = predict(fit, pxl, ~data.frame(spatial = space,
                                       lambda = exp(Intercept + depth + space)))
```

Finally, we can plot the maps of the spatial effect

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




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



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

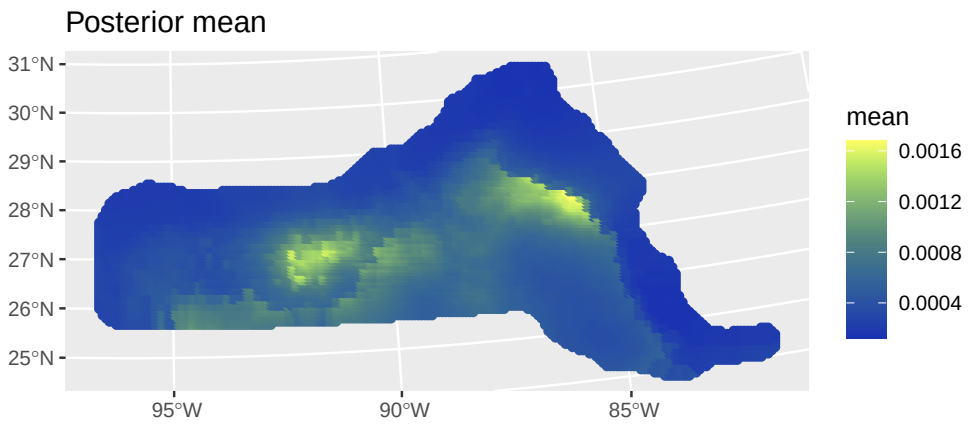
Using the predictions stored in `pr.int`, produce a map of the posterior mean intensity.

Take hint

Recall that the predicted intensity is given by $\lambda(s) = \exp\{\beta_0 + \beta_1 \text{ depth}(s) + \xi(s)\}$

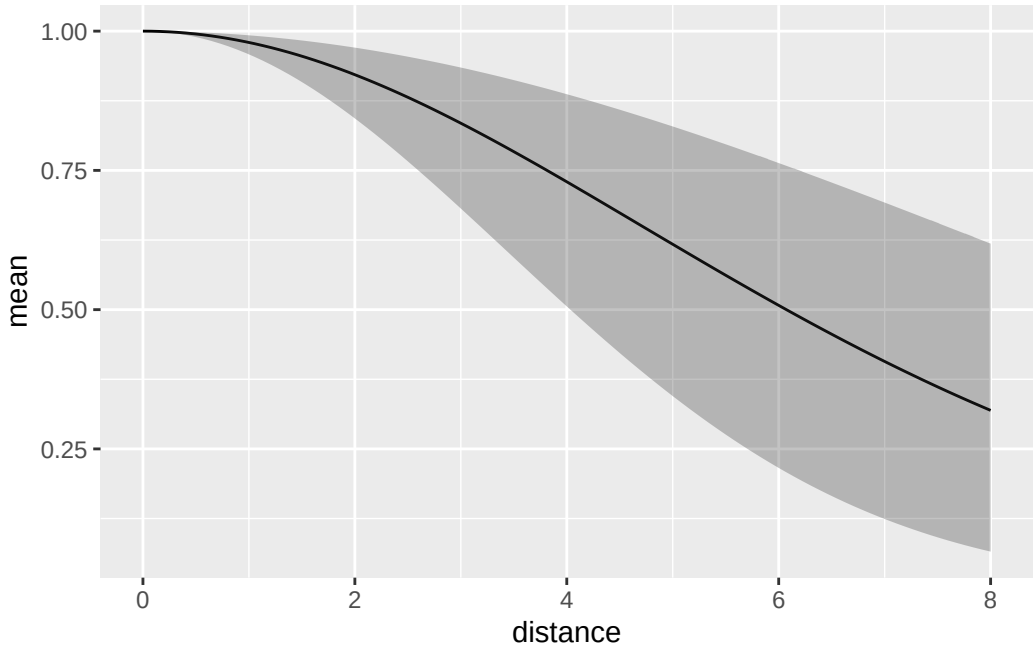
Click here to see the solution

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

```
     mean       sd   q0.025     q0.5   q0.975   median sd.mc_std_err
1 254.3822 63.79155 156.9544 244.3494 404.9458 244.3494      4.877447
  mean.mc_std_err
1        7.354645
```

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):

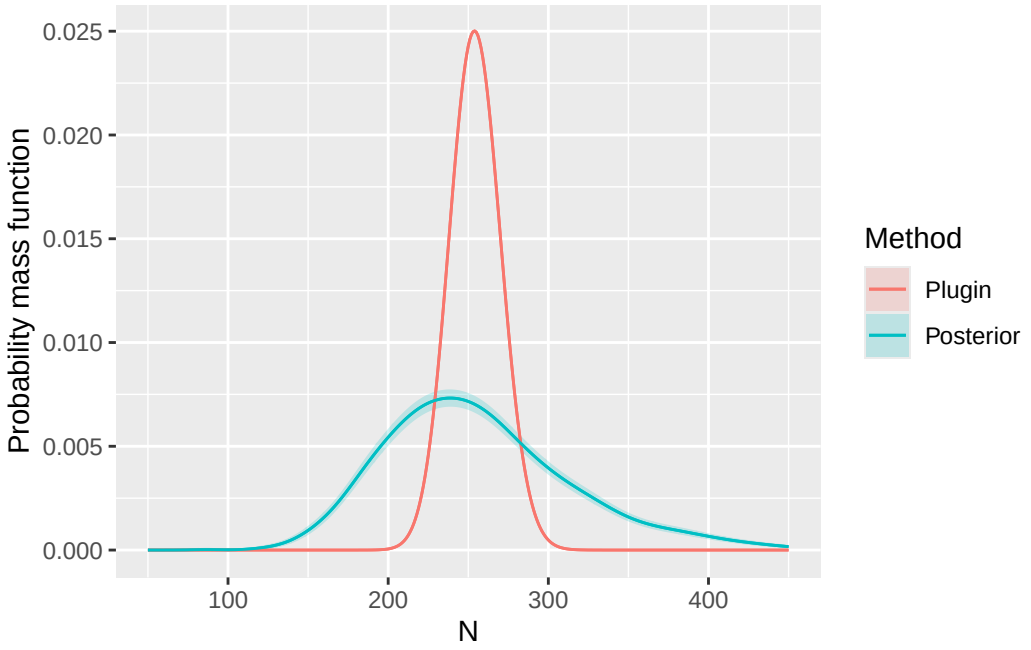
```
Ns <- seq(50, 450, by = 1)
Nest <- predict(fit, predpts,
  ~ data.frame(
    N = Ns,
    density = dpois(
      Ns,
      lambda = sum(weight * exp(space + Intercept))
    )
  ),
  n.samples = 2000
)
```

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:

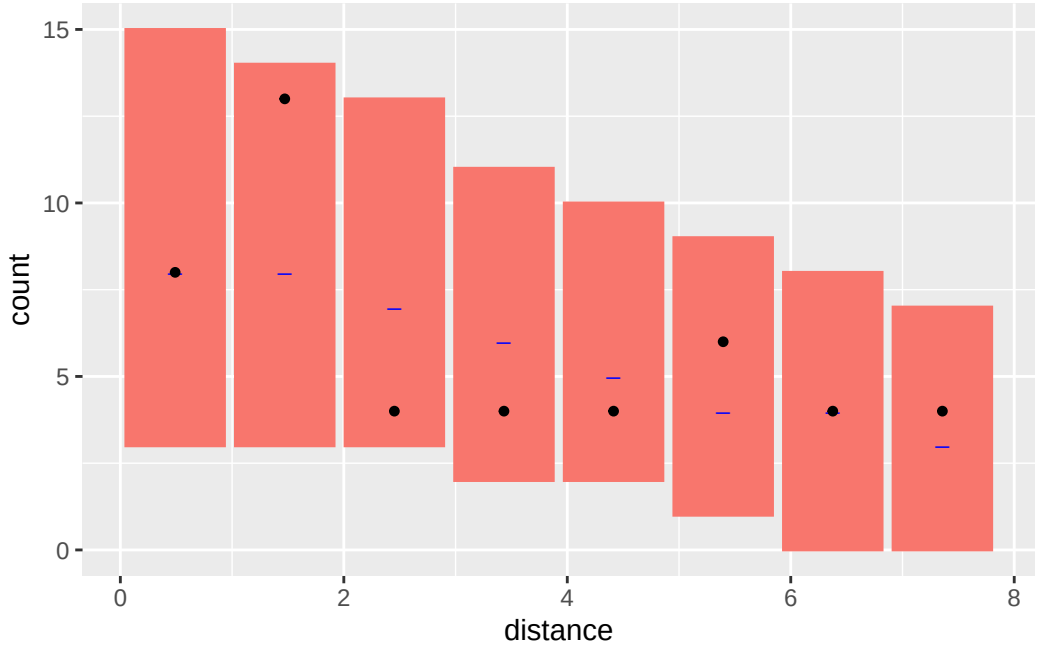
```
ggplot(data = Nest) +
  geom_line(aes(x = N, y = mean, colour = Method)) +
  geom_ribbon(
    aes(
      x = N,
      ymin = mean - 2 * mean.mc_std_err,
      ymax = mean + 2 * mean.mc_std_err,
      fill = Method,
    ),
    alpha = 0.2
  ) +
  geom_line(aes(x = N, y = mean, colour = Method)) +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



1 Model checks

Lastly, we can assess the goodness-of-fit of the models by comparing the observed counts across different distance bins and the expected counts and their associated uncertainty:

```
bc <- bincount(
  result = fit,
  observations = mexdolphin$points$distance,
  breaks = seq(0, max(mexdolphin$points$distance), length.out = 9),
  predictor = distance ~ hn(distance, sigma)
)
attributes(bc)$ggp
```



Task

Fit a model using a hazard detection function instead and compare the GoF of this model with that from the half-normal detection model. Recall that the hazard detection function is given by:

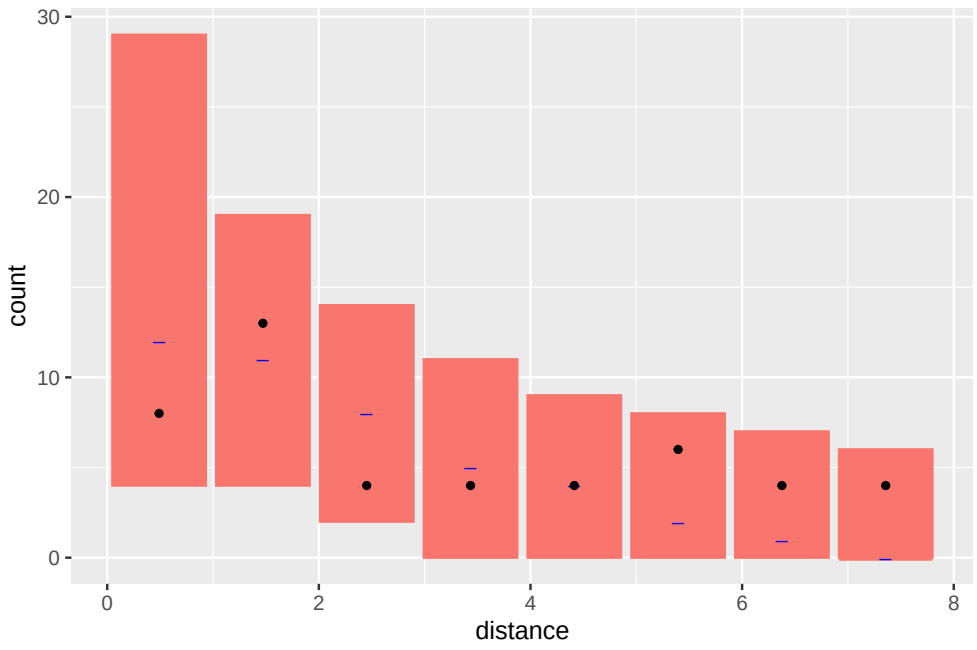
$$g(\mathbf{s}|\sigma) = 1 - \exp(-(d(\mathbf{s})/\sigma)^{-1})$$

Take hint
The hazard function can be codes as:

```
hr <- function(distance, sigma) {
  1 - exp(-(distance / sigma)^-1)
}
```

You can use the same prior for the `sigma` parameter as for the half-Normal model (such parameters aren't always comparable, but in this example it's a reasonable choice). You can also use the `lgcp` function as a shortcut to fit the model (type `?lgcp` for further details).
Click here to see the solution

```
formula1 <- geometry + distance ~ space +
  log(hr(distance, sigma)) +
  Intercept + log(2)

# here we use the shorcut to specify the model
fit1 <- lgcp(
  components = cmp,
  mexdolphin$points,
  samplers = mexdolphin$samplers,
  domain = list(
    geometry = mexdolphin$mesh,
    distance = fm_mesh_1d(seq(0, 8, length.out = 30))
  ),
  formula = formula1
)

bc1 <- bincount(
  result = fit1,
  observations = mexdolphin$points$distance,
  breaks = seq(0, max(mexdolphin$points$distance), length.out = 9),
  predictor = distance ~ hn(distance, sigma)
)
attributes(bc1)$ggp
```



2 A spacetime model for transect survey data

In this practical we will:

- Fit a spatio-temporal model to distance sampled data

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)
library(mapview)
```

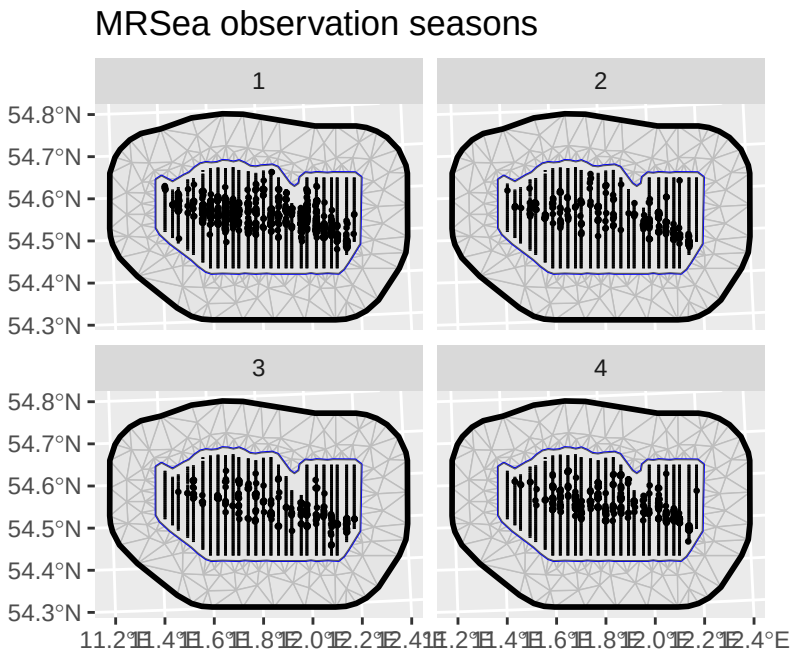
2 The data

In the next exercise, we will work with simulate spatiotemporal transect survey data from the `MRSea` package which is available on `inlabru`.

We can load and visualize the data as follows:

```
mrsea <- inlabru::mrsea

ggplot() +
  geom_fm(data = mrsea$mesh) +
```

```
  gg(mrsea$boundary) +
  gg(mrsea$samplers) +
  gg(mrsea$points, size = 0.5) +
  facet_wrap(~season) +
  ggtitle("MRSea observation seasons")
```



2 The workflow

For computational purposes we will assume perfect detection along the transects and thus, the spatiotemporal animal density is modelled using a point process model of the form:

2.2.1

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

The SPDE model

First, we define the SPDE model:

```
matern <- inla.spde2.pcmatern(mrsea$mesh,
  prior.sigma = c(0.1, 0.01),
  prior.range = c(10, 0.01)
)
```

2.2.2 Model components

In this example we will employ a spatio-temporal SPDE component. Note how the `group` and `ngroup` parameters are employed to let the SPDE model know about the name of the time dimension (season) and the total number of distinct points in time. Further control of the spatio-temporal filed can be passed as a list through the the `control.group` argument. In this case, we specify an `iid` effect to describe how the spatial field evolves in

time (this could be change to accommodate a time dependent structure such as an AR(1) model).

```
cmp <- ~ Intercept(1) +
  space_time(
    geometry,
    model = matern,
    group = season,
    ngroup = 4,
    control.group = list(model="iid")
  )
```

Task

Complete the following code to define the linear predictor:

```
  eta <- ... + ... ~ ... + ...
```

Click here to see the solution

```
  eta <- geometry + season ~ space_time +
    Intercept
```

2.2.3 Build the observational model

We can now build the integration scheme using the `fm_int()` function by specifying the domain and samplers arguments. Note that omitting the season dimension from domain would lead to aggregation of all sampling regions over time.

```
ips <- fm_int(
  domain = list(geometry = mrsea$mesh, season = 1:4),
  samplers = mrsea$samplers
)
```

Task

Complete the following arguments to construct the observational model and the fit the model using the `bru` function

```
  lik = bru_obs(formula = ...,
      family = ...,
      data = ...,
      ips  = ...)
  fit = bru(..., ...)
```

Click here to see the solution

```
  lik = bru_obs(formula = eta,
      family = "cp",
      data = mrsea$points,
      ips  = ips)
  fit = bru(cmp, lik)
```

Once the model is fitted we can predict and plot the intensity for all seasons:

```
ppxl <- fm_pixels(mrsea$mesh, mask = mrsea$boundary, format = "sf")
ppxl_all <- fm_cprod(ppxl, data.frame(season = seq_len(4)))

lambda1 <- predict(
  fit,
  ppxl_all,
  ~ data.frame(season = season, lambda = exp(space_time + Intercept))
)

pl1 <- ggplot() +
  gg(lambda1, geom = "tile", aes(fill = q0.5)) +
  gg(mrsea$points, size = 0.3) +
  facet_wrap(~season) +
  coord_sf()
pl1
```