PRACTICAL 8

Aim of this practical:

In this practical we are going to look at some model comparison and validation techniques.

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)
```

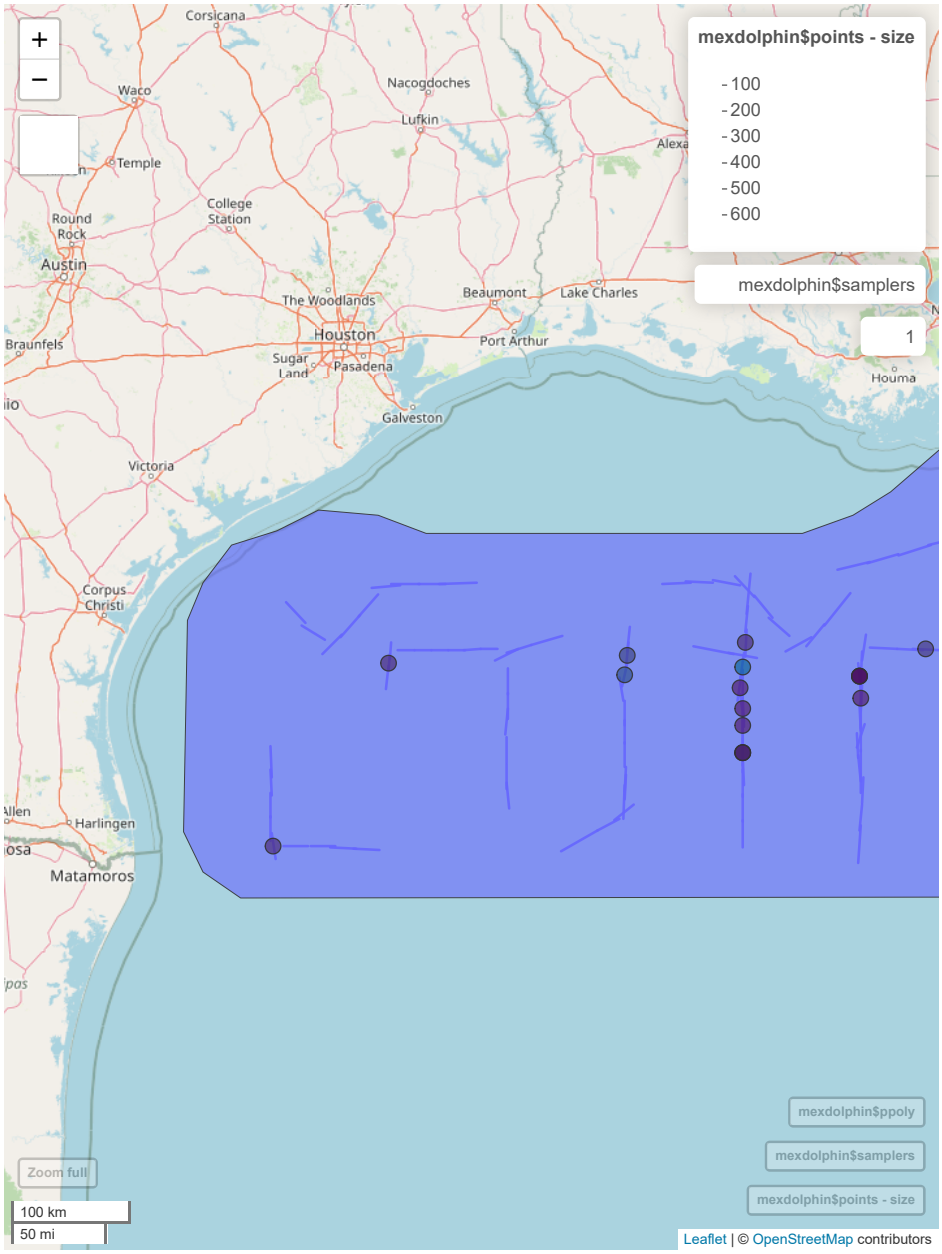
1 The data

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 obtianed 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

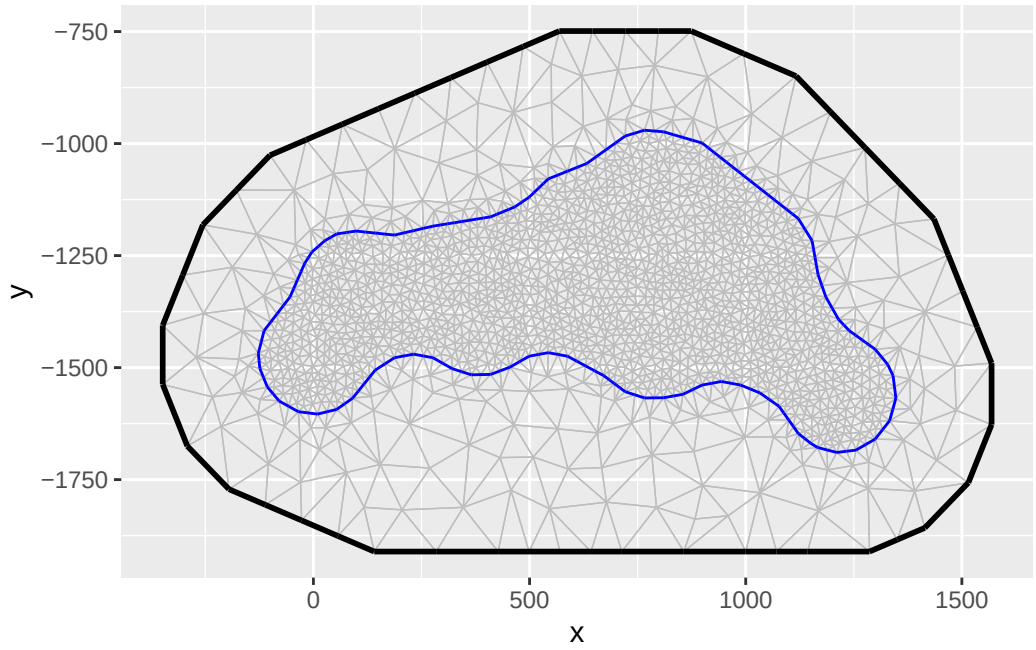
2. Define the SPDE representation of the spatial GF. This includes defining the priors for the range and sd of the spatial GF
3. Define the *components* of the linear predictor. This includes the spatial GF and all eventual covariates
4. Define the observation model using the `bru_obs()` function
5.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

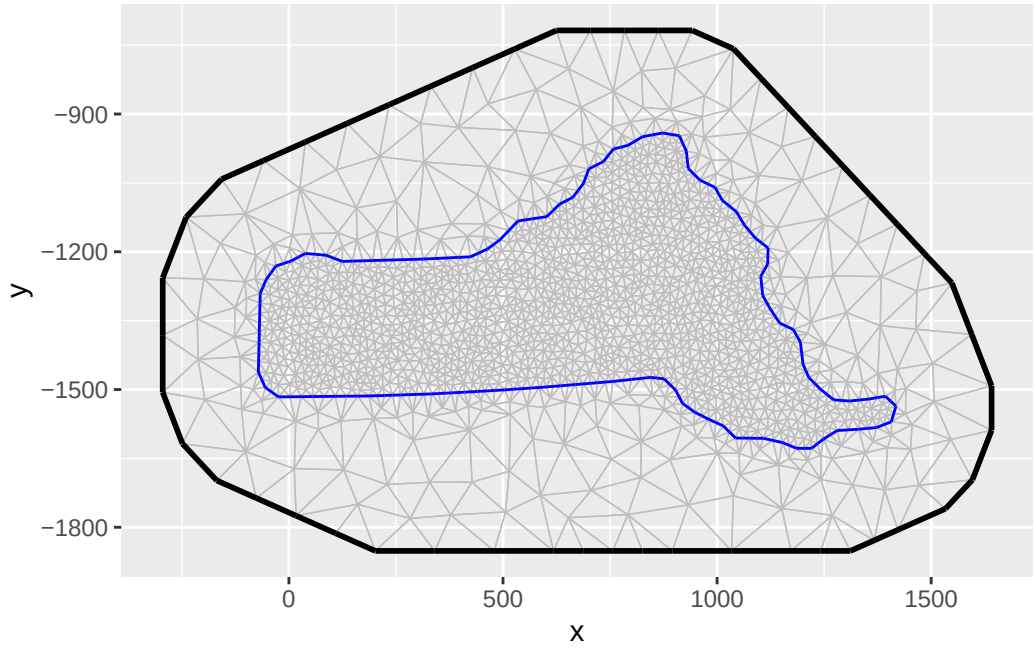
```
boundary0 = fm_nonconvex_hull(mexdolphin$points,convex = -0.1)

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.

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 Matern 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)
  ) +
  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

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

```
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 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 observaional 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 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

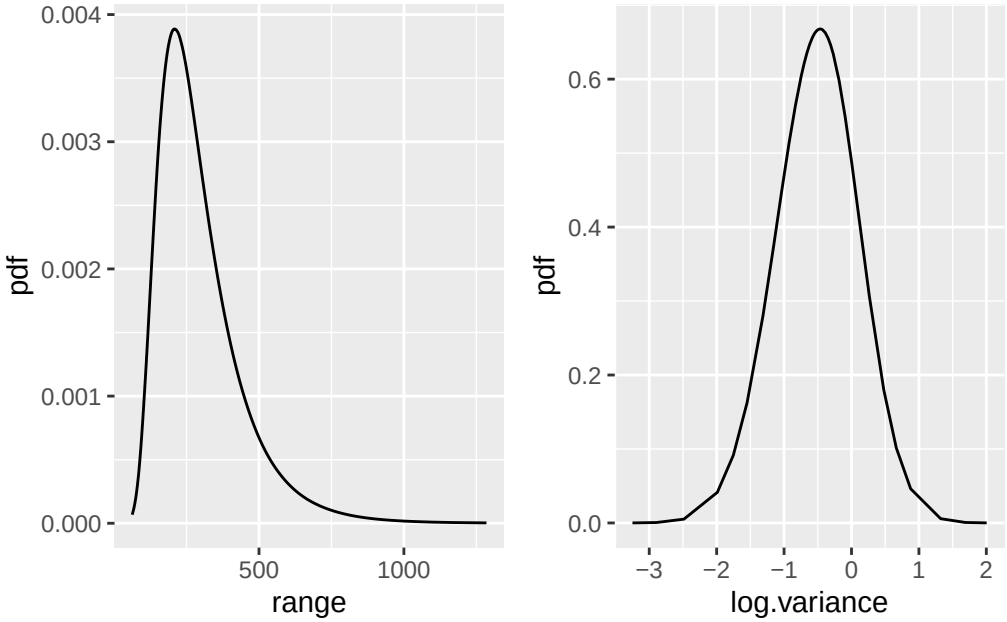
	mean	0.025quant	0.975quant
sigma	-0.05	-0.46	0.36
Intercept	-8.16	-9.29	-7.34
Range for space	295.48	110.54	673.68
Stdev for space	0.81	0.42	1.39

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

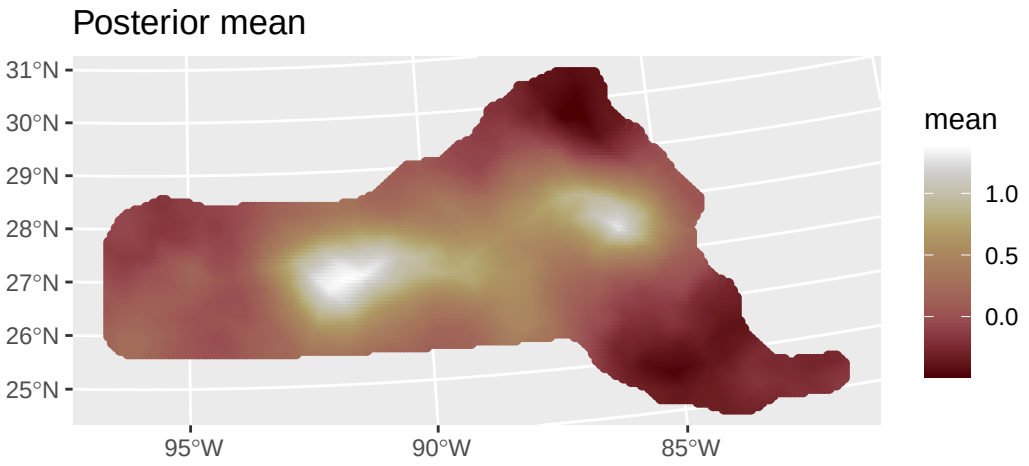
```
pxl <- fm_pixels(mexdolphin$mesh, dims = c(200, 100), mask = mexdolphin$ppoly)
```

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

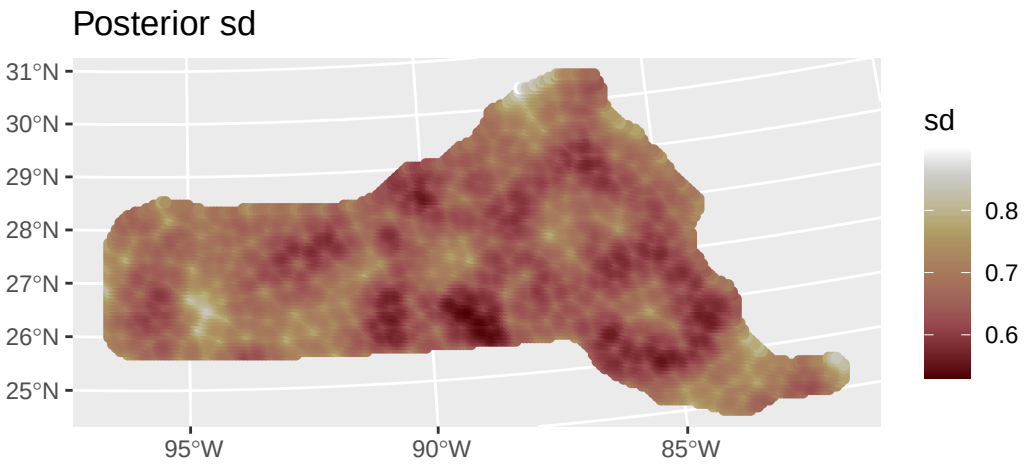
```
pr.int = predict(fit, pxl, ~data.frame(spatial = space,
                                       lambda = exp(Intercept + space)))
```

Finally, we can plot the maps of the spatial effect

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




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

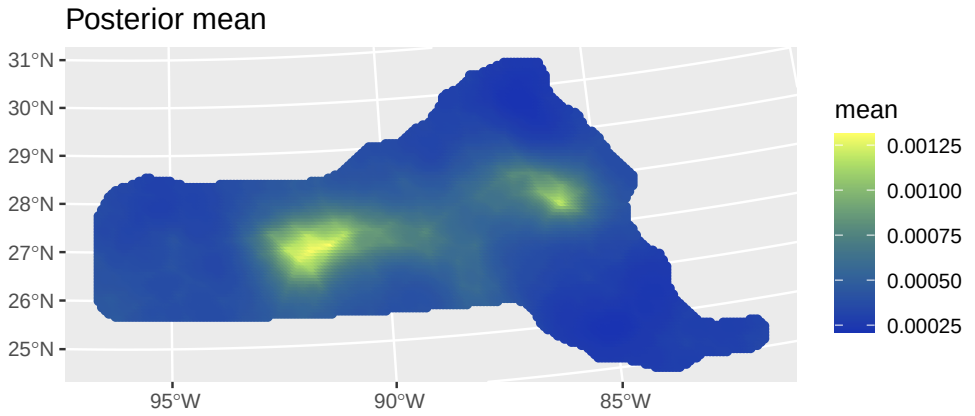


Note The posterior sd is lowest at the observation points. Note how the posterior sd is inflated around the border, this is the "border effect" due to the SPDE representation.

Task

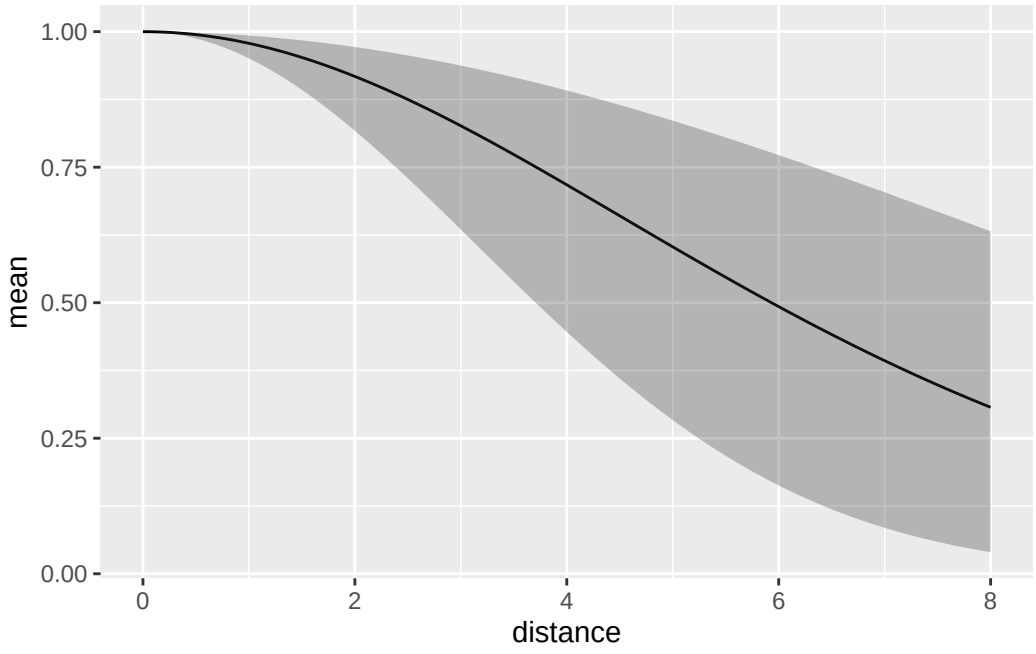
Using the predictions stored in `pr.int`, produce a map of the posterior mean intensity.
Take hint
Recall that the predicted intensity is given by $\lambda(s) = \exp(\beta_0 + \xi(s))$
Click here to see the solution

```
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
1 245.8929 49.52867 164.5492 246.5204 351.8935 246.5204      3.726218
  mean.mc_std_err
1        5.698111
```

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
)
```

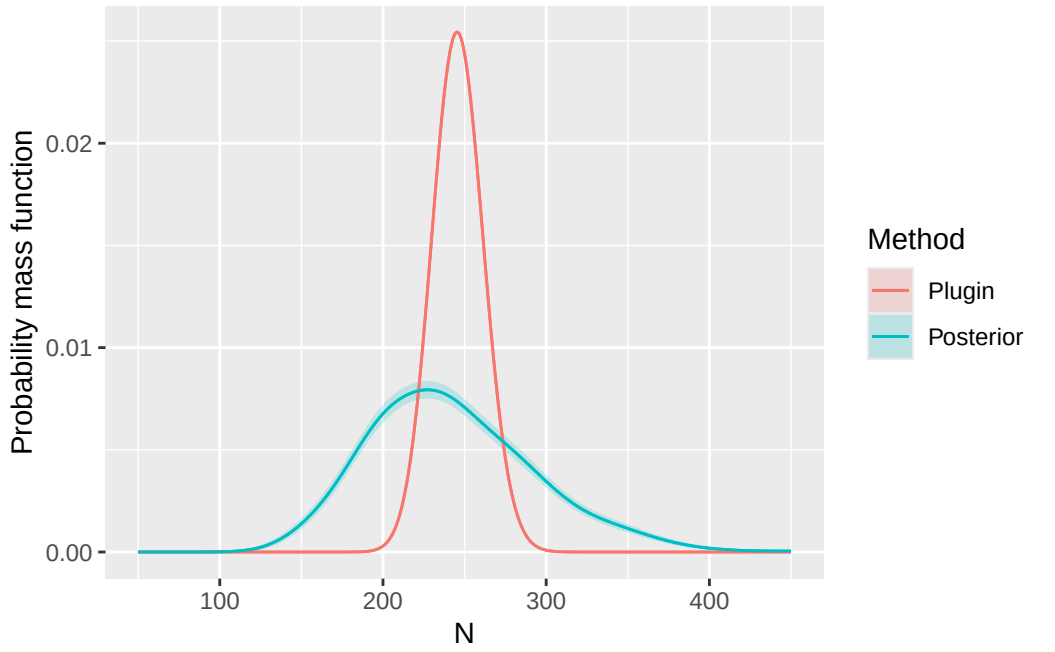
We can compare this with a simpler "plug-in" approximation:

```
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
```

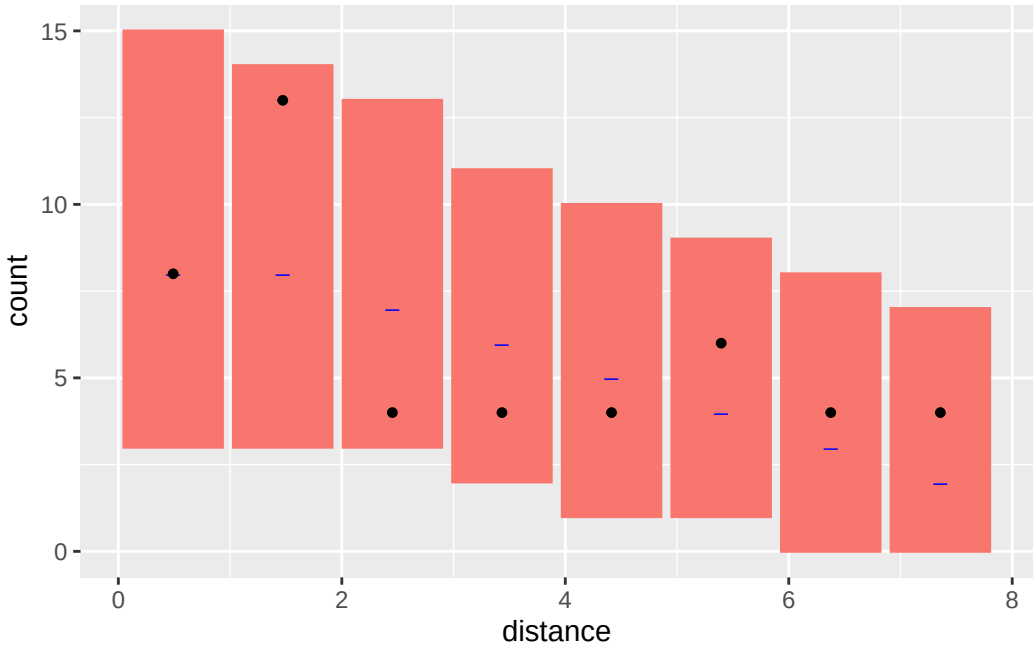
```
  ) +
  geom_line(aes(x = N, y = mean, colour = Method)) +
  ylab("Probability mass function")
```



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

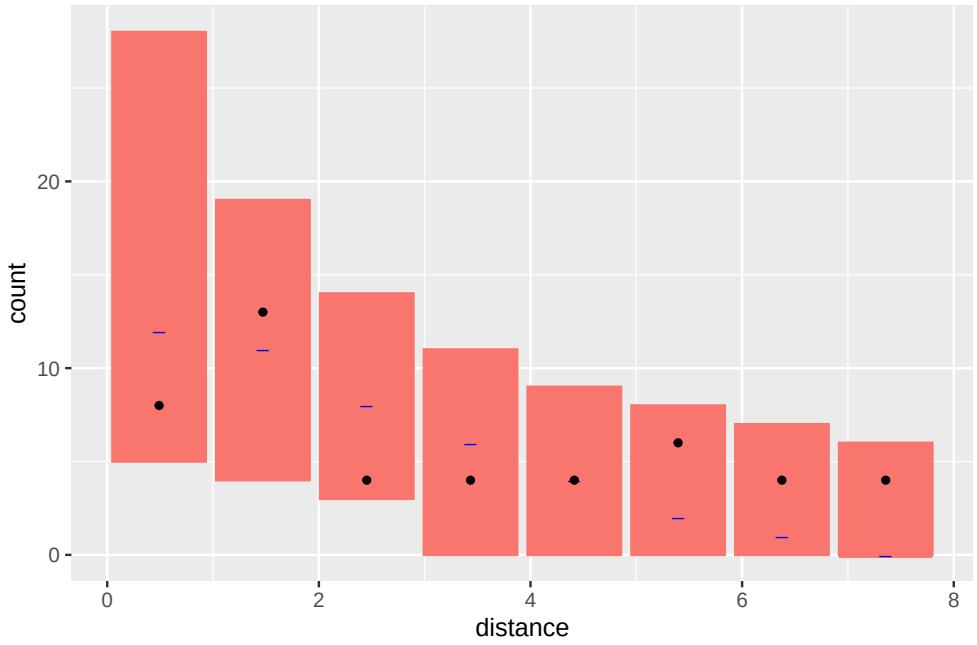
The hazard function can be codes as:

```
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
```




```
library(dplyr)
library(ggplot2)
library(inlabru)
library(INLA)
library(terra)
library(sf)
library(scico)
```

```
library(magrittr)
library(patchwork)
library(tidyterra)


# We want to obtain CPO data from the estimations
bru_options_set(control.compute = list(dic = TRUE,
                                       waic = TRUE,
                                       mlik = TRUE,
                                       cpo = TRUE))
```

In this practical we are going to work with data with excess zeros. We will

- Create count data from a `gorillas` dataset
- Fit a zero inflated model
- Fit a hurdle model
- Fit a hurdle model using two likelihoods
- Fit a hurdle model using two likelihoods and a shared component

2 Data Preparation

The following example use the `gorillas` dataset available in the `inlabru` library.

The data give the locations of Gorilla's nests in an area: ::: {.cell layout-align="center"}

```
gorillas_sf <- inlabru::gorillas_sf
nests <- gorillas_sf$nests
boundary <- gorillas_sf$boundary

ggplot() + geom_sf(data = nests) +
  geom_sf(data = boundary, alpha = 0)
```

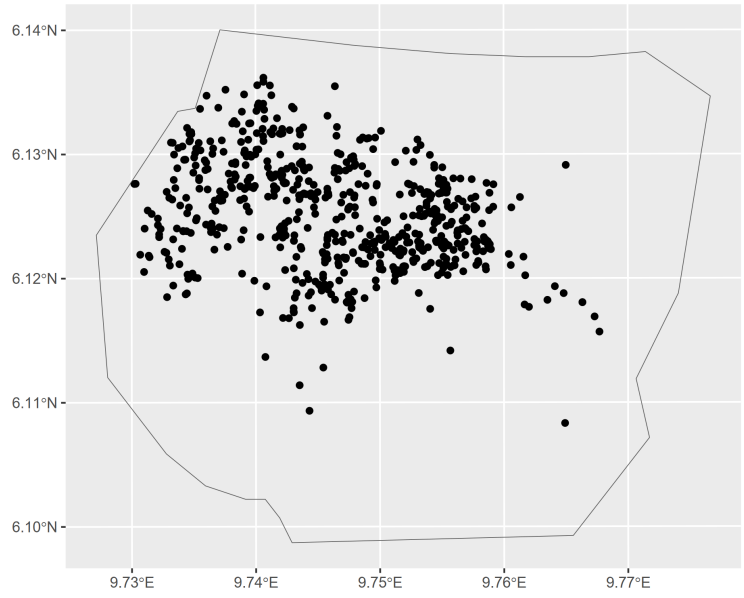


Figure 1: Location of gorilla nests

:::

The dataset also contains covariates in the form or raster data. We consider two of them here: ::: {.cell}

```
gcov = gorillas_sf_gcov()
elev_cov <- gcov$elevation
dist_cov <-  gcov$waterdist
```

:::

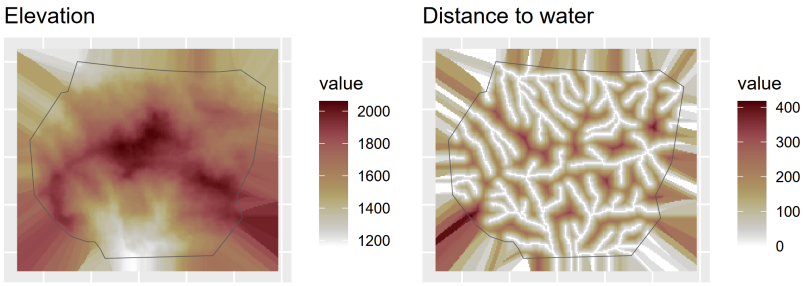


Figure 2: Covariates

Note: the covariates have been expanded to cover all the nodes in the mesh.

To obtain the count data, we rasterize the species counts to match the spatial resolution of the covariates available. Then we aggregate the pixels to a rougher resolution (5x5 pixels in the original covariate raster dimensions). Finally, we mask regions outside the study area.

In addition we compute the area of each grid cell.

```
# Rasterize data
counts_rstr <-
  terra::rasterize(vect(nests), gcov, fun = sum, background = 0) %>%
  terra::aggregate(fact = 5, fun = sum) %>%
  mask(vect(sf::st_geometry(boundary)))
plot(counts_rstr)
```

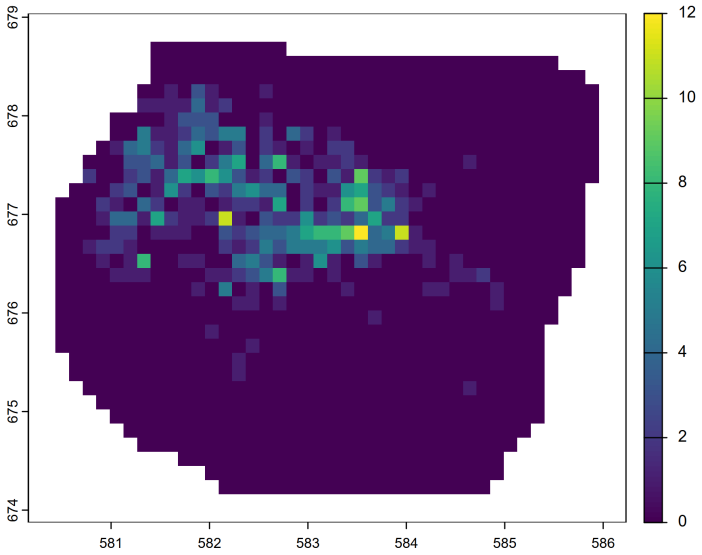


Figure 3: Counts of gorilla nests

```
# compute cell area
counts_rstr <- counts_rstr %>%
  cellSize(unit = "km") %>%
  c(counts_rstr)
```

To create our dataset of counts, we extract also the coordinate of center point of each raster pixel. In addition we create a column with presences and one with the pixel area

```
counts_df <- crds(counts_rstr, df = TRUE, na.rm = TRUE) %>%
  bind_cols(values(counts_rstr, mat = TRUE, na.rm = TRUE)) %>%
  rename(count = sum) %>%
  mutate(present = (count > 0) * 1L) %>%
  st_as_sf(coords = c("x", "y"), crs = st_crs(nests))
```

We then aggregate the covariates to the same resolution as the nest counts and scale them.

```
elev_cov1 <- elev_cov %>%
  terra::aggregate(fact = 5, fun = mean) %>% scale()
dist_cov1 <- dist_cov %>%
  terra::aggregate(fact = 5, fun = mean) %>% scale()
```

3 Mesh building

We now define the mesh and the spde object.

```
mesh <- fm_mesh_2d(
  loc = st_as_sfc(counts_df),
  max.edge = c(0.5, 1),
  crs = st_crs(counts_df)
)
```

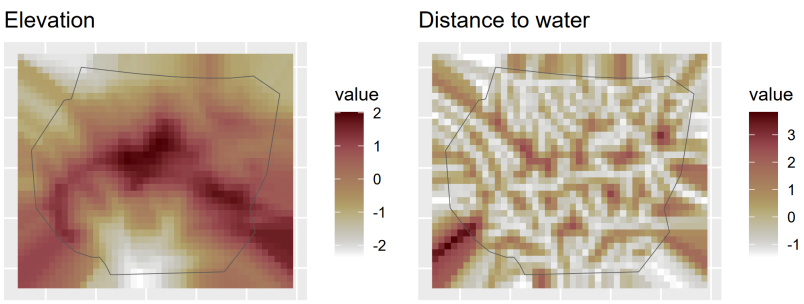


Figure 4: Covariates

```
matern <- inla.spde2.pcmatern(mesh,
  prior.sigma = c(1, 0.01),
  prior.range = c(5, 0.01)
)
```

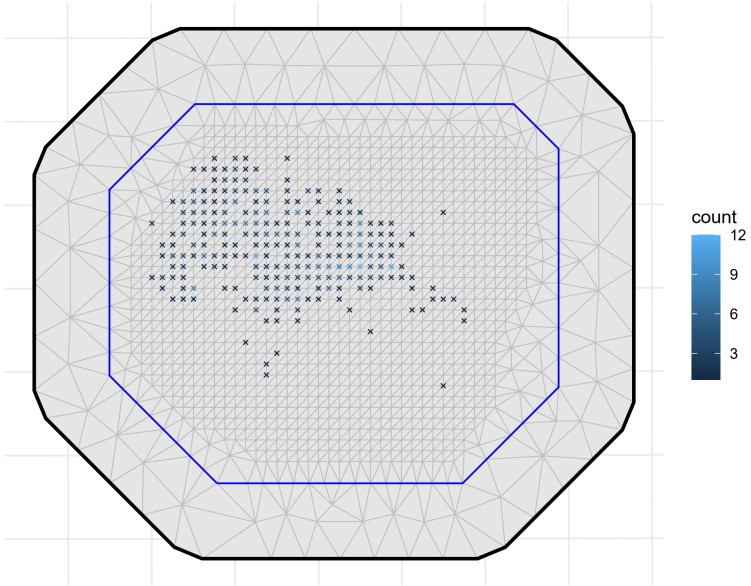


Figure 5: Mesh over the count locations

In our dataset, the number of zeros is quite substantial, and our model may struggle to account for them adequately. To address this, we should select a model capable of handling an “inflated” number of zeros, exceeding what a standard Poisson model would imply. For this purpose, we opt for a “zero-inflated Poisson model,” commonly abbreviated as ZIP.

4 Zero-Inflated model (Type1)

We fit now a Zero-Inflated model to our data.

The Type 1 Zero-inflated Poisson model is defined as follows:

$$\text{Prob}(y|\dots) = \pi \times 1_{y=0} + (1-\pi) \times \text{Poisson}(y)$$

Here, $\pi = \text{logit}^{-1}(\theta)$

The expected value and variance for the counts are calculated as:

$$\begin{aligned} E(count) &= (1-\pi)\lambda \\ Var(count) &= (1-\pi)(\lambda + \pi\lambda^2) \end{aligned} \tag{2}$$

This model has two parameters:

- The probability of excess zero π - This is a *hyperparameter* and therefore it is constant
- The mean of the Poisson distribution λ. This is linked to the linear predictor as:

$$\eta = E\,\log(\lambda) = \log(E) + \beta_0 + \beta_1 \text{Elevation} + \beta_2 \text{Distance} + u$$

where $\log(E)$ is an offset (the area of the pixel) that accounts for the size of the cell.

Task

Fit a zero-inflated model to the data (`zeroinflatedpoisson1`) by completing the following code: ::: {.cell}

```
cmp = ~ Intercept(1) + elevation(...) + distance(...) + space(...)

lik = bru_obs(...,
    E = area)

fit_zip <- bru(cmp, lik)
```

Take hint

The `E = area` is an offset that adjusts for the size of each cell.

Click here to see the solution

```
cmp = ~ Intercept(1) + elevation(elev_cov1, model = "linear") + distance(dist_cov1, model =



lik = bru_obs(formula = count ~ .,
    family = "zeroinflatedpoisson1",
    data = counts_df,
    E = area)

fit_zip <- bru(cmp, lik)
```

:::

Once the model is fitted we can look at the results

Task

Check what the estimated excess zero probaility is.
Use the `predict()` function to look at the estimated $\lambda(s)$ and mean count in Equation 2
Take hint
To get the right name for the hyperparameters to use in the `predict()` function, you can use the function `bru_names()`.
Click here to see the solution

```
# to check the estimated excess zero probability:
# fit_zip$summary.hyperpar

pred_zip <- predict(
  fit_zip,
  counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_1
    lambda <- area * exp( distance + elevation + space + Intercept)
    expect <- (1-pi) * lambda
    variance <- (1-pi) * (lambda + pi * lambda^2)
    list(
      lambda = lambda,
      expect = expect,
      variance = variance
    )
  },
  n.samples = 2500
)
```

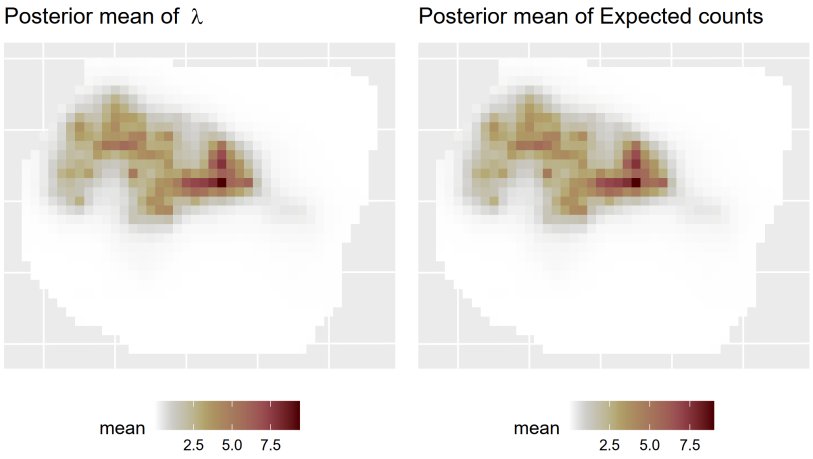


Figure 6: Estimated λ (left) and expected counts (right) with zero inflated model

5 Hurdle model (Type0)

We now fit a hurdle model to the same data.

In the `zeroinflatedpoisson0` model is defined by the following observation probability model

$$\text{Prob}(y|\dots) = \pi \times 1_{y=0} + (1-\pi) \times \text{Poisson}(y|y>0)$$

where π is the probability of zero.

The expectation and variance of the counts are as follows:

$$\begin{aligned} E(\text{count}) &= \frac{1}{1-\exp(-\lambda)}\pi\lambda \\ Var(\text{count}) &= E(\text{count})\left(1-\exp(-\lambda)E(\text{count})\right) \end{aligned} \tag{3}$$

Task

Fit a hurdle model to the data using the `zeroinflatedpoisson0` likelihood
Take hint
You do not need to redefine the components as the linear predictor is not changing.
Click here to see the solution

```
lik = bru_obs(formula = count ~ .,
    family = "zeroinflatedpoisson0",
    data = counts_df,
    E = area)

fit_zap <- bru(cmp, lik)
```

Task

As before, check what the estimated probability of zero is and use `predict()` to obtain a map of the estimated mean counts in Equation 3 over the domain.
Take hint
Click here to see the solution

```
pred_zap <- predict( fit_zap, counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_0
    lambda <- area * exp( distance + elevation + space + Intercept)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
    list(
      lambda = lambda,
      expect = expect
    )
  },
  n.samples = 2500
)
```

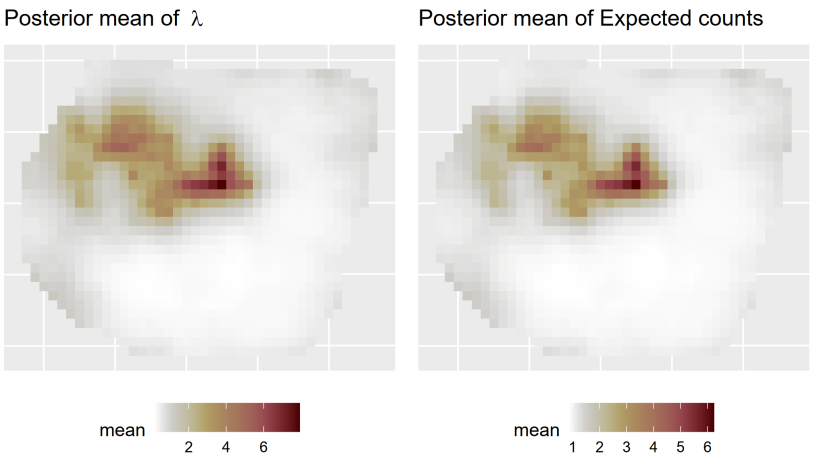


Figure 7: Estimated λ (left) and expected counts (right) with hurdle model

6 Hurdle model using two likelihoods

Here the model is the same as in Section 5, but this time we also want to model π using covariates and random effects. Therefore we define a second linear predictor

$$\eta^2 = \beta_0^2 + \beta_1^2 \text{Elevation} + \beta_2^2 \text{Distance} + u^2$$

Note here we have defined the two linear predictor to use the same covariates, but this is not necessary, they can be totally independent.

To fit this model we have to define two likelihoods: - One will account for the presence-absence process and has a Binomial model - One will account for the counts and has a truncated Poisson model

> Task
>
> Complete the following code to fit a hurdle model based on two likelihoods:

```
# define components
cmp <- ~
  Intercept_count(1) +
    elev_count(elev_cov1, model = "linear") +
    dist_count(dist_cov1, model = "linear") +
    space_count(geometry, model = matern) +
    Intercept_presence(1) +
    elev_presence(elev_cov1, model = "linear") +
    dist_presence(dist_cov1, model = "linear") +
    space_presence(geometry, model = matern)

# positive count model
pos_count_obs <- bru_obs(formula = ...,
      family = ...,
      data = counts_df[counts_df$present > 0, ],
      E = area)

# presence model
presence_obs <- bru_obs(formula ...,
  family = ...,
  data = counts_df,
)

# fit the model
fit_zap2 <- bru(...)
```

Take hint
Add hint details here…
Click here to see the solution

```
cmp <- ~
  Intercept_count(1) +
    elev_count(elev_cov1, model = "linear") +
    dist_count(dist_cov1, model = "linear") +
    space_count(geometry, model = matern) +
    Intercept_presence(1) +
    elev_presence(elev_cov1, model = "linear") +
    dist_presence(dist_cov1, model = "linear") +
    space_presence(geometry, model = matern)


pos_count_obs <- bru_obs(formula = count ~ Intercept_count + elev_count +
                                   dist_count + space_count,
      family = "nzpoisson",
      data = counts_df[counts_df$present > 0, ],
      E = area)


presence_obs <- bru_obs(formula = present ~ Intercept_presence + elev_presence + dist_pres
                          space_presence,
  family = "binomial",
  data = counts_df,
)

fit_zap2 <- bru(
  cmp,
  presence_obs,
  pos_count_obs
)
```

7 Hurdle model using two likelihoods and a shared component

Note that in the model above, there is no direct link between the parameters of the two observation parts, and we could estimate them separately. However, the two likelihoods could share some of the components; for example the `space_count` component could be used for both predictors. This would be possible using the `copy` argument.

We would then need to define one component as `space(geometry, model = matern)` and then a copy of it as `space_copy(geometry, copy = "space", fixed = FALSE)`.

The results from the model in **?@sec-sec-two-lik** show that the estimated covariance parameters for the two fields are very different, so it is probably not sensible to share the same component between the two parts. We do it anyway to show an example:

```
cmp <- ~
  Intercept_count(1) +
    elev_count(elev_cov1, model = "linear") +
    dist_count(dist_cov1, model = "linear") +
```

```
    Intercept_presence(1) +
    elev_presence(elev_cov1, model = "linear") +
    dist_presence(dist_cov1, model = "linear") +
    space(geometry, model = matern) +
  space_copy(geometry, copy = "space", fixed = FALSE)


pos_count_obs <- bru_obs(formula = count ~ Intercept_count + elev_count + dist_count + space
      family = "nzpoisson",
      data = counts_df[counts_df$present > 0, ],
      E = area)

presence_obs <- bru_obs(formula = present ~ Intercept_presence + elev_presence + dist_preser
  family = "binomial",
  data = counts_df)

fit_zap3 <- bru(
  cmp,
  presence_obs,
  pos_count_obs)
```

8 Comparing models

We have fitted four different models. Now we want to compare them and see how they fit the data.

8 Comparing model predictions

We first want to compare the estimated surfaces of expected counts. To do this we want to produce the estimated expected counts, similar to what we did in Section 4 and Section 5 for all four models and plot them together:

```
pred_zip <- predict(
  fit_zip,
  counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_1
    lambda <- area * exp( distance + elevation + space + Intercept)
    expect <- (1-pi) * lambda
    variance <- (1-pi) * (lambda + pi * lambda^2)
    list(
      expect = expect
    )
  },n.samples = 2500)

pred_zap <- predict( fit_zap, counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_0
    lambda <- area * exp( distance + elevation + space + Intercept)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
```

```
    list(
      expect = expect)
  },n.samples = 2500)

inv.logit = function(x) (exp(x)/(1+exp(x)))

pred_zap2 <- predict( fit_zap2, counts_df,
  ~ {
    pi <- inv.logit(Intercept_presence + elev_presence + dist_presence + space_presence)
    lambda <- area * exp( dist_count + elev_count + space_count + Intercept_count)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
    list(
      expect = expect)
  },n.samples = 2500)

pred_zap3 <- predict( fit_zap3, counts_df,
  ~ {
    pi <- inv.logit(Intercept_presence + elev_presence + dist_presence + space_copy)
    lambda <- area * exp( dist_count + elev_count + space + Intercept_count)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
    list(
      expect = expect)
  },n.samples = 2500)



  data.frame(x = st_coordinates(counts_df)[,1],
             y = st_coordinates(counts_df)[,2],
    zip = pred_zip$expect$mean,
         hurdle = pred_zap$expect$mean,
         hurdle2 = pred_zap2$expect$mean,
         hurdle3 = pred_zap3$expect$mean)  %>%
  pivot_longer(-c(x,y)) %>%
  ggplot() + geom_tile(aes(x,y, fill = value)) + facet_wrap(.~name) +
    theme_map + scale_fill_scico(direction = -1)
```

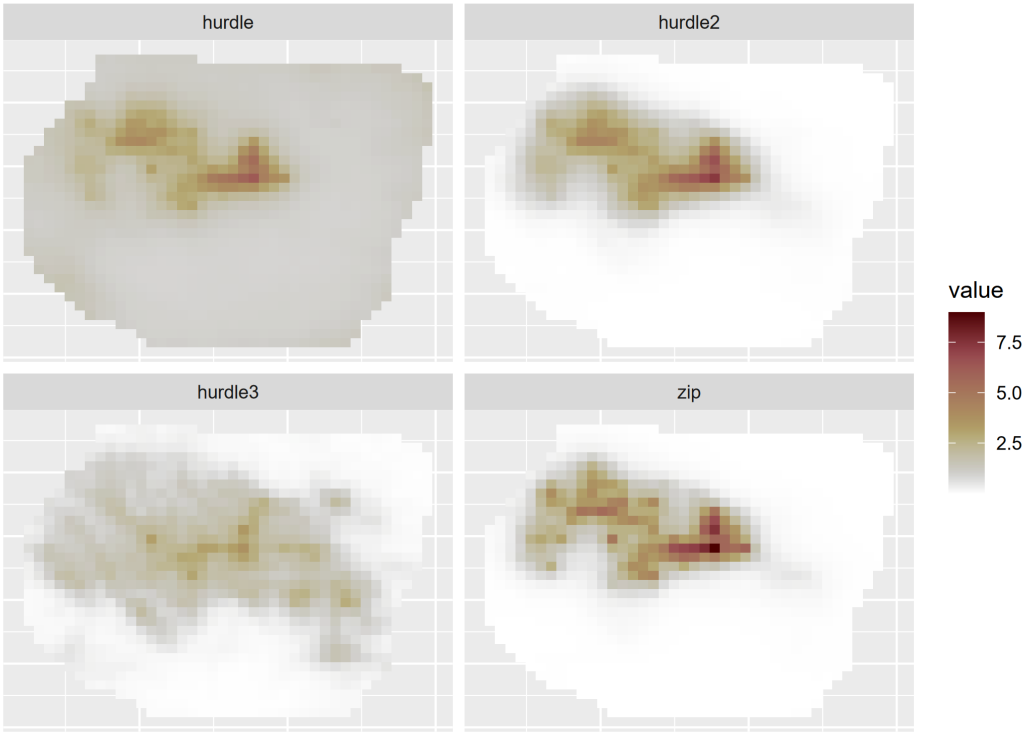
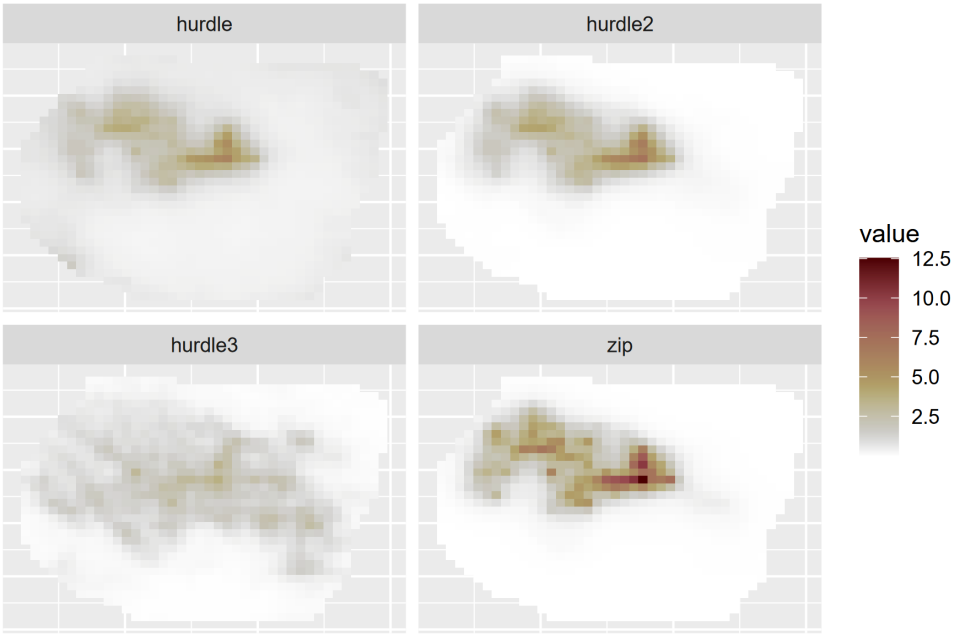


Figure 8: Estimated expected counts for all four models

Task

Create plots of the estimated variance of the counts.
Take hint
The fomulas for the variances are in Equation 2 and Equation 3.
Click here to see the solution

```
pred_zip <- predict(
  fit_zip,
  counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_1
    lambda <- area * exp( distance + elevation + space + Intercept)
    variance <- (1-pi) * (lambda + pi * lambda^2)
    list( variance = variance)
  },n.samples = 2500)

pred_zap <- predict( fit_zap, counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_0
    lambda <- area * exp( distance + elevation + space + Intercept)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
    variance = expect *(1-exp(-lambda) * expect)
    list(variance = variance)
  },
  n.samples = 2500)

inv.logit = function(x) (exp(x)/(1+exp(x)))

pred_zap2 <- predict( fit_zap2, counts_df,
  ~ {
    pi <- inv.logit(Intercept_presence + elev_presence + dist_presence + space_presence)
    lambda <- area * exp( dist_count + elev_count + space_count + Intercept_count)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
    variance = expect *(1-exp(-lambda) * expect)
    list(variance = variance)
  },
  n.samples = 2500)

pred_zap3 <- predict( fit_zap3, counts_df,
  ~ {
    pi <- inv.logit(Intercept_presence + elev_presence + dist_presence + space_copy)
    lambda <- area * exp( dist_count + elev_count + space + Intercept_count)
    expect <- ((1-exp(-lambda))^(-1) * pi * lambda)
    variance = expect *(1-exp(-lambda) * expect)
    list(variance = variance)
  },
  n.samples = 2500)




  data.frame(x = st_coordinates(counts_df)[,1],
             y = st_coordinates(counts_df)[,2],
    zip = pred_zip$variance$mean,
         hurdle = pred_zap$variance$mean,
         hurdle2 = pred_zap2$variance$mean,
         hurdle3 = pred_zap3$variance$mean)  %>%
  pivot_longer(-c(x,y)) %>%
  ggplot() + geom_tile(aes(x,y, fill = value)) + facet_wrap(.~name) +
    theme_map + scale_fill_scico(direction = -1)
```



8 Using scores

We can compare model using the scores that the *bru()* function computes since we have set, at the beginning. the options to ::: {.cell}

```
bru_options_set(control.compute = list(dic = TRUE,
                                       waic = TRUE,
                                       mlik = TRUE,
                                       cpo = TRUE))
```

:::

Lets use these scores to compare the models.

Task

Extract the DIC, WAIC and MLIK values for the four models and compare them
Click here to see the solution

```
data.frame( Model = c("ZIP", "HURDLE", "HURDLE_2","HURDLE_3" ),
  DIC = c(fit_zip$dic$dic, fit_zap$dic$dic, WAIC = fit_zap2$dic$dic, fit_zap3$dic$dic),
            WAIC = c(fit_zip$waic$waic, fit_zap$waic$waic, fit_zap2$waic$waic, fit_zap3$wa
            MLIK = c(fit_zip$mlik[1], fit_zap$mlik[1], fit_zap2$mlik[1], fit_zap3$mlik[1])
#>       Model      DIC     WAIC      MLIK
#> 1       ZIP 1214.141 1223.721 -686.9496
#> 2    HURDLE 1886.066 1909.722 -994.3807
#> 3 HURDLE_2 1268.319 1285.522 -734.3697
#> 4 HURDLE_3 1465.658 1515.336 -871.7430
```

From the table above we can see that the model that best balances complexity and fit is the zero inflated one (ZIP).