# Template for 2D data analysis with INLABru

J Matthiopoulos (collated from INLABru vignettes)

2022-05-27

## I. Preparation

### I.1. Load libraries

```
# Essential
library(inlabru)
library(INLA)
bru_options_set(inla.mode = "experimental")

# Visualisation
library(ggplot2)
library(RColorBrewer)

# Loaded dependencies
#library(sp)
#library(Matrix)
#library(foreach)
#library(parallel)

# Optional
library(mgcv) # For independent model performance comparisons, used as an exact method


#library(sf)
#library(raster)
#library(rmapshaper)
#library(tidyr)
```

### I.2. Load data

In the example below, it is assumed that the data reside in a package, such as 'inlabru'. The 'try' option explores the list of available datasets. The second line loads the particular one. Other ways of importing the data, assuming they are not in a package ('?data') with option 'lib.loc' for pathname.

```
#try(data(package="inlabru"))
data(gorillas, package = "inlabru")
```

### I.3. Ensure data formatting

The overall structure of the data can be explored by 'str()'. The point locations (here, 'nests') need to be a 'SpatialPointsDataFrame'. If the point data are not in this form then, they will need to be converted by providing an appropriate spatial projection ('?SpatialPointsDataFrame').

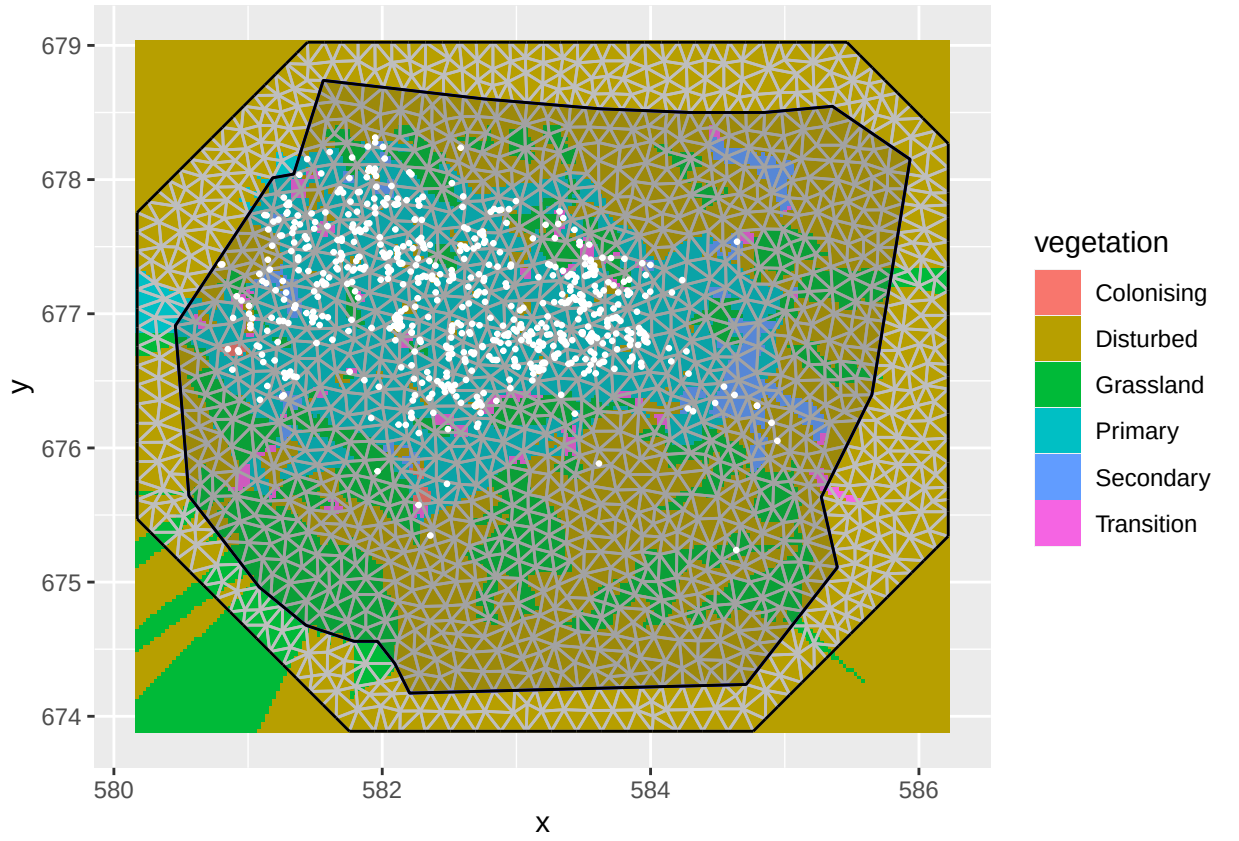
```
#str(gorillas)
#str(gorillas$nests)
myPoints<-gorillas$nests # assign to shorthand
myCovs <- gorillas$gcov # Covariate data
```

Here, the data set comes with in-built mesh and boundary components, but details on mesh specification are in Section III.1.

```
#str(gorillas$mesh)
#str(gorillas$boundary)
myMesh<-gorillas$mesh # assign to shorthand
myBoundary<-gorillas$boundary # assign to shorthand
```

Visualise the factor covariate data (`vegetation` type)

```
ggplot() +
  gg(myCovs$vegetation) +
  gg(myMesh) +
  gg(myBoundary) +
  gg(myPoints, color = "white", cex = 0.5) +
  coord_equal()
```




# II. GLMs

## II.1 A GLM with a factor covariate only

——> Preparation of a factor covariate layer on a mesh is likely to be challenging. Most data layers will come in raster form and while the projection to a mesh is easy for continuous variables (e.g., `pixel_n <- raster(spatialpixeldf)`), this is not straightforward with factors. Of course, in the data above this comes pre-made. As a result, attempting to run a model using this covariate data together with a custom mesh created below is not going to work.

To construct a model with vegetation type as a fixed effect, we need to tell 'lgcp' how to find the vegetation type at any point in space, and we do this by creating model components with a fixed effect that we call `vegetation` (we could call it anything), as follows:

```
myFactComp <- coordinates ~ vegetation(myCovs$vegetation, model = "factor_full") - 1
```

Notes:

- We need to tell 'lgcp' that this is a factor fixed effect, which we do with `model="factor_full"`, giving one coefficient for each factor level.
- We need to be careful about overparameterisation when using factors. Unlike regression models like `lm()`, `glm()` or `gam()`, `lgcp()`, `inlabru` does not automatically remove the first level and absorb it into an intercept. Instead, we can either use `model="factor_full"` without an intercept, or `model="factor_contrast"`, which does remove the first level.

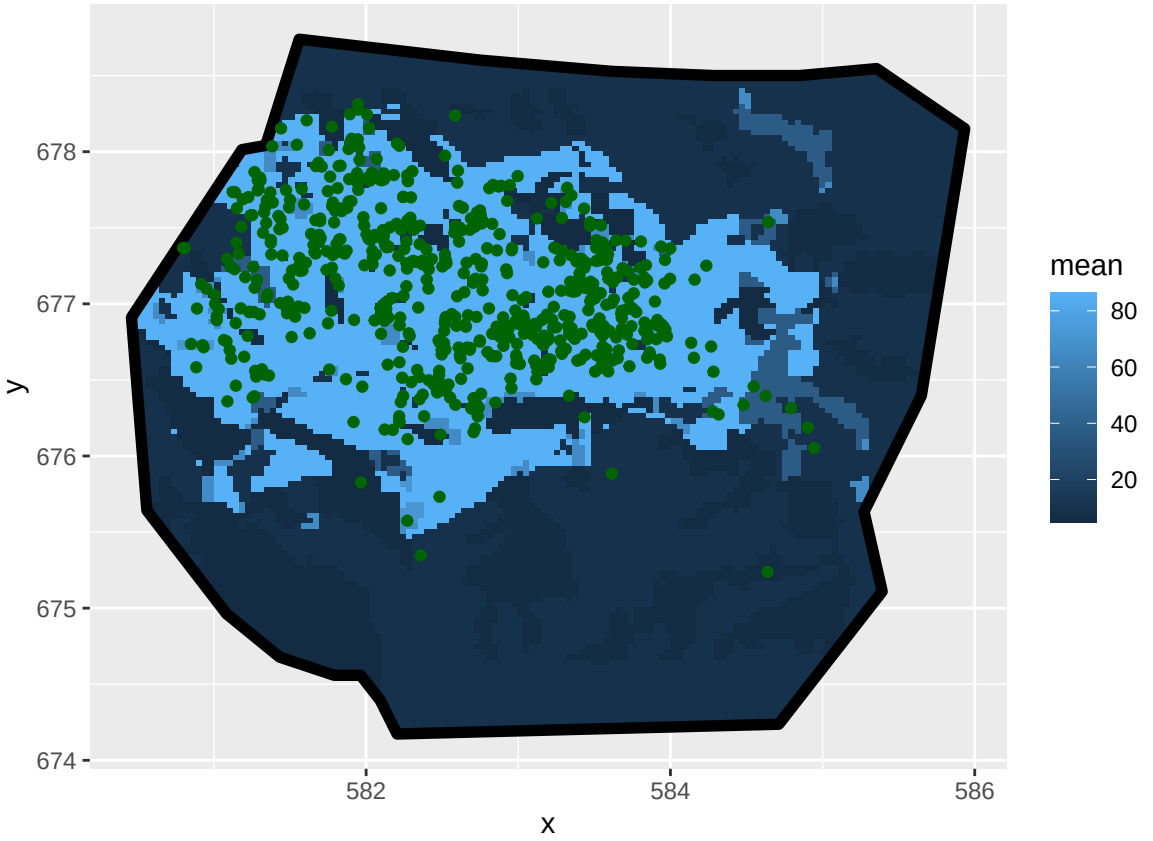
```
myFactCompAlt <- coordinates ~ vegetation(myCovs$vegetation, model = "factor_contrast") + Intercept(1)
```

The model can be fitted as follows:

```
myFactorGLM <- lgcp(myFactComp, myPoints, samplers = myBoundary, domain = list(coordinates = myMesh))
```

To predict the intensity, and plot the median intensity surface, the `predidct` function of `inlabru` takes into its `data` argument a `SpatialPointsDataFrame`, a `SpatialPixelsDataFrame` or a `data.frame`. We can use the `inlabru` function `pixels` to generate a `SpatialPixelsDataFrame` only within the boundary, using its `mask` argument, as shown below.

```
df <- pixels(myMesh, mask = myBoundary)
int1 <- predict(myFactorGLM, data = df, ~ exp(vegetation))
ggplot() +
  gg(int1) +
  gg(myBoundary, alpha = 0, lwd = 2) +
  gg(myPoints, color = "DarkGreen") +
  coord_equal()
```




The estimated total abundance of points (but not full posterior) can be obtained as follows.The integration `weight` values (the quadrature points) are contained in the `ipoints` output.

```
ips <- ipoints(myBoundary, myMesh)
Lambda1 <- predict(myFactorGLM, ips, ~ sum(weight * exp(vegetation)))
Lambda1
```

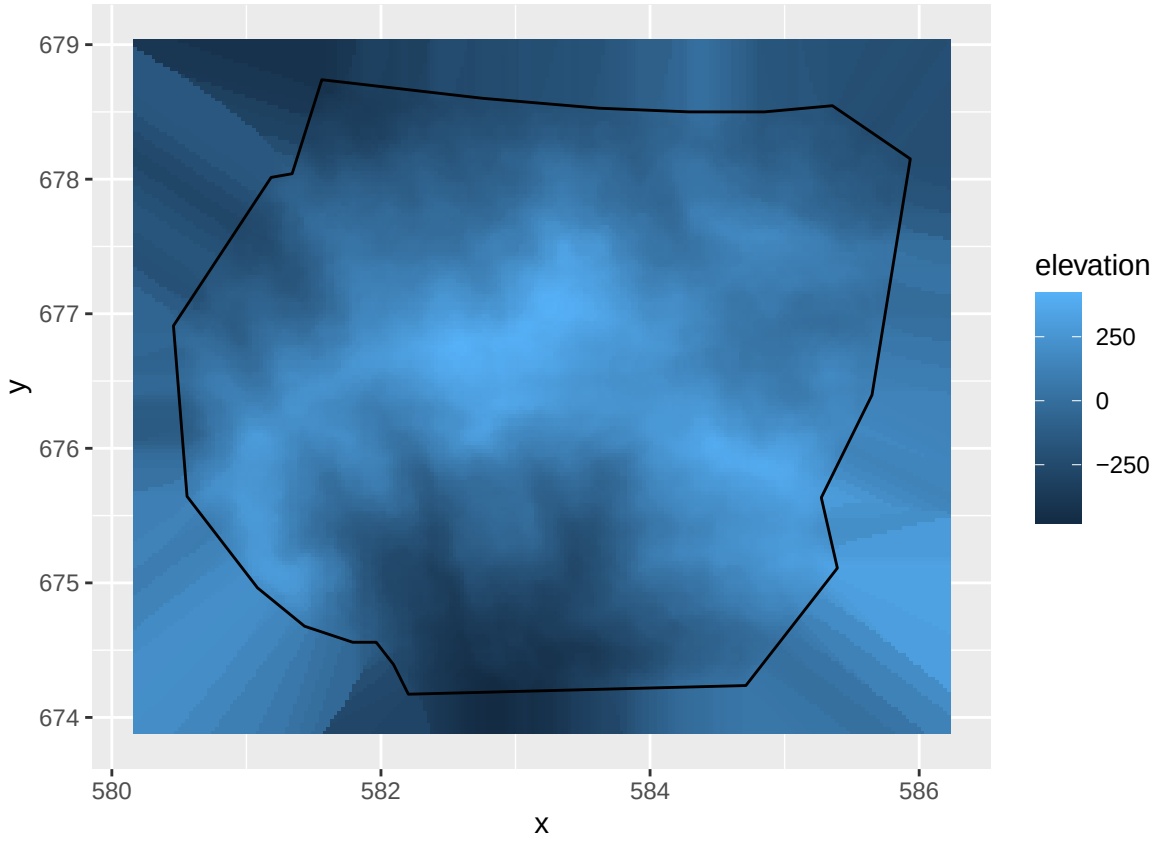
```
##       mean       sd  q0.025   median   q0.975     smin     smax        cv
## 1 643.5324 25.74303 600.702 642.1559 693.0005 592.7326 724.8745 0.0400027
##        var
## 1 662.7038
```

### II.1 A GLM with a continuous covariate only

Now lets try a model with elevation as a (continuous) explanatory variable. (First centre elevations for more stable fitting.)

```
elev <- myCovs$elevation
elev$elevation <- elev$elevation - mean(elev$elevation, na.rm = TRUE)

ggplot() +
  gg(elev) +
  gg(myBoundary, alpha = 0) +
  coord_fixed()
```




The elevation variable here is of class `SpatialGridDataFrame`, that can be handled in the same way as the vegetation covariate. However, since in some cases data may be stored differently, and other methods are needed to access the stored values. In such cases, we can define a function that knows how to evaluate the covariate at arbitrary points in the survey region, and call that function in the component definition. In this case, we can use a powerful method from the 'sp' package to do this. We use this to create the needed function.

```
f.elev <- function(x, y) {
  # turn coordinates into SpatialPoints object:
  # with the appropriate coordinate reference system (CRS)
  spp <- SpatialPoints(data.frame(x = x, y = y), proj4string = fm_sp_get_crs(elev))
  proj4string(spp) <- fm_sp_get_crs(elev)
  # Extract elevation values at spp coords, from our elev SpatialGridDataFrame
  v <- over(spp, elev)
  if (any(is.na(v$elevation))) {
    v$elevation <- inlabru:::bru_fill_missing(elev, spp, v$elevation)
  }
  return(v$elevation)
}
```

The model is fitted as follows:

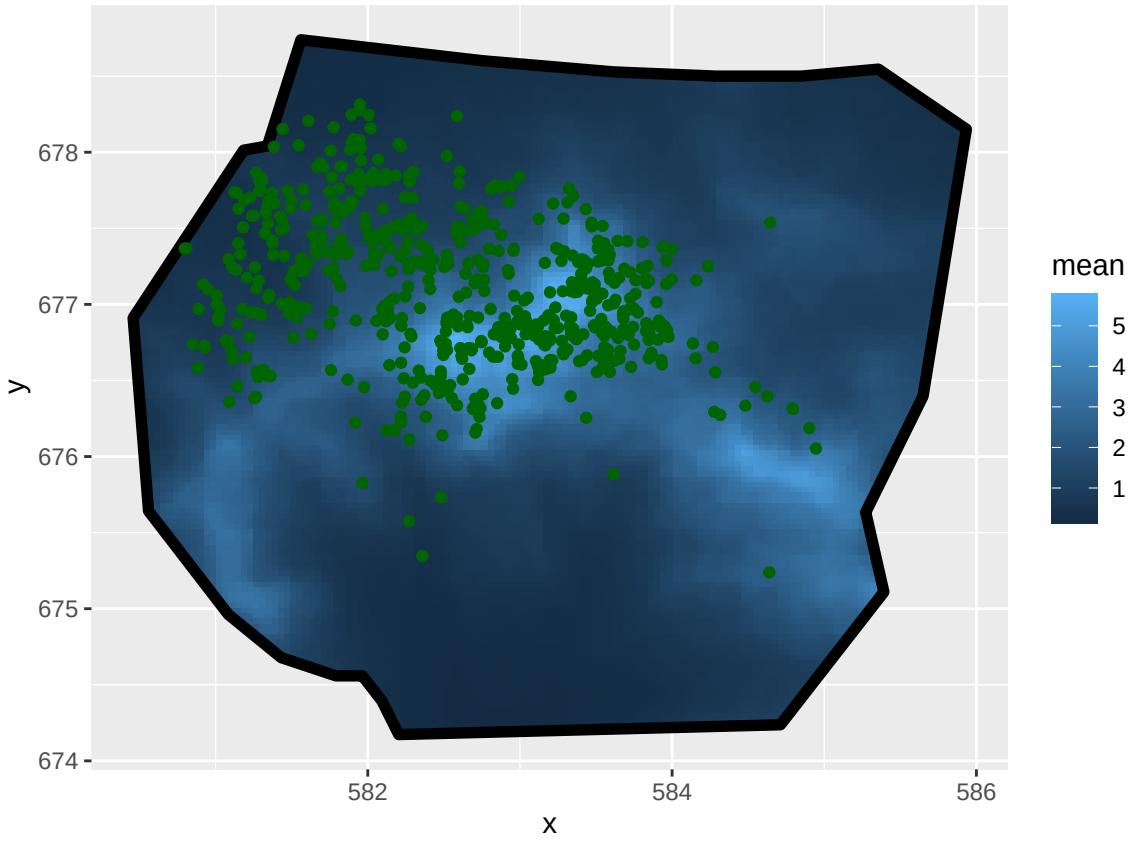
```
ecomp <- coordinates ~ elev(f.elev(x, y), model = "linear") + Intercept(1)
myVarGLM<- lgcp(ecomp, myPoints, samplers = myBoundary, domain = list(coordinates = myMesh))
summary(myVarGLM)
```

```
## inlabru version: 2.5.2
## INLA version: 22.05.07
## Components:
##   elev: Model types main='linear', group='exchangeable', replicate='iid'
##   Intercept: Model types main='linear', group='exchangeable', replicate='iid'
## Likelihoods:
##   Family: 'cp'
##     Data class: 'SpatialPointsDataFrame'
##     Predictor: coordinates ~ .
## Time used:
##     Pre = 0.328, Running = 0.123, Post = 0.0326, Total = 0.484
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant mode kld
## elev       0.004 0.000      0.004    0.004      0.005   NA   0
## Intercept 3.068 0.056      2.959    3.068      3.177   NA   0
##
## Deviance Information Criterion (DIC) ...............: -952.12
## Deviance Information Criterion (DIC, saturated) ....: -17729.16
## Effective number of parameters .....................: -2246.11
##
## Watanabe-Akaike information criterion (WAIC) ...: 1368.06
## Effective number of parameters .................: 1.94
##
## Marginal log-Likelihood:  -1788.51
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

Predictions are made in same way as with factor variable

```
df <- pixels(myMesh, mask = myBoundary)
int1 <- predict(myVarGLM, data = df, ~ exp(elev))
ggplot() +
```

```
  gg(int1) +
  gg(myBoundary, alpha = 0, lwd = 2) +
  gg(myPoints, color = "DarkGreen") +
  coord_equal()
```




## III. SPDE model, no covariates

### III.1. Build the SPDE mesh

For this section I will ignore the fact that the gorillas data set comes with a predefined mesh and develop one from scratch. There are several arguments that can be used to build the mesh. The arguments for a two-dimensional mesh construction are the following:

```
args(inla.mesh.2d)
```

```
## function (loc = NULL, loc.domain = NULL, offset = NULL, n = NULL,
##     boundary = NULL, interior = NULL, max.edge = NULL, min.angle = NULL,
##     cutoff = 1e-12, max.n.strict = NULL, max.n = NULL, plot.delay = NULL,
##     crs = NULL)
## NULL
```

First, some reference about the study region is needed, which can be provided by either:

1. The location of points, supplied on the `loc` argument [1].
2. A boundary of the region defined by a set of polygons (e.g., a polygon defining the coastline of the study) supplied on the `boundary` argument.
3. The domain extent which can be supplied as a single polygon on the `loc.domain` argument.

Note that if either (1) the location of points or (3) the domain extent are specified, the mesh will be constructed based on a convex hull (a polygon of triangles out of the domain area). Alternatively, it is possible to include a non-convex hull as a boundary in the mesh construction instead of the `loc` or `loc.domain` arguments. This will result in the triangulation to be constrained by the boundary. A non-convex hull mesh can also be created by building a boundary for the points using the `inla.nonconvex.hull()` function. Finally, the other compulsory argument that needs to be specified is `max.edge` which determines the largest allowed triangle length (the lower the value for max.edge the higher the resolution). The value supplied to this argument can be either a scalar, in which case the value controls the triangle edge lengths in the inner domain, or a length two vector that controls the edge lengths in the inner domain and in the outer extension respectively. Notice that The value (or values) passed to the `max.edge` option must be on the same scale unit as the coordinates.

While there is no general rule for setting a correct value for the `max.edge`, a value for `max.edge` that is too close to the spatial range will make the task of fitting a smooth SPDE difficult. On the other hand, if the `max.edge` value is too small compared to the spatial range, the mesh will have a large number of vertices leading to a more computationally demanding fitting process (which might not necessarily lead to better results). Thus, it is better to begin the analysis with a coarse matrix and evaluate the model on a finer grid as a final step. The `cutoff` option regulates the minimum length of each edge (could have been called "min.edge", more intuitively?).

I first develop the mesh by using the boundary of the study area. (The `ggplot2` function `coord_fixed()` sets the aspect ratio, which defaults to 1.)

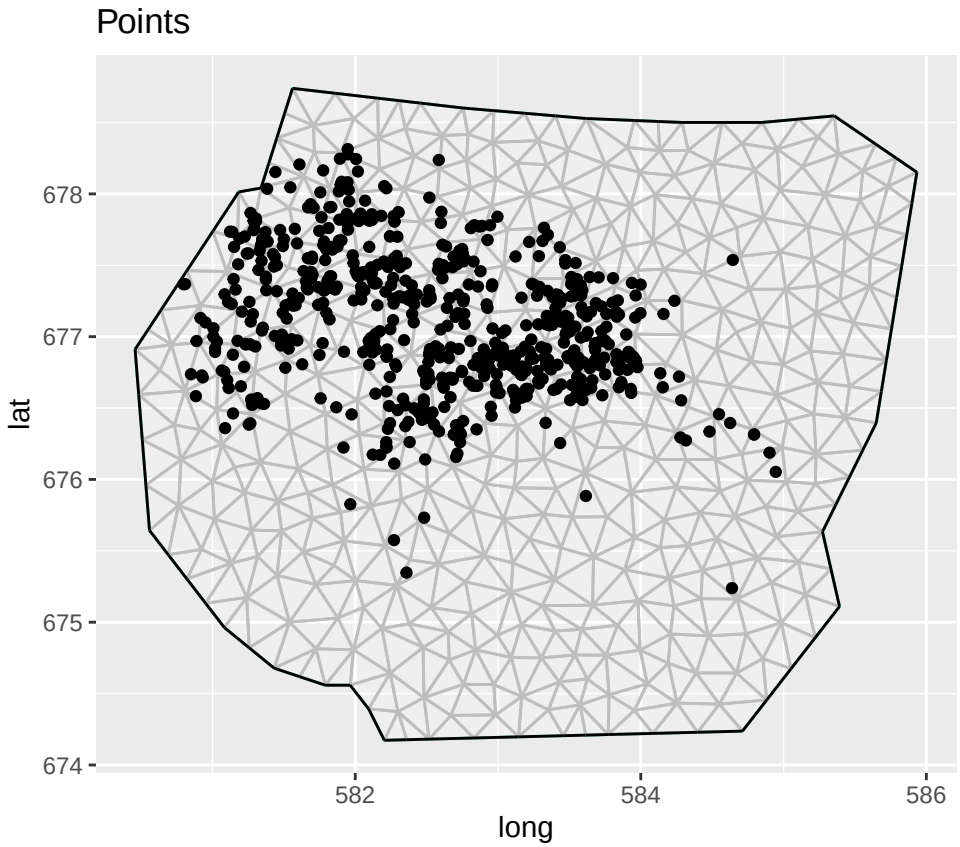
```
# Build the mesh
bbox<-bbox(myBoundary)
max.edge <- 1/20*sqrt((bbox[1,1]-bbox[1,2])^2+(bbox[2,1]-bbox[2,2])^2)

myMesh2 <- inla.mesh.2d(boundary = myBoundary,
                    max.edge = max.edge,
                    cutoff=.1)

#plot(mesh)

ggplot() +
     gg(data=myBoundary,color='turquoise',fill='transparent')+
  gg(myMesh2)+
  gg(myPoints) +
  coord_fixed() +
  ggtitle("Points")
```

[1] Matrix of point locations to be used as initial triangulation nodes. Can alternatively be a `SpatialPoints` or `SpatialPointsDataFrame` object.




We can also specify an outer layer with a lower triangle density (i.e. where no points occur) to avoid this boundary effect. This can be done by supplying a vector of two values so that the spatial domain is divided into an inner and an outer area. Here, we will define the max.edge such that the outer layer will have a triangle density two times lower than than the inner layer (i.e. twice the length for the outer layer edges).The amount to which the domain should be extended in the inner and outer part can be controlled with the offset argument of the inla.mesh.2d function. For this example we will expand the inner layer by the same amount as the max.edge and the outer layer by the range we assumed when defining the inner max.edge value (i.e. 1/4 of the spatial extent).

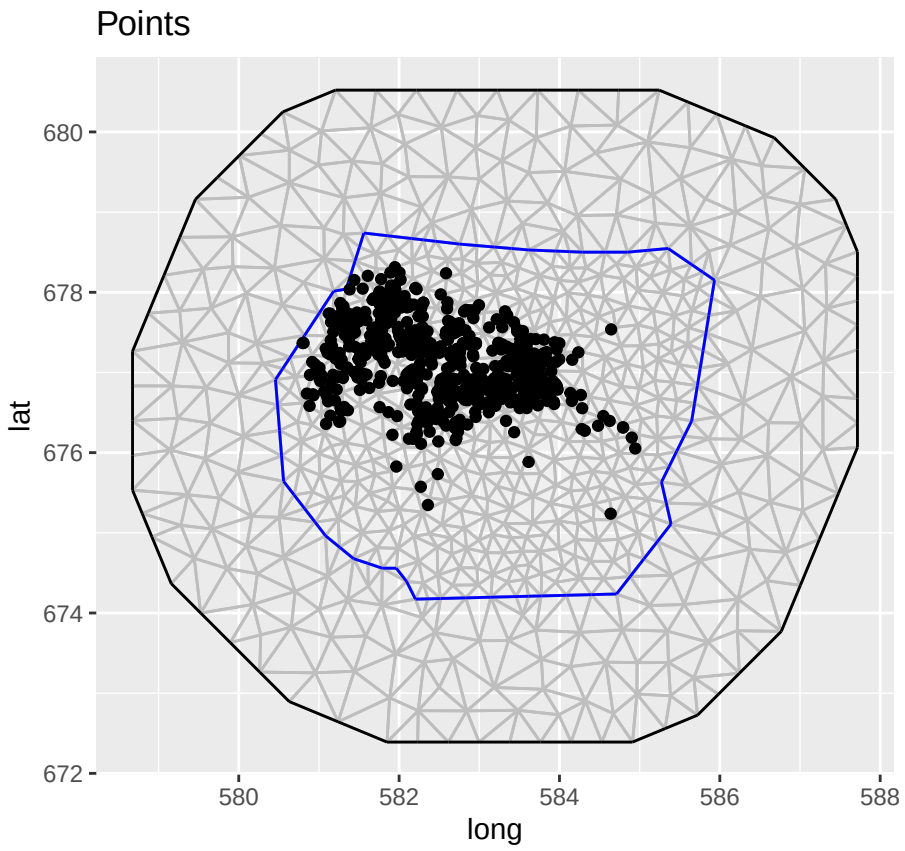
```
# Build the mesh
bbox<-bbox(myBoundary)
max.edge <- 1/20*sqrt((bbox[1,1]-bbox[1,2])^2+(bbox[2,1]-bbox[2,2])^2)
bound.outer <- 1/4*sqrt((bbox[1,1]-bbox[1,2])^2+(bbox[2,1]-bbox[2,2])^2)

myMesh3 <- inla.mesh.2d(boundary = myBoundary,
                    max.edge = c(1,2)*max.edge,
                    cutoff=.1,
                    offset=c(max.edge, bound.outer))

#plot(mesh)

ggplot() +
     gg(data=myBoundary,color='turquoise',fill='transparent')+
  gg(myMesh3)+
  gg(myPoints) +
  coord_fixed() +
```

```
  ggtitle("Points")
```

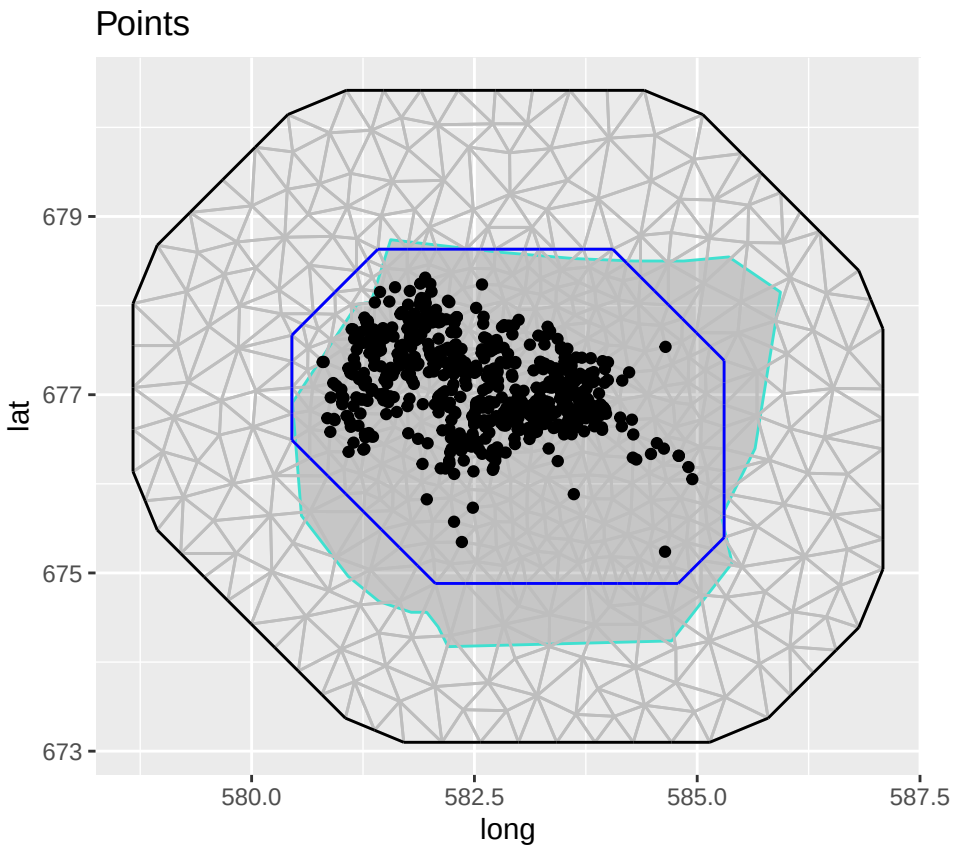



Alternatively, the mesh can be defined by the observation points.

```
# Build the mesh
bbox<-bbox(myBoundary)
max.edge <- 1/20*sqrt((bbox[1,1]-bbox[1,2])^2+(bbox[2,1]-bbox[2,2])^2)
bound.outer <- 1/4*sqrt((bbox[1,1]-bbox[1,2])^2+(bbox[2,1]-bbox[2,2])^2)

myMesh4 <- inla.mesh.2d(loc = myPoints,
                    max.edge = c(1,2)*max.edge,
                    cutoff=.1,
                    offset=c(max.edge, bound.outer))
#plot(mesh)

ggplot() +
     gg(data=myBoundary,color='turquoise')+
  gg(myMesh4)+
  gg(myPoints) +
  coord_fixed() +
  ggtitle("Points")
```




### III.2. The model

First, specify spatial correlation structure. The following is an example using a Matern correlation structure with a PC prior.

```
myCorrelation<-inla.spde2.pcmatern(myMesh, prior.range = c(5, 0.01), prior.sigma = c(0.1, 0.01))
```

Then, define the model. The model formula requires the explicit name 'coordinates' to recognise the mesh information that it will receive later, but can use the user-defined 'mySmooth()' to specify the spatial error term.

```
mySpdeComp<-coordinates~mySmooth(coordinates, model=myCorrelation) + Intercept(1)
```

The `lcgp` models is fitted as follows:

```
mySpdeFit<-lgcp(mySpdeComp, data=myPoints, samplers=myBoundary, domain=list(coordinates=myMesh))
```

Summary statistics:

```
summary(mySpdeFit)
```

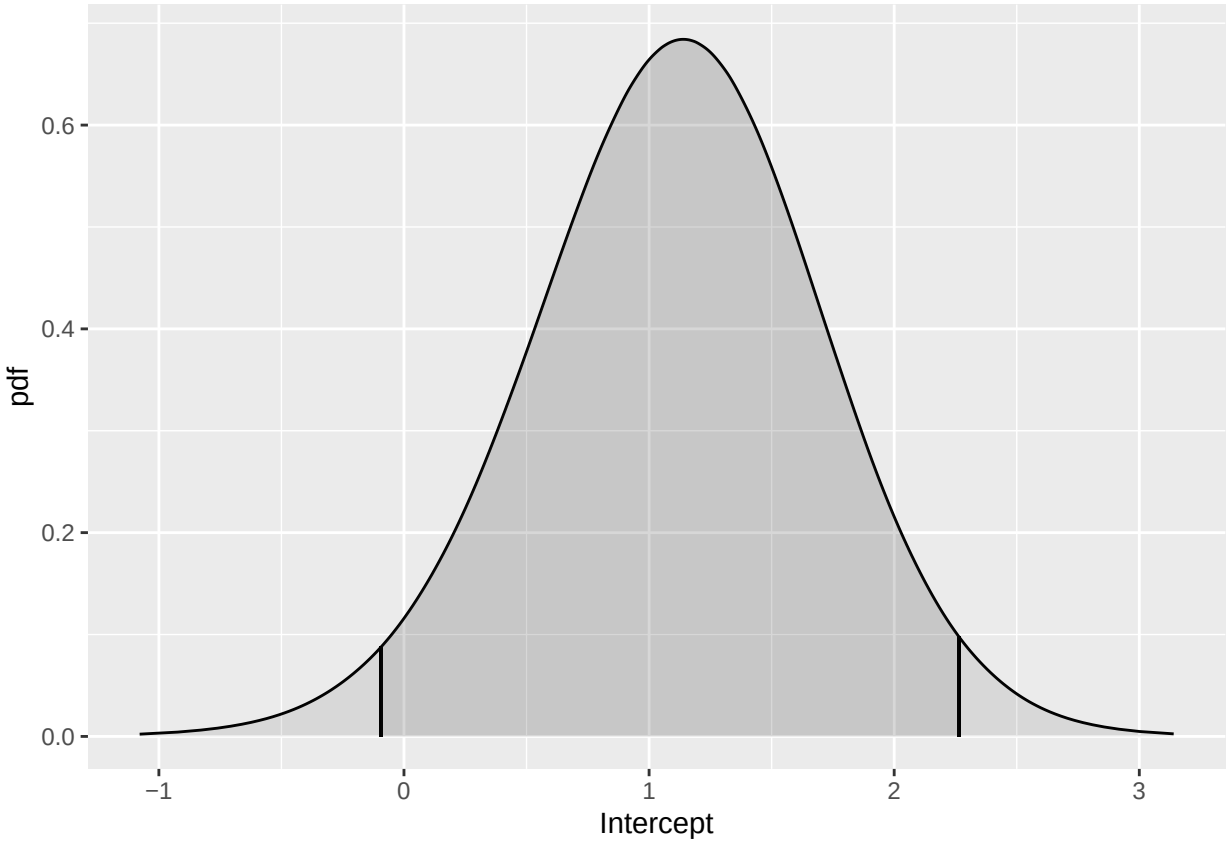
```
## inlabru version: 2.5.2
## INLA version: 22.05.07
```

```
## Components:
##   mySmooth: Model types main='spde', group='exchangeable', replicate='iid'
##   Intercept: Model types main='linear', group='exchangeable', replicate='iid'
## Likelihoods:
##   Family: 'cp'
##     Data class: 'SpatialPointsDataFrame'
##     Predictor: coordinates ~ .
## Time used:
##     Pre = 0.788, Running = 8.94, Post = 0.583, Total = 10.3
## Fixed effects:
##            mean  sd 0.025quant 0.5quant 0.975quant mode kld
## Intercept 1.111 0.6     -0.099    1.121       2.27   NA   0
##
## Random effects:
##   Name     Model
##     mySmooth SPDE2 model
##
## Model hyperparameters:
##                    mean    sd 0.025quant 0.5quant 0.975quant mode
## Range for mySmooth 2.12 0.242      1.688     2.10       2.64   NA
## Stdev for mySmooth 1.10 0.095      0.931     1.10       1.30   NA
##
## Deviance Information Criterion (DIC) ...............: 509.41
## Deviance Information Criterion (DIC, saturated) ....: -16267.64
## Effective number of parameters .....................: -837.43
##
## Watanabe-Akaike information criterion (WAIC) ...: 1584.57
## Effective number of parameters .................: 138.94
##
## Marginal log-Likelihood:  -1259.87
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

### III.3 Inference

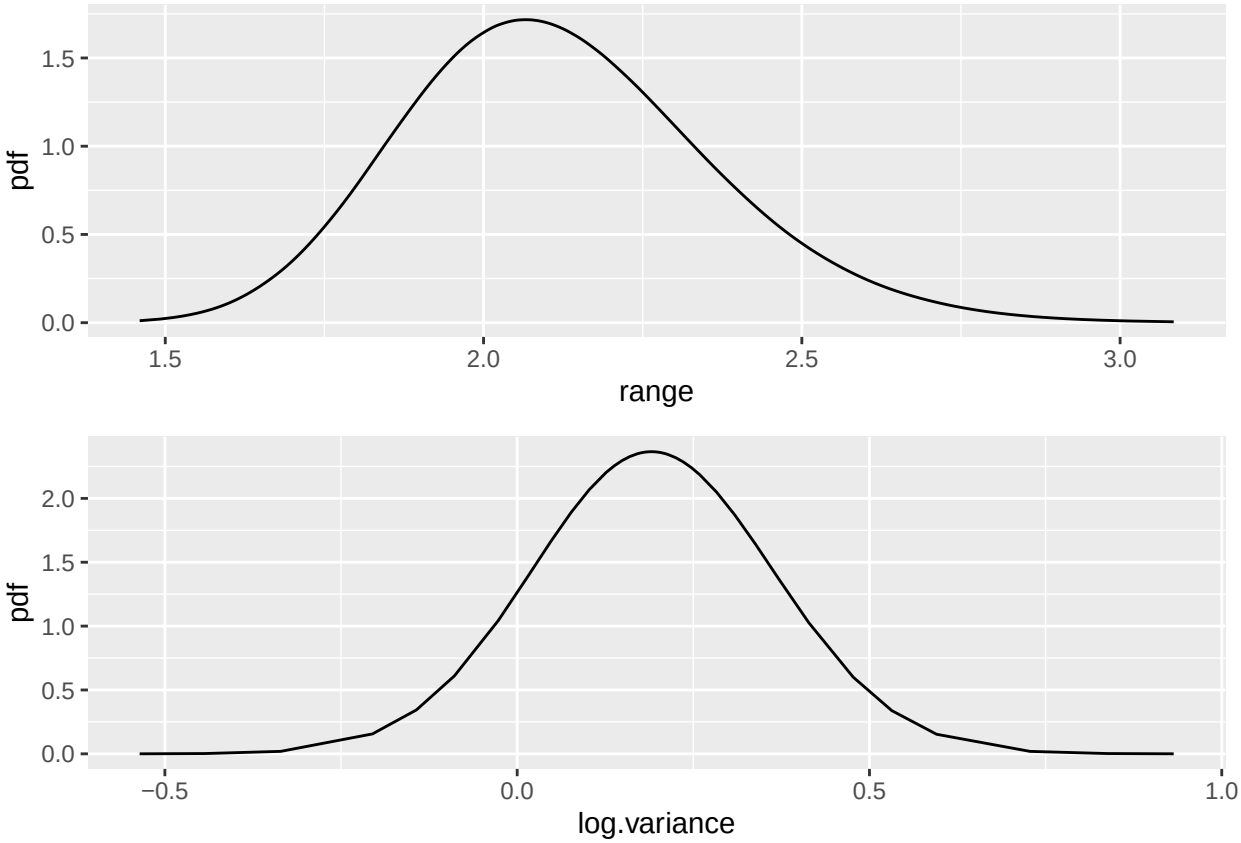
Plotting fixed effect parameters

```
plot(mySpdeFit, "Intercept")
```



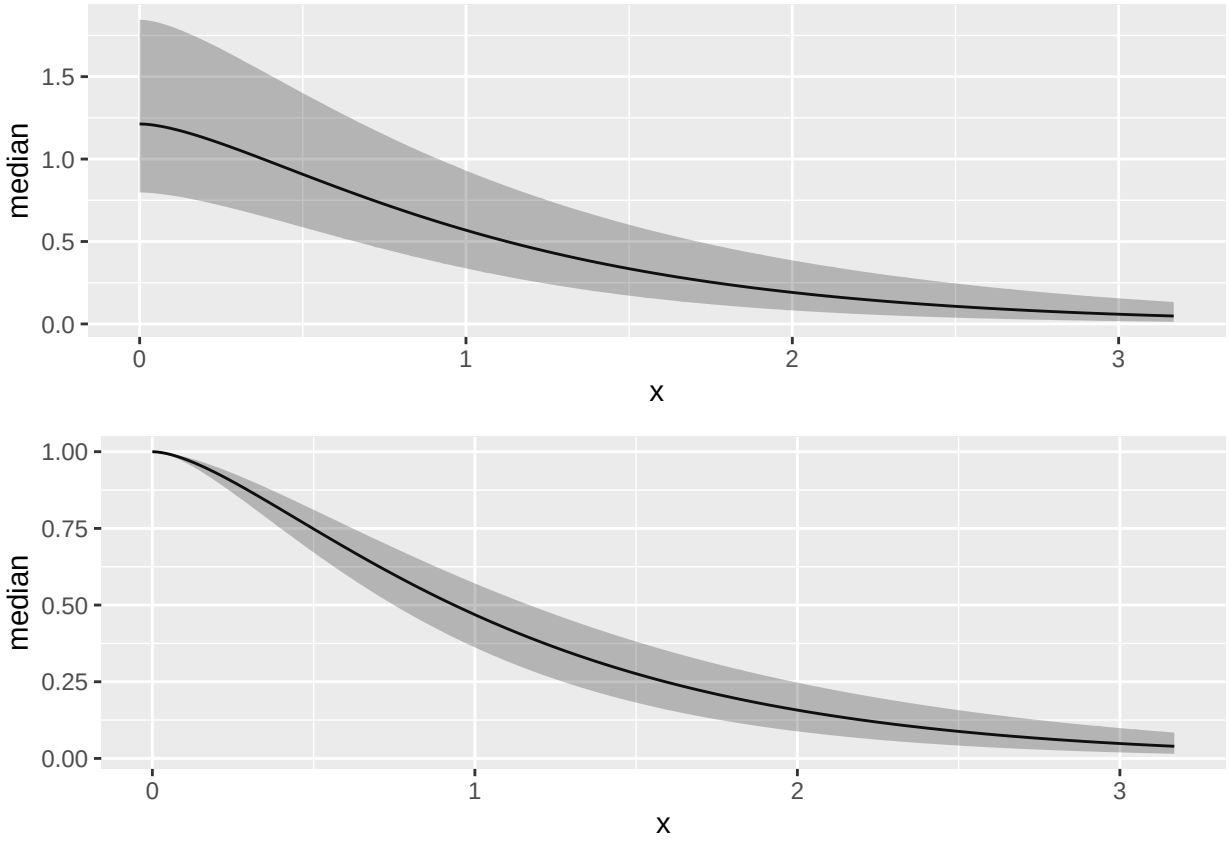

Plotting spatial random effects Plots of the individual parameters

```
spde.range <- spde.posterior(mySpdeFit, "mySmooth", what = "range")
spde.logvar <- spde.posterior(mySpdeFit, "mySmooth", what = "log.variance")
range.plot <- plot(spde.range)
var.plot <- plot(spde.logvar)
multiplot(range.plot, var.plot)
```




Plots of the correlation and covariance functions

```
corplot <- plot(spde.posterior(mySpdeFit, "mySmooth", what = "matern.correlation"))
covplot <- plot(spde.posterior(mySpdeFit, "mySmooth", what = "matern.covariance"))
multiplot(covplot, corplot)
```




### III.4 Model predictions

First need to generate the prediction data frame. The 'pixels()' command generates a regular grid of points which can be used for the prediction. This is stored as a spatial data frame in the user-defined 'myPredFrame'.

```
myPredFrame<-pixels(myMesh, nx = 50, ny = 50, mask = FALSE)
```

To constrain the predictions to a particular region (e.g. the boundary of the mesh), set the mask option in the 'pixels()' command to 'mask=myBoundary'.

Now we can generate the predictions.

```
myPreds<-predict(mySpdeFit, myPredFrame,~ exp(mySmooth + Intercept))
```

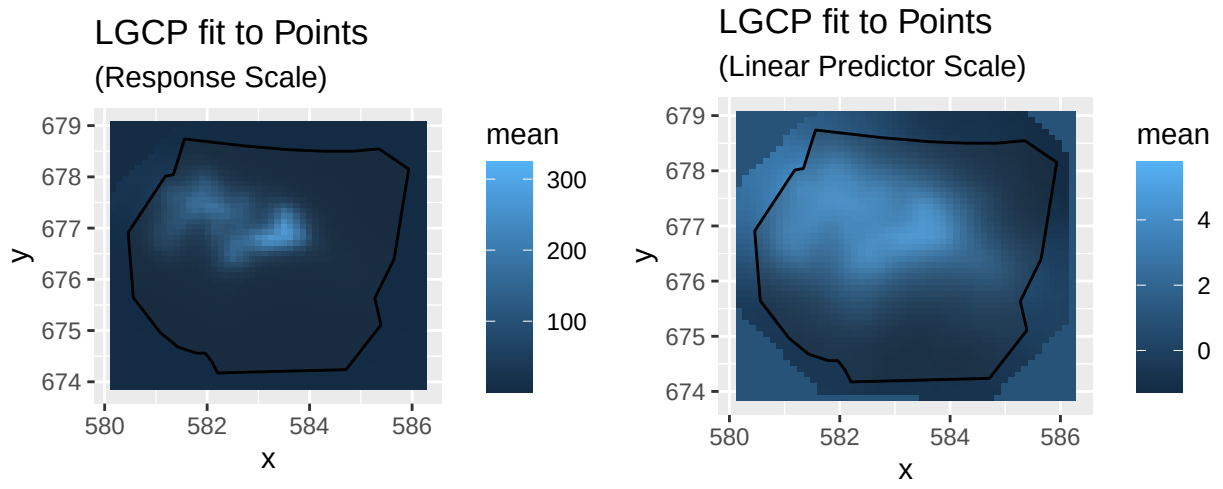
Note that multiple functions and linear predictors can be predicted simultaneously, under different names.

```
myPreds<-predict(mySpdeFit, myPredFrame,
                 ~ data.frame(myLambda = exp(mySmooth + Intercept),
                              myLoglambda = mySmooth + Intercept)
                )
```

We can visualize multiple aspects of the predictions

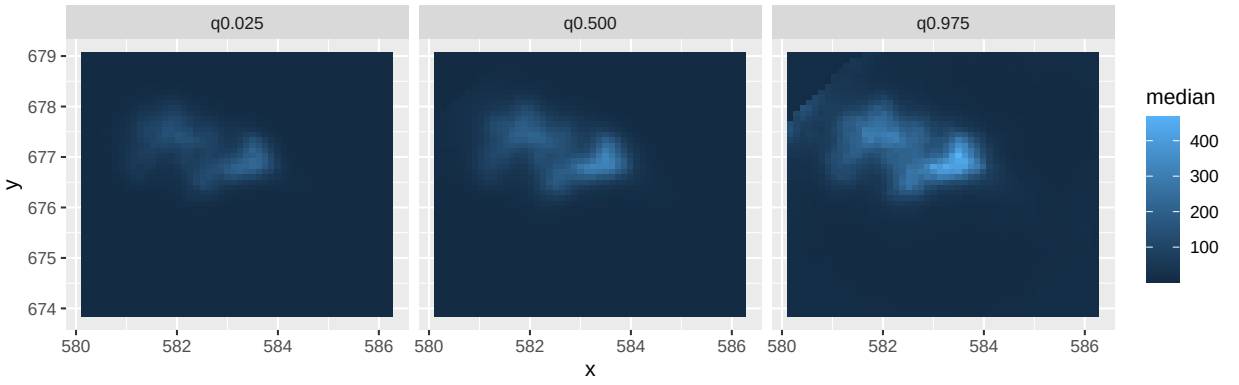
Plotting intensity and log-intensity surfaces

```
pl1 <- ggplot() +
  gg(myPreds$myLambda) +
  gg(myBoundary) +
  ggtitle("LGCP fit to Points", subtitle = "(Response Scale)") +
  coord_fixed()
pl2 <- ggplot() +
  gg(myPreds$myLoglambda) +
  gg(myBoundary) +
  ggtitle("LGCP fit to Points", subtitle = "(Linear Predictor Scale)") +
  coord_fixed()
multiplot(pl1, pl2, cols = 2)
```




Alternatively, plotting maps of median, lower 95% and upper 95% density surfaces as follows (assuming that the predicted intensity is in object `myLambda`).

```
ggplot() +
  gg(cbind(myPreds$myLambda, data.frame(property = "q0.500")), aes(fill = median)) +
  gg(cbind(myPreds$myLambda, data.frame(property = "q0.025")), aes(fill = q0.025)) +
  gg(cbind(myPreds$myLambda, data.frame(property = "q0.975")), aes(fill = q0.975)) +
  coord_equal() +
  facet_wrap(~property)
```




### III.5 Estimating abundance

Estimating abundance uses the `predict` function. As a first step we need an estimate for the integrated lambda (denoted 'Lambda' with an upper case L). The integration `weight` values (the quadrature points) are contained in the `ipoints` output.

```
Lambda <- predict(
  mySpdeFit,
  ipoints(myBoundary, myMesh),
  ~ sum(weight * exp(mySmooth + Intercept))
)
Lambda
```

```
##       mean       sd   q0.025   median   q0.975     smin     smax         cv
## 1 670.8013 27.49344 613.7847 671.8076 709.4127 602.2589 761.5181 0.04098597
##        var
## 1 755.8893
```

Use the median and 95%iles of this to determine interval boundaries for estimating the posterior abundance distribution (prediction, not credible interval).

```
abundance <- predict(
  mySpdeFit, ipoints(myBoundary, myMesh),
  ~ data.frame(
    N = 500:800,
    dpois(500:800,
      lambda = sum(weight * exp(mySmooth + Intercept))
    )
  )
)
```

Can get the quantiles of the posterior for abundance via

```
inla.qmarginal(c(0.025, 0.5, 0.975), marginal = list(x = abundance$N, y = abundance$mean))
```

```
## [1] 602.2388 672.7339 748.8601
```

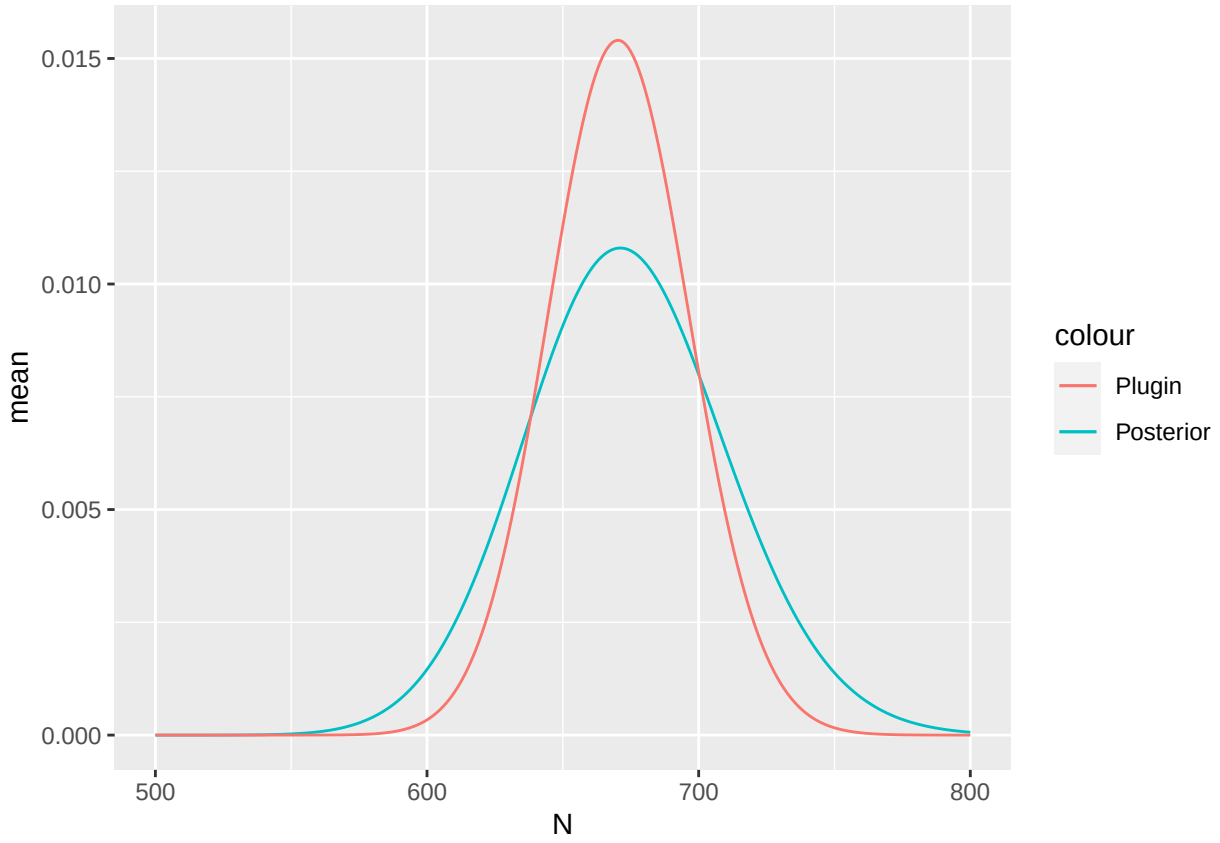
... the mean via

```
inla.emarginal(identity, marginal = list(x = abundance$N, y = abundance$mean))
```

```
## [1] 673.5515
```

and plot posteriors:

```
abundance$plugin_estimate <- dpois(abundance$N, lambda = Lambda$mean)
ggplot(data = abundance) +
  geom_line(aes(x = N, y = mean, colour = "Posterior")) +
  geom_line(aes(x = N, y = plugin_estimate, colour = "Plugin"))
```




## IV. Factor GLM with SPDE spatial term

This will borrow the same (Matern) correlation structure as defined in part III above `myCorrelation`. The combined model is defined as follows. Note the removal of the intercept under the factor model 'factor_full'.

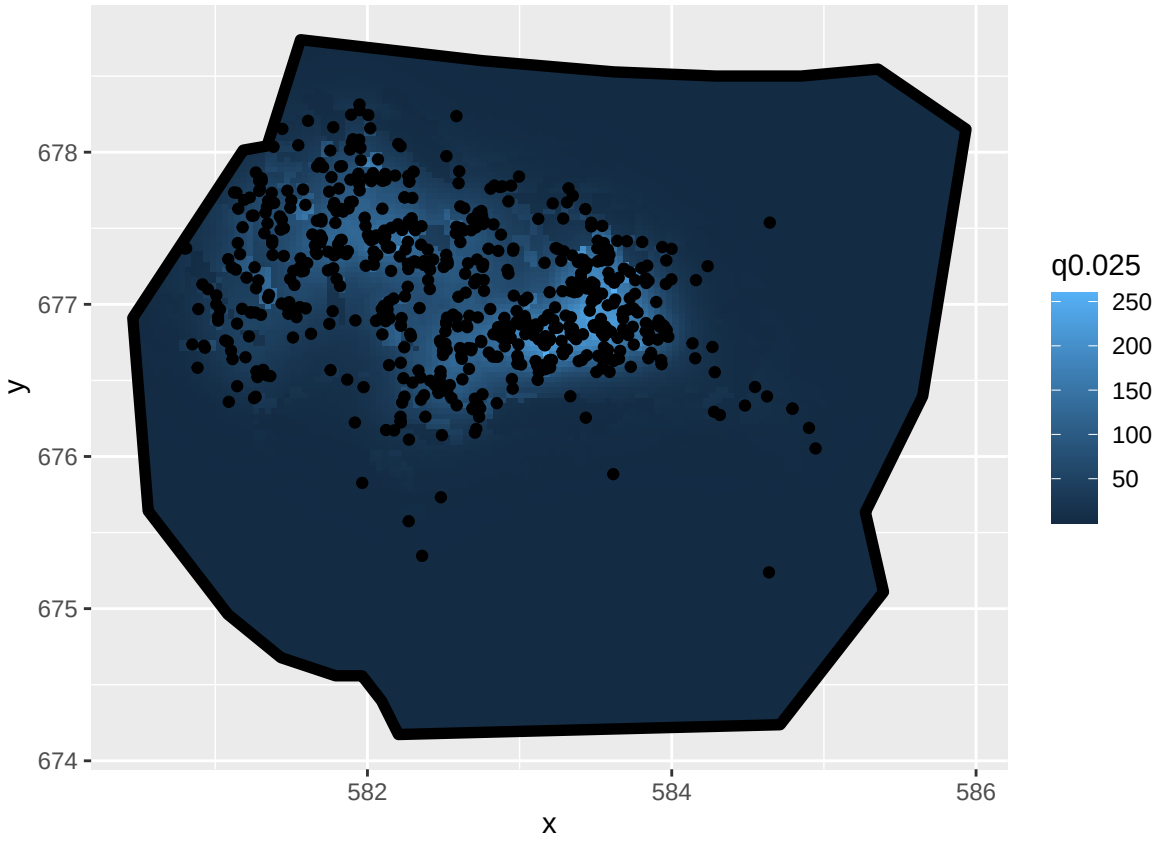
```
mySpdeGlmComp <- coordinates ~
-1 +vegetation(myCovs$vegetation, model = "factor_full") +
  mySmooth(coordinates, model = myCorrelation)
```

Model is fitted:

```
fitSpdeGlm <- lgcp(mySpdeGlmComp, myPoints, samplers = myBoundary, domain = list(coordinates = myMesh))
```

Spatial plot of the fitted (median) intensity surface:

```
df <- pixels(myMesh, mask = myBoundary)
int2 <- predict(fitSpdeGlm , df, ~ exp(mySmooth + vegetation), n.samples = 1000)
ggplot() +
  gg(int2, aes(fill = q0.025)) +
  gg(myBoundary, alpha = 0, lwd = 2) +
  gg(myPoints) +
  coord_equal()
```




. . . and the expected integrated intensity (mean of abundance)

```
Lambda2 <- predict(
  fitSpdeGlm,
  ipoints(myBoundary, myMesh),
  ~ sum(weight * exp(mySmooth + vegetation))
)
Lambda2
```

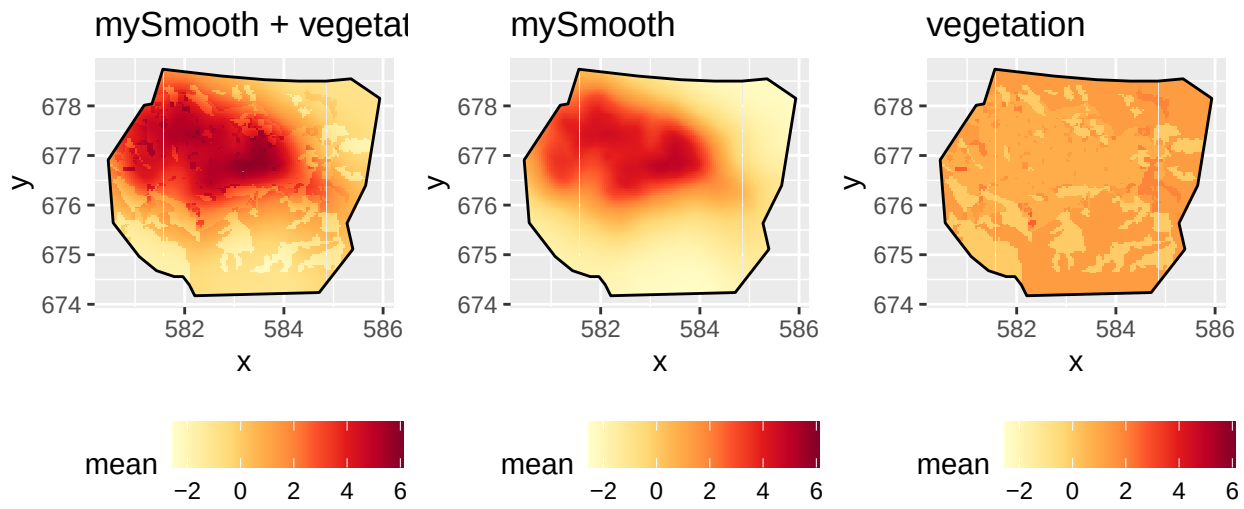
```
##       mean       sd   q0.025   median   q0.975     smin     smax        cv
## 1 678.6762 26.48365 632.2878 674.9475 739.3736 629.2411 743.7722 0.0390225
##        var
## 1 701.3835
```

To look at the contributions to the linear predictor from the SPDE and from vegetation, we can first generate predictions from those combined, and individually.

```
lp2 <- predict(fitSpdeGlm, df, ~ list(
  smooth_veg = mySmooth + vegetation,
  smooth = mySmooth,
  veg = vegetation
))
```

The function `scale_fill_gradientn` sets the scale for the plot legend. Here, we set it to span the range of the three linear predictor components being plotted (medians are plotted by default).

```
lprange <- range(lp2$smooth_veg$median, lp2$smooth$median, lp2$veg$median)
csc <- scale_fill_gradientn(colours = brewer.pal(9, "YlOrRd"), limits = lprange)
plot.lp2 <- ggplot() +
  gg(lp2$smooth_veg) +
  csc +
  theme(legend.position = "bottom") +
  gg(myBoundary, alpha = 0) +
  ggtitle("mySmooth + vegetation") +
  coord_equal()
plot.lp2.spde <- ggplot() +
  gg(lp2$smooth) +
  csc +
  theme(legend.position = "bottom") +
  gg(myBoundary, alpha = 0) +
  ggtitle("mySmooth") +
  coord_equal()
plot.lp2.veg <- ggplot() +
  gg(lp2$veg) +
  csc +
  theme(legend.position = "bottom") +
  gg(myBoundary, alpha = 0) +
  ggtitle("vegetation") +
  coord_equal()
multiplot(plot.lp2, plot.lp2.spde, plot.lp2.veg, cols = 3)
```




**Model comparisons**

NOTE: the behaviour of DIC and WAIC is currently a bit unclear, in particular for experimental mode, and is being investigated./1

```
knitr::kable(cbind(
  deltaIC(myFactorGLM , mySpdeFit, fitSpdeGlm, criterion = c("DIC")),
  deltaIC(myFactorGLM , mySpdeFit, fitSpdeGlm, criterion = c("WAIC"))))
```

| Model | DIC | Delta.DIC | Model | WAIC | Delta.WAIC |
|---|---|---|---|---|---|
| myFactorGLM | -563.3583 | 0.000 | myFactorGLM | 1373.241 | 0.0000 |
| mySpdeFit | 509.4094 | 1072.768 | mySpdeFit | 1584.569 | 211.3279 |
| fitSpdeGlm | 597.6010 | 1160.959 | fitSpdeGlm | 1636.821 | 263.5800 |