The R-INLA tutorial: SPDE models

Warning: work in progress...

Suggestions are welcome to `elias@r-inla.org`

Elias T. Krainski

March 28, 2013

Abstract

In this tutorial we presents how we fit models to spatial point refered data, the called geostatistical model, using INLA and SPDE. Before a fast introduction on such models, we start with a toy example to show details. In forward examples we presents some funcionalities to fit more complex models, such as non-Gaussian, models with two likelihoods, ...

Contents

Chapter 1

Introduction

If we have data measured on some locations, by locations we mean some system of coordinate reference where are each data from, so we have a point refereed data set. The point refereed data is very common in many areas of science. These type of data appear on mining, climate modeling, ecology, agriculture and other areas. If we want do model that data incorporating the reference about where are each data from, we want do use a model to point refereed data.

1.1 The spatial dependence

It is possible that we build a regression model considering each coordinates as covariate. But in some cases it is necessary a very complicated function based on coordinates to get a adequate description of the mean. For example, any complex non-linear function or a non-parametric function. If these type of model are only to incorporate some trend on the mean based on the coordinates. Also, this type of model is a fixed effect model.

But, we want a model to measure the first geography law in a simple way. This law says: "Everything is related to everything else, but near things are more related than distant things", [Tobler, 1970]. So, we need a model to incorporate the property that a observation is more correlated with a observation collected on a neighbour local, than another that is collected on more distant local. One option to modelled this dependence is the use of a random effect spatially structured. This type of model is a model to spatial dependency, different of the spatial trend. But, it is possible to include both terms in a model. Also, the models for spatial dependency are used within more general models as random effects, the spatially structured random effects.

In spatial statistics we have models to incorporate the spatial dependency when the locations are areas (states, cities, etc.) and when the locations are points. In the last case, it is also possible that such locations are fixed or are random. The models to point refereed data with a spatially structured random effect is commonly called a geostatistical models. There are a specific area of spatial statistics that study these models, the geostatistics. See [Cressie, 1993] for a good introduction on the spatial statistics.

1.2 The Gaussian field

To fix the notation, let s any location on the study area and $X(s)$ is the random effect at this location. We have $X(s)$ a stochastic process, with $s \in \boldsymbol{D}$, were $\boldsymbol{D}$ is the domain area of the locations and $\boldsymbol{D} \in \Re^d$. Suppose, for example, that we have $\boldsymbol{D}$ any country and we have any data measured on geographical locations, $d = 2$, within this country.

Suppose that we assume that we have a realization of $x(s_i)$, $i = 1, 2, ..., n$, a realization of $X(s)$ in n locations. It is common assumed that $x(s)$ has a multivariate Gaussian distribution. Also, if we assume that $X(s)$ continuous over space, we have a continuously

indexed Gaussian field (GF). It is because we suppose that it is possible that we get data in any location within the study region. To complete the specification of the distribution of $x(s)$, is necessary to defines it mean and covariance.

A very simple option, is the definition of a correlation function based only on euclidean distance between locations. This assume that if we have two pairs of points separated same distance h, both pairs have same correlation. Also, is intuitive to choose any function decreasing with h. There is some work about the GF and correlation functions in [Abrahamsen, 1997].

A very popular correlation function is the Matérn correlation function, that depends on a scale parameter $\kappa > 0$ and a smoothness parameter $\nu > 0$. Considering two locations s_i and s_j, the stationary and isotropic Matérn correlation function is:

$$Cor_M(X(s_i), X(s_j)) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\kappa \parallel s_i - s_j \parallel)^{\nu} K_{\nu}(\kappa \parallel s_i - s_j \parallel) \tag{1.1}$$

where $\parallel . \parallel$ denotes the euclidean distance and K_{ν} is the modified Bessel function of second kind order. Also, we defines the Matérn covariance function by $\sigma_x Cor(X(s_i), X(s_j))$, where σ_x is the marginal variance of the process.

If we have a realization $x(s)$ on n locations, we write the joint correlation, or joint covariance, matrix Σ making each entry $\Sigma_{i,j} = \sigma_x Cor_M(X(s_i), X(s_j))$. It is common to assume that $X(x)$ has a zero mean. So, we have completely defined a multivariate distribution to $x(s)$.

Now, suppose now that we have a data y_i observed at locations s_i, $i = 1, ..., n$. If we suppose that we have an underlie GF that generate these data, we are going to fit the parameters of this process, making the identity $y(s_i) = x(s_i)$, and $y(s_i)$ is just a realization of the GF. In this case, we the likelihood function is just the multivariate distribution with mean μ_x and covariance Σ. If we assume $\mu_x = \beta_0$, we have four parameters to estimate.

In many situations we assume that we have an underlie GF but we no observe it and observe a data with a measurement error, i. e., $y(s_i) = x(s_i) + e_i$. Additionally, it is common to assume that e_i independent of e_j for all $i \neq j$ and $e_i \sim N(0, \sigma_e)$. These additional parameter, σ_e, measures the noise effect, called nugget effect. In this case the covariance of marginal distribution of $y(s)$ is $\sigma_e^2 I + \Sigma$. This model is a short extension of the basic GF model, and in this case, we have one additional parameter to estimate. To look more about this model see [Diggle and Ribeiro Jr, 2007].

It is possible to describe this model within a larger class of models, the hierarchical models. Suppose that we have observations y_i on locations s_i, $i = 1, ..., n$. We start with

$$\begin{aligned} y_i|\theta, \beta, x_i, F_i &\sim P(y_i|\mu_i, \phi) \\ \boldsymbol{x} &\sim GF(0, \Sigma) \end{aligned} \tag{1.2}$$

where $\mu_i = h(F_i^T\beta + x_i)$, F is a matrix of covariates, x is the random effects, θ are parameters of random effects, β are covariate coefficients, $h()$ is a function mapping the linear predictor $F_i^T\beta + x_i$ to $E(y_i) = \mu_i$ and ϕ is a dispersion parameter of the distribution, in the exponential family, assumed to y_i. To write the GF with nugget effect on this class, we replace $F_i^T\beta$ by β_0, consider the Gaussian distribution to y_i, with variance σ_e^2 and x as GF.

We have many extensions of this basic hierarchical model. Now, we stop these extensions and return on some extensions in later sections. But, if we know the properties of the GF, we are able to study all the practical models that contain, or are based on, this random effect.

It is mentioned that the data, or the random effect, on a finite number of n points that we have observed data is considered a realization of a multivariate Gaussian distribution. But, to evaluate the likelihood function, or the random effect distribution, we need to make computations of the multivariate Gaussian density. So, we have, in the log scale,

the expression

$$-\frac{n}{2}log(2\pi) - |\Sigma| - \frac{1}{2}(x(s) - \mu_x)^T\Sigma^{-1}(x(s) - \mu_x) \tag{1.3}$$

where Σ is a dense $n \times n$. To compute this, we need a factorization of this matrix. Because this matrix is a dense, this is a operation of order $O(n^3)$, so is one 'big n problem'.

An alternative used in some software's to do geostatistical analysis, is the use of the empirical variogram to fit the parameters of the correlation function. This option don't use any likelihood for the data and the multivariate Gaussian distribution to the random effects. A good description of these techniques is made on [Cressie, 1993].

However, it is adequate to assume any likelihood for the data and a GF for the spatial dependence, the model based approach on geostatistics, [Diggle and Ribeiro Jr, 2007]. So, in some times we need the use the multivariate Gaussian distribution to the random effects. But, if the dimension of the GF is big, it is impractical to make model based inference.

In another area of the spatial statistics, the analysis of areal data, there is models specified by conditional distributions that implies a joint distribution with a sparse precision matrix. These models are called the Gaussian Markov random fields (GMRF), [Rue and Held, 2005]. So, the inference when we use GMRF is more easy to do than we use the GF, because to work with two dimensional GMRF models, we have cost of $O(n^{3/2})$ on the computations with its precision matrix. So, it is more easy to make analysis with big 'n'.

1.3 The SPDE approach

In the literature there is some ideas to fit GF by an approximation of the GF to any GMRF. The very good alternative found is the use of the stochastic partial differential equation approach (SPDE), [Lindgren et al., 2011]. This approach got a explicit link between GF to GMRF.

The SPDE approach is based on two main results. The first one extends the result obtained by [Besag, 1981]. This result is to approximate a GF with generalized covariance function, obtained when $\nu \to 0$ in the Matérn correlation function. This approximation, considering a regular two-dimensional lattice with number of sites tending to infinite, is that the full conditional have

$$E(x_{ij}|x_{-ij}) = \frac{1}{a}(x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}) \tag{1.4}$$

and $Var(x_{ij}|x_{-ij}) = 1/a$ for $|a| > 4$. In the representation using precision matrix, we have, for one single site, just the upper right quadrant and with a as the central element, that

$$\begin{matrix} -1 & \\ a & -1 \end{matrix} \tag{1.5}$$

Considering a GF $x(\boldsymbol{u})$ with the Matérn covariance is a solution to the linear fractional SPDE

$$(\kappa^2 - \Delta)^{\alpha/2}x(\boldsymbol{u}) = \boldsymbol{W}(\boldsymbol{u}), \quad \boldsymbol{u} \in \Re^d, \quad \alpha = \nu + d/2, \ \ \kappa > 0, \ \ \nu > 0, \tag{1.6}$$

[Lindgren et al., 2011] show that for $\nu = 1$ and $\nu = 2$ the GMRF representations are convolutions of (1.5). So, for $\nu = 1$, in that representation we have:

$$\begin{matrix} & 1 & \\ -2a & & 2 \\ 4 + a^2 & -2a & 1 \end{matrix} \tag{1.7}$$

and, for $\nu = 2$:

$$\begin{matrix} & -1 & & \\ 3a & & -3 & \\ -3(a^2 + 3) & & 6a & 3 \\ a(a^2 + 12) & -3(a^2 + 3) & 3a & -1 \end{matrix} \tag{1.8}$$

This is an intuitive result, because if we have larger ν on the Matérn correlation function of the GF, we need more non zero neighbours sites in the GMRF representation. Remember that it is the smoothness parameter, so if the process is more smooth, we need a more larger neighbourhood on the GMRF representation.

If the spatial locations are on a irregular grid, it is necessary the use of the second result on [Lindgren et al., 2011]. To extend the first result, a suggestion is the use of the finite element method (FEM) for a interpolation of the locations of observations to the nearest grid point. To do it, suppose that the $\Re^2$ is subdivided into a set of non-intersecting triangles, where any two triangles meet in at most a common edge or corner. The three corners of a triangle are named *vertices*. The suggestion is to start with the location of the observed points and add some triangles (heuristically) with restriction to maximize the allowed edge length and minimize the allowed angles. The the approximation is

$$x(\boldsymbol{u}) = \sum_{k=1}^{n} \psi_k(\boldsymbol{u}) w_k$$

for some chosen basis functions ψ_k, Gaussian distributed weights w_k and n the number of vertices on the triangulation. If the functions ψ_k are piecewise linear in each triangle, ψ_k is 1 at vertices k and 0 at all other vertices.

The second result is obtained using the $n \times n$ matrices $\boldsymbol{C}$, $\boldsymbol{G}$ and $\boldsymbol{K}$ with entries

$$C_{i,j} = \langle \psi_i, \psi_j \rangle, \quad G_{i,j} = \langle \nabla\psi_i, \nabla\psi_j \rangle, \quad (\boldsymbol{K}_{\kappa^2})_{i,j} = \kappa^2 C_{i,j} + G_{i,j} \tag{1.9}$$

to get the precision matrix $\boldsymbol{Q}_{\alpha,\kappa}$ as a function of κ^2 and α:

$$\begin{aligned} \boldsymbol{Q}_{1,\kappa^2} &= \boldsymbol{K}_{\kappa^2}, \\ \boldsymbol{Q}_{2,\kappa^2} &= \boldsymbol{K}_{\kappa^2}\boldsymbol{C}^{-1}\boldsymbol{K}_{\kappa^2}, \\ \boldsymbol{Q}_{\alpha,\kappa^2} &= \boldsymbol{K}_{\kappa^2}\boldsymbol{C}^{-1}\boldsymbol{Q}_{\alpha-2,\kappa^2}\boldsymbol{C}^{-1}\boldsymbol{K}_{\kappa^2}, \quad \text{for } \alpha = 3, 4, \ldots . \end{aligned} \tag{1.10}$$

Here we have too the notion that if ν increases, we need a more dense precision matrix.

The $\boldsymbol{Q}$ precision matrix is generalized for a fractional values of α (or ν) using a Taylor approximation, see the author's discussion response in [Lindgren et al., 2011]. From this approximation, we have the polynomial of order $p = \lceil \alpha \rceil$ for the precision matrix

$$\boldsymbol{Q} = \sum_{i=0}^{p} b_i \boldsymbol{C}(\boldsymbol{C}^{-1}\boldsymbol{G})^i. \tag{1.11}$$

For $\alpha = 1$ and $\alpha = 2$ we have the (1.10). Because, for $\alpha = 1$, we have $b_0 = \kappa^2$ and $b_1 = 1$, and for $\alpha = 2$, we have $b_0 = \kappa^4$, $b_1 = \alpha\kappa^4$ and $b_2 = 1$. For fractional $\alpha = 1/2$ $b_0 = 3\kappa/4$ and $b_1 = \kappa^{-1}3/8$. And, for $\alpha = 3/2$, ($\nu = 0.5$, the exponential case), $b_0 = 15\kappa^3/16$, $b_1 = 15\kappa/8$, $b_2 = 15\kappa^{-1}/128$. Using these results combined with recursive construction, for $\alpha > 2$, we have GMRF approximations for all positive integers and half-integers. To start, we look at the model and fitting process by view of parametrization used in a software commonly used to Analise the point refereed data in **R**, the **geoR** package. But, before it, we show how we simulate a dataset.

1.4 Simulation of a data set

We remember here that one realization of a GF is just one realization of a multivariate Gaussian distribution with an appropriate covariance matrix. To specify these matrix, we need a set of locations and the matrix of distance between each point with all others. Based on this matrix $n \times n$ of distances, we compute the covariance matrix and do one simulation of a multivariate Gaussian distribution.

Suppose that we have a set of $n = 100$ locations, on a square with area one with bottom left and top right limits: (0,0) and (1,1). We choose these locations with density in left bottom corner higher than top right corner. The **R** code to do it is:

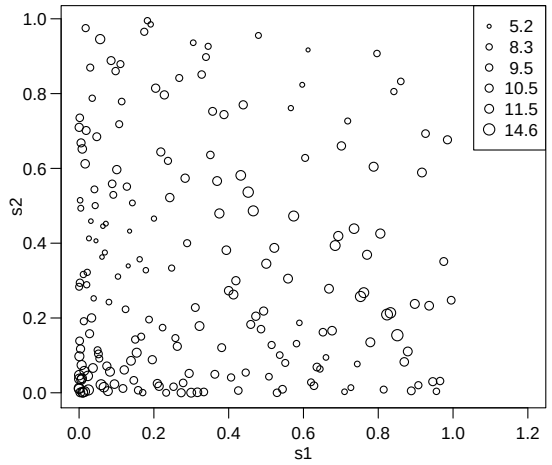

Figure 1.1: Visualization of the simulated data

```
> n <- 200;  set.seed(123)
> pts <- cbind(s1=sample(1:n/n-0.5/n)^2, s2=sample(1:n/n-0.5/n)^2)
```

and for get the (lower triangle) matrix of distances we do

```
> dmat <- dist(pts)
```

and for the Matérn covariance we need the parameters: σ_x^2, κ and ν. Additionally, we need the mean β_0 and the nugget parameter σ_e^2. We declare an values to such parameters by

```
> beta0 <- 10; sigma2e <- 0.3; sigma2x <- 5; kappa <- 7; nu <- 1
```

We, first, consider y as mean β_0 and that covariance $\sigma_e^2 I + \Sigma$. And, we get the covariance by

```
> mcor <- as.matrix(2^(1-nu)*(kappa*dmat)^nu*
+                   besselK(dmat*kappa,nu)/gamma(nu))
> diag(mcor) <- 1;   mcov <- sigma2e*diag(n) + sigma2x*mcor
```

Now we going to simulate one realization of a geostatistical model. First, we do the simulation of one realization of the multivariate Gaussian distribution. Here we remember that if we want to get $y \sim N(\mu, \Sigma)$ from $z \sim N(0, I)$, we do $y = \mu + zL$, where L is a matrix that $L^T L = \Sigma$. So, we use the Cholesky factorization of the covariance matrix, because if L is the Cholesky of the Σ, so $L^T L = \Sigma$. In **R** we use:

```
> L <- chol(mcov);   set.seed(234);   y1 <- beta0 + drop(rnorm(n)%*%L)
```

We show this simulated data in a graph of the locations with size of points proportional of the simulated values on Figure 1.1 with code below

```
> par(mar=c(3,3,1,1), mgp=c(1.7, 0.7, 0), las=1)
> plot(pts, asp=1, xlim=c(0,1.2), cex=y1/10)
> q <- quantile(y1, 0:5/5)
> legend('topright', format(q, dig=2), pch=1, pt.cex=q/10)
```

1.5 Maximum likelihood estimation

In this section we get the maximum likelihood estimation of the parameters of the model used to simulate the data on the last section. We know that is adequate the use of the partial derivatives with respect to each parameter and the Fisher information matrix to use the Fisher scoring algorithm. But we take derivatives only with respect to β and, for the others parameters we naively use one of the quasi-Newton methods, without using derivatives. In [Diggle and Ribeiro Jr, 2007] it have more detail to estimate the other parameters. It is easy to get

$$(\boldsymbol{F}^{'}\Sigma^{-1}\boldsymbol{F})\hat{\beta} = \boldsymbol{F}^{'}\boldsymbol{F}$$

So, we want to write a likelihood function as a function of the unknown parameters and the data.

One detail is that we use the matrix of distances between the points because it is not necessary to recalculate it at each interaction of the algorithm. Another is that the function defined return the negative of the logarithm of the likelihood function, because, by default, the algorithm find the minimum of a function. A bit of comment is that in the calculus of the likelihood, we take vantages of the symmetry property of the covariance matrix to invert it.

So, we have the function defined bellow as function of four parameters (σ_e^2, σ_x^2, κ and ν) and the data

```
> nllf <- function(pars, ff, data, distmat) {
+   m <- 2^(1-pars[4])*(pars[3]*distmat)^pars[4]*
+     besselK(distmat*pars[3],pars[4])/gamma(pars[4])
+   diag(m) <- 1;   m <- pars[1]*diag(length(data)) + pars[2]*m
+   L <- chol(m);   m <- chol2inv(L)
+   beta <- solve(crossprod(ff, m)%*%ff, crossprod(ff, m)%*%data)
+   z <- data-ff%*%beta;   ss <- drop(crossprod(z,m)%*%z)
+   return(length(data)*log(2*pi)/2 +sum(log(diag(L))) +0.5*ss)
+ }
```

the first argument of this function is the vector of the five parameters of the model, such as β_0, σ_e^2, σ_s^2, κ and ν.

For test, we calculates the likelihood at true values of the parameters and at another set of the values

```
> c(nllf(c(sigma2e, sigma2x, kappa, nu), matrix(1,n), y1, as.matrix(dmat)),
+   nllf(c(0, sigma2x, kappa, nu), matrix(1,n), y1, as.matrix(dmat)))
```

```
[1] 282.1201 811.4252
```

We get the maximum likelihood estimates with 'L-BFGS-B' method implemented on `optim()` function. Also, using `hessian=TRUE`, we got the approximate Hessian matrix to use in asymptotic inference. We got the maximum likelihood estimates

```
> (ores <- optim(c(sigma2e, sigma2x, kappa, nu), nllf, hessian=TRUE,
+                ff=matrix(1,n), data=y1, distmat=as.matrix(dmat),
+                method='L-BFGS-B', lower=rep(1e-5,4)))$par
```

```
[1] 0.2824531 3.2739337 9.4016876 1.0864144
```

and the squared root of the hessian diagonal

```
> (se <- sqrt(diag(solve(ores$hessian))))
```

```
[1] 0.08000842 0.93752841 4.08361796 0.42725613
```

This naive solution by likelihood is to show how it works. But, in the **geoR** package, we have functions to get simulations and to do estimation. The difference is that the **geoR** package uses $\phi = 1/kappa$ for the scale parameter and it calls kappa for the smoothness parameter. The `grf()` function can be used to get samples of the geostatistical model from many correlation functions. To get exactly the same data, we use

```
> require(geoR);   set.seed(234)
> grf1 <- grf(grid=pts, cov.pars=c(sigma2x, 1/kappa), mean=beta0,
+             nugget=sigma2e, kappa=nu, messages=FALSE)
```

Also, we have the `likfit()` function to perform the maximum likelihood estimation. This function for solve of the estimation of such five parameters don't uses a optimization algorithm in five dimensions. Because it is possible to get close forms expressions to β_0, σ_e^2 and for σ_s^2, [Diggle and Ribeiro Jr, 2007]. So this function works better than the 'naive' solution presented here. We got the maximum likelihood estimates by

```
> (g1res <- likfit(grf1, ini=c(sigma2x, 1/kappa), messages=FALSE,
+                  nugget=sigma2e, kappa=nu, fix.kappa=FALSE))

likfit: estimated model parameters:
    beta    tausq  sigmasq      phi    kappa
"9.5457" "0.2824" "3.2741" "0.1064" "1.0862"
Practical Range with cor=0.05 for asymptotic range: 0.4403836

likfit: maximised log-likelihood = -281.1
```

the `tausq` is σ_e^2 and `kappa` is the smoothness parameter.

But, for comparison with INLA results, we fix the smoothness parameter on the true value used to simulate the data, and find the maximum likelihood estimates to another parameters by

```
> (fit.l <- likfit(grf1, ini.cov.pars=c(sigma2x, 1/kappa),
+                  nugget=sigma2e, kappa=1, messages=FALSE))

likfit: estimated model parameters:
    beta    tausq  sigmasq      phi
"9.5349" "0.2709" "3.3234" "0.1156"
Practical Range with cor=0.05 for asymptotic range: 0.4620667

likfit: maximised log-likelihood = -281.1
```

Notice that we also estimate the mean parameter β, the parameter of fixed effect part of the model. Also, we note that the likelihood has similar value, probably because the estimated value of ν is similar to its true value.

Chapter 2

The SPDE approach with INLA: a toy example

In this section we use a very simple data set to show how we fit a geostatistical model using the SPDE approach, [Lindgren et al., 2011]. We use Bayesian inference and found the posterior distributions by the Integrated Nested Laplace Approximation - INLA, [Rue et al., 2009]. We show use of **INLA R** package to make this approach. The ideas for application of the SPDE approach with **INLA**, are well described on [Lindgren, 2012] and [Cameletti et al., 2012].

Suppose that we have a tree column `data.frame` or `matrix` with the two first are the coordinates and the third is the response collected at this locations.

```
> str(toy <- read.csv2("data/toydat.csv"))
```

```
'data.frame':        200 obs. of  3 variables:
 $ s1: num  0.0827 0.6123 0.162 0.7526 0.851 ...
 $ s2: num  0.0564 0.9168 0.357 0.2576 0.1541 ...
 $ y : num  11.52 5.28 6.9 13.18 14.6 ...
```

We consider the n observations y_i on locations the s_i, $i = 1, ..., n$, and we define the model

$$\begin{aligned} y_i|\beta_0, x_i, \sigma_e^2 &\sim N(\beta_0 + x_i, \sigma_e^2) \\ \boldsymbol{x} &\sim GF(0, \Sigma) \end{aligned} \tag{2.1}$$

We consider that x is a realization of a Gaussian Field, with Matérn correlation function parametrized by the smoothness parameter ν and the scale κ, such the parametrization in [Lindgren et al., 2011].

With this toy example we show with details how we make a good triangulation, prepare the data, fit the model, extract the results from output and make predictions on locations where we don't have observed the response. In this section we use the default priors for all the parameters.

2.1 The triangulation

The first step to fit the model is the construction of the 'mesh'. The `inla.mesh.create.helper()` function creates the Constrained Refined Delaunay Triangulation (CRDT) and we call mesh. There are a several options on is function:

```
> args(inla.mesh.create.helper)
```

```
function (points = NULL, points.domain = NULL, offset = c(-0.05,
    -0.15), n = c(8, 16), boundary = NULL, interior = NULL, max.edge,
    min.angle = c(21, 21), cutoff = 0, plot.delay = NULL)
NULL
```

where two of these arguments must be provided.

One is about the locations or the region where the mesh will be made. If the `points` is provided, the triangles vertices includes the locations, if `cutoff=0`. If `cuttof=a`, the points with distance less than a are replaced by a single vertex. If the `points.domain` is provided and `points` is not, the vertices are found to cover the domain using the restrictions on other arguments. If both are provided, both restrictions are combined. In another more specific cases, the `boundary` and `interior` arguments also can be used.

The another mandatory argument is the `max.edge`. This argument specifies the maximum allowed triangle edge lengths in the inner domain and in the outer extension. So, this argument depends of the distance measure on that the coordinates are projected. To understand how this function works, we apply to first five locations varying some of these arguments.

Firstly, we defines the domain

```
> pl01 <- matrix(c(0,1,1,0,0, 0,0,1,1,0), ncol=2)
```

and create some triangulations with code bellow:

```
> m1 <- inla.mesh.create.helper(as.matrix(toy[1:5,1:2]), max.edge=c(1,1))
> m2 <- inla.mesh.create.helper(as.matrix(toy[1:5,1:2]), max.edge=c(.3,1))
> m3 <- inla.mesh.create.helper(, pl01, max.edge=c(1,1))
> m4 <- inla.mesh.create.helper(as.matrix(toy[1:5,1:2]), pl01, max.edge=c(.3,1))
> m5 <- inla.mesh.create.helper(as.matrix(toy[1:5,1:2]), pl01,
+                                max.edge=c(.3,1), min.angle=30)
> m6 <- inla.mesh.create.helper(, pl01, max.edge=c(.3,.5), offset=c(0.15, 0.3))
```

We visualize these outputs on the Figure 2.1, with the code below

```
> par(mfrow=c(2, 3), mar=c(0,0,0,0))
> for (i in 1:6) {
+   plot(toy[1:5,1:2], xlim=c(-.4,1.4), asp=1, pch=19, axes=FALSE, col=2)
+   plot(get(paste('m', i, sep='')), add=TRUE)
+   points(toy[1:5, 1:2], pch=19, col=2); lines(pl01, col=4)
+ }
```

We see on Figure 2.1 that we have triangles out of the domain area. The reason for this is to avoid the boundary effect, [Lindgren, 2012]. A triangulation without additional border is made with the `inla.mesh.create()` function with `refine = FALSE`.

The graph on top left of this Figure shows the mesh created when the input is the points and the max length of vertices are 1. In this mesh we have additional points because we also have the restriction on the angles size. The mesh on top mid have max length of internal vertices 0.3. This mesh has same shape but two more vertices than the first one. The mesh on top right is created only with the domain input and max length of vertices are 1. So, the points are not in the set of vertices, and the vertices cover the domain area. Also, the shape of the triangles created on this way are expected to be more regular.

In the mesh at bottom left is made when both points and points domain are provided and with max length of internal vertices are 0.3. The mesh at mid bottom graph is similar to left bottom but with angles of triangles greater than 30. The default minimum angle is 21, because algorithm is guaranteed to converge for 'min.angle' at most 21.

The mesh at right bottom is made with domain and without points, with length of internal edges less than 0.3 and with offset 0.1 and 0.3. The default offset is -0.05 and -0.15. When it argument is negative, is interpreted as a factor relative to the approximate data diameter. If we inform two positive values, they are interpreted on the unit of the distance scale of the coordinates. The extension is calculated in respect to points if no domain is provided and in respect of the domain if it is provided.

The object returned by `inla.mesh.create.helper()` function has class 'inla.mesh' and contains a set of things:

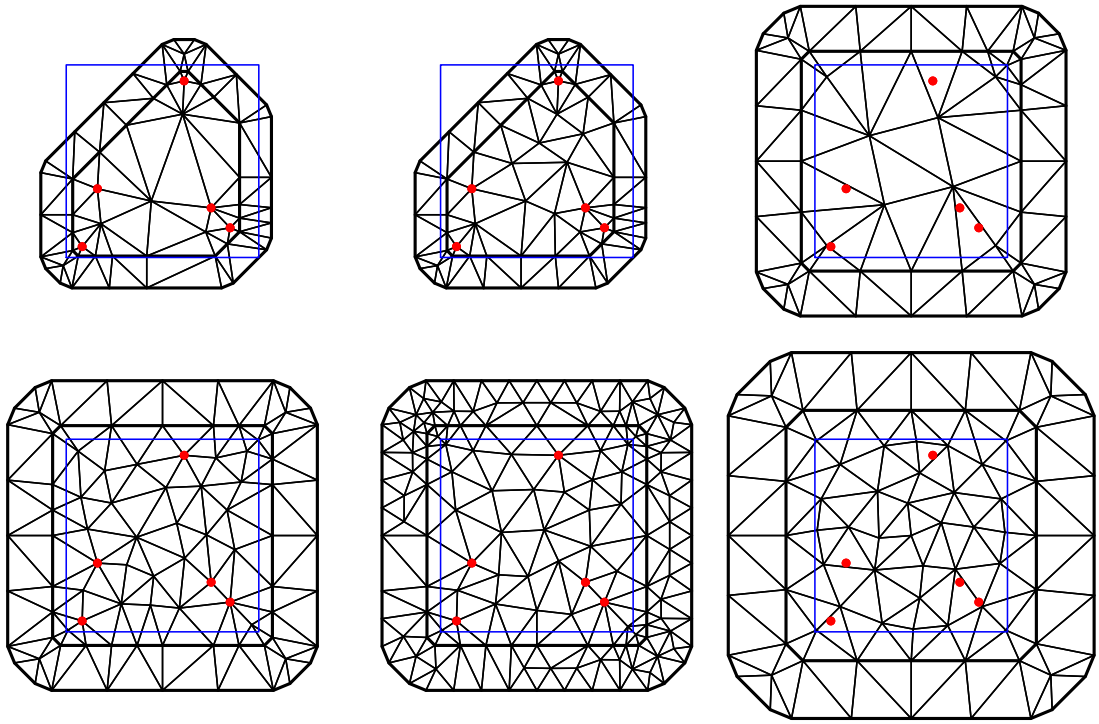

Figure 2.1: Triangulation with different restrictions.

```
> class(m1)

[1] "inla.mesh"

> names(m1)

[1] "meta"     "manifold" "n"        "loc"      "graph"    "segm"
[7] "idx"
```

The number of vertices on each mesh is

```
> c(m1$n, m2$n, m3$n, m4$n, m5$n, m6$n)

[1]  60  67  63  91 159  85
```

The 'graph' element represents the CRDT obtained. More, on 'graph' element we have the a matrix that represents the graph of neighborhood structure. For example, for `m1` we have 'A'

```
> dim(m1$graph$vv)

[1] 60 60
```

The vertices that correspond the location points are identified on 'idx' element

```
> m1$idx$loc

[1] 24 25 26 27 28
```

To analyze the toy data set, we use six triangulations options to make comparisons on section 2.3. The first and sixth mesh were made only using the points with two different restrictions on the maximum edges length. The another were made using only the points domain, but, we have different maximum edges length restriction. The third mesh has same maximum length edges than the first one. Also, the second has same maximum length edges than the sixth.

We did this sixth mesh with the code bellow

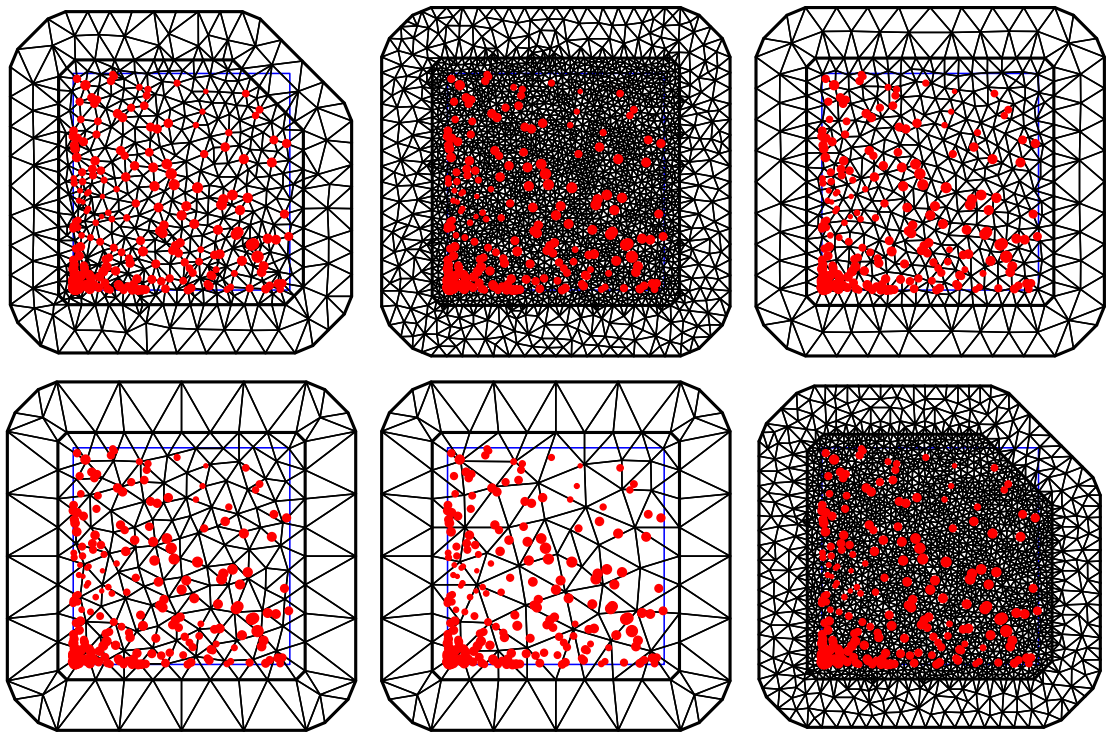

Figure 2.2: Six triangulation options for the toy example. The red points are the data locations and its size are proportional to response value.

```
> mesh1 <- inla.mesh.create.helper(as.matrix(toy[,1:2]),
+                                  max.edge=c(0.1, 0.2))
> mesh2 <- inla.mesh.create.helper(, pl01, max.e=c(0.04, 0.1))
> mesh3 <- inla.mesh.create.helper(, pl01, max.e=c(0.1, 0.2))
> mesh4 <- inla.mesh.create.helper(, pl01, max.e=c(0.17, 0.35))
> mesh5 <- inla.mesh.create.helper(, pl01, max.e=c(0.25, 0.5))
> mesh6 <- inla.mesh.create.helper(as.matrix(toy[,1:2]),
+                                  max.edge=c(0.04, 0.1))
> c(mesh1$n, mesh2$n, mesh3$n, mesh4$n, mesh5$n, mesh6$n)

[1]  593 2461  436  168  106 2314
```

We have small number of vertices on third mesh than the first one due the pattern of the data locations. We have a relative large number of vertices on second mesh and we have less triangles than data on the fifth and sixth. They are showed on Figure 2.2 with code below

```
> par(mfrow=c(2,3), mar=c(0,0,0,0))
> for (i in 1:6) {
+   plot(pl01, col=4, xlim=c(-.3,1.3), asp=1, axes=FALSE, type='l')
+   plot(get(paste('mesh',i,sep='')), add=T)
+   points(toy[,1:2], pch=19, col=2, cex=toy[,3]/10)
+ }
```

2.2 The estimation

After the mesh is made, we will define the SPDE model based on it. We use the function `inla.spde2.matern()` to define the SPDE model. The principal arguments are the mesh

object and the α parameter, related to the smoothness parameter of the process. This data were simulated with $\alpha = 2$. We use the three first mesh made on previous section to illustrate some details on the estimation process on next sub-sections.

2.2.1 Simple estimation with mesh with points together

In this section, we use the mesh created with location points. To define the SPDE model we use the `inla.spde2.matern()` function

```
> spde1 <- inla.spde2.matern(mesh1, alpha=2)
```

We get the posterior marginal distributions of the parameters using the main function `inla()` of the **INLA** package

```
> res1 <- inla(y ~ f(i, model=spde1),
+              control.predictor=list(compute=TRUE),
+              data=data.frame(y=toy$y, i=1:nrow(toy)))
```

On the object returned by `inla()` function we have summaries, marginal posterior densities of each parameter on the model and some other results. So, we have the posterior marginal distribution, and summaries, of β_0, σ_e^2, σ_x^2, κ and for each element of $X(s_i)$. We want to explore more of these output elements later.

For example, a summary of the posterior marginal distribution of β parameter we get by:

```
> res1$summary.fix
```

```
                mean        sd 0.025quant 0.5quant 0.975quant          kld
(Intercept) 9.875267 0.2880549   9.257399 9.872619   10.50889 5.423419e-31
```

The summary of $1/\sigma_e^2$ is obtained by

```
> res1$summary.hy[1,]
```

```
      mean         sd 0.025quant   0.5quant 0.975quant
0.29547461 0.03598577 0.23384162 0.29219407 0.37470557
```

and to get a summary of σ_e, the square root of the σ_e^2, we do a transformation on the posterior marginal distribution of $1/\sigma_e^2$ and we got the marginal expectation by

```
> inla.emarginal(function(x) sqrt(1/x),
+                res1$marginals.hy$'Precision for the Gaussian obs')
```

```
[1] 1.849698
```

Also, we have the summary of posterior marginal distribution of the $log(\kappa)$ with

```
> res1$summary.hy[3,]
```

```
      mean         sd 0.025quant   0.5quant 0.975quant
  2.561572   1.411563  -0.126077   2.526000   5.414056
```

and to get the marginal expectation of the κ we do

```
> inla.emarginal(function(x) exp(x), res1$marginals.hyperpar$Theta2)
```

```
[1] 36.94199
```

To get the marginal variance of x, we want to get an additional procedure because it is function of two parameters κ and ν.

```
> res1.field <- inla.spde2.result(res1, 'i', spde1, do.transf=TRUE)
> inla.emarginal(function(x) x, res1.field$marginals.variance.nominal[[1]])
```

```
[1] 0.1837687
```

with this additional procedure, we also have the marginal posterior of the empirically range corresponding to correlations near 0.1. We get the posterior mean of it by

```
> inla.emarginal(function(x) x, res1.field$marginals.range.nominal[[1]])
```

```
[1] 0.5418659
```

2.2.2 When locations are not vertices of triangulation

Remember that we have constructed six mesh on section 2.1. The first one is used on previous section to fit the model. On this mesh, we have an index vector mapping the vertices to the points

```
> str(mesh1$idx$loc)
```

```
 int [1:200] 68 69 70 71 72 73 74 75 76 77 ...
```

On a mesh constructed using only the points domain, without using the points locations, we don't have this index. Also, when we use `cutoff` greater than zero. In this case, we don't have an explicit index between points locations and the mesh triangles. To fit the model, we need to use an appropriate specification of the linear predictor, see [Lindgren, 2012].

In this section, we describe the estimation on these case, but on the sub-section 2.2.3 we show the stack functionality, a general approach to build more complex models. And, this functionality is especially useful if we have covariates on the model.

Let the matrix $\boldsymbol{A}$ that links the response to the triangles on the mesh, the projector matrix. Here, we define the η^* as

$$\eta^* = \boldsymbol{A}(\boldsymbol{x} + 1\beta_0)$$

We define the projector matrix with

```
> dim(A2 <- inla.spde.make.A(mesh2, loc=as.matrix(toy[,1:2])))
```

```
[1]  200 2461
```

Because each point is inside one triangle, each row of the projector matrix has three non-zero elements

```
> table(rowSums(A2>0))
```

```
  3
200
```

Also, we have columns in the projector matrix that don't have non-zero elements, because the correspondent vertice don't have point bellow

```
> table(colSums(A2)>0)
```

```
FALSE  TRUE
 2004   457
```

So, we have some redundant triangles and the stack functionality automatically handles this situation.

To build the model, we want to define the data list taking into account that we have this not square projector matrix. We must be put into list of data the response and the index for the random field, but the index vector has length equal the number of columns of projector matrix. Additionally, we must be pass the projector matrix on `control.predictor` argument. But, when any matrix is used on `control.predictor`, the intercept must be removed from the formula. So, if we want to have the intercept estimate, we must be pass a vector of ones as a covariate.

We fit the model with

```
> spde2 <- inla.spde2.matern(mesh2)
> res2 <- inla(resp ~ 0 +  m + f(i, model=spde2),
+              data=list(resp=toy$y, i=1:mesh2$n, m=rep(1,mesh2$n)),
+              control.predictor=list(A=A2, compute=TRUE))
```

and we have

```
> res2$summary.fix

      mean        sd 0.025quant 0.5quant 0.975quant kld
m 9.520132 0.7161512   8.019181 9.535821   10.93313   0

> res2$summary.hy[1,]

      mean         sd 0.025quant   0.5quant 0.975quant
 3.4619977  0.6647869  2.3327387  3.4027473  4.9352415
```

and we look that the precision parameter has different posterior mean than previous model fitted. We return on this in section 2.3.

2.2.3 The stack functionality

The stack functionality is useful to works with SPDE on **INLA** package. This is a general form to implement an model with SPDE component on **INLA**. This functionality is adequate to avoid errors in the index construction, addressed on previous sub-section, [Lindgren, 2012]. The `inla.stack()` function is designed to be applied on many types of models using the SPDE model. An example is the application of spatio temporal models in [Cameletti et al., 2012].

In the previous section we needed a caution on the adequate index construction. Here, we show that using the `inla.stack()` the indexes are automatically provided. To show it, we fit the model with the third mesh constructed on section 2.1.

Firstly, we define the projector matrix based on this mesh

```
> A3 <- inla.spde.make.A(mesh3, loc=as.matrix(toy[,1:2]))
```

and the the indexes set provided

```
> ind3 <- inla.spde.make.index(name='i', n.mesh=mesh3$n)
```

This function is designed to works also when we have replications of the process, see [Lindgren, 2012].

The `inla.stack()` function stacks the data, indexes and covariates in the adequate form to build the model later. This function is used firstly to arrange the response, the predictor matrix and the effects

```
> st3.dat <- inla.stack(data=list(resp=toy$y), A=list(A3),
+                       effects=list(c(ind3, list(m=1))), tag='est')
```

and we use the `inla.stack.data()` to get the data, on the adequate order, to fit the model. This function automatically eliminates the elements when any column of the predictor matrix have zero sum. In this example we have

```
> table(colSums(A3)>0)

FALSE  TRUE
  209   227

> str(inla.stack.data(st3.dat))

List of 5
 $ resp   : num [1:200] 11.52 5.28 6.9 13.18 14.6 ...
 $ i      : int [1:227] 1 28 42 46 47 51 53 69 70 71 ...
 $ i.group: int [1:227] 1 1 1 1 1 1 1 1 1 1 ...
 $ i.repl : int [1:227] 1 1 1 1 1 1 1 1 1 1 ...
 $ m      : num [1:227] 1 1 1 1 1 1 1 1 1 1 ...
```

Also, we use the `inla.stack.A()` to extract the simplified predictor matrix

```
> dim(inla.stack.A(st3.dat))

[1] 200 227
```

To fit the model we use

```
> spde3 <- inla.spde2.matern(mesh3)
> res3 <- inla(resp ~ 0 + m + f(i, model=spde3),
+              data=inla.stack.data(st3.dat),
+              control.predictor=list(A=inla.stack.A(st3.dat), compute=TRUE))
```

2.3 Comparing different triangulations

In this section we compare six models fitted with the six different mesh made on section 2.1. The estimation process of three of this models are showed on past section. To do this comparison, we just plot the posterior marginal distributions of the model parameters. So, we show here the extraction of the marginal posterior distribution of the parameters. We evaluate these models by the addiction of the true values used on the simulation of the toy dataset. Also, we add the maximum likelihood estimates.

The true values are: $\beta_0 = 10$, $\sigma_e^2 = 0.3$, $\sigma_x^2 = 5$, $\kappa = 7$ and $\nu = 1$. The ν parameter is fixed on the true value when we define $\alpha = 2$ on definition of the SPDE model.

```
> beta0 <- 10; sigma2e <- 0.3; sigma2x <- 5; kappa <- 7; nu <- 1
```

and the maximum likelihood estimates are

```
> lk.est

[1] 9.5348771 0.2708695 3.3234253 8.6536164
```

In the code bellow we extract, respectively for β_0, σ_e^2, σ_x^2 and κ, the marginal posterior distributions, and add a vertical dashed line on the true value. We show in this graph results from the models fitted using the six meshes.

```
> par(mfrow=c(2,2), mar=c(2.5,2.5,1,.5), mgp=c(1.5,.5,0), las=1)
> plot(res1$marginals.fix[[1]], type='l',
+      xlab=expression(beta[0]), ylab='', xlim=c(7,15))
> for (i in 2:6)
+   lines(get(paste('res', i, sep=''))$marginals.fix[[1]], col=i)
> abline(v=beta0, lty=2); abline(v=lk.est[1], lty=3)
> plot.default(inla.tmarginal(function(x) 1/x, res1$marginals.hy[[1]]),
+              type='l', xlim=c(0,4), ylim=c(0,7),
+              xlab=expression(sigma[e]^2), ylab='')
> for (i in 2:6)
+   lines(inla.tmarginal(function(x) 1/x,
+                        get(paste('res', i, sep=''))$marginals.hy[[1]]), col=i)
> abline(v=sigma2e, lty=2); abline(v=lk.est[2], lty=3)
> plot.default(res1.field$marginals.variance.nominal[[1]], type='l',
+              xlim=c(0,10), ylim=c(0,3), xlab=expression(sigma[x]^2), ylab='')
> for (i in 2:6)
+   lines(get(paste('res', i, '.field',sep=''))$marginals.variance.n[[1]], col=i)
> abline(v=sigma2x, lty=2); abline(v=lk.est[3], lty=3)
> plot.default(inla.tmarginal(function(x) exp(x),
+                             res1$marginals.hyperpar$Theta2), type='l',
+              xlim=c(0,25), ylim=c(0,.25), xlab=expression(kappa), ylab='')
> for (i in 2:6)
+   lines(inla.tmarginal(function(x) exp(x),
+                        get(paste('res', i, sep=''))$marginals.hy$Theta2), col=i)
> abline(v=kappa, lty=2);  abline(v=lk.est[4], lty=3)
> legend('topright', c(paste('mesh', 1:6), 'True', 'Likelihood'),
+        lty=c(rep(1,6), 2, 3), col=c(1:6,1,1), bty='n')
```

At the Figure 2.3 we see that the better results is obtained when we use the meshes build without the points locations. The main conclusion here is made considering that the meshes that not contain the location points have bad triangulation. Because there aren't regularly shaped triangles.

The design of the points location is this example is bad if we want to use the points the construction of the mesh with `cotoff=0`, and we have a set of triangles with irregular shape. The word here is to use a mesh with regular size of triangles.

The second word is in respect of the process range. The difference is on the meshes found using only the points domain are the maximum length of the inner edges. On the last mesh, this value is 0.3. But, the marginal range, using the results from second or third mesh, is near these length. It explains the fact of the model fitted using the last mesh have not very well result for σ_e^2. So, we need a mesh with adequate size of the triangles, in relation to the practical range of the process.

A third word is in respect of the computational aspect. Both second and third mesh proved to be suitable. But, the mesh with less number of edges is preferred due the computationally advantage on work.

2.4 Prediction of the random field

A very common objective when we have spatial data collected on some locations is the prediction to another locations that the data are not observed. In this section we show two approaches to make prediction of the random field, one is after the estimation process and other is jointly on estimation process. To compare both approaches, we predict the random field on three target locations: (0.1,0.1), (0.5,0.5), (0.9,0.9).

```
> pts3 <- rbind(c(.1,.1), c(.5,.5), c(.9,.9))
```

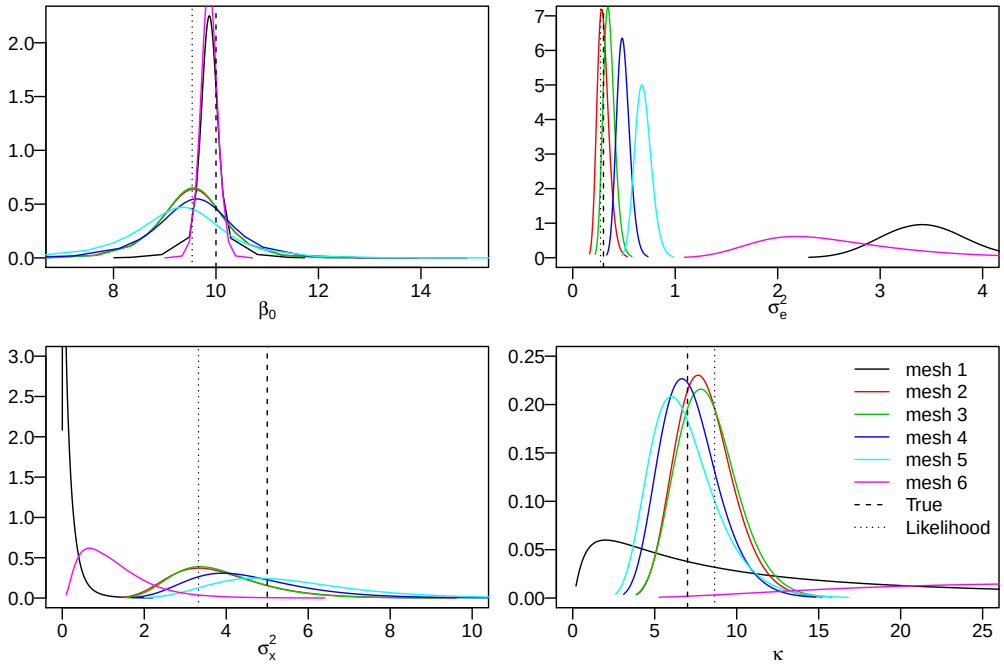

Figure 2.3: Marginal posterior distribution for β_0 (top left), σ_e^2 (top right), σ_x^2 (bottom left) and κ (bottom right).

2.4.1 Jointly with the estimation process

The prediction of the random field joint the parameter estimation process in Bayesian inference is the most common. In this case, we just compute the marginal posterior distribution of the random field in target locations. If the target points are on the mesh, so we have automatically this distribution. If the target points are not on the mesh, we must define the projector matrix for the target points. We will use the model with the third mesh and we have

```
> dim(A3pts3 <- inla.spde.make.A(mesh3, loc=pts3))
```

```
[1]   3 436
```

Now, we just need to defines a stack for the prediction. This just contains the data as NA, the predictor matrix and its corresponding effects

```
> stk3prd <- inla.stack(data=list(resp=NA), A=list(A3pts3),
+                       effects=list(ind3), tag='prd3')
```

Also, we join the stack of data and the stack of predictor

```
> st3.full <- inla.stack(st3.dat, stk3prd)
```

Now, we fit the model again with the full stack

```
> res3b <- inla(resp ~ 0 + m + f(i, model=spde3),
+               data=inla.stack.data(st3.full),
+               control.predictor=list(A=inla.stack.A(st3.full), compute=TRUE))
```

We need to find the index on predicted values that corresponds the predicted random field on the target locations. We extract the index from the full stack using the adequate 'tag'.

```
> indd3b <- inla.stack.index(st3.full, tag='prd3')$data
```

and get the summary of the posterior distributions of the random field on the target locations by

```
> res3b$summary.linear.pred[indd3b,]
```

```
                     mean       sd 0.025quant   0.5quant 0.975quant
Apredictor.201  0.5047583 0.786741  -1.025679  0.4896919   2.128530
Apredictor.202  3.3740318 1.074836   1.274173  3.3654663   5.525271
Apredictor.203 -0.5777399 1.199268  -2.937856 -0.5813870   1.804929
                        kld
Apredictor.201 0.000000e+00
Apredictor.202 0.000000e+00
Apredictor.203 1.929826e-17
```

Also we have the marginal distributions. A marginal distribution on `inla()` output is just two vector, one represents the parameter values and another the density. We extract the marginals posterior distributions with

```
> marg3 <- res3b$marginals.linear[indd3b]
```

and get the 95% HPD interval for the second by

```
> inla.hpdmarginal(0.95, marg3[[2]])
```

```
                low     high
level:0.95 1.253284 5.501168
```

and see that around the point (0.5,0.5) the random field has positive values, see Figure 2.4.

2.4.2 After the estimation process

The prediction after the estimation process is usually just a matrix operation. Let $f(x)$ a functional of interest (the posterior mean for example). We have $f(\hat{x}_m)$ the estimated functional on the mesh vertices, We take $f(\hat{x}_m)$ from the output of `inla()` function.

In the SPDE approach, we use the projector matrix of the basis functions on the target locations. Let $P^{(m)}$ a matrix that evaluate the basis functions from set of locations of the mesh to set of target points. The estimated functional of interest is the product $P^{(m)} f(\hat{x}_m)$.

We have this projector matrix from previous section, from the third mesh. A summary of the random field on vertices of the mesh are stored on the element i (the name coded for the spatial random effect) of the `summary.random` element on `inla()` output. So, we 'project' it on the target locations by

```
> drop(A3pts3%*%res3$summary.random$i$mean)
```

```
[1]  0.5042745  3.3737890 -0.5784691
```

or using the `inla.mesh.projector()` function

```
> inla.mesh.project(inla.mesh.projector(mesh3, loc=pts3),
+                   res3$summary.random$i$mean)
```

```
[1]  0.5042745  3.3737890 -0.5784691
```

and for the standard deviation

```
> (drop((A3pts3)%*%(res3$summary.random$i$sd)))
```

```
[1] 0.8990971 1.1553657 1.4463271
```

```
> sqrt(drop((A3pts3^2)%*%(res3$summary.random$i$sd^2)))
```

```
[1] 0.5550365 0.9755613 0.9502450
```

and see that for the mean we have similar values than those on previous section. For the standard deviation we have a little difference.

This approach is good to get the map of the random field and its standard error with a low computational cost. To do it we project the posterior mean and the posterior standard error on a grid. The `inla.mesh.projector()` function get the projector matrix automatically for a grid of points over a square that contains the mesh.

```
> pgrid0 <- inla.mesh.projector(mesh3)
```

Because we need a grid without the borders of the mesh, we construct the coordinates with

```
> grid <- expand.grid(seq(0, 1, length=101),
+                     seq(0, 1, length=101))
> pgrid <- inla.mesh.projector(mesh3, loc=as.matrix(grid))
```

and project the posterior mean and the posterior standard deviation on the both grid with

```
> prd0.m <- inla.mesh.project(pgrid0,  res3b$summary.ran$i$m)
> prd0.s <- inla.mesh.project(pgrid0,  res3b$summary.ran$i$s)
> prd.m <- inla.mesh.project(pgrid,  res3b$summary.ran$i$m)
> prd.s <- inla.mesh.project(pgrid,  res3b$summary.ran$i$s)
```

We visualize this values projected on the grid on Figure 2.4.

2.5 Prediction of the response

Now, we want to predict the response on a set of non observed locations. In similar way that on past section, it is possible to find the marginal distribution or to make a projection of some functional of the response.

2.5.1 By sum of linear predictor components

Here we presents a naive procedure to predict the response. This consists on the sum of the predictor components.

In this toy example we just sum the posterior mean of the intercept to the posterior mean of the random field to get the posterior mean of the response. Using the previous results we have

```
> res3$summary.fix[1,1] + drop(A3pts3%*%res3$summary.random$i$mean)
```

```
[1] 10.029357 12.898871  8.946613
```

If there are covariates, the prediction also can be made in similar way, see .

2.5.2 By the posterior distribution

In this case, we want to define a adequate predictor of the response and build the model again. Using the third mesh, we defines a new stack to predict the response. This is similar to the stack to predict the random field, but here we add the intercept on the list of predictor matrix and on the list of effects

```
> stk3presp <- inla.stack(data=list(resp=NA), A=list(A3pts3,1),
+                         effects=list(ind3, m=rep(1,3)), tag='prd3resp')
```

and join with the data stack to build the model again

```
> st3.presp <- inla.stack(st3.dat, stk3presp)
> res3r <- inla(resp ~ 0 + m + f(i, model=spde3),
+              data=inla.stack.data(st3.presp),
+              control.predictor=list(A=inla.stack.A(st3.presp), compute=TRUE))
```

We find the index of the predictor that corresponds the predicted values of the response on the target locations. We extract the index from the full stack by

```
> indd3r <- inla.stack.index(st3.presp, 'prd3resp')$data
```

To get the summary of the posterior distributions of the response on target locations we do

```
> res3r$summary.fitted.values[indd3r,]
```

```
                           mean        sd 0.025quant  0.5quant 0.975quant
fitted.Apredictor.201 10.028825 0.3575207   9.322228 10.030491   10.72629
fitted.Apredictor.202 12.899081 0.8221512  11.281276 12.899104   14.51760
fitted.Apredictor.203  8.947253 1.0575678   6.872903  8.945826   11.03065
```

Also, we extract the marginals posterior distributions with

```
> marg3r <- res3r$marginals.fitted.values[indd3r]
```

and get the 95% HPD interval for the second by

```
> inla.hpdmarginal(0.95, marg3r[[2]])
```

```
               low     high
level:0.95 11.2776 14.51255
```

and see that around the point (0.5,0.5) we have the values of the response significantly larger than β_0, see Figure 2.4.

Now, suppose that we want to predict the response on a large number of locations, for example on a grid. So, the computation of all marginal posterior distributions is computationally expensive. But, we usually not uses the marginal distributions. We usually uses just the mean and standard deviation. So, we don't need the storage of all the marginal distributions. Also, we don't need the quantiles of the marginal distributions.

On the code below, we build the model again but we disable the storage of the marginal posterior distributions to random effects and to posterior predictor values. Also, we disable the computation of the quantiles. We want to storage only the mean and standard deviation of the response on a grid. We uses the same grid of the previous section.

```
> stkgrid <- inla.stack(data=list(resp=NA), A=list(pgrid$proj$A,1),
+                       effects=list(ind3, m=rep(1,101*101)), tag='prd3rg')
> stk.all <- inla.stack(st3.dat, stkgrid)
> res3g <- inla(resp ~ 0 + m + f(i, model=spde3),
```

```
+                data=inla.stack.data(stk.all),
+                control.predictor=list(A=inla.stack.A(stk.all),
+                  compute=TRUE), quantiles=NULL,
+                control.results=list(return.marginals.random=FALSE,
+                  return.marginals.predictor=FALSE))
> round(head(res3g$summary.fitt, 3),4)

                           mean     sd
fitted.Apredictor.00001 11.0150 0.2592
fitted.Apredictor.00002  5.6768 0.5404
fitted.Apredictor.00003  6.3804 0.3926

> object.size(res3g$marginals.random)

0 bytes
```

We get the indexes

```
> indd3g <- inla.stack.index(stk.all, 'prd3rg')$data
```

and use it to visualize, together the predictions of the random field on previous section, on Figure 2.4 with the commands bellow

```
> require(gridExtra)
> grid.arrange(levelplot(prd0.m, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)),
+              levelplot(matrix(prd.m, 101), xlab='', ylab='',
+                        col.regions=topo.colors(99), scales=list(draw=FALSE)),
+              levelplot(matrix(res3g$summary.fitt[indd3g,1], 101),
+                        xlab='', ylab='',
+                        col.regions=topo.colors(99), scales=list(draw=FALSE)),
+              levelplot(prd0.s, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)),
+              levelplot(matrix(prd.s, 101), xlab='', ylab='',
+                        col.regions=topo.colors(99), scales=list(draw=FALSE)),
+              levelplot(matrix(res3g$summary.fitt[indd3g,2], 101),
+                        xlab='', ylab='',
+                        col.regions=topo.colors(99), scales=list(draw=FALSE)),
+              nrow=2)
```

We see on the top right graph of Figure 2.4 that on border of the mesh domain we have the larger standard errors. Also, on bottom left we see that at this borders we have, except of the bottom left corner, the values around the mean of the process. We see that we have a significantly spatial dependence. Because we have a variation from -4 to 4 on the spatial effect and we have standard deviations around 0.8 to 1.6 on most part of the domain region. So we have a range of length 8, on the random field and on the response surface, and the standard deviation is less than the half of this range in both cases.

Another thing is that the standard deviation of the response is has less values on region near the corner (0, 0) and greater on the corner (1,1) due the density of the locations. Also, we have, on corner (0,0), less values to standard deviation of the response than to standard deviation of the random field.

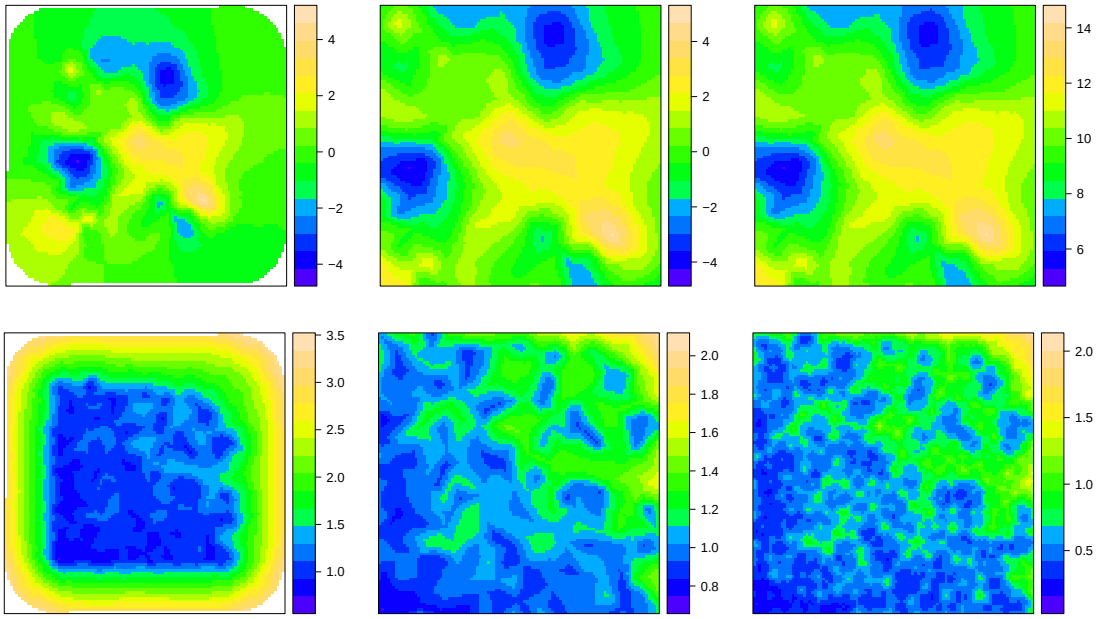

Figure 2.4: The mean and standard deviation of the random field (top left and bottom left) on all mesh domain, the same on study region domain (top mid and bottom mid) and the mean and standard variation of the response (top right and bottom right)

Chapter 3

Non-Gaussian response: Precipitation on Paraná

A very common data in spatial statistics is the climate data. On this section we analyze a data set from Water National Agency in Brazil, in Portuguese it is *Agencia Nacional de Águas* - ANA. The ANA collect data on many locations over Brazil. All these data are freely available from the ANA website.

3.1 The data set

It have data on more one thousand of locations within Paraná, a state at south of Brazil. We have daily rainfall data on first 90 days of the year 2011 on 612 locations, including stations within the Paraná state and around its border.

We read the data and the border of Paraná state with

```
> prec <- read.table('data/rainfallParana.txt', header=TRUE)
> pr.bord <- read.table('data/borderParana.txt', header=TRUE)
```

We have the coordinates on first two columns, altitude, and more 90 columns, one for each day of January to march, with daily accumulated precipitation. We show bellow some data of four stations: the with missing altitude with less latitude, the stations with extremes longitudes and the station with greater altitude.

```
> prec[ii <- c(which(is.na(prec$A))[which.min(prec$S[is.na(prec$A)])],
+              which(prec$E%in%range(prec$E)), which.max(prec$A)), 1:10]
```

	East	South	Altitude	jan01	jan02	jan03	jan04	jan05	jan06	jan07
534	705.9321	7102.748	NA	20.5	7.9	8.0	0.8	0.1	15.6	31.0
302	780.7440	7222.958	9	18.1	8.1	2.3	11.3	23.6	0.0	22.6
606	150.0059	7163.822	231	43.8	0.2	4.6	0.4	0.0	0.0	0.0
386	448.1731	7153.772	1446	0.0	14.7	0.0	0.0	28.1	2.5	26.6

We visualize this four stations in red points on the graph at right of Figure 3.1.

We have some problems on this data set. For example, we have a spatio-temporal data. Also, there are seven stations with missing altitude and missing data on daily rainfall. The red stations with missing altitude is the red points on graph at left of Figure 3.1. It's easy to get the altitude only from coordinates information. For example, using an digital elevation model, google earth or another on line information. For example, the station with missing altitude located at south and out of border of Paraná state is located on a coast city. So in this case it is reasonable to assign a small value. But, it is possible to build a stochastic model for altitude and predict these missing values.

However, the principal problem is that we have 47 stations with missing data and that 10 stations on 45 or more days. This problem is treated on next section. In this section,

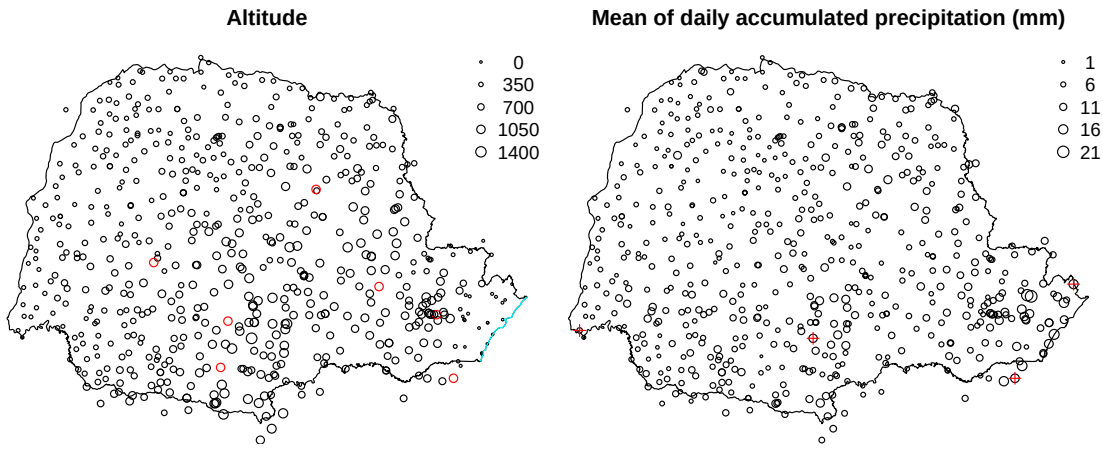

Figure 3.1: Locations of Paraná stations, altitude and average of daily accumulated precipitation (mm) on January.

we analyze the mean of daily accumulated precipitation on January of 2012. We compute this covariate with

```
> prec$precMean <- rowMeans(prec[,3+1:31], na.rm=TRUE)
```

We visualize the locations on Figure 3.1, with the commands bellow

```
> par(mfrow=c(1,2), mar=c(0,0,2,0))
> plot(pr.bord, type='l', asp=1, axes=FALSE, main='Altitude')
> points(prec[1:2], col=is.na(prec$Alt)+1,
+        cex=ifelse(is.na(prec$Alt), 1, .3+prec$Alt/1500))
> legend('topright', format(0:4*350), bty='n', pch=1, pt.cex=.3+0:4*35/150)
> lines(pr.bord[1034:1078, ], col='cyan')
> plot(pr.bord, type='l', asp=1, axes=FALSE,
+      main=paste('Mean of daily accumulated precipitation (mm)'))
> points(prec[1:2], cex=0.3+prec$precMean/20)
> legend('topright', format(seq(1,21,5)),
+        bty='n', pch=1, pt.cex=0.3+seq(1,21,5)/20)
> points(prec[ii, 1:2], pch=3, col=2)
```

The size of the points on left graph are proportional to altitude of the locations. The cyan line in this graph is the Paraná boundary with the Atlantic Ocean. So, we have height altitudes at mid and south, low altitudes near the sea and the altitude decrease at north and west of Paraná state. On the right graph, the points sizer are proportional to average of daily accumulated precipitation. We see that near the coast we have large values.

3.2 The model and covariate selection

In this section, we analise the average of daily accumulated precipitation on that period. The average of daily precipitation must be positive. So, we consider in this section the model

$$\begin{aligned} y_i|F_i, \beta, x_i, \theta &\sim Gamma(a_i, b_i) \\ log(\mu_i) &= F_i^T\beta + x_i \\ x_i &\sim GF(0, \Sigma) \end{aligned}$$

where, F_i is a vector of covariates (the location coordinates and altitude) as covariates, with the vector of coefficients β, and a latent Gaussian random field x, with its covariance matrix function of parameters ν, κ and σ_x^2. Also, with the Gamma likelihood, we consider

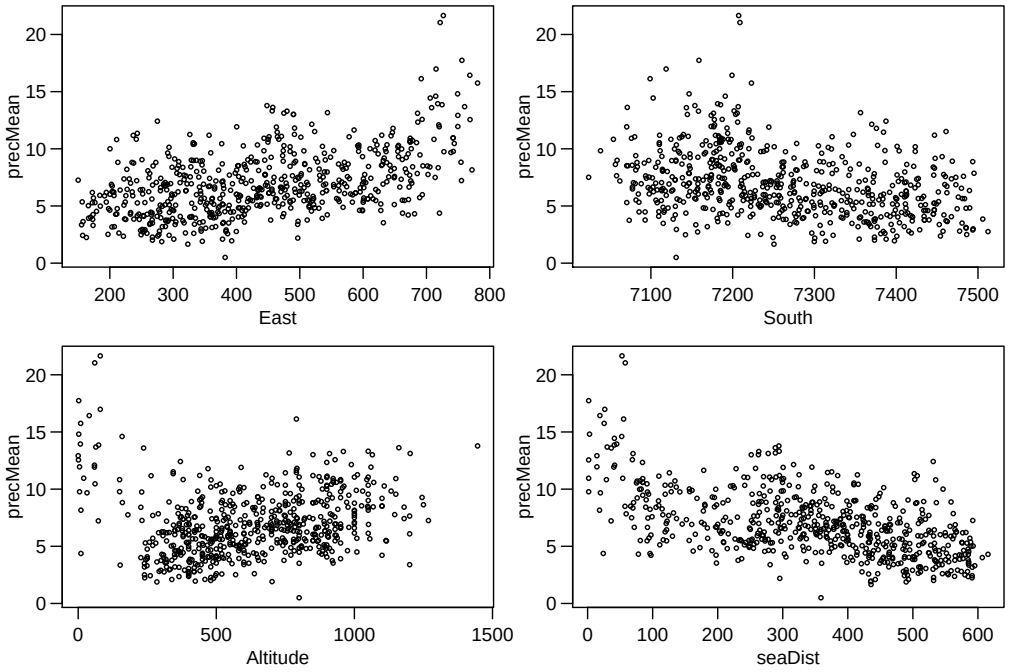

Figure 3.2: Dispersion plots of average of daily accumulated precipitation mean by Longitude (top left), Latitude (top right), Altitude (bottom left) and distance to sea (bottom right).

that $E(y_i) = \mu_i = a_i/b_i$ and $V(y_i) = a_i/b_i^2 = \mu_i^2/\phi$, where ϕ us a precision parameter and we define a linear predictor to $\log(\mu_i)$.

To make an initial exploration of relationship between the precipitation and the covariates, we visualize some dispersion diagrams. After preliminary tests, we see that is more adequate the construction of a new covariate: the distance from each station to Atlantic Ocean. We found the coordinates of Paraná state border that share frontier with the sea (plotted as cyan line on left graph at Figure 3.1) and computes the distance from each station to the neighbor coordinate of this line. We compute these distances using the `nndistF()` function provided by **splancs** package, [Rowlingson et al., 2012].

```
> prec$"seaDist" <- nndistF(as.matrix(pr.bord[1034:1078,]),
+                           as.matrix(prec[,1:2]))
```

We see the dispersion plots at Figure 3.2.

```
> par(mfrow=c(2,2), mar=c(3,3,0.5,0.5), mgp=c(1.7,.7,0), las=1)
> for (i in c(1:3, ncol(prec)))  plot(prec[c(i,ncol(prec)-1)], cex=.5)
```

With the Figure 3.2, we conclude that have a not well defined non-linear relationship with Longitude, and there is a similar, but inverse, relation with sea distance. So, we build two models, one with longitude as covariate and another with distance to sea as covariate. And compute the DIC to decide what model is better.

To consider a the non-linear relationship, we consider the coefficients of these covariates with a semi-parametric trend on the relationship, such that the coefficients over the range of the covariate values have a first order random walk prior. So we have the model adopted is of the form

$$log(\mu_i) = \alpha + \sum_{j=1}^{k} w_k I(F_i = F0_k) + x_i$$

were F is the distance to sea or the longitude and $F0_k$ is a knot (choose on range of the covariate values) and w_k are regression coefficients with first order random walk prior.

We start with the construction of the constrained refined Delaunay triangulation (CRDT). The use of the points with `cutoff=0` is not good in this case because we have some stations very close than another and other far of another. Using the `nndistG()` function of the **splancs**, [Rowlingson et al., 2012], we compute the distance from each station to its neighbor station

```
> summary(nndistG(as.matrix(prec[,1:2]))$dists)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   7.194  10.980  10.710  14.680  35.060
```

and we have station at same location to another and stations at 35 kilometers from other. So, is more good to use `cutoff` greater than 0.

The Paraná state border is formed by a sequence of 2055 coordinates, that form the segments. The half of this segments are separated by less than 1.007789 Kilometer and we have segments up to 14.13614 Kilometers.

```
> summary(sqrt(rowSums((pr.bord[-1,]-pr.bord[-nrow(pr.bord),])^2)))

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
 0.004195  0.615800  1.008000  1.327000  1.664000 14.140000
```

In this case, we need `cutoff` greater than 0. Because if we use `cutoff=0`, we have small triangles on the mesh on left of Figure 3.3. On the right mesh on this Figure, we have a good mesh created using `cutoff=15` and this same value to the maximum length of inner edges.

```
> (mesh0 <- inla.mesh.create.helper(, as.matrix(pr.bord), cutoff=0,
+                                   max.edge=c(15, 30), offset=c(15,30)))$n

[1] 3413

> (mesh <- inla.mesh.create.helper(, as.matrix(pr.bord), cutoff=15,
+                                   max.edge=c(15, 30), offset=c(15,30)))$n

[1] 1041
```

These meshes are visualized on Figure 3.3 with commands below

```
> xlm <- range(pr.bord[,1])*c(.9, 1.1)
> par(mfrow=c(1,2), mar=c(0,0,2,0))
> plot(pr.bord, type='l', asp=1, axes=FALSE, xlim=xlm)
> plot(mesh0, add=TRUE);   lines(pr.bord, col=2)
> plot(pr.bord, type='l', asp=1, axes=FALSE, xlim=xlm)
> plot(mesh, add=TRUE);   lines(pr.bord, col=2)
```

.

To fit the model, we use the stack functionality to prepare the data.

```
> A <- inla.spde.make.A(mesh, loc=as.matrix(prec[,1:2]))
> ind <- inla.spde.make.index(name='i', n.mesh=mesh$n)
> stk.dat <- inla.stack(data=list(y=prec$precMean), A=list(A,1,1),
+                       effects=list(c(ind, list(Intercept=1)),
+                         gEast=inla.group(prec$East),
+                         gSeaDist=inla.group(prec$seaDist)), tag='dat')
```

In this stack the intercept is included as one element on the index list set and the covariates as new elements of effects list. So the effects list in this example has three elements: the index set and two covariates. A different way is used on Chapter 4.

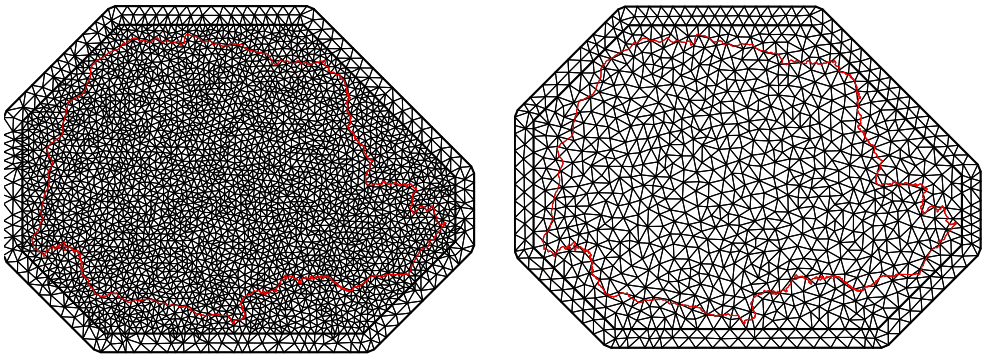

Figure 3.3: Two meshes to Paraná state

```
> spde <- inla.spde2.matern(mesh, alpha=2)
> f.east <- y ~ 0 + Intercept + f(gEast, model='rw1') + f(i, model=spde)
> f.sead <- y ~ 0 + Intercept + f(gSeaDist, model='rw1') + f(i, model=spde)
> r.east <- inla(f.east, family='Gamma', control.compute=list(dic=TRUE),
+                data=inla.stack.data(stk.dat),
+                control.predictor=list(A=inla.stack.A(stk.dat), compute=TRUE))
> r.sead <- inla(f.sead, family='Gamma', control.compute=list(dic=TRUE),
+                data=inla.stack.data(stk.dat),
+                control.predictor=list(A=inla.stack.A(stk.dat), compute=TRUE))
```

We have the DIC of each model with

```
> c(r.east$dic$dic, r.sead$dic$dic)

[1] 2542.432 2533.587
```

and choose the model with distance to sea as covariate. We got the summary of posterior distribution of the intercept with

```
> r.sead$summary.fixed

            mean         sd 0.025quant 0.5quant 0.975quant        kld
Intercept 1.9335 0.04368249   1.845974 1.933525   2.020701 0.02073266
```

To dispersion parameter of the gamma likelihood we have

```
> r.sead$summary.hy[1,]

      mean         sd 0.025quant   0.5quant 0.975quant
 14.222512   1.315507  11.818298  14.159459  16.977054
```

And, to $log(\kappa)$ we have

```
> r.sead$summary.hy[3,]

      mean         sd 0.025quant   0.5quant 0.975quant
 3.1758607  0.3079947  2.5712140  3.1758264  3.7812070
```

The variance and range of the spatial process we need a post processing. We obtain the marginal distribution for both by

```
> r.f <- inla.spde2.result(r.sead, 'i', spde, do.transf=TRUE)
```

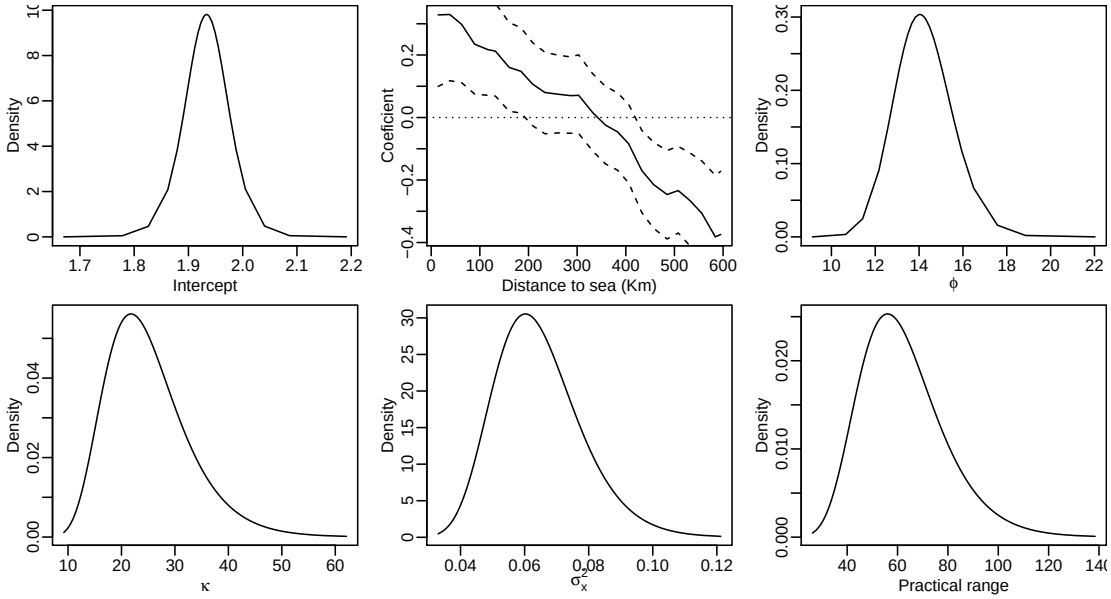

Figure 3.4: Posterior marginal distribution of β_0 (top left), the posterior mean (continuous line) and 95% credibility interval (dashed lines) to effect of distance to sea (top mid), posterior marginal distribution to ϕ (top right), posterior marginal distribution of κ (bottom left), posterior marginal distribution to nominal variance of the random field (bottom mid) and posterior marginal distribution of the practical range (bottom right).

The posterior mean for the marginal variance

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

```
[1] 0.06448124
```

And to the practical range of the process we have

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

```
[1] 62.43645
```

At the Figure 3.4 we look the posterior distribution to β_0, ϕ, κ, σ_x^2, practical range and to the mean and 95% credibility interval of the distance to sea effect at Figure 3.4.

```
> par(mfrow=c(2,3), mar=c(3,3.5,0,0), mgp=c(1.5, .5, 0), las=0)
> plot(r.sead$marginals.fix[[1]], type='l', xlab='Intercept', ylab='Density')
> plot(r.sead$summary.random[[1]][,1:2], type='l',
+      xlab='Distance to sea (Km)', ylab='Coeficient'); abline(h=0, lty=3)
> for (i in c(4,6)) lines(r.sead$summary.random[[1]][,c(1,i)], lty=2)
> plot(r.sead$marginals.hy[[1]], type='l', ylab='Density', xlab=expression(phi))
> plot.default(inla.tmarginal(function(x) exp(x), r.sead$marginals.hy[[3]]),
+              type='l', xlab=expression(kappa), ylab='Density')
> plot.default(r.f$marginals.variance.nominal[[1]], type='l',
+              xlab=expression(sigma[x]^2), ylab='Density')
> plot.default(r.f$marginals.range.nominal[[1]], type='l',
+              xlab='Practical range', ylab='Density')
```

3.3 Testing the significance of spatial effect

Now, we want to test the significance of the spatial random effect component on the model. To access the significance of this effect, can be use the DIC measure, comparing the model with this component with the DIC of the model without this component, fitted below

```
> r0.sead <- inla(y ~ 0 + Intercept + f(gSeaDist, model='rw1'),
+                 family='Gamma', control.compute=list(dic=TRUE),
+                 data=inla.stack.data(stk.dat),
+                 control.predictor=list(A=inla.stack.A(stk.dat), compute=TRUE))
```

So, we have the DIC of both models with

```
> c(r0.sead$dic$dic, r.sead$dic$dic)

[1] 2720.500 2533.587
```

and conclude that the spatial effect is significantly to the model for precipitation.

3.4 Prediction of the random field

To show the spatial effect, we make prediction of this effect on a fine grid. The `inla.mesh.projector()` get a projector matrix to a regular grid and with equal dimension in each direction by default. The Paraná state shape is more width than height. So, we choose diferent shape of grid to get an adequate cell size. We use

```
> (nxy <- round(c(diff(range(pr.bord[,1])), diff(range(pr.bord[,2])))/4))

[1] 166 116
```

Now, we get the projector with

```
> projgrid <- inla.mesh.projector(mesh, xlim=range(pr.bord[,1]),
+                                 ylim=range(pr.bord[,2]), dims=nxy)
```

and get the posterior mean and posterior standard deviation with

```
> xmean <- inla.mesh.project(projgrid, r.sead$summary.random$i$mean)
> xsd <- inla.mesh.project(projgrid, r.sead$summary.random$i$sd)
```

To good visualization, we make NA the values corresponding of the points of the grid out of the border of the Paraná state. To do it, we use the function `inout()` on **splancs** package, with

```
> table(xy.in <- inout(projgrid$lattice$loc,
+                      cbind(pr.bord[,1], pr.bord[,2])))

FALSE  TRUE
 7000 12256

> xmean[!xy.in] <- xsd[!xy.in] <- NA
```

We visualize the this on Figure 3.5. We see on top left graph at Figure 3.5 that the random field has variation from -0.6 to 0.4. It isn't despicable, because we have, on the top right graph, the standard errors around 0.2. The variation on the mean of this random effect express the remain variation after the consideration of the effect of the covariate on the model. The variation on its standard deviation is due to density of stations over the region. For example, on the top right graph we see, the first blue region from right to left is near Curitiba and around there many stations.

3.5 Prediction of the response on a grid

We want to predict the response on a grid. A naive approach is made by the projection of the sum of the linear predictor components and applying exponentiation to it, because we use the log link. A better alternative is the joint prediction with the estimation process. But it is computationally expensive when we have large grid.

Using the grid from previous section we have points discarded because they are not within the Paraná border. To make the predictions only on the points within the Paraná state we select the coordinates of the grid and the correspondent rows of the predictor matrix by

```
> prdcoo <- projgrid$lattice$loc[which(xy.in),]
> Aprd <- projgrid$proj$A[which(xy.in), ]
```

Now we get the covariate values at each point on the grid. First we compute the distance to sea

```
> seaDist0 <- nndistF(as.matrix(pr.bord[1034:1078,]), prdcoo)
```

and found the knot group, using the same used on the model

```
> ug.seadist <- sort(unique(inla.group(prec$seaDist)))
> ig0 <- apply(abs(outer(ug.seadist, seaDist0, '-')),2,which.min)
> g.seadist <- ug.seadist[ig0]
```

and put it into on a stack to prediction and join with the stack data

```
> stk.prd <- inla.stack(data=list(y=NA), A=list(Aprd,1),
+                       effects=list(c(ind, list(Intercept=1)),
+                         gSeaDist=g.seadist), tag='prd')
> stk.all <- inla.stack(stk.dat, stk.prd)
```

Now, we fit the model joining the prediction data together the observed data stack. But, now we don't need the computation of DIC, marginal distributions and quantiles, just the mean and standard deviation of the response

```
> r2.sead <- inla(f.sead, family='Gamma', data=inla.stack.data(stk.all),
+                 control.predictor=list(A=inla.stack.A(stk.all),
+                   compute=TRUE), quantiles=NULL,
+                 control.results=list(return.marginals.random=FALSE,
+                   return.marginals.predictor=FALSE))
```

Now, we extract the index on the predictor that corresponds to these wanted. Also, we put the mean and standard deviation on the matrix format, in the adequate positions.

```
> id.prd <- inla.stack.index(stk.all, 'prd')$data
> sd.prd <- m.prd <- matrix(NA, nxy[1], nxy[2])
> m.prd[xy.in] <- r2.sead$summary.fitted.values$mean[id.prd]
> sd.prd[xy.in] <- r2.sead$summary.fitted.values$sd[id.prd]
```

and we visualize on the Figure 3.5 the posterior mean and standard deviation to response with commands bellow on

```
> grid.arrange(levelplot(xmean, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)),
+              levelplot(xsd, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)),
+              levelplot(m.prd, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)),
+              levelplot(sd.prd, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)))
```

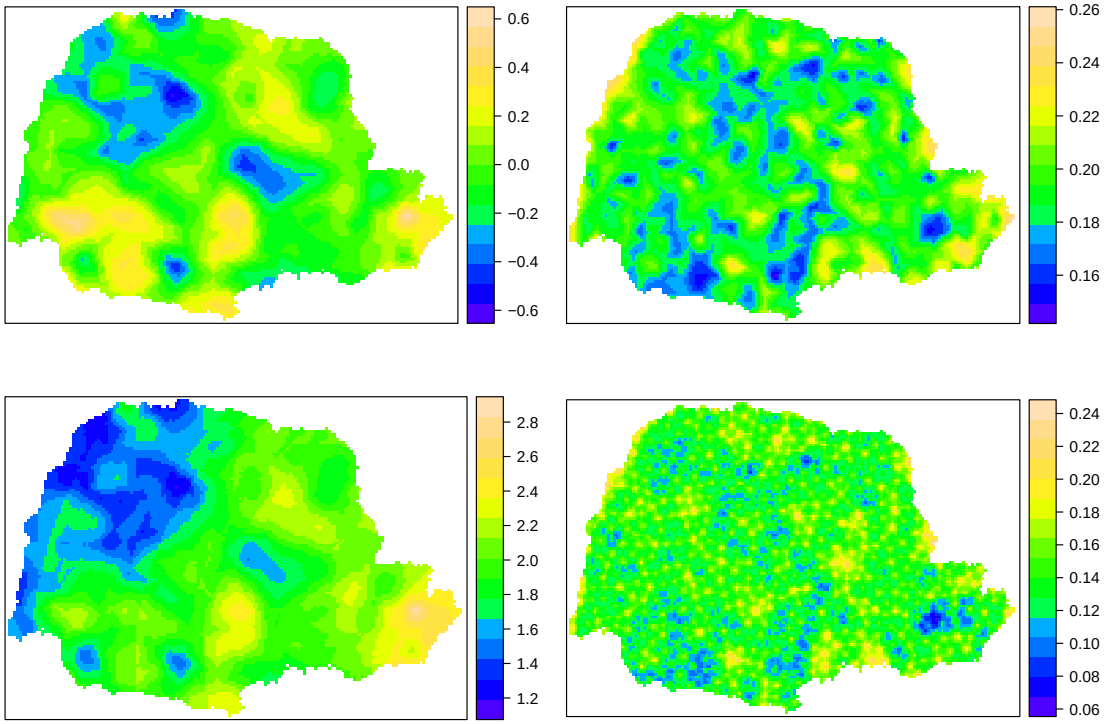

Figure 3.5: Posterior mean and standard deviation of the random field (top left and right respectively). Posterior mean and standard deviation of the response (top left and right respectively).

We see at this Figure a clear pattern on the average of the daily accumulated precipitation occurred on January of 2012. It's high near to sea and lower at north east part of the Paraná state.

Chapter 4

Semicontinuous model to daily rainfall

The daily rainfall data is one type of data that is not easy to fit a simple model. Because there are not days when it rains and it rains in the days, the distribution of rainfall is very asymmetric. In this section we build a jointly model to rainfall taking into account this fact. We build a semicontinuous model that is a jointly model to occurence and to positive values.

4.1 The data and the model

The data is

```
> prec <- read.table('data/rainfallParana.txt', header=TRUE)
> pr.bord <- read.table('data/borderParana.txt', header=TRUE)
> prec[1:3, 1:10]
```

```
      East    South Altitude jan01 jan02 jan03 jan04 jan05 jan06 jan07
1 512.8858 7472.958      365     0     0     0   0.0     0     0   2.5
2 523.4650 7460.923      344     0     1     0   0.0     0     0   6.0
3 535.9126 7461.972      904     0     0     0   3.3     0     0   5.1
```

and we have, for the first five days on January, the summary and the number of station with rain

```
> sapply(prec[,4:9], summary)
```

```
          jan01 jan02  jan03  jan04  jan05  jan06
Min.     0.0000  0.00  0.000  0.000  0.000  0.000
1st Qu.  0.0000  0.00  0.000  0.000  0.000  0.000
Median   0.0000  0.00  0.300  0.000  0.000  0.000
Mean     0.8074  2.87  5.928  3.567  2.793  4.059
3rd Qu.  0.0000  2.10  7.000  3.475  0.600  3.850
Max.    43.8000 55.60 78.500 62.900 64.100 81.400
NA's     2.0000  3.00  3.000  2.000  2.000  4.000
```

```
> apply(prec[,4:9]>0, 2, summary)
```

```
      jan01     jan02     jan03     jan04     jan05     jan06
Mode  "logical" "logical" "logical" "logical" "logical" "logical"
FALSE "538"     "399"     "294"     "373"     "429"     "368"
TRUE  "72"      "210"     "315"     "237"     "181"     "240"
NA's  "2"       "3"       "3"       "2"       "2"       "4"
```

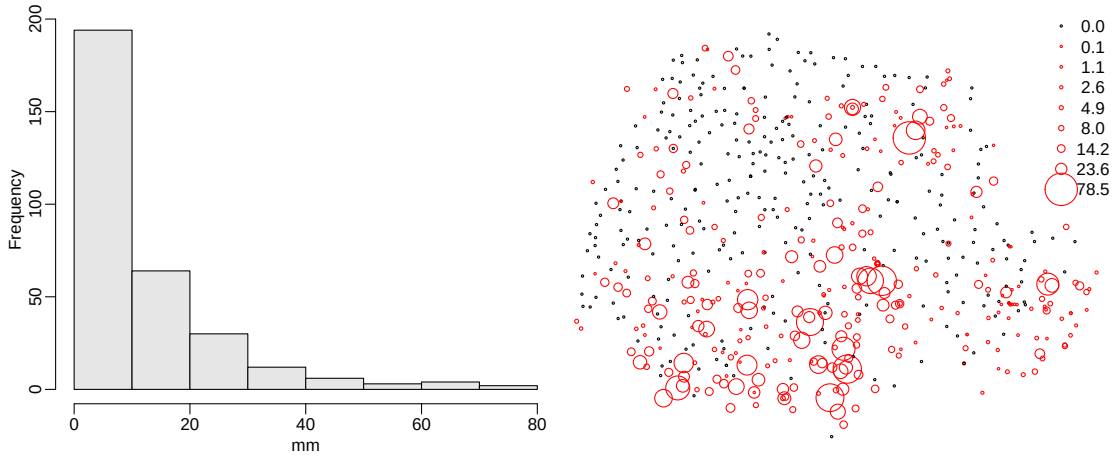

Figure 4.1: Map of precipitation (left) and histogram of positive precipitation (right).

and we see that we have no rain in 538 stations on the first day, 399 on second and 294 on third. We also have missing values.

Let r_i the rain on the location s_i. We use the augmented data by definition of two new variables. We define the occurence variable

$$z_i = \begin{cases} 1, & \text{if } r_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

and the amoung variable

$$y_i = \begin{cases} NA, & \text{if } r_i = 0 \\ r_i, & \text{otherwise} \end{cases}$$

Using the data on the third day we have

```
> jday <- 6
> z <- (prec[,jday]>0)+0
> y <- ifelse(prec[,jday]>0, prec[,jday], NA)
```

The map of this data are showed on the left graph of the Figure 4.1. Also, the histogram of the positive precipitation are on the rigth graph of this Figure. Both produced by the code below

```
> par(mfrow=c(1, 2), mar=c(3,3,.5,0), mgp=c(1.5,0.5,0))
> hist(y, xlab='mm', col=gray(.9), main='')
> par(mar=c(0,0,0,0))
> plot(prec[,1:2], cex=0.3 + prec[,jday]/20, asp=1, col=z+1, axes=FALSE)
> q.y <- c(0, quantile(y, 0:7/7, na.rm=T))
> legend('topright', format(q.y, dig=2), pch=1, bty='n',
+        pt.cex=0.3+q.y/20, col=c(1,rep(2,length(q.y)-1)))
```

We use a model with two likelihoods. We use the Bernoulli likelihood for z_i and the gamma for y_i.

$$z_i \sim \text{Bernoulli}(p_i) \qquad y_i \sim \text{Gamma}(a_i, b_i)$$

So, we need a two column response matrix one for each response and $2 \times n$ rows. We put the first response on first column at first n rows and the second response on last n rows of second column.

```
> n <- nrow(prec)
> Y <- matrix(NA, 2*n, 2)
> Y[1:n, 1] <- z
> Y[n + 1:n, 2] <- y
```

later we associate, on the model formulae, one likelihood for each column.

We define the linear predictor to first component by

$$\text{logit}(p_i) = \alpha_z + x_i$$

where α_z is an intercept and x_i is a random effect modeled by a Gaussian field through the SPDE approach.

To second component we consider that $E(y_i) = \mu_i = a_i/b_i$ and $V(y_i) = a_i/b_i^2 = \mu_i^2/\phi$, where ϕ us a precision parameter and we define a linear predictor to $\log(\mu_i)$. We have

$$\log(\mu_i) = \alpha_y + \beta x_i$$

where α_y is an intercept and β is a scaling parameter to x_i that is a shared random effect with the first component of the model.

4.2 Fitting the model and some results

To define a SPDE model we need a mesh. We prepare the mesh using the border coordinates with

```
> mesh <- inla.mesh.create.helper(, as.matrix(pr.bord), cutoff=15,
+                                   max.edge=c(15, 30), offset=c(15,30))
```

Also, we need to prepare the data in addequate form using the stack functionality. First we get the predictor matrix and the indexes

```
> A <- inla.spde.make.A(mesh, loc=as.matrix(prec[,1:2]))
> ind.z <- inla.spde.make.index(name='iz', n.mesh=mesh$n)
> ind.y <- inla.spde.make.index(name='iy', n.mesh=mesh$n)
```

because we have a 2 times n data, we need predictor matrix with $2 \times n$ rows. Specificaly, we need additional rows with zeros, below `A` to first part, and at end of `A` to second part. Also, for the intercept we need a vector with 1 on first n elements to estimate α_z and 0 on last n components. And the referse of this to estimate α_y. So, we have

```
> a0 <- Matrix(0, nrow(A), ncol(A))
> zIntrcpt <- rep(1:0, each=n)
> stk <- inla.stack(tag='est', data=list(y=Y),
+                   A=list(rBind(A, a0), 1, rBind(a0, A), 1),
+                   effects=list(ind.z, zIntercept=zIntrcpt,
+                     ind.y, yIntercept=1-zIntrcpt))
```

The intercepts here are estimated in same way when we have covariate. In this case, the covariate is assigned 0 (zero) at positions corresponding to the response unaffected by it. So, we have list of effects with length four, the index set of random effect on each response and the intercept on each response. A different way is used on Chapter 3.

To fit the model, we define one SPDE model and we make inference to β usint the copy of the strategy with `fixed=FALSE`.

```
> spde <- inla.spde2.matern(mesh, alpha=2)
> res <- inla(y ~ 0 + zIntercept + yIntercept +
+             f(iz, model=spde) + f(iy, copy='iz', fixed=FALSE),
+             family=c('binomial', 'gamma'),
+             data=inla.stack.data(stk), control.compute=list(dic=TRUE),
+             control.predictor=list(A=inla.stack.A(stk), compute=TRUE))
```

Now, we look at the summary of the posterior distribution to α_z and α_y

```
> round(res$summary.fix, 4)

             mean     sd 0.025quant 0.5quant 0.975quant    kld
zIntercept 0.0250 0.4855    -0.9677   0.0250     1.0177 0.0000
yIntercept 2.1029 0.1919     1.7134   2.1067     2.4759 0.0021
```

We apply the inverse of the link function to look at response scale with

```
> c(binomial(link='logit')$linkinv(res$summary.fix[1,1]),
+   exp(res$summary.fix[2,1]))

[1] 0.5062495 8.1898811
```

and it looks similar to observed:

```
> c(occ=mean(z, na.rm=TRUE), rain=mean(y, na.rm=TRUE))

       occ       rain
 0.5172414 11.4606349
```

The summary of the posterior distribution of the precision parameter on second component is

```
> res$summary.hy[1,]

      mean         sd 0.025quant   0.5quant 0.975quant
0.87265591 0.06675028 0.75037174 0.86947782 1.01226111
```

And the summary of the posterior distributio of the scaling parameter β is

```
> res$summary.hy[4,]

      mean         sd 0.025quant   0.5quant 0.975quant
0.33707805 0.07664981 0.18905815 0.33595279 0.49039996
```

and we see on this summary that the random effect x_i is shared by z and y. In other words, the probability of rain occurrence is correlated with the amoung of the rain, like the prefencential sampling problem [Diggle et al., 2010].

We look the posterior marginal distributions of α_z, α_y, ϕ and β on Figure 4.2 with commands below

```
> par(mfrow=c(2,2), mar=c(3,3,.5,.5), mgp=c(1.5,.5,0), las=1)
> plot(res$marginals.fix[[1]], type='l', ylab='Density',
+      xlab=expression(alpha[z]))
> plot(res$marginals.fix[[2]], type='l', ylab='Density',
+      xlab=expression(alpha[y]))
> plot(res$marginals.hy[[1]], type='l', ylab='Density',
+      xlab=expression(phi))
> plot(res$marginals.hy[[4]], type='l', ylab='Density',
+      xlab=expression(beta))
```

To decide if the occurence rain and the amoung of the rain have spatially dependency, we want to test the significance of the random effect x.

One approach is looking at the 2,5% and 97,5% quantiles of each element of the random effect. If for some of they has both quantiles with same signal, them they are significantly not null. We see on the Figure 4.3 the mean and this quantiles for each x in order of the posterior mean, with code below

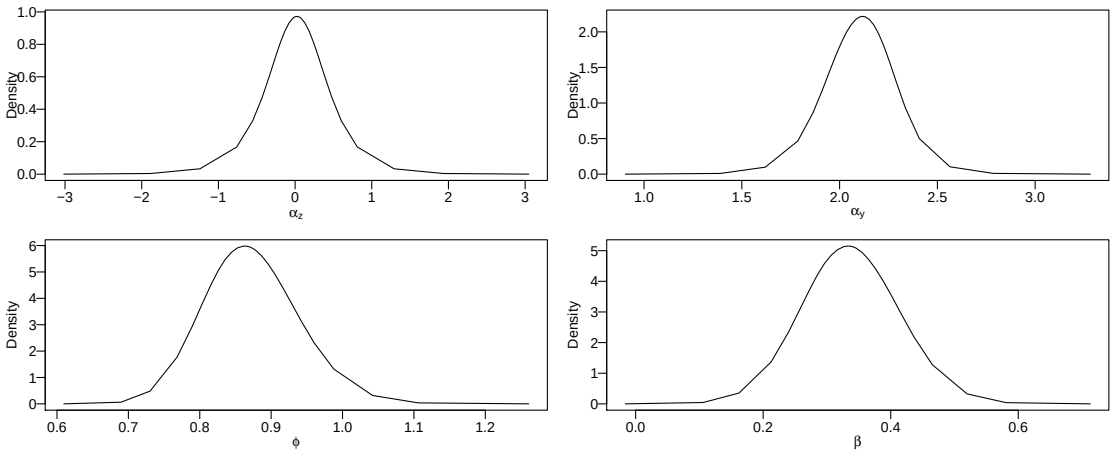

Figure 4.2: Posterior marginal distributions of α_z (top left), α_y (top right), ϕ (bottom left) and β (bottom right).

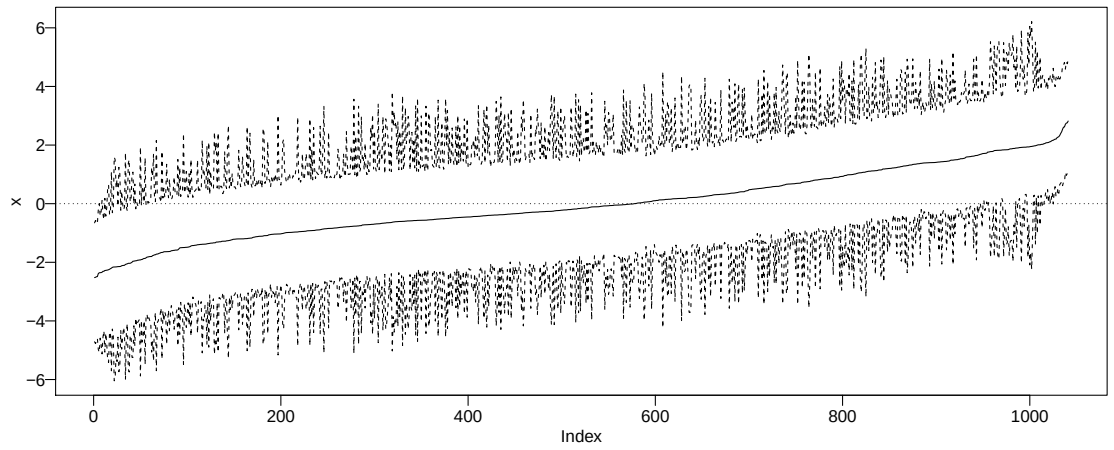

Figure 4.3: Posterior man of each x_i (continuous line) and its 2.5% and 97.5% quantiles of the posterior marginal distributions (dashed lines).

```
> ordx <- order(res$summary.random$iz$mean)
> par(mar=c(3,3,0.5,0.5), mgp=c(1.5,0.5,0), las=1)
> plot(res$summary.random$iz$mean[ordx], type='l', ylab='x',
+      ylim=range(res$summary.random$iz[, 4:6]))
> for (i in c(4,6)) lines(res$summary.random$iz[ordx,i], lty=2)
> abline(h=0, lty=3)
```

Other approach is to compare the DIC of the model with x with the DIC of the model without it.

We fit the model without x with

```
> res0 <- inla(y ~ 0 + zIntercept + yIntercept,
+              family=c('binomial', 'gamma'),
+              data=inla.stack.data(stk), control.compute=list(dic=TRUE),
+              control.predictor=list(A=inla.stack.A(stk), compute=TRUE))
```

and we have

```
> c(res0$dic$dic, res$dic$dic)

[1] 3002.941 2838.211
```

and conclude that the spatial random effect is significative to occurence and to amoung of the rain.

4.3 Results for the spatial random effect

The spatial random effect modeled by the SPDE approach has the κ scaling parameter, the marginal variance parameter σ_x^2 and the practical range, [Lindgren et al., 2011].

The summary of the posterior distribution of $log(\kappa)$ is

```
> res$summary.hy[3,]

      mean         sd 0.025quant   0.5quant 0.975quant
-3.8233199  0.3228686 -4.4482471 -3.8275276 -3.1804271
```

To obtain results for the variance and for the practical range we need a post processing. We obtain the marginal distribution for both by

```
> res.f <- inla.spde2.result(res, 'iz', spde, do.transf=TRUE)
```

The posterior mean for the marginal variance

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

[1] 2.062439
```

And to the practical range of the process we have

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

[1] 136.1469
```

At the Figure 4.4 we look the posterior distribution to κ, σ_x^2 and of the practical range.

```
> par(mfrow=c(1,3), mar=c(3,3.5,0,0), mgp=c(1.5, .5, 0), las=0)
> plot.default(inla.tmarginal(function(x) exp(x), res$marginals.hy[[3]]),
+              type='l', xlab=expression(kappa), ylab='Density')
> plot.default(res.f$marginals.variance.nominal[[1]], type='l',
+              xlab=expression(sigma[x]^2), ylab='Density')
> plot.default(res.f$marginals.range.nominal[[1]], type='l',
+              xlab='Practical range', ylab='Density')
```

Another interesting result is the map of the random field. To get it we use get the projector with

```
> (nxy <- round(c(diff(range(pr.bord[,1])), diff(range(pr.bord[,2])))/4))

[1] 166 116

> projgrid <- inla.mesh.projector(mesh, xlim=range(pr.bord[,1]),
+                                 ylim=range(pr.bord[,2]), dims=nxy)
```

and get the posterior mean and posterior standard deviation with

```
> xmean <- inla.mesh.project(projgrid, res$summary.random$iz$mean)
> xsd <- inla.mesh.project(projgrid, res$summary.random$iz$sd)
```

To good visualization, we make NA the values corresponding of the points of the grid out of the border of the Paraná state. To do it, we use the function `inout()` on **splancs** package, with

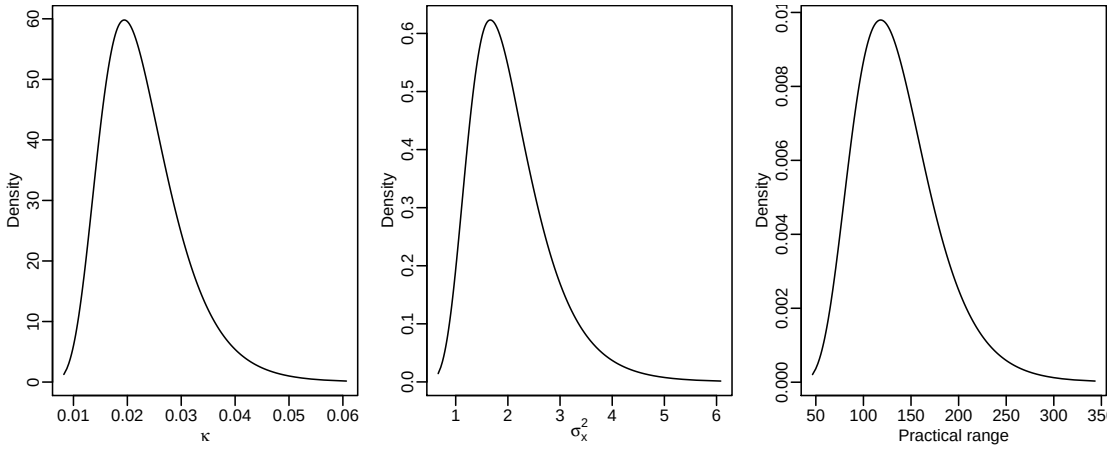

Figure 4.4: Posterior marginal distribution of κ (left), σ_x^2 (mid) and of the practical range (right).

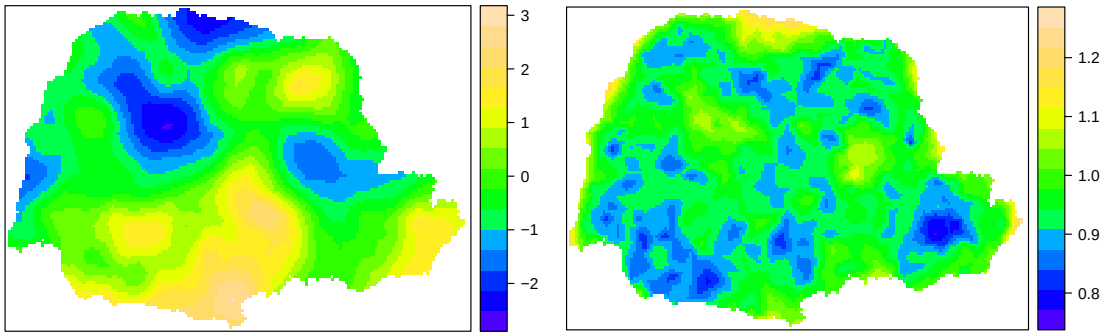

Figure 4.5: Posterior mean (left) and standard deviation (right) of the random field.

```
> table(xy.in <- inout(projgrid$lattice$loc,
+                      cbind(pr.bord[,1], pr.bord[,2])))

FALSE  TRUE
 7000 12256

> xmean[!xy.in] <- xsd[!xy.in] <- NA
```

We visualize the this on Figure 4.5. with code below

```
> grid.arrange(levelplot(xmean, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)),
+              levelplot(xsd, col.regions=topo.colors(99),
+                        xlab='', ylab='', scales=list(draw=FALSE)), nrow=1)
```

Comparing the left graph on Figure 4.5 with the right graph on the Figure 4.1 we have a more clear pattern of the rain on third day of 2012 on Paraná state. We see that in fact we have rain near the sea (the right border of the map), at south and a small point on northeast. Also we have no rain in some parts of nothwest.

Chapter 5

Joint modeling a covariate with misalignment

Here we focus on the situation when we have a response y and a covariate c. But we have misalignment, i.e., we have y observed at n_y locations and x observed at n_c locations. In this example, we design a soluction that not depends if we have or not some common observed locations for c and y.

A restriction is the assumption that c have spatial dependency. This restriction is made because we want a good model to predict c at locations of y. So, the goodness of prediction is proportional to the spatial dependency.

5.1 The model

Taking into acount that c have spatial dependency, a simple approach is to define a model for c, predict it on locations that we have y and build the model for y. But, in this two stage model, we don't take into account the prediction error of c on second model. The measurement error models is an approach to solve similar problems, [Muff et al., 2013]. But, here we are able to consider the spatial dependency on c. So we build a spatial model for c and another spatial model for y, and do the estimation process jointly.

We consider the following likelihood for c

$$c_i \sim N(m_i, \sigma_c^2)$$

where m_i is modeled as a random field and σ_c is a measurement error of c. So, when we predict c at location s, we have $m(s)$.

Let following model for y

$$y_i \sim N(\alpha_y + \beta c_i + x_i, \sigma_y^2)$$

where α_y is an intercept, β is the regression coefficient on c, c_i is the covariate at location of y_i, x_i is an zero mean random field and σ_y^2 measures the error that remain unexplained on Y.

A particular case is when we don't have the x term in the model for y. Another case, is when $\sigma_c^2 = 0$ and we don't have white noise in the covariate, i. e., the covariate is considered just a realization of a random field.

5.1.1 Simulation from the model

To test the approach on next section, we do a simulation from this model. First, we set the model parameters.

```
> n.y <- 123;          n.c <- 234
> alpha.y <- -5;       beta <- -3
> sigma2.y <- 0.5;     sigma2.c <- 0.2
```

First, we do the simulation of the locations

```
> set.seed(1)
> loc.c <- cbind(runif(n.c), runif(n.c))
> loc.y <- cbind(runif(n.y), runif(n.y))
```

Let the parameters of both random fields m and x:

```
> kappa.m <- 3;          sigma2.m <- 5
> kappa.x <- 7;          sigma2.x <- 3
```

and we use the `grf()` function of the **geoR**, [Ribeiro Jr and Diggle, 2001], package to do the simulation of both random fields. We need the simulation of m in both set of locations

```
> library(geoR)
> set.seed(2)
> mm <- grf(grid=rbind(loc.c, loc.y), messages=FALSE,
+           cov.pars=c(sigma2.m, 1/kappa.m), kappa=1)$data
> xx <- grf(grid=loc.y, messages=FALSE,
+           cov.pars=c(sigma2.x, 1/kappa.x), kappa=1)$data
```

and simulate the covariate and the response with

```
> set.seed(3)
> cc <- mm + rnorm(n.c+n.y, 0, sqrt(sigma2.c))
> yy <- alpha.y + beta*mm[n.c+1:n.y] + xx +
+   rnorm(n.y, 0, sqrt(sigma2.y))
```

5.2 Fitting the model

First we make the mesh

```
> pl01 <- matrix(c(0,1,1,0,0, 0,0,1,1,0), ncol=2)
> (mesh <- inla.mesh.create.helper(, pl01, cutoff=.03,
+                                  max.edge=c(.05,.1)))$n
```

```
[1] 1334
```

and use it for both the random fields. So, the both index set based on the mesh have the same values.

We do simulations of the covariate on the locations of the response just to simulate the response. But, in the problem that we want to solve in pratice, we don't have the covariate on the response locations. The misalignment implies in different predictor matrix for response and covariate. Also, we need a matrix with zeros to add on each of the predictor matrix to associate with the NA elements on response matrix defined later

```
> Ac <- rBind(inla.spde.make.A(mesh, loc=loc.c),
+             Matrix(0, n.y, mesh$n))
> Ay <- rBind(Matrix(0, n.c, mesh$n),
+             inla.spde.make.A(mesh, loc=loc.y))
```

To predict the covariate jointly, we need to model it jointly and we need two likelihoods. So, the response is formed by two colums matrix. We fill the first lines of first column of this matrix with the covariate values and the last lines of the second column with the response values. All the another positions is filled with NA.

We need an additional carefull in the data management. Because we want to estimate the effect of the covariate with the copy strategy, we need that the index on the both random field math the random field

```
> stk <- inla.stack(list(Y=cbind(c(cc[1:n.c], rep(NA,n.y)),
+                          c(rep(NA,n.c), yy))),
+                   A=list(Ac, Ay), tag='dat',
+                   effects=list(data.frame(m=1:mesh$n),
+                     data.frame(a.y=1, c.y=1:mesh$n, x=1:mesh$n)))
```

The estimation of the regression coefficient in this approach is treated as a hyperparameter, such as copy parameter of an latent field. In this case, we need to do a good prior specification. For example, is possible to know, a priori, the signal. We set a $N(-3, 25)$ prior to β. Also, we define the formulae for the model.

```
> form <- Y ~  0 + f(m, model=spde) +
+   a.y + f(x, model=spde) + f(c.y, copy='m', fixed=FALSE,
+                hyper=list(theta=list(param=c(-3,25), initial=0)))
```

define the spde model and fit the model with

```
> spde <- inla.spde2.matern(mesh)
> res <- inla(form, data=inla.stack.data(stk), family=rep('gaussian',2),
+             control.predictor=list(compute=TRUE, A=inla.stack.A(stk)),
+             inla.call='remote')
```

5.3 The results

The true values of the intercept and the summary of its posterior marginal

```
> round(c(True=alpha.y, res$summary.fix), 4)
```

```
   True
-5.0000 -4.5117  1.2598 -7.1621 -4.5181 -1.8195  0.0000
```

The true values of the precisions and the summary of the posterior marginals

```
> round(cbind(True=1/c(Prec.c=sigma2.c, Prec.y=sigma2.y),
+             res$summary.hy[1:2,]), 4)
```

```
       True   mean     sd 0.025quant 0.5quant 0.975quant
Prec.c    5 5.7731 0.7761     4.3897   5.7250     7.4362
Prec.y    2 1.5560 0.4052     0.9026   1.5087     2.4819
```

The true value of the regression coefficient and the summary of the posterior marginal

```
> round(c(True=beta, res$summary.hy[7,]), 4)
```

```
      True       mean         sd 0.025quant   0.5quant 0.975quant
   -3.0000    -2.7467     0.1584    -3.0622    -2.7450    -2.4398
```

The true values for the precision of the both random fields and the summary of the posterior marginals

```
> m.rf <- inla.spde2.result(res, 'm', spde)
> x.rf <- inla.spde2.result(res, 'x', spde)
> round(cbind(True=c(s2m=sigma2.m, s2x=sigma2.x),
+             mean=c(inla.emarginal(function(x) x,
+               m.rf$marginals.variance.n[[1]]),
+               inla.emarginal(function(x) x,
+                              x.rf$marginals.variance.n[[1]])),
+             rbind(inla.hpdmarginal(.95, m.rf$marginals.variance.n[[1]]),
+                   inla.hpdmarginal(.95, x.rf$marginals.variance.n[[1]]))), 4)
```

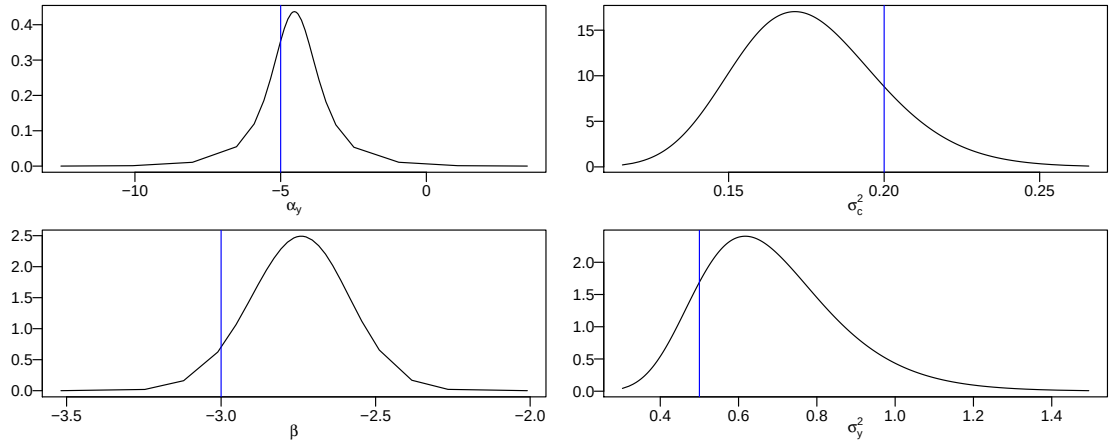

Figure 5.1: Posterior distribution of the likelihood parameters.

```
    True   mean    low   high
s2m    5 4.4952 1.4031 8.5673
s2x    3 4.5725 1.3504 8.8683
```

The true values for the scale parameter κ the mean of the posterior marginals and the 95% HPD credibility interval.

```
> post.k <- lapply(res$marginals.hy[c(4,6)], function(y)
+                  inla.tmarginal(function(x) exp(x), y))
> round(cbind(True=c(kappa.m=kappa.m, kappa.x=kappa.x),
+             t(sapply(post.k, function(x)
+                      c(mean=inla.emarginal(function(y) y, x),
+                        CI=inla.hpdmarginal(.95, x))))), 4)
```

```
        True   mean    CI1     CI2
kappa.m    3 3.7680 2.0325  5.6377
kappa.x    7 6.2617 2.3715 10.6976
```

We see the posterior distribution of likelihood parameters on Figure 5.1 generated with comands below

```
> par(mfcol=c(2,2), mar=c(3,3,.1,.1), mgp=c(1.5,.5,0), las=1)
> plot(res$marginals.fix[[1]], type='l',
+      xlab=expression(alpha[y]), ylab='')
> abline(v=alpha.y, col=4)
> plot(res$marginals.hy[[7]], type='l',
+      xlab=expression(beta), ylab='')
> abline(v=beta, col=4)
> plot.default(inla.tmarginal(function(x) 1/x, res$marginals.hy[[1]]),
+              type='l', xlab=expression(sigma[c]^2), ylab='')
> abline(v=sigma2.c, col=4)
> plot.default(inla.tmarginal(function(x) 1/x, res$marginals.hy[[2]]),
+              type='l', xlab=expression(sigma[y]^2), ylab='')
> abline(v=sigma2.y, col=4)
```

We see on the Figure 5.1 that the posterior distribution covers the true values of all the parameters.

Now we want to see the posterior distribution of the both random fields. The practical range of the process is $sqrt(8\nu)/\kappa$, where $\nu = 1$ on this analisys. We see these parameters on Figure 5.2 generated with comands below

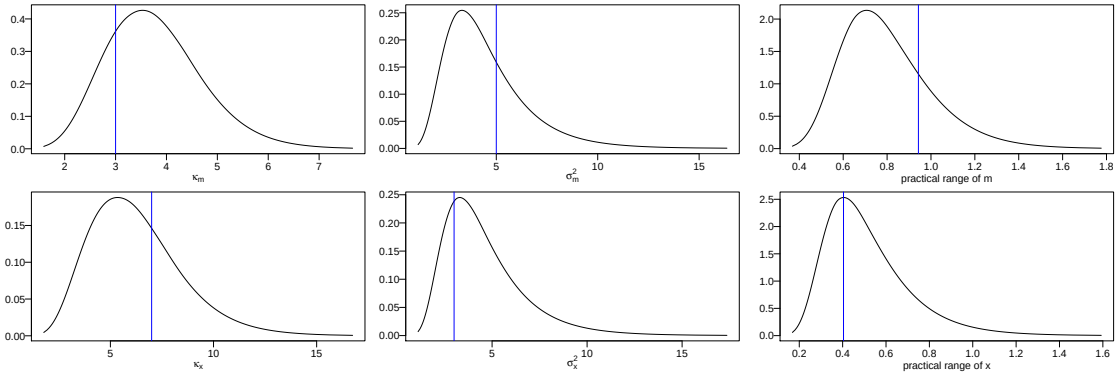

Figure 5.2: Posterior distribution of all the parameters of both random field.

```
> par(mfcol=c(2,3), mar=c(3,3,.1,.3), mgp=c(1.5,.5,0), las=1)
> plot.default(post.k[[1]], type='l', xlab=expression(kappa[m]), ylab='')
> abline(v=kappa.m, col=4)
> plot.default(post.k[[2]], type='l', xlab=expression(kappa[x]), ylab='')
> abline(v=kappa.x, col=4)
> plot.default(m.rf$marginals.variance.n[[1]], type='l',
+              xlab=expression(sigma[m]^2), ylab='')
> abline(v=sigma2.m, col=4)
> plot.default(x.rf$marginals.variance.n[[1]], type='l',
+              xlab=expression(sigma[x]^2), ylab='')
> abline(v=sigma2.x, col=4)
> plot.default(m.rf$marginals.range.n[[1]], type='l',
+              xlab='practical range of m', ylab='')
> abline(v=sqrt(8)/kappa.m, col=4)
> plot.default(x.rf$marginals.range.n[[1]], type='l',
+              xlab='practical range of x', ylab='')
> abline(v=sqrt(8)/kappa.x, col=4)
```

We see on Figure 5.2 that the posterior marginal distribution of the all parameters of both spatial process cover the true values well.

Another intersting result is the prediction of the covariate on the locations of the response. We have the simulated values of m on that locations. So, we are able to see if the predictions are good.

The predictor matrix used on the estimation proces maps the nodes from mesh vertices to the data locations. The first lines of the predictor matrix for the covariate can be used to access the predictions on the locations of the covariate. Also, we have the predictor matrix used to the response. The last lines of this matrix that maps the mesh vertices to the response locations. Because we have the covariate simulated in the both set of locations, we use the correspondent parts of both predictor matrix to project the posterior mean and the posterior variance on the locations.

We get this matrix by

```
> mesh2locs <- rBind(Ac[1:n.c,], Ay[n.c+1:n.y, ])
```

and the posterior mean and posterior standard deviations with

```
> m.mprd <- drop(mesh2locs%*%res$summary.ran$m$mean)
> sd.mprd <- drop(mesh2locs%*%res$summary.ran$m$sd)
```

With this aproach for this both posterior summary can be an aproximation to 95% credibility interval, with normally supposition. We see it this results with comands below

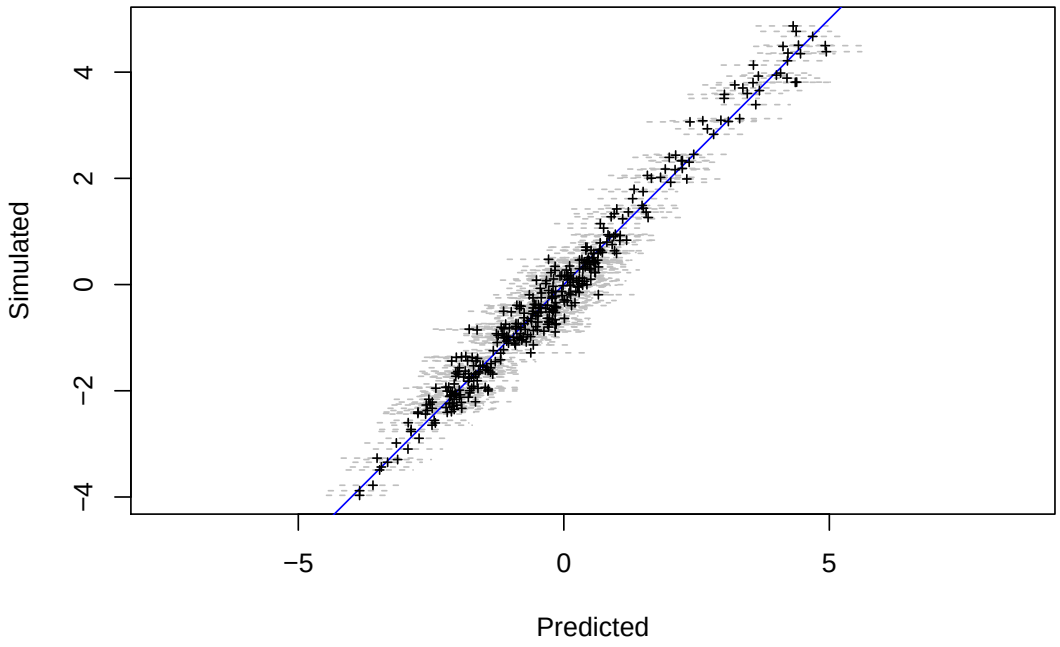

Figure 5.3: Simulated versus predicted values of m (+) and the approximated credibility intervals.

```
> plot(m.mprd, mm, asp=1, type='n',
+      xlab='Predicted', ylab='Simulated')
> segments(m.mprd-2*sd.mprd, mm, m.mprd+2*sd.mprd, mm,
+          lty=2, col=gray(.75))
> abline(c(0,1), col=4); points(m.mprd, mm, pch=3, cex=.5)
```

on the Figure 5.3. The blue line represents the situation where predicted is equal to simulated. We see that the prediction are wery well.

Chapter 6

Non stationary model

6.1 Introduction

To introduce the non stationarity we need to remember the definition of the precision matrix of the GMRF that defines the RF. This matrix is defined on equations (1.10) and (**??**). The non stationarity is made by a redefinition of the precision matrix.

This new definition is

$$\boldsymbol{Q} = \boldsymbol{D}^{(0)}(\boldsymbol{D}^{(1)}\boldsymbol{M}^{(0)}\boldsymbol{D}^{(1)} + \boldsymbol{D}^{(2)}\boldsymbol{D}^{(1)}\boldsymbol{M}^{(1)} + (\boldsymbol{M}^{(1)})^T\boldsymbol{D}^{(1)}\boldsymbol{D}^{(2)} + \boldsymbol{M}^{(2)})\boldsymbol{D}^{(0)} \qquad (6.1)$$

where $\boldsymbol{M}^{(0)}$, $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$, are provided from the finite element method - FEM based on the mesh. For $\alpha = 1$ ($\nu = 0$), we have $\boldsymbol{M}^{(0)} = \boldsymbol{C}$, $(\boldsymbol{M}^{(1)})_{ij} = 0$ and $\boldsymbol{M}^{(2)} = \boldsymbol{G}$. For $\alpha = 2$ ($\nu = 1$), we have $\boldsymbol{M}^{(0)} = \boldsymbol{C}$, $\boldsymbol{M}^{(1)} = \boldsymbol{G}$ and $\boldsymbol{M}^{(2)} = \boldsymbol{G}\boldsymbol{C}^{-1}\boldsymbol{G}$.

All three $\boldsymbol{D}^{(0)}$, $\boldsymbol{D}^{(1)}$ and $\boldsymbol{D}^{(2)}$ are diagonal with elements used to describe the non-stationarity. The definition of these matrixes are

$$\begin{aligned}
\boldsymbol{D}^{(0)} &= diag\{\boldsymbol{D}_i^{(0)}\} = diag\{e^{\phi_i^{(0)}}\} \\
\boldsymbol{D}^{(1)} &= diag\{\boldsymbol{D}_i^{(1)}\} = diag\{e^{\phi_i^{(1)}}\} \\
\boldsymbol{D}^{(2)} &= diag\{\boldsymbol{D}_i^{(2)}\} = diag\{\phi_i^{(2)}\}
\end{aligned}$$

where

$$\phi_i^{(k)} = \boldsymbol{B}_{i,0}^{(k)} + \sum_{j=1}^{p} \boldsymbol{B}_{i,j}^{(k)}\theta_j, \quad i = 1, \ldots, n$$

with the $\boldsymbol{B}^{(k)}$: n-by-$(p+1)$ user defined matrix.

The default stationary SPDE model uses $\boldsymbol{B} = [0\ 1\ 0]$ (one by three) matrix for the marginal variance and uses $\boldsymbol{B} = [0\ 0\ 1]$ (one by three) matrix for the scaling parameter κ. When this matrix have just one line, the another lines are formed using the same element of this first line. In the next section, we add one of the location coordinates as a fourth column to build a non stationary model.

6.2 An example

In this section we define a model where the variance depends in one of the coordinates. Note that the any precision matrix defined on the equation (6.1) we need $\boldsymbol{M}^{(0)}$, $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ that are based on any mesh.

First, we define a polygon to define a mesh. We define an unitary square

```
> pl01 <- cbind(c(0,1,1,0,0), c(0,0,1,1,0))
```

and build a mesh on the polygon with

```
> (mesh <- inla.mesh.create.helper(, pl01, max.edge=c(0.07,.12)))$n
```

```
[1] 924
```

Now, we define the non stationary SPDE model. The non stationarity is defined on the two $\boldsymbol{B}$ matrix, one for τ and another for κ. We want to define a model where the variance depends on the first coordinate. We just choose to put it into a fourth column on the $\boldsymbol{B}$ matrix for τ. Also, we need a prior for the three dimentional θ, simplified by definition of a vetor of mean and precision (considering independence of the priors).

```
> spde <- inla.spde2.matern(mesh, B.tau=cbind(0, 1, 0, sin(pi*mesh$loc[,1])),
+                           B.kappa=cbind(0, 0, 1, 0), ##mesh$loc[,1]),
+                           theta.prior.mean=rep(0, 3),
+                           theta.prior.prec=rep(1, 3))
```

To finalize the precision matrix definition, we need to define the values of θ. Just to test, we define two different precision matrix, both with non stationarity based on the same $\boldsymbol{B}$ matrix. The θ vector has length equal the number of columns of $\boldsymbol{B}$ minus one. The two different θ vectors are

```
> theta1 <- c(-1, 2, -1)
> theta2 <- c(-1, 2, 1)
```

and we have both the precision matrix with

```
> Q1 <- inla.spde2.precision(spde, theta=theta1)
> Q2 <- inla.spde2.precision(spde, theta=theta2)
```

The first and second values of these vetors are the same. The structure of the $\boldsymbol{B}$ matrix indicate that we have

$$\tau_i = e^{\theta_1 + \theta_3 sin(\pi loc[i,1])}$$

and we have that the second precision matrix with larger values of τ_i, because its values are increased by a positive value.

To make more clear, we compute both covariance matrix implied. The covariance matrix of

$$x(s) = \sum_{k=1}^{n} A_k(s) w_k$$

is easy to obtain when we have s (the locations) equals the locations of the mesh vertices. It is just the inverse of the precision matrix defined.

```
> cov1 <- inla.qinv(Q1);          cov2 <- inla.qinv(Q2)
```

and we see a summary of the variance (diagonal of the covariance matrix) for both covariance matrix

```
> v1 <- diag(cov1);       v2 <- diag(cov2)
> rbind(v1=summary(v1),  v2=summary(v2))
```

```
       Min.  1st Qu.   Median    Mean 3rd Qu.   Max.
v1 0.004241 0.011520 0.031180 0.04007 0.06624 0.1786
v2 0.001495 0.002265 0.004924 0.01621 0.01432 0.1565
```

and we see that the first has larger variances. Its because the precision is less than the second.

We see the variance of both process on the Figure **??** by

```
> par(mar=c(3,3,.5,.5), mgp=c(1.7, .5, 0), las=1)
> plot(mesh$loc[,1], v1, ylim=range(v1,v2),
+      xlab='x-coordinate', ylab='variance', las=1)
> points(mesh$loc[,1], v2, col=2)
```

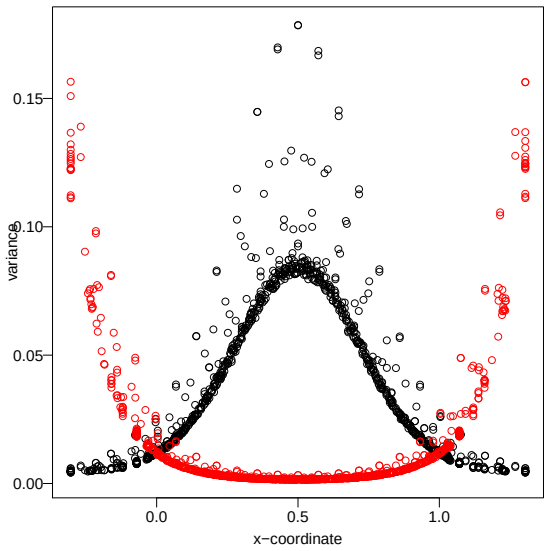

Figure 6.1: Variances implied by both non stationary process defined.

6.3 Simulation on the mesh vertices

Both precision matrix defined on previous the section consider that the locations are the triangles vertices of mesh. So, the simulation made with it is a realization of the random field on each point of the triangles vertices of the mesh. We use the same seed for each simulation, just to show it.

```
> sample1 <-  as.vector(inla.qsample(1, Q1, seed=1))
> sample2 <-  as.vector(inla.qsample(1, Q2, seed=1))
```

We compute the standard deviations for both the samples considering groups defined in accord to the first coordinate of the locations:

```
> tapply(sample1, round(inla.group(mesh$loc[,1], 5),3), var)

     -0.073        0.18       0.496       0.825       1.071
0.008349852 0.021988985 0.051450362 0.022831676 0.007851851

> tapply(sample2, round(inla.group(mesh$loc[,1], 5),3), var)

     -0.073        0.18       0.496       0.825       1.071
0.026432644 0.003051414 0.001069714 0.004342003 0.024451202
```

We observe that the variance of the sample from the first random field increase near 0.5 and decrease near 0 and near 1. For the sample of the second random field the oposite happens and we have larger values, because we have less precision.

We see the simulated values projected on a grid on Figure 6.2. We use a projector matrix to project the simulated values on the grid limited on the unit square with limits (0,0) and (1,1) with

```
> proj <- inla.mesh.projector(mesh, xlim=0:1, ylim=0:1)
> grid.arrange(levelplot(inla.mesh.project(proj, field=sample1),
+                        xlab='', ylab='', scale=list(draw=FALSE),
+                        col.regions=topo.colors(100)),
+              levelplot(inla.mesh.project(proj,field=sample2),
+                        xlab='', ylab='', scale=list(draw=FALSE),
+                        col.regions=topo.colors(100)), nrow=1)
```

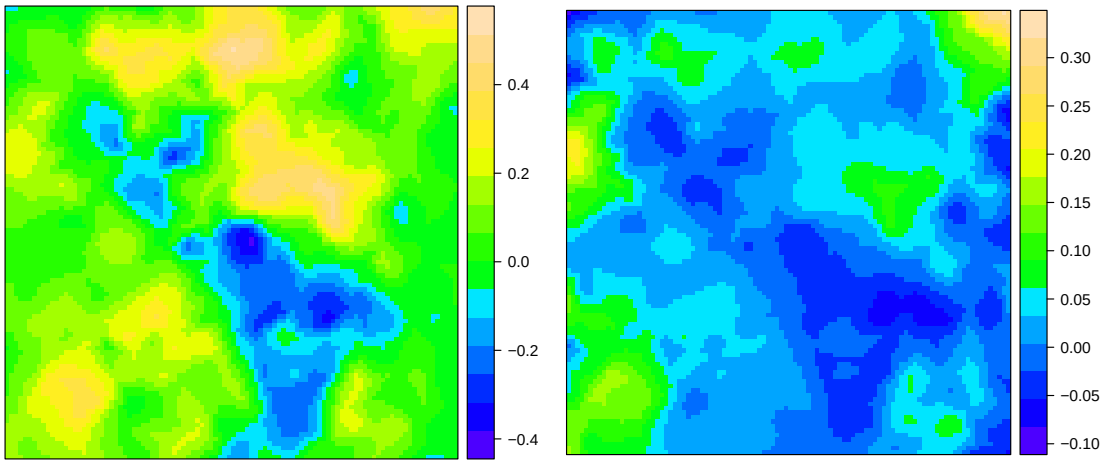

Figure 6.2: Two simulated random fields, using two diferent θ on same basis functions.

6.3.1 Simulation with linear constraint

The linear constraint is common on models such as random walk, in one or two dimentions. Its because these models are improper. In this section we just want to show how we do linear constraint in the SPDE models.

Because the SPDE models are based on the Finite Element Method (FEM) aproximation, the sum-to-zero restriction in this case is non trivial.

The issue is that

$$\sum_k x_k$$

doesn't mean anything for the mesh-based spde-models. Whereas

$$\int x(s)ds = \int \Psi_(s)x_k ds$$

does mean something, and that integral is equal to

$$\sum_k C_k k x_k$$

so the constraint

$$\int x(s)ds = 0$$

is provided by

$$A * x = 0, where A_{1k} = C_{kk}$$

Where $\boldsymbol{C}$ is the matrix used on the FEM.

Using $A = (1, ..., 1)$ instead of $diag(\boldsymbol{C})$ leads to very bad behaviour for irregular meshes. So, if we want a linear constrait, we need to use $\boldsymbol{C}$.

That matrix is obtained by the `inla.mesh.fem()` function:

```
> Aconstr <- matrix(diag(inla.mesh.fem(mesh)$c0), 1)
```

and we do the simulation with

```
> s1r <-  as.vector(inla.qsample(1, Q1, seed=1, constr=list(A=Aconstr,e=0)))
> s2r <-  as.vector(inla.qsample(1, Q2, seed=1, constr=list(A=Aconstr,e=0)))
```

and we have

```
> rbind(s1r=summary(s1r), s2r=summary(s2r))
```

```
       Min.  1st Qu.   Median      Mean 3rd Qu.   Max.
s1r -0.5478 -0.07647 -0.00151 -0.003233 0.08569 0.6383
s2r -0.6664 -0.03234  0.01279  0.006573 0.04994 0.4133
```

```
> c(cor1=cor(sample1, s1r), cor2=cor(sample2, s2r))
```

```
     cor1      cor2
0.9622621 0.9874610
```

where the mean of the process simulated on the mesh vertices have mean near zero.

6.4 Estimation with data simulated on the mesh vertices

The model can be fitted easly with the data simulated on mesh vertices. Considering that we have data exactly on each vertice of the mesh, we don't need the use of any predictor matrix and the stack functionality. Because we have just realizations of the random field, we don't have noise and need to fix the precision of the Gaussian likelihood on higth value, for example on the value e^{20}

```
> clik <- list(hyper=list(theta=list(initial=20, fixed=TRUE)))
```

Remember that we have a zero mean random field, so we also don't have fixed parameters to fit. We just do

```
> formula <- y ~ 0 + f(i, model=spde)
> fit1 <- inla(formula, control.family=clik,
+              data=data.frame(y=sample1, i=1:mesh$n))
> fit2 <- inla(formula, control.family=clik,
+              data=data.frame(y=sample2, i=1:mesh$n))
```

We look at the summary of the posterior for θ (joined with the true values). For the first sample

```
> round(cbind(true=theta1, fit1$summary.hy), 4)
```

```
             true    mean     sd 0.025quant 0.5quant 0.975quant
Theta1 for i   -1 -0.9928 0.0330    -1.0632  -0.9903    -0.9342
Theta2 for i    2  2.0984 0.1185     1.8941   2.0877     2.3545
Theta3 for i   -1 -0.9660 0.0446    -1.0463  -0.9690    -0.8719
```

and for the second

```
> round(cbind(true=theta2, fit2$summary.hy), 4)
```

```
             true    mean     sd 0.025quant 0.5quant 0.975quant
Theta1 for i   -1 -0.9471 0.0303    -1.0137  -0.9439    -0.8962
Theta2 for i    2  2.0874 0.1598     1.8382   2.0653     2.4487
Theta3 for i    1  1.0782 0.0491     0.9950   1.0733     1.1854
```

We see that for both we have good results. In next section we see more results.

6.5 Estimation with locations not on mesh vertices

Suppose that we have the data on the locations simulated by the commands below

```
> set.seed(2);     n <- 100
> loc <- cbind(runif(n), runif(n))
```

Now, we project the data simulated on the mesh vertices to this locations. To do it, we need a projection matrix

```
> projloc <- inla.mesh.projector(mesh, loc)
```

with

```
> x1 <- inla.mesh.project(projloc, sample1)
> x2 <- inla.mesh.project(projloc, sample2)
```

and we have the sample data on these locations.

Now, because the this locations aren't vertices of the mesh, we need to use the stack functionality. First, we need the predictor matrix. But this is the same used to 'sample' the data.

And we define the stack for each one of the samples

```
> stk1 <- inla.stack(list(y=x1), A=list(projloc$proj$A), tag='d',
+                    effects=list(data.frame(i=1:mesh$n)))
> stk2 <- inla.stack(list(y=x2), A=list(projloc$proj$A), tag='d',
+                    effects=list(data.frame(i=1:mesh$n)))
```

And we fit the model with

```
> res1 <- inla(formula, data=inla.stack.data(stk1), control.family=clik,
+              control.predictor=list(compute=TRUE, A=inla.stack.A(stk1)))
> res2 <- inla(formula, data=inla.stack.data(stk2), control.family=clik,
+              control.predictor=list(compute=TRUE, A=inla.stack.A(stk2)))
```

The true and summary of marginal posterior distribution for θ:

```
> round(cbind(True=theta1, res1$summary.hy), 4)
```

```
             True    mean     sd 0.025quant 0.5quant 0.975quant
Theta1 for i   -1 -1.0439 0.1808    -1.4071  -1.0406    -0.6962
Theta2 for i    2  2.1971 0.1846     1.8202   2.2034     2.5442
Theta3 for i   -1 -0.9788 0.2424    -1.4430  -0.9838    -0.4923
```

```
> round(cbind(True=theta2, res2$summary.hy), 4)
```

```
             True    mean     sd 0.025quant 0.5quant 0.975quant
Theta1 for i   -1 -0.9798 0.1829    -1.3337  -0.9822    -0.6150
Theta2 for i    2  2.1625 0.2039     1.7413   2.1716     2.5411
Theta3 for i    1  0.9154 0.2459     0.4309   0.9161     1.3965
```

To make the visualization more good, we take the logarithmum of the variance.

```
> x1.mean <- inla.mesh.project(proj, field=res1$summary.ran$i$mean)
> x1.var <- inla.mesh.project(proj, field=res1$summary.ran$i$sd^2)
> x2.mean <- inla.mesh.project(proj, field=res2$summary.ran$i$mean)
> x2.var <- inla.mesh.project(proj, field=res2$summary.ran$i$sd^2)
```

We visualize, for both random fields, the simulated, the predicted (posterior mean) and the posterior variance on Figure 6.3 with commands below

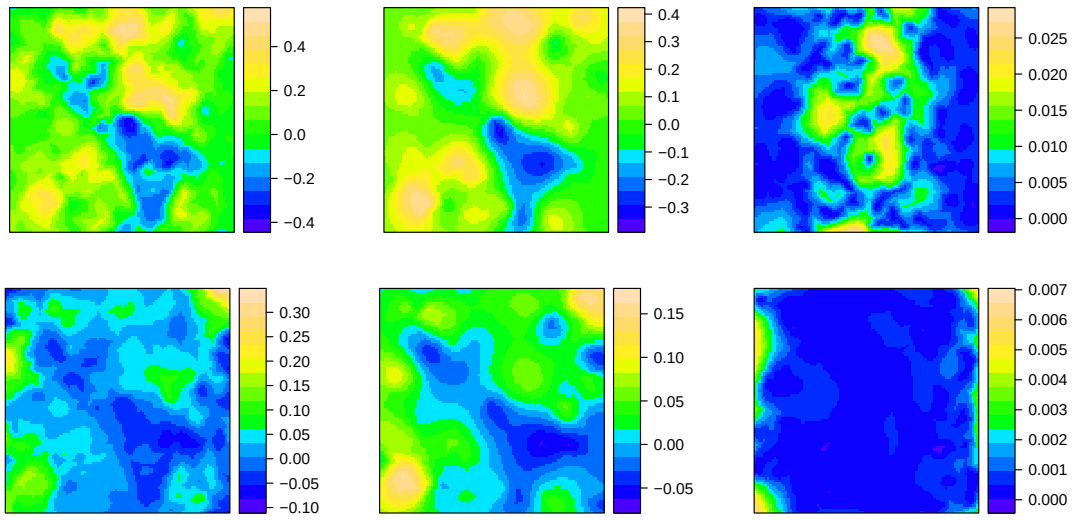

Figure 6.3: Simulated (top and bottom left), posterior mean (top and bottom mid) and the posterior variance (top and bottom right) for both random fields.

```
> do.call(function(...) grid.arrange(..., nrow=2),
+         lapply(list(inla.mesh.project(proj, sample1), x1.mean, x1.var,
+                     inla.mesh.project(proj, sample2), x2.mean, x2.var),
+                levelplot, xlab='', ylab='',
+                col.regions=topo.colors(100), scale=list(draw=FALSE)))
```

We see on the Figure 6.3 that the predicted values are similar to the simulated ones. Also, we see that the posterior variance of the first model increase near 0.5 on the first coordinate. And we see the oposite for the second random field. Also, we see that the variance of the first is greater than the second.

Bibliography

[Abrahamsen, 1997] Abrahamsen, P. (1997). A review of gaussian random fields and correlation functions. Norwegian Compting Center report No. 917.

[Besag, 1981] Besag, J. (1981). On a system of two-dimensional recurrence equations. *J. R. Statist. Soc. B*, 43(3):302–309.

[Cameletti et al., 2012] Cameletti, M., Lindgren, F., Simpson, D., and Rue, H. (2012). Spatio-temporal modeling of particulate matter concentration through the spde approach. *Advances in Statistical Analysis*.

[Cressie, 1993] Cressie, N. (1993). *Statistics for Spatial Data*. Wiley, N. Y. 990p.

[Diggle et al., 2010] Diggle, P. J., Menezes, R., and Su, T. (2010). Geostatistical inference under preferential sampling. *Journal of the Royal Statistical Society, Series C*, 59(2):191–232.

[Diggle and Ribeiro Jr, 2007] Diggle, P. J. and Ribeiro Jr, P. J. (2007). *Model-Based Geostatistics*. Springer Series in Statistics. Hardcover. 230p.

[Lindgren, 2012] Lindgren, F. (2012). Continuous domain spatial models in r-inla. *The ISBA Bulletin*, 19(4). URL: `http://www.r-inla.org/examples/tutorials/spde-from-the-isba-bulletin`.

[Lindgren et al., 2011] Lindgren, F., Rue, H., and Lindström, J. (2011). An explicit link between gaussian fields and gaussian markov random fields: the stochastic partial differential equation approach. *J. R. Statist. Soc. B*, 73(4):423–498.

[Muff et al., 2013] Muff, S., Riebler, A., Rue, H., Saner, P., and Held, L. (2013). Measurement error in glmms with inla. *submitted*.

[Ribeiro Jr and Diggle, 2001] Ribeiro Jr, P. J. and Diggle, P. J. (2001). geoR: a package for geostatistical analysis. *R-NEWS*, 1(2):14–18. ISSN 1609-3631.

[Rowlingson et al., 2012] Rowlingson, B., Diggle, P., adapted, packaged for R by Roger Bivand, pcp functions by Giovanni Petris, and goodness of fit by Stephen Eglen (2012). *splancs: Spatial and Space-Time Point Pattern Analysis*. R package version 2.01-31.

[Rue and Held, 2005] Rue, H. and Held, L. (2005). *Gaussian Markov Random Fields: Theory and Applications*. Monographs on Statistics & Applied Probability. Boca Raton: Chapman and Hall.

[Rue et al., 2009] Rue, H., Martino, S., and Chopin, N. (2009). Approximate bayesian inference for latent gaussian models using integrated nested laplace approximations (with discussion). *Journal of the Royal Statistical Society, Series B*, 71(2):319–392.

[Tobler, 1970] Tobler, W. R. (1970). A computer movie simulating urban growth in the detroid region. *Economic Geography*, 2(46):234–240.