PRACTICAL 2

Aim of this practical:

In this practical we are going to look at commonly used spatial models

1. CAR model for areal data
2. A geostatistical model for binary presence-absence species distribution data
3. A Point process model for georeferenced species occurrence data

0 Areal (lattice) data

In this practical we will:

- Explore tools for areal spatial data wrangling and visualization.
- Learn how to fit an areal model in `inlabru`

In areal data our measurements are summarised across a set of discrete, non-overlapping spatial units such as postcode areas, health board or pixels on a satellite image. In consequence, the spatial domain is a countable collection of (regular or irregular) areal units at which variables are observed. Many public health studies use data aggregated over groups rather than data on individuals - often this is for privacy reasons, but it may also be for convenience.

In the next example we are going to explore data on respiratory hospitalisations for Greater Glasgow and Clyde between 2007 and 2011. The data are available from the `CARBayesdata` R Package:

```
library(CARBayesdata)

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

The `pollutionhealthdata` contains the spatiotemporal data on respiratory hospitalisations, 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 were the measruments were taken
- `observed`: observed numbers of hospitalisations due to respiratory disease.
- `expected`: expected numbers of hospitalisations due to respiratory disease computed using indirect standardisation from Scotland-wide respiratory hospitalisation 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).

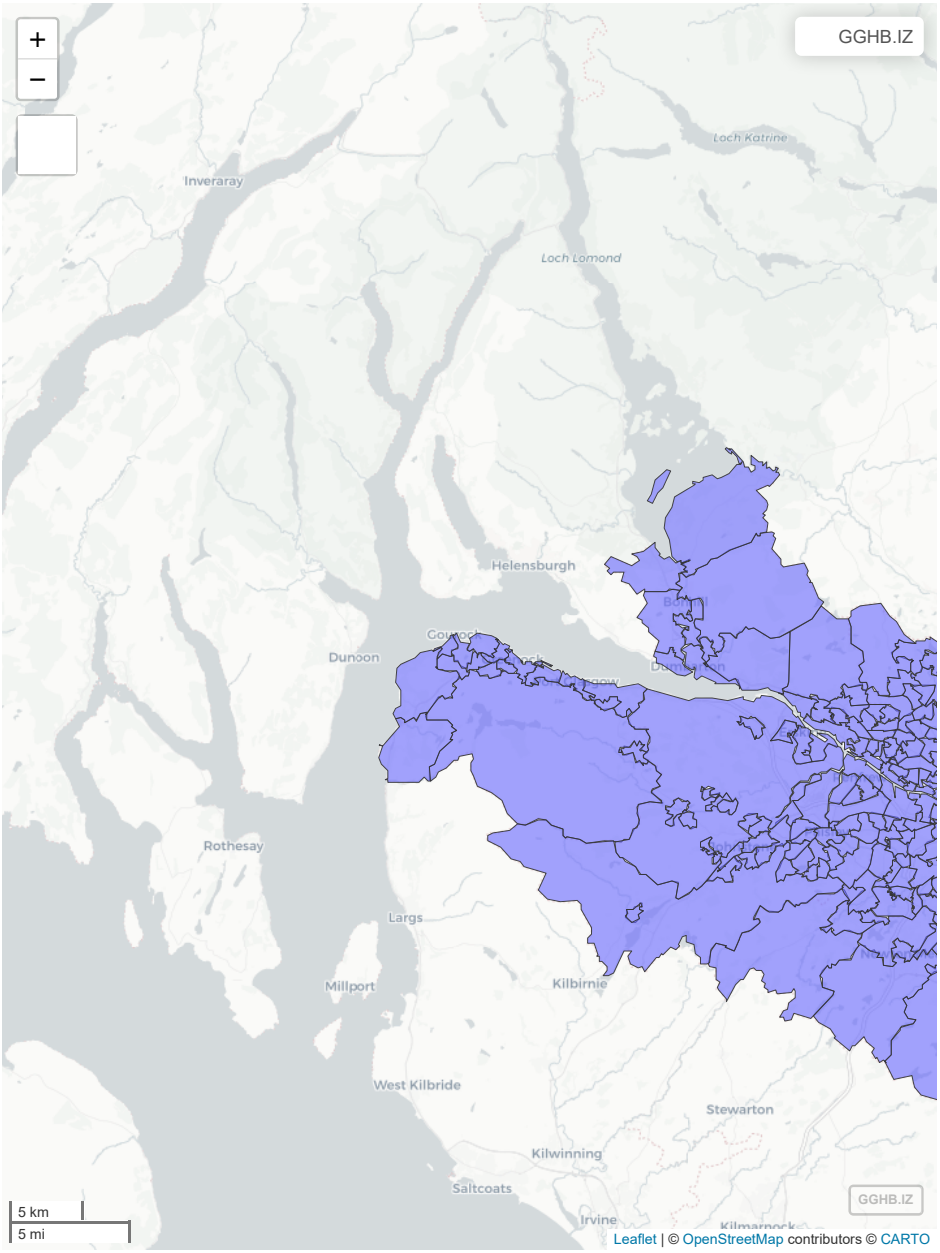


Figure 1: Greater Glasgow and Clyde health board represented by 271 Intermediate Zones

0.1.1 Maipulating and visualizing areal data

Let's start by loading useful libraries:

```
library(dplyr)
library(INLA)
library(ggplot2)
```

```
library(patchwork)
library(inlabru)
library(mapview)
library(sf)

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

The `sf` package allows us to work with vector data which is used to represent points, lines, and polygons. It can also be used to read vector data stored as a shapefiles.

First, lets combine both data sets based on the Intermediate Zones (IZ) variable using the `merge` function from `base` R, and select only one year of data

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

In epidemiology, disease risk is usually estimated using Standardized Mortality Ratios (SMR). The SMR for a given spatial areal unit i is defined as the ratio between the observed (Y_i) and expected (E_i) number of cases:

$$SMR_i = \frac{Y_i}{E_i}$$

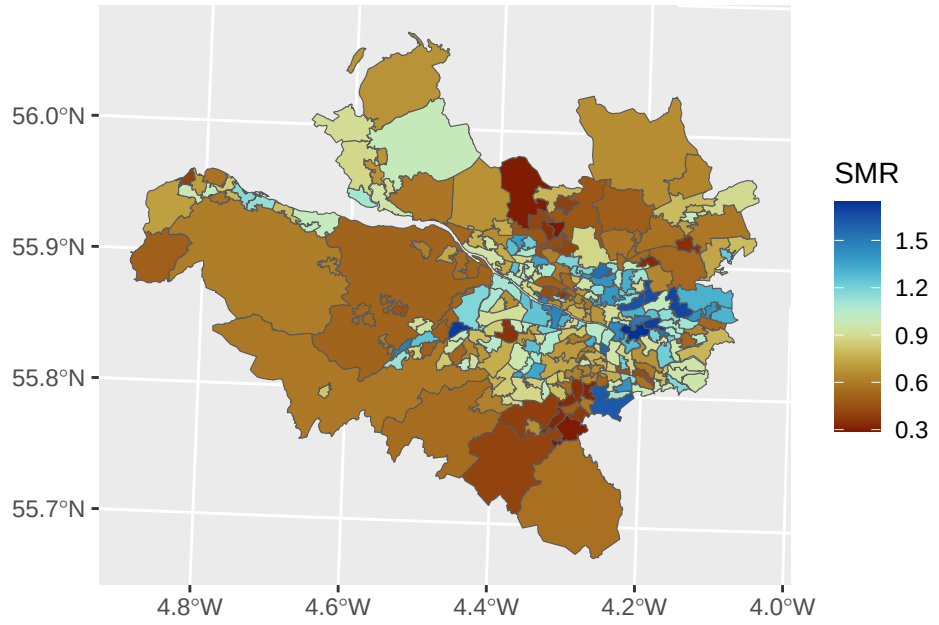
A value $SMR > 1$ indicates that there are more observed cases than expected which corresponds to a high risk area. On the other hand, if $SMR < 1$ then there are fewer observed cases than expected, suggesting a low risk area.

We can manipulate `sf` objects the same way we manipulate standard data frame objects via the `dplyr` package. Lets use the pipeline command %>% and the `mutate` function to calculate the yearly SMR values for each IZ:

```
resp_cases <- resp_cases %>%
  mutate(SMR = observed/expected )
```

Now we use `ggplot` to visualize our data by adding a `geom_sf` layer and coloring it according to our variable of interest (i.e., SMR).

```
ggplot()+
  geom_sf(data=resp_cases,aes(fill=SMR))+
  scale_fill_scico(palette = "roma")
```



As with the other types of spatial modelling, our goal is to observe and explain spatial variation in our data. Generally, we are aiming to produce a smoothed map which summarises the spatial patterns we observe in our data.

0.1.2 Spatial neighbourhood structures

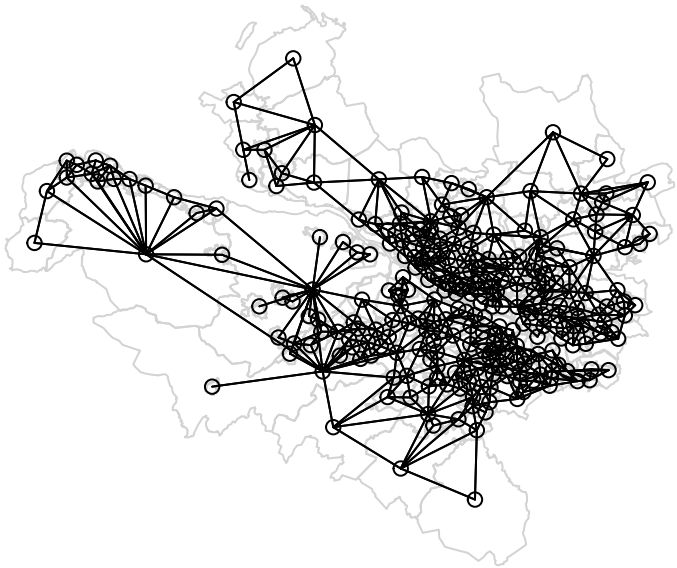
A key aspect of any spatial analysis is that observations closer together in space are likely to have more in common than those further apart. This can lead us towards approaches similar to those used in time series, where we consider the spatial *closeness* of our regions in terms of a *neighbourhood structure*.

The function `poly2nb()` of the `spdep` package can be used to construct a list of neighbors based on areas with contiguous boundaries (e.g., using Queen contiguity).

```
library(spdep)

W.nb <- poly2nb(GGHB.IZ,queen = TRUE)
W.nb
```

```
Neighbour list object:
Number of regions: 271
Number of nonzero links: 1424
Percentage nonzero weights: 1.938971
Average number of links: 5.254613
2 disjoint connected subgraphs
```

```
plot(st_geometry(GGHB.IZ), border = "lightgray")
plot.nb(W.nb, st_geometry(GGHB.IZ), add = TRUE)
```



> **Note**
>
> You could use the `snap` argument within `poly2nb` to set a distance at which the different regions centroids are consider neighbours.

With this neighborhood matrix, we can then fit a conditional autoregressive (CAR) model. One of the most popular CAR approaches to model spatial correlation is the Besag model a.k.a. Intrinsic Conditional Autoregressive (ICAR) model.

The conditional distribution for u_i given $\mathbf{u}_{-i} = (u_i, \dots, u_{i-1}, u_{i+1}, \dots, u_n)^T$ is

$$u_i|\mathbf{u}_{-i} \sim N\left(\frac{1}{d_i}\sum_{j\sim i} u_j, \frac{1}{d_i\tau_u}\right)$$

where τ_u is the precision parameter, $j \sim i$ denotes that i and j are neighbors, and d_i is the number of neighbors. Thus, the mean of u_i is equivalent to the the mean of the effects over all neighbours, and the precision is proportional to the number of neighbors. The joint distribution is given by:

$$\mathbf{u}|\tau_u \sim N\left(0, \frac{1}{\tau_u}Q^{-1}\right),$$

Where Q denotes the structure matrix defined as

$$Q_{i,j} = \begin{cases} d_i, & i = j \\ -1, & i \sim j \\ 0, & \text{otherwise} \end{cases}$$

This structure matrix directly defines the neighbourhood structure and is sparse. We can compute the adjacency matrix using the function `nb2mat()` in the `spdep` library. Then convert the adjacency matrix into the precision matrix **Q** of the CAR model as follows:

```
library(spdep)
R <- nb2mat(W.nb, style = "B", zero.policy = TRUE)
diag = apply(R,1,sum)
Q = -R
diag(Q) = diag
```

The ICAR model accounts only for spatially structured variability and does not include a limiting case where no spatial structure is present. Therefore, it is typically combined with an additional unstructured random effect $z_i|\tau_z \sim N(0, \tau_z^{-1})$. The resulting model $v_i = u_i + z_i$ is known as the Besag-York-Mollié model (BYM) which is an extension to the intrinsic CAR model that contains an i.i.d. model component.

0.1.3 Fitting an ICAR model in `inlabru`

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

$$\eta_i = \text{log}(\lambda_i) = \beta_0 + u_i + z_i$$

where

- β_0 is a common intercept
- **u** $= (u_1, \dots, u_k)$ is a conditional Autoregressive model (CAR) with precision matrix τ_u**Q**
- **z** $= (z_1, \dots, z_k)$ is an unstructured random effect with precision τ_z

Stage 3 Hyperparameters

The hyperparameters of the model are τ_u and τ_z

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

> Question
>
> 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?

0.1.4 Areal model predictions

We now look at the predictions over space.

Task

Complete the code below to produce:

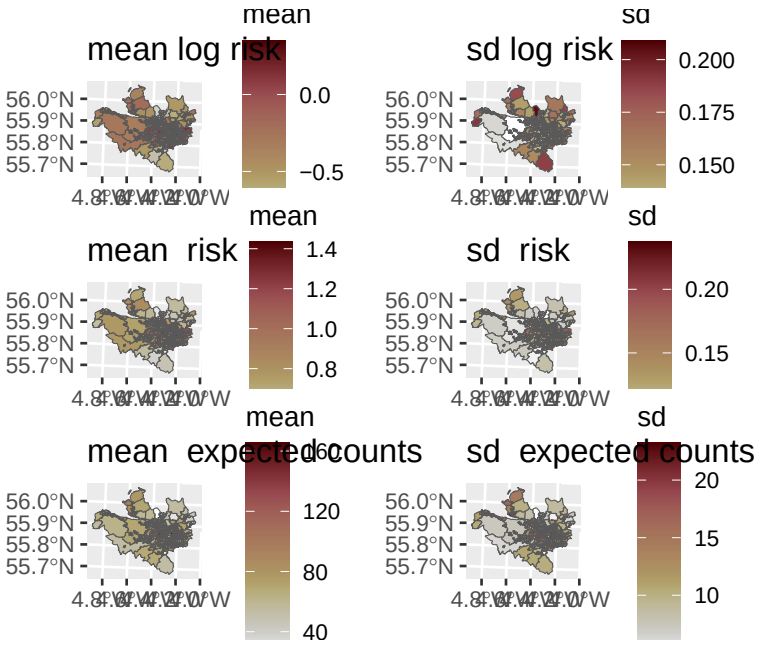
1. prediction of the linear predictor η_i
2. The risk rate λ_i
3. The expected cases E_i exp(λ_i) over the whole space of interest.

Plot the mean and sd of the resulting surfaces.

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

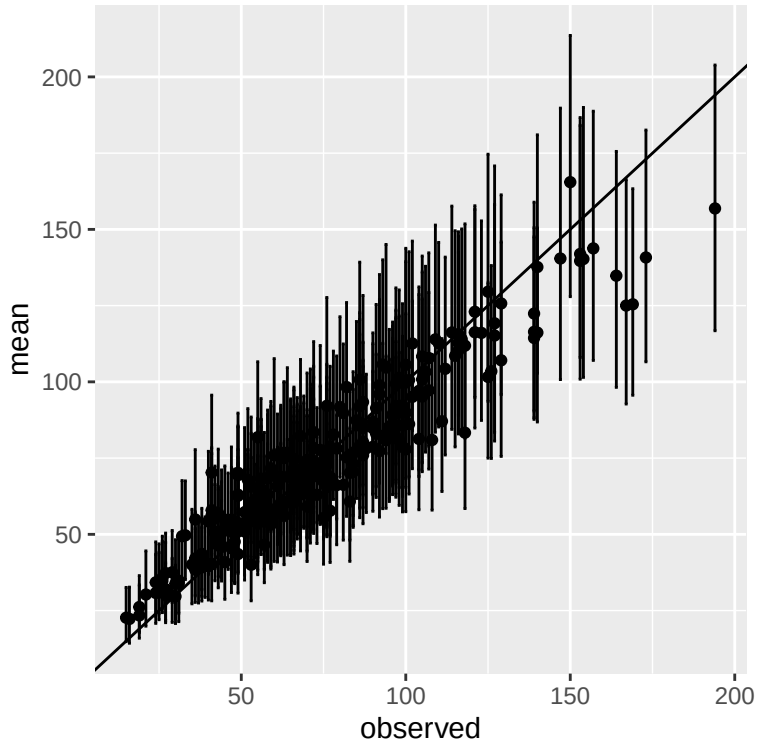
See Solution

```
# 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$log_risk, aes(fill = mean)) + scale_fill_scico(direction = -1) +
  ggtitle("mean log risk")
p2 = ggplot() +
  geom_sf(data = pred$log_risk, aes(fill = sd)) + scale_fill_scico(direction = -1) +
  ggtitle("sd log risk")

p3 = ggplot() +
  geom_sf(data = pred$risk, aes(fill = mean)) + scale_fill_scico(direction = -1) +
  ggtitle("mean  risk")

p4 = ggplot() +
  geom_sf(data = pred$risk, aes(fill = sd)) +
  scale_fill_scico(direction = -1) +
  ggtitle("sd  risk")

p5 = ggplot() + geom_sf(data = pred$cases, aes(fill = mean)) + scale_fill_scico(direction
  ggtitle("mean  expected counts")
p6 = ggplot() + geom_sf(data = pred$cases, aes(fill = sd)) + scale_fill_scico(direction =
  ggtitle("sd  expected counts")

p1 + p2 + p3 + p4 +p5 + p6 + plot_layout(ncol=2)
```



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)
```



Here we are predicting the *mean* of counts, not the counts. Predicting the counts is beyond the scope of this short course but you can check the supplementary material below.

Supplementary Material

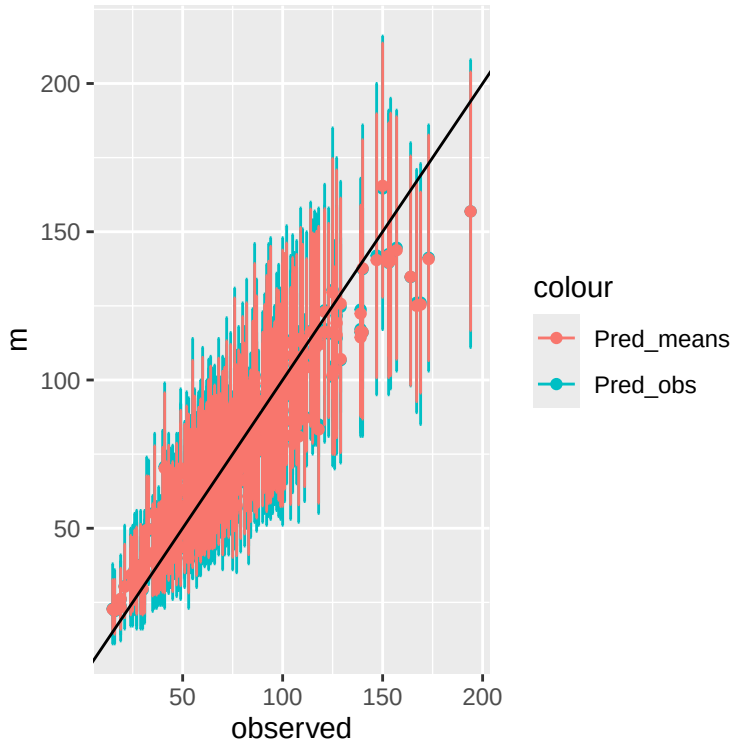
Posterior predictive distributions, i.e., $\pi(y_i^{\text{new}}|\mathbf{y})$ are of interest in many applied problems. The `bru()` function does not return predictive densities. In the previous step we have computed predictions for the `expected counts` $\pi(E_i\lambda_i|\mathbf{y})$.
The predictive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

1. Simulate n replicates of $g^k = E_i\lambda_i$ for $k = 1, \ldots, n$ 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.
4.

```
# 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)
                         )
# Plot the observations against the predicted new counts and the predicted expected counts

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)
```



0 Geostatistical data

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

- Explore tools for geostatistical spatial data wrangling and visualization.
- Learn how to fit a geostatistical model in `inlabru`
- Learn how to add spatial covariates to the model
- Learn how to do predictions

Geostatistical data are the most common form of spatial data found in environmental setting. In these data we regularly take measurements of a spatial ecological or environmental process at a set of fixed locations. This could be data from transects (e.g, where the height of trees is recorded), samples taken across a region (e.g., water depth in a lake) or from monitoring stations as part of a network (e.g., air pollution). In each of these cases, our goal is to estimate the value of our variable across the entire space.

Let D be our two-dimensional region of interest. In principle, there are infinite locations within D, each of which can be represented by mathematical coordinates (e.g., latitude and longitude). We then can identify any individual location as $s_i = (x_i, y_i)$, where x_i and y_i are their coordinates.

We can treat our variable of interest as a random variable, Z which can be observed at any location as $Z(\mathbf{s}_i)$.

Our geostatistical process can therefore be written as:

$$\{Z(\mathbf{s}); \mathbf{s} \in D\}$$

In practice, our data are observed at a finite number of locations, m, and can be denoted as:

$$z = \{z(\mathbf{s}_1), \dots z(\mathbf{s}_m)\}$$

In the next example, 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 survey. The `qcs_grid` data contain the depth values stored as 2×2 km grid for Queen Charlotte Sound.

0.2.1 Exploring and visualizing species distribution data

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 an `sf` object and assign the rough coordinate reference to it:

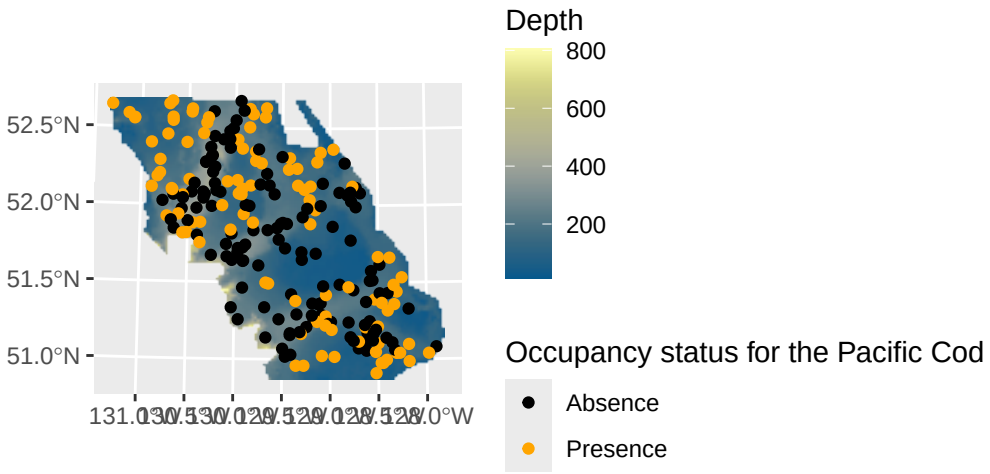
```
pcod_sf =   st_as_sf(pcod_df, coords = c("lon","lat"), crs = 4326)
pcod_sf = st_transform(pcod_sf,
                       crs = "+proj=utm +zone=9 +datum=WGS84 +no_defs +type=crs +units=km" )
```

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

```
library(terra)
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 load also the `tidyterra` library.

```
library(tidyterra)
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("")
```



0.2.2 Fitting a spatial geostatistical species distribution 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!!

0.2.3 The workflow

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

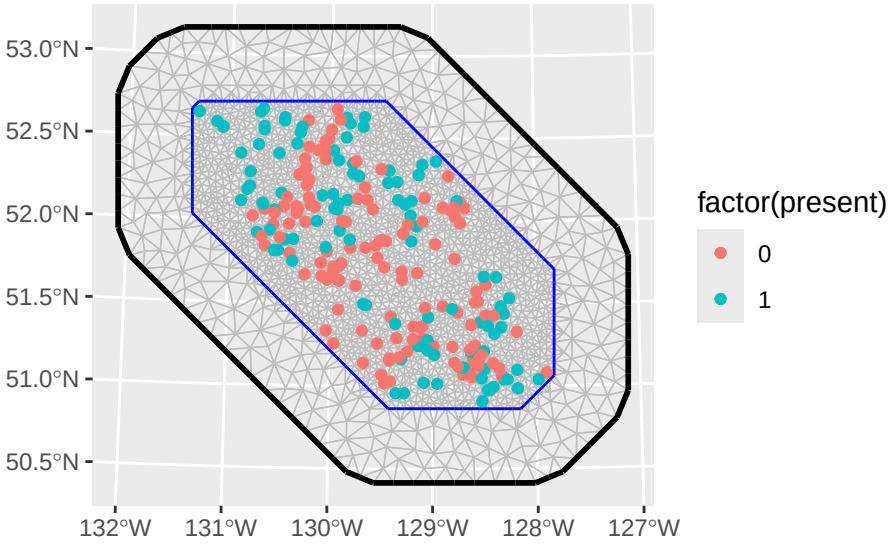
1. Build the mesh
2. Define the SPDE representation of the spatial GF. This includes defining the priors for the range and sd of the spatial GF
3. Define the *components* of the linear predictor. This includes the spatial GF and all eventual covariates

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

0.2.3.1 Step 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_mesh_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
                  cutoff = 2,
                  max.edge = c(7,20),      # 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))) +
  xlab("") + ylab("")
```



Task

Look at the documentation for the `fm_mesh_2d` function typing

```
?fm_mesh_2d
```

Experiment with the different options and create different meshes (see here for further details on mesh construction).
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 artifact in the estimated variance.

0.2.3.2 Step 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 parameters 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

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

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))
```

Question

Considering the `pcod_sf` spatial extension and type of the data, which of the previous choices is more reasonable?
Remember that a prior should be reasonable..but the model should not totally depend on it.
Take hint
You can use the `summary()` function to check the coordinate range of an `sf` object.

- (A) spde_model1
- (B) spde_model2

- (C) spde_model3

0.2.3.3 Step 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 data frame we use (`pcod_sf`) is an `sf` object the input in the `space()` component is the geometry of the dataset.

0.2.3.4 Step 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")
```

0.2.3.5 Step 5. Run the model

Finally we are ready to run the model

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

0.2.4 Model results

0.2.4.1 Hyperparameters

Task

What are the posterior for the range ρ and the standard deviation σ? Plot the posterior together with the prior for both parameters.
Take hint
The `spde.posterior()` can be used to calculate the posterior distribution of the range and variance of a model's SPDE component. (type `?spde.posterior` for further detials)
Click here to see the solution

```
# Extract marginal for the range

library(patchwork)
spde.posterior(fit1, "space", what = "range") %>% plot() +
spde.posterior(fit1, "space", what = "log.variance") %>% plot()
```

0.2.5 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

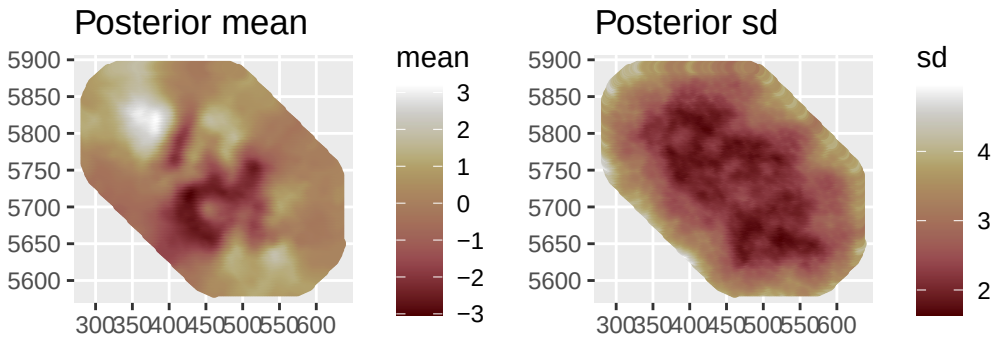
```
pxl = fm_pixels(mesh)
```

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



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

0.2.6 An alternative model (including spatial covariates)

We now want to check if the `depth` covariate has an influence 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

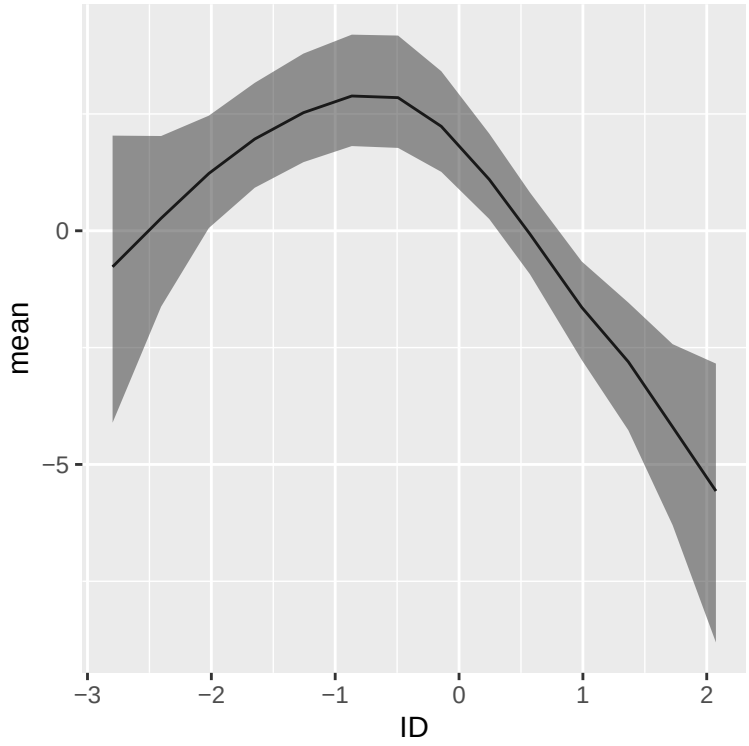
Fit model 1. Define components, observation model and use the `bru()` function to estimate the parameters.
Note Use the scaled version of the covariate stored in `depth_r$depth_scaled`.
What is the liner effect of depth on the logit probability?
Take hint
The `pcod_sf` object already contains the `depth_scaled` containing the squared depth values at each location. However, `inlabru` also allows to specify a raster object directly in the model components. If your raster contains multiple layers, then de desired layer can be called using the $ symbol (e.g., `my_raster$layer_1`).
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 classes. 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)
```



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")) %>% dplyr::select(-depth)
```

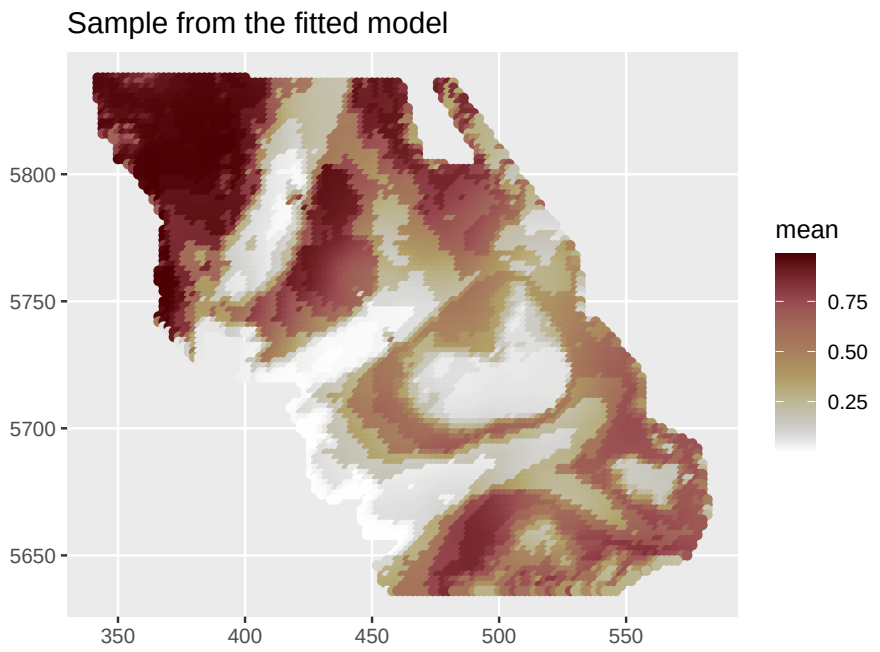
Task

Create a map of predicted *probability* from Model 3 by predicting prediction over `pxl1`. 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

```
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")
```



0 Point process data

In this practical we are going to fit a log Gaussian Cox Proces (LGCP) model to point-referenced data. We will:

- Learn how to fit a LGCP model in `inlabru`
- Learn how to add spatial covariates to the model
- Learn how to do predictions

In **point processes** we measure the locations where events occur (e.g. trees in a forest, earthquakes) and the coordinates of such occurrences are our data. A spatial point process is a random variable operating in continuous space, and we observe realisations of this variable as point patterns across space.

Consider a fixed geographical region A. The set of locations at which events occur are denoted $\mathbf{s} = s_1, \dots, s_n$. We let $N(A)$ be the random variable which represents the number of events in region A.

We typically assume that a spatial point pattern is generated by an unique point process over the whole study area. This means that the delimitation of the study area will affect the observed point patters.

We can define the intensity of a point process as the expected number of events per unit area. This can also be thought of as a measure of the density of our points. In some cases,

the intensity will be constant over space (homogeneous), while in other cases it can vary by location (inhomogeneous or heterogenous).

In the next example we will be looking at the location where forest fires occurred in the Castilla-La Mancha region of Spain between 1998 and 2007.

0.3.1 Point-referenced data visualization

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) %>%
                dplyr::mutate(x = x,
                              y = y),
              coords = c("x","y"),
              crs = NA) %>%
  dplyr::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)
```

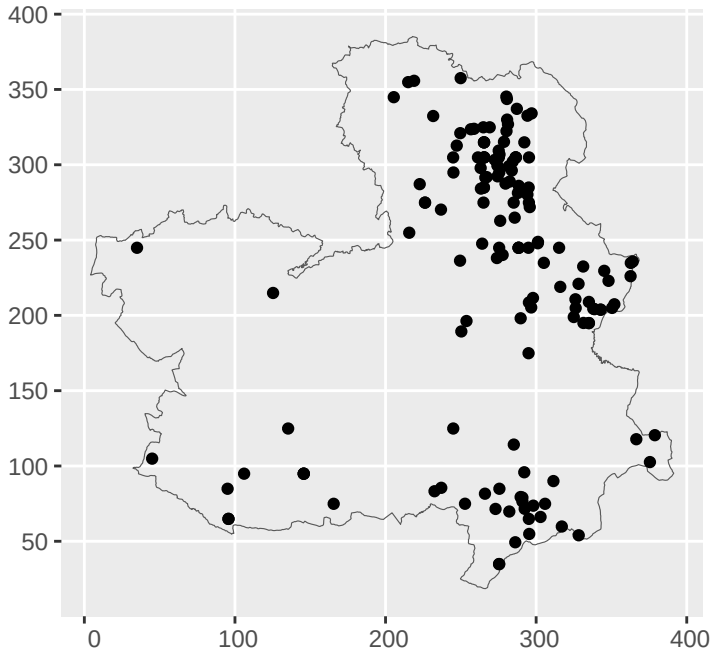


Figure 2: Distribution of the observed forest fires caused by lightning in Castilla-La Mancha in 2004

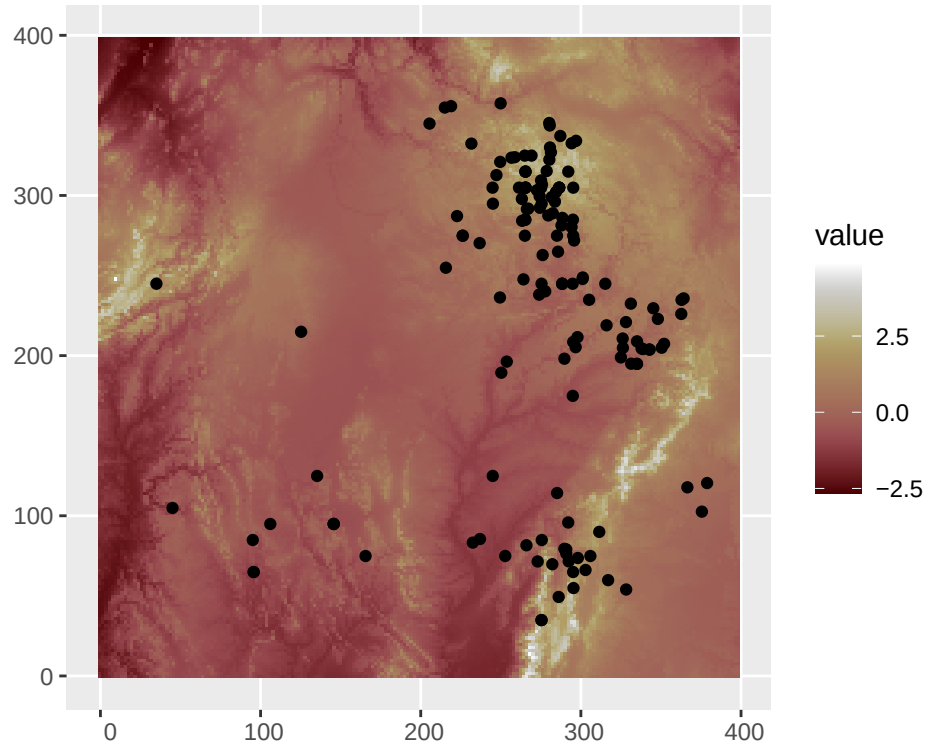
The library `spatstat` contains also some covariates that can help explain the fires distribution. We can a raster for the scaled values of elevation using the following code:

```
elev_raster = rast(clmfires.extra[[2]]$elevation)
elev_raster = scale(elev_raster)
```

Task

Using `tidyterra` and `ggplot`, produce a map of the elevation profile in La Mancha region and overlay the spatial point pattern of the fire locations. Use an appropriate colouring scheme for the elevation values. Do you see any pattern?
Take hint
You can use the `geom_spatraster()` to add a raster layer to a ggplot object. Furthermore the `scico` library contains a nice range of coloring palettes you can choose, type `scico_palette_show()` to see the color palettes that are available.
Click here to see the solution

```
library(ggplot2)
library(tidyterra)
library(scico)

ggplot() +
  geom_spatraster(data = elev_raster) +
  geom_sf(data = pp) + scale_fill_scico()
```



0.3.2 Workflow for Fitting a LGCP model

The procedure for fitting a point process model in `inlabru`, specifically a log-Gaussian Cox process, follows a similar workflow to that of a geostatistical model, these are:

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 observational model
5. Run the Model

0.3.2.1 Step 1. Building the mesh for a LGCP First, we need to create the mesh used to approximate the random field. When analyzing point patterns, mesh nodes (integration points) are not typically placed at point locations. Instead, a mesh is created using the `fm_mesh_2d()` function from the `fmesher` library with boundary being our study area.

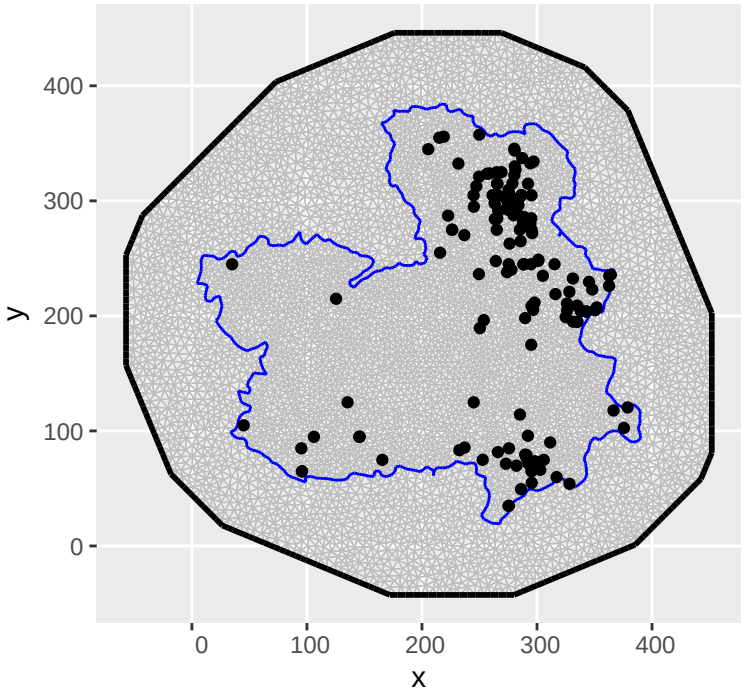
Key parameters in mesh construction include: `max.edge` for maximum triangle edge lengths, `offset` for inner and outer extensions (to prevent edge effects), and cutoff to avoid overly small triangles in clustered areas.

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.

```
# mesh options

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

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



0.3.2.2 Step 2. Defining the SPDE model We can now define our SPDE model using the `inla.spde2.pcmatern` function. To help us chose some sensible model parameters it is often useful to consider the spatial extension of our study.

```
st_area(region)
```

```
[1] 79354.67
```

We can use PC-priors for the range ρ and the standard deviation σ of the Matérn process

- Define the prior for the range `prior.range = (range0,Prange)` Prob$(\rho < \rho_0) = p_\rho$
- Define the prior for the range `prior.sigma = (sigma0,Psigma)` Prob$(\sigma > \sigma_0) = p_\sigma$

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

0.3.2.3 Step 3. Defining model components **Stage 1** Model for the response

The total number of points in the study region is a Poisson random variable with a spatially varying intensity and log-likelihood given by :

$$l(\beta; s) = \sum_{i=1}^{m} \log[\lambda(s_i)] - \int_A \lambda(s) ds.$$

The integral in this expression can be interpreted as the expected number of points in the whole study region. However, the integral of the intensity function has no close form solution and thus we need to approximate it using numerical integration. `inlabru` has implemented the `fm_int` function to create integration schemes that are especially well suited to integrating the intensity in models with an SPDE effect. We strongly recommend that users use these integration schemes in this context. See `?fm_int` for more information.

```
# build integration scheme
ips = fm_int(mesh,
             samplers = region)
```

Now, for a point process models, the spatial covariates (i.e., the elevation raster) have to be also available at both data-points and quadrature locations. We can check this using the `eval_spatial` function from `inlabru` :

```
eval_spatial(elev_raster,pp) %>% is.na() %>% any()
```

```
[1] FALSE
```

```
eval_spatial(elev_raster,ips) %>% is.na() %>% any()
```

```
[1] TRUE
```

Here, we notice that there is a single point that 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:

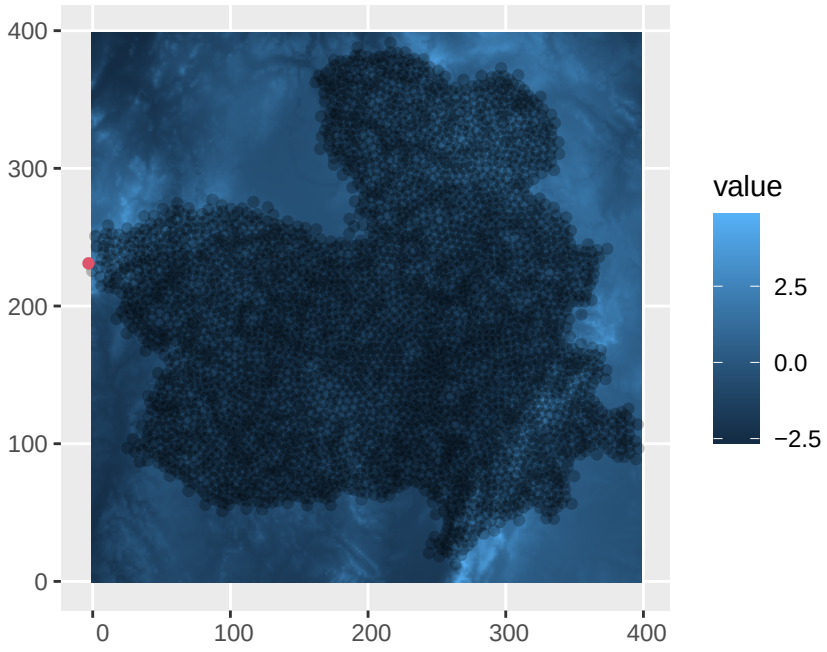
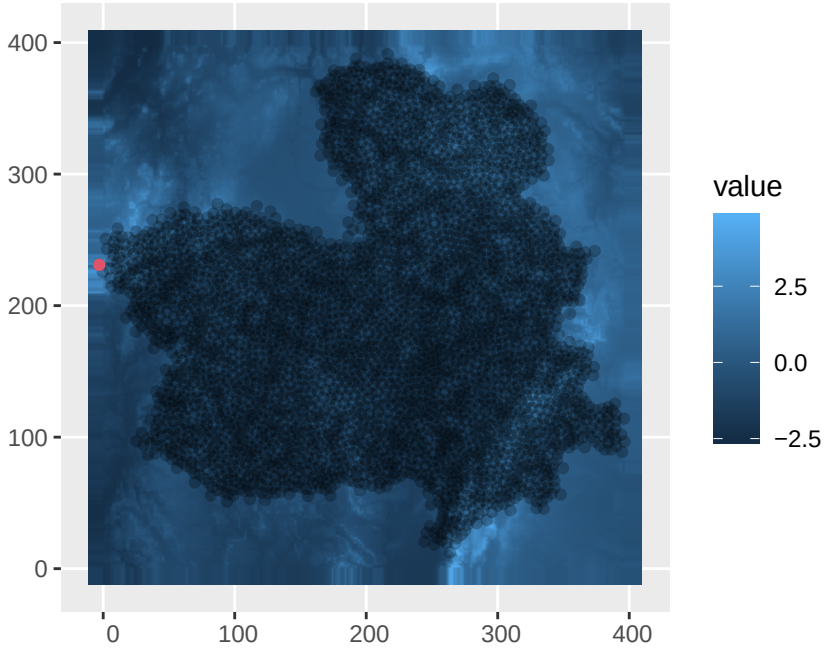


Figure 3: Integration scheme for numerical approximation of the stochastic integral in La Mancha Region

```
# Extend raster ext by 5 % of the original raster
re <- extend(elev_raster, ext(elev_raster)*1.05)
# 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()
```



Stage 2 Latent field model

We will model the fire locations as a point process whose intensity function $\lambda(s)$ is additive on the log-scale:

$$\eta(s) = \log\ \lambda(s) = \beta_0 + \beta_1\ \text{elevation}(s) + \omega(s),$$

Here, $\omega(s)$ is the Matérn Gaussian field capturing the spatial structure of all the locations where fires have occurred (these locations are assumed to be independent given the Gaussian field).

Stage 3 Hyperparameters

The hyperparameters of the model are ρ and σ corresponding to

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

NOTE In this case the linear predictor $\eta(s)$ consists of three components.

After the mesh and a SPDE representation of the spatial GF have been defined, the model components can be specified using the formula syntax (recall that this allows users to choose meaningful names for model components).

```
cmp_lgcp <-  geometry ~  Intercept(1)  +
  elev(elev_rast_p, model = "linear") +
  space(geometry, model = spde_model)
```

Recall that the labels `Intercept`, `elev` (elevation effect) and `space` are used to name the components of the model but they equally well could be something else.

Now, notice that we have called the `elev_rast_p` raster data within the `elevation` component. Recall that `inlabru`provides support for `sf` and `terra` data structures, allowing it to extract information from spatial data objects. This is particularly relevant for LGCP, as spatial covariates (e.g., the elevation raster) must be available across the whole study area.

```
formula = geometry ~ Intercept  + elev + space
```

Recall that in an `sf` object, the geo-referenced information of our points is stored in the `geometry` column, and hence we specify this as our response

0.3.2.4 Step 4. Defining th observational model

`inlabru` has support for latent Gaussian Cox processes through the `cp` likelihood family. We just need to supply the `sf` object as our data and the integration scheme `ips`:

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

> **Note**
>
> `inlabru` supports a shortcut for defining the integration points using the `domain` and `samplers` argument of `like()`. 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 = formula,
>               data = pp,
>               family = "cp",
>               domain = list(geometry = mesh),
>               samplers = region)
> ```

0.3.3 Step 5. Run the model

Finally, we can fit the model as usual

```
fit_lgcp = bru(cmp_lgcp,lik)
```

Posterior summaries of fixed effects and hyper parameters can be obtained using the `summary()` function.

```
summary(fit_lgcp)
```

0.3.4 Model predictions

Model predictions can be computed using the `predict` function by supplying the coordinates where the covariate is defined. We can do this by defining a `sf` object using coordi-

mean	sd	0.025quant	0.5quant	0.975quant	mode
-7.49	0.85	-9.25	-7.48	-5.83	-7.48
-0.07	0.17	-0.41	-0.07	0.26	-0.07
132.90	37.50	76.98	127.05	223.21	115.20
1.76	0.35	1.19	1.72	2.56	1.63

nates in our original raster data (we will crop the extension to that of *La Mancha* Region).

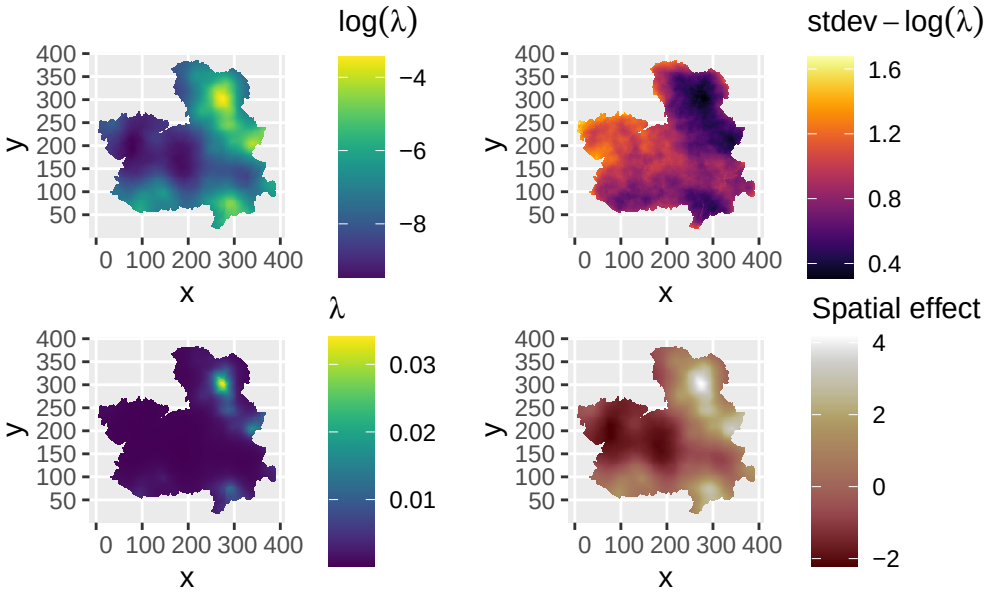
```
elev_crop <- terra::crop(x = elev_raster,y = region,mask=TRUE)


pxl1 = data.frame(crds(elev_crop),
                  as.data.frame(elev_crop$lyr.1)) %>%
       filter(!is.na(lyr.1)) %>%
st_as_sf(coords = c("x","y")) %>%
  dplyr::select(-lyr.1)
```

The formula object for the prediction can be a generic R expression that references model components using the user-defined names.

The `predict()` method returns an object in the same data format as was used in the predict call which, in this case, is an `sf` points object.

Support for plotting `sf` data objects is available in the `ggplot2` package.

1 Model predictions



2 R Code

```
lgcp_pred <- predict(
  fit_lgcp,
  pxl1,
  ~ data.frame(
    lambda = exp(Intercept + elev + space), # intensity
    loglambda = Intercept + elev +space,  #log-intensity
    GF = space # matern field
  )
)

# predicted log intensity
ggplot() + gg(lgcp_pred$loglambda, geom = "tile")
# standard deviation of the predicted log intensity
ggplot() + gg(lgcp_pred$loglambda, geom = "tile",aes(fill=sd))
# predicted intensity
ggplot() +  gg(lgcp_pred$lambda, geom = "tile")
# spatial field
ggplot() +  gg(lgcp_pred$GF, geom = "tile")
```