Session 6: Introduction to Stochastic Partial Differential Equations (SPDE) using R-INLA

Learning Objectives

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

- understand the basics of the Stochastic Partial Differential Equation (**SPDE**) approach;
- implement the SPDE approach using the `R-INLA` package;
- perform spatial prediction and mapping.

The topics covered in this lecture can be found in:

- Sections 6.5:6.8 of the book **Spatial and Spatio-Temporal Bayesian models with R-INLA** (website for data and code: https://github.com/michelacameletti/Spatial-and-Spatio-Temporal-Bayesian-Models-with-R-INLA)
- Chapters 8 and 9 of the book **Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny** https://www.paulamoraga.com/book-geospatial/index.html
- Chapter 2 of the book **Advanced Spatial Modeling with Stochastic Partial Differential Equations Using R and INLA** https://becarioprecario.bitbucket.io/spde-gitbook/

Outline

1. Basics of the SPDE approach
2. The SPDE approach with `R-INLA`
3. Spatial prediction using SPDE approach

The SPDE approach

A common model for geostatistical (noisy) data

- Usually the following (mixed-effects) model is assumed

$$y(\boldsymbol{s}) = \mu(\boldsymbol{s}) + \xi(\boldsymbol{s}) + \epsilon(\boldsymbol{s})$$

where

- $\mu(\boldsymbol{s})$ is the so-called large scale component, including linear or non linear of covariates.
- $\xi(\boldsymbol{s})$ is a zero mean latent **Gaussian spatial process** commonly assumed to be stationary and isotropic with covariance function $Cov(\xi(\boldsymbol{s}_i), \xi(\boldsymbol{s}_j))$ which depends only on the distance between the locations.
- $\epsilon(\boldsymbol{s})$ represents the Gaussian **measurement error** (independent from $\xi(\boldsymbol{s})$) and its variance (σ^2_ϵ) is usually known as **nugget effect**.

Given the data from n locations $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ we have:

$$\begin{gathered} y(s_i) \mid \mu(\boldsymbol{s}_i), \xi(\boldsymbol{s}_i), \sigma^2_\epsilon \sim \text{Normal}(\mu(\boldsymbol{s}_i) + \xi(\boldsymbol{s}_i), \sigma^2_\epsilon) \\ \boldsymbol{\xi} \sim \text{GF}(\mathbf{0}, \boldsymbol{\Sigma}) \end{gathered}$$

where $\boldsymbol{\Sigma}$ is a **dense** matrix defined by a spatial covariance function $\mathcal{C}(||\boldsymbol{s}_i - \boldsymbol{s}_j||)$.

- Several functions are available for the **spatial covariance function** (e.g. exponential, Matérn, spherical, etc.) parameterized by some parameters, e.g. spatial variance, range (Banerjee, Carlin, and Gelfand, 2014).

Matérn covariance function

The **Matérn covariance function** is defined by

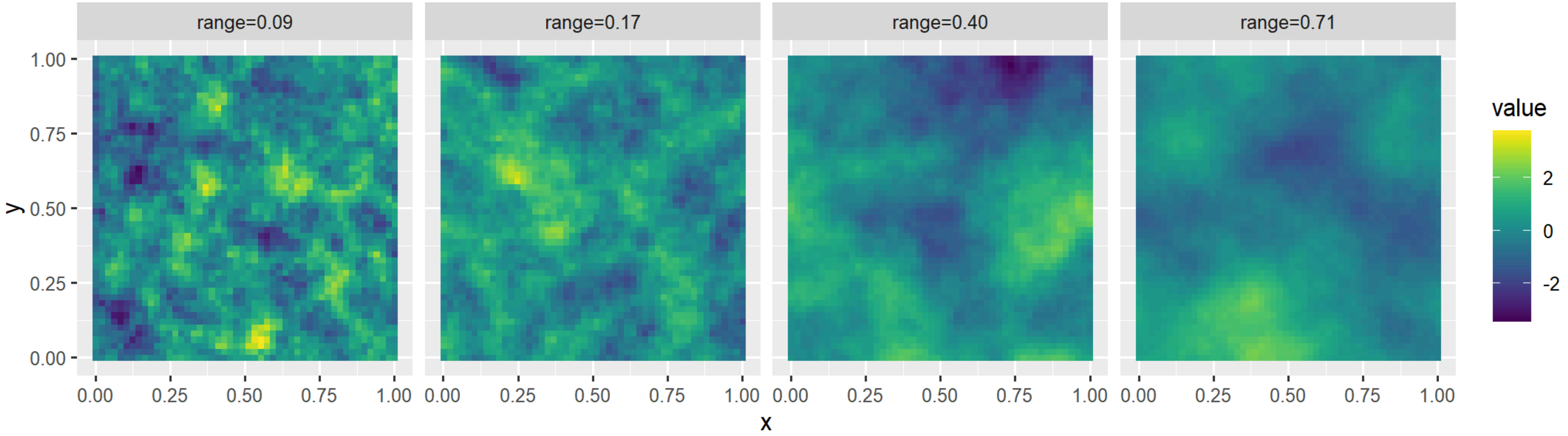
$$\mathrm{Cov}(\xi(\boldsymbol{s}_i), \xi(\boldsymbol{s}_j)) = \mathrm{Cov}(\xi_i, \xi_j) = \frac{\sigma^2}{\Gamma(\lambda)2^{\lambda-1}}(\kappa||\boldsymbol{s}_i - \boldsymbol{s}_j||)^\lambda K_\lambda(\kappa||\boldsymbol{s}_i - \boldsymbol{s}_j||)$$

where:

- $||\boldsymbol{s}_i - \boldsymbol{s}_j||$ is the Euclidean distance between two generic locations $\boldsymbol{s}_i, \boldsymbol{s}_j \in \mathbb{R}^2$,
- σ^2 is the marginal variance of the GF,
- K_λ denotes the modified Bessel function of second kind and order $\lambda > 0$, which measures the degree of **smoothness** of the process (it is usually kept fixed due to poor identifiability),
- $\kappa > 0$ is a **scale parameter** related to the **range** r, i.e. the distance at which the spatial correlation becomes almost null. Typically, the empirically derived definition for the range is $r = \frac{\sqrt{8\lambda}}{\kappa}$ (Lindgren, Rue, and Lindström, 2011), with r corresponding to the distance at which the spatial correlation is close to 0.1, for each $\lambda \geq 1/2$.
- The Matérn family is a very flexible class of covariance functions able to cover a wide range of spatial fields.

The effect of range

Simulation of different GFs with $\sigma = 1$ and different ranges:



Model based approach for estimation

- In the Bayesian framework, the classical approach for model estimation is Markov chain Monte Carlo methods (MCMC) considering that the likelihood function is a multivariate Gaussian distribution (see for example the `spBayes` and `spTimer` R packages).

- This requires to compute the Cholesky factorization of the dense covariance matrix $\boldsymbol{\Sigma}$ which is computationally very expensive. This is known as **big n problem** .

- The **stochastic partial differential equation (SPDE) approach** (Lindgren, Rue, and Lindström, 2011) is an alternative to the use of MCMC: it represents the continuous spatial process $\xi(\boldsymbol{s})$ with Matérn covariance function using a discretely indexed spatial random process (i.e., a Gaussian Markov Random Field - GMRF), which is characterized by a sparse precision matrix and enjoys computational benefits in terms of fast inference.

Introduction to the SPDE approach

The SPDE approach for Matérn GF

- The starting point is the **linear fractional stochastic partial differential equation** (SPDE)

$$(\kappa^2 - \Delta)^{\alpha/2}(\tau\xi(\boldsymbol{s})) = \mathcal{W}(\boldsymbol{s})$$

where $\boldsymbol{s} \in \mathbb{R}^d$, Δ is the Laplacian operator, $\alpha > 0$ is the **smoothness** term, $\kappa > 0$ is the scale parameter, τ controls the variance and $\mathcal{W}(\boldsymbol{s})$ is a Gaussian spatial white noise process (with unit variance).

- **Whittle** in 1954 showed that the exact and stationary solution to this SPDE is the stationary Gaussian field $\xi(\boldsymbol{s})$ with Matérn covariance function.
- **Lindgren, Rue, and Lindström (2011)** represent the solution of the SPDE using the finite element method (this is possible only for some values of the smothness parameter).

- In $\mathbb{R}^2$ the link between the SPDE parameters τ, α, κ and the Matérn function parameters $\sigma^2, \lambda, \kappa$ is given by

$$\lambda = \alpha - 1 \qquad \sigma^2 = \frac{\Gamma(\lambda)}{\Gamma(\alpha)(4\pi)\kappa^{2\lambda}\tau^2}$$

- In `R-INLA` the default value for the smoothness parameter is $\alpha = 2 \rightarrow \lambda = 1$. Consequently

$$r = \sqrt{8\lambda}/\kappa = \sqrt{8}/\kappa \text{ (see slide 6)} \qquad \sigma^2 = 1/(4\pi\kappa^2\tau^2)$$

Piecewise linear approximation

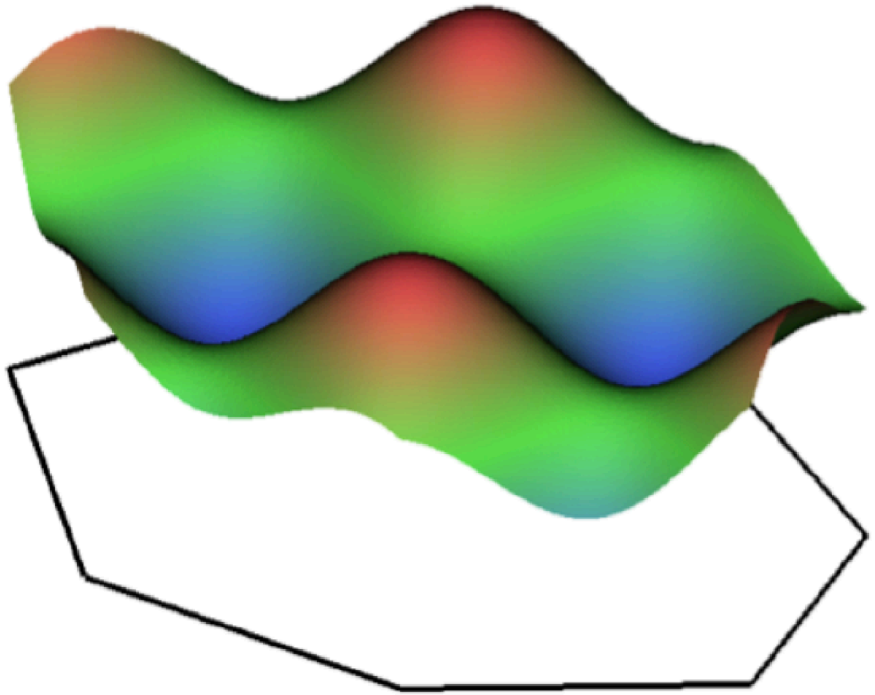
- The solution to the SPDE can be approximated through a **basis function representation** defined on a **triangulation** of the spatial domain

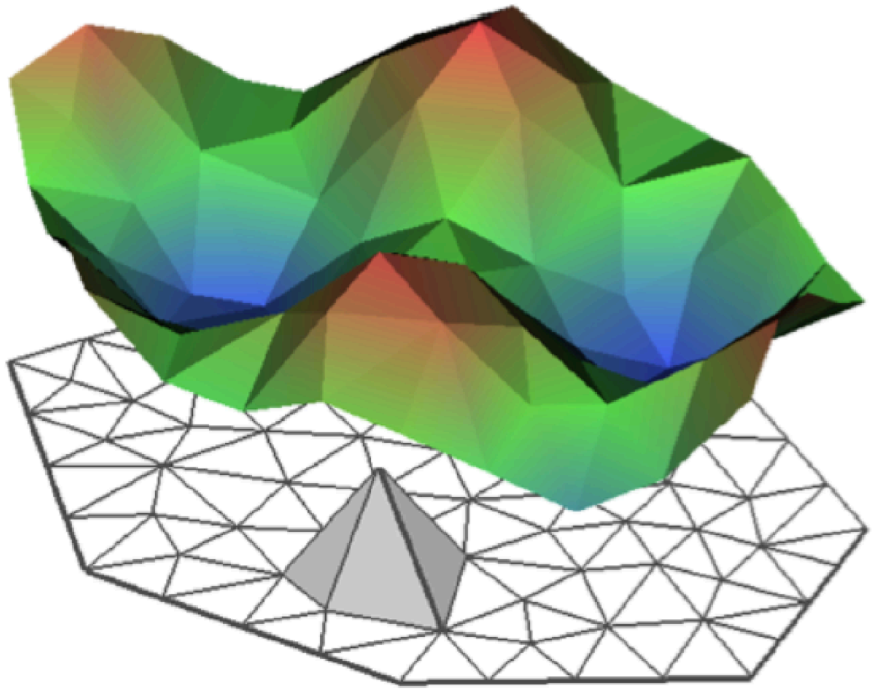
$$\xi(\boldsymbol{s}) = \sum_{g=1}^{G} \varphi_g(\boldsymbol{s})\tilde{\xi}_g$$

where G is the total number of triangulation vertices, $\{\varphi_g\}$ is the set of (deterministic) basis functions and $\{\tilde{\xi}_g\}$ are zero mean Gaussian distributed weights.

- In order to obtain a Markov structure, the basis functions are chosen to be piecewise linear in each triangle, i.e. φ_g is 1 at vertex g and 0 at all other vertices.

Approximation in 2-Dimensions





Piecewise linear approximation

- The precision matrix $\boldsymbol{Q}$ for the Gaussian weight vector $\tilde{\boldsymbol{\xi}} = \{\tilde{\xi}_1, \ldots, \tilde{\xi}_G\}$ is given by

$$\boldsymbol{Q} = \tau^2 \left(\kappa^4 \boldsymbol{C} + 2\kappa^2 \boldsymbol{G} + \boldsymbol{G}\boldsymbol{C}^{-1}\boldsymbol{G}\right)$$

where

- the generic element of the diagonal matrix $\boldsymbol{C}$ is $C_{ii} = \int \varphi_i(\boldsymbol{s})\mathrm{d}\boldsymbol{s}$,
- the generic element of the sparse matrix $\boldsymbol{G}$ is $G_{ij} = \int \nabla\varphi_i(\boldsymbol{s})\nabla\varphi_j(\boldsymbol{s})\mathrm{d}\boldsymbol{s}$ (where ∇ denotes the gradient),

The precision matrix $\boldsymbol{Q}$, whose elements depend on τ and κ, is sparse and consequently $\boldsymbol{\xi}$ is a GMRF distributed as $\mathrm{Normal}(\boldsymbol{0}, \boldsymbol{Q}^{-1})$: it represents the approximated solution to the SPDE.

Take home message

- Do the modelling using GF and the computations using the GMRF representation (computational advantages thanks to algorithms for sparse matrices):

$$\boldsymbol{\xi} \sim \mathrm{Normal}(\mathbf{0}, \boldsymbol{\Sigma}) \Longrightarrow \tilde{\boldsymbol{\xi}} \sim \mathrm{Normal}\left(\mathbf{0}, \boldsymbol{Q}^{-1}\right)$$

The SPDE approach with `R-INLA`

SPDE toy example

We use the `SPDEtoy` dataset , consisting in 200 simulated values for the variable `y` which refer to as many randomly sampled locations in the unit square area delimited by the points $(0, 0)$ and $(1, 1)$ and with coordinates given by `s1` and `s2`.

```
> library(INLA)
> data(SPDEtoy)
> dim(SPDEtoy)
```

```
[1] 200   3
```

```
> head(SPDEtoy, n=3)
```

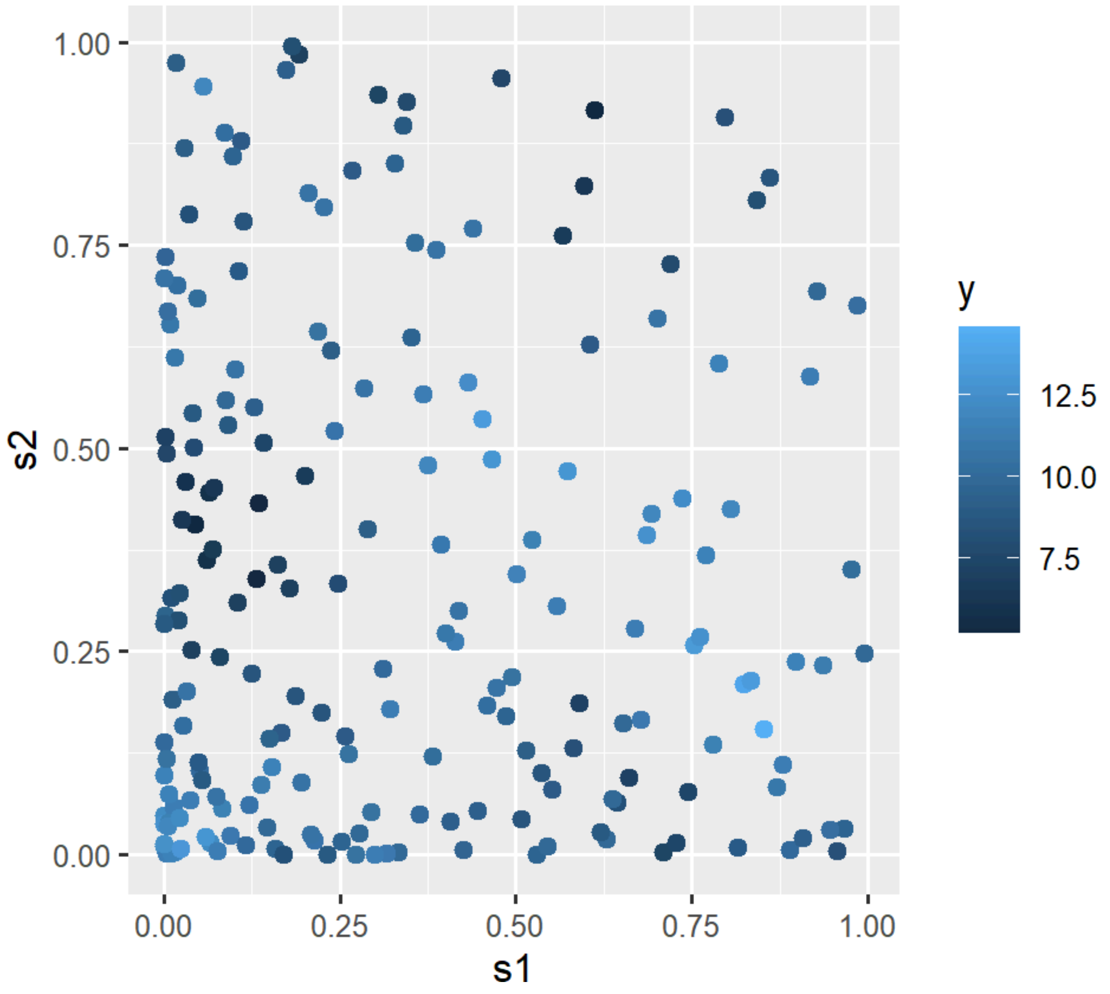
```
          s1         s2         y
1 0.08265625 0.05640625 11.521206
2 0.61230625 0.91680625  5.277960
3 0.16200625 0.35700625  6.902959
```

```
> summary(SPDEtoy$y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  5.210   8.688  10.023   9.858  11.307  14.600 
```

Plot of toy data

```
> library(tidyverse)
>
> SPDEtoy %>%
+   ggplot() +
+   geom_point(aes(s1,s2, col=y),size=2)
```



SPDE toy example: model for simulation

- The model used for simulating the `SPDEtoy` data assumes that the distribution of the observation y_i is

$$y_i \mid \eta_i, \sigma^2_e \sim \text{Normal}(\eta_i, \sigma^2_e) \qquad i = 1, \ldots, 200$$

where σ^2_e is the variance of the zero mean measurement error e_i which is supposed to be Gaussian iid.

- The response mean, which coincides with the **linear predictor**, is defined as

$$\eta_i = b_0 + \xi_i$$

and includes the intercept b_0 and a random effect represented by ξ_i, which is the realization of the latent GF $\xi(\boldsymbol{s}) \sim \text{MVNormal}(\mathbf{0}, \boldsymbol{\Sigma})$. The covariance matrix $\boldsymbol{\Sigma}$ is defined by the Matérn spatial covariance function.

- The parameter values chosen for simulating the data are: $b_0 = 10$, $\sigma^2_e = 0.3$, $\sigma^2 = 5$, $\kappa = 7$, $r = \frac{\sqrt{8}}{\kappa} = 0.404$.

The SPDE representation and the projection matrix

- Using the SPDE basis function representation, the linear predictor η_i can be rewritten as

$$\eta_i = b_0 + \sum_{g=1}^{G} \varphi_g(\boldsymbol{s}_i)\tilde{\xi}_g$$

where $\varphi_g(\boldsymbol{s}_i)$ is the value of the g-th basis function evaluated in $\boldsymbol{s}_i$.

- More generally it is possible to express the linear predictor as

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

with $A_{ig} = \varphi_g(\boldsymbol{s}_i)$ being the generic element of the sparse matrix $\boldsymbol{A}$ (known as **projection matrix**) which maps the GMRF $\tilde{\boldsymbol{\xi}}$ from the G triangulation vertices to the n observation locations. This allows the SPDE model to be treated as standard indexed random effects.

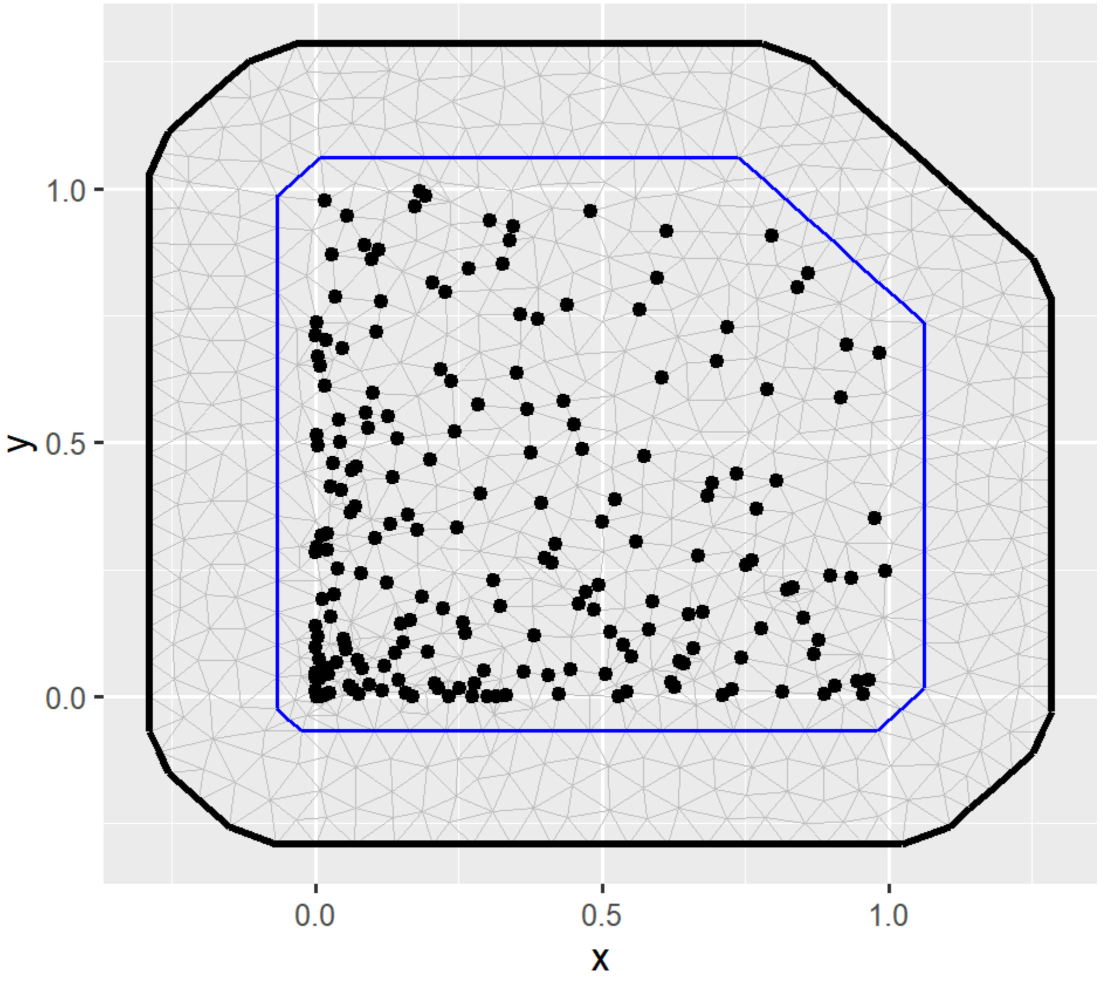
Mesh construction

- The SPDE approach is based on a triangulation of the spatial domain given by the **mesh**.
- The definition of the mesh is a **trade-off** between the accuracy of the GMRF representation and computational costs, both depending on the number of vertices used in the triangulation: the bigger the number of mesh triangles, the finer the GF approximation but the higher the computational costs.
- To create the mesh in `R-INLA` we use the helper function `inla.mesh.2d`. The arguments for a two-dimensional mesh construction can be checked using `args(inla.mesh.2d)`.
- Here we are going to start with the following options:
  - `loc` or `loc.domain`: specify information about the spatial domain
  - `max.edge`: specify the largest allowed triangle edge length. If a vector of two values is provided, the spatial domain is divided into an inner and an outer area whose triangle resolution is specified by `max.edge` (the higher the value for `max.edge` the lower the resolution and the accuracy).

Mesh: max.edge - mesh1

```
> coords = as.matrix(SPDEtoy[,1:2])
> mesh1 = inla.mesh.2d(loc = coords,
+                      max.edge = c(0.1, 0.1))
```

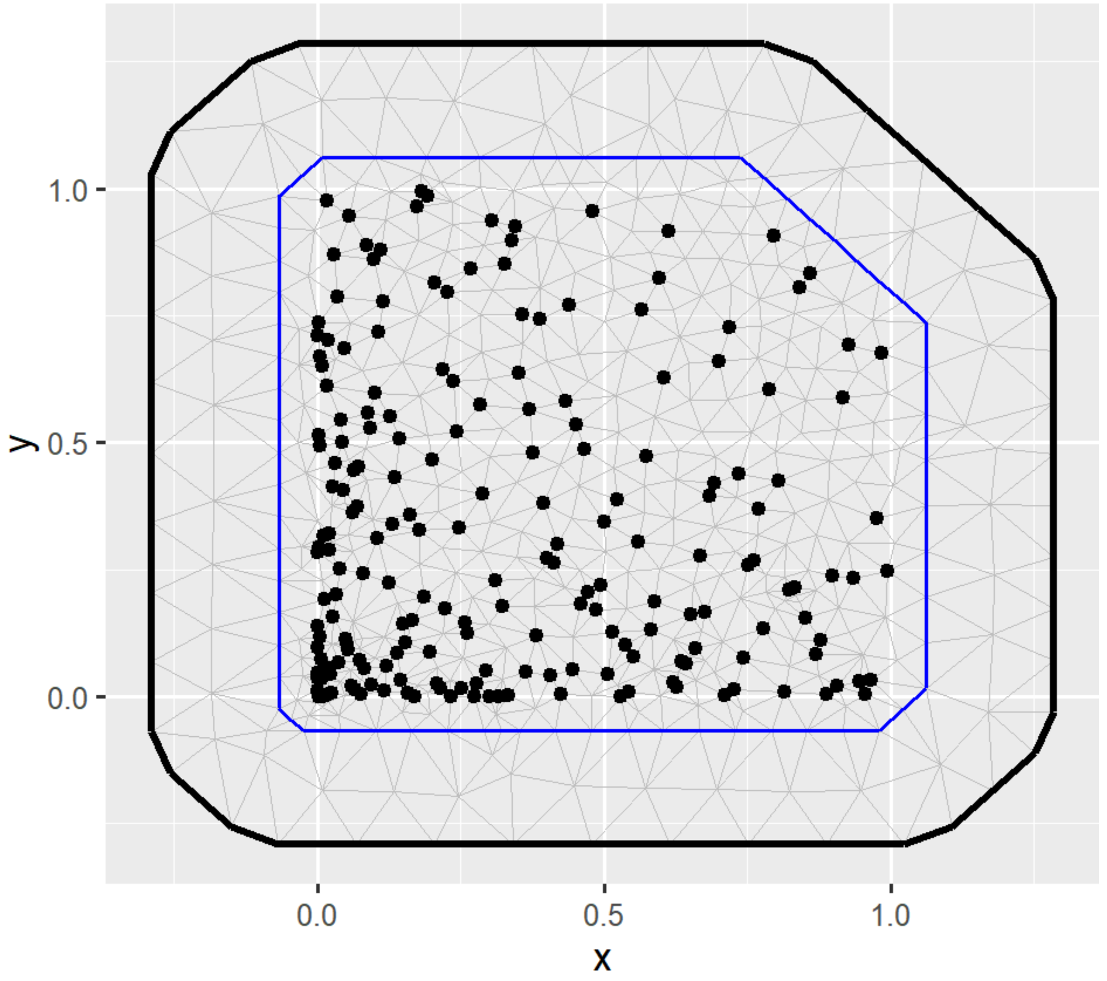
```
> library(inlabru)
> ggplot() +
+   gg(mesh1) +
+   geom_point(data = data.frame(coords),
+              aes(s1, s2))
```



Mesh: max.edge - mesh2

```
> mesh2 = inla.mesh.2d(loc = coords,
+                       max.edge = c(0.1, 0.2))
```

```
> ggplot() +
+   gg(mesh2) +
+   geom_point(data = data.frame(coords),
+              aes(s1, s2))
```

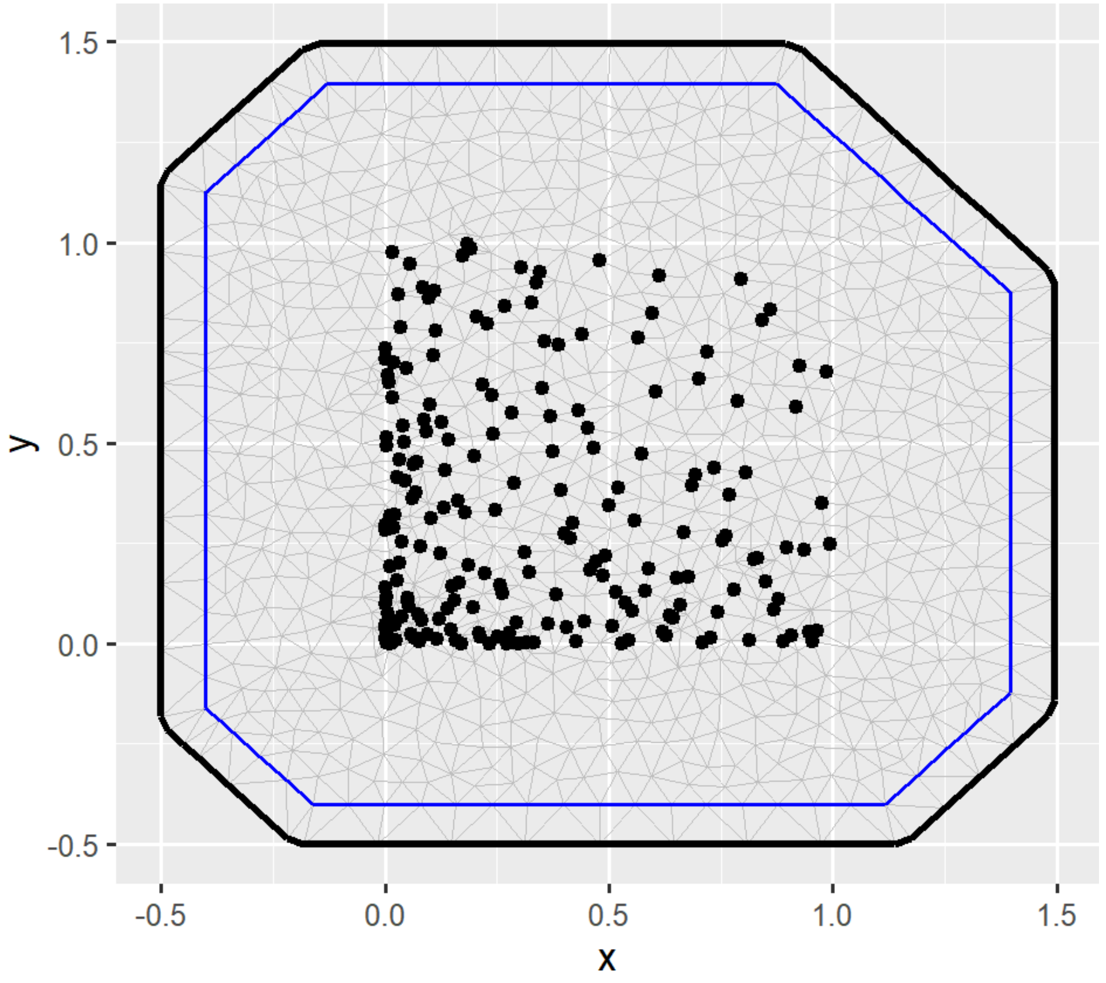


Mesh optional arguments: offset - mesh3

The option `offset` of the `inla.mesh.2d` function can be used to define how much the domain should be extended in the inner and outer part. The default values are `offset = c(-0.05, -0.15)`.

```
> mesh3 = inla.mesh.2d(loc = coords,
+                      max.edge = c(0.1, 0.2),
+                      offset = c(0.4,0.1))
```

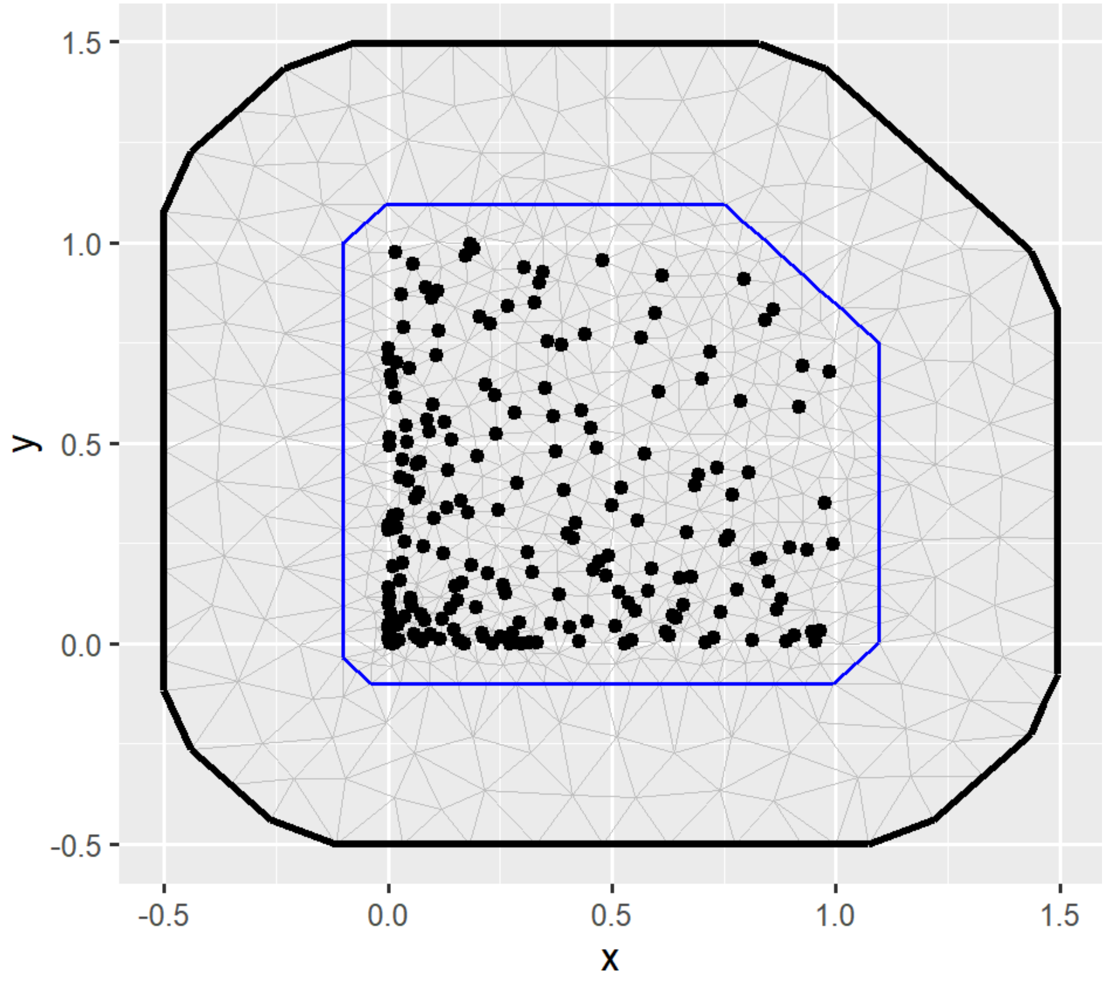
```
> ggplot() +
+   gg(mesh3)+
+   geom_point(data = data.frame(coords),
+              aes(s1, s2))
```



Mesh optional arguments: offset - mesh4

```
> mesh4 = inla.mesh.2d(loc = coords,
+                      max.edge = c(0.1, 0.2),
+                      offset = c(0.1,0.4))
```

```
> ggplot() +
+   gg(mesh4) +
+   geom_point(data = data.frame(coords),
+              aes(s1, s2))
```

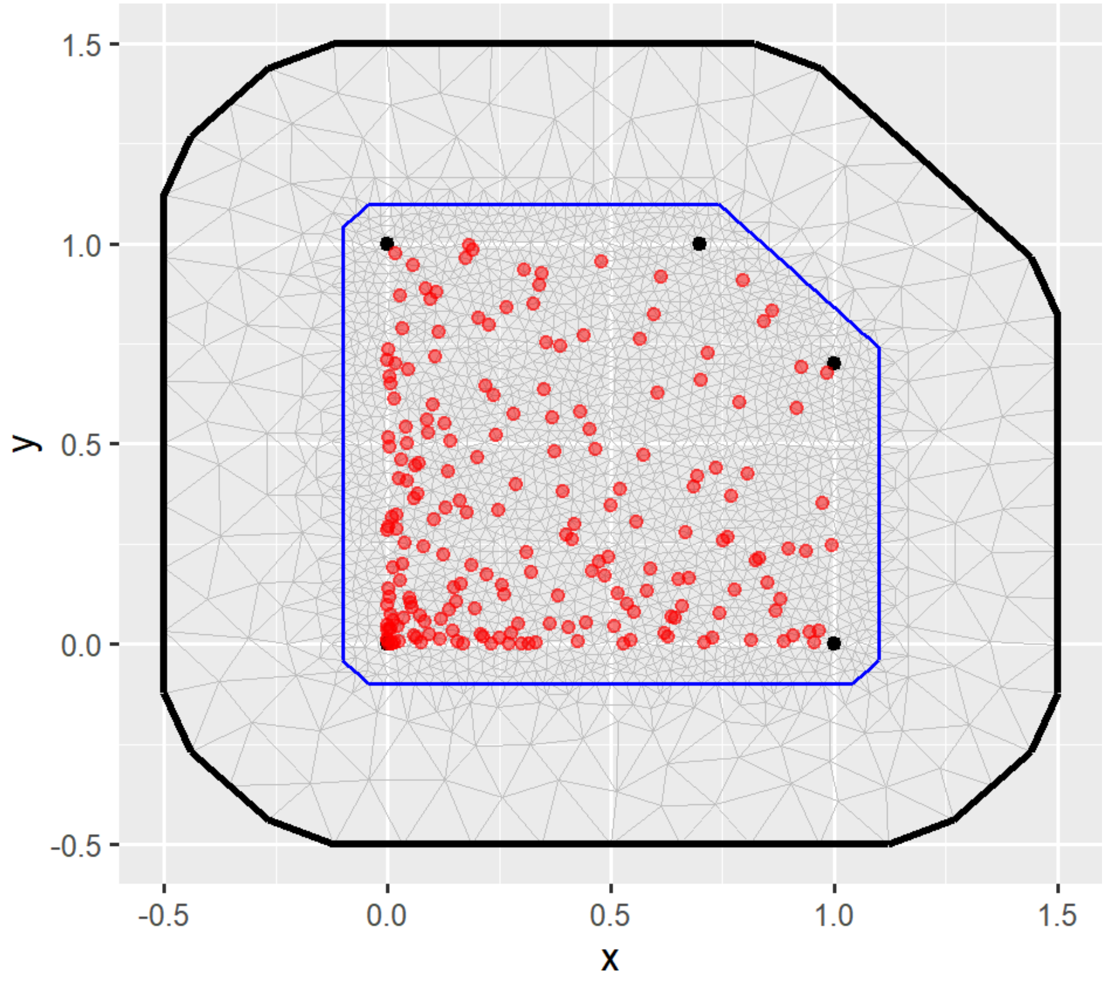


Mesh optional arguments cutoff

Instead of the point coordinates, it is possible to use other point locations used to determine the domain extent.

```
> domain = matrix(cbind(c(0,1,1,0.7,0),
+                       c(0,0,0.7,1,1)), ncol=2)
> mesh5domain = inla.mesh.2d(loc.domain = domain,
+                            max.edge = c(0.04, 0.2),
+                            offset = c(0.1, 0.4))
```

```
> ggplot() +
+   gg(mesh5domain) +
+   geom_point(data = data.frame(domain),
+              aes(X1, X2)) +
+     geom_point(data = data.frame(coords),
+              aes(s1, s2), col = "red",
+              alpha = 0.5)
```

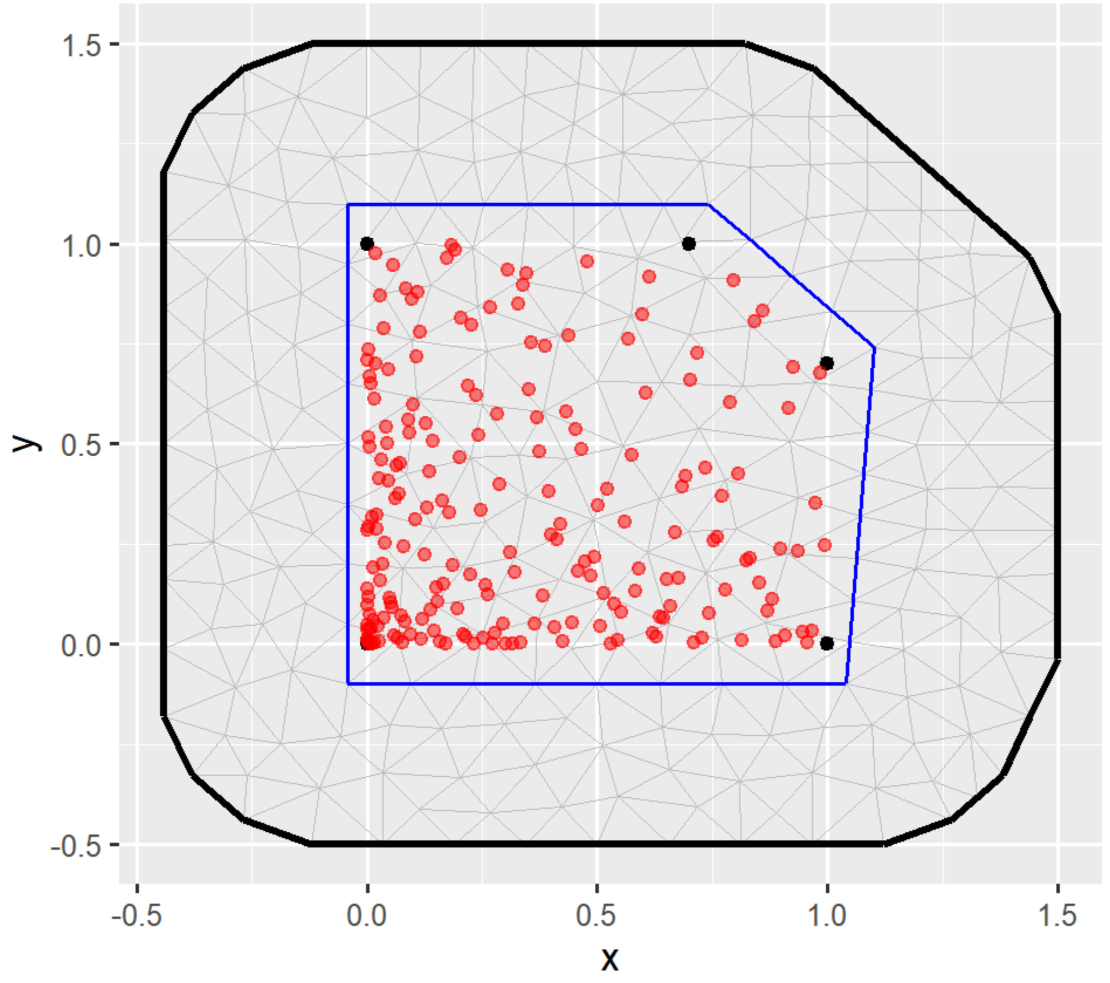


Mesh optional arguments cutoff - mesh5

The option `cutoff` can be used to avoid building too many small triangles around clustered data locations (the default value is equal to 0).

```
> mesh5 = inla.mesh.2d(loc.domain = domain,
+                       max.edge = c(0.04, 0.2),
+                       cutoff = 0.5,
+                       offset = c(0.1, 0.4))
```

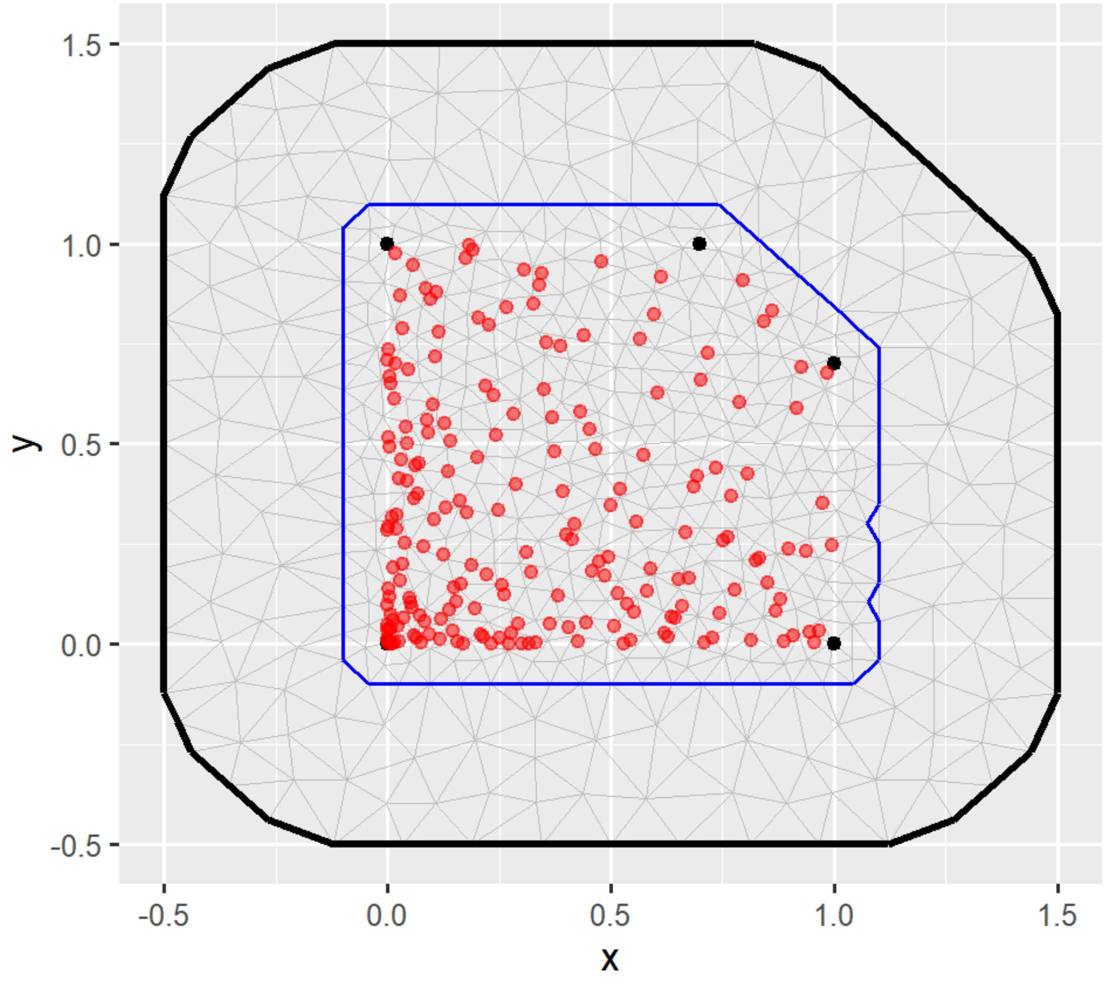
```
> ggplot() +
+   gg(mesh5) +
+   geom_point(data = data.frame(domain),
+              aes(X1, X2)) +
+     geom_point(data = data.frame(coords),
+              aes(s1, s2), col = "red",
+              alpha = 0.5)
```



Putting all together - mesh6

```
> mesh6 = inla.mesh.2d(loc.domain = domain,
+                      max.edge = c(0.04, 0.2),
+                      cutoff = 0.05,
+                      offset = c(0.1, 0.4))
```

```
> ggplot() +
+   gg(mesh6) +
+   geom_point(data = data.frame(domain),
+              aes(X1, X2)) +
+     geom_point(data = data.frame(coords),
+              aes(s1, s2), col = "red",
+              alpha = 0.5)
```

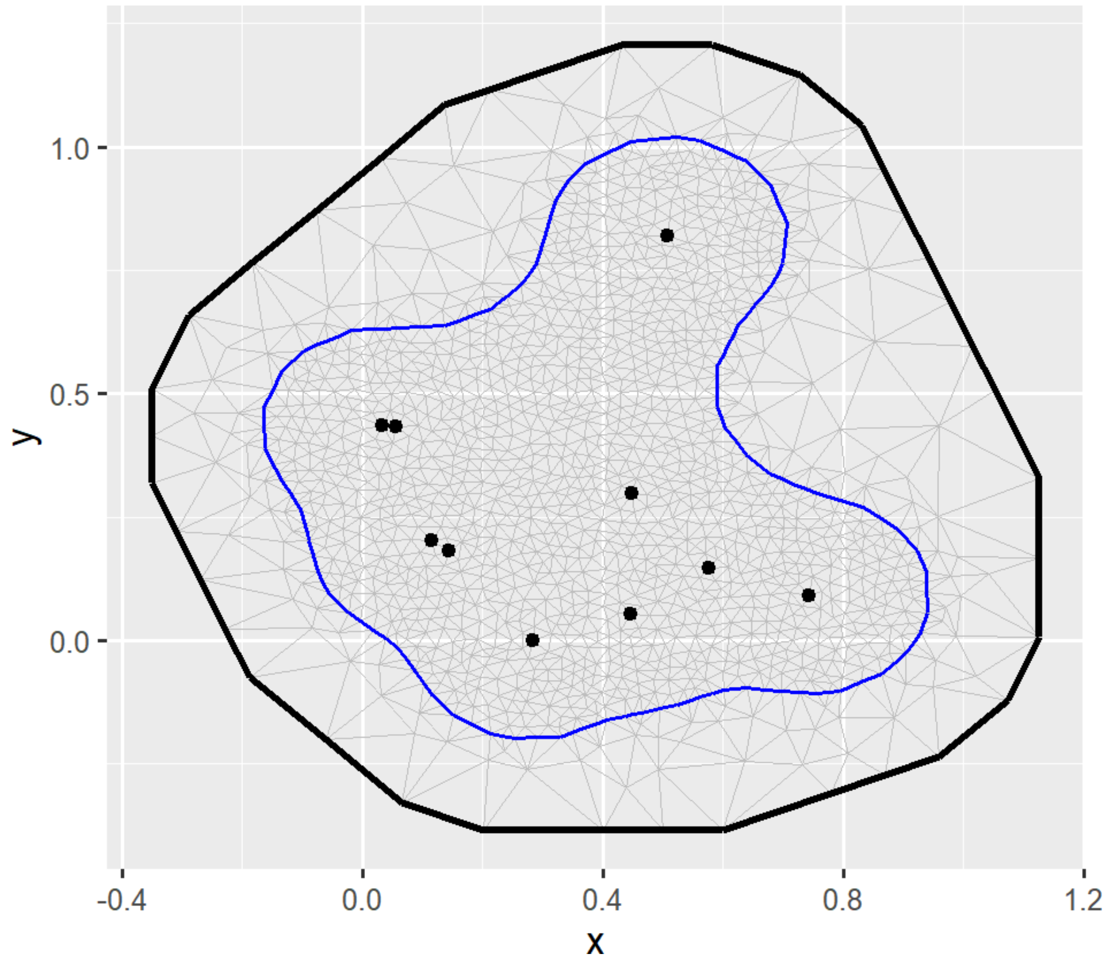


Mesh: non convex hull

A feature in `R-INLA` named `inla.nonconvex.hull` makes it possible to compute a non convex hull to be included as boundary in the mesh construction. This can be particularly useful when the shape of the domain is of some importance.

```
> set.seed(44)
> loc = matrix(runif(20), 10, 2)
>
> boundary = inla.nonconvex.hull(loc,
+                               convex=0.2)
> meshNC = inla.mesh.2d(loc = loc,
+                       boundary = boundary,
+                       max.edge = c(0.04, 0.2))
```

```
> ggplot() +
+   gg(meshNC)+
+   geom_point(data = data.frame(loc),
+              aes(X1, X2))
```



Creation of the Matérn SPDE model for `mesh6`

- We create the Matérn **SPDE model object** with the function `inla.spde2.matern` on the mesh. By default the smoothness term is 2
- Here we create the SPDE object using the `mesh6` triangulation:

```
> spde = inla.spde2.matern(mesh = mesh6)
> spde$n.spde # number of vertices in the mesh
```

```
[1] 549
```

PC prior for the Matérn model

- Instead of `inla.spde2.matern` it is possible to use `inla.spde2.pcmatern` for creating an `inla.spde2` model object using a PC prior for the range r and the marginal standard deviation σ (Simpson, Rue, Riebler, Martins, and Sorbye, 2017).
- The prior for σ is such that
$$Pr(\sigma > \sigma_0) = p$$
and this requires to specify σ_0 and p with the option `prior.sigma = c(sigma0,p)`.
- The prior for r is such that
$$Pr(r < r_0) = p$$
and this requires to specify r_0 and p with the option `prior.range = c(r0,p)`.

For example:

```
> spde = inla.spde2.pcmatern(mesh,
+                            prior.range = c(0.01,0.1),
+                            prior.sigma = c(100,0.1))
```

Creation of the indexes for the SPDE model

- Now we construct the index set for the latent spatial Gaussian model using the function `inla.spde.make.index`. It will be useful in the stack preparation (see next slides)

```
> s.index = inla.spde.make.index(name = "spatial.field", n.spde = spde$n.spde)
```

Here the `name` represents the name of the effect which will be used in the `formula`

- The index set for the latent field does not depend on the data set locations. It only depends on the SPDE model size

Projection matrix for mesh6

- Now, we create a projection matrix, which maps the GMRF from the mesh nodes to the n observation locations
- The projection matrix is build using the `inla.spde.make.A()` function

```
> A.est6 = inla.spde.make.A(mesh = mesh6,
+                           loc = coords)
> dim(A.est6)
```

```
[1] 200 549
```

The matrix has the number of rows equal to the number of observations and the number of columns equal to the number of vertices of the triangulation.

The points are not necessarily on the vertices and thus each point will be associated with up to three non-zero weights (that collectively add up to 1)

```
> #No more than 3 elements in each line are non-z
> table(rowSums(A.est6>0))
```

The sum of each row is one:

```
> table(rowSums(A.est6))
```

```
  1
200
```

There are some columns whose sum is zero corresponding to triangles with no point location inside:

```
> table(colSums(A.est6) > 0)
```

```
FALSE  TRUE
  313   236
```

The `inla.stack()` function

- The function `inla.stack()` allows us to perform an optimal and easy management of the SPDE objects (data, covariates, indices and projection 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.

- The main arguments of the `inla.stack()` function are:
  – `data`: a vector list with the data
  – `A`: a list of projection 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

We define the `inla.stack` object for model fitting as follows:

```
> stack.est = inla.stack(
+   data = list(y = SPDEtoy$y),
+   A = list(1, A.est6),
+   effects = list(intercept = rep(1,nrow(SPDEtoy)),
+                  s.index),
+   tag="est")
```

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

Model fitting

- The vector of parameters is defined as $\boldsymbol{\theta} = \{\tilde{\boldsymbol{\xi}}, b_0\}$ with hyper-parameter vector $\boldsymbol{\psi} = (\sigma_e^2, r, \tau)$, where $\tau = 1/\sigma^2$ and r (depending on κ) are the Matérn covariance function parameters.

- In R-INLA the default **internal representation** for the SPDE parameters is $\log(\tau) = \theta_1$ and $\log(\kappa) = \theta_2$, with θ_1 and θ_2 being given two independent Normal prior distributions.
- Steps: (1) Define the formula, then (2) Run INLA

1. **Define the linear predictor** through the `formula`. In R-INLA the Matérn GF is part of the linear predictor and is specified in the `formula` environment using a proper specification for `f()`.

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

where `spatial.field` is a proper index variable (from `inla.spde.make.index` function), and `spde` is the model created previously with `inla.spde2.matern`. Note that the intercept is removed and is added manually in the linear predictor.

2. Run `inla`! The function `inla.stack.data()` and `inla.stack.A()` are used for extracting the data and the projection 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))
```

Note that the projection matrix is passed to `inla` through `control.predictor`. Moreover, with the option `compute = TRUE` we ask for the computation of the marginals of the linear predictor.

Exploring the output: fixed effects and hyperparameters

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

```
              mean 0.025quant 0.975quant
intercept 9.502363   8.014827   10.88159
```

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

```
                                             mean 0.025quant 0.975quant
Precision for the Gaussian observations  2.866417   2.040619   3.903301
Theta1 for spatial.field                -4.033909  -4.342832  -3.712049
Theta2 for spatial.field                 2.094492   1.594871   2.568667
```

Exploring the output: spatial parameters

If we are interested in the posterior summaries of the spatial parameters on the scale of the variance $\sigma^2 = 1/\tau$ and range r (instead of the internal scale regarding $\theta_1 = \log(\tau)$ and $\theta_2 = \log(\kappa)$) we use

```
> output6.field = inla.spde2.result(inla = output6,
+                                   name = "spatial.field",
+                                   spde = spde)
```

The resulting list contains the following elements:

```
> names(output6.field)
```

```
 [1] "summary.values"                 "marginals.values"
 [3] "summary.hyperpar"               "summary.theta"
 [5] "summary.log.tau"                "summary.log.kappa"
 [7] "summary.log.variance.nominal"   "summary.log.range.nominal"
 [9] "marginals.theta"                "marginals.log.tau"
[11] "marginals.log.kappa"            "marginals.log.variance.nominal"
[13] "marginals.log.range.nominal"    "marginals.tau"
[15] "marginals.kappa"                "marginals.variance.nominal"
[17] "marginals.range.nominal"
```

Exploring the output

The posterior mean of σ^2 and the range r can be obtained by typing

```
> inla.emarginal(function(x) x, output6.field$marginals.variance.nominal[[1]])
```

```
[1] 4.007354
```

```
> inla.emarginal(function(x) x, output6.field$marginals.range.nominal[[1]])
```

```
[1] 0.3589909
```

Also the other standard INLA functions can be applied to the marginal posteriors:

```
> inla.zmarginal(output6.field$marginals.range.nominal[[1]])
```

```
Mean               0.358991
Stdev              0.0905221
Quantile  0.025 0.217412
Quantile  0.25  0.293988
Quantile  0.5   0.346256
Quantile  0.75  0.410384
Quantile  0.975 0.570953
```

Spatial prediction using SPDE approach

Spatial predictions

- 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

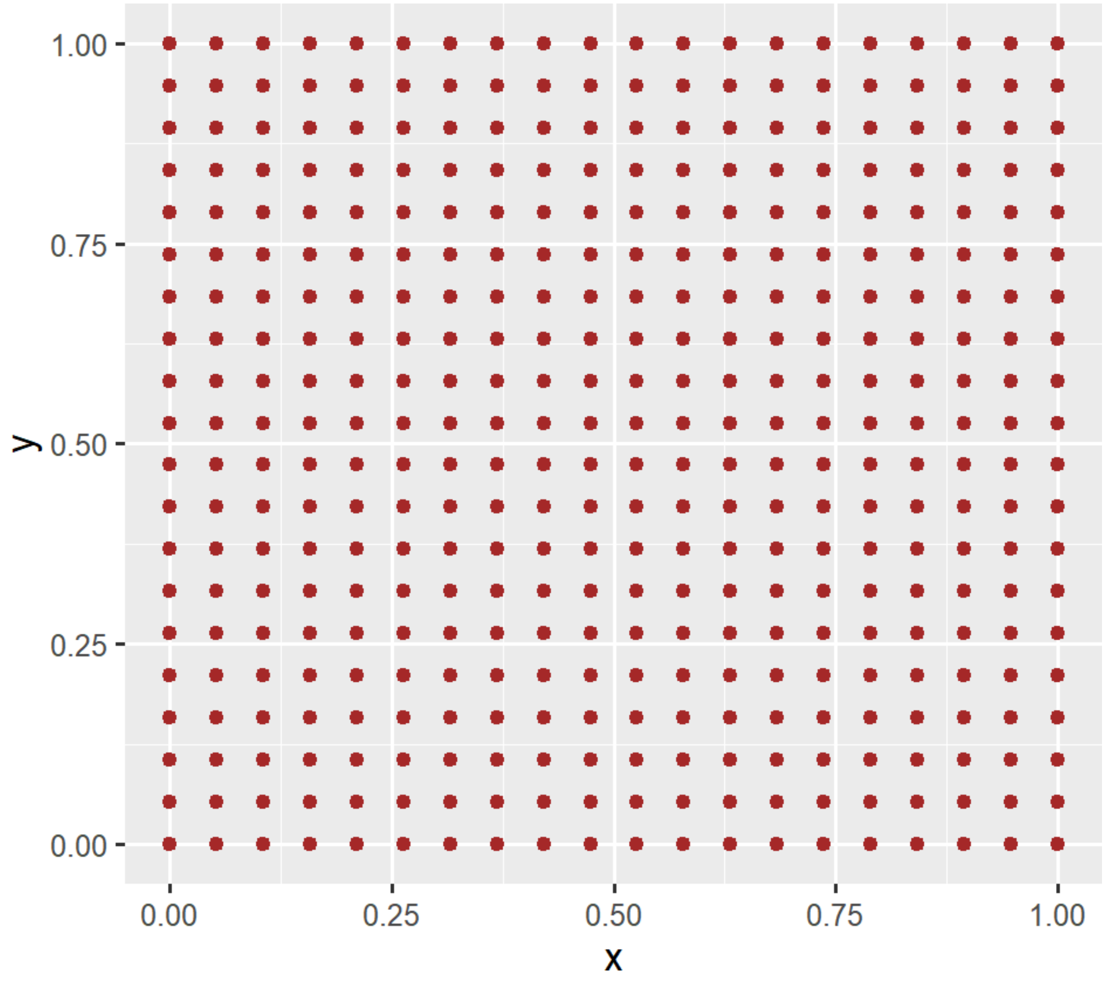
Consider the following regular grid of points:

```
> grid.x = 20
> grid.y = 20
> pred.grid = expand.grid(
+             x = seq(0, 1, length.out = grid.x),
+             y = seq(0, 1, length.out = grid.y))
```

It is necessary to define a new projection matrix:

```
> A.pred6 = inla.spde.make.A(mesh = mesh6,
+                 loc = as.matrix(pred.grid))
> 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.predictor=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")]
```

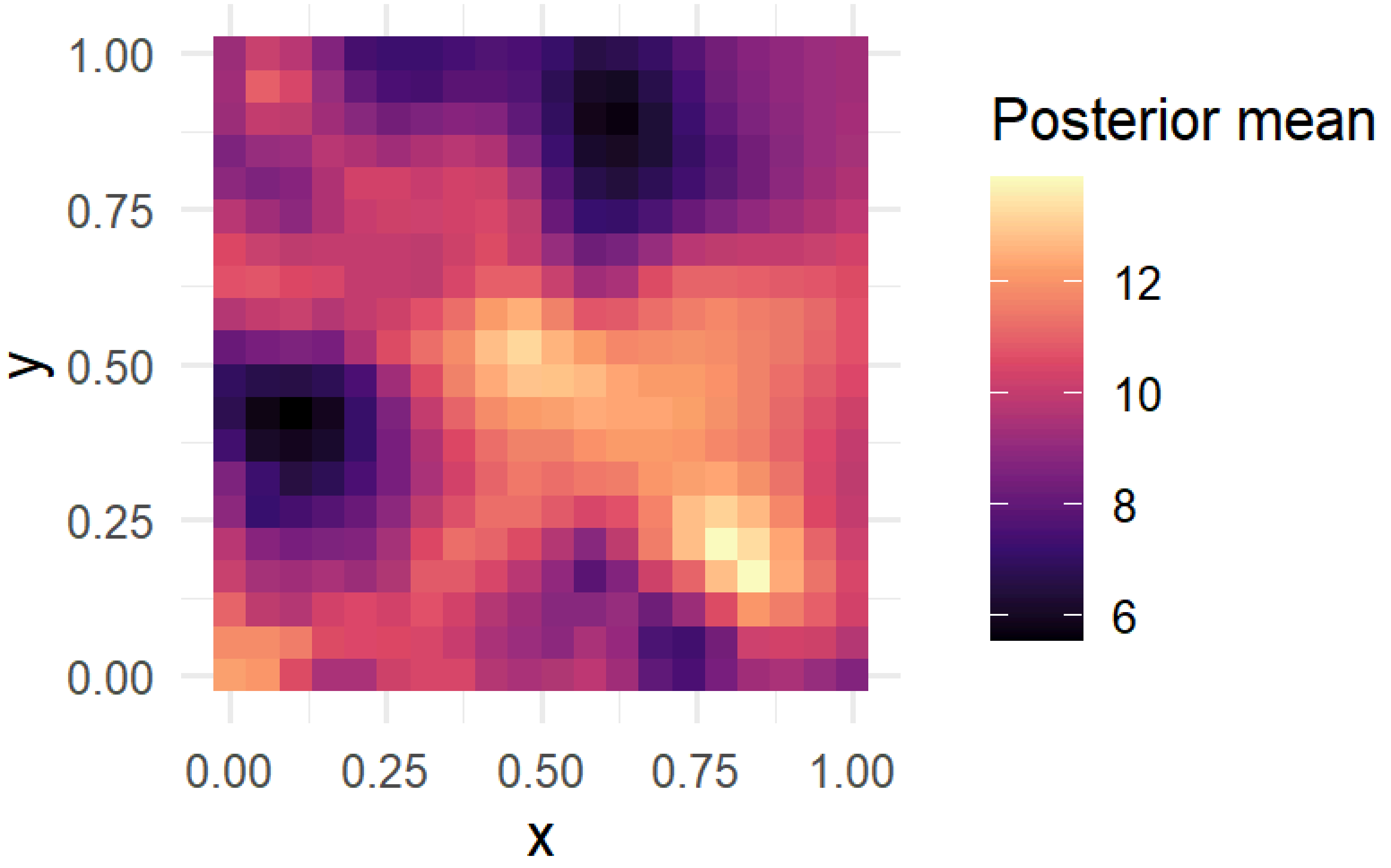
```
                   mean        sd
APredictor.201 12.24876 0.2614732
APredictor.202 12.06026 0.4238819
APredictor.203 10.58893 0.4336448
```

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

Mapping the linear predictor: posterior mean

```
> # extract posterior mean and standard deviation of the predictions
> grid.df <- data.frame(
+      x = pred.grid[, "x"],
+      y = pred.grid[, "y"],
+      mean = output6pred$summary.linear.predictor[index.pred, "mean"],
+      sd   = output6pred$summary.linear.predictor[index.pred, "sd"]
+  )
>
> library(ggplot2)
> library(viridis)
>
> ggplot(grid.df, aes(x = x, y = y, fill = mean)) +
+   geom_tile() +
+   coord_fixed() +
+   scale_fill_viridis_c(option = "magma") +
+   theme_minimal() +
+   labs(
+     fill = "Posterior mean",
+     title = "Posterior mean of linear predictor"
+   )
```

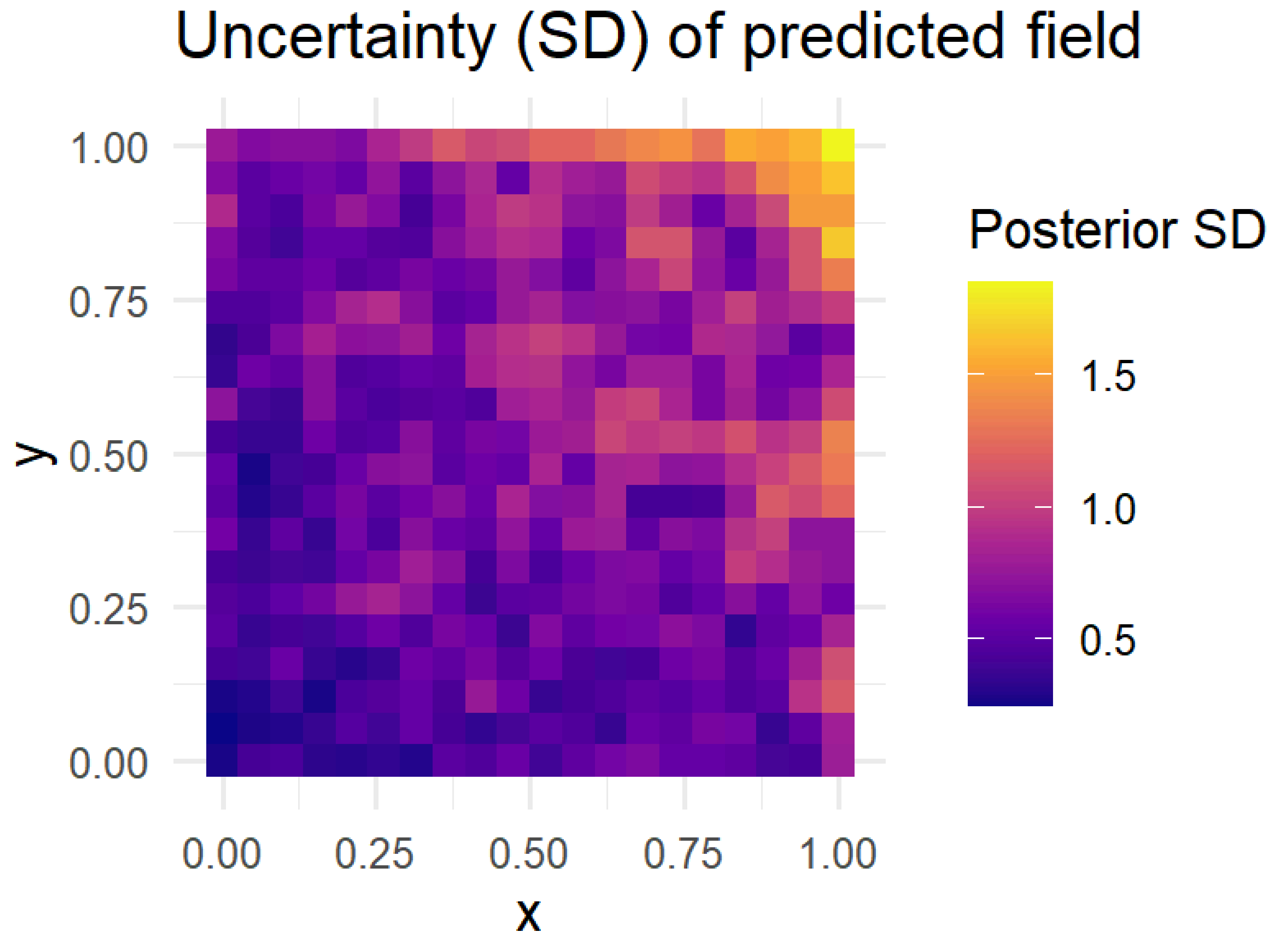
Posterior mean of linear predictor



Mapping the linear predictor: posterior standard deviation

Now we plot now the posterior standard deviation of the linear predictor at the grid level

```
> ggplot(grid.df, aes(x = x, y = y, fill = sd)) +
+   geom_tile() +
+   coord_fixed() +
+   scale_fill_viridis_c(option = "plasma") +
+   theme_minimal() +
+   labs(
+     fill = "Posterior SD",
+     title = "Uncertainty (SD) of predicted field"
+   )
```


Uncertainty (SD) of predicted field
Posterior SD
1.5
1.0
0.5
1.00
0.75
0.50
0.25
0.00
y
x

Wrap-up: SPDE modelling pipeline in R-INLA (step-by-step)

- Load, among the others, the R packages: `INLA`, `fmesher`, `inlabru`
- **Construct the mesh** to obtain a triangulation of the domain using the function `inla.mesh.2d`. Recently, a new function has been made available from the package `fmesher`: `fm_mesh_2d_inla`; you can check the documentation typing `?fm_mesh_2d_inla` in R
- **Construct the SPDE model** to define the spatial dependence structure using the function `inla.spde2.matern` or `inla.spde2.pcmatern`
- **Construct the index set** (it creates identifiers for the latent spatial field components) to connect the GMRF components with the stack using the function `inla.spde.make.index`
- **Construct the projection matrix** to map the GMRF (on mesh nodes) to observation or prediction locations using the function `inla.spde.make.A`
- **Put everything together in a stack object** using the function `inla.stack`
- **Run the model!**

References

Banerjee, S., B. P. Carlin, and A. E. Gelfand (2014). *Hierarchical modeling and analysis for spatial data*. Chapman and Hall/CRC.

Lindgren, F. and H. Rue (2015). "Bayesian Spatial Modelling with R-INLA". In: *Journal of Statistical Software* 63.19, pp. 1-25.

Lindgren, F., H. Rue, and J. Lindström (2011). "An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach (with discussion)". In: *J. R. Statist. Soc. B* 73.4, pp. 423-498.

Simpson, D., H. Rue, A. Riebler, et al. (2017). "Penalising Model Component Complexity: A Principled, Practical Approach to Constructing Priors". In: *Statistical Science* 32.1, pp. 1-28.