PRACTICAL 5

Aim of this practical:

In this first practical we are going to fit spatial modes for Areal, Geostatistical and Point-Process Data.

1 Models for areal data

In this practical we are going to fit an areal model. We will:

- Learn how to fit an areal model in `inlabru`
- Learn how to add spatial covariates to the model
- Learn how to do predictions
- Learn how to simulate from the fitted model

Libraries to load:

```
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
library(mapview)

# load some libraries to generate nice map plots
library(scico)
```

1 The data

We consider data on respiratory hospitalizations for Greater Glasgow and Clyde in 2007. The data are available from the `CARBayesdata` R Package:

```
library(CARBayesdata)

data(pollutionhealthdata)
data(GGHB.IZ)
```

The `pollutionhealthdata` contains the spatiotemporal data on respiratory hospitalizations, air pollution concentrations and socio-economic deprivation covariates for the 271 Intermediate Zones (IZ) that make up the Greater Glasgow and Clyde health board in Scotland. Data are provided by the Scottish Government and the available variables are:

- `IZ`: unique identifier for each IZ.
- `year`: the year when the measurements were taken

- `observed`: observed numbers of hospitalizations due to respiratory disease.
- `expected`: expected numbers of hospitalizations due to respiratory disease computed using indirect standardisation from Scotland-wide respiratory hospitalization rates.
- `pm10`: Average particulate matter (less than 10 microns) concentrations.
- `jsa`: The percentage of working age people who are in receipt of Job Seekers Allowance
- `price`: Average property price (divided by 100,000).

The `GGHB.IZ` data is a Simple Features (`sf`) object containing the spatial polygon information for the set of 271 Intermediate Zones (IZ), that make up of the Greater Glasgow and Clyde health board in Scotland (Figure 1).

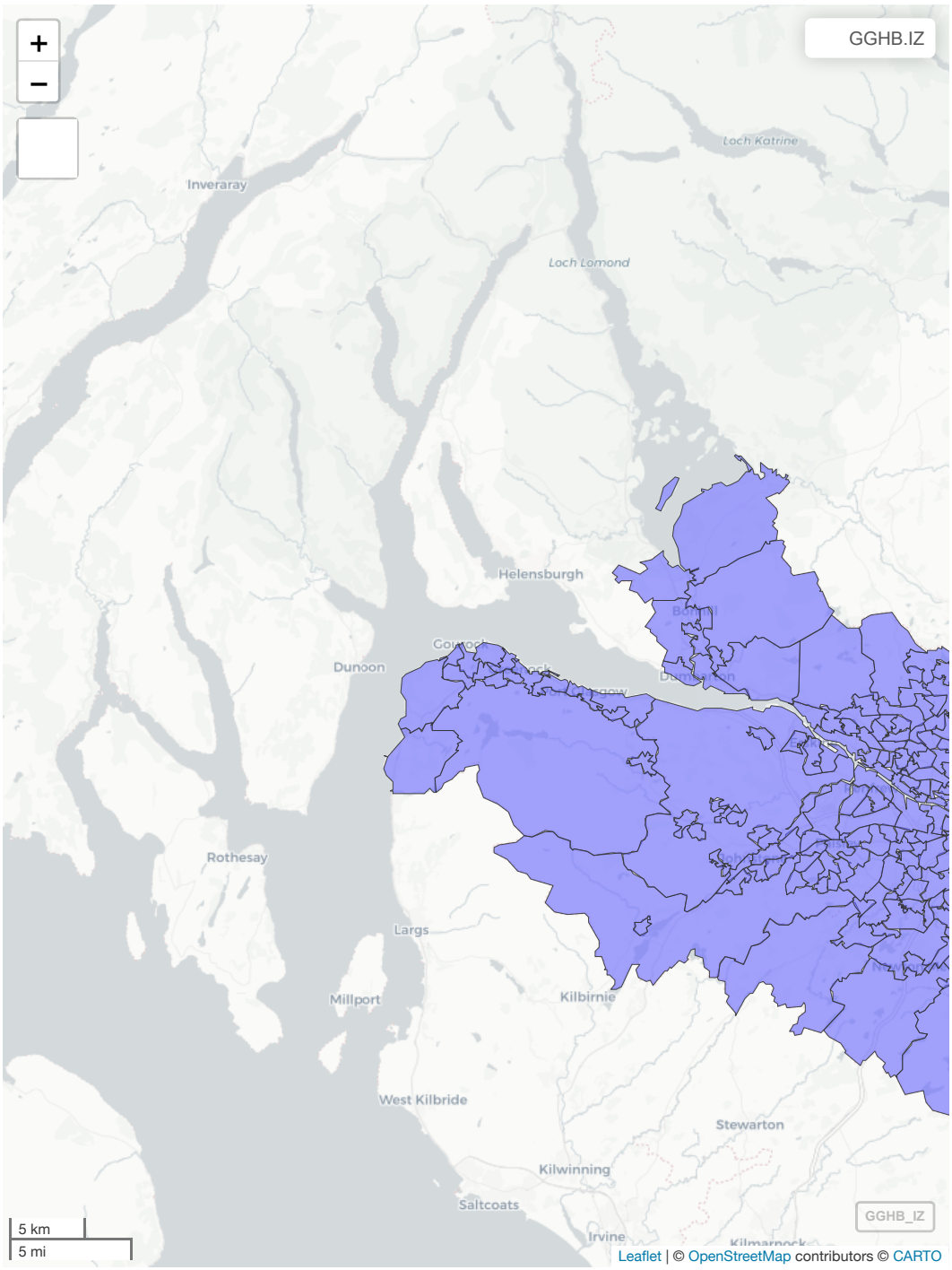
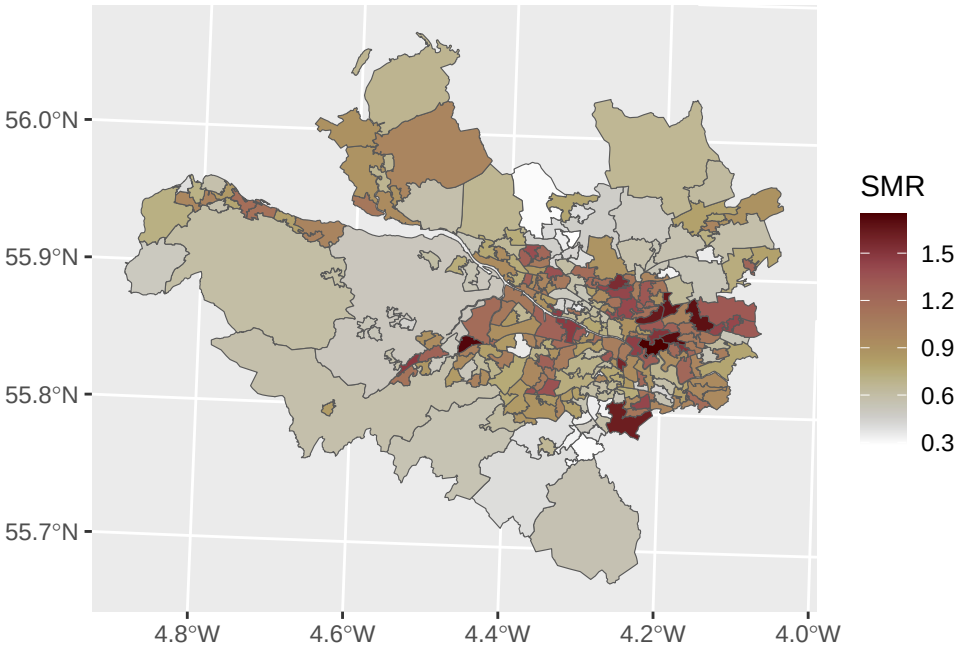


Figure 1: Greater Glasgow and Clyde health board represented by 271 Intermediate Zones

We first merge the two dataset and select only one year of data, compute the SME and plot the observed

```
resp_cases <- merge(GGHB.IZ %>%
                      mutate(space = 1:dim(GGHB.IZ)[1]),
                            pollutionhealthdata, by = "IZ") %>%
  filter(year == 2007) %>%
    mutate(SMR = observed/expected)

ggplot() + geom_sf(data = resp_cases, aes(fill = SMR)) + scale_fill_scico(direction = -1)
```



Then we compute the adjacency matrix using the functions `poly2nb()` and `nb2mat()` in the `spdep` library. We then convert the adjacency matrix into the precision matrix **Q** of the CAR model. Remember this matrix has, on the diagonal the number of e

```
library(spdep)

W.nb <- poly2nb(GGHB.IZ,queen = TRUE)
R <- nb2mat(W.nb, style = "B", zero.policy = TRUE)

diag = apply(R,1,sum)
Q = -R
diag(Q) = diag
```

1 The model

We fit a first model to the data where we consider a Poisson model for the observed cases.

Stage 1 Model for the response

$$y_i|\eta_i \sim \text{Poisson}(E_i\lambda_i)$$

where E_i are the expected cases for area i.

Stage 2 Latent field model

$$\eta_i = \log(\lambda_i) = \beta_0 + \omega_i + z_i$$

where

- β_0 is a common intercept
- $= (\omega_1, \dots, \omega_k)$ is a conditional Autorgressive model (CAR) with precision matrix $\tau_1 \mathbf{Q}$
- $\mathbf{z} = (z_1, \dots, z_k)$ is an unstrictured random effect with precision τ_2

Stage 3 Hyperparameters

The hyperparameters of the model are τ_1 and τ_2

NOTE In this case the linear predictor η consists of three components!!

Task

Fit the above model in using `inlabru` by completing the following code:

```
cmp = ~ Intercept(1) + space(...) + iid(...)

formula = ...


lik = bru_obs(formula = formula,
              family = ...,
              E = ...,
              data = ...)

fit = bru(cmp, lik)
```

Answer

```
cmp = ~ Intercept(1) + space(space, model = "besag", graph = Q) + iid(space, model = "iid"

formula = observed ~ Intercept + space + iid

lik = bru_obs(formula = formula,
              family = "poisson",
              E = expected,
              data = resp_cases)

fit = bru(cmp, lik)
```

After fitting the model we want to extract results.

Question

1. What is the estimated value for β_0? _____
2. Look at the estimated values of the hyperparameters using `fit$summary.hyperpar` , which of the two spatial components (structured or unstructured) explains more of the variability in the counts?

- (A) structured

- (B) unstructured

We now look at the predictions over space.

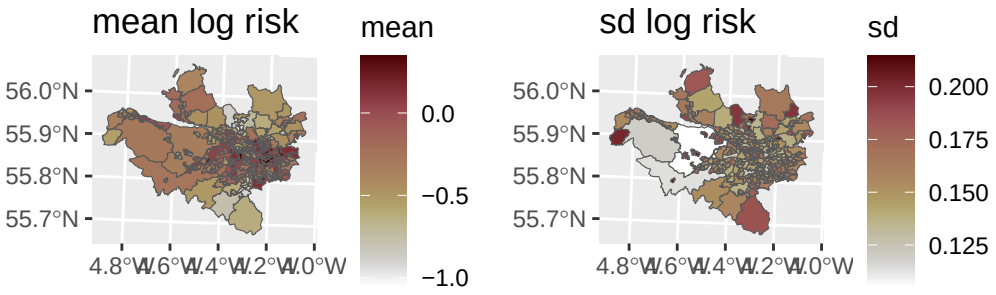
Task

Complete the code below to produce prediction of the linear predictor η_i and of the risk λ_i and of the expected cases $E_i \exp(\lambda_i)$ over the whole space of interest. Then plot the mean and sd of the resulting surfaces.

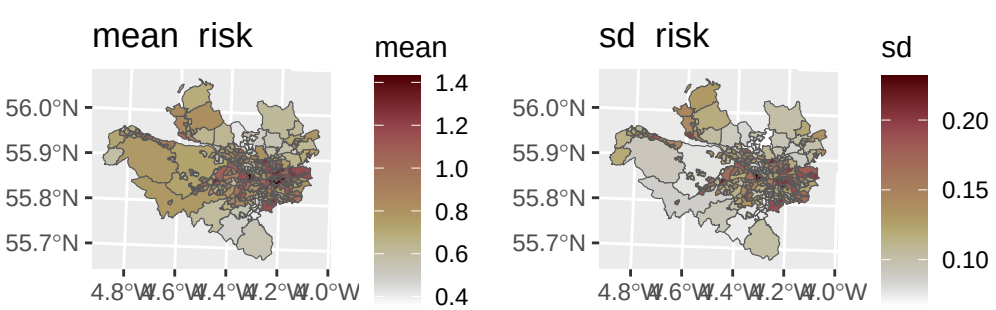
```
pred = predict(fit, resp_cases, ~data.frame(log_risk = ...,
                                            risk = exp(...),
                                            cases = ...
                                            ),
               n.samples = 1000)
```

Show Answer

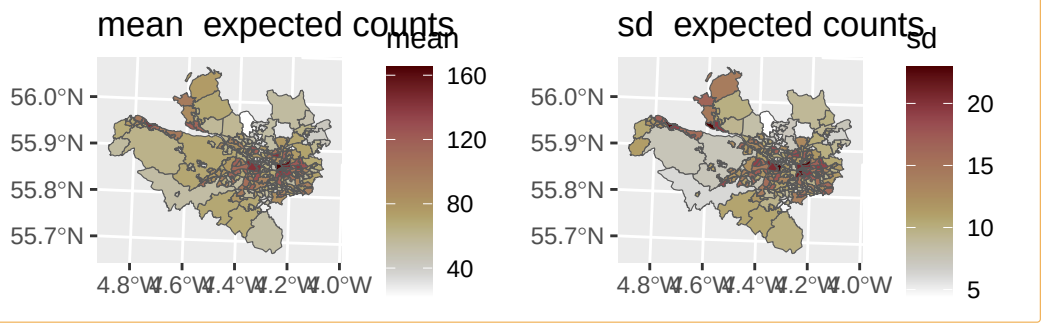
```
# produce predictions
pred = predict(fit, resp_cases, ~data.frame(log_risk = Intercept + space,
                                            risk = exp(Intercept + space),
                                            cases = expected * exp(Intercept + space)
                                            ),
               n.samples = 1000)
# plot the predictions

p1 = ggplot() + geom_sf(data = pred$log_risk, aes(fill = mean)) + scale_fill_scico(directi
p2 = ggplot() + geom_sf(data = pred$log_risk, aes(fill = sd)) + scale_fill_scico(direction
p1 + p2
```




```
p1 = ggplot() + geom_sf(data = pred$risk, aes(fill = mean)) + scale_fill_scico(direction =
p2 = ggplot() + geom_sf(data = pred$risk, aes(fill = sd)) + scale_fill_scico(direction = -
p1 + p2
```

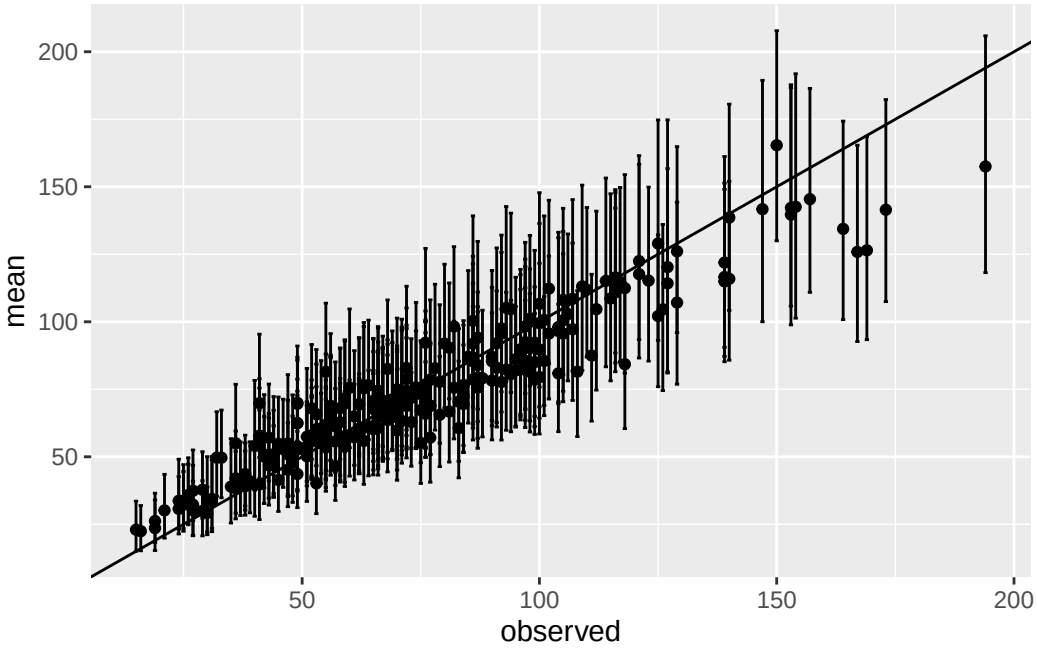


```
p1 = ggplot() + geom_sf(data = pred$cases, aes(fill = mean)) + scale_fill_scico(direction
p2 = ggplot() + geom_sf(data = pred$cases, aes(fill = sd)) + scale_fill_scico(direction =
p1 + p2
```



Finally we want to compare our observations y_i with the predicted means of the Poisson distribution $E_i \exp(\lambda_i)$

```
pred$cases %>% ggplot() + geom_point(aes(observed, mean)) +
  geom_errorbar(aes(observed, ymin = q0.025, ymax = q0.975)) +
  geom_abline(intercept = 0, slope = 1)
```



Note: Here we are predicting the *mean* of counts, not the counts!!! Predicting counts is the theme of the next task!

1 Getting prediction densities

Posterior predictive distributions, that is $\pi(y_i^{\text{new}}|\mathbf{y})$ are of interest in many applied problems. The `bru()` function does not return predictive densities. In the previous step we have computed predictions for the `expected counts` $\pi(E_i\lambda_i|\mathbf{y})$. The predictive distribution is then:

$$\pi(y_i^{\text{new}}|\mathbf{y}) = \int \pi(y_i|E_i\lambda_i)\pi(E_i\lambda_i|\mathbf{y})\, dE_i\lambda_i$$

where, in our case, $\pi(y_i|E_i\lambda_i)$ is Poisson with mean $E_i\lambda_i$. We can achieve this using the following algorith:

1. Simulate n replicates of $g^k = E_i\lambda_i$ for $k = 1, \dots, n$ using the function *generate()* which takes the same input as *predict()*
2. For each of the k replicates simulate a new value y_i^{new} using the function *rpois()*
3. Summarise the n samples of y_i^{new} using, for example the mean and the 0.025 and 0.975 quantiles.

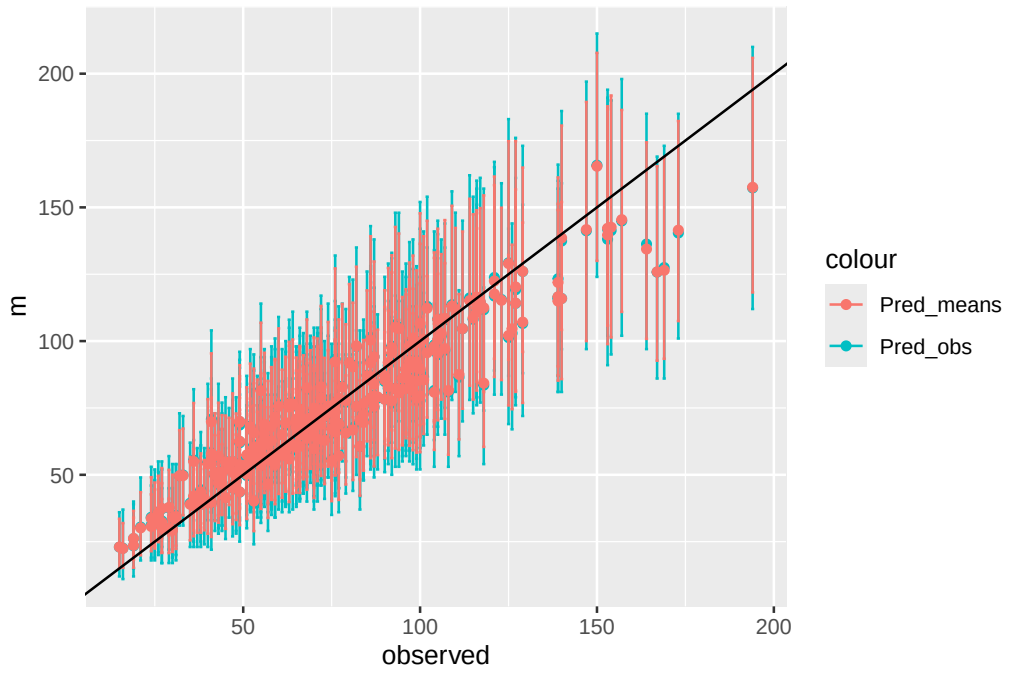
Here is the code:

```
# simulate 1000 realizations of E_i\lambda_i
expected_counts = generate(fit, resp_cases,
                           ~ expected * exp(Intercept + space),
                           n.samples = 1000)


# simulate poisson data
aa = rpois(271*1000, lambda = as.vector(expected_counts))
sim_counts = matrix(aa, 271, 1000)

# summarise the samples with posterior means and quantiles
pred_counts = data.frame(observed = resp_cases$observed,
                         m = apply(sim_counts,1,mean),
                         q1 = apply(sim_counts,1,quantile, 0.025),
                         q2 = apply(sim_counts,1,quantile, 0.975),
                         vv = apply(sim_counts,1,var)
                         )
```

Task

Plot the observations against the predicted new counts and the predicted expected counts. Include the uncertainty and compare the two.
Click here to see the solution

```
ggplot() +
  geom_point(data = pred_counts, aes(observed, m, color = "Pred_obs")) +
  geom_errorbar(data = pred_counts, aes(observed, ymin = q1, ymax = q2, color = "Pred_obs"
  geom_point(data = pred$cases, aes(observed, mean, color = "Pred_means")) +
  geom_errorbar(data = pred$cases, aes(observed, ymin = q0.025, ymax = q0.975, color = "Pr

  geom_abline(intercept = 0, slope =1)
```



1 Adding a spatial covariate

Finally, we want to add a covariate to the model. We are going to check if the PM10 levels influence the numbers of hospitalizations.

Task

Add the effect of PM10 in the previous model. First as a linear effect and then as a smooth effect (RW2). Check the results.
Take hint
To use the RW2 model you first need to group the values of the PM10 using the `inla.group()` function.
Click here to see the solution

```
## Linear Effect
cmp = ~ Intercept(1) + space(space, model = "besag", graph = Q) + iid(space, model = "iid"
  cov(pm10, model = "linear")

formula = observed ~ Intercept + space + iid + cov

lik = bru_obs(formula = formula,
              family = "poisson",
              E = expected,
              data = resp_cases)

fit_lin = bru(cmp, lik)
# effect of the covaraite
# fit_lin$summary.fixed

## Linear Effect
resp_cases$pm10_group=  inla.group(resp_cases$pm10)
cmp = ~ Intercept(1) + space(space, model = "besag", graph = Q) + iid(space, model = "iid"
  cov(pm10_group, model = "rw2")

formula = observed ~ Intercept + space + iid + cov

lik = bru_obs(formula = formula,
              family = "poisson",
              E = expected,
              data = resp_cases)

fit_smooth = bru(cmp, lik)

#check the smooth effect of the covariate
#fit_smooth$summary.random$cov %>% ggplot() + geom_line(aes(ID,mean)) +
#  geom_ribbon(aes(ID, ymin =`0.025quant`, ymax = `0.975quant`), alpha = 0.5)
```

2 Geostatistics

In this practical we are going to fit a geostatistical model. We will:

- Learn how to fit a geostatistical model in `inlabru`
- Learn how to build a mesh for the SPDE representation
- Learn how to add spatial covariates to the model
- Learn how to do predictions
- Learn how to simulate from the fitted model

Libraries to load:

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


# load some libraries to generate nice map plots
library(scico)
library(ggplot2)
library(patchwork)
library(mapview)
library(tidyterra)
```

2 The data

In this practical, we will explore data on the Pacific Cod (*Gadus macrocephalus*) from a trawl survey in Queen Charlotte Sound. The `pcod` dataset is available from the `sdmTMB` package and contains the presence/absence records of the Pacific Cod during each surveys along with the biomass density of Pacific cod in the area swept (kg/Km^2). The `qcs_grid` data contain the depth values stored as 2×2 km grid for Queen Charlotte Sound.

The dataset contains presence/absence data from 2003 to 2017. In this practical we only consider year 2003.

We first load the dataset and select the year of interest

```
library(sdmTMB)

pcod_df = sdmTMB::pcod %>% filter(year==2003)
qcs_grid = sdmTMB::qcs_grid
```

Then, we create ab `sf` object and assign the rough coordinate reference to it:

```
pcod_sf =   st_as_sf(pcod_df, coords = c("lon","lat"), crs = 4326)
pcod_sf = st_transform(pcod_sf,
crs = "+proj=utm +zone=9 +datum=WGS84 +no_defs +type=crs +units=km" )
```

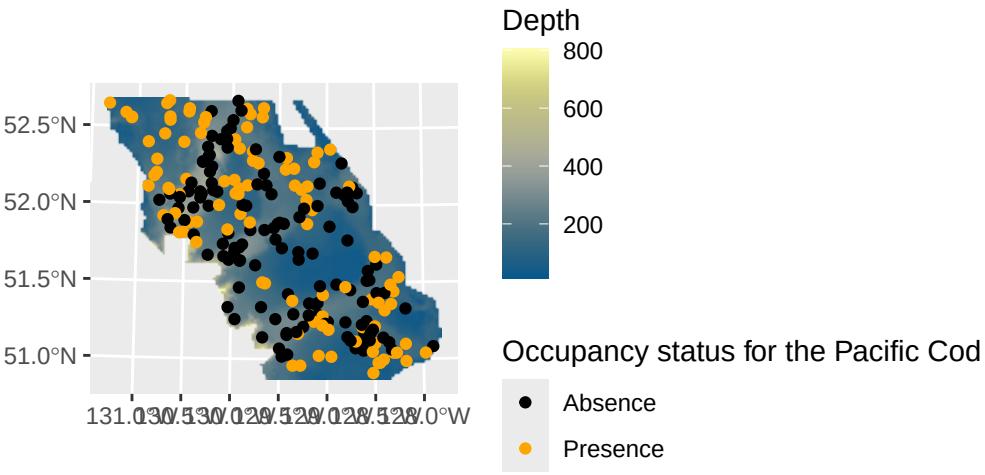
We convert the covariate into a raster and assign the same coordinate reference:

```
depth_r <- rast(qcs_grid, type = "xyz")
crs(depth_r) <- crs(pcod_sf)
```

Finally we can plot our dataset. Note that to plot the raster we need to upload also the `tidyterra` library.

```
ggplot()+
  geom_spatraster(data=depth_r$depth)+
  geom_sf(data=pcod_sf,aes(color=factor(present))) +
    scale_color_manual(name="Occupancy status for the Pacific Cod",
                       values = c("black","orange"),
                       labels= c("Absence","Presence"))+
```

```
    scale_fill_scico(name = "Depth",
                     palette = "nuuk",
                     na.value = "transparent" ) + xlab("") + ylab("")
```



2 The model

We first fit a simple model where we consider the observation as Bernoulli and where the linear predictor contains only one intercept and the GR field defined through the SPDE approach. The model is defined as:

Stage 1 Model for the response

$$y(s)|\eta(s) \sim \text{Binom}(1, p(s))$$

Stage 2 Latent field model

$$\eta(s) = \text{logit}(p(s)) = \beta_0 + \omega(s)$$

with

$$\omega(s) \sim \text{ GF with range } \rho \text{ and maginal variance } \sigma^2$$

Stage 3 Hyperparameters

The hyperparameters of the model are ρ and σ

NOTE In this case the linear predictor η consists of two components!!

2.2.1 The workflow

When fitting a geostatistical 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

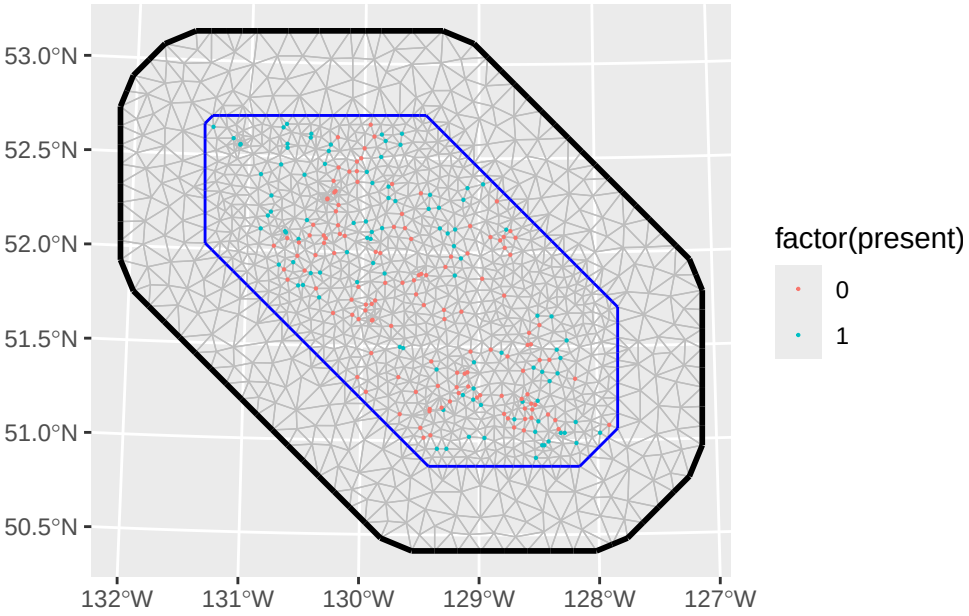
4. Define the observation model using the `bru_obs()` function
5. Run the model using the `bru()` function

2.2.2 1. Building the mesh

The first task, when dealing with geostatistical models in `inlabru` is to build the mesh that covers the area of interest. For this purpose we use the function `fm_messh_2d`.

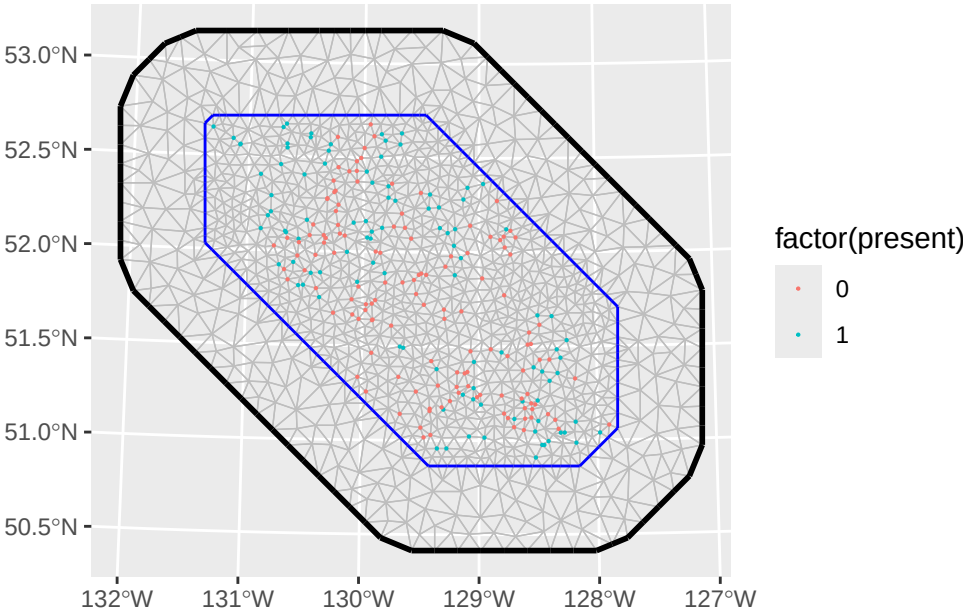
One way to build the mesh is to start from the locations where we have observations, these are contained in the dataset `pcod_sf`.

```
mesh = fm_mesh_2d(loc = pcod_sf,            # Build the mesh
                  max.edge = c(10,20),      # The largest allowed triangle edge length.
                  offset = c(5,50))        # The automatic extension distance
ggplot() + gg(mesh) + geom_sf(data= pcod_sf, aes(color = factor(present)), size = 0.1) + xla
```



As you can see from the plot above, some of the locations are very close to each other, this causes some very small triangles. This can be avoided using the option `cutoff` = which collapses the locations that are closer than a cutoff (those points are collapsed in the mesh construction but, of course, not when it come to estimaation.)

```
mesh = fm_mesh_2d(loc = pcod_sf,            # Build the mesh
                  cutoff = 2,
                  max.edge = c(10,20),      # The largest allowed triangle edge length.
                  offset = c(5,50))        # The automatic extension distance
ggplot() + gg(mesh) + geom_sf(data= pcod_sf, aes(color = factor(present)), size = 0.1) + xla
```



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.
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.

2.2.3 2. 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 margianal variance σ you need to define two parameters σ_0 and p_σ such that you believe it is reasonable that

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

You can use the following function to compute and plot the prior distributions for the range and sd of the Matern field.

```
dens_prior_range = function(rho_0, p_alpha)
{
  # compute the density of the PC prior for the
  # range rho of the Matern field
  # rho_0 and p_alpha are defined such that
  # P(rho<rho_0) = p_alpha
  rho = seq(0, rho_0*10, length.out =100)
  alpha1_tilde = -log(p_alpha) * rho_0
  dens_rho =  alpha1_tilde / rho^2 * exp(-alpha1_tilde / rho)
  return(data.frame(x = rho, y = dens_rho))
}

dens_prior_sd = function(sigma_0, p_sigma)
{
  # compute the density of the PC prior for the
  # sd sigma of the Matern field
  # sigma_0 and p_sigma are defined such that
  # P(sigma>sigma_0) = p_sigma
  sigma = seq(0, sigma_0*10, length.out =100)
  alpha2_tilde = -log(p_sigma)/sigma_0
  dens_sigma = alpha2_tilde* exp(-alpha2_tilde * sigma)
  return(data.frame(x = sigma, y = dens_sigma))
}
```

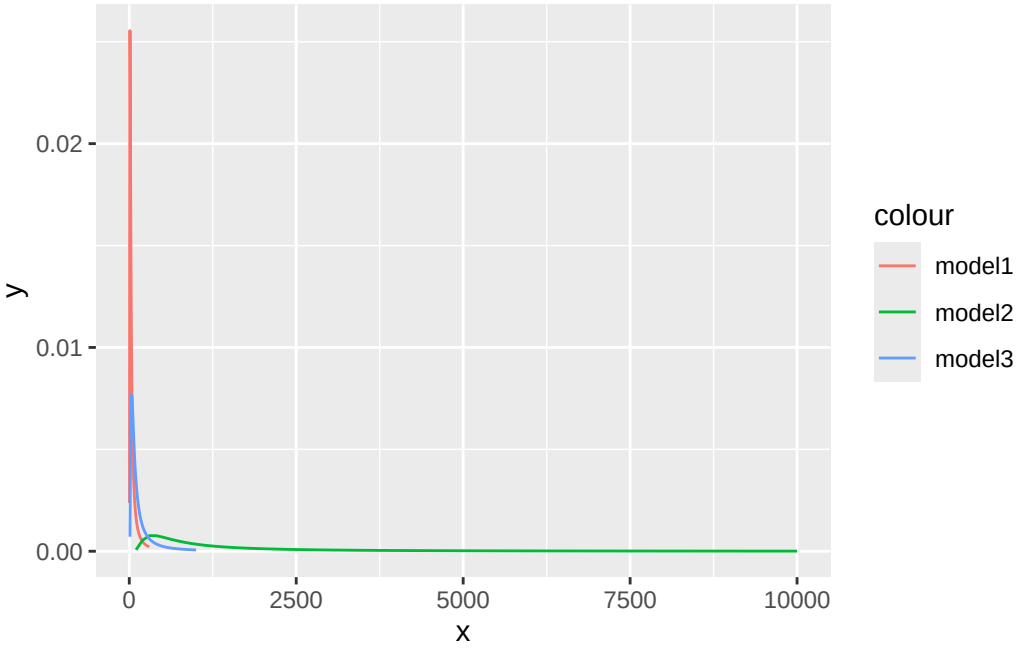
Here are some alternatives for defining priors for our model

```
spde_model1 =  inla.spde2.pcmatern(mesh,
                                   prior.sigma = c(.1, 0.5),
                                   prior.range = c(30, 0.5))
spde_model2 =  inla.spde2.pcmatern(mesh,
                                   prior.sigma = c(10, 0.5),
                                   prior.range = c(1000, 0.5))
spde_model3 =  inla.spde2.pcmatern(mesh,
```

```
                                    prior.sigma = c(1, 0.5),
                                    prior.range = c(100, 0.5))
```

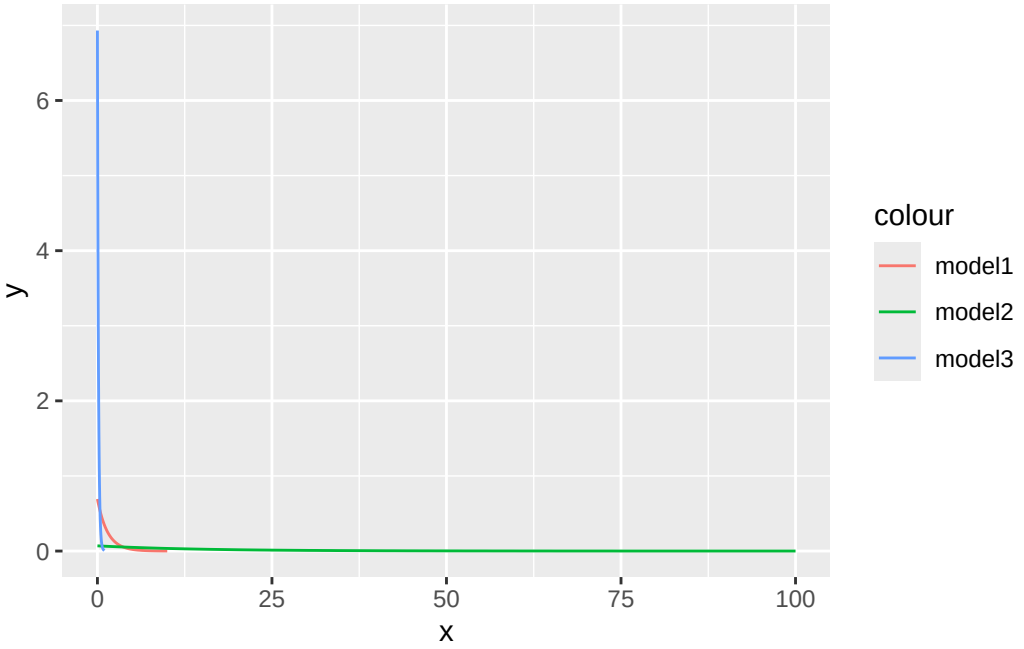
And here we plot the different priors for the range:

```
ggplot() +
  geom_line(data = dens_prior_range(30,.5), aes(x,y, color = "model1")) +
  geom_line(data = dens_prior_range(1000,.5), aes(x,y, color = "model2")) +
  geom_line(data = dens_prior_range(100,.5), aes(x,y, color = "model3"))
```



and for the sd:

```
ggplot() +
  geom_line(data = dens_prior_sd(1,.5), aes(x,y, color = "model1")) +
  geom_line(data = dens_prior_sd(10,.5), aes(x,y, color = "model2")) +
  geom_line(data = dens_prior_sd(.1,.5), aes(x,y, color = "model3"))
```



Question

Consider the `pcod_sf`, the spatial extension and type of the data…is some of the previous choices more reasonable than other?

- (A) spde_model1
- (B) spde_model2
- (C) spde_model3

NOTE Remember that a prior should be reasonable..but the model should not totally depend on it.

2.2.4 3. 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:

```
cmp = ~ Intercept(1) + space(geometry, model = spde_model3)
```

NOTE since the dataframe we use (`pcod_sf`) is an `sf` object the input in the `space()` component is the geometry of the dataset.

2.2.5 4. Define the observation model

Our data are Bernoulli distributed so we can define the observation model as:

```
formula = present ~ Intercept  + space

lik = bru_obs(formula = formula,
              data = pcod_sf,
              family = "binomial")
```

2.2.6 5. Run the model

Finally we are ready to run the model

```
fit1 = bru(cmp,lik)
```

2 Extract results

2.3.1 Hyperparameters

Task

Plot the posterior densities for the range ρ and the standard deviation σ alog with the prior for both parameters.
Take hint
Posterior marginals for the hyperparameters can be extracted from the fitted model as:
Click here to see the solution

```
fit1$marginals.hyperpar$'name of the parameter'
```

Click here to see the solution

```
# Extract marginal for the range

ggplot() +
  geom_line(data = fit1$marginals.hyperpar$`Range for space`,
            aes(x,y, color = "Posterior")) +
  geom_line(data = dens_prior_range(100,.5),
            aes(x,y, color = "Prior"))


ggplot() +
  geom_line(data = fit1$marginals.hyperpar$`Stdev for space`,
            aes(x,y, color = "Posterior")) +
  geom_line(data = dens_prior_sd(1,.5), aes(x,y, color = "Prior"))
```

2 Spatial prediction

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

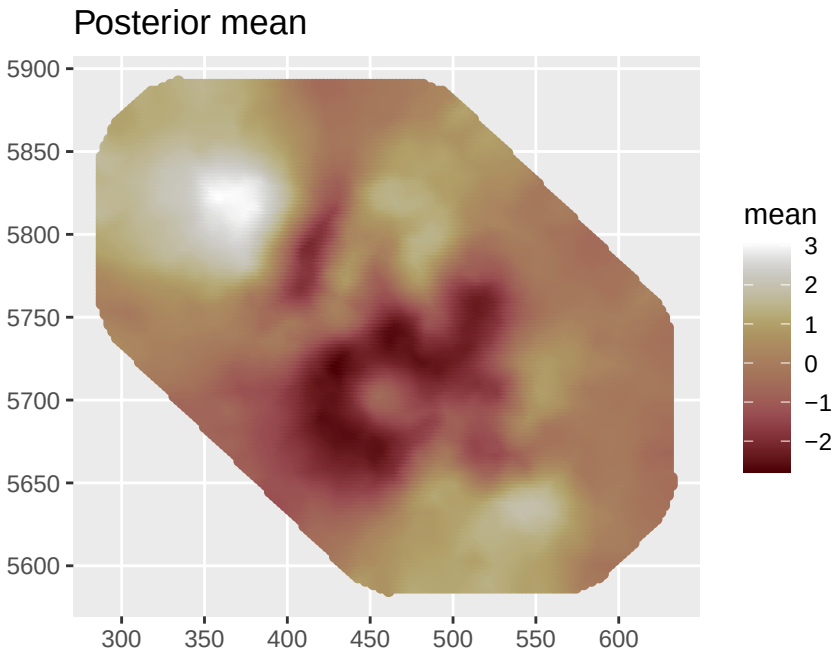
```
pxl = fm_pixels(mesh)
```

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

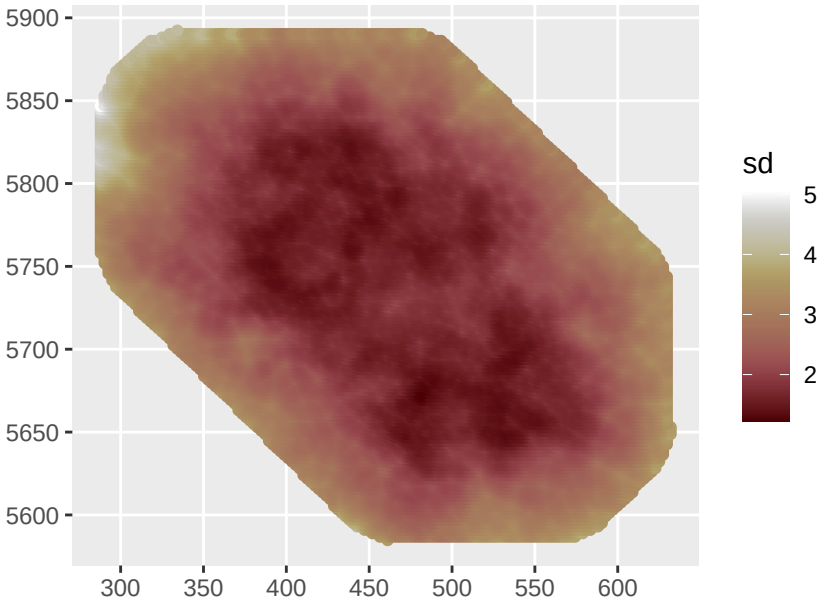
```
preds = predict(fit1, pxl, ~data.frame(spatial = space,
                                       total = Intercept + space))
```

Finally, we can plot the maps

```
ggplot() + geom_sf(data = preds$spatial,aes(color = mean)) + scale_color_scico() + ggtitle("
```




```
ggplot() + geom_sf(data = preds$spatial,aes(color = sd)) + scale_color_scico() + ggtitle("Po
```



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.

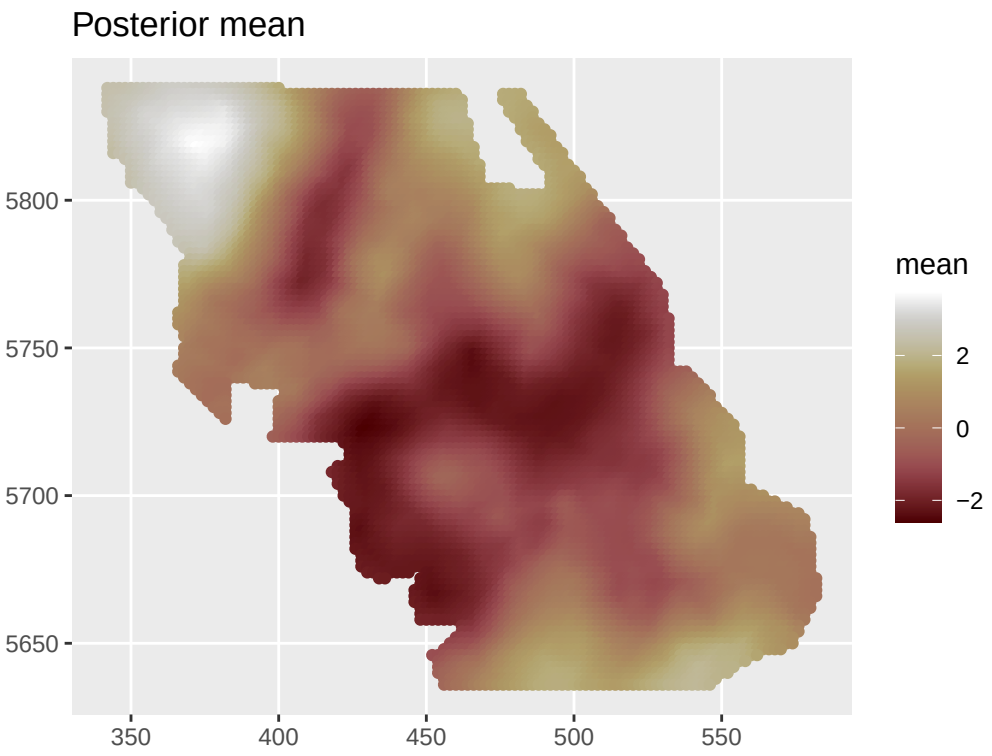
Instead of predicting over a grid covering the whole mesh, we can limit our predictions to the points where the covariate is defined. We can do this by defining a `sf` object using coordinates in the object `depth_r`.

```
pxl1 = data.frame(crds(depth_r),
                  as.data.frame(depth_r$depth)) %>%
       filter(!is.na(depth)) %>%
st_as_sf(coords = c("x","y"))
```

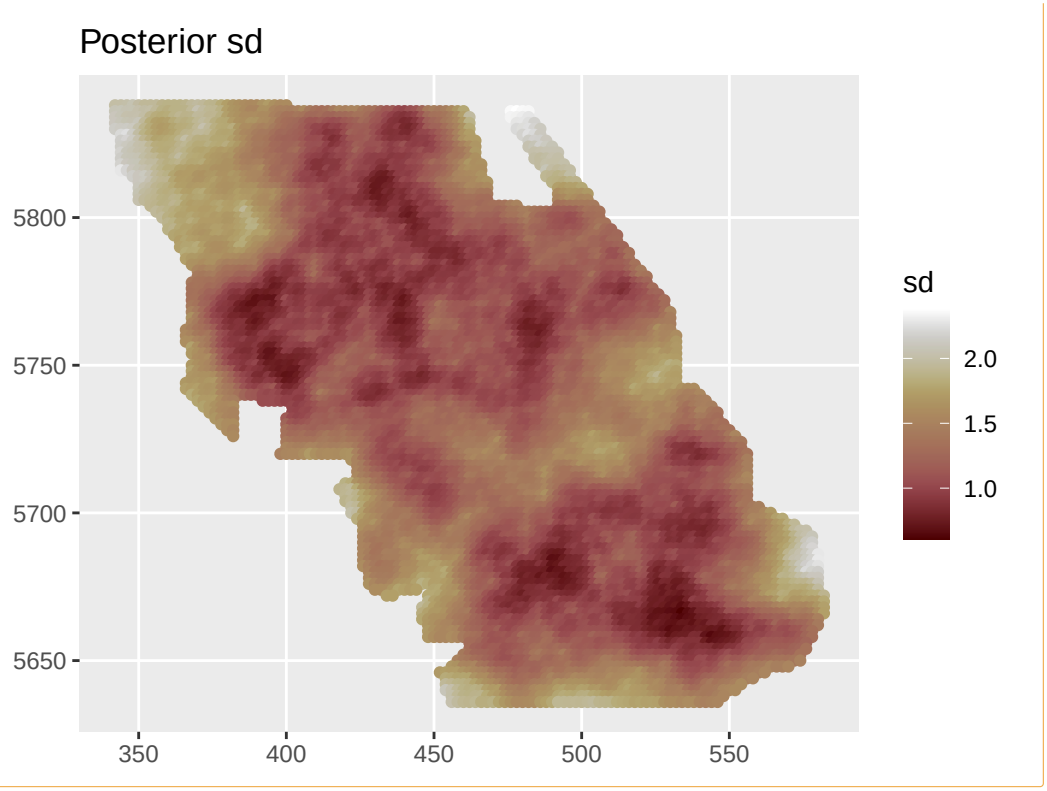
Task

Produce prediction over `pxl1` unsing the same techniques as before. Plot your results.
Take hint
Add hint details here…
Click here to see the solution

```
pred_pxl1 = predict(fit1, pxl1, ~data.frame(spatial = space,
                                            total = Intercept + space))

ggplot() + geom_sf(data = pred_pxl1$total,aes(color = mean)) + scale_color_scico() + ggtit
```

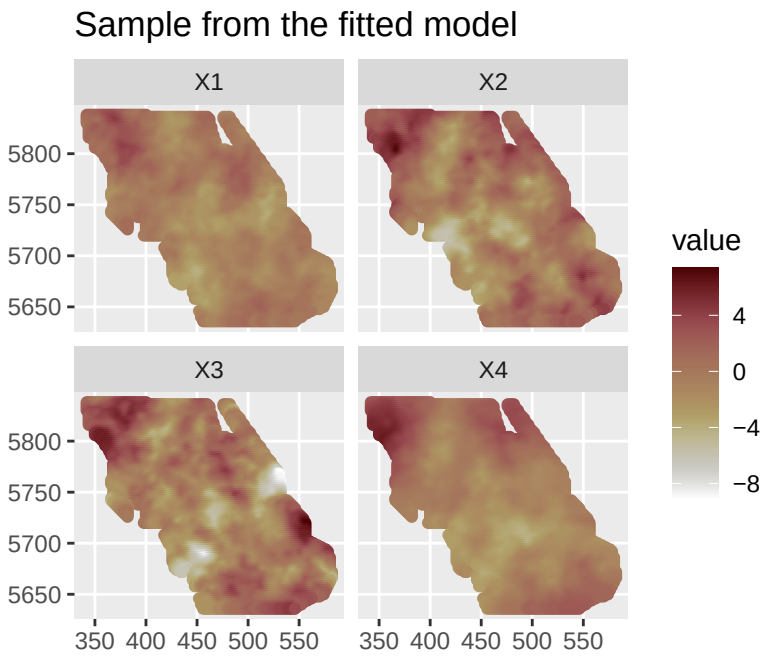



```
ggplot() + geom_sf(data = pred_pxl1$total,aes(color = sd)) + scale_color_scico() + ggtitle
```



Instead of computing the posterior mean and standard deviations of the estimated surface, one can also *simulate* possible realizations of such surface. This will give the user a better idea of the type of realized surfaces one can expect. We can do this using the function `generate()`.

```
# we simulate 4 samples from the
gens = generate(fit1, pxl1, ~ (Intercept + space),
                n.samples = 4)

pp = cbind(pxl1, gens)

pp %>% select(-depth) %>%
  pivot_longer(-geometry) %>%
    ggplot() +
      geom_sf(aes(color = value)) +
      facet_wrap(.~name) +
        scale_color_scico(direction = -1) +
        ggtitle("Sample from the fitted model")
```



2 An alternative model

We now want to check if the `depth` covatiate has an influende on the probability of presence. We do this in two different models

1. **Model 1** The depth enters the model in a linear way. The linear predictor is then defined as:

$$\eta(s) = \text{logit}(p(s)) = \beta_0 + \omega(s) + \beta_1 \text{ depth}(s)$$

2. **Model 1** The depth enters the model in a non linear way. The linear predictor is then defined as:

$$\eta(s) = \text{logit}(p(s)) = \beta_0 + \omega(s) + f(\text{depth}(s))$$

where $f(.)$ is a smooth function. We will use a RW2 model for this.

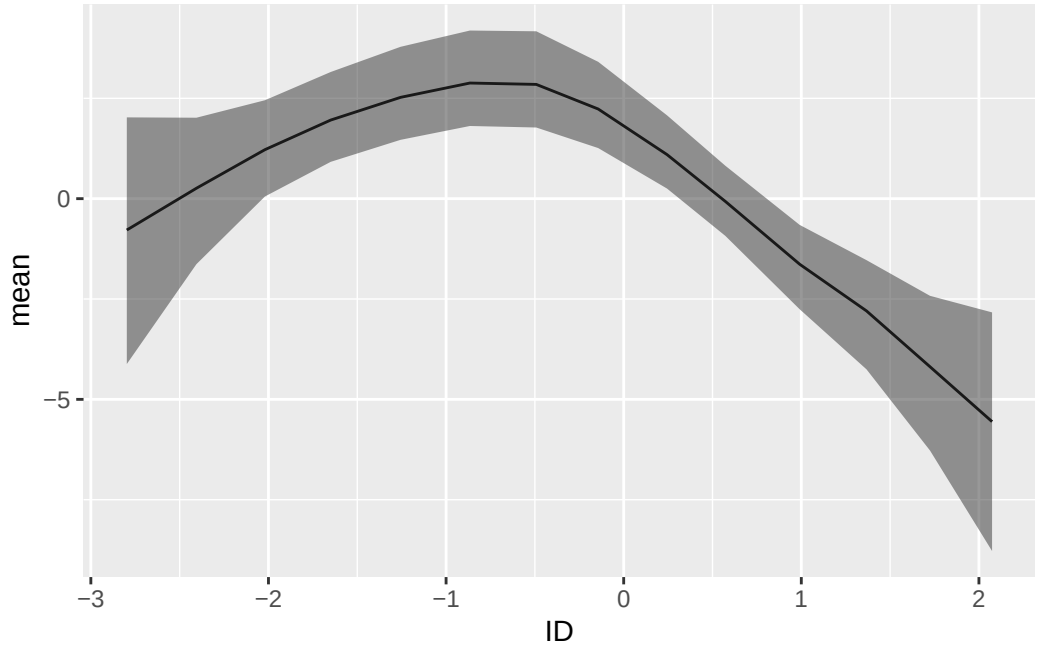
Task

Fit model 1. Define components, observation model and use the `bru()` function to estimate the parameters.
Note Use the scaled version of the covariate stored in `depth_r$depth_scaled`.
What is the liner effect of depth on the logit probability?
Take hint
Add hint details here…
Click here to see the solution

```
cmp = ~ Intercept(1) + space(geometry, model = spde_model3) +
        covariate(depth_r$depth_scaled, model = "linear")

formula = present ~ Intercept  + space + covariate

lik = bru_obs(formula = formula,
              data = pcod_sf,
              family = "binomial")


fit2 = bru(cmp, lik)
```

We now want to fit **Model 2** where we allow the effect of depth to be non-linear. To use the RW2 model we need to *group* the values of depth into distinct classe. To do this we use the function `inla.group()` which, by default, creates 20 groups. The we can fit the model as usual

```
# create the grouped variable
depth_r$depth_group = inla.group(values(depth_r$depth_scaled))

# run the model
cmp = ~ Intercept(1) + space(geometry, model = spde_model3) +
        covariate(depth_r$depth_group, model = "rw2")

formula = present ~ Intercept  + space + covariate

lik = bru_obs(formula = formula,
              data = pcod_sf,
              family = "binomial")


fit3 = bru(cmp, lik)

# plot the estimated effect of depth

fit3$summary.random$covariate %>%
  ggplot() + geom_line(aes(ID,mean)) +
                                  geom_ribbon(aes(ID, ymin = `0.025quant`,
                                                      ymax = `0.975quant`), alpha = 0.5)
```



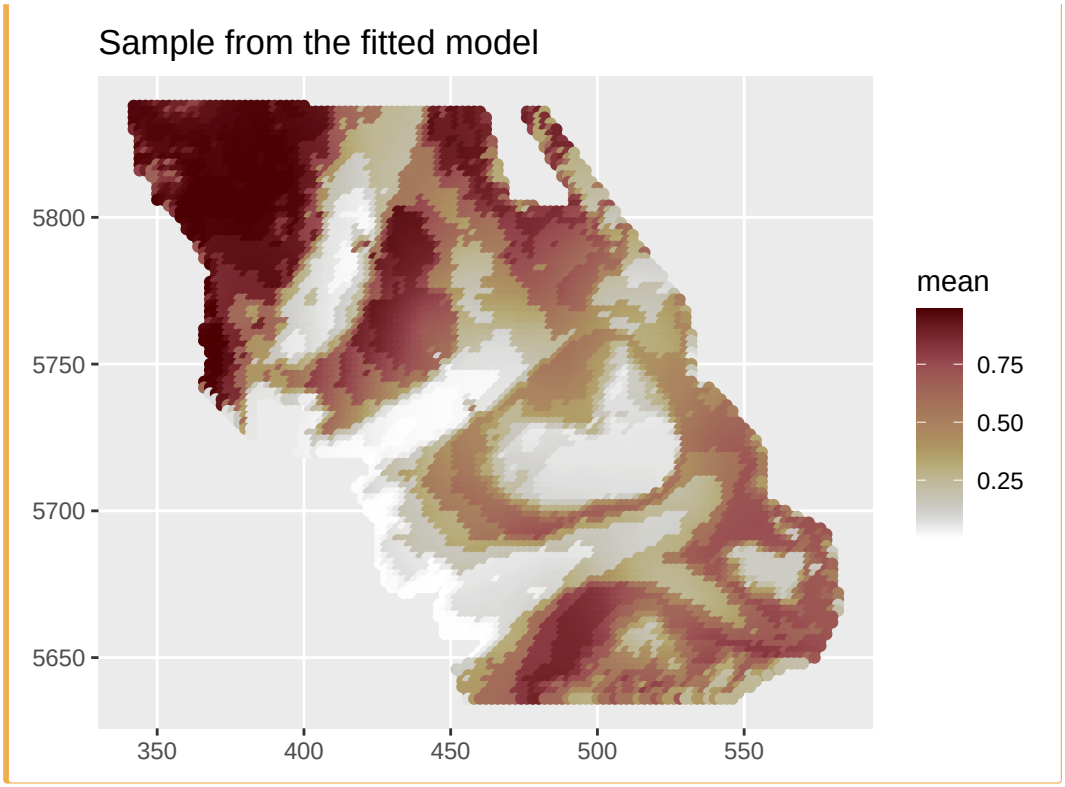
Task

Create a map of predicted *probability* from Model 3. You can use a inverse logit function defined as

```
inv_logit = function(x) (1+exp(-x))^(-1)
```

Take hint
The `predict()` function can take as input also functions of elements of the components you want to consider
Click here to see the solution

```
inv_logit = function(x) (1+exp(-x))^(-1)

pred3  = predict(fit3, pxl1, ~inv_logit(Intercept + space + covariate) )

pred3 %>% ggplot() +
      geom_sf(aes(color = mean)) +
        scale_color_scico(direction = -1) +
        ggtitle("Sample from the fitted model")
```



3 Log-Gaussian Cox Processes

In this practical we will:

- Fit a homogeneous Point Process
- Fit a non-homogeneous Point Process
- Fit a LGCP (log-Gaussian Cox Process)

Libraries to load:

```
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
library(spatstat)
library(sf)
library(scico)
library(spatstat)
library(lubridate)
library(terra)
library(tidyterra)
```

---

3 The data

In this practical we consider the data `clmfires` in the `spatstat` library.

This dataset is a record of forest fires in the Castilla-La Mancha region of Spain between 1998 and 2007. This region is approximately 400 by 400 kilometres. The coordinates are recorded in kilometres. For more info about the data you can type:

```
?clmfires
```

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

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

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

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

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

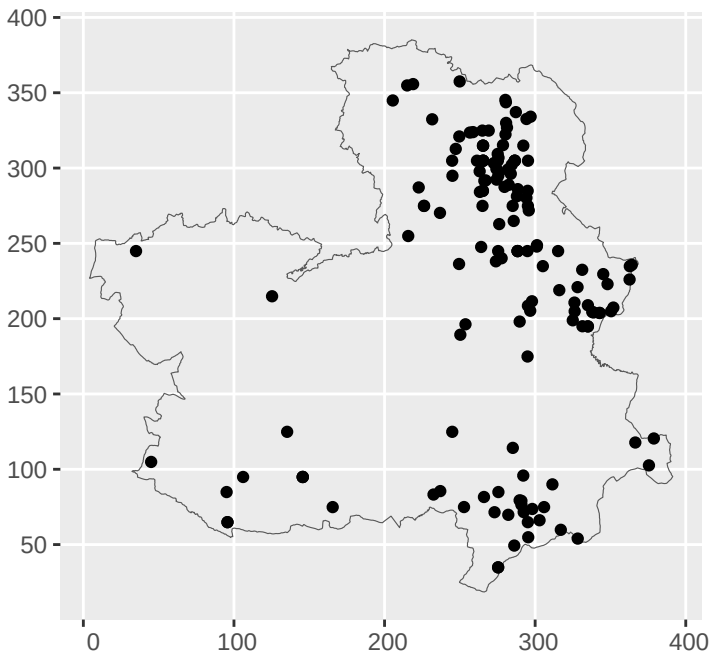


Figure 2: Distribution of the observed forest fires caused by lightning in Castilla-La Mancha in 2004

3 Fit a homogeneous Poisson Process

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

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

so the likelihood can be written as:

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

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

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

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

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

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

```
# define integration scheme

ips = st_sf(
geometry = st_sample(region, 1)) # some random location inside the domain
ips$weight = st_area(region) # integration weight is the area of the domain

cmp = ~ 0 + beta_0(1)

formula = geometry ~ beta_0

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

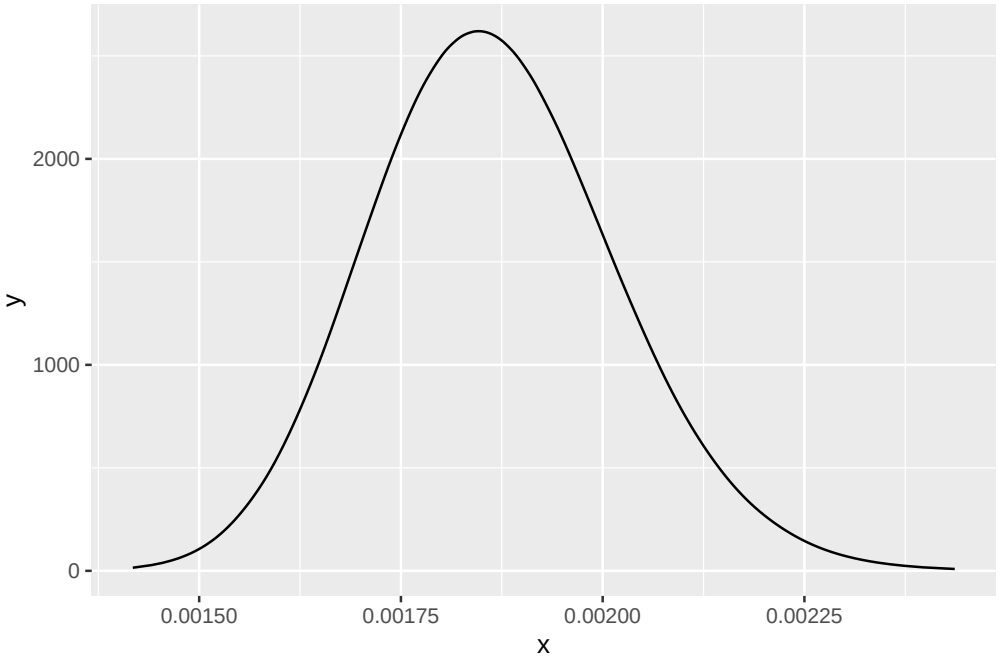
Task

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

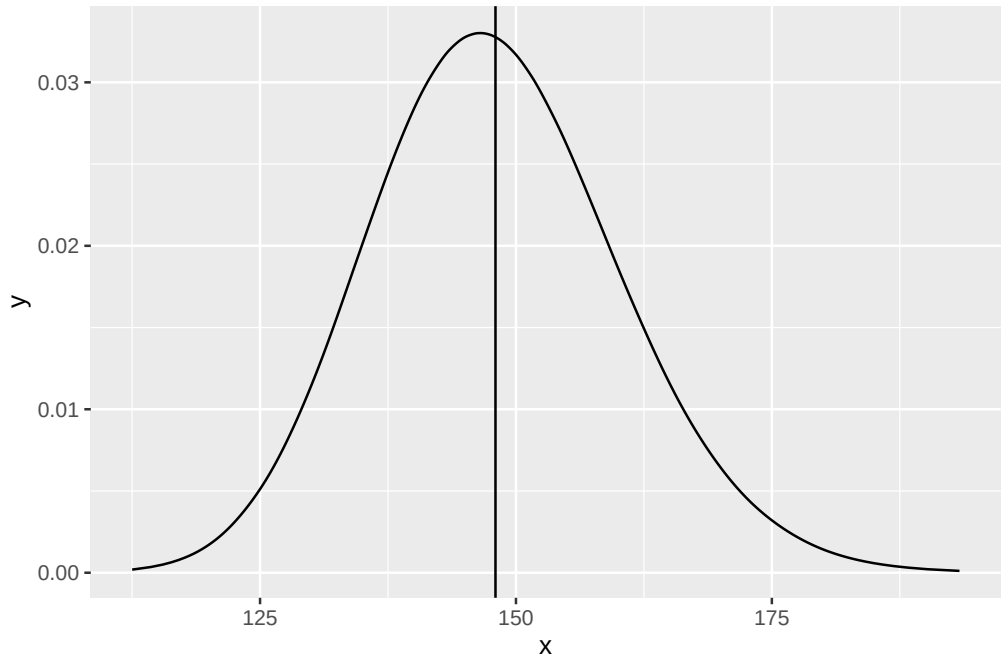
Take hint

Remember that in the *inlabru* framework we model the log intensity $\eta = log(\lambda)$
Click here to see the solution

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

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




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

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

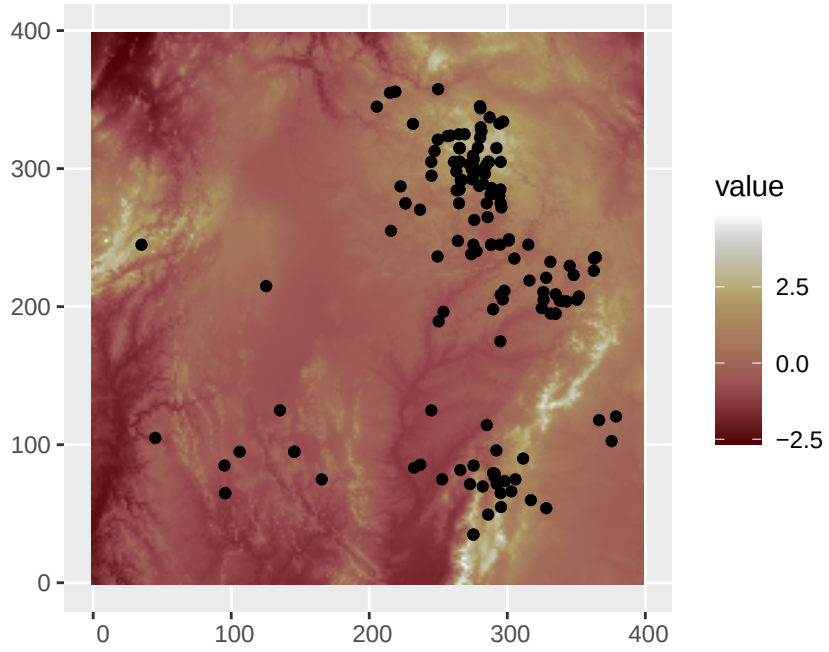


3 Fit an Inhomogeneous Poisson Process

The model above has the clear disadvantages that assumes a constant intensity and from Figure 2 we clearly see that this is not the case.

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

```
#|label: fig-altitude
#|fig-cap: "Distribution of the observed forest fires and scaled altitude"
#|
elev_raster = rast(clmfires.extra[[2]]$elevation)
elev_raster = scale(elev_raster)
ggplot() +
  geom_spatraster(data = elev_raster) +
  geom_sf(data = pp) +
  scale_fill_scico()
```



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

Our model is

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

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

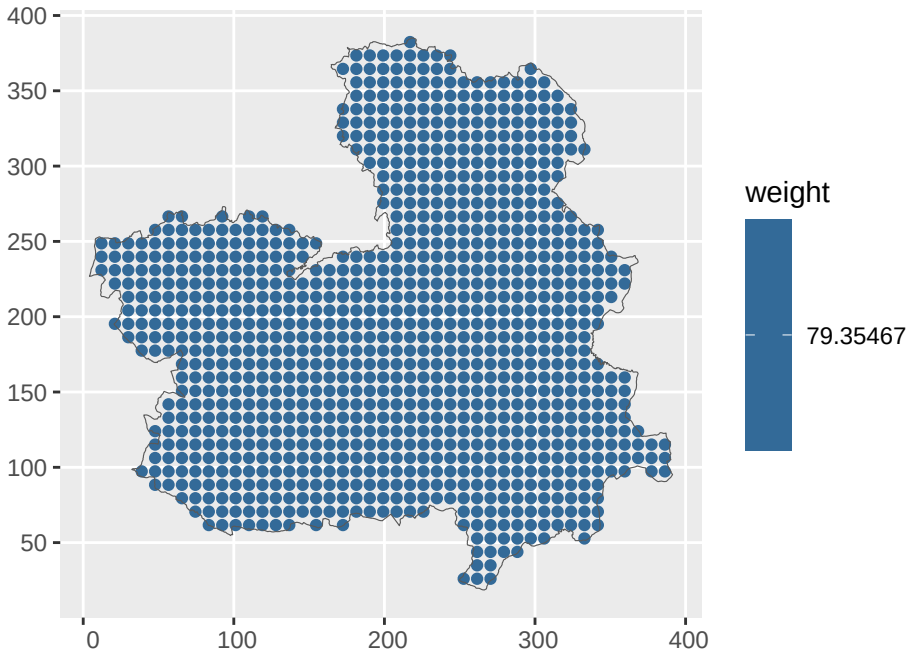
The likelihood becomes:

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

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

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

```
#|label: fig-int2
#|fig-cap: "Integration scheme."

n.int = 1000
ips = st_sf(geometry = st_sample(region,
            size = n.int,
            type = "regular"))

ips$weight = st_area(region) / n.int
ggplot() + geom_sf(data = ips, aes(color = weight)) + geom_sf(data= region, alpha = 0)
```



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

We can now fit the model:

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

Task

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

```
fit2$summary.fixed
```

```
               mean          sd 0.025quant    0.5quant 0.975quant        mode
Intercept -6.6372781 0.10431635 -6.8417344 -6.6372781 -
6.4328218 -6.6372781
elev       0.6977397 0.07387933  0.5529388  0.6977397  0.8425405  0.6977397
                  kld
Intercept 2.621372e-13
elev      0.000000e+00
```

Warning

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

Task

Rerun the model with the altitude as covariate, but this time change the integration scheme as follows:

```
n.int2 = 50

ips2 = st_sf(geometry = st_sample(region,
            size = n.int2,
            type = "regular"))
ips2$weight = st_area(region) / n.int2
```

What happens to the effect of the covariate?
Take hint
Re-run the model changing the integration scheme in the *ips* input of the *bru_obs()* function.
Click here to see the solution

```
lik_bis = bru_obs(data = pp,
              family = "cp",
              formula = formula,
              ips = ips2)

fit2bis = bru(cmp, lik_bis)

# you can the check the differences between the two models
rbind(fit2$summary.fixed,
fit2bis$summary.fixed)
```

```
                mean         sd 0.025quant   0.5quant 0.975quant       mode
Intercept  -6.6372781 0.10431635 -6.8417344 -6.6372781 -
6.4328218 -6.6372781
elev        0.6977397 0.07387933  0.5529388  0.6977397  0.8425405  0.6977397
Intercept1 -6.6402230 0.09943662 -6.8351152 -6.6402230 -
6.4453309 -6.6402230
elev1       0.6785744 0.06436394  0.5524234  0.6785744  0.8047254  0.6785744
                  kld
Intercept  2.621372e-13
elev       0.000000e+00
Intercept1 3.133385e-13
elev1      5.455274e-15
```

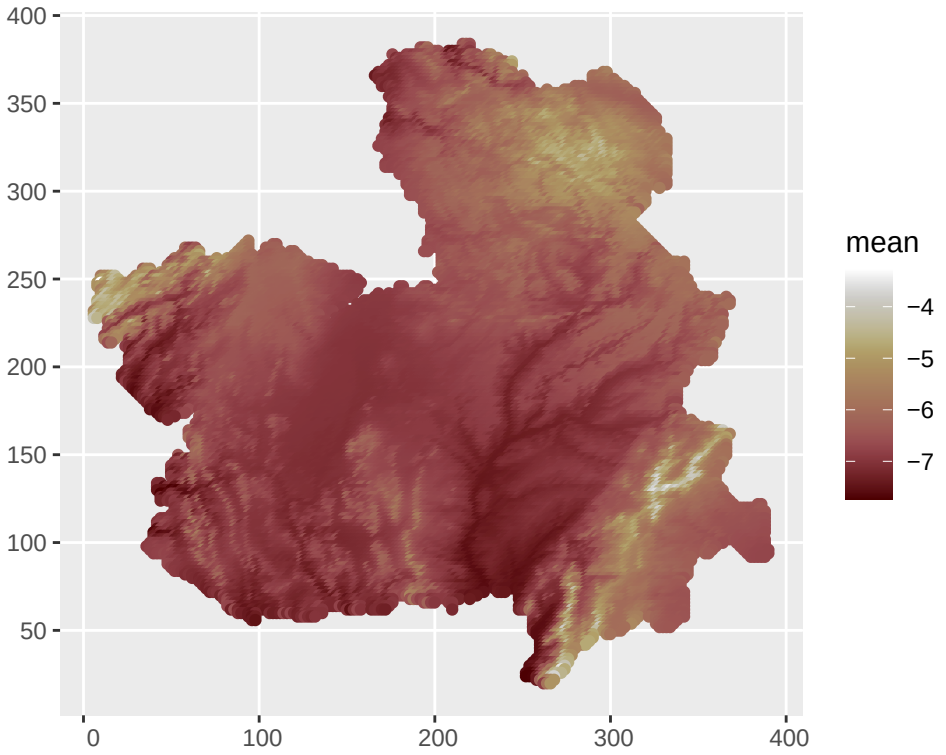
Task

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

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

Click here to see the solution

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

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



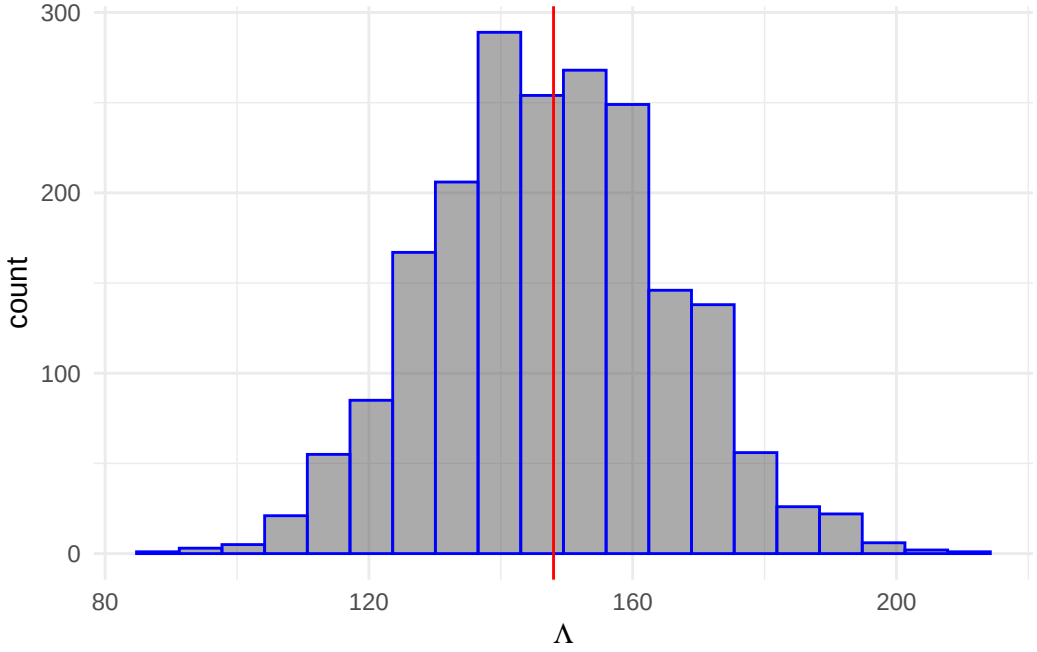
Finally, we want to use the fitted model to estimate the total number of fires over the whole region. To do this we first have to fine the expected number of fires as:

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

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

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

ggplot(data = data.frame(N = as.vector(N_fires))) +
  geom_histogram(aes(x = N),
                 colour = "blue",
                 alpha = 0.5,
                 bins = 20) +
  geom_vline(xintercept = nrow(pp),
```

```
            colour = "red") +
  theme_minimal() +
  xlab(expression(Lambda))
```



3 Fit a Log-Gaussian Cox Process

Finally we want to fit a LGCP with log intensity:

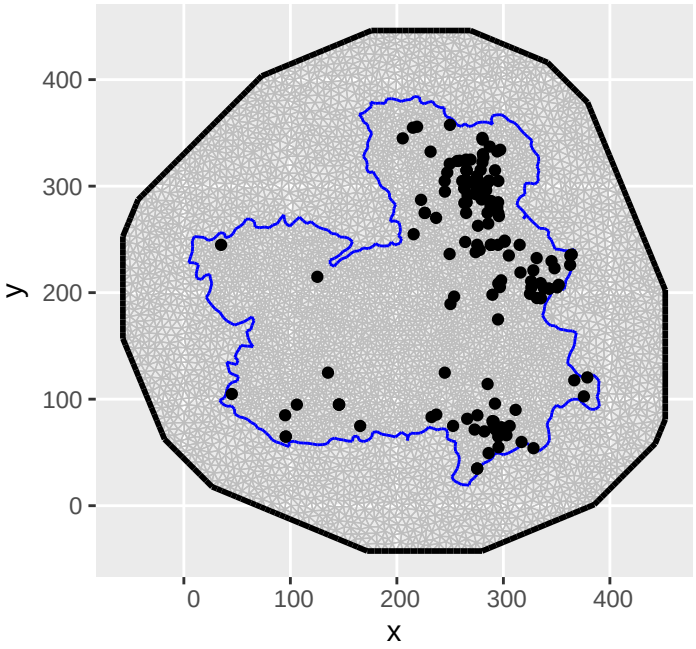
$$\log(s) = \beta_0 + \beta_1 x + u(s)$$

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

3.4.1 Define the mesh

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

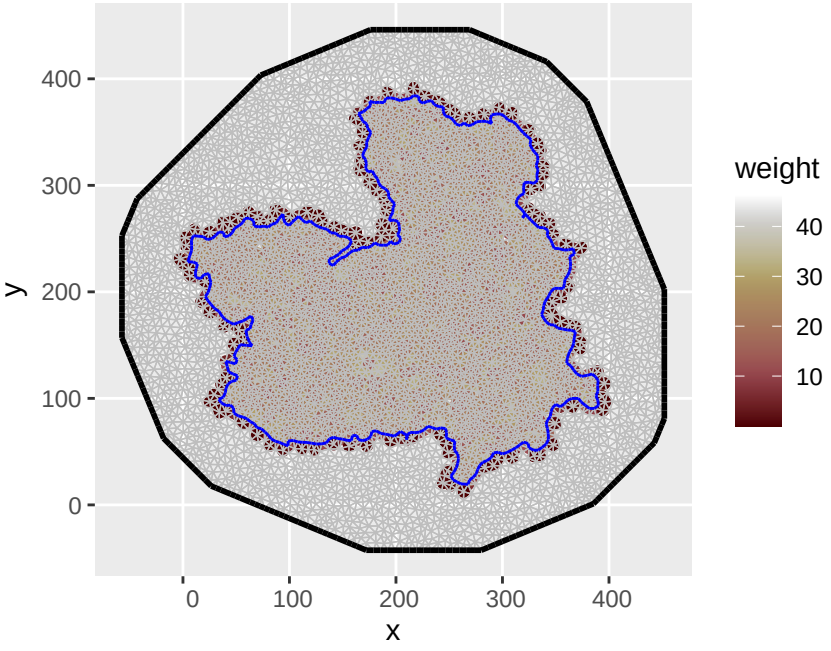
```
mesh = fm_mesh_2d(boundary = region,
                  max.edge = c(5, 10),
                  cutoff = 4, crs = NA)

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




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

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

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

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



3.4.2 Run the model

Task

Set up the components and the formula for the model above by completing the code below and run the model.

```
cmp = ~ ...

formula = geometry ~ ...

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

fit3 = bru(cmp, lik)
```

Take hint
The model has three components: intercept, linear effect of altitude and the spatial GRF
Click here to see the solution

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

formula = geometry ~ Intercept + space + elev

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

fit3 = bru(cmp, lik)
```

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

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

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

Here, we notice that there is a single point for which elevation values are missing (see Figure 3 the red point that lies outside the raster extension).

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

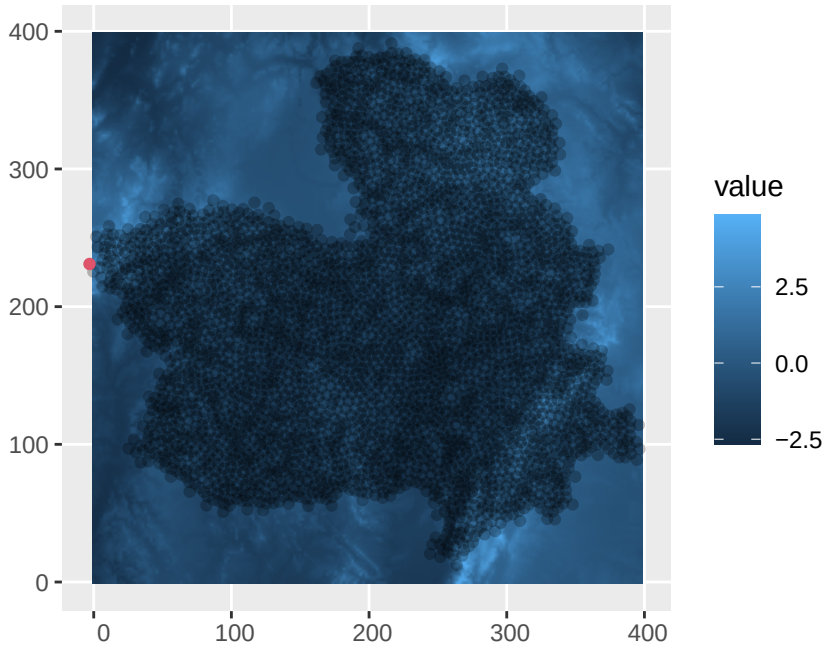
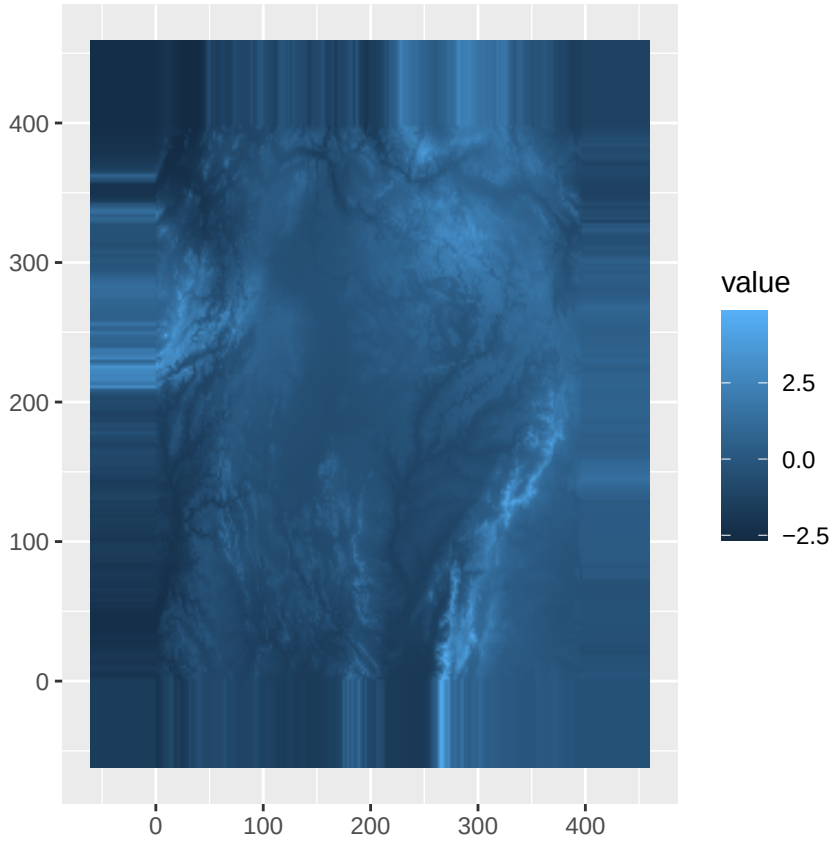


Figure 3: Integration scheme for numerical approximation of the stochastic integral in La Mancha Region

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

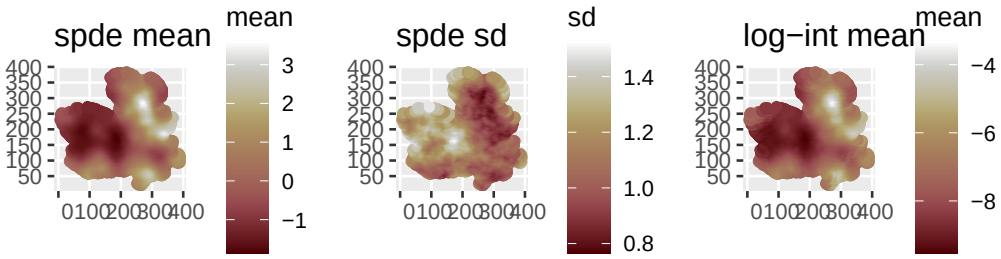


> **Note**
>
> The `bru_fill_missing()` function was added mainly to handle very local infilling on domain boundaries. For properly missing data, one should consider doing a proper model of the spatial field instead.

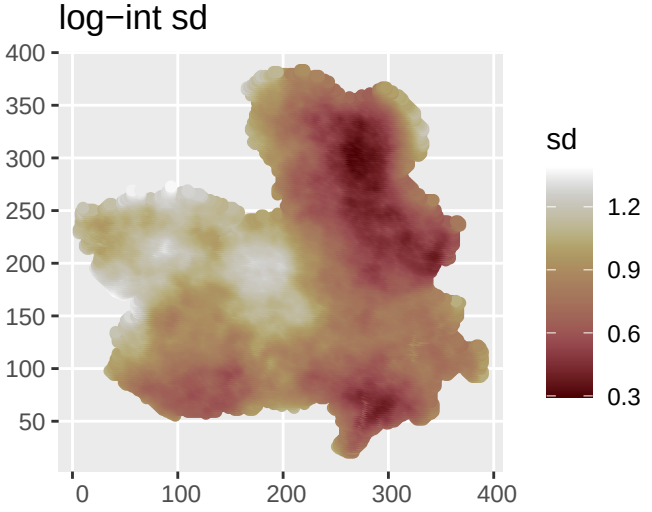
3 Results

> Task
>
> Plot the estimated mean and standard deviation of the spatial GF and the log-intensity over the domain of interest
> Take hint
> Use the *fm_pixels()* and *predict()* functions.
> Click here to see the solution

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

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

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

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



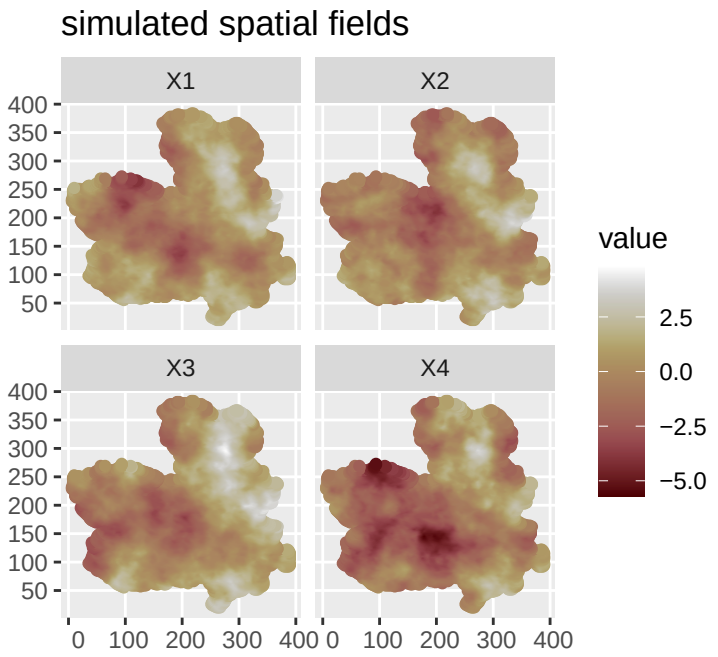

```
ggplot(data=preds$log_int) +
  geom_sf(aes(color = sd)) +
  scale_color_scico() +
 ggtitle("log-int sd") +
  plot_layout(ncol=2)
```



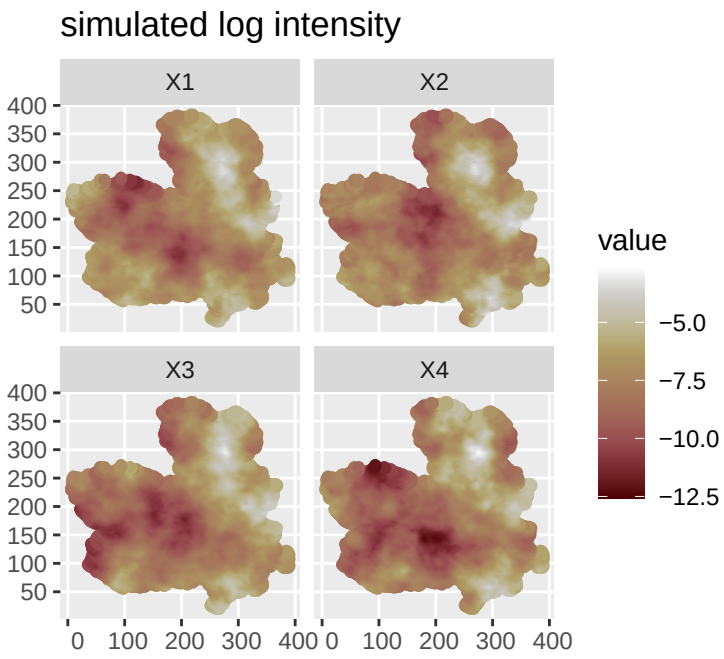
Instead of just looking at the posterior mean and standard deviation, it can be usefull to

look at `simulated fields` from the posterior distribution. This is because the mean field is, by definition, smoother than any realization of the field. So looking at simulation can give us a better idea of how the field might look like. We can do this using the *generate()* function:

```
sim_fields = generate(fit3, pxl, ~data.frame(spde = space,
                                             log_int = Intercept + space + elev),
                      n.samples = 4)

cbind(pxl,sapply(sim_fields, function(x) x$spde)) %>%
  pivot_longer(-geometry) %>%
  ggplot() + geom_sf(aes(color = value)) +
  facet_wrap(.~name) + scale_color_scico() +
  ggtitle("simulated spatial fields")
```




```
cbind(pxl,sapply(sim_fields, function(x) x$log_int)) %>%
  pivot_longer(-geometry) %>%
  ggplot() + geom_sf(aes(color = value)) +
  facet_wrap(.~name) + scale_color_scico() +
  ggtitle("simulated log intensity")
```


simulated log intensity
X1
X2
X3
X4
400
350
300
250
200
150
100
50
0
100
200
300
400
value
−5.0
−7.5
−10.0
−12.5