PRACTICAL 8 - ZERO INFLATED

Aim of this practical:

In this practical we are going to look zero-inflated models.

1 ZIP/ZAP and Hurdle Models

```
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

1 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:

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

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

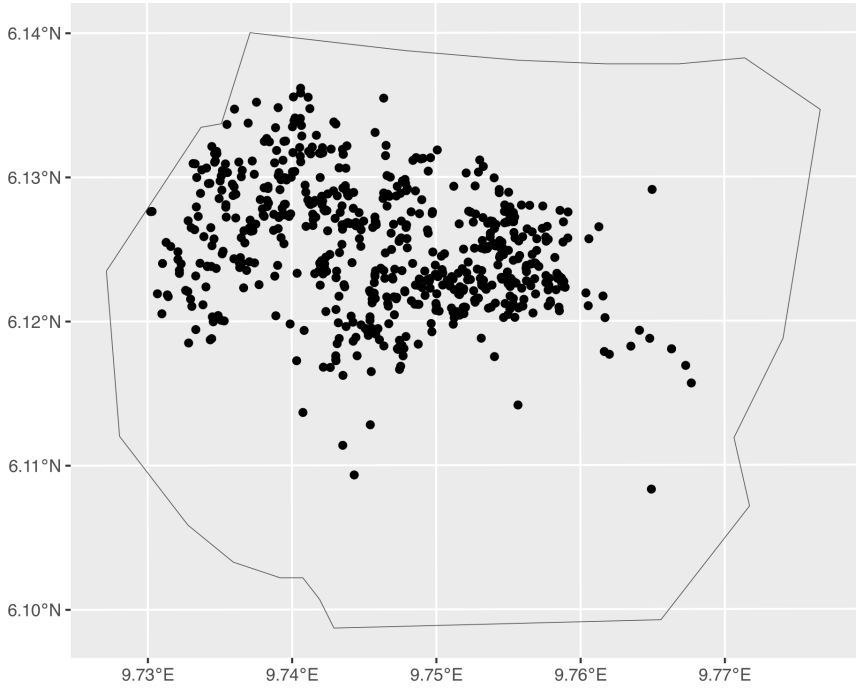


Figure 1: Location of gorilla nests

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

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

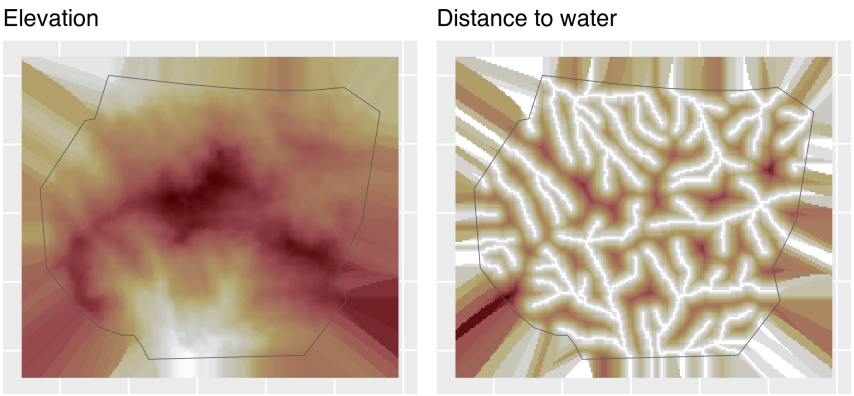


Figure 2: Covariates

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)
```

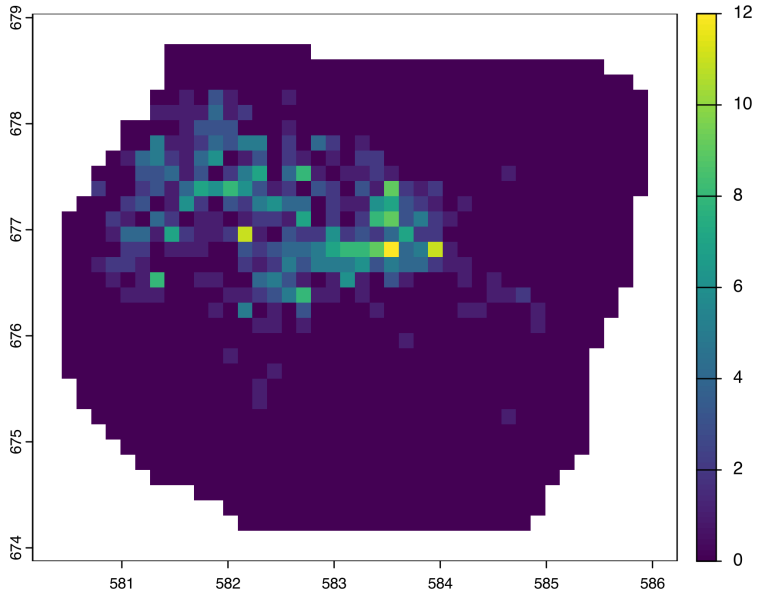


Figure 3: Counts of gorilla nests

```
# 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()
```

1.1.1 Mesh building

We now define the mesh and the spde object.

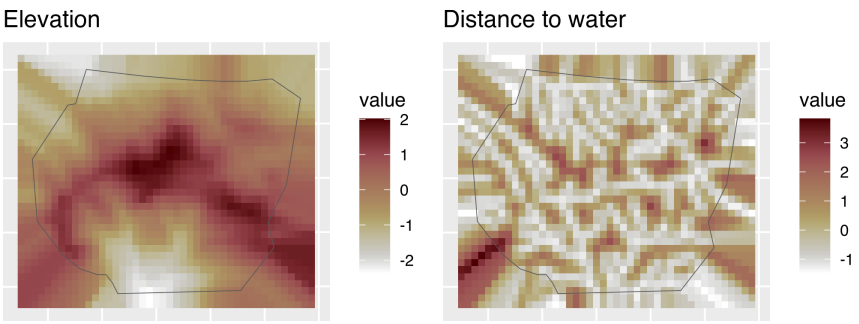


Figure 4: Covariates

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

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

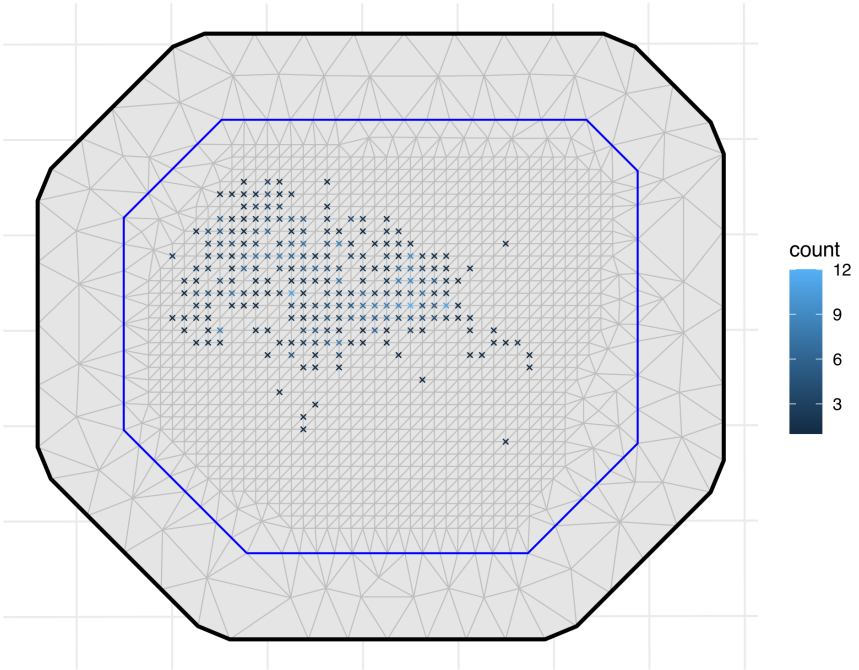


Figure 5: Mesh over the count locations

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.

1 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:

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

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:

```
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 1
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
)
```

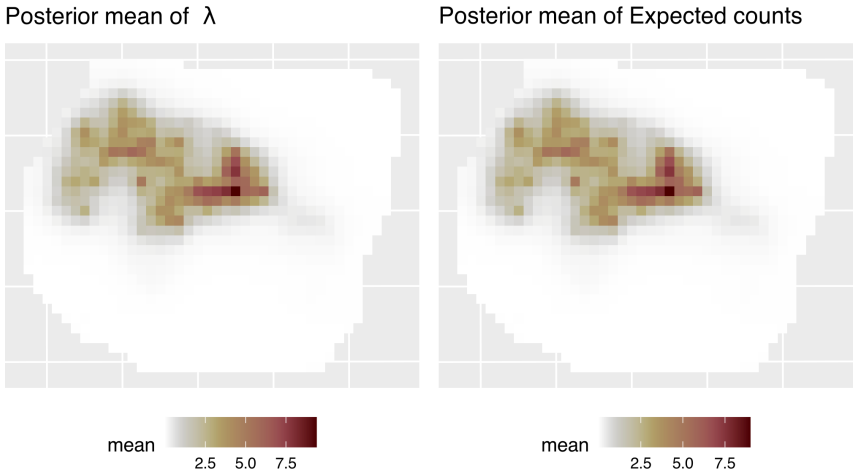


Figure 6: Estimated λ (left) and expected counts (right) with zero inflated model

1 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:

$$\begin{aligned} E(\text{count}) &= \frac{1}{1-\exp(-\lambda)}\pi\lambda \\ Var(\text{count}) &= E(\text{count})\left(1-\exp(-\lambda)E(\text{count})\right) \end{aligned} \tag{2}$$

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 2 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
)
```

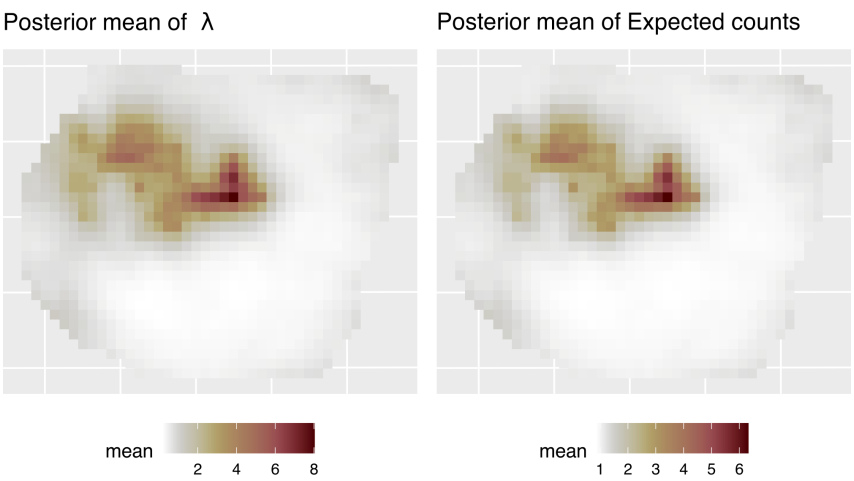


Figure 7: Estimated λ (left) and expected counts (right) with hurdle model

1 Hurdle model using two likelihoods

Here the model is the same as in Section 1.3, 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
)
```

1 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)
```

1 Comparing models

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

1.6.1 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 1.2 and Section 1.3 for all four models and plot them together:

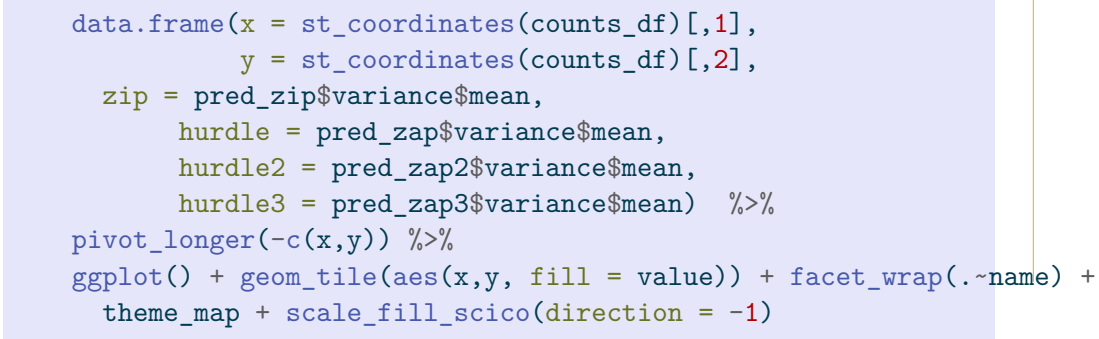
```
pred_zip <- predict(
  fit_zip,
  counts_df,
  ~ {
    pi <- zero_probability_parameter_for_zero_inflated_poisson_1
    lambda <- area * exp( distance + elevation + space + Intercept)
    expect <- (1-pi) * lambda
    variance <- (1-pi) * (lambda + pi * lambda^2)
    list(
      expect = expect
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




p =   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)
```

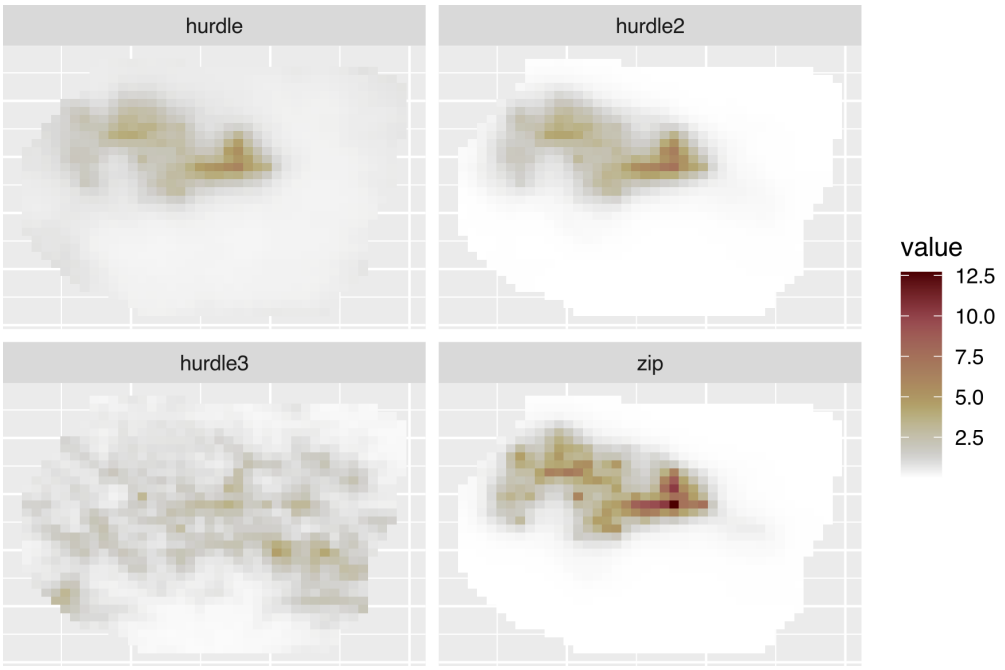
Task

Create plots of the estimated variance of the counts.
Take hint
The fomulas for the variances are in Equation 1 and Equation 2.
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)
```



1.6.2 Using scores

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

```
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 1214.140 1223.719 -686.9494
#> 2   HURDLE 1886.066 1909.722 -994.3807
#> 3 HURDLE_2 1268.307 1285.508 -734.3704
#> 4 HURDLE_3 1885.425 3908.932 -858.1401
```

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