# Session 4.2: `inla.stack` and `inlabru`

Spatial and Spatio-Temporal Bayesian Models with `R-INLA`, University of São Paulo

29 September 2022

# Learning Objectives

At the end of this session you should be able to:

- use `R-INLA` to implement a spatial geostatistical model with the `inla.stack` approach;
- use `inlabru` to implement a spatial geostatistical model;
- perform spatial prediction and mapping.

The topics treated in this lecture can be found in **Section 6.7 -- 6.9** of the INLA book.

# Outline

1. Model fitting and spatial prediction with the `inla.stack` approach

2. Model fitting and spatial prediction with the `inlabru` package

# Model fitting and spatial prediction with the `inla.stack` approach

# The `inla.stack()` function

- A function named `inla.stack()` has been introduced in `R-INLA` for an optimal and easy management of the SPDE objects (data, covariates, indices and projector matrices) and for the construction of the linear predictor (Lindgren and Rue, 2015).

- In the `SPDEtoy` example the **linear predictor** is given by

$$\eta_i = b_0 + \xi_i = b_0 + \sum_{g=1}^{G} A_{ig}\tilde{\xi}_g$$

  and can be written as

$$\boldsymbol{\eta} = \mathbf{1} b_0 + \boldsymbol{A}\tilde{\boldsymbol{\xi}}$$

  where the first term refers to the intercept and the second to the spatial effect. Note that each term in the linear predictor is represented as the product of a projector matrix and an effect.

- The main arguments of the `inla.stack()` function are:
  – `data`: a vector list with the data
  – `A`: a list of projector matrices
  – `effects`: the list of effects
  – `tag` (optional): a label for the data stack

# The inla.stack() function for model fitting with the SPDEtoy data

1. Define the `inla.stack` object for model fitting:

```
> stack.est <- inla.stack(
+   data = list(y = SPDEtoy$y),
+   A = list(1, A.est6),
+   effects = list(intercept = rep(1,nrow(SPDEtoy)),
+                  spatial.field = 1:spde$n.spde),
+   tag="est")
```

Note that the function `inla.stack()` will take care of eliminating any column in the projector matrix which is full of zeros.

2. Define the `formula` by specifying an explicit intercept:

```
> formula = y ~ -1 + intercept + f(spatial.field, model = spde)
```

3. Run `inla`! The function `inla.stack.data()` and `inla.stack.A()` are used for extracting the data and the projector matrix from the `stack.est` object:

```
> output6 <- inla(formula,
+                 data = inla.stack.data(stack.est),
+                 control.predictor = list(A = inla.stack.A(stack.est), compute = TRUE))
```

# Exploring the output

The output is exactly equal to the one presented in Section 2.1. The same posterior summary statistics can be computed also when the `inla.stack` approach is adopted.

```
> output6$summary.fixed[,c("mean","0.025quant","0.975quant")]
```

```
              mean 0.025quant 0.975quant
intercept 9.505379   8.041105   10.86208
```

```
> output6$summary.hyperpar[,c("mean","0.025quant","0.975quant")]
```

```
                                             mean 0.025quant 0.975quant
Precision for the Gaussian observations  2.860432   2.045075   3.867535
Theta1 for spatial.field                -4.029614  -4.323940  -3.716477
Theta2 for spatial.field                 2.087758   1.605681   2.534946
```

# Spatial prediction with the `inla.stack()` approach

- In geostatistics we are interested in predicting the (latent) spatial field (i.e. the linear predictor) at new spatial locations where we do not have data.

- It is possible to perform the spatial prediction jointly with the estimation by using the `inla.stack` approach.

- Consider the response variable distribution

$$\boldsymbol{y} \sim \text{Normal}(\boldsymbol{\eta} = \mathbf{1}b_0 + \boldsymbol{A}\tilde{\boldsymbol{\xi}}, \sigma_e^2\boldsymbol{I})$$

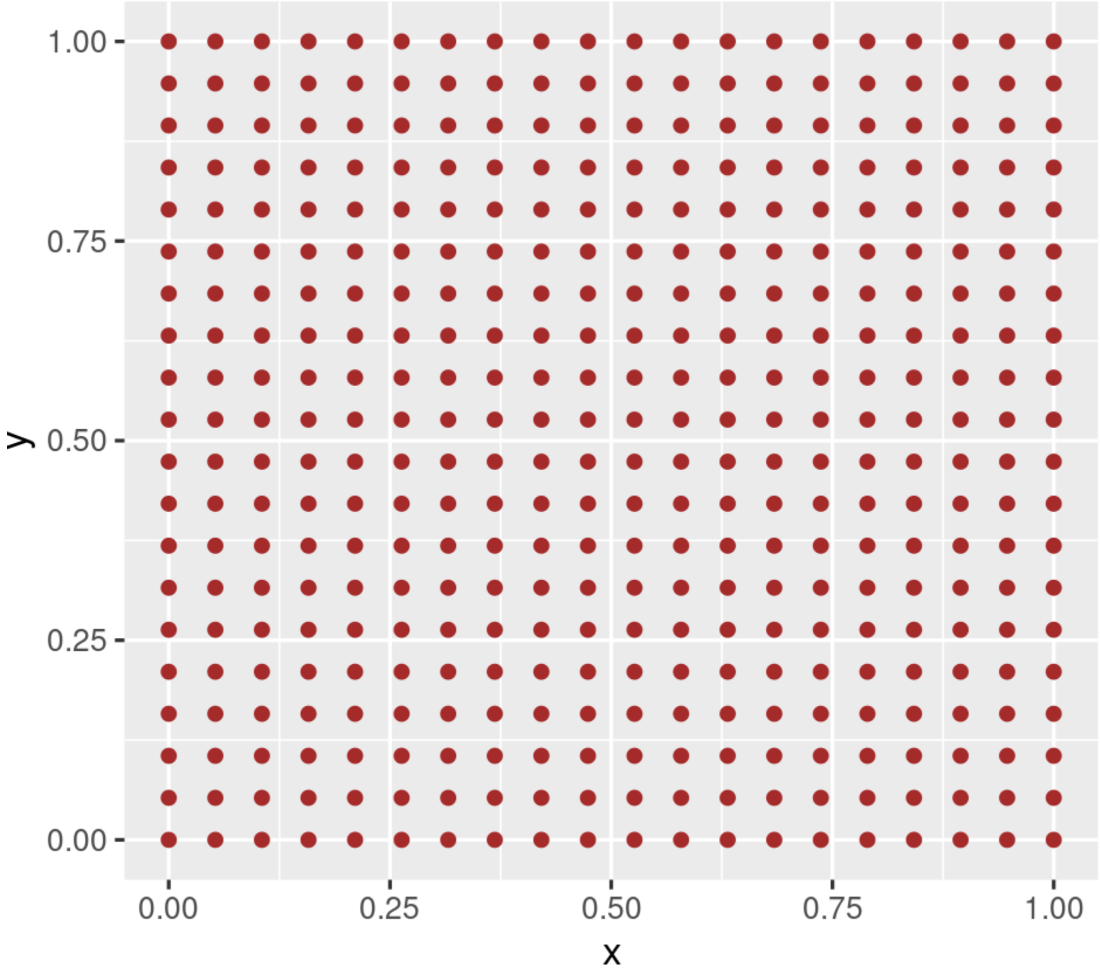
  we are interested in the posterior distribution of the linear predictor $\boldsymbol{\eta}$ everywhere in space, especially where we don't have observed data and `y=NA`.

- With regard to spatial prediction, it is worth noting that the INLA-SPDE algorithm provides the posterior conditional distribution of $\boldsymbol{\eta}$ for all the triangulation vertices. By using the SPDE approximation, it is then immediate to get a prediction for $\boldsymbol{\eta}$ for any location in the triangulated domain (i.e. the posterior predictive distribution).

# Grid for spatial prediction

Consider the following regular grid of points:

```
> grid.x = 20
> grid.y = 20
> pred.grid <- expand.grid(x = seq(0, 1, length.c
+                          y = seq(0, 1, length.c
```




It is necessary to define a new projector matrix:

```
> A.pred6 <- inla.spde.make.A(mesh = mesh6,
+                             loc = as.matrix(pre
> dim(A.pred6)
```

```
[1] 400 549
```

# Model fitting jointly with spatial prediction

For performing **jointly** the estimation and the prediction, we create a new `inla.stack` object:

```
> stack.pred <- inla.stack(data = list(y = NA),
+           A = list(1, A.pred6),
+           effects = list(intercept = rep(1, nrow(pred.grid)),
+                          spatial.field = 1:spde$n.spde),
+           tag = "pred")
```

and then we join it to the `inla.stack` object created previously for the estimation (`stack.est`):

```
> join.stack <- inla.stack(stack.est, stack.pred) #full stack object
```

And finally, we run `INLA` again:

```
> output6pred <- inla(formula,
+                   data = inla.stack.data(join.stack),
+                   control.compute = list(return.marginals=TRUE),
+                   control.predictor = list(A = inla.stack.A(join.stack), compute = TRUE)
+                   )
```

The option `return.marginals.predictor=TRUE` is necessary to obtain the marginals for the linear predictor.

# Retrieve the predictions

To access the predictions (posterior summary stats or marginal distribution) at the target grid locations, we extract with the `inla.stack.index()` function the corresponding indexes from the full stack object using the corresponding tag set before (`pred`):

```
> index.pred <- inla.stack.index(join.stack, tag = "pred")$data
> length(index.pred)
```

```
[1] 400
```

We then extract the prediction posterior mean and sd at the first 3 grid points:

```
> output6pred$summary.linear.predictor[index.pred[1:3],c("mean","sd")]
```

```
                    mean        sd
APredictor.201 12.24746 0.2611298
APredictor.202 12.06138 0.4239807
APredictor.203 10.58629 0.4333241
```

In this case (identity link) `output6pred$summary.fitted.values` would return the same output.

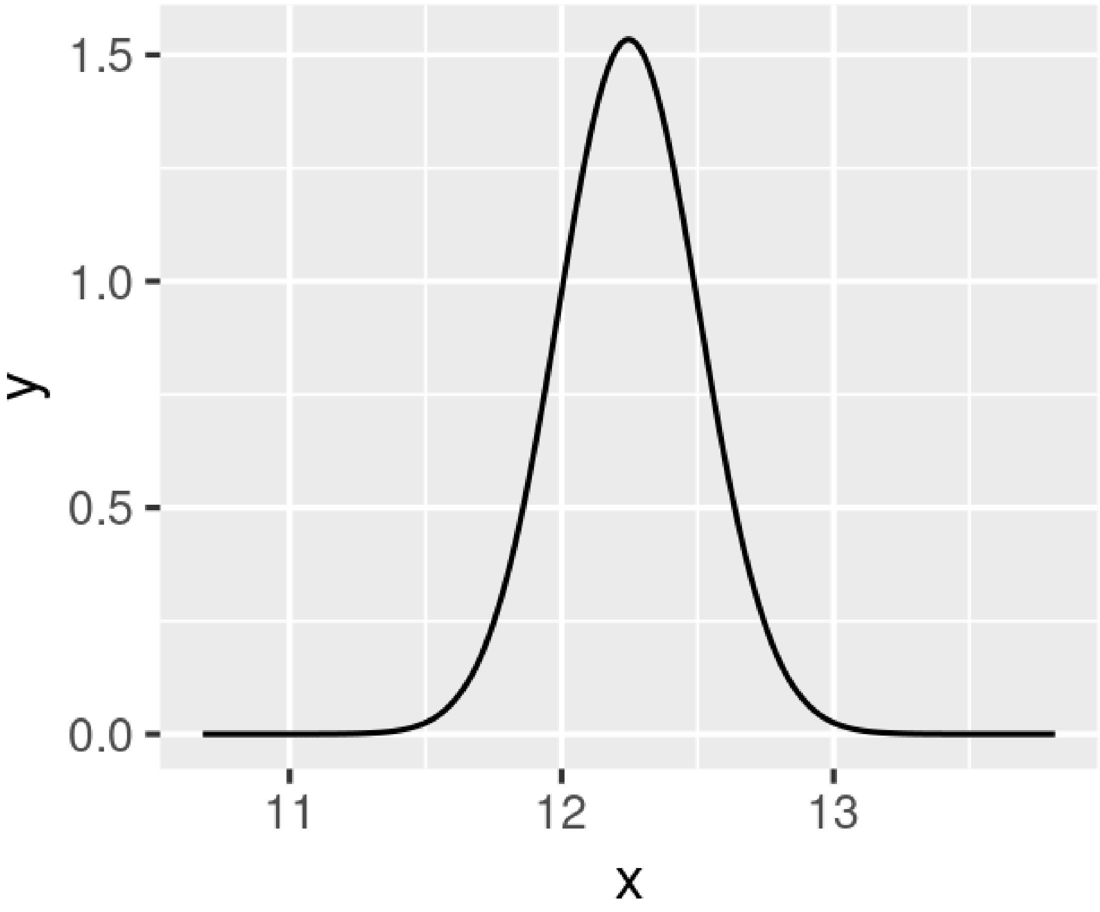
# Manipulate the posterior predictive distribution

As described in Day 1, it is possible to manipulate marginal distributions.

Consider for example the first grid point and its posterior predictive distribution:

```
> distr.point1 = output6pred$marginals.linear.pre
> distr.point1.smooth = inla.smarginal(distr.poir
```

```
> ggplot(data.frame(distr.point1.smooth)) +
+   geom_line(aes(x,y))
```




We can be also interested in computing the posterior probability of getting a value bigger than 13:

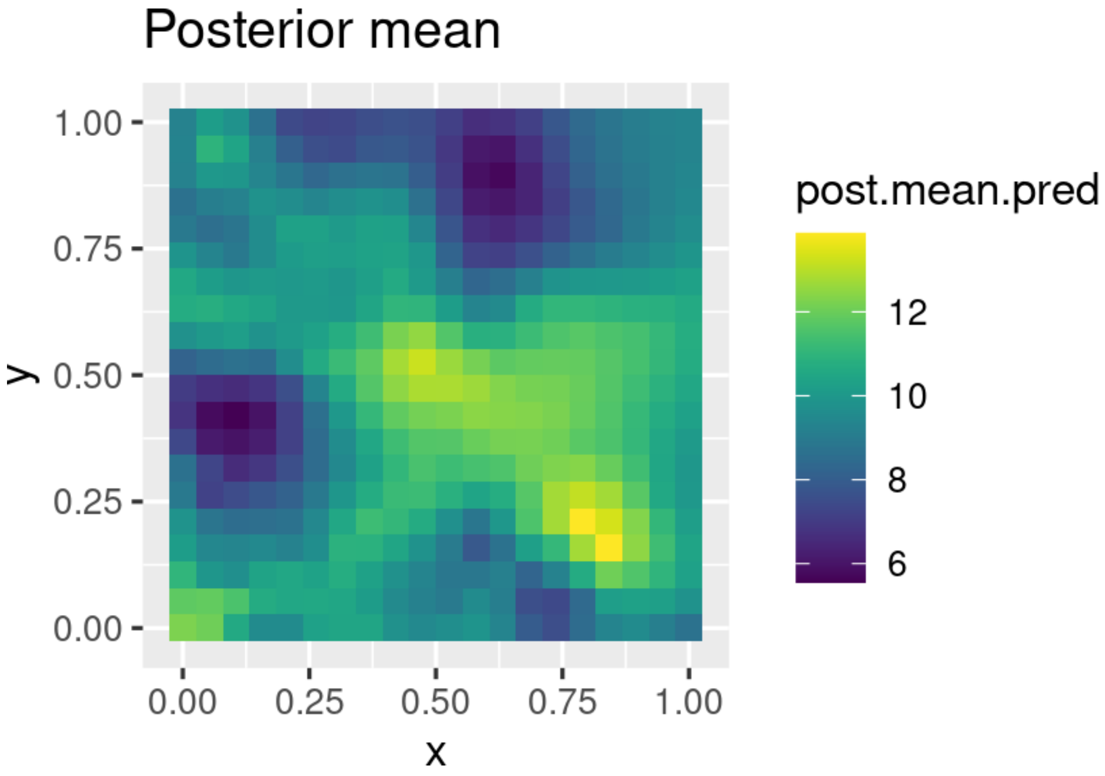
```
> 1 - inla.pmarginal(13, distr.point1)
```

```
[1] 0.002157059
```

# Mapping the linear predictor: posterior mean

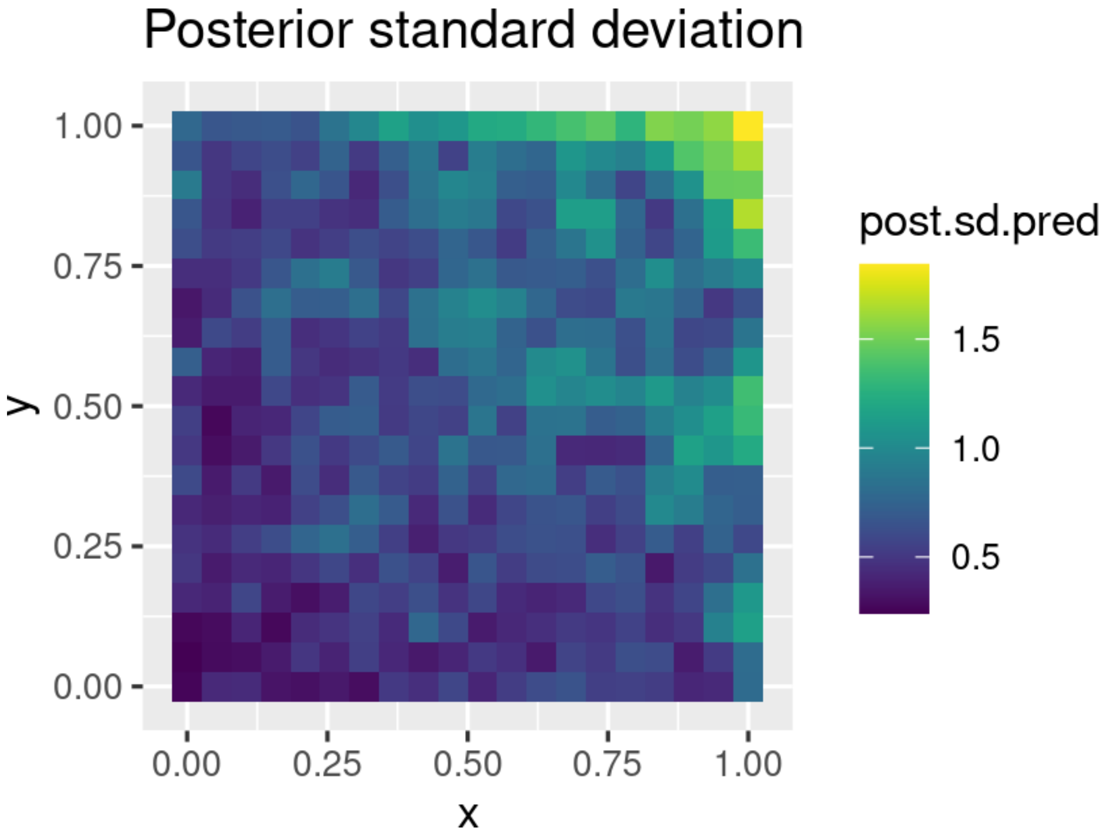
We plot now the posterior mean of the linear predictor at the grid level.

```
> library(inlabru)
> post.mean.pred = output6pred$summary.linear.pre
> post.mean.df = SpatialPixelsDataFrame(SpatialPo
+                                        data = da
>
>
> library(viridis)
> ggplot() +
+   gg(post.mean.df, aes(x, y, post.mean.pred)) +
+   ggtitle("Posterior mean") +
+   coord_fixed() + #x and y axis with the same l
+   scale_fill_viridis()
```



We plot the posterior standard deviation of the linear predictor at the grid level.

```
> post.sd.pred = output6pred$summary.linear.predi
>
> post.sd.df = SpatialPixelsDataFrame(SpatialPoir
+                                      data = data
>
> ggplot()+
+   gg(post.sd.df, aes(x, y, post.sd.pred)) +
+   ggtitle("Posterior standard deviation") +
+   coord_fixed() +
+  scale_fill_viridis()
```



# Model fitting and spatial prediction with `inlabru`

# inlabru

- `inlabru` is an `R` package for Bayesian spatial modelling, originally developed for ecological applications (Bachl, Lindgren, Borchers, and Illian, 2019) and the implementation of Log Gaussian Cox processes.

- `inlabru` is a wrapper around `INLA` taylored towards spatial data. It makes fitting spatial models with INLA easier as there is no more need to deal with projector matrices and stack objects. It also extends the class of models that can be fitted, including also distance sampling.

- The `inlabru` package works with spatial objects from the `sp` package (e.g. `SpatialPointsDataFrame`, `SpatialGridDataFrame`, `SpatialPixelsDataFrame`): https://cran.r-project.org/web/packages/sp/index.html

- The function `gg()` is an extension of the `ggplot()` function for generating geometries from spatial fitted object.

See:

- **Website**: https://sites.google.com/inlabru.org/inlabru
- **Github**: https://github.com/inlabru-org/inlabru

# Model fitting with inlabru

The function for fitting the model is called `bru()`:

```
> library(inlabru)
>
> bru(components = ...,
+     ...,
+     options = list(...)
+     )
```

where

- `components` is a formula-like specification of the model components
- `...` requires the specification of the likelihood and of the data. This will be done using the function `like`.
- When running the `bru` some options are set (for example `control.compute$dic=TRUE`): see all of them with `bru_options_default()`. With `options` we can for example specify the `inla` `control.fixed` and `control.compute` options.

# The function like()

The function `like()` makes it possible to specify the likelihood and other options related to the likelihood:

```
> like(
+   formula = . ~ .,
+   family = "gaussian",
+   data = NULL,
+   E = NULL,
+   Ntrials = NULL,
+   control.family = NULL,
+   ...)
```

where

- `formula` specifies how the `components` are combined to create the linear predictor (you will use the arbitrary names adopted for specifying the model components). It is not required when the linear predictor is the sum of all the terms in `components`.
- `family` (string) specifies the probability density function (PDF) of the response. All family types supported by the `INLA` package are supported by `inlabru`.

# The inlabru model components

- The function `f()` used with `INLA` is replaced by an **arbitrary name** which is assigned to the effect. For example consider a model with a linear prediction including an intercept, the linear effect of a covariate `x` and a generic random effect:

```
> y ~ Intercept(1) + x + yourREnane(main = index/covariate, model = "...", ...)
```

- Note that the specification of an implicit latent intercept is deprecated, and `+ Intercept(1)` or `+1` should be used instead.
- The available `model` specifications are:
  - `iid`
  - a spatial effect created previously using `inla.spde2.matern()` or `inla.spde2.pcmatern()`
  - all the models accepted by the INLA `f()` function.
- The linear effect model of the covariate `x` can also be expressed in the formula as

```
> beta(x, model="linear")
```

The difference is in the name used in the output: `x` (method 1) or `beta` (method 2).

# Very simple example with simulated data

```
> # Data simulation
> n1 <- 200
> x1 <- runif(n1)
> y1 <- rnorm(n1, mean = 3 + 2 * x1 )
> df1 <- data.frame(y = y1, x = x1)
>
> library(inlabru)
> # Model component
> cmp1 = y ~  Intercept(1) + x
>
> # Likelihood
> lik1 = like(formula = y ~ x + Intercept,
+             #formula = y ~ .,
+             family = "gaussian",
+             data = df1)
>
> # Model fit
> fit1 <- bru(cmp1, lik1)
> fit1$summary.fixed[,c("mean","sd")]
```

```
              mean        sd
Intercept 2.903588 0.1396076
x         1.928730 0.2536232
```

```
> # Model component
> cmp2 = y ~  Intercept(1) + beta(x, model="linea
> # Likelihood
> lik2 = like(formula = y ~ beta + Intercept,
+             family = "gaussian",
+             data = df1)
> # Model fit
> fit2 <- bru(cmp2, lik2)
> fit2$summary.fixed[,c("mean","sd")]
```

```
              mean        sd
Intercept 2.903588 0.1396076
beta      1.928730 0.2536232
```

# SPDEtoy example with inlabru

1. We first trasform the SPDEtoy dataframe into a `SpatialPointsDataFrame`:

```
> coordinates(SPDEtoy) = c("s1","s2")
> class(SPDEtoy)
```

```
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

2. We define the mesh and the `spde` object as done in Lecture 4.1:

```
> domain <- matrix(cbind(c(0,1,1,0.7,0), c(0,0,0.7,1,1)),ncol=2)
>
> mesh6 <- inla.mesh.2d(loc.domain = domain,
+                       max.edge = c(0.04, 0.2),
+                       cutoff = 0.05,
+                       offset = c(0.1, 0.4))
>
> spde = inla.spde2.matern(mesh = mesh6)
```

# SPDEtoy example with inlabru

3. Fit the model using `inlabru`:

```
> # Model components
> cmp_toy <- y ~ Intercept(1) + s.field(main = coordinates, model = spde)
> # Likehood
> like_toy <- like(formula = y ~ Intercept + s.field,
+                  data = SPDEtoy,
+                  family = "gaussian")
> # Run inlabru!
> fit <- bru(cmp_toy, like_toy)
> class(fit)
```

```
[1] "bru"   "iinla" "inla"
```

- Note that the function takes care of the construction of the projection matrices requied for the spatial SPDE model.
- The post-processing is exactly as with `INLA`:

```
> fit$summary.fixed[,c("mean","sd")]
```

```
              mean        sd
Intercept 9.507882 0.6833965
```

# Prediction with inlabru

4. It is suggested to transform the prediction grid into a `SpatialPixels` object (it is also possible to use the function `pixel()` that generates a `SpatialPixels` object covering an `inla.mesh`):

```
> coordinates(pred.grid) = c("x","y")
> gridded(pred.grid) = TRUE
> class(pred.grid)
```

```
[1] "SpatialPixels"
attr(,"package")
[1] "sp"
```

5. For the prediction we use the `predict` function that internally calls `generate()` in order to draw samples from the fitted model. It takes a fitted object given by `bru()` and produces predictions (100 by default) given a new set of values for the model covariates or the original values used for the model fit.

```
> pred <- predict(fit, pred.grid, ~ Intercept + s.field, seed = 1, n.samples = 500)
> class(pred)
```

```
[1] "SpatialPixelsDataFrame"
attr(,"package")
[1] "sp"
```

# Output from the predict function

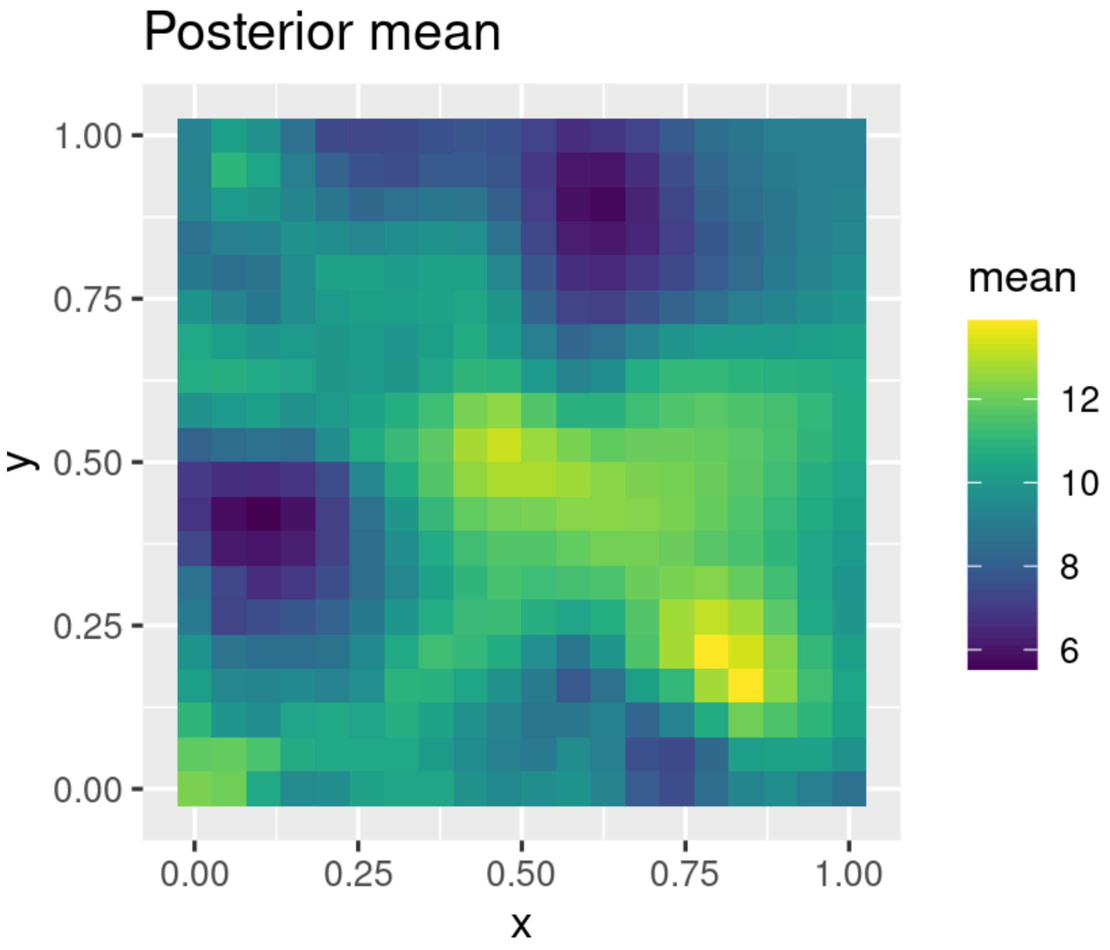
The argument `n.samples` (by default 100) specifies the number of samples to draw in order to calculate the posterior statistics.

```
> head(pred@data)
```

```
       mean        sd    q0.025    median   q0.975      smin     smax
1 12.240176 0.2654447 11.730881 12.234493 12.79419 11.558092 13.01762
2 12.059057 0.4088444 11.319155 12.055598 12.92104 10.833721 13.52903
3 10.556931 0.4230253  9.785362 10.568902 11.32133  9.029392 12.16952
4  9.505962 0.3029436  8.917529  9.494116 10.11430  8.711334 10.30435
5  9.512059 0.3047513  8.950011  9.506243 10.11148  8.536000 10.56451
6 10.254195 0.3309043  9.623914 10.247885 10.86162  8.775430 11.33845
          cv        var
1 0.02168634 0.07046086
2 0.03390351 0.16715372
3 0.04007085 0.17895037
4 0.03186881 0.09177485
5 0.03203841 0.09287334
6 0.03227014 0.10949766
```

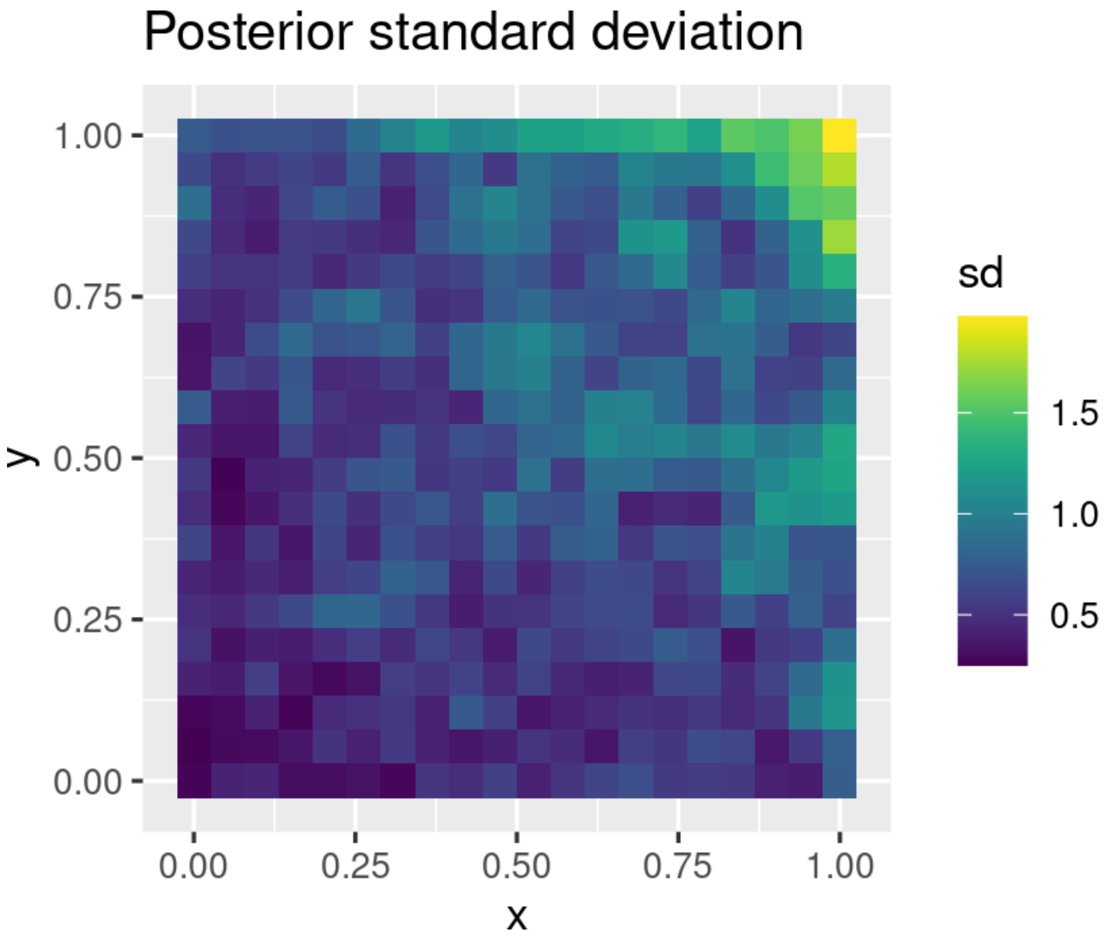
Note that it is possible to predict any function of any subset of the components of the model specification.

```
> ggplot() +
+   gg(pred, aes(x, y, fill = mean)) +
+   ggtitle("Posterior mean") +
+   coord_fixed() +
+   scale_fill_viridis()
```



# Mapping using gg: posterior standard deviation

```
> ggplot() +
+   gg(pred, aes(x, y, fill = sd)) +
+   ggtitle("Posterior standard deviation") +
+   coord_fixed() +
+   scale_fill_viridis()
```



# References

Bachl, F. E., F. Lindgren, D. L. Borchers, et al. (2019). "inlabru: an R package for Bayesian spatial modelling from ecological survey data". In: *Methods in Ecology and Evolution* 10.6, pp. 760-766.

Lindgren, F. and H. Rue (2015). "Bayesian Spatial Modelling with R-INLA". In: *Journal of Statistical Software* 63.19, pp. 1�-25. DOI: 10.18637/jss.v063.i19. URL: https://www.jstatsoft.org/index.php/jss/article/view/v063i19.