The R-INLA tutorial: SPDE models

Warning: work in progress...

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

Elias T. Krainski and Finn Lindgren

April 15, 2014

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 the mesh building (necessary to fit the models) and show the application on a toy example with details. In forward examples we presents some funcionalities to fit more complex models, such as non-Gaussian, models with two likelihoods, such as a semi-continuous, missaligned and preferential sampling, where we model jointly the point pattern process. Also, we introduce the non-stationary SPDE models, space-time model, data-cloning.

2014 March 10: fix log-Cox example, add covariate
2013 December 21:
 * fix several missprints
 * add details on: likelihood, semicontinuous and spacetime examples
2013 October 08:
 * Finn's suggestions on two likelihood examples
2013 October 02:
 * mesh news: inla.mesh.2d, inla.noncovexhull, SpatialPolygons
 * toy-example improved (maybe more clear...)
 * new chapters: likelihood through SPDE, point process,
  preferential sampling, spatio temporal, data cloning
2013 March 21:
 * non-stationary example and joint covariate modelling
2013 March 01:
 * first draft: introduction, toy example, rainfall on Parana State

Contents

Chapter 1

Introduction

If we have data measured at some locations, so we have a point refereed data set. By locations we mean some coordinates reference system where are each data from, for example, the longitude and latitude coordiantes. 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.

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.

1.1 The Gaussian random field

To introduce some 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

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

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

$$-\frac{1}{2}\left(nlog(2\pi) + log(|\Sigma|) + [x(s) - \mu_x]^T\Sigma^{-1}[x(s) - \mu_x]\right) \tag{1.3}$$

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'.

1.2 Simulation of a data set

First we took a look at the model from parametrization used in a commonly software used to analyse the point refereed data in **R**, the **geoR** package, [Ribeiro Jr and Diggle, 2001]. In this section we show how we simulate a dataset with this parametrization.

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:

```
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:

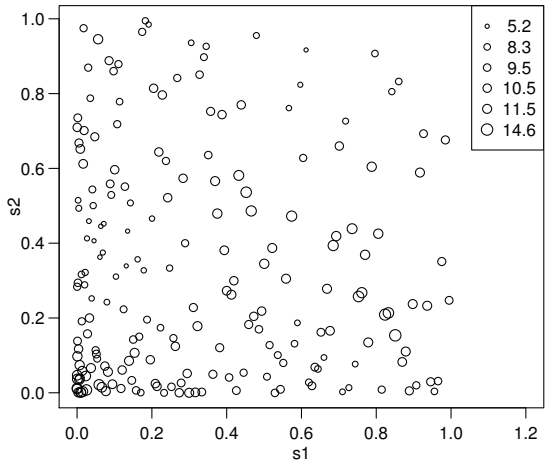

Figure 1.1: Visualization of the simulated data

```
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)
```

1.3 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 previous 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. Also, we can have closed expressions for some parameters.

To get good performance on the maximum likelihood estimation, we can write the variance parameter of the noise, σ_e^2, in terms of the relative one, $r = \sigma_e^2/\sigma_x^2$, such that

$$Cov(\boldsymbol{y}) = \sigma_x^2(r\boldsymbol{I} + \boldsymbol{C})$$

where $\boldsymbol{C}$ is the correlation matrix.

Under Gaussian likelihood we have that the mean and variance are independently. Taking the derivative with respect to β we get

$$(\boldsymbol{F}^{'}\boldsymbol{W}\boldsymbol{F})\hat{\beta} = \boldsymbol{F}^{'}\boldsymbol{W}\boldsymbol{y}$$

where $\boldsymbol{W} = (r\boldsymbol{I} + \boldsymbol{C})^{-1}$. With this parametrization we can derive with respect to σ_x^2 and have

$$\hat{\sigma}_x^2 = (y_i - \boldsymbol{F}\hat{\beta})^{'}\boldsymbol{W}(y_i - \boldsymbol{F}\hat{\beta})/n \ .$$

Now, we can work with a concentrated log-likelihood, that is a function of the noise relative variance r, the scale κ and the smoothness parameter ν

$$-\frac{1}{2}(|\boldsymbol{W}| + n(\log(2\pi\hat{\sigma}_x^2) + 1))$$

We use the quasi-Newton methods to find the maximum likelihood estimates of these three parameters.

The likelihood function is just the the multivariate normal density, considering the mean as a regression with identity link and the covariance with Matérn covariance plus the nugget effect. In the implemented function (bellow), we have the also that the matrix of distances between the points because are one of the arguments, because it is not necessary to recalculate at each interaction of the minimization algorithm.

We use the negative of the logarithm of the likelihood function because, by default, the algorithm find the minimum of a function. In the computation of the likelihood, we use the Cholesky factorization to compute the determinant and also the inverse of the covariance matrix.

```
nllf <- function(pars, ff, y, m) {
    m <- 2^(1-pars[3])*(pars[2]*m)^pars[3]*
      besselK(m*pars[2],pars[3])/gamma(pars[3])
    diag(m) <- 1 + pars[1]
    m <- chol(m)
    ldet.5 <- sum(log(diag(m)))
    m <- chol2inv(m)
    beta <- solve(crossprod(ff, m)%*%ff,
                  crossprod(ff, m)%*%y)
    z <- y-ff%*%beta
    s2x.hat <- mean(crossprod(m,z)*z)
    res <- ldet.5 + nrow(m)*(1+log(2*pi*s2x.hat))/2
    attr(res, 'param') <- ### to return the parameters together
        c(beta=beta, s2e=pars[1]*s2x.hat,
          s2x=s2x.hat, kappa=pars[2], nu=pars[3])
    return(res)
  }
```

The implemented function is a function of the three length parameters vector (r, κ and ν), the covariate (design) matrix, the data and the matrix of distances:

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

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

```
[1] 281.6389
attr(,"param")
     beta       s2e       s2x     kappa        nu
9.4656790 0.2715257 4.5254278 7.0000000 1.0000000
```

and at another set of the values

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

```
[1] 394.9953
attr(,"param")
     beta       s2e       s2x     kappa        nu
 8.899256  0.000000 35.641913  7.000000  1.000000
```

We get the maximum likelihood estimates with 'L-BFGS-B' method implemented on `optim()` function. The maximum likelihood estimates for r, κ and ν are

```
(ores <- optim(c(sigma2e/sigma2x, kappa, nu), nllf, hessian=TRUE,
                ff=matrix(1,n), y=y1, m=as.matrix(dmat),
                method='L-BFGS-B', lower=rep(1e-5,3)))$par
```

```
[1] 0.08628548 9.40307960 1.08654743
```

and we got all the estimated parameters just evaluating our concentrated log-likelihood again to get all estimatives

```
(lkhat <- attr(nllf(ores$par, matrix(1,n),
                    y1, as.matrix(dmat)), 'param'))
```

```
     beta       s2e       s2x     kappa        nu
9.5457270 0.2824723 3.2736940 9.4030796 1.0865474
```

This solution by likelihood is to show how it works. In the **geoR** package, we have functions to do simulations and also to perform likelihood estimation. The difference is that the **geoR** package uses $\phi = 1/\kappa$ for the scale parameter and it calls kappa for the smoothness parameter. Also σ_e^2 is called `tausq`

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. With this function we also got the maximum likelihood estimates for the smoothness parameter 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
```

If we fix the smoothness parameter ν on the value, we 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 likelihood here has similar value than previous, just because the previously estimated value of ν is close to the fixed value here.

Chapter 2

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 shuch one that found a explicit link between the an stochastic partial differential equation (SPDE) and the Matérn Gaussian random fields, [Lindgren et al., 2011]. The solution of the that SPDE thought the Finite Element Method (FEM) provides a explicit link between GF to GMRF.

2.1 The [Lindgren et al., 2011] results

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

$$E(x_{ij}|x_{-ij}) = \frac{1}{a}(x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}) \tag{2.1}$$

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

$$\begin{matrix} -1 & \\ a & -1 \end{matrix} \tag{2.2}$$

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

$$(\kappa^2 - \Delta)^{\alpha/2} x(\boldsymbol{u}) = \boldsymbol{W}(\boldsymbol{u}), \quad \boldsymbol{u} \in \Re^d, \quad \alpha = \nu + d/2, \ \ \kappa > 0, \ \ \nu > 0, \tag{2.3}$$

[Lindgren et al., 2011] show that for $\nu = 1$ and $\nu = 2$ the GMRF representations are convolutions of (2.2). So, for $\nu = 1$, in that representation we have:

$$\begin{matrix} 1 & & \\ -2a & 2 & \\ 4 + a^2 & -2a & 1 \end{matrix} \tag{2.4}$$

and, for $\nu = 2$:

$$\begin{matrix} -1 & & & \\ 3a & -3 & & \\ -3(a^2+3) & 6a & 3 & \\ a(a^2+12) & -3(a^2+3) & 3a & -1 \end{matrix} \tag{2.5}$$

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

$$C_{i,j} = \langle \psi_i, \psi_j \rangle, \quad G_{i,j} = \langle \nabla\psi_i, \nabla\psi_j \rangle, \quad (\boldsymbol{K}_{\kappa^2})_{i,j} = \kappa^2 C_{i,j} + G_{i,j} \tag{2.6}$$

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

$$\begin{aligned} \boldsymbol{Q}_{1,\kappa^2} &= \boldsymbol{K}_{\kappa^2}, \\ \boldsymbol{Q}_{2,\kappa^2} &= \boldsymbol{K}_{\kappa^2}\boldsymbol{C}^{-1}\boldsymbol{K}_{\kappa^2}, \\ \boldsymbol{Q}_{\alpha,\kappa^2} &= \boldsymbol{K}_{\kappa^2}\boldsymbol{C}^{-1}Q_{\alpha-2,\kappa^2}\boldsymbol{C}^{-1}\boldsymbol{K}_{\kappa^2}, \quad \text{for } \alpha = 3, 4, \ldots . \end{aligned} \tag{2.7}$$

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

$$\boldsymbol{Q} = \sum_{i=0}^{p} b_i \boldsymbol{C}(\boldsymbol{C}^{-1}\boldsymbol{G})^i. \tag{2.8}$$

For $\alpha = 1$ and $\alpha = 2$ we have the (2.7). 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.

2.2 The triangulation

The first step to fit the model is the construction of the 'mesh'. This step must be made VERY CAREFULLY. It is similar to choosing the integration points on a numeric ingration algorithm. We need regular ones? How many points is needed?

Additionally, here we need, yet, to add, ALSO CAREFULLY, additional points around the boundary, the outer extension. It is necessary to avoid the boundary effect were we have variance twice at border than that one within domain [Lindgren, 2012]. For more about it please see [Lindgren and Rue, 2013].

2.2.1 Getting started

For two dimentional mesh, we have a main function `inla.mesh.2d()` that is recommended to use for building a mesh. This function creates the Constrained Refined Delaunay Triangulation (CRDT) that we just call mesh. There are a several options on is function:

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

```
function (loc = NULL, loc.domain = NULL, offset = NULL, n = NULL,
    boundary = NULL, interior = NULL, max.edge, min.angle = NULL,
    cutoff = 0, plot.delay = NULL)
NULL
```

We need some reference about the study region. It is possible to be provided by the location points, supplied on the `loc` argument, or by a location domain (a polygon for example). If we supply the points location, or the domain is supplied `loc.domain` argument, the algorithm find a convex hull mesh. A non convex hull mesh can be made when we provide a (list of) set of polygons on the `boundary` argument, each elements of this list of `inla.mesh.segment()` class.

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, it is a scalar or length two vector. This argument is a numeric on the same scale unit of the coordinates.

The another arguments are used on as additional conditions. The `offset` is a numeric, or length two vector. If negative, interpreted as a factor relative to the approximate data diameter. If positive, is the extension distance on same scale unit to the coordinates provided.

The argument `n` is the initial number of points on the extended boundary. The `interior` is a list of segments to specify interior constraints, each one of `inla.mesh.segment` class. A good mesh need to have triangles more regular as possible, on size and shape. To help this requeriment, upward the `max.edge`, we have the `min.angle` argument, that can be scalar or length two vector, to specify the minimum internal angles of the triangles on the inner domain and on the outer extension. This values at most 21 guarantee the convergence of the algorithm.

Still on the shape of the triangles, we also have the `cutoff` argument that is the minimum allowed distance between points. It means that points at with less distance than the supplied value are replaced by a single vertex. So, it avoids small triangles and must be provided a positive number, specially when we have some very close points, on points location or on the domain.

To understand how this function works, we apply this function, varying some arguments, to first five locations of the toy dataset.

```
data(SPDEtoy)
coords <- as.matrix(SPDEtoy[,1:2]) ;   p5 <- coords[1:5,]
```

We also build some meshes using the domain and not the points, we define the domain with

```
pl.dom <- cbind(c(0,1,1,0.7,0), c(0,0,0.7,1,1))
```

Creating some meshes for the first five points:

```
m1 <- inla.mesh.2d(p5, max.edge=c(0.5, 0.5))
m2 <- inla.mesh.2d(p5, max.edge=c(0.5, 0.5), cutoff=0.1)
m3 <- inla.mesh.2d(p5, max.edge=c(0.1, 0.5), cutoff=0.1)
m4 <- inla.mesh.2d(p5, max.edge=c(0.1, 0.5), offset=c(0,-0.65))
m5 <- inla.mesh.2d(, pl.dom, max.edge=c(0.3, 0.5), offset=c(0.03, 0.5))
m6 <- inla.mesh.2d(, pl.dom, max.edge=c(0.3, 0.5), offset=c(0.03, 0.5), cutoff=0.1)
m7 <- inla.mesh.2d(, pl.dom, max.edge=c(0.3, 0.5), n=5, offset=c(.05,.1))
m8 <- inla.mesh.2d(, pl.dom, max.edge=c(.3, 0.5), n=7, offset=c(.01,.3))
m9 <- inla.mesh.2d(, pl.dom, max.edge=c(.3, 0.5), n=4, offset=c(.05,.3))
```

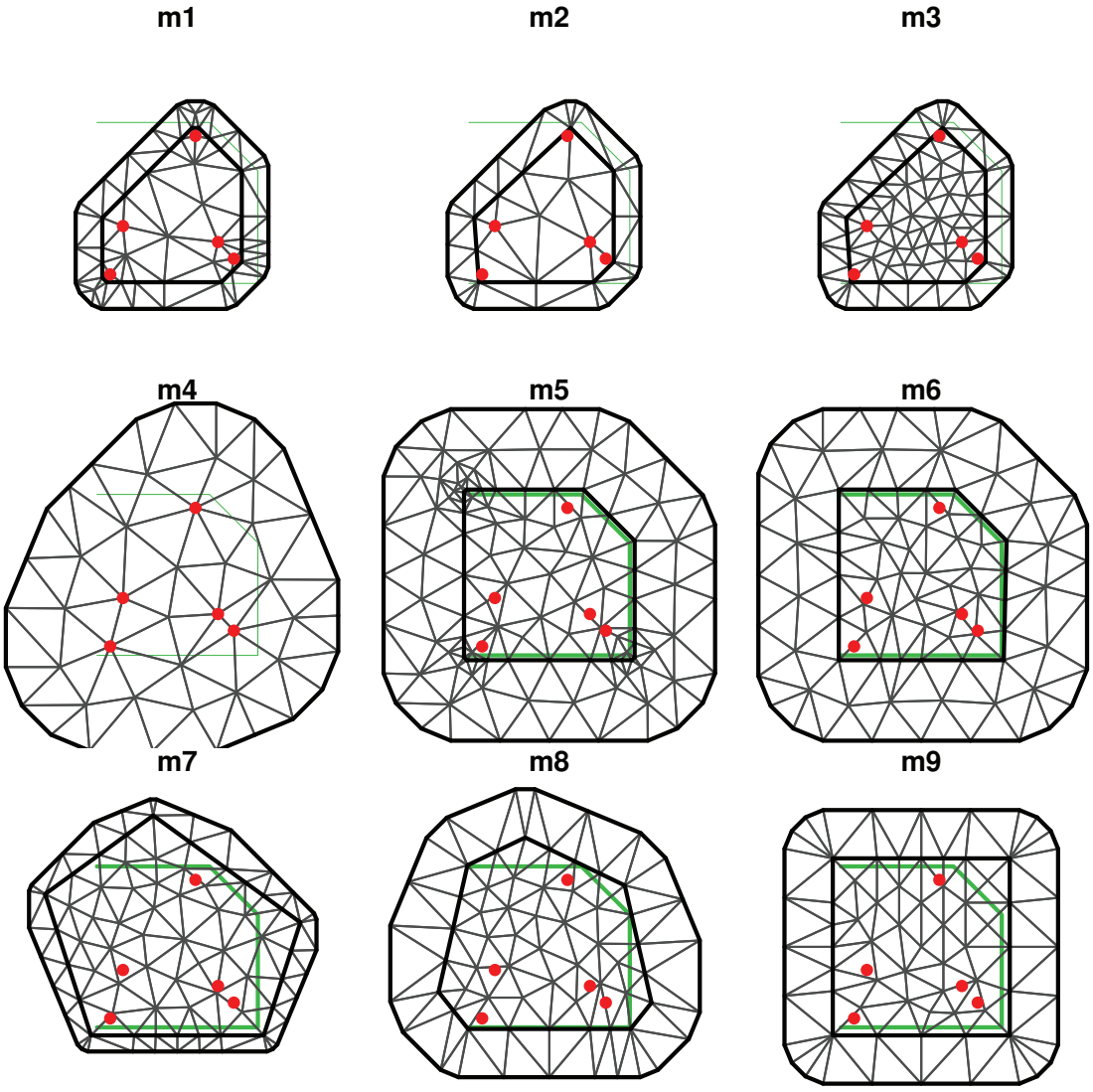

Figure 2.1: Triangulation with different restrictions.

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

```
par(mfrow=c(3, 3), mar=c(0,0,1,0))
for (i in 1:9) {
    plot(pl.dom, type='l', col=3, lwd=2*(i>4), xlim=c(-0.57,1.57),
         main = paste('m',i,sep=''), asp=1, axes=FALSE)
    plot(get(paste('m', i, sep='')), add=TRUE)
    points(p5, pch=19, col=2)
  }
```

In the `m1` mesh has two main problems: 1) some triangles with small inner angles, 2) some large triangles on inner domain. In the `m2` mesh, we relax the restriction on the locations, because points with distance less than the cutoff are considered single vertice. This avoids some of the triangles (at bottom right side) with small angles on the previous one. So the **cutoff is a VERY GOOD idea**! The `m3` mesh on the top right has its inner triangles has less than 0.1 and this mesh looks better than the two previous.

The `m4` was made with no first convex hull extension around the points. It has just the second one. In this case, the length of inner triangles does not works (first value on `max.edge` argument) and we have just triangles with length edges at most 0.5. The shapes of the triangles looks good, except for these ones with vertices including the two points at

bottom rigth side.

The m5 mesh is made just using the domain polygon and it has shape similar to the domain area. In this mesh we have some small triangles at corners just due the fact that is is build without cutoff. Also, we have a (relatively) small first extension and a (relatively) large second one. On the m6 mesh we have add the a cutoff and get a better mesh than the previous one.

In the last tree meshes we change the initial number of extension points. Its can be usefull to change when we have some polygons domain to get convergence. Here we show the shape of the final mesh that we got with, for example, n=5 on the m7 mesh, This number produces a mesh that seems not adequate for this domain because we have non uniform exension behind the border. The m9 mesh has very bad triangles shapes.

The object returned by inla.mesh.2d() function has class inla.mesh and contains a list of things:

```
 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, m7$n, m8$n, m9$n)

[1]  61  36  79  47 117  88  87  69  72
```

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] 61 61
```

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

```
 m1$idx$loc

[1] 24 25 26 27 28
```

2.2.2 Non-convex hull meshes

All the meshes on Figure 2.1 are made to have a convex hull boundary. A convex hull is a polygon triangles out of the domain area, the extension made to avoid the boundary effect. A triangulation without additional border can be made by supplying the boundary argument instead the location or loc.domain argument. One way is to build a boundary for the points and supply it on boundary argument. But, in this case we need a boundary

We can build boundaries using the inla.nonconvex.hull() function

```
 args(inla.nonconvex.hull)

function (points, convex = -0.15, concave = convex, resolution = 40,
    eps = NULL)
NULL
```

In this function we provide the points set and some constraint arguments. We are able to control the shape of the boundary by its convexity, concavity and resolution. We make some boundaries and build a mesh with each one to understand better it.

```
bound1 <- inla.nonconvex.hull(p5)
bound2 <- inla.nonconvex.hull(p5, convex=0.5, concave=-0.15)
bound3 <- inla.nonconvex.hull(p5, concave=0.5)
bound4 <- inla.nonconvex.hull(p5, concave=0.5, resolution=c(20, 20))
m10 <- inla.mesh.2d(boundary=bound1, cutoff=0.05, max.edge=c(.1,.2))
m11 <- inla.mesh.2d(boundary=bound2, cutoff=0.05, max.edge=c(.1,.2))
m12 <- inla.mesh.2d(boundary=bound3, cutoff=0.05, max.edge=c(.1,.2))
m13 <- inla.mesh.2d(boundary=bound4, cutoff=0.05, max.edge=c(.1,.2))
```

These meshes are visualized on Figure 2.2 by commands bellow

```
par(mfrow=c(2,2), mar=c(0,0,1,0))
for (i in 10:13) {
    plot(get(paste('m', i, sep='')), asp=1, main='')
    points(p5, pch=19, col=2); title(main=paste('m', i, sep=''))
  }
```

The `m10` mesh is builded with a boundary that we got with default arguments on `inla.nonconvex.hull()` function. The default `convex` and `concave` arguments are both equal `0.15` proportion of the points domain radius, that is computed by

```
max(diff(range(p5[,1])), diff(range(p5[,2])))*.15
```

```
[1] 0.12906
```

If we supply larger convex value, like the other used to generate the `m11`, we got a larger boundary. It's because all circles with centre on each point and radius less than the convex value are incide to the boundary. When we choose a larger concave value, the boundary used to the 'm12' and 'm13' meshes, we don't have circles with radius less than the concave value outside the boudary. If we choose a smaller resolution, we got a boundary with small resolution (in terms of number of points), for example, comparing the `m12` and `m13` meshes.

2.2.3 Meshes for the toy example

To analyze the toy data set, we use six triangulations options to make comparisons on section 3.5. The first mesh forces the location points as vertices of the mesh.

```
mesh1 <- inla.mesh.2d(coords, max.edge=c(0.035, 0.1))
mesh2 <- inla.mesh.2d(coords, max.edge=c(0.15, 0.2))
```

The second and third meshes are based on the points, but we use the cutoff greather than zero to avoid small triangles on regions where we have more density observations

```
mesh3 <- inla.mesh.2d(coords, max.edge=c(0.15, 0.2), cutoff=0.02)
```

We also build three other meshes based on the domain area. These are builded to have aproximately the same number of vertices than previous

```
mesh4 <- inla.mesh.2d(, pl.dom, max.e=c(0.0355, 0.1))
mesh5 <- inla.mesh.2d(, pl.dom, max.e=c(0.092, 0.2))
mesh6 <- inla.mesh.2d(, pl.dom, max.e=c(0.11, 0.2))
```

The number of nodes on each one of these meshes are

```
c(mesh1$n, mesh2$n, mesh3$n, mesh4$n, mesh5$n, mesh6$n)
```

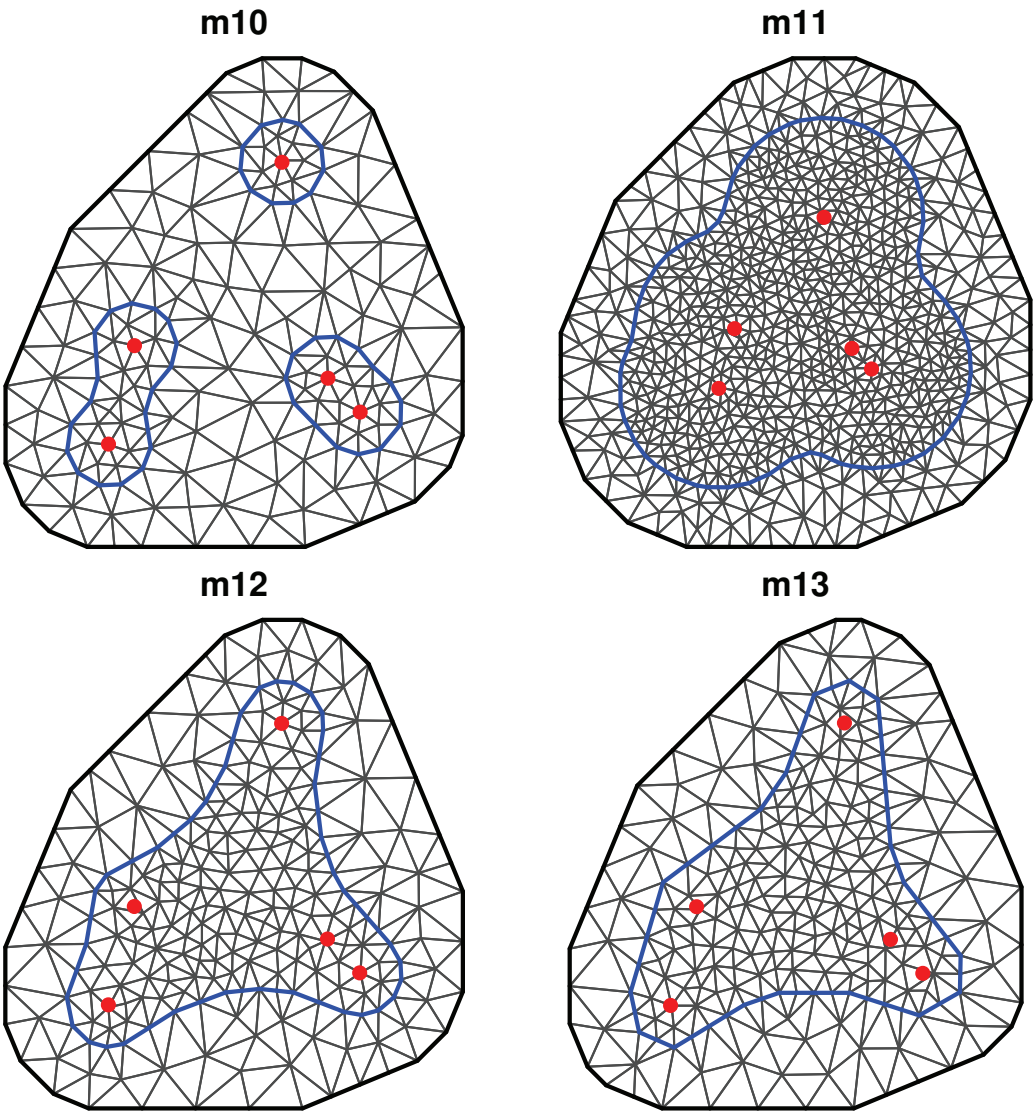

Figure 2.2: Non-convex meshes with different boundaries.

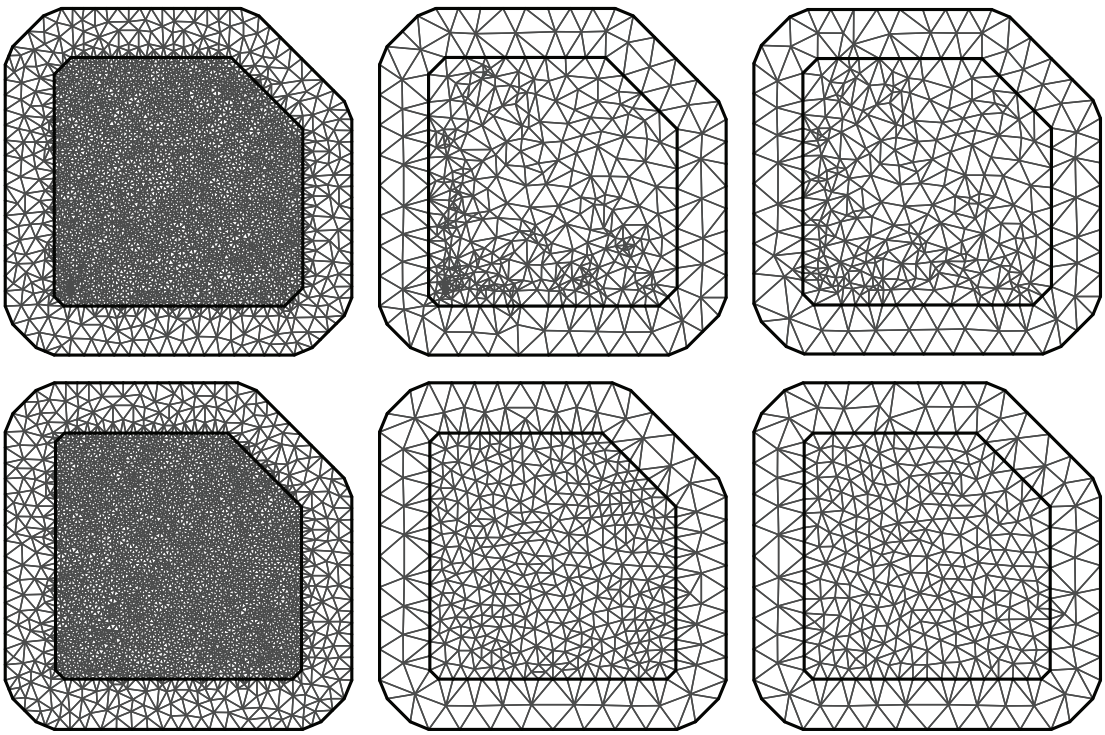

Figure 2.3: Six triangulation options for the toy example.

```
[1] 2900  488  371 2878  490  375
```

These six meshes are showed on Figure 2.3 with code below

```
par(mfrow=c(2,3), mar=c(0,0,0,0))
for (i in 1:6)
    plot(get(paste('mesh',i,sep='')), asp=1, main='')
```

2.2.4 Meshes for Paraná state

We have some examples using data collected on Paraná state, in Brasil. In this case we need to take into account two things: one is the shape of these domain area and other is the sistem of coordinates reference.

We have the daily rainfall data

```
data(PRprec)
dim(PRprec)
```

```
[1] 616 368
```

```
PRprec[1:3, 1:10]
```

```
  Longitude Latitude Altitude d0101 d0102 d0103 d0104 d0105 d0106 d0107
1  -50.8744 -22.8511      365     0     0     0   0.0     0     0   2.5
3  -50.7711 -22.9597      344     0     1     0   0.0     0     0   6.0
4  -50.6497 -22.9500      904     0     0     0   3.3     0     0   5.1
```

that consists of the dayli rainfall data from 616 stations for each day of the 2012 year. The coordinates (two first colums) are on the latlong projection.

Also, we have the Paraná state polygon with

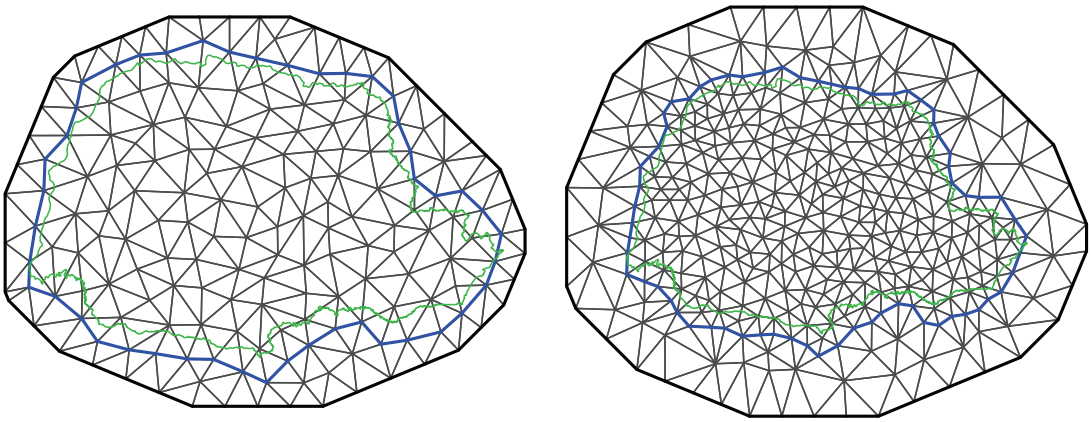

Figure 2.4: Mesh for Paraná state

```
data(PRborder)
dim(PRborder)
```

```
[1] 2055    2
```

that consists of a set of 2055 points on the latlong projection.

In this case is best to use a non-convex hull mesh. We start it with building a non-convex domain with

```
prdomain <- inla.nonconvex.hull(as.matrix(PRprec[,1:2]),
                                -0.03, -0.05, resolution=c(100,100))
```

with this defined domaind we build two meshes with different resolution (max edge length) on the inner domain

```
(prmesh1 <- inla.mesh.2d(boundary=prdomain, max.edge=c(.7,.7),
                         cutoff=0.35, offset=c(-0.05, -0.05)))$n
```

```
[1] 187
```

```
(prmesh2 <- inla.mesh.2d(boundary=prdomain, max.edge=c(.45,1), cutoff=0.2))$n
```

```
[1] 382
```

We can visualize this both meshes on the Figure 5.2 with commands bellow

```
par(mfrow=c(1,2), mar=c(0,0,0,0))
plot(prmesh1, asp=1, main='');   lines(PRborder, col=3)
plot(prmesh2, asp=1, main='');   lines(PRborder, col=3)
```

2.2.5 Triangulation with a SpatialPolygonsDataFrame

Suppose that we have a map of the domain region. In **R** the representation of an spatial object is made by object classes on **sp** package, see [Pebesma and Bivand, 2005] and [Bivand et al., 2008]. To show an aplication in this case, we use the North Caroline map, on package **spdep**, [Bivand et al., 2012].

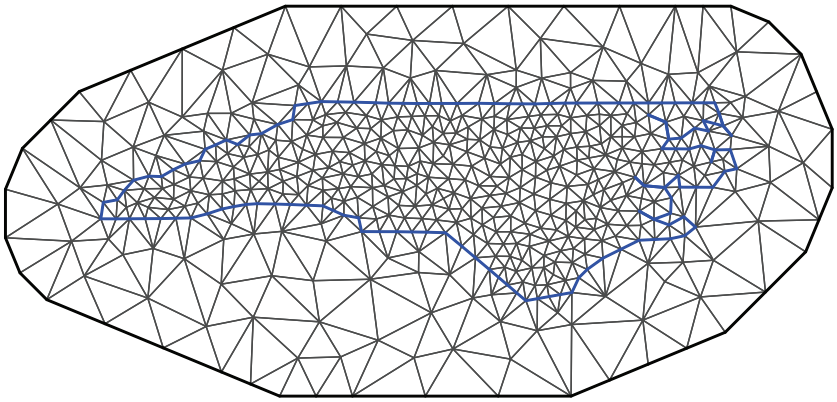

Figure 2.5: Mesh constructed using the North Carolina map

```
require(maptools)
nc.fl <- system.file("etc/shapes/sids.shp", package="spdep")[1]
nc.sids <- readShapePoly(nc.fl, ID="FIPSNO",
                         proj4string=CRS("+proj=longlat +ellps=clrk66"))
```

We simplify this map uniting all the areas together. To do it, we use the `unionSpatialPolygons()` **spdep** function that uses function of the **rgeos** package [Bivand and Rundel, 2013]. If we don't have the **rgeos** we can use the **gpclib** package, but we need to set a permition before the work with this library

```
gpclibPermit()
```

```
[1] FALSE
```

```
nc.border <- unionSpatialPolygons(nc.sids, rep(1, nrow(nc.sids)))
```

Now, we use the `inla.sp2segment()` to extract the boundary of the `SpatialPolygons` object that contains the border of the map

```
require(INLA)
nc.bdry <- inla.sp2segment(nc.border)
```

and creates the mesh

```
(nc.mesh <- inla.mesh.2d(boundary=nc.bdry, cutoff=0.15,
                         max.edge=c(0.3, 1)))$n
```

```
[1] 534
```

that is visualized on Figure 2.5 with the commands bellow.

```
par(mar=c(0,0,0,0))
plot(nc.mesh, asp=1, main='')
```

2.3 The projector matrix

Because the SPDE model is defined on the mesh, we need an appropriate specification of the linear predictor for the data response. For details on this please see [Lindgren, 2012].

The key is that we have a random field modeled at the mesh vertices, with dimention m, and a response at n locations. So, we need to define how the random field and other model components are linked to the response.

Let the matrix $\boldsymbol{A}$ that links the process on the mesh vertices triangles with the locations response. This matrix is called the projector matrix because it do the projection of the latent field modeled at mesh vertices to data locations.

The projector matrix can be builded with the `inla.spde.make.A` function. Considering that each mesh vertices has a weight, the value for one point within one triangle is the projection of the plane (formed by these three weights) at this point location. This projection is just a weighted average with the weights computed by the `inla.spde.make.A()` function. Using the toy data set and the mesh five we have

```
coords <- as.matrix(SPDEtoy[,1:2])
A5 <- inla.spde.make.A(mesh5, loc=coords)
```

This matrix has dimention equal the number of data locations by the number of vertices on the mesh

```
dim(A5)
```

```
[1] 200 490
```

We have exact three non-zero elemens on each line

```
table(rowSums(A5>0))
```

```
  3
200
```

because each location points are inside one of the triangles. These three non-zero elements on each line sums one

```
table(rowSums(A5))
```

```
  1
200
```

because multiplication with any vector weights at mesh nodes by this matrix is the projection of these weights at the location points.

We have some columns on the projector matrix with all elements equals zero.

```
table(colSums(A5)>0)
```

```
FALSE  TRUE
  242   248
```

This columns corresponds the triangles without any location point inside. This columns that can be droped and the stack functionality (section 3.1) automatically handles this situation.

When we have a mesh where each location points are on the vertice set, the projector matrix has exact one line with nonzero element

```
A1 <- inla.spde.make.A(mesh1, loc=coords)
table(rowSums(A1>0))
```

```
  1
200
```

and all these elements are equals one

```
table(rowSums(A1))
```

```
  1
200
```

because in this case the the projection on the location points are just the weight at the corresponding node (at same location) of the mesh.

2.4 GRF sampling through SPDE

The main thing is that the SPDE make the model on the mesh vertices and we need to make simulation on a set o location points. We start by defining the spde model

```
spde5 <- inla.spde2.matern(mesh5, alpha=2)
```

and build the precision matrix associated.

It can be done by the `inla.spde2.precision()` function. On this function, we have to provide the SPDE model and the parameters θ on the SPDE parametrization. θ is a length two vector where the second element is the logarithim of the scale parameter, $\log(\kappa)$, and the first one is the 'local variance parameter' τ such that the marginal variance σ_x^2 is

$$\sigma_x^2 = \frac{1}{4\pi\tau^2\kappa^2}$$

and we have that

$$\log(\tau) = \theta_1 = -\log(4\pi\kappa^2\sigma_x^2)/2 \; .$$

We do the simulation considering $\kappa = 5$ and $\sigma_x^2 = 1$

```
kappa <- 5;     s2x <- 1
```

so, we have

```
theta <- c(-log(4*pi*s2x*kappa^2), log(kappa))
```

and we have the precision matrix $\boldsymbol{Q}(\theta)$ by

```
Q5 <- inla.spde2.precision(spde5, theta)
```

Now, we can use the `inla.qsample()` function to get a sample at mesh nodes by

```
x5 <- inla.qsample(Q=Q5)
```

and to get a sample at the locations, we just project it with the projector matrix by

```
x <- drop(A5%*%x5)
```

We can have a more general function to generate one or more samples, by defining a function that need just the coordinates and model parameters. A mesh can be provided and if not it is also generated within this function

```
rspde
```

```
function (coords, kappa, variance = 1, alpha = 2, n = 1, verbose = TRUE,
    mesh, return.attributes = TRUE)
{
    t0 <- Sys.time()
    theta <- c(-0.5 * log(4 * pi * variance * kappa^2), log(kappa))
    if (verbose)
```

```
        cat("theta =", theta, "\n")
    if (missing(mesh)) {
        mesh.pars <- c(0.5, 1, 0.1, 0.5, 1)/kappa
        if (verbose)
            cat("mesh.pars =", mesh.pars, "\n")
        attributes <- list(mesh = inla.mesh.2d(, coords[chull(coords),
            ], max.edge = mesh.pars[1:2], cutoff = mesh.pars[3],
            offset = mesh.pars[4:5]))
        if (verbose)
            cat("n.mesh =", attributes$mesh$n, "\n")
    }
    else attributes <- list(mesh = mesh)
    attributes$spde <- inla.spde2.matern(attributes$mesh, alpha = alpha)
    attributes$Q <- inla.spde2.precision(attributes$spde, theta = theta)
    attributes$A <- inla.mesh.project(mesh = attributes$mesh,
        loc = coords)$A
    if (n == 1)
        result <- drop(attributes$A %*% inla.qsample(Q = attributes$Q,
            constr = attributes$spde$f$extraconstr))
    t1 <- Sys.time()
    result <- inla.qsample(n, attributes$Q, constr = attributes$spde$f$extraconstr)
    if (nrow(result) < nrow(attributes$A)) {
        result <- rbind(result, matrix(NA, nrow(attributes$A) -
            nrow(result), ncol(result)))
        dimnames(result)[[1]] <- paste("x", 1:nrow(result), sep = "")
        for (j in 1:ncol(result)) result[, j] <- drop(attributes$A %*%
            result[1:ncol(attributes$A), j])
    }
    else {
        for (j in 1:ncol(result)) result[1:nrow(attributes$A),
            j] <- drop(attributes$A %*% result[, j])
        result <- result[1:nrow(attributes$A), ]
    }
    t2 <- Sys.time()
    attributes$cpu <- c(prep = t1 - t0, sample = t2 - t1, total = t2 -
        t0)
    if (return.attributes)
        attributes(result) <- c(attributes(result), attributes)
    return(drop(result))
}
```

We can generate samples at a milion of locations in some seconds

```
pts1 <- matrix(runif(2e6), ncol=2)
x1 <- rspde(pts1, kappa=5)
```

```
theta = -2.87495 1.609438
mesh.pars = 0.1 0.2 0.02 0.1 0.2
n.mesh = 473
```

Some summary

```
summary(x1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.4970 -1.2790 -0.6940 -0.6674 -0.1061  1.7540
```

```
 table(x1>0)

 FALSE   TRUE
787931 212069
```

and we have the attributes together

```
 names(attributes(x1))

[1] "mesh"  "spde"  "Q"     "A"     "cpu"   "names"

 attr(x1, 'cpu') ### time in seconds

    prep   sample    total
3.240567 2.356873 5.597440

 attr(x1, 'mesh')$n ### number of mesh nodes

[1] 473

 dim(attr(x1, 'Q'))

[1] 473 473

 dim(attr(x1, 'A'))

[1] 1000000     473
```

This function is used on some of next Chapters.

2.5 Maximum likelihood inference

We have that the latent random field x is distributed as

$$x|\boldsymbol{Q} \sim N(0, \boldsymbol{Q}^{-1})$$

where $\boldsymbol{Q} = \boldsymbol{Q}(\theta)$, $\theta = \{\theta_1, \theta_2\}$ with

θ_1=log(τ) and θ_2=log(κ). Under the toy model, whe have an GF plus noise for the observation data. So, we have

$$y|\mu_y, x, \boldsymbol{A}, \boldsymbol{Q}, \sigma_y^2 \sim N(\mu_y + \boldsymbol{A}x, \boldsymbol{I}\sigma_y^2) \; .$$

Integrating out x, we have that

$$y|\mu_y, \boldsymbol{Q}, \boldsymbol{A}, \sigma_y^2 \sim N(\mu_y, (\boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{A}' + \boldsymbol{I}\sigma_y^2)^{-1}) \; .$$

The log-likelihood function, marginalized with respect x, becomes

$$-\frac{n}{2}\log(2\pi) - |\Sigma| - \frac{1}{2}(y - \boldsymbol{F}\beta)^{'}\Sigma^{-1}(y - \boldsymbol{F}\beta)$$

where $\Sigma = (\boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{A}' + \boldsymbol{I}\sigma_y^2)^{-1}$ and $\boldsymbol{Q} = \boldsymbol{Q}(\theta) = \boldsymbol{Q}(\tau, \kappa)$. We choose $\tau^{'}$ to have marginal variance of x equal 1, $\boldsymbol{Q}(\tau^{'}, \kappa)) = \boldsymbol{Q}^{'}$. In this case we write $\boldsymbol{W} = (r\boldsymbol{I} + Q(\tau^{'}, \kappa))$ where $r = \frac{\sigma_e^2}{\sigma_x^2}$ and

$$\Sigma = \sigma_x^2 \boldsymbol{W} \; .$$

Now, we can write a concentrated likelihood deriving the likelihood with respect to σ_x^2 and β. We have that

$$(\boldsymbol{F}^{'}\boldsymbol{W}\boldsymbol{F})\hat{\beta} = \boldsymbol{F}^{'}\boldsymbol{W}\boldsymbol{y} \; .$$

and

$$\hat{\sigma}_x^2 = (y_i - \boldsymbol{F}\hat{\beta})^{'}\boldsymbol{W}(y_i - \boldsymbol{F}\hat{\beta})/n$$

Now, we can work with the log-likelihood as a function of the noise relative variance and the scale parameter κ. The concentrated log-likelihood can be writed as

$$-\frac{1}{2}\{|\boldsymbol{W}| + n[\log(2\pi\hat{\sigma}_x^2) + 1]\}$$

And we can use one of the quasi-Newton optimization algorithim to find the maximum likelihood estimate of each the two parameters on the concentrated likelihood. These parameters are these one the $\boldsymbol{W}$ matrix and are parametrized as $\log(r)$ and $\log(\kappa)$.

To eval the likelihood function efficiently we need to take care on the evaluation of the determinant $|\boldsymbol{W}|$ and of $\boldsymbol{W}$ itself.

$\boldsymbol{W} = (r\boldsymbol{I} + Q(\tau^{'}, \kappa))$

To evaluate the determinant, we use a matrix determinant lemma such that we have

$$\det(r\boldsymbol{I} + \boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^{'}) = \det(\boldsymbol{Q} + \boldsymbol{A}^{'}r\boldsymbol{A})\det(\boldsymbol{Q})\det(r\boldsymbol{I}) \ .$$

We implement the following function to compute it

```
function (U, Q, r)
{
    d1 <- determinant(crossprod(U/r, U) + Q)$modulus
    return(new("numeric", (d1 - determinant(Q)$modulus) + nrow(U) *
        log(r)))
}
```

The precision matrix can be evaluated using the Woodbury matrix identity. It says that for $\boldsymbol{C}$ and $\boldsymbol{W}$ invertible matrices we have

$$(r\boldsymbol{I} + \boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^{'})^{-1} = r^{-1}\boldsymbol{I} - r^{-1}\boldsymbol{I}\boldsymbol{A}(\boldsymbol{Q} + \boldsymbol{A}r^{-1}\boldsymbol{A})^{-1}\boldsymbol{A}r^{-1} \ .$$

We implement it with the following function

```
function (U, Q, r)
{
    Bi <- Diagonal(nrow(U), rep(1/r, nrow(U)))
    VBi <- crossprod(U, Bi)
    R <- solve(Q + VBi %*% U, VBi)
    forceSymmetric(Bi - crossprod(VBi, R))
}
```

Considering θ and $\log(\sigma_y^2)$ the three parameter vetor, we have the following negative of the log likelihood function

```
function (pars, X, A, y, spde)
{
    m <- inla.spde2.precision(spde, c(-pars[2] - 1.265512, pars[2]))
    ldet <- precDetFun(A, m, exp(pars[1]))
    m <- precFun(A, m, exp(pars[1]))
    Xm <- crossprod(X, m)
    betah <- drop(solve(Xm %*% X, Xm %*% y))
    z <- drop(y - X %*% betah)
    s2x.h <- mean(crossprod(m, z) * z)
    return((ldet + nrow(m) * (1 + log(2 * pi * s2x.h)))/2)
}
```

We proceed the maximum likelihood estimation, for the toy example dataset and compare with the results from **geoR** package of previous Chapter. We remember also the true parameter values

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

We call the data

```
data(SPDEtoy)
```

build the mesh (using the true scale parameter to determine the edge lengths and `cutoff`), SPDE model (with $\alpha = 2$) and the projector matrix

```
xy <- as.matrix(SPDEtoy[,1:2])
(mesh <- inla.mesh.2d(xy, max.edge=c(0.5, 1)/kappa,
                          cutoff=0.1/kappa))$n
```

```
[1] 817
```

```
spde <- inla.spde2.matern(mesh)
A <- inla.mesh.project(mesh, loc=xy)$A
```

The maximum likelihood estimation is done using the `optim()` function. Using the default Nelder-Mead method

```
opt <- optim(log(c(sigma2e/sigma2x, kappa)), negLogLikFun, hessian=TRUE,
               X=matrix(1,nrow(xy)), A=A, y=SPDEtoy[,3], spde=spde)
```

We also define a function to map the SPDE model parameters to the σ_x^2 and κ user parameters. The function bellow recieve θ, $\log(\sigma_y^2)$ and β (regression parameters) and return σ_x^2, κ, σ_y^2 and β.

```
function (pars, X, A, y, spde)
{
    m <- inla.spde2.precision(spde, c(-pars[2] - 1.265512, pars[2]))
    m <- precFun(A, m, exp(pars[1]))
    Xm <- crossprod(X, m)
    beta <- drop(solve(Xm %*% X, Xm %*% y))
    z <- drop(y - X %*% beta)
    s2x.h <- mean(crossprod(m, z) * z)
    c(beta = beta, s2e = exp(pars[1]) * s2x.h, s2x = s2x.h, kappa = exp(pars[2]))
}
```

We use this function to compare the results with the results by `geoR` package

```
require(geoR)
system.time(lkf <- likfit(as.geodata(SPDEtoy),
                          ini=c(sigma2x, 1/kappa),
                          kappa=1, ### kappa in geoR is nu
                          nugget=sigma2e))
```

```
---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
```

```
likfit: WARNING: This step can be time demanding!
----------------------------------------------------------------
likfit: end of numerical maximisation.
   user  system elapsed
  1.716   0.016   1.665
```

and have similar results

```
 rbind(spde=par2user(opt$par, matrix(1, nrow(A)), A, SPDEtoy[,3], spde),
         geoR=c(lkf$beta, lkf$nugget, lkf$sigmasq, 1/lkf$phi),
         true=c(beta0, sigma2e, sigma2x, kappa))
```

```
          beta       s2e      s2x    kappa
spde  9.520085 0.2762273 3.194561 7.552244
geoR  9.534878 0.2708729 3.323400 8.653599
true 10.000000 0.3000000 5.000000 7.000000
```

Chapter 3

A toy example

In this section we start to show the fitting process using the SPDE approach, [Lindgren et al., 2011]. This starter is by fitting a toy geostatistical model: Gaussian response without covariate. We use the Bayesian approach and found the posterior marginal distributions using the Integrated Nested Laplace Approximation - INLA, [Rue et al., 2009] implemented on the **INLA R** package. The ideas for application of the SPDE approach using the **INLA** package are well described on [Lindgren, 2012] and on [Lindgren and Rue, 2013].

The dataset used are a tree column `data.frame` simulated on the previous section and provided on **INLA** package. It can be called by

```
data(SPDEtoy)
```

this is a `data.frame` where the two first columns are the coordinates and the third is the response simulated at this locations

```
str(SPDEtoy)
```

```
'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

$$\begin{aligned} y_i|\beta_0, x_i, \sigma_e^2 &\sim N(\beta_0 + x_i, \sigma_e^2) \\ \boldsymbol{x} &\sim GF(0, \Sigma) \end{aligned} \tag{3.1}$$

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.

To the estimation mesh we need to define the mesh. We show it on Chapter 2.2 and here we use the fift mesh builded for the toy example on Chapter 2.2

With that mesh, we define the SPDE model using the function `inla.spde2.matern()`

```
args(inla.spde2.matern)
```

```
function (mesh, alpha = 2, param = NULL, constr = FALSE, extraconstr.int = NULL,
    extraconstr = NULL, fractional.method = c("parsimonious",
        "null"), B.tau = matrix(c(0, 1, 0), 1, 3), B.kappa = matrix(c(0,
        0, 1), 1, 3), prior.variance.nominal = 1, prior.range.nominal = NULL,
```

```
    prior.tau = NULL, prior.kappa = NULL, theta.prior.mean = NULL,
    theta.prior.prec = 0.1)
NULL
```

The principal arguments are the mesh object and the α parameter, related to the smoothness parameter of the process.

The toy dataset was simulated with $\alpha = 2$ and we use this value here (with the mesh five)

```
spde5 <- inla.spde2.matern(mesh5, alpha=2)
```

Also, from section 2.3 we define the projector matrix

```
coords <- as.matrix(SPDEtoy[,1:2])
A5 <- inla.spde.make.A(mesh5, loc=coords)
```

3.1 The stack functionality

The stack functionality is a very useful functionality to work with SPDE on **INLA** package. This allow to fit more general SPDE models, such as replicated or correlated ones, or more general models that includes more than one projector matrix. Using it we avoid errors in the index construction, [Lindgren, 2012]. Examples on more complex models, with the details, can be found on [Lindgren, 2012], [Cameletti et al., 2012] and [Lindgren and Rue, 2013].

In a section on the previous Chapter we define the projector matrix to project the latent field on the response locations. If we have have covariates measured at same locations (with same dimention of the response), we need a more general definition of the linear predictor. On the toy example, we have to include the effect of the random field and the intercept that is treated as a covariate. So, we define the new intercept η^* as

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

that is a sum of two components each one can be represented as a product of a projector matrix and an effect.

We have that the SPDE approach defines a model on the mesh nodes, and usually the number of nodes are not equal to the number of locations where we have data observed. The `inla.stack` function allow us to work with predictors that includes terms with different dimentions. The three main `inla.stack()` arguments are the `data` vectors list, a list of projector matrices (each one related to one block effect) and the effects.

We need two projector matrices, the projector matrix for the latend field and a matrix to map one-to-one the 'covariate' and the response. This last one can be just a constant instead a diagonal matrix. So, we have

```
stk5 <- inla.stack(data=list(resp=SPDEtoy$y), A=list(A5,1),
                   effects=list(i=1:spde5$n.spde,
                     m=rep(1,nrow(SPDEtoy))), tag='est')
```

The `inla.stack()` function automatically eliminates the elements when any column of each projector matrix has zero sum, generating a correspondent simplified projector matrix. The `inla.stack.A()` extracts a simplified predictor matrix to use on the `inla()` function and the `inla.stack.data()` function extract the correspondent organized data.

The simplified projector matrix from the stack is the binded by column the simplified projectors matrices from each effect block. So, in this case we have

```
dim(inla.stack.A(stk5))
```

```
[1] 200 249
```

one columns more than the number of columns with non null elements of the projector matrix.

3.2 Model fitting and some results

To fit the model need to remove the intercept from the formulae and add it as a covariate term, because we a have projector matrix that allows it as a covariate effect. Of course, we must have to pass these predictor matrix on `control.predictor` argument of the `inla` function

```
res5 <- inla(resp ~ 0 + m + f(i, model=spde5),
               data=inla.stack.data(stk5),
               control.predictor=list(A=inla.stack.A(stk5)))
```

An object from `inla()` function has a set of several results. These includes summaries, marginal posterior densities of each parameter on the model: the regression parameters, each element of the latent field and all the hyperparameters.

The summary of β_0 is obtained by

```
res5$summary.fix
```

```
      mean        sd 0.025quant 0.5quant 0.975quant     mode          kld
m 9.515102 0.7320211   7.978098  9.53028   10.96508 9.555105 4.473199e-10
```

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

```
res5$summary.hy[1,]
```

```
      mean         sd 0.025quant   0.5quant 0.975quant       mode
 2.7452552  0.4297739  1.9826594  2.7187301  3.6650041  2.6716887
```

A marginal distribution on `inla()` output is just two vector, where one represents the parameter values and another the density. Any posterior marginal can be transformed. If we want the posterior marginal for σ_e, the square root of the σ_e^2, we use

```
post.se <- inla.tmarginal(function(x) sqrt(1/x),
                          res5$marginals.hy[[1]])
```

and now we are able to summarize this distribution

```
inla.emarginal(function(x) x, post.se)
```

```
[1] 0.6090609
```

```
inla.qmarginal(c(0.025, 0.5, 0.975), post.se)
```

```
[1] 0.5229069 0.6063310 0.7089318
```

```
inla.hpdmarginal(0.95, post.se)
```

```
                low      high
level:0.95 0.5186903 0.7038194
```

```
inla.pmarginal(c(0.5, 0.7), post.se)
```

```
[1] 0.004553421 0.964046183
```

and, of course, we can visualize it.

The parameters of the latent field is parametrized as $log(\kappa)$ and $log(\tau)$, where τ is the local variance parameter. We have the posterior marginals for κ, σ_x^2 and for the nominal range (the distance that we have correlation equals 0.1). This can be done with the `inla.spde2.result` function

```
res5.field <- inla.spde2.result(res5, 'i', spde5, do.transf=TRUE)
```

and we get the posterior mean of each of these parameters by

```
inla.emarginal(function(x) x, res5.field$marginals.kappa[[1]])
```

```
[1] 7.892637
```

```
inla.emarginal(function(x) x, res5.field$marginals.variance.nominal[[1]])
```

```
[1] 3.874041
```

```
inla.emarginal(function(x) x, res5.field$marginals.range.nominal[[1]])
```

```
[1] 0.3774963
```

also we can get other summary statistics, HPD interval and visualize it.

3.3 Prediction of the random field

A very common objective when we have spatial data collected on some locations is the prediction on a fine grid to get hight resolution maps. 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), (0.5,0.55), (0.7,0.9).

```
pts3 <- rbind(c(.1,.1), c(.5,.55), c(.7,.9))
```

3.3.1 Jointly with the estimation process

The prediction of the random field joint the parameter estimation process in Bayesian inference is the common approach. This approach is made by the computation of the marginal posterior distribution of the random field at 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.

The predictor matrix for the target locations is

```
dim(A5pts3 <- inla.spde.make.A(mesh5, loc=pts3))
```

```
[1]   3 490
```

We can show the columns with non-zero elements of this matrix

```
(jj3 <- which(colSums(A5pts3)>0))
```

```
[1]  85  89 153 162 197 242 244 285 290
```

```
round(A5pts3[, jj3],3)
```

```
3 x 9 sparse Matrix of class "dgCMatrix"

[1,] .     .     0.094 .     .     .     0.513 0.393 .
[2,] 0.219 .     .     0.324 .     0.458 .     .     .
[3,] .     0.119 .     .     0.268 .     .     .     0.612
```

We have to define a data stack for the prediction and join it with the data stack of the observations. The prediction data stack contais the effect set, predictor matrices and assign NA to response

```
stk5p.rf <- inla.stack(data=list(resp=NA), A=list(A5pts3),
                       effects=list(i=1:spde5$n.spde), tag='prd5r')
```

Also, we join both stacks by

```
stk5.jp <- inla.stack(stk5, stk5p.rf)
```

and fit the model again with the full stack setting `compute=TRUE` on `control.predictor`

```
res5p <- inla(resp ~ 0 + m + f(i, model=spde5),
              data=inla.stack.data(stk5.jp),
              control.predictor=list(A=inla.stack.A(stk5.jp), compute=TRUE))
```

To access the posterior marginal distribution of the random field at the target locations, we extract the index from the full stack using the adequate `tag`.

```
(indd5p <- inla.stack.index(stk5.jp, tag='prd5r')$data)
```

```
[1] 201 202 203
```

The summary of the posterior distributions of the random field on the target locations is

```
round(res5p$summary.linear.pred[indd5p,], 4)
```

```
                  mean     sd 0.025quant 0.5quant 0.975quant    mode kld
Apredictor.201  0.1682 0.8113    -1.4012   0.1479     1.8608  0.1143   0
Apredictor.202  3.0478 0.9606     1.1771   3.0366     4.9875  3.0167   0
Apredictor.203 -2.7851 1.2083    -5.1877  -2.7781    -0.4230 -2.7650   0
```

that includes the posterior mean, standard deviation, quantiles and mode.

Because it is a full bayesian analysis, we also we have the marginal distributions. We extract the marginals posterior distributions with

```
marg3 <- res5p$marginals.linear[indd5p]
```

and get the 95% HPD interval for the random field at the second target location by

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

```
                low     high
level:0.95 1.151357 4.959367
```

and see that around the point (0.5,0.5) the random field has positive values, see Figure 3.1.

3.3.2 After the estimation process

If we need just the prediction we can do the prediction after the estimation process with a very small computational cost. It is just a matrix operation in way that we just project the posterior mean of the the random field on mesh nodes to target locations, using the correspondent projector matrix.

So, we 'project' the posterior mean of the latend random field to the target locations by

```
drop(A5pts3%*%res5$summary.random$i$mean)
```

```
[1]  0.167959  3.047722 -2.785755
```

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

```
inla.mesh.project(inla.mesh.projector(mesh5, loc=pts3),
                  res5$summary.random$i$mean)
```

```
[1]  0.167959  3.047722 -2.785755
```

and see that for the mean we have similar values than those on previous section.

Also, we can get the standard deviation

```
drop(A5pts3%*%res5$summary.random$i$sd)
```

```
[1] 0.8944447 1.1129519 1.3729666
```

and we have a little difference.

```
sqrt(drop((A5pts3^2)%*%(res5$summary.random$i$sd^2)))
```

```
[1] 0.5895172 0.6845791 0.9341301
```

3.3.3 Projection on a grid

The approach by the projection of the posterior mean random field is computationaly cheap. So, it can be used to get the map of the random field on a fine grid. The `inla.mesh.projector()` function get the projector matrix automatically for a grid of points over a square that contains the mesh.

To get projection on a grid at the domain $(0,1) \times (0,1)$ we just inform these limits

```
pgrid0 <- inla.mesh.projector(mesh5, xlim=0:1, ylim=0:1, dims=c(101,101))
```

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

```
prd0.m <- inla.mesh.project(pgrid0,  res5$summary.ran$i$mean)
prd0.s <- inla.mesh.project(pgrid0,  res5$summary.ran$i$s)
```

We visualize this values projected on the grid on Figure 3.1.

3.4 Prediction of the response

Another commom result that we want on spatially continuous modelling is the prediction of the response on a target locations that we don't have data observed. 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.

3.4.1 By the posterior distribution

In this case, we want to define a adequate predictor of the response and build the model again. 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

```
stk5.presp <- inla.stack(data=list(resp=NA), A=list(A5pts3,1),
                         effects=list(i=1:spde5$n.spde, m=rep(1,3)),
                         tag='prd5.resp')
```

and join with the data stack to build the model again

```
stk5.full <- inla.stack(stk5, stk5.presp)
r5presp <- inla(resp ~ 0 + m + f(i, model=spde5),
                data=inla.stack.data(stk5.full),
                control.predictor=list(A=inla.stack.A(stk5.full), 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(stk5.full, 'prd5.resp')$data)
```

```
[1] 201 202 203
```

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

```
round(r5presp$summary.fitted.values[indd3r,], 3)
```

```
                        mean    sd 0.025quant 0.5quant 0.975quant   mode
fitted.Apredictor.201  9.683 0.343      9.012    9.682     10.358  9.681
fitted.Apredictor.202 12.563 0.626     11.336   12.561     13.796 12.559
fitted.Apredictor.203  6.730 0.990      4.794    6.726      8.688  6.719
```

Also, we extract the marginals posterior distributions with

```
marg3r <- r5presp$marginals.fitted.values[indd3r]
```

and get the 95% HPD interval for the response at second target location by

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

```
                low     high
level:0.95 11.33153 13.79031
```

and see that around the point (0.5,0.5) we have the values of the response significantly larger than β_0, see Figure 3.1.

3.4.2 By sum of linear predictor components

A computational cheap approach is to (naively) sum the projected posterior mean to the regression term. 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.

If there are covariates, the prediction also can be made in similar way, see . That approach can be used here considering just the intercept

```
res5$summary.fix[1,1] + drop(A5pts3%*%res5$summary.random$i$mean)
```

```
[1]  9.683061 12.562824  6.729347
```

For the standard error, we need to take into account the error of the covariate values and regression coefficients.

```
summary(rvar <- res5$summary.random$i$sd^2)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5923  1.0360  1.4280  2.5150  2.6060 11.0700
```

```
sqrt(1^2+res5$summary.fix[1,2]^2 + drop(A5pts3%*%rvar))
```

```
[1] 1.529057 1.668779 1.850322
```

3.4.3 Response on a grid

The computation of all marginal posterior distributions on a grid 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. Only the mean and standard defiation are stored.

We use the projector matrix on the projector object that we use to project the posterior mean on the grid

```
stkgrid <- inla.stack(data=list(resp=NA), A=list(pgrid0$proj$A,1),
                      effects=list(i=1:spde5$n.spde,
                        m=rep(1,101*101)), tag='prd.gr')
stk.all <- inla.stack(stk5, stkgrid)
res5g <- inla(resp ~ 0 + m + f(i, model=spde5),
                data=inla.stack.data(stk.all),
                control.predictor=list(A=inla.stack.A(stk.all),
                  compute=TRUE), quantiles=NULL,
                control.results=list(return.marginals.random=FALSE,
                  return.marginals.predictor=FALSE))
res5g$cpu
```

```
Pre-processing    Running inla Post-processing           Total
     0.5055149       8.7927294       0.1059375       9.4041817
```

We get the indexes

```
igr <- inla.stack.index(stk.all, 'prd.gr')$data
```

and use it to visualize, together the prediction of the random field on previous section, on Figure 3.1 with the commands bellow

```
require(gridExtra)
grid.arrange(levelplot(prd0.m, col.regions=topo.colors(99), main='latent field mean',
                       xlab='', ylab='', scales=list(draw=FALSE)),
             levelplot(matrix(res5g$summary.fitt[igr,1], 101),
                       xlab='', ylab='', main='response mean',
                       col.regions=topo.colors(99), scales=list(draw=FALSE)),
             levelplot(prd0.s, col.regions=topo.colors(99), main='latent field SD',
                       xlab='', ylab='', scales=list(draw=FALSE)),
             levelplot(matrix(res5g$summary.fitt[igr,2], 101),
                       xlab='', ylab='', main='response SD',
                       col.regions=topo.colors(99), scales=list(draw=FALSE)),
             nrow=2)
```

We see on Figure 3.1 that we have a variation from -4 to 4 on the spatial effect. Considering also that we have standard deviations around 0.8 to 1.6, the spatial dependence is significantly.

Another thing is that the standard deviation of both, random field and the response, are less near the corner (0, 0) and greater near the corner (1,1). This is just proportional to the locations density.

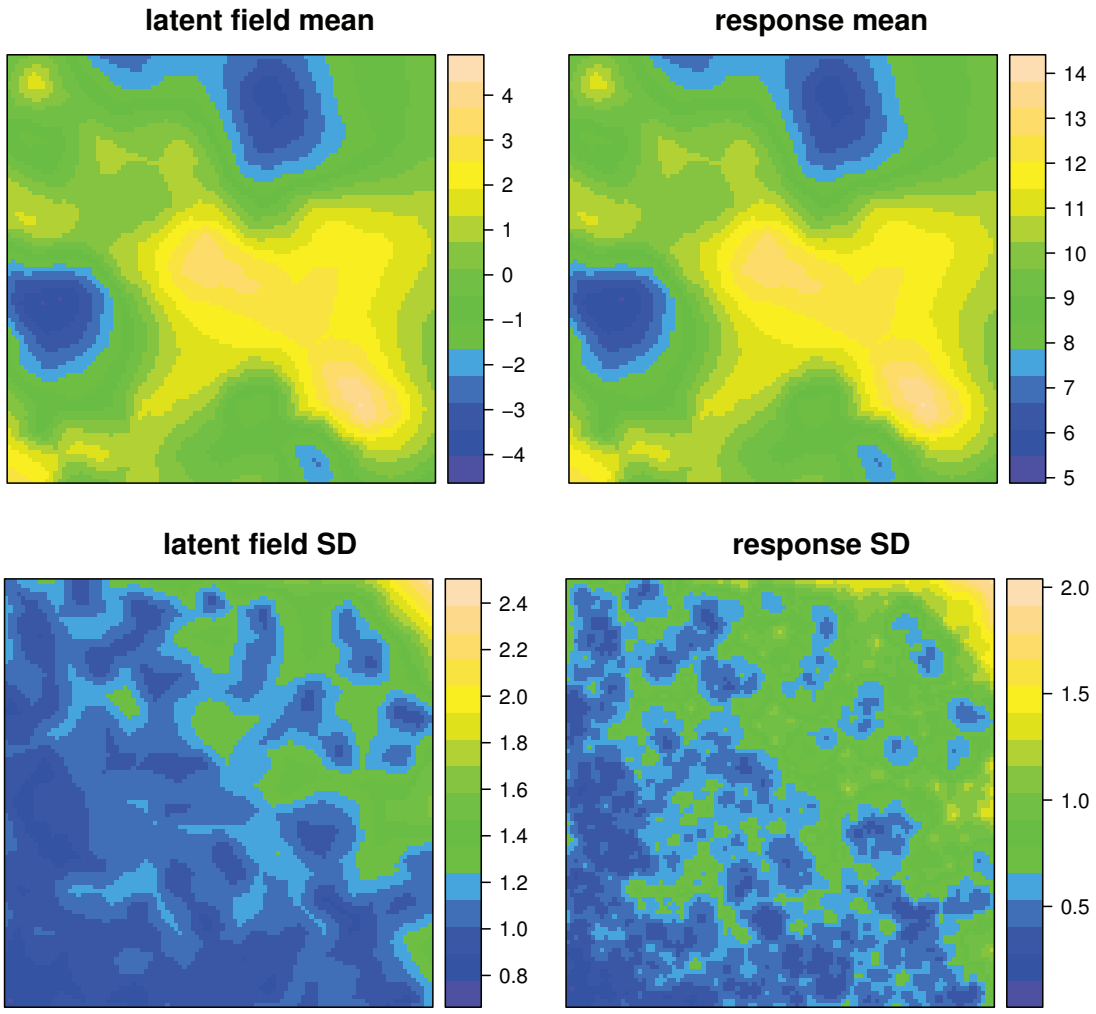

Figure 3.1: The mean and standard deviation of the random field (top left and bottom left) and the mean and standard variation of the response (top right and bottom right)

3.5 Results from different meshes

In this section we compare six results for the toy dataset based on the six different meshs builded on section 2.2. To do this comparison, we just plot the posterior marginal distributions of the model parameters. We evaluate the meshes by the addiction of the true values used on the simulation of the toy dataset. Also, we add the maximum likelihood estimates from **geoR** package, [Ribeiro Jr and Diggle, 2001].

We fit the model, using each one of the six meshes, and put the results in a list with the code bellow

```
lrf <- lres <- l.dat <- l.spde <- l.a <- list()
for (k in 1:6) {
    l.a[[k]] <- inla.spde.make.A(get(paste('mesh', k, sep='')), loc=coords)
    l.spde[[k]] <- inla.spde2.matern(get(paste('mesh', k, sep='')), alpha=2)
    l.dat[[k]] <- list(y=SPDEtoy[,3], i=1:ncol(l.a[[k]]),
                       m=rep(1, ncol(l.a[[k]])))
    lres[[k]] <- inla(y ~ 0 + m + f(i, model=l.spde[[k]]),
                      data=l.dat[[k]], control.predictor=list(A=l.a[[k]]))
    lrf[[k]] <- inla.spde2.result(lres[[k]], 'i', l.spde[[k]], do.transf=TRUE)
  }
```

The mesh size influences the computational time needed to fit the model. More nodes on the mesh need more computational time. The time running inla for these six meshes are

```
round(sapply(lres, function(x) x$cpu[2]), 2)
```

```
Running inla Running inla Running inla Running inla Running inla
       11.24         1.43         0.92        11.94         1.37
Running inla
        1.06
```

We compute the distribution for σ_e^2 for each fitted model

```
s2.marg <- lapply(lres, function(m)
                  inla.tmarginal(function(x) 1/x, m$marginals.hy[[1]]))
```

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
```

```
     beta       s2e       s2x     kappa
9.5400961 0.3739484 3.3227266 8.4134845
```

We want to visualize the posterior marginal distributions for β_0, σ_e^2, σ_x^2, κ, nominal range and the local variance τ. This can be done with the code bellow

```
rcols <- rainbow(6)##c(rgb(4:1/4,0:3/5,0), c(rgb(0,0:3/5,4:1/4)))
par(mfrow=c(2,3), mar=c(2.5,2.5,1,.5), mgp=c(1.5,.5,0), las=1)
xrange <- range(sapply(lres, function(x) range(x$marginals.fix[[1]][,1])))
yrange <- range(sapply(lres, function(x) range(x$marginals.fix[[1]][,2])))
plot(lres[[1]]$marginals.fix[[1]], type='l', xlim=xrange, ylim=yrange,
       xlab=expression(beta[0]), ylab='Density')
```

```
for (k in 1:6)
    lines(lres[[k]]$marginals.fix[[1]], col=rcols[k], lwd=2)
abline(v=beta0, lty=2, lwd=2, col=3)
abline(v=lk.est[1], lty=3, lwd=2, col=3)
xrange <- range(sapply(s2.marg, function(x) range(x[,1])))
yrange <- range(sapply(s2.marg, function(x) range(x[,2])))
plot.default(s2.marg[[1]], type='l', xlim=xrange, ylim=yrange,
               xlab=expression(sigma[e]^2), ylab='Density')
for (k in 1:6)
    lines(s2.marg[[k]], col=rcols[k], lwd=2)
abline(v=sigma2e, lty=2, lwd=2, col=3)
abline(v=lk.est[2], lty=3, lwd=2, col=3)
xrange <- range(sapply(lrf, function(r) range(r$marginals.variance.nominal[[1]][,1])))
yrange <- range(sapply(lrf, function(r) range(r$marginals.variance.nominal[[1]][,2])))
plot(lrf[[1]]$marginals.variance.nominal[[1]], type='l',
       xlim=xrange, ylim=yrange, xlab=expression(sigma[x]^2), ylab='Density')
for (k in 1:6)
    lines(lrf[[k]]$marginals.variance.nominal[[1]], col=rcols[k], lwd=2)
abline(v=sigma2x, lty=2, lwd=2, col=3)
abline(v=lk.est[3], lty=3, lwd=2, col=3)
xrange <- range(sapply(lrf, function(r) range(r$marginals.kappa[[1]][,1])))
yrange <- range(sapply(lrf, function(r) range(r$marginals.kappa[[1]][,2])))
plot(lrf[[1]]$marginals.kappa[[1]], type='l',
       xlim=xrange, ylim=yrange, xlab=expression(kappa), ylab='Density')
for (k in 1:6)
    lines(lrf[[k]]$marginals.kappa[[1]], col=rcols[k], lwd=2)
abline(v=kappa, lty=2, lwd=2, col=3)
abline(v=lk.est[4], lty=3, lwd=2, col=3)
xrange <- range(sapply(lrf, function(r) range(r$marginals.range.nominal[[1]][,1])))
yrange <- range(sapply(lrf, function(r) range(r$marginals.range.nominal[[1]][,2])))
plot(lrf[[1]]$marginals.range.nominal[[1]], type='l',
       xlim=xrange, ylim=yrange, xlab='nominal range', ylab='Density')
for (k in 1:6)
    lines(lrf[[k]]$marginals.range.nominal[[1]], col=rcols[k], lwd=2)
abline(v=sqrt(8)/kappa, lty=2, lwd=2, col=3)
abline(v=sqrt(8)/lk.est[4], lty=3, lwd=2, col=3)
xrange <- range(sapply(lrf, function(r) range(r$marginals.tau[[1]][,1])))
yrange <- range(sapply(lrf, function(r) range(r$marginals.tau[[1]][,2])))
plot(lrf[[1]]$marginals.tau[[1]], type='l',
       xlim=xrange, ylim=yrange, xlab=expression(tau), ylab='Density')
for (k in 1:6)
    lines(lrf[[k]]$marginals.tau[[1]], col=rcols[k], lwd=2)
legend('topright', c(paste('mesh', 1:6, sep=''), 'True', 'Likelihood'),
         lty=c(rep(1,6), 2, 3), lwd=rep(2, 6), col=c(rcols,3,3), bty='n')
```

At the Figure 3.2 we can see that the posterior marginal distribution for the intercept has mode on the likelihood estimate, considering the results from all six meshes.

```
1/kappa
```

```
[1] 0.1428571
```

The main differences are on the noise variance σ_e^2 (the nugget effect). The result from the mesh based on the points and with small triangles mode less than the likelihood estimate, the second has mode near likelihood estimate and the third large. Considering

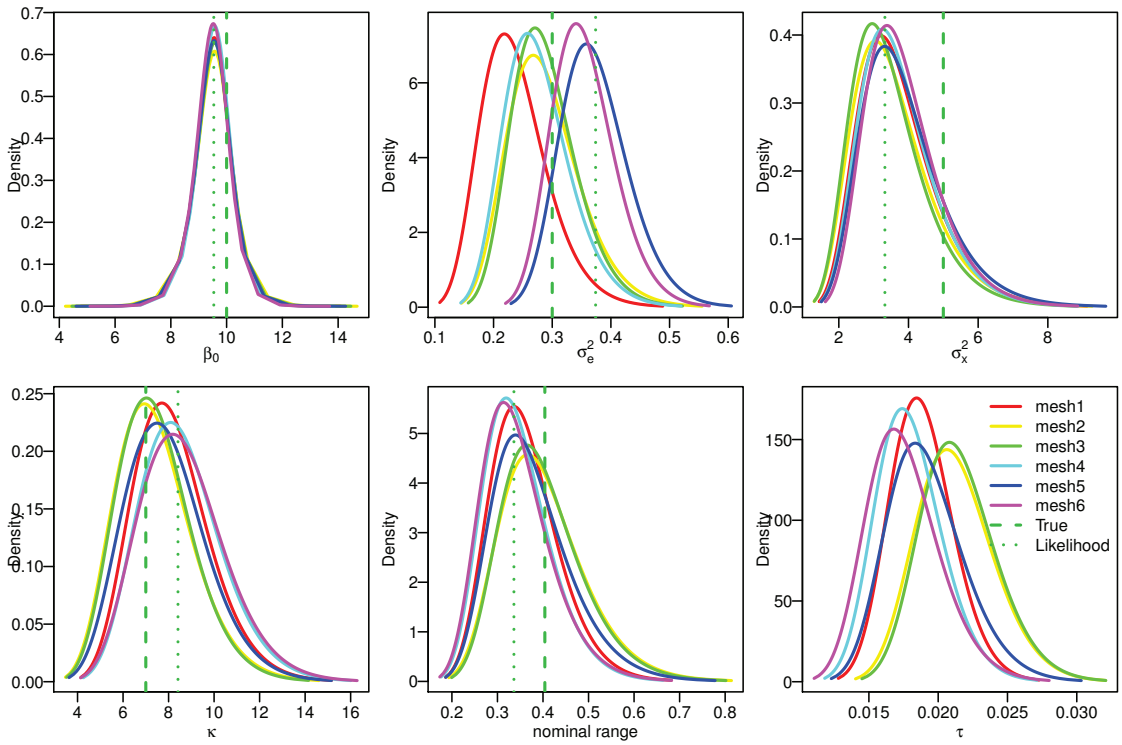

Figure 3.2: Marginal posterior distribution for β_0 (top left), σ_e^2 (top mid), σ_x^2 (top right), κ (bottom left), nominal range (bottom mid) and τ (bottom right).

the other meshes, the mesh four has mode around likelihood estimate and the other two litle larger, similar to the third mesh, such is based on points but with some freedom (`cutoff` greather than zero).

For the marginal variance of the latent field, σ_x^2, the results with all meshes had mode near the likelihood estimate. For the scale parameter κ all meshes has mode less than the likelihood estimate. The posterior distribution from the meshes based on points are that ones with less mode and that the mode from third mesh are the less. For the practical range the opposite happens.

These results are not conclusive, but a general comment is that is good to have a mesh with some tune on the points locations, to access noise variance, but with some flexibility to avoid many variability on the triangles size and shape, to get good latent field parameters estimation.

Chapter 4

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.

4.1 The data set

It have data on more one thousand of locations within Paraná state, a state at south of Brazil. We have daily rainfall data on each day of the year 2011 on 616 locations, including stations within the Paraná state and around its border.

We have these dataset on the **INLA** package and call it with

```
data(PRprec)
```

We have the coordinates at first two columns, altitude at third column and more 365 columns, one for each day with the 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.

```
PRprec[ii <- c(which(is.na(PRprec$A))[which.min(PRprec$La[is.na(PRprec$A)])],
               which(PRprec$Lo%in%range(PRprec$Lo)), which.max(PRprec$A)), 1:10]
```

```
     Longitude Latitude Altitude d0101 d0102 d0103 d0104 d0105 d0106 d0107
1239  -48.9394 -26.1800       NA  20.5   7.9   8.0   0.8   0.1  15.6  31.0
658   -48.2167 -25.0831        9  18.1   8.1   2.3  11.3  23.6   0.0  22.6
1325  -54.4842 -25.6017      231  43.8   0.2   4.6   0.4   0.0   0.0   0.0
885   -51.5167 -25.7331     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 4.1.

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 4.1. 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,

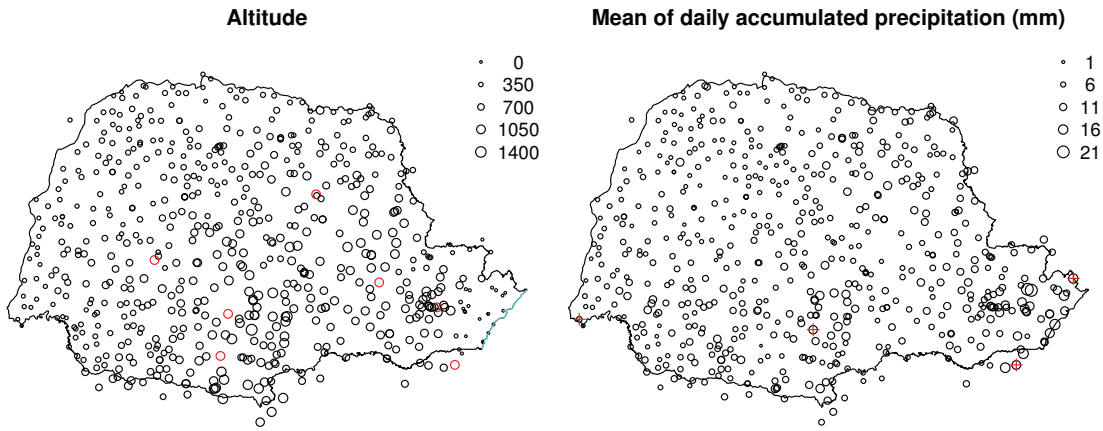

Figure 4.1: Locations of Paraná stations, altitude and average of daily accumulated precipitation (mm) on January.

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

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

Also we have the Paraná state border

```
data(PRborder)
```

We visualize the locations on Figure 4.1, with the commands bellow

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

4.2 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

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

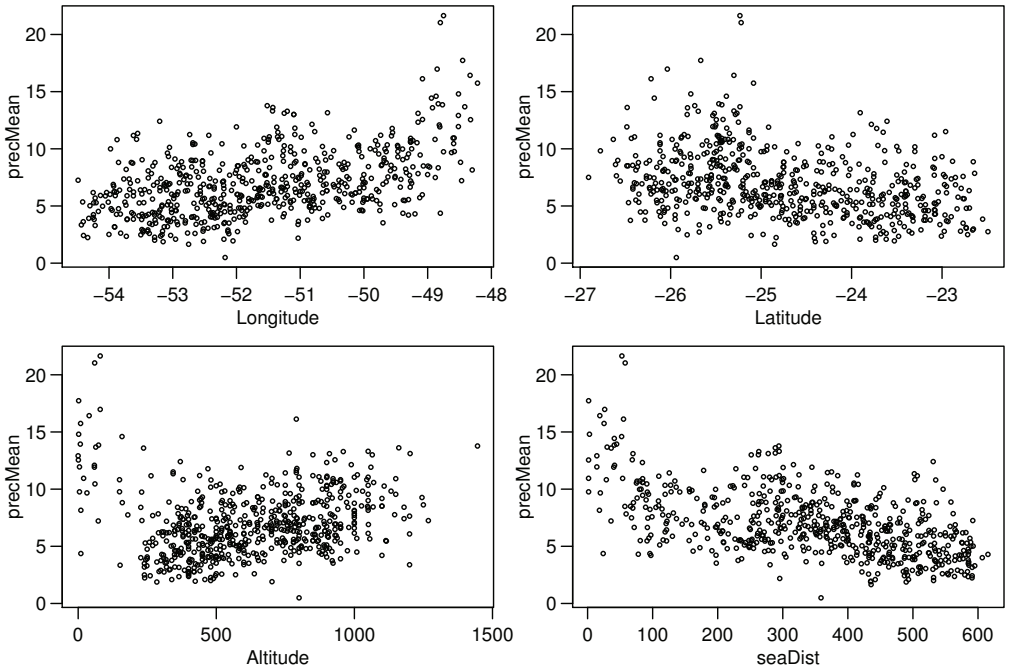

Figure 4.2: Dispersion plots of average of daily accumulated precipitation mean by Longitude (top left), Latitude (top right), Altitude (bottom left) and distance to sea (bottom right).

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 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 4.1) and computes the distance from each station to the neighbor coordinate of this line.

We compute the distances in Km units using the `spDists()` function from **sp** package

```
coords <- as.matrix(PRprec[,1:2])
mat.dists <- spDists(coords, PRborder[1034:1078,], longlat=TRUE)
```

This function computes the distance between each location on the first points set to each one on the second points set. So, we need to take the minimum

```
PRprec$"seaDist" <- apply(mat.dists, 1, min)
```

We see the dispersion plots at Figure 4.2.

```
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(PRprec)))  plot(PRprec[c(i,ncol(PRprec)-1)], cex=.5)
```

With the Figure 4.2, 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 use the mesh builded for the Paraná state on the Chapter 2.2.

The projector matrix can be done by

```
A <- inla.spde.make.A(prmesh2, loc=coords)
```

The SPDE model is defined by

```
spde <- inla.spde2.matern(prmesh2, alpha=2)
```

and the stack data is defined to include four effects: the GRF, intercept, west coordinate and distance to sea

```
stk.dat <- inla.stack(data=list(y=PRprec$precMean),
                      A=list(A,1), tag='dat',
                      effects=list(list(i=1:spde$n.spde),
                        data.frame(Intercept=1,
                                   gWest=inla.group(coords[,1]),
                                   gSeaDist=inla.group(PRprec$seaDist))))
```

We fit the two models using the same stack data. We just use different formula. For the model with west coordinate we have

```
f.west <- y ~ 0 + Intercept + f(gWest, model='rw1') + f(i, model=spde)
r.west <- inla(f.west, family='Gamma', control.compute=list(dic=TRUE),
                 data=inla.stack.data(stk.dat),
                 control.predictor=list(A=inla.stack.A(stk.dat), compute=TRUE))
```

For the model with distance to sea covariate we have

```
f.sead <- y ~ 0 + Intercept + f(gSeaDist, model='rw1') + f(i, model=spde)
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.west$dic$dic, r.sead$dic$dic)
```

```
[1] 2567.958 2565.172
```

and choose the model that 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     mode
Intercept 1.944486 0.05138276   1.841195 1.944319   2.048473 1.943871
                   kld
Intercept 1.963384e-10
```

To dispersion parameter of the gamma likelihood we have

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

```
      mean         sd 0.025quant   0.5quant 0.975quant       mode
12.6022702  0.8865177 10.9317355 12.5786034 14.4156926 12.5386946
```

To dispersion parameter of the coefficients of the sea distance covariate we have

```
r.sead$summary.hy[2,]
```

```
     mean         sd 0.025quant   0.5quant 0.975quant       mode
 8119.105   5144.439   2310.839   6823.275  21483.081   4919.289
```

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

```
r.sead$summary.hy[4,]
```

```
     mean         sd 0.025quant   0.5quant 0.975quant       mode
1.3795452  0.3085759  0.7624940  1.3844033  1.9743411  1.3989754
```

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)
```

The posterior mean for the marginal variance

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

```
[1] 0.06108988
```

And to the practical range of the process we have

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

```
[1] 0.7460229
```

At the Figure 4.3 we look the posterior distribution to β_0, ϕ, $1/\kappa$, σ_x^2, practical range and to the mean and 95% credibility interval of the distance to sea effect at Figure 4.3. We choose $1/\kappa$ instead *kappa* because $1/\kappa$ is the range parameter and in this case is expressed in degrees units.

```
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) 1/exp(x), r.sead$marginals.hy[[4]]),
               type='l', xlab=expression(kappa), ylab='Density')
plot.default(r.f$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')
```

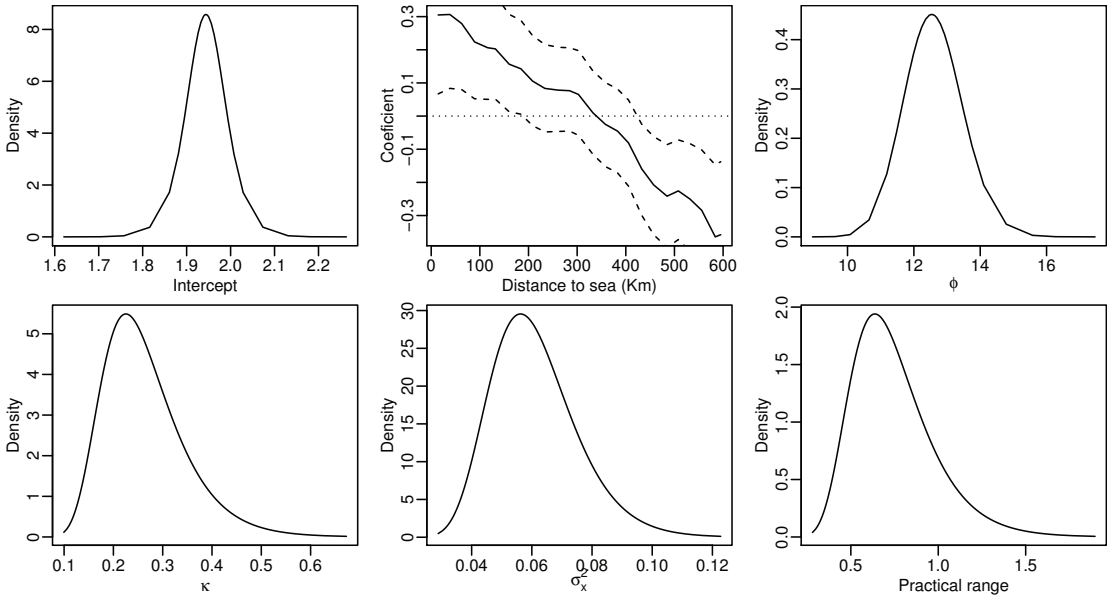

Figure 4.3: Posterior marginal distribution of β_0 (top left), the posterior mean (continuous line) and 95% credibility interval (dashed lines) to effect of distance to sea (top mid), posterior marginal distribution to ϕ (top right), posterior marginal distribution of κ (bottom left), posterior marginal distribution to nominal variance of the random field (bottom mid) and posterior marginal distribution of the practical range (bottom right).

4.3 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] 2720.364 2565.172
```

and conclude that the spatial effect is significantly to explain the precipitation.

4.4 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

```
(stepsize <- 4*1/111)
```

```
[1] 0.03603604
```

```
(nxy <- round(c(diff(range(PRborder[,1])), diff(range(PRborder[,2])))/stepsize))
```

```
[1] 183 117
```

Here we divide the range of each coordinate by 4 kilometers on degree scale. We use (4/111) factor because each degree has aproximately 111 kilometers. Its because we have the coordinates with units in kilometers and we want each cell on the grid with 4×4 km.

Now, we get the projector with

```
projgrid <- inla.mesh.projector(prmesh2, xlim=range(PRborder[,1]),
                                ylim=range(PRborder[,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

```
require(splancs)
table(xy.in <- inout(projgrid$lattice$loc,
                     cbind(PRborder[,1], PRborder[,2])))
```

```
FALSE  TRUE
 7865 13546
```

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

We visualize the this on Figure 4.4. We see on top left graph at Figure 4.4 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.

4.5 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 <- apply(spDists(PRborder[1034:1078,], prdcoo, longlat=TRUE), 2, min)
```

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

```
ug.seadist <- sort(unique(inla.group(PRprec$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(i=1:spde$n.spde,
                            data.frame(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 4.4 the posterior mean and standard deviation to response with commands bellow

```
library(gridExtra)
do.call('grid.arrange',
          lapply(list(xmean, xsd, m.prd, sd.prd),
                 levelplot, col.regions=terrain.colors(16),
                 xlab='', ylab='', scales=list(draw=FALSE)))
```

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 west side of the Paraná state.

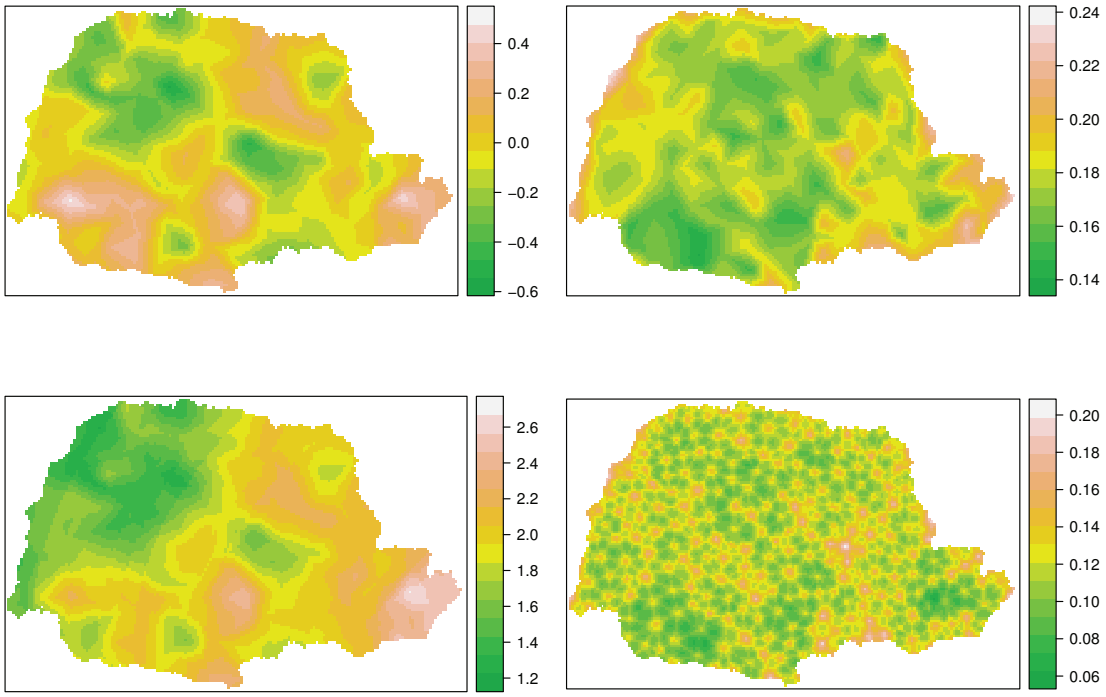

Figure 4.4: Posterior mean and standard deviation of the random field (top left and right respectively). Posterior mean and standard deviation of the response (top left and right respectively).

Chapter 5

Semicontinuous model to daily rainfall

In some applications we have data with is hard to fit a single distribution to. For example, when whe have more zeros than the expected under a Poisson distribution. In that case there is a waste literature considering zero-inflated models. These models are implemented in many softwares as well.

With hierarchical models, sometimes a random effect is sufficient to explain the zeroinflation. This is true in same sense where a random effect is sufficient to explain the extra variation, considering the overdispersion models. These cases happens when a specified random effect is sufficient to explain the zeroinflation (or the overdispersion).

A different problem happens with daily rainfall. In this case is not easy to fit a simple model. There are days without rain and when we have precipitation on a day, we have a continuous amount of water. A particular case in this problem is that the daily cumulated precipitation is commonly very asymmetric.

5.1 The model and data

In this section we build a jointly model for occurrence and rainfall amount. We call it a semicontinuous model because we have zeros and positive continuous values.

Let r_i the rainfall amount at location i. We define two new variables. One is the occurence variable

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

and other the rainfall amoung variable

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

Now we can define a model for each one variable. We use a model with two likelihoods. We use the Bernoulli likelihood for z_i and the gamma for y_i.

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

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

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

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

We use a rainfall data collected by the Brasilian National Wather Agency. We have a dataset collected at 616 gauge stations for each day of 2012. These stations covers the Paraná State in South of Brasil.

These data are available on **INLA** package and can be called by

```
data(PRprec)
```

and we have, at first three columns, the coordinates and altitude. We can see this informations and the rainfall for the first seven days at three stations with

```
PRprec[1:3, 1:10]
```

```
  Longitude Latitude Altitude d0101 d0102 d0103 d0104 d0105 d0106 d0107
1  -50.8744 -22.8511      365     0     0     0   0.0     0     0   2.5
3  -50.7711 -22.9597      344     0     1     0   0.0     0     0   6.0
4  -50.6497 -22.9500      904     0     0     0   3.3     0     0   5.1
```

Also considering the first seven days, a summary over stations and the number of stations with rain

```
sapply(PRprec[,4:10], summary)
```

```
          d0101 d0102  d0103  d0104  d0105  d0106  d0107
Min.     0.0000  0.00  0.000  0.000  0.000  0.000  0.000
1st Qu.  0.0000  0.00  0.000  0.000  0.000  0.000  0.000
Median   0.0000  0.00  0.300  0.000  0.000  0.000  0.000
Mean     0.8074  2.87  5.928  3.567  2.793  4.059  5.999
3rd Qu.  0.0000  2.10  7.000  3.475  0.600  3.850  6.700
Max.    43.8000 55.60 78.500 62.900 64.100 81.400 75.700
NA's     6.0000  7.00  7.000  6.000  6.000  8.000  8.000
```

```
colSums(PRprec[,4:10]>0, na.rm=TRUE)
```

```
d0101 d0102 d0103 d0104 d0105 d0106 d0107
   72   210   315   237   181   240   239
```

and we see, for example, that we have that the third quartil for the first day is zero when we have rain only at 72 stations. We also have missing values.

On this section we consider the data at third day. We define the two response variables with

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

The map of this data are showed on the left graph of the Figure 5.1. Also, the histogram of the positive precipitation are on the rigth graph of this Figure. Both produced by the code below

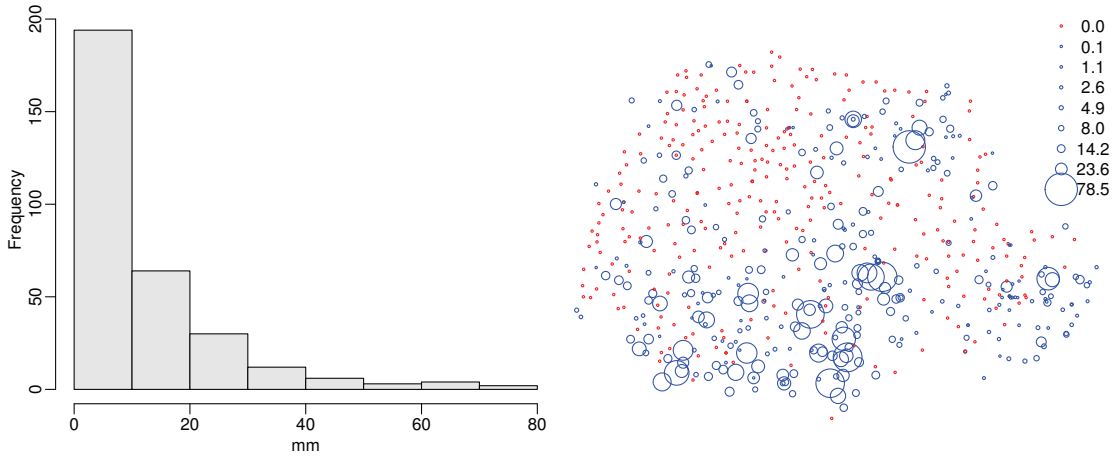

Figure 5.1: Histogram of rainfall amount and map of precipitation.

■vprmesh,figure=TRUE,eps=FALSE,width=5.5,height=5■= ■plotprmesh■ @

Figure 5.2: Mesh for the Paraná state.

```
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(PRprec[,1:2], cex=0.3 + PRprec[,jday]/20, asp=1,
       col=c('red', 'blue')[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('red',rep('blue',length(q.y)-1)))
```

5.2 Fitting the model and some results

First we need to define the mesh and prepare the data to fit the model.

```
[1] 871
```

This mesh can be visualized at Figure 5.2 with following commands

```
plot(mesh, asp=1)
```

The SPDE model definition is made by

```
spde <- inla.spde2.matern(mesh, alpha=2)
```

We have to prepare the data in addequate form using the stack functionality. First we get the predictor matrix for the SPDE model effect with

```
A <- inla.spde.make.A(mesh, loc=as.matrix(PRprec[,1:2]))
```

For comparison pourpose, we can fit a model only with the rainfall occurrence and another only with rainfall amount. Because we have to fit the joint model, we prepare the stack for each separated model considering each response and also the response for the jointly model. For the jointly model, we need to have a two columns response.

The stack for the occurrence is

```
stk.z <- inla.stack(tag='est.z',
                    data=list(z=z, ### occurrence for separate model
```

```
                        y=cbind(z, NA)), ### z at first column of y
                    A=list(A, 1),
                    effects=list(
                      list(i.z=1:spde$n.spde),
                      list(z.b0=rep(1,length(z)))))
```

The data stack for the precipitation amount is

```
stk.y <- inla.stack(tag='est.y',
                    data=list(r=y, ### rainfall for separate model
                        y=cbind(NA, y)), ### rainfall at second column
                    A=list(A, 1),
                    effects=list(
                      list(i.y=1:spde$n.spde),
                      list(y.b0=rep(1,length(y)))))
```

Fitting the model for each response separately

```
res.z <- inla(z ~ 0 + z.b0 + f(i.z, model=spde), family='binomial',
                data=inla.stack.data(stk.z), control.compute=list(dic=TRUE),
                control.predictor=list(A=inla.stack.A(stk.z), compute=TRUE))
res.y <- inla(r ~ 0 +  y.b0 + f(i.y, model=spde), family='gamma',
                data=inla.stack.data(stk.y), control.compute=list(dic=TRUE),
                control.predictor=list(A=inla.stack.A(stk.y), compute=TRUE))
```

Defining a full data stack that join both responses for jointly model

```
stk.zy <- inla.stack(stk.z, stk.y)
```

To fit the model, we make inference to β_x using the copy of the strategy with `fixed=FALSE`.

```
res.zy <- inla(y ~ 0 + z.b0 + y.b0 +
                 f(i.z, model=spde) + f(i.y, copy='i.z', fixed=FALSE),
                 family=c('binomial', 'gamma'),
                 data=inla.stack.data(stk.zy), control.compute=list(dic=TRUE),
                 control.predictor=list(A=inla.stack.A(stk.zy), compute=TRUE))
```

Also to compare, we fit the model by defining the SPDE model on the rainfall amount equation and copying it on the occurrence equation

```
res.yz <- inla(y ~ 0 + z.b0 + y.b0 +
                 f(i.y, model=spde) + f(i.z, copy='i.y', fixed=FALSE),
                 family=c('binomial', 'gamma'),
                 data=inla.stack.data(stk.zy), control.compute=list(dic=TRUE),
                 control.predictor=list(A=inla.stack.A(stk.zy), compute=TRUE))
```

We can see if the spatial effect are correlated by looking at posterior of the β_x parameter. We see that on both jointly models it is different from zero. So, the rainfall ocurrence and rainfall amount share the same spatial pattern.

```
round(rbind(beta.zy=res.zy$summary.hy[4, ], beta.yz=res.yz$summary.hy[4, ]), 4)
```

```
          mean     sd 0.025quant 0.5quant 0.975quant   mode
beta.zy 0.3491 0.0792     0.1948   0.3487     0.5057 0.3473
beta.yz 1.7692 0.2175     1.3512   1.7653     2.2057 1.7535
```

These values for β_x are different due that they are fitted taking into account two factores. One is relatinship existence between the spatial effect on both linear predictor. We infer this looking if β_x is less or greater than zero. The second factor is related to the variance of the effect on the first equation. The value considering the two different orders are equal only if the variance of the effect on first equation is one. 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 can compare the results obtained of the two separated models with the two joint models looking at its DIC values. But, we need to take care on DIC value from the joint models. It computed by summing the local DIC for each observation of both responses. So, we have to compute the correct DIC value for each response by sum of local DIC values correspoinding to each one.

```
iz <- inla.stack.index(stk.zy, tag='est.z')$data
dic.zy <- c(dic.z = sum(res.zy$dic$local.dic[iz], na.rm=TRUE),
            dic.y = sum(res.zy$dic$local.dic[-iz], na.rm=TRUE))
dic.yz <- c(dic.z = sum(res.yz$dic$local.dic[iz], na.rm=TRUE),
            dic.y = sum(res.yz$dic$local.dic[-iz], na.rm=TRUE))
```

We see that, considering occurrence, the DIC from jointly model is similar than one from separate model, when the SPDE model is declared on the occurrence equation.

```
rbind(sep=c(res.z$dic$dic, res.y$dic$dic), joint.zy=dic.zy, joint.yz=dic.yz)
```

```
            dic.z    dic.y
sep      703.5882 2080.197
joint.zy 704.7564 2130.295
joint.yz 714.4875 2120.928
```

Now, we look at the summary of the posterior distribution for α_z considering the three models fitted that includes this parameter

```
round(rbind(sep=res.z$summary.fix, joint.zy=res.zy$summary.fix[1,],
            joint.yz=res.yz$summary.fix[1,]), 4)
```

```
           mean     sd 0.025quant 0.5quant 0.975quant   mode kld
z.b0     0.2305 0.8523    -1.5089   0.2012     2.1185 0.1633   0
joint.zy 0.0682 0.4968    -0.9339   0.0627     1.1023 0.0545   0
joint.yz 0.0256 0.3349    -0.6526   0.0266     0.6994 0.0289   0
```

and we see that the posterior mean of these three results are different from separate model to jointly ones. But, α_z still not different from zero, because with separate model, when the posterior mean value is more far from zero, the posterior standard deviation is greater. Due to it, when we compute the cumulative distribution at zero, under the posterior distribution from the three results, the values are similar.

```
sapply(list(sep=res.z$marginals.fix[[1]], joint.zy=res.zy$marginals.fix[[1]],
            joint.yz=res.yz$marginals.fix[[1]]), function(m) inla.pmarginal(0, m))
```

```
      sep  joint.zy  joint.yz
0.3769128 0.4434749 0.4674468
```

For α_y

```
round(rbind(sep=res.y$summary.fix, joint.zy=res.zy$summary.fix[2,],
            joint.yz=res.yz$summary.fix[2,]), 4)
```

```
           mean     sd 0.025quant 0.5quant 0.975quant   mode kld
y.b0     2.1348 0.1297     1.8781   2.1347     2.3918 2.1345   0
joint.zy 2.0849 0.2054     1.6742   2.0869     2.4907 2.0968   0
joint.yz 1.9044 0.2227     1.4652   1.9035     2.3518 1.9041   0
```

we have quite similar posterior mean from the three fitted models.

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

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

```
[1] 0.5170337 8.0435688
```

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.yz$summary.hy[1, ]
```

```
     mean         sd 0.025quant   0.5quant 0.975quant       mode
0.94697470 0.08803519 0.78281221 0.94413840 1.12831764 0.93964323
```

We look the posterior marginal distributions of α_z, α_y, ϕ and β on Figure 5.3 with commands below

```
par(mfrow=c(2,2), mar=c(3,3,.5,.5), mgp=c(1.5,.5,0), las=1)
plot(res.yz$marginals.fix[[1]], type='l', ylab='Density',
       xlab=expression(alpha[z]))
plot(res.yz$marginals.fix[[2]], type='l', ylab='Density',
       xlab=expression(alpha[y]))
plot(res.yz$marginals.hy[[1]], type='l', ylab='Density',
       xlab=expression(phi))
plot(res.yz$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 5.4 the mean and this quantiles for each x in order of the posterior mean, with code below

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

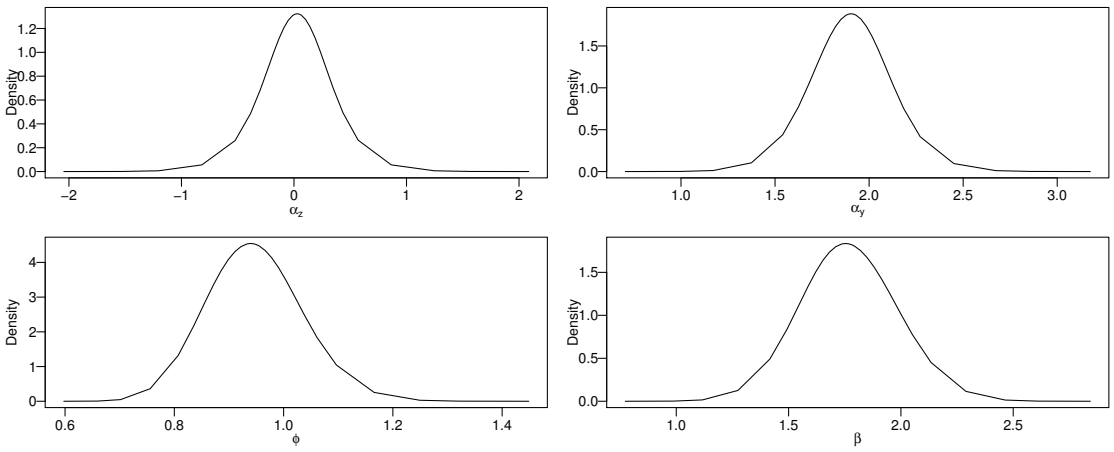

Figure 5.3: Posterior marginal distributions of α_z (top left), α_y (top right), ϕ (bottom left) and β (bottom right).

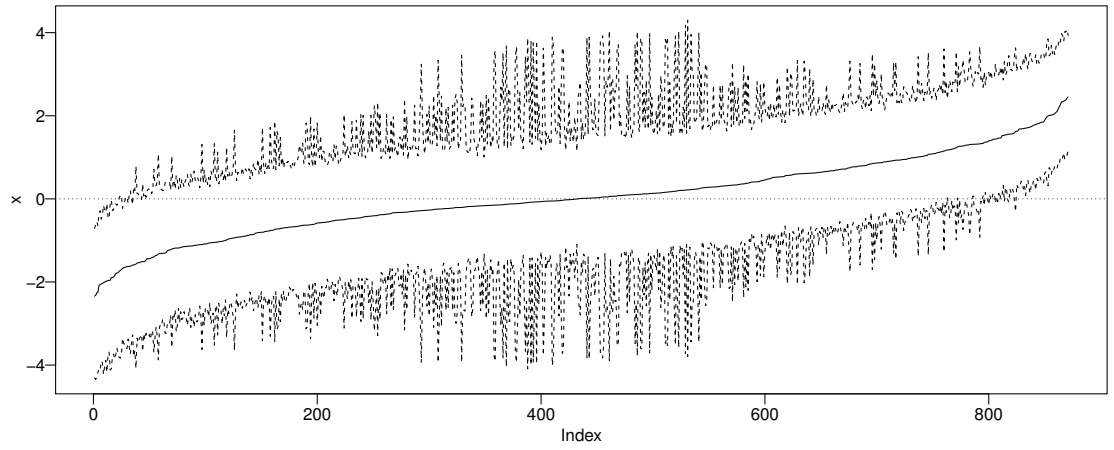

Figure 5.4: Posterior man of each x_i (continuous line) and its 2.5% and 97.5% quantiles of the posterior marginal distributions (dashed lines).

```
res.yz0 <- inla(y ~ 0 + z.b0 + y.b0,
              family=c('binomial', 'gamma'),
              data=inla.stack.data(stk.zy), control.compute=list(dic=TRUE),
              control.predictor=list(A=inla.stack.A(stk.zy), compute=TRUE))
```

and we have

```
rbind(dic.0=c(dic.yz0=sum(res.yz0$dic$local.dic[iz], na.rm=TRUE),
            dic.yz0=sum(res.yz0$dic$local.dic[-iz], na.rm=TRUE)), dic.s=dic.yz)
```

```
       dic.yz0  dic.yz0
dic.0 845.5270 2157.415
dic.s 714.4875 2120.928
```

and conclude that the spatial random effect is significative for both occurence and amoung of rainfall.

5.3 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].

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.yz.f <- inla.spde2.result(res.yz, 'i.y', spde, do.transf=TRUE)
```

The posterior mean for the marginal variance

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

```
[1] 0.5181122
```

And to the practical range of the process we have

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

```
[1] 1.081148
```

At the Figure 5.5 we look the posterior distribution to κ, σ_x^2 and of the practical range. The κ and practical range are on degrees units.

```
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.yz$marginals.hy[[3]]),
              type='l', xlab=expression(kappa), ylab='Density')
plot.default(res.yz.f$marginals.variance.nominal[[1]], type='l',
              xlab=expression(sigma[x]^2), ylab='Density')
plot.default(res.yz.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

```
data(PRborder)
(nxy <- round(c(diff(range(PRborder[,1])), diff(range(PRborder[,2])))/.02))
```

```
[1] 330 210
```

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

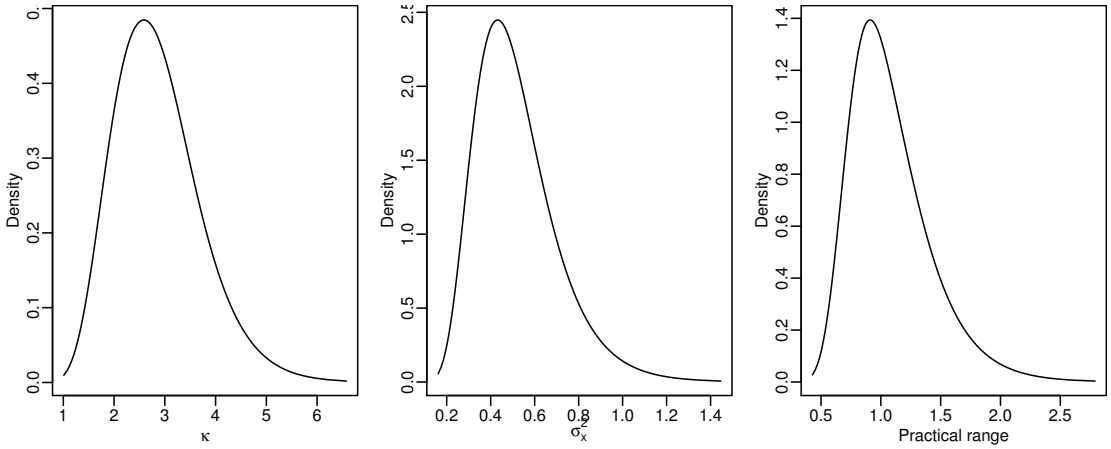

Figure 5.5: Posterior marginal distribution of κ (left), σ_x^2 (mid) and of the practical range (right).

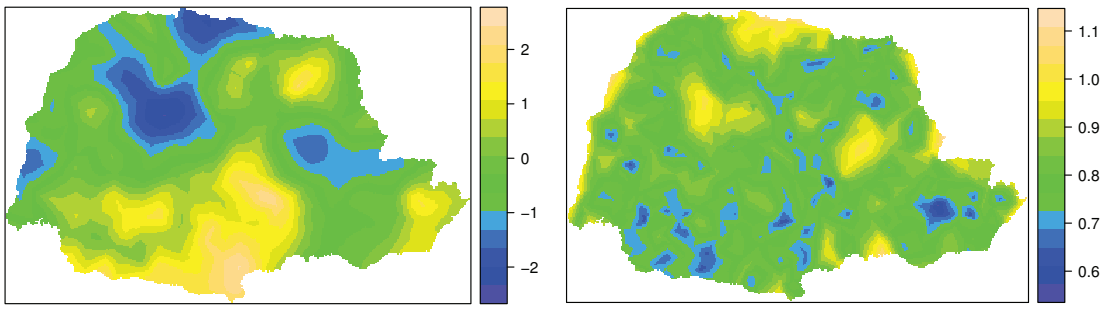

Figure 5.6: Posterior mean (left) and standard deviation (right) of the random field.

and get the posterior mean and posterior standard deviation with

```
xmean <- inla.mesh.project(projgrid, res.yz$summary.random$i.z$mean)
xsd <- inla.mesh.project(projgrid, res.yz$summary.random$i.z$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, PRborder))
```

```
FALSE  TRUE
25188 44112
```

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

We visualize the this on Figure 5.6. 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 5.6 with the right graph on the Figure 5.1 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 6

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.

6.1 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.

6.1.1 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))
```

6.2 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.

```
Ac <- inla.spde.make.A(mesh, loc=loc.c)
Ay <- 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. The data stack can be obtained by

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

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=-3)))
```

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)))
```

6.3 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 -4.5115  1.3811 -7.4151 -4.5175 -1.5682 -4.5254  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   mode
Prec.c    5 5.7439 0.7719     4.3671   5.6965     7.3970 5.6063
Prec.y    2 1.4997 0.3844     0.8893   1.4508     2.3898 1.3575
```

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    -2.7658     0.1585    -3.0754    -2.7662    -2.4533
      mode
   -2.7676
```

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)
```

```
    True   mean   low   high
s2m    5 4.4352 1.405 8.4117
s2x    3 4.9245 1.205 9.9352
```

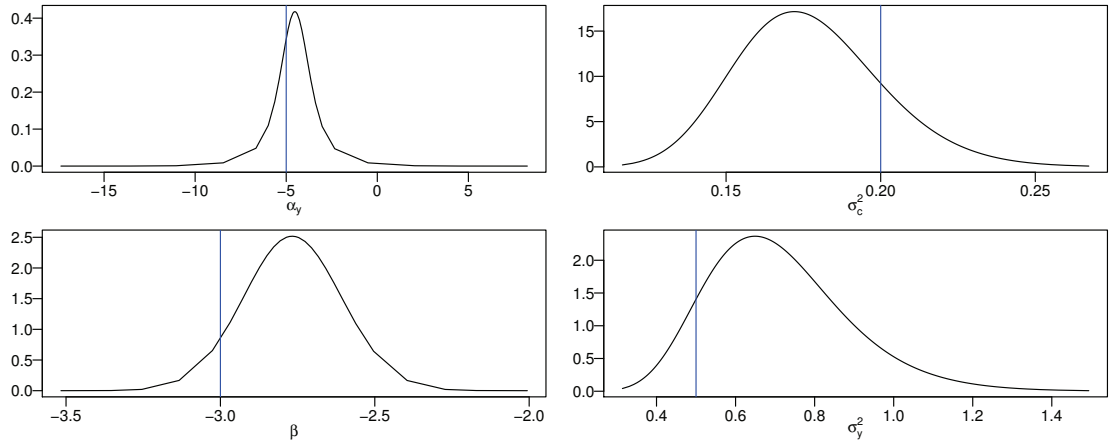

Figure 6.1: Posterior distribution of the likelihood parameters.

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 3.7800 2.0607  5.6360
kappa.x    7 5.9067 2.2333 10.3653
```

We see the posterior distribution of likelihood parameters on Figure 6.1 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 6.1 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 6.2 generated with comands below

```
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)
```

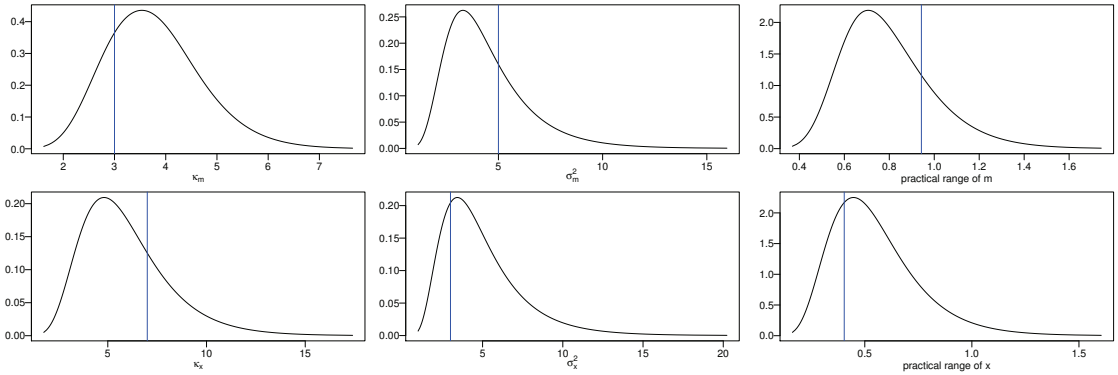

Figure 6.2: Posterior distribution of all the parameters of both random field.

```
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 6.2 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, Ay)
```

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

```
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,
```

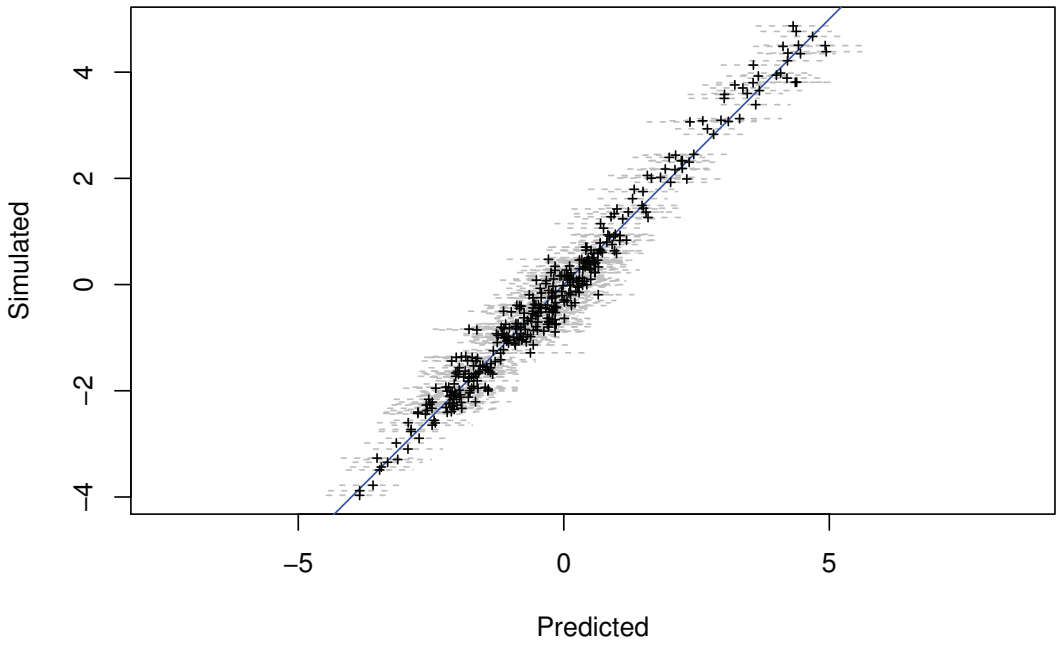

Figure 6.3: Simulated versus predicted values of m (+) and the approximated credibility intervals.

```
            lty=2, col=gray(.75))
abline(c(0,1), col=4); points(m.mprd, mm, pch=3, cex=.5)
```

on the Figure 6.3. The blue line represents the situation where predicted is equal to simulated. We see that the prediction are wery well.

Chapter 7

Non stationary model

7.1 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 (2.7) and (2.8). The non stationarity is made by a redefinition of the precision matrix.

This new definition is

$$\boldsymbol{Q} = \boldsymbol{D}^{(0)}(\boldsymbol{D}^{(1)}\boldsymbol{M}^{(0)}\boldsymbol{D}^{(1)} + \boldsymbol{D}^{(2)}\boldsymbol{D}^{(1)}\boldsymbol{M}^{(1)} + (\boldsymbol{M}^{(1)})^T\boldsymbol{D}^{(1)}\boldsymbol{D}^{(2)} + \boldsymbol{M}^{(2)})\boldsymbol{D}^{(0)} \qquad (7.1)$$

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.

7.2 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 (7.1) 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.2d(, pl01, cutoff=0.03,
                        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 7.1 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)
```

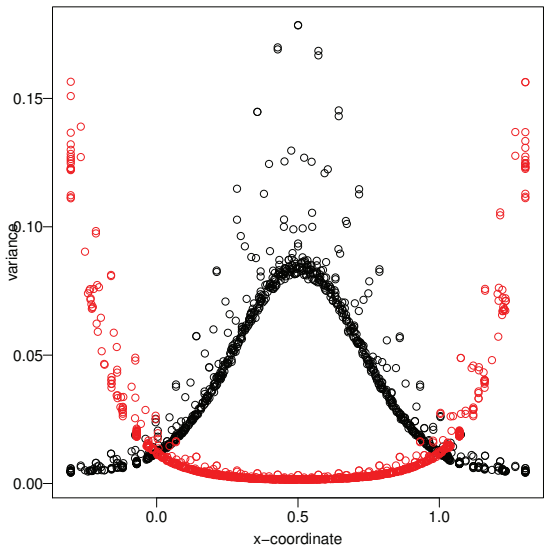

Figure 7.1: Variances implied by both non stationary process defined.

7.3 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 7.2. 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)
```

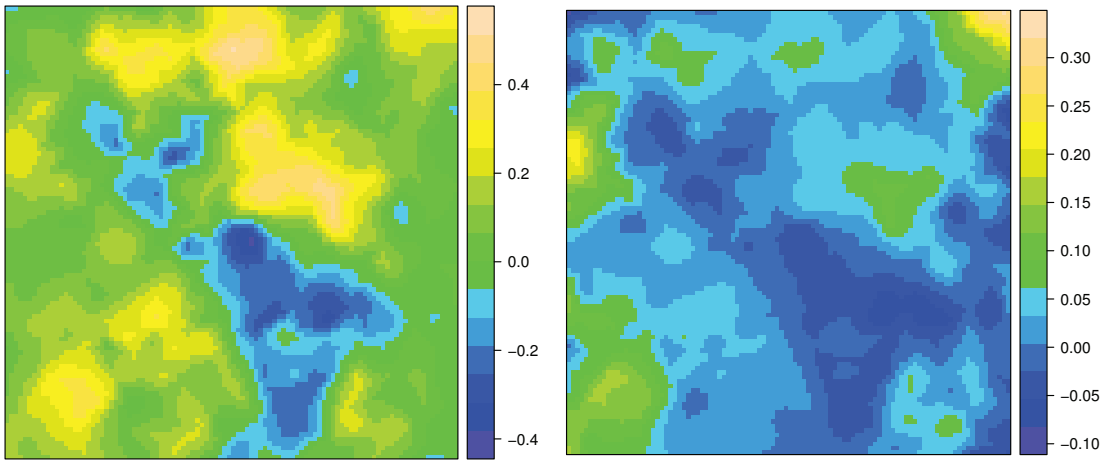

Figure 7.2: Two simulated random fields, using two diferent θ on same basis functions.

7.3.1 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 and is available directly in outputs of `inla.spde2.matern()` function. So, we do the simulation with

```
s1r <-  as.vector(inla.qsample(1, Q1, seed=1, constr=spde$f$extraconstr))
s2r <-  as.vector(inla.qsample(1, Q2, seed=1, constr=spde$f$extraconstr))
```

and we have

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

```
       Min.  1st Qu.  Median    Mean 3rd Qu.   Max.
s1r -0.4399 -0.03012 0.05297 0.06406 0.15710 0.7765
s2r -0.5973 -0.01840 0.02174 0.02401 0.06044 0.4819
```

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

```
cor1 cor2
   1    1
```

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

7.4 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    mode
Theta1 for i   -1 -0.9554 0.0295    -1.0193  -0.9526    -0.9046 -0.9442
Theta2 for i    2  1.9045 0.1233     1.6893   1.8940     2.1703  1.8628
Theta3 for i   -1 -0.9676 0.0505    -1.0517  -0.9732    -0.8563 -0.9918
```

and for the second

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

```
             true    mean     sd 0.025quant 0.5quant 0.975quant    mode
Theta1 for i   -1 -1.0107 0.0333    -1.0753  -1.0111    -0.9447 -1.0121
Theta2 for i    2  2.0656 0.1059     1.8481   2.0698     2.2631  2.0824
Theta3 for i    1  1.0675 0.0443     0.9814   1.0672     1.1552  1.0661
```

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

7.5 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    mode
Theta1 for i   -1 -1.0496 0.1817    -1.4073  -1.0494    -0.6936 -1.0489
Theta2 for i    2  2.1947 0.1862     1.8175   2.1996     2.5481  2.2141
Theta3 for i   -1 -0.9677 0.2424    -1.4429  -0.9684    -0.4900 -0.9704
```

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

```
             True    mean     sd 0.025quant 0.5quant 0.975quant    mode
Theta1 for i   -1 -0.9624 0.1771    -1.3091  -0.9630    -0.6124 -0.9650
Theta2 for i    2  2.1677 0.1899     1.7823   2.1734     2.5263  2.1901
Theta3 for i    1  0.9308 0.2413     0.4562   0.9311     1.4041  0.9321
```

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 7.3 with commands below

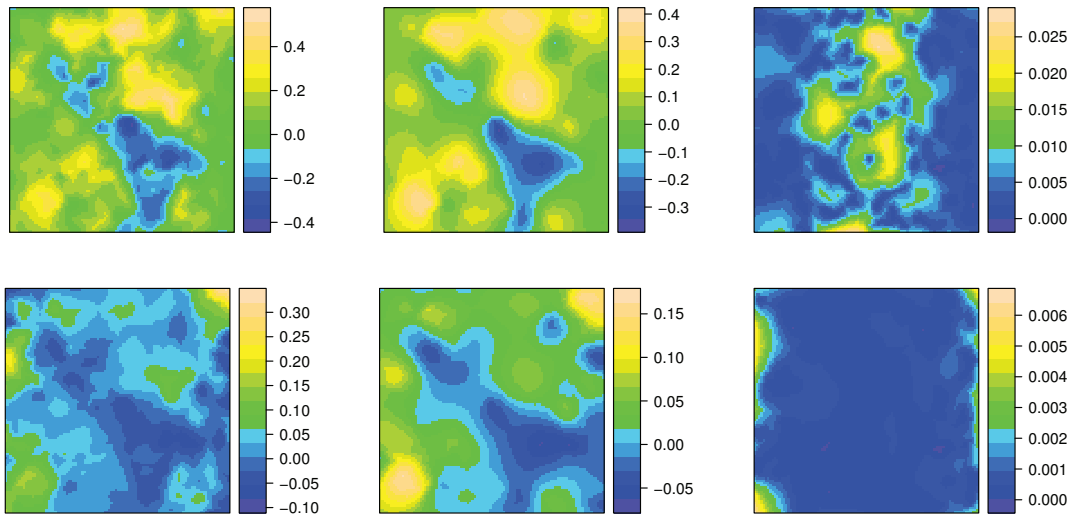

Figure 7.3: Simulated (top and bottom left), posterior mean (top and bottom mid) and the posterior variance (top and bottom right) for both random fields.

```
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 7.3 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.

Chapter 8

Point process: inference on the log-Cox process

Under the log-Cox model assumption, there is a latent Gaussian Random Field (LGRF) and the inference can be done using **INLA** [Illian et al., 2012].

A common aproach to fit the log-Cox process is to divide the study region into cells, that forms a lattice, and count the number of points into each one. This counts are modeled using the Poisson likelihood.

A good approach for inference of log-Cox model is to use the SPDE approach instead occurence counts on cells, [Simpson et al., 2011]. There are the main advantages on this approach the consideration of the loc-Cox likelihood directly.

8.1 Data simulation

The data simulated on this section is used also on the Chapter 10.

In this section we use the `rLGCP()` function from **spatstat** package to do the simulation. By default that function do simulation on window over the $(0,1)\times(0,1)$ square. We choose to do simulation over the $(0,3)\times(0,3)$ square.

```
require(spatstat)
win <- owin(c(0,3), c(0,3))
```

This function uses the `GaussRF()` function from the **RandomFields** package. The `rLGCP` uses the `GaussRF()` function to do simulation of the LGRF over a grid on the provided window and use it to do the point process simulation.

There is an internal parameter to control the resolution of the grid. We change it to

```
spatstat.options(npixel=300)
```

First we define the model parameter for the model is the mean of the LGRF. This is directly related to expected number of points of the spatial pattern. The expected number of points is its exponential times the area fo the window. We use

```
beta0 <- 3
```

So, the expected number of points is

```
exp(beta0) * diff(range(win$x)) * diff(range(win$y))
```

```
[1] 180.7698
```

Is also possible to use a functional, see Chapter 9.

On the estimation process we use the Matern covariance function with $\nu = 1$. So, here we just fix it on this value. The other parameters are the variance and scale

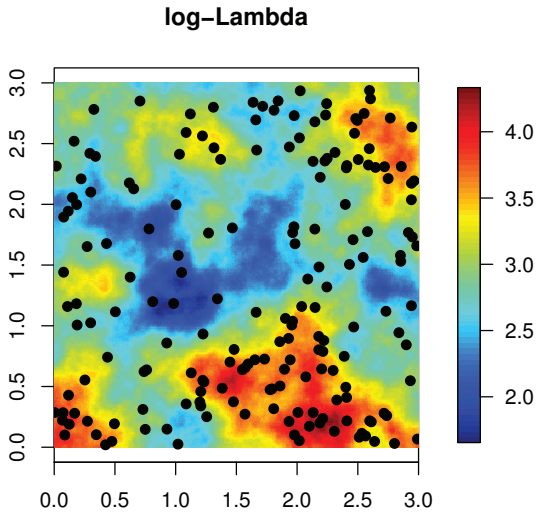

Figure 8.1: Simulated intensity of the point process (left), simulated point pattern (right).

```
sigma2x <- 0.2;      kappa <- 2
```

Doing the simulation

```
set.seed(1)
lg.s <- rLGCP('matern', beta0,
                c(0, variance=sigma2x, nugget=0, scale=1/kappa, nu=1), win=win)
```

Both, the LGRF and the point pattern, are returned. The point pattern locations are

```
(n <- nrow(xy <- cbind(lg.s$x, lg.s$y)[,2:1]))
```

```
[1] 185
```

The exponential of simulated values of the LGRF are returned as the `Lambda` attribute of the object. We extract the Λ and see a summary of the $log(\Lambda)$ bellow

```
Lam <- attr(lg.s, 'Lambda')
summary(as.vector(rf.s <- log(Lam$v)))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.655   2.630   2.883   2.937   3.232   4.333
```

On the Figure 8.1 we can see the simulated LGRF over the grid and the point pattern simulated

```
par(mfrow=c(1,1))
require(fields)
image.plot(list(x=Lam$yrow, y=Lam$xcol, z=rf.s), main='log-Lambda', asp=1)
points(xy, pch=19)
```

8.2 Inference

Following [Simpson et al., 2011] we can estimate the parameters of the log-Cox point process model using few command lines.

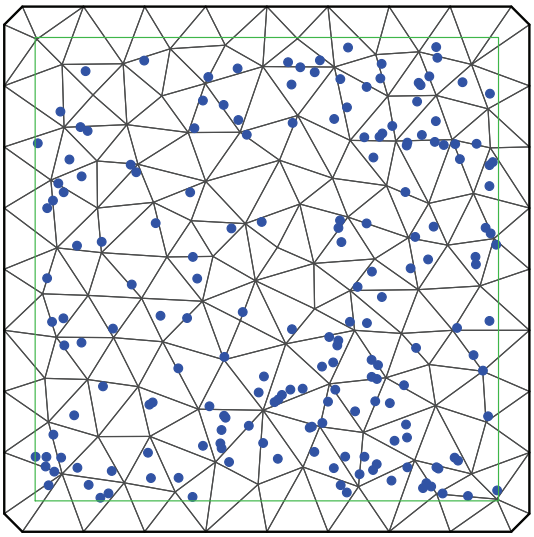

Figure 8.2: Mesh used to inference for the log-Cox process.

8.2.1 The mesh and the weights

To do inference for the log-Cox point process model we also need some care on building the mesh and on using it.

To do inference for the log Cox process, it is not necessarly better to have any location points as any of the mesh nodes, as on the geostatistical analysis where it helps a bit for the estimation of the nugget effect, see 3.5. We just need a mesh that covers the study region. So, we use the `loc.domain` argument to build the mesh.

An additional thing is that we ignore the second outer extension and we use a small first outer extension. This is because it is not necessary to have nodes out of the study region when it recieves zero weights (see weight computation bellow).

```
loc.d <- 3*t(matrix(c(0,0,1,0,1,1,0,1,0,0), 2))
(nv <- (mesh <- inla.mesh.2d(loc.d=loc.d, off=.2, max.e=.5, cut=.1))$n)
```

```
[1] 132
```

which is visualized at Figure 8.2 with following commands

```
par(mar=c(0,0,0,0))
plot(mesh, asp=1, main='')
points(xy, col=4, pch=19); lines(loc.d, col=3)
```

Defining the SPDE model

```
spde <- inla.spde2.matern(mesh=mesh, alpha=2)
```

The SPDE approach defines the model on the nodes of the mesh. To fit the the log-Cox point process model these points are considered the integration points. Them, the *relative* area of these poins are considered proportional to the expected number of events. It means that at the node on the mesh with has the larger edges we have larger expected value. The `diag(spde$param.inla$M0)` gives this value.

But, the mesh has nodes out of the domain and we have area larger than our domain when we have nodes outer the domain:

```
sum(diag(spde$param.inla$M0))
```

```
[1] 11.53255
```

We can use these values for the nodes on the inner domain and with its neighbours also inside the domain. For the nodes near the boundary it becomes hard to deal with. If it happens to be any node out of the study region and not connected to any inside the istudy region, it recieve zero. So, this is way we build a mesh without a large outer extension.

First, we get the Voronoi triangulation for the mesh nodes. We can use the `deldir` function from package **deldir**, [Turner, 2014] to compute it.

```
require(deldir)
dd <- deldir(mesh$loc[,1], mesh$loc[,2])
```

Second, we get the polygons (around each mesh nodes) with

```
tiles <- tile.list(dd)
```

These polygons are defined in a special way. For each of the reference points (the mesh nodes in our case), the polygon around is defined in a way that any location inside this polygon is closer to the respective reference point, rather another of the reference points. We can see these polygons on Figure 8.3.

Third, we get the interection polygons between these polygons and the study region polygon, using functions from the **gpclib** package:

```
require(gpclib)
area.poly(pl.study <- as(loc.d, 'gpc.poly'))
```

```
[1] 9
```

```
sum(w <- sapply(tiles, function(p) area.poly(intersect(
      as(cbind(p$x, p$y), 'gpc.poly'), pl.study))))
```

```
[1] 9
```

and we have some points without any weights (the red ones on Figure 8.3).

```
table(w>0)
```

```
FALSE  TRUE
   19   113
```

At Figure 8.3 we can see the Voronoi polygons for the mesh nodes. We can see that the polygons around some mesh does not intersect the domain study area.

```
par(mar=c(2,2,1,1), mgp=2:0)
plot(mesh$loc, asp=1, col=(w==0)+1, pch=19, xlab='', ylab='')
for (i in 1:length(tiles))
      lines(c(tiles[[i]]$x, tiles[[i]]$x[1]), c(tiles[[i]]$y, tiles[[i]]$y[1]))
lines(loc.d, col=3)
```

8.2.2 The data and projector matrices

This vector is just what we need to use as the exposure (expected) for the Poisson likelihood and is related to the augumented data that we need to fit using the Poisson likelihood. We can specify that the first observations (number of nodes) are zero and the last are ones (number of events).

```
y.pp <- rep(0:1, c(nv, n))
```

So, the expected vector can be defined by

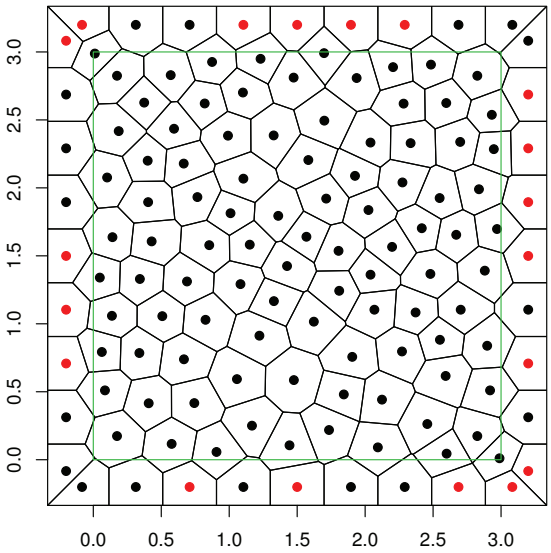

Figure 8.3: Voronoy polygons for the mesh used to inference for the log-Cox process.

```
e.pp <- c(w, rep(0, n))
```

We must have to define the projector matrix to do inference using the SPDE approach, [Lindgren, 2012]. For the observed points locations we have

```
lmat <- inla.spde.make.A(mesh, xy)
```

We need also a projector matrix for the integration points and this is just a diagonal matrix because this locations are just the mesh vertices.

```
imat <- Diagonal(nv, rep(1, nv))
```

So, the entire projector matrix is

```
A.pp <- rBind(imat, lmat)
```

The data stack can be made by

```
stk.pp <- inla.stack(data=list(y=y.pp, e=e.pp),
                     A=list(1,A.pp), tag='pp',
                     effects=list(list(b0=rep(1,nv+n)), list(i=1:nv)))
```

8.2.3 Posterior marginals

The posterior marginals for the parameters of the log-Cox model (on the SPDE approach scale) and for the latent random field at integration and location points are obtained by

```
pp.res <- inla(y ~ 0 + b0 + f(i, model=spde),
               family='poisson', data=inla.stack.data(stk.pp),
               control.predictor=list(A=inla.stack.A(stk.pp)),
               E=inla.stack.data(stk.pp)$e)
```

To get the model parameters on the user-scale paramenters such as the scale κ, nominal variance σ_x^2 and nominal range we use the `inla.spde2.result()` function

```
pp.rf <- inla.spde2.result(pp.res, 'i', spde)
```

The posterior distribution of the log-Cox model parameters are visualized on the Figure 8.4.

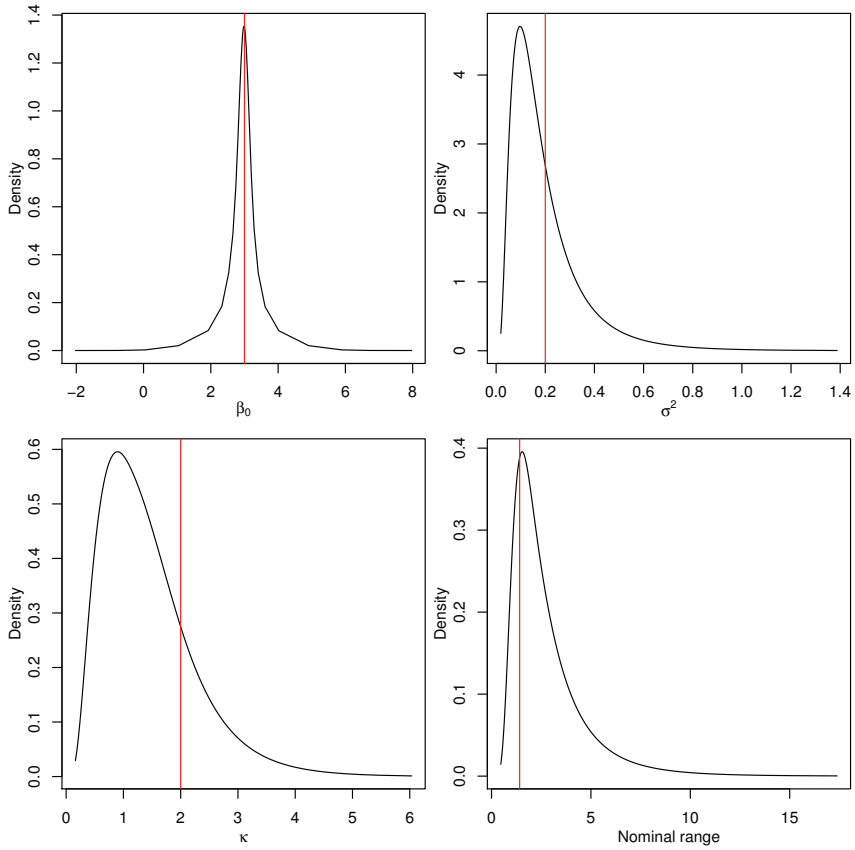

Figure 8.4: Posterior distribution for the parameters of the log-Cox model σ^2 (left), κ (mid) and the nominal range (right)

```
par(mfrow=c(2,2), mar=c(3,3,1,0.3), mgp=c(2,1,0))
plot(pp.res$marginals.fix[[1]], type='l',
       xlab=expression(beta[0]), ylab='Density')
abline(v=beta0, col=2)
plot(pp.rf$marginals.variance.nominal[[1]], type='l',
       xlab=expression(sigma^2), ylab='Density')
abline(v=sigma2x, col=2)
plot(pp.rf$marginals.kappa[[1]], type='l',
       xlab=expression(kappa), ylab='Density')
abline(v=kappa, col=2)
plot(pp.rf$marginals.range.nominal[[1]], type='l',
       xlab='Nominal range', ylab='Density')
abline(v=sqrt(8*1)/kappa, col=2)
```

Chapter 9

Including a covariate on the log-Cox process

In Chapter 8 we have done simulation considering the underline intensity as just the exponential of a realization of a Gaussian random field. In this chapter we consider that we have an additional effect, which is treated as a covariate. In order to fit the model, it is needed the covariate value everywhere, at the location points and at the integration points.

9.1 Covariate everywhere

The simulation is done considering that the covariate effect is available at the same grid points where the Gaussian process is simulated. So, first we create an artificial covariate at the grid

```
y0 <- x0 <- seq(win$xrange[1], win$xrange[2],
                  length=spatstat.options()$npixel)
gridcov <- outer(x0, y0, function(x,y) cos(x) - sin(y-2))
```

Now, the expected number of points is function of the covariate

```
beta1 <- -0.5
sum(exp(beta0 + beta1*gridcov) * diff(x0[1:2])*diff(y0[1:2]))
```

```
[1] 169.1388
```

Doing the simulation

```
set.seed(1)
lg.s.c <- rLGCP('matern', im(beta0 + beta1*gridcov, xcol=x0, yrow=y0),
                  c(0, variance=sigma2x, nugget=0, scale=1/kappa, nu=1), win=win)
```

Both, the LGRF and the point pattern, are returned. The point pattern locations are

```
(n.c <- nrow(xy.c <- cbind(lg.s.c$x, lg.s.c$y)[,2:1]))
```

```
[1] 193
```

On the Figure 9.1 we can see the covariate values and the simulated LGRF over the grid

```
require(fields)
par(mfrow=c(1,2), mar=c(2,2,1,1), mgp=c(1,0.5,0))
image.plot(list(x=x0, y=y0, z=gridcov), main='Covariate', asp=1)
image.plot(list(x=x0, y=y0, z=log(attr(lg.s.c, 'Lambda')$v)),
             main='log-Lambda', asp=1)
points(xy.c, pch=19)
```

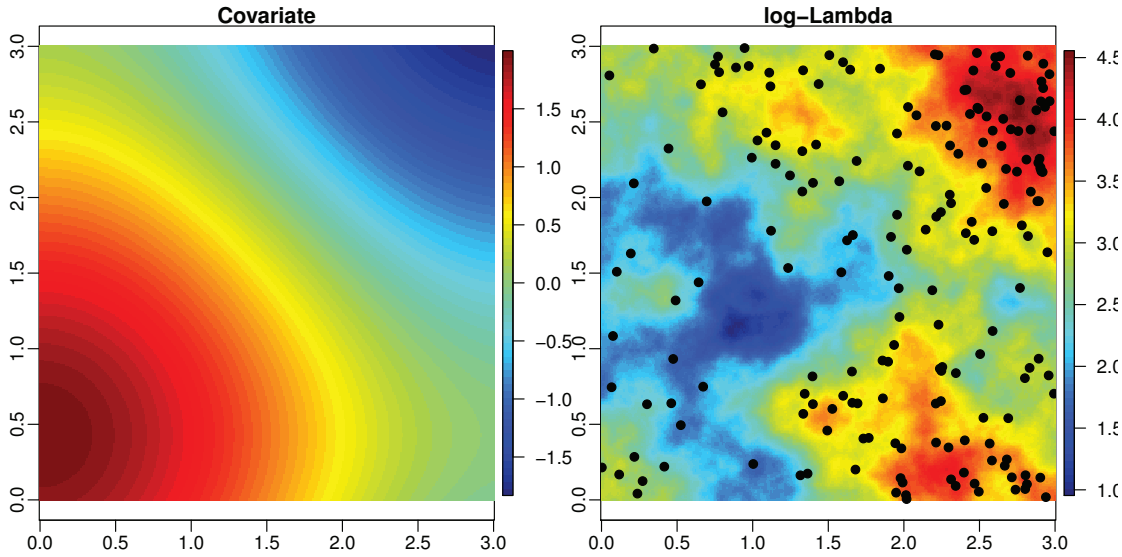

Figure 9.1: Covariate (left), simulated intensity of the point process (mid), simulated point pattern (right).

9.2 Inference

We have to include the covariate values to do the inference. We need to collect it at the point pattern locations and at the mesh nodes from the grid.

We collect the covariate with the command bellow

```
covariate = gridcov[Reduce('cbind', nearest.pixel(
      c(mesh$loc[,1], xy.c[,1]), c(mesh$loc[,2], xy.c[,2]),
      im(gridcov, x0, y0)))]
```

The augumented response data is created in same way as before.

```
y.pp.c <- rep(0:1, c(nv, n.c))
e.pp.c <- c(w, rep(0, n.c))
```

The projector matrix for the observed points locations

```
lmat.c <- inla.spde.make.A(mesh, xy.c)
```

The entire projector matrix, using the previous for the integration points, is

```
A.pp.c <- rBind(imat, lmat.c)
```

The data stack is

```
stk.pp.c <- inla.stack(data=list(y=y.pp.c, e=e.pp.c),
                       A=list(1, A.pp.c), tag='pp.c',
                       effects=list(list(b0=1, covariate=covariate),
                           list(i=1:nv)))
```

The model is fitted by

```
pp.c.res <- inla(y ~ 0 + b0 + covariate + f(i, model=spde),
                 family='poisson', data=inla.stack.data(stk.pp.c),
                 control.predictor=list(A=inla.stack.A(stk.pp.c)),
                 E=inla.stack.data(stk.pp.c)$e)
```

Getting the model parameters on the user-scale

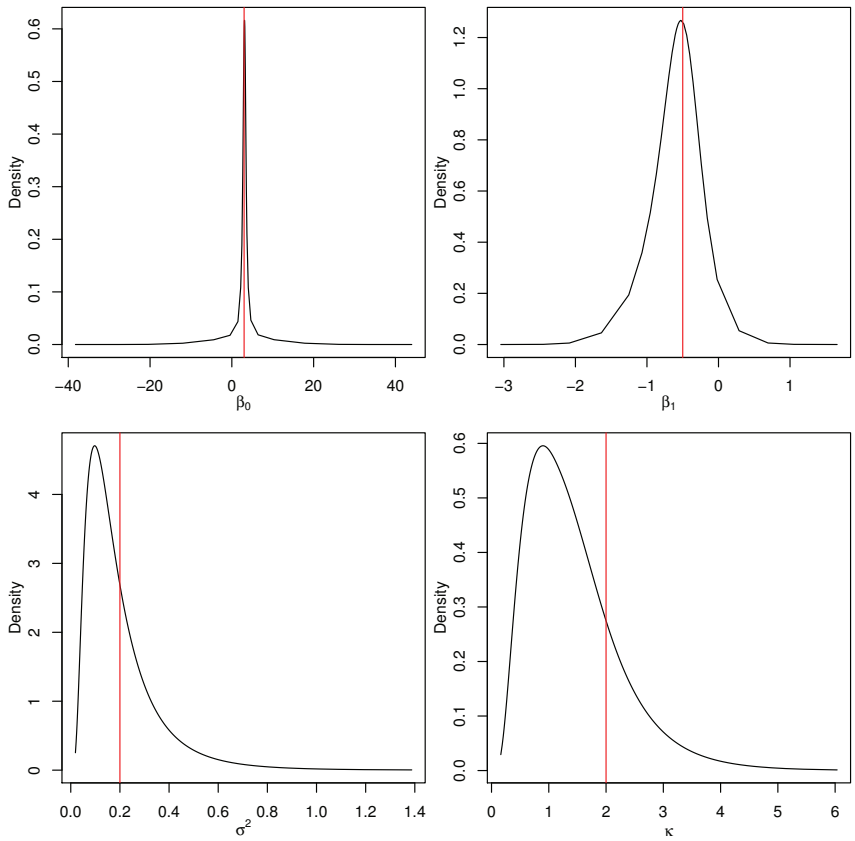

Figure 9.2: Posterior distribution for the intercept (top left), coefficient of the covariate (top right) and the parameters of the log-Cox model σ^2 (bottom left), κ (bottom right)

```
pp.c.rf <- inla.spde2.result(pp.c.res, 'i', spde)
```

The posterior distribution of the log-Cox model parameters are visualized on the Figure 9.2.

```
par(mfrow=c(2,2), mar=c(3,3,1,0.3), mgp=c(2,1,0))
plot(pp.c.res$marginals.fix[[1]], type='l', ylab='Density',
       xlab=expression(beta[0])); abline(v=beta0, col=2)
plot(pp.c.res$marginals.fix[[2]], type='l', ylab='Density',
       xlab=expression(beta[1])); abline(v=beta1, col=2)
plot(pp.rf$marginals.variance.nominal[[1]], type='l', ylab='Density',
       xlab=expression(sigma^2)); abline(v=sigma2x, col=2)
plot(pp.rf$marginals.kappa[[1]], type='l', ylab='Density',
       xlab=expression(kappa)); abline(v=kappa, col=2)
```

Chapter 10

Geostatistical inference under preferential sampling

In some cases the effort on sampling depends on the response. For example, is more common to have stations collecting data about pollution on industrial area than on rural ones. To make inference in this case, we can test if we have a preferential sampling problem in our data. One approach is to build a joint model considering a log-Cox model for the point pattern (the locations) and the response, [Diggle et al., 2010]. So, we need also to make inference for a point process model jointly.

An illustration of the use **INLA** for the preferential sampling problem is on the case studies section of the **INLA** web page, precisely on `http://www.r-inla.org/examples/case-studies/diggle09`. This example uses the two dimentional random walk model for the latent random field. Here, we show geoestatistical inference under preferencial sampling using SPDE.

We use the values of the LGRF simulated on the Chapter 8 to define the response. We just take the values of closest grid centers to each location of the point pattern. The values of the LGRF is collected (and a summary) at closest grid centers with

```
summary(z <- log(t(Lam$v)[Reduce(
    'cbind', nearest.pixel(xy[,1], xy[,2], Lam))]))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.784   2.805   3.157   3.178   3.611   4.274
```

These values are the latent field with zero mean plus the defined intercept. We define the response as a different intercept β_y and multiply the zero mean random field with a $1/\beta$, where β is the parameter as the sharing parameter between the intensity of the point process locations and the response. Considering $\beta < 0$, it means that the response values is inversly proportional to the points density.

```
beta0.y <- 10;   beta <- -1;   prec.y <- 16
set.seed(2)
summary(resp <- beta0.y + (z-beta0)/beta +
          rnorm(length(z), 0, sqrt(1/prec.y)))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  8.302   9.397   9.846   9.825  10.260  11.220
```

10.1 Fitting the usual model

Here, we just fit the geoestatistical model using the usual approach. In this approach we just use the locations as fixed. We use the mesh of the previous Chapter.

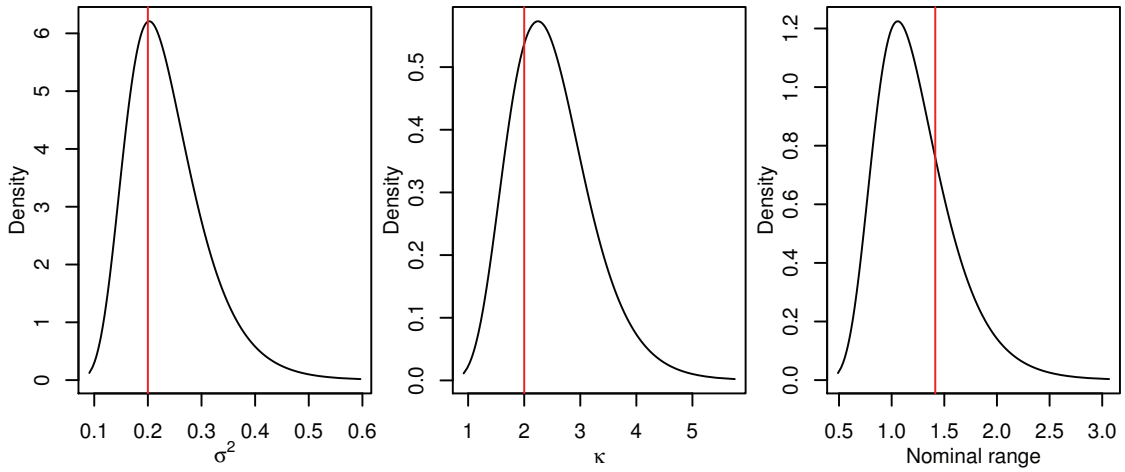

Figure 10.1: Posterior distribution for σ^2, κ and the nominal range just using the response.

```
stk.u <- inla.stack(data=list(y=resp), A=list(lmat, 1),
                    effects=list(i=1:nv, b0=rep(1,length(resp))))
u.res <- inla(y ~ 0 + b0 + f(i, model=spde),
              data=inla.stack.data(stk.u),
              control.predictor=list(A=inla.stack.A(stk.u)))
```

Summaries

```
round(cbind(True=c(beta0y=beta0.y, prec.y=prec.y),
            rbind(u.res$summary.fix[, 1:6], u.res$summary.hy[1,])), 4)
```

```
       True    mean     sd 0.025quant 0.5quant 0.975quant    mode
beta0y   10  9.9623 0.2563     9.4261   9.9627    10.4954  9.9633
prec.y   16 11.1307 1.4346     8.5534  11.0495    14.1825 10.8978
```

We have to build also the posterior marginals on the user-scale paramenters: scale, nominal variance and nominal range

```
u.rf <- inla.spde2.result(u.res, 'i', spde)
```

In the Figure 10.1 we see the marginals posterior distributions for the model parameters with code bellow

```
par(mfrow=c(1,3), mar=c(3, 3, 0.3, 0.3), mgp=c(2,1,0))
plot(u.rf$marginals.variance.nominal[[1]], type='l',
       xlab=expression(sigma^2), ylab='Density')
abline(v=sigma2x, col=2)
plot(u.rf$marginals.kappa[[1]], type='l',
       xlab=expression(kappa), ylab='Density')
abline(v=kappa, col=2)
plot(u.rf$marginals.range.nominal[[1]], type='l',
       xlab='Nominal range', ylab='Density')
abline(v=sqrt(8*1)/kappa, col=2)
```

10.2 Estimation under preferential sampling

In this situation we fit the model where a LGRF is considered to model both point pattern and the response. Using **INLA** it can be done using two likelihoods, one for the point

pattern and another for the response. To do it we need a matrix response and a new index set to specify the model for the LGRF. It is more easy by using the `inla.stack()` following previous examples for two likelihood models.

We consider the point pattern 'observation' on the first column and the response values on the second column. So, we just redefine the stack for the response and also for the point process. We put the response on the first column and the Poisson data for the point process as the second column. Also, to avoid the expected number of cases as NA for the Poisson likelihood, we set it as zero on the response data stack. For the SPDE effect on the point process part we have to model it as a copy of the SPDE effect at response part. We do it by defining a index set with different name and use it on the copy feature later.

```
stk2.y <- inla.stack(data=list(y=cbind(resp,NA), e=rep(0,n)),
                        A=list(lmat, 1), tag='resp2',
                        effects=list(i=1:nv, b0.y=rep(1,n)))
stk2.pp <- inla.stack(data=list(y=cbind(NA,y.pp), e=e.pp),
                         A=list(A.pp, 1), tag='pp2',
                         effects=list(j=1:nv, b0.pp=rep(1,nv+n)))
```

and join both together

```
j.stk <- inla.stack(stk2.y, stk2.pp)
```

Now, we fit the geostatistical model under preferential sampling. To put the LGRF on both likelihood, we have to use the copy strategy.

```
j.res <- inla(y ~ 0 + b0.pp + b0.y + f(i, model=spde) +
                f(j, copy='i', fixed=FALSE),
                data=inla.stack.data(j.stk),
                family=c('gaussian', 'poisson'),
                control.predictor=list(A=inla.stack.A(j.stk)),
                E=inla.stack.data(j.stk)$e)
round(cbind(True=c(beta0, beta0.y), j.res$summary.fix), 4)
```

```
      True   mean     sd 0.025quant 0.5quant 0.975quant   mode kld
b0.pp    3 3.0161 0.1544     2.6967   3.0180     3.3261 3.0216   0
b0.y    10 9.9813 0.2746     9.4073   9.9818    10.5512 9.9826   0
```

Computing the marginals posterior distributions on the user-scale random field parameters

```
j.rf <- inla.spde2.result(j.res, 'i', spde)
```

We can visualize the posterior distribution on the Figure 10.2. The negative signal to the copy parameter β is due to the fact that we define the response with opposite signal of the latent field used to do the simulation of the point process.

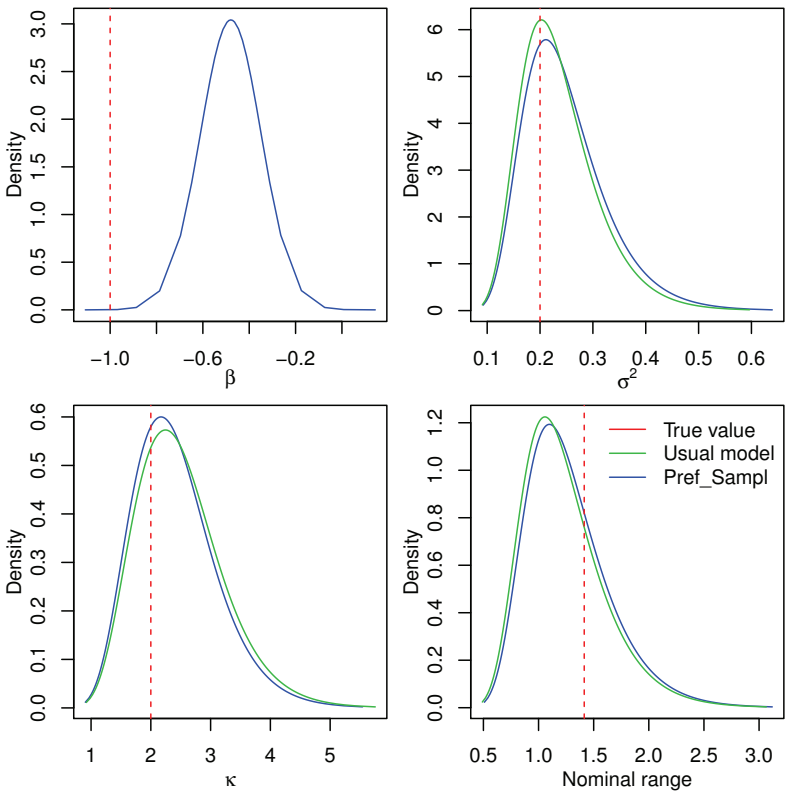

Figure 10.2: Posterior distribution for β_0, σ^2, κ and the nominal range under preferential sampling.

Chapter 11

A space time example

In this chapter we show an example on fitting a spatio-temporal model. This model is a separable one described on [Cameletti et al., 2012]. Basically the model is defined as a SPDE model for the spatial domain and another one for the temporal domain. The spatio-temporal separable model is defined by the kronecker product between the precision of these two models.

In this example we show that it is allowed to have different locations on different times. Also, we show the use of a categorical covariate.

11.1 Data simulation

We use the Paraná state border, available on **INLA** package, as the domain.

```
data(PRborder)
```

We start by defining the spatial model. Because we need that the example run faster, we use the low resolution mesh for Paraná state border created on the section 2.2.

There is two options to simulate from Cameletti's model. One is based on the marginal distribution of the latent field and another is on the conditional distribution at each time. The second one is easy for us by use the function defined on the section 2.4 to simulate an independent random field realization each time.

First we define a set of points as the same on the `PRprec` data

```
data(PRprec)
coords <- as.matrix(PRprec[,1:2])
```

and set $k = 12$, the time dimention

```
k <- 12
```

The k independent realizations can be done by

```
params <- c(variance=1, kappa=1/2)
set.seed(1)
x.k <- rspde(coords, kappa=1/2, n=k, mesh=prmesh1)
```

```
theta = -0.5723649 -0.6931472
```

```
dim(x.k)
```

```
[1] 616  12
```

Now, we define the autoregressive parameter ρ

```
rho <- 0.7
```

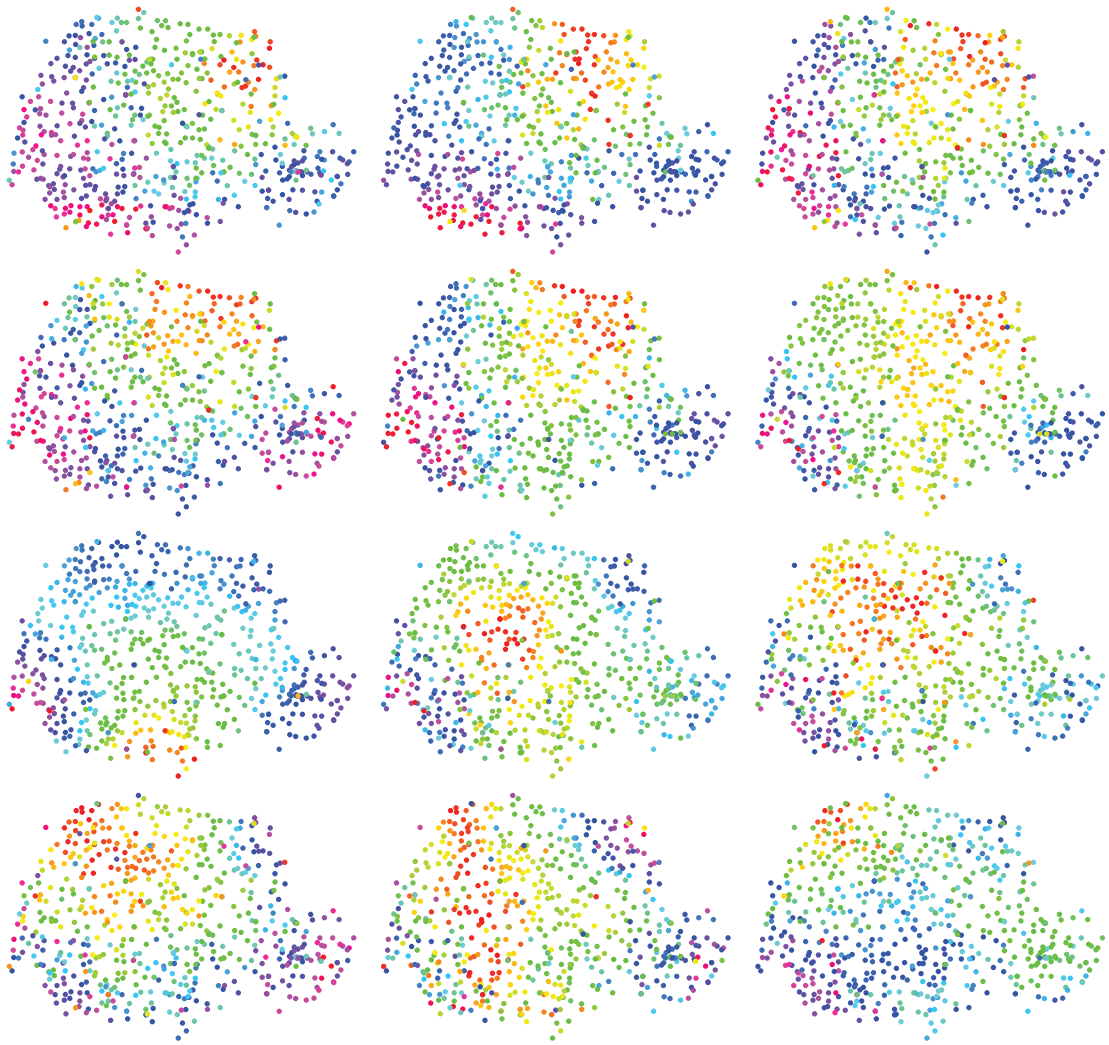

Figure 11.1: Realization of the spatio temporal random field.

and get the sample

```
x <- x.k
for (j in 2:k)
      x[,j] <- rho*x[,j-1] + x.k[,j]
```

We can visualize the realization at the figure 11.1 with commands bellow

```
c100 <- rainbow(101)
par(mfrow=c(4,3), mar=c(0,0,0,0))
for (j in 1:k)
    plot(coords, col=c100[round(100*(x[,j]-min(x[,j]))/diff(range(x[,j])))],
         axes=FALSE, asp=1, pch=19, cex=0.5)
```

In this example we need to show the use of a categorical covariate. First we do the simulation of the covariate as

```
n <- nrow(coords)
set.seed(2)
table(ccov <- factor(sample(LETTERS[1:3], n*k, replace=TRUE)) )
```

```
   A    B    C
2458 2438 2496
```

and the regression parameters as

```
beta <- -1:1
```

The response is

```
y <- beta[unclass(ccov)] + x
tapply(y, ccov, mean)
```

```
       A         B         C
-2.464774 -1.417898 -0.497180
```

To show that is allowed to have different locations on different times, we drop some of the observations. We do it by just selecting a half of the simulated data. We do it by creating a index for the selected observations

```
isel <- sample(1:(n*k), n*k/2)
```

and we organize the data on a `data.frame`

```
dat <- data.frame(y=as.vector(y), w=ccov,
                  time=rep(1:k, each=n),
                  xcoo=rep(coords[,1], k),
                  ycoo=rep(coords[,2], k))[isel, ]
```

In real applications some times we have completely missaligned locations between different times. The code provided here to fit the model still work on this situation.

11.2 Data stack preparation

Now, we need the data preparation to build the spatio-temporal model. The index set is made taking into account the number of weights on the SPDE model and the number of groups

```
spde <- attr(x.k, 'spde')
iset <- inla.spde.make.index('i', n.spde=spde$n.spde, n.group=k)
```

Notice that the index set for the latent field is not depending on the data set locations. It only depends on the SPDE model size and on the time dimention.

The projector matrix must be defined considering the coordinates of the observed data. We have to inform the time index for the group to build the projector matrix. This also must be defined on the `inla.spde.make.A()` function

```
A <- inla.spde.make.A(mesh=prmesh1,
                      loc=cbind(dat$xcoo, dat$ycoo),
                      group=dat$time)
```

The effects on the stack are a list with two elements, one is the index set and another the categorical covariate. The stack data is defined as

```
sdat <- inla.stack(tag='stdata', data=list(y=dat$y),
                   A=list(A,1),  effects=list(iset, w=dat$w))
```

11.3 Fitting the model and some results

We set the prior on the temporal autoregressive parameter with zero mean and variance equals 5, and define the initial value equals the true with

```
h.spec <- list(theta=list(initial=0.7, param=c(0, 5)))
```

The likelihood hyperparameter is fixed on a hight precision, just because we haven't noise. We choose the `strategy='gaussian'` to it run faster. To deal with the categorical covariate we need to set `expand.factor.strategy='inla'` on the `control.fixed` argument list.

```
formulae <- y ~ 0 + w +
      f(i, model=spde, group=i.group,
        control.group=list(model='ar1', hyper=h.spec))
res <- inla(formulae,  data=inla.stack.data(sdat),
              control.predictor=list(compute=TRUE, A=inla.stack.A(sdat)),
              control.family=list(initial=20, fixed=TRUE),
              control.inla=list(strategy='gaussian'),
              control.fixed=list(expand.factor.strategy='inla'))
```

The summary of the covariate coefficients (together the observed mean for each covariate level)

```
round(cbind(observed=tapply(dat$y, dat$w, mean), res$summary.fixed), 4)
```

```
  observed    mean     sd 0.025quant 0.5quant 0.975quant    mode kld
A  -2.4242 -1.8982 1.0516    -3.9886  -1.8993     0.1962 -1.9014   0
B  -1.5023 -0.8982 1.0516    -2.9886  -0.8993     1.1962 -0.9014   0
C  -0.5389  0.1018 1.0516    -1.9886   0.1007     2.1962  0.0986   0
```

The summary for the posterior marginal distribution of the temporal correlation

```
round(res$summary.hy[3,], 5)
```

```
     mean         sd 0.025quant   0.5quant 0.975quant       mode
  0.67718    0.01801    0.64325    0.67657    0.71365    0.67422
```

Bellow, we get the marginals distributions for random field parameters on the user scale

```
rf <- inla.spde2.result(res, 'i', attr(x.k, 'spde'), do.transf=TRUE)
```

and we see these distributions, also the marginal ditribution for the temporal correlation, on the Figure 11.2 with the commands bellow

```
par(mfrow=c(2,2), mar=c(3,3,1,0.1), mgp=2:0)
plot(res$marginals.hyper[[3]], type='l',
       xlab=expression(beta), ylab='Density')
abline(v=rho, col=2)
plot(rf$marginals.variance.nominal[[1]], type='l',
       xlab=expression(sigma[x]), ylab='Density')
abline(v=params[1], col=2)
plot(rf$marginals.kappa[[1]], type='l',
       xlab=expression(kappa), ylab='Density')
abline(v=params[2], col=2)
plot(rf$marginals.range.nominal[[1]], type='l',
       xlab='range nominal', ylab='Density')
abline(v=sqrt(8)/params[2], col=2)
```

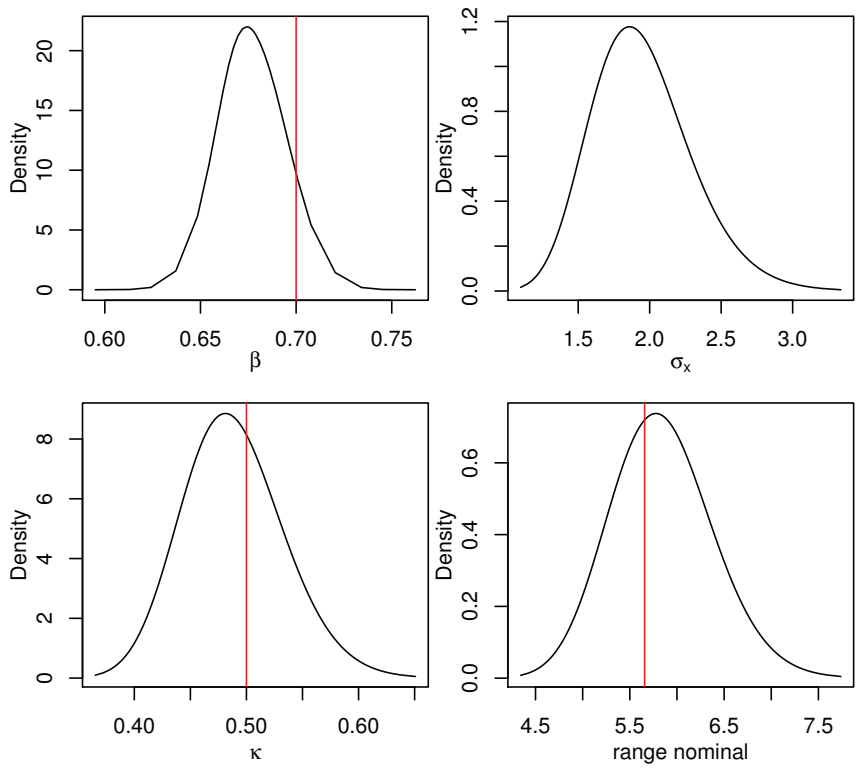

Figure 11.2: Marginal posterior distribution for β (top left), σ_x (top right), κ (bottom left) and the nominal range (bottom right). The red vertical lines are placed at true value.

11.4 A look at the posterior random field

The first look at the random field posterior distribution is to compare the realized random field with the posterior mean, median or/and mode and any quantile. We just compute the correlation between the simulated data response and the posterior mean of the predicted values.

First, we found the index for the random field at data locations

```
str(idat <- inla.stack.index(sdat, 'stdata')$data)
```

```
int [1:3696] 1 2 3 4 5 6 7 8 9 10 ...
```

and compute the correlation between the the posterior mean and the response by

```
cor(dat$y, res$summary.linear.predictor$mean[idat])
```

```
[1] 1
```

We also can do prediction for each time and visualize it. First, we define the projection grid in the same way that on the example on Chapter 4.

```
stepsize <- 4*1/111
nxy <- round(c(diff(range(coords[,1])), diff(range(coords[,2])))/stepsize)
projgrid <- inla.mesh.projector(prmesh1, xlim=range(coords[,1]),
                                ylim=range(coords[,2]), dims=nxy)
```

The prediction for each time can be done by

```
xmean <- list()
for (j in 1:k)
    xmean[[j]] <- inla.mesh.project(
      projgrid, res$summary.random$i$mean[iset$i.group==j])
```

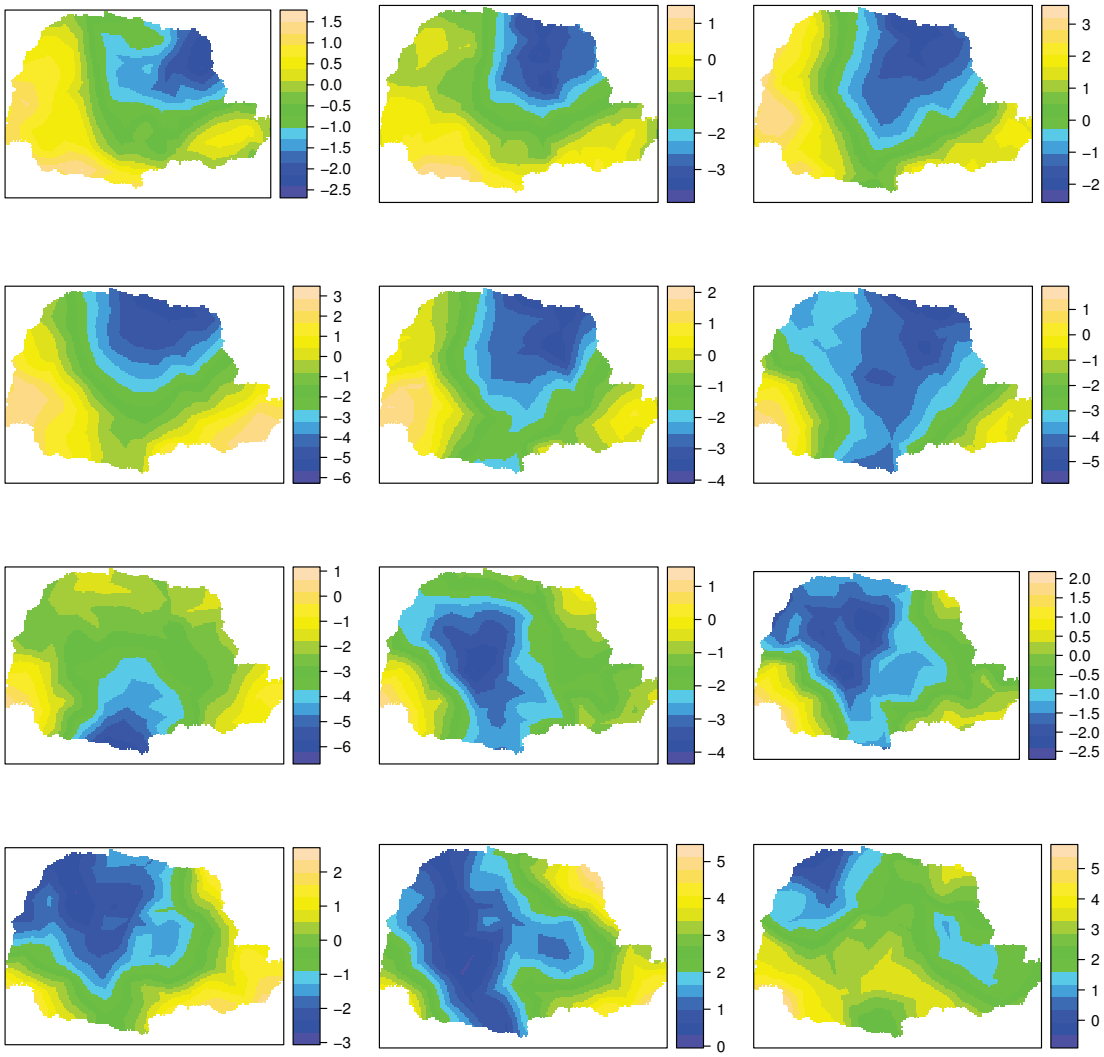

Figure 11.3: Visualization of the posterior mean of the spatio temporal random field.

We found what points of the grid are inside the Paraná state border.

```
require(splancs)
xy.in <- inout(projgrid$lattice$loc, cbind(PRborder[,1], PRborder[,2]))
```

And, to get better visualization, we set NA to the points of the grid out of the Paraná border.

```
for (j in 1:k)   xmean[[j]][!xy.in] <- NA
```

The visualization at Figure 11.3 can be made by the comands bellow

```
require(gridExtra)
do.call(function(...) grid.arrange(..., nrow=4),
          lapply(xmean, levelplot,  xlab='', ylab='',
                 col.regions=topo.colors(16), scale=list(draw=FALSE)))
```

11.5 Validation

The results on previous section are done using part of the simulated data. Now we prepare another stack with the simulated data don't used in the simulation. This part of the simulated data are used as a evaluation data.

```
vdat <- data.frame(y=as.vector(y), w=ccov,
                   time=rep(1:k, each=n),
                   xcoo=rep(coords[,1], k),
                   ycoo=rep(coords[,2], k))[-isel, ]
Aval <- inla.spde.make.A(prmesh1, loc=cbind(vdat$xcoo, vdat$ycoo),
                         group=vdat$time)
stval <- inla.stack(tag='stval', data=list(y=NA),
                    A=list(Aval,1),  effects=list(iset, w=vdat$w))
```

Now, we just use a full data stack to fit the model

```
stfull <- inla.stack(sdat, stval)
vres <- inla(formulae,  data=inla.stack.data(stfull),
               control.predictor=list(compute=TRUE, A=inla.stack.A(stfull)),
               control.family=list(initial=20, fixed=TRUE),
               control.inla=list(strategy='gaussian'),
               control.fixed=list(expand.factor.strategy='inla'))
```

We can look at fitted values for the validation data. We can plot the predicted versus observed values to look at goodness of fit. First, we found the index for this data from full stack data.

```
ival <- inla.stack.index(stfull, 'stval')$data
```

We plot it with following commands and visualize at Figure 11.4.

```
plot(vres$summary.fitted.values$mean[ival], vdat$y,
       asp=1, xlab='Posterior mean', ylab='Observed')
abline(0:1, col=gray(.7))
```

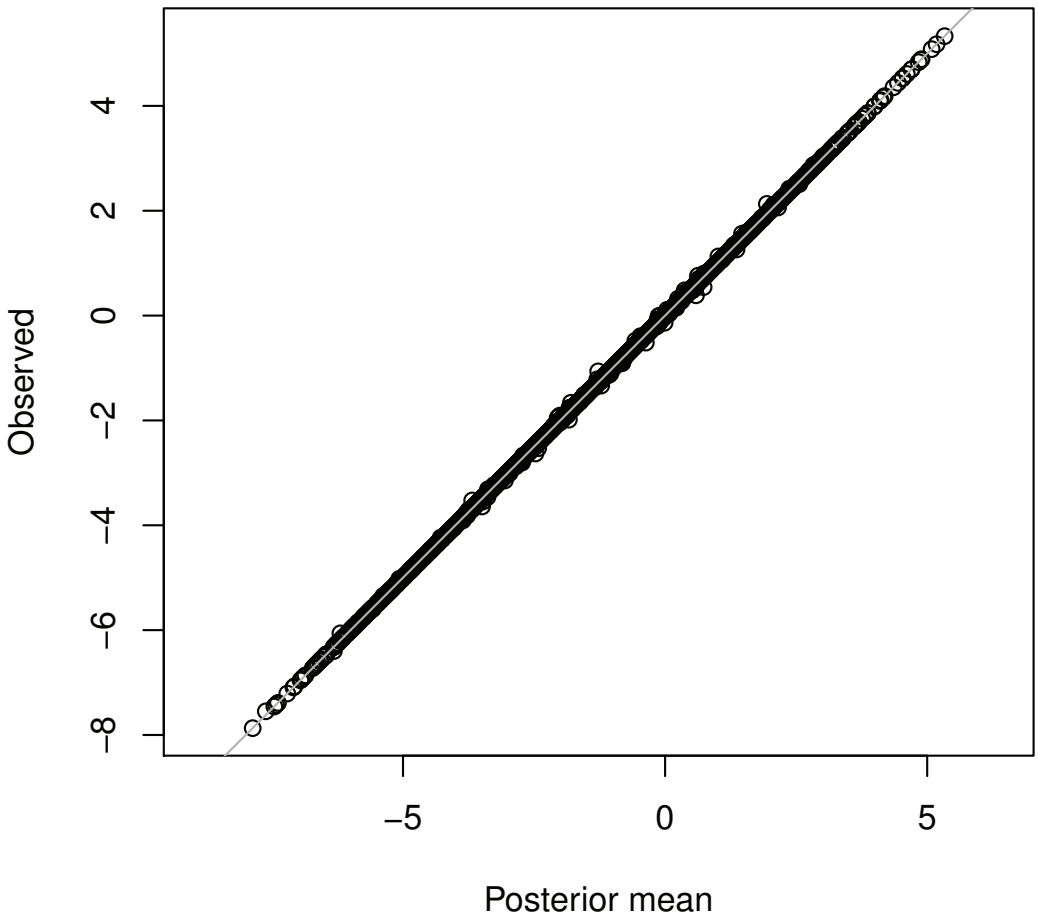

Figure 11.4: Validation: Observed versus posterior mean.

Chapter 12

Data cloning example

The data clone algorithm for spatial models is described in http://www.sciencedirect.com/science/article/pii/S0167947310004329. In the **INLA** package we have an example with the hybrid data clone when the random effect are 'iid' on http://www.math.ntnu.no/inla/r-inla.org/papers/FS8-2010-R.pdf

Here we use the hybrid data cloning approach to see what happens with the parameters of the toy spatial continuous model considering the SPDE model.

We use the toy data set

```
data(SPDEtoy)
coords <- as.matrix(SPDEtoy[,1:2])
```

with the mesh

```
(mesh <- inla.mesh.2d(coords, cutoff=0.05,
                        max.edge=c(0.1, 0.2)))$n
```

```
[1] 406
```

to build the model

```
spde <- inla.spde2.matern(mesh)
```

We consider a set of values for the number of clones. By cloning we use the `replica` option on the latent field definition function `f()`. The `inla.spde.make.index` function allow us to make the replica index.

In the following code we do a loop on the number o clones an fit the model with replications

```
n <- nrow(coords); k <- c(1,2,3,5,10); names(k) <- k
resk <- list()
for (i in 1:length(k)) {
    kk <- k[i]
    A <- inla.spde.make.A(mesh, loc=coords,
                          index=rep(1:n, kk), repl=rep(1:kk, each=n))
    ind <- inla.spde.make.index(name='i', n.spde=spde$n.spde, n.repl=kk)
    st.dat <- inla.stack(data=list(resp=rep(SPDEtoy[,3], kk)),
                         A=list(A,1),
                         effects=list(ind, list(m=rep(1,n*kk))), tag='est')
    resk[[i]] <- inla(resp ~ 0 + m + f(i, model=spde, replicate=i.repl),
                      data=inla.stack.data(st.dat),
                      control.predictor=list(A=inla.stack.A(st.dat),
                        compute=TRUE))
  }
```

Also we get the posterior marginal distribution of the marginal variance and practical range with

```
res.f <- lapply(1:length(resk), function(i)
                  inla.spde2.result(resk[[i]], 'i', spde, do.transf=TRUE))
```

We collect the results of the posterior mean and posterior variance of each parameter with

```
r <- list()
r$Intercept <- sapply(resk, function(x)
                       c(x$summary.fix[1,1], x$summary.fix[1,2]^2))
r$Likelihood.Variance <- sapply(resk, function(x) {
    d <- inla.tmarginal(function(x) 1/x, x$marginals.hyperpar[[1]])
    r <- inla.emarginal(function(x) x, d)
    c(r, inla.emarginal(function(x) (x-r)^2, d))
  })
r$Marginal.RF.Variance <- sapply(res.f, function(x) {
    e <- inla.emarginal(function(x) x, x$marginals.variance.nom[[1]])
    c(e, inla.emarginal(function(x) (x - e)^2,
                        x$marginals.variance.nominal[[1]]))
  })
r$Kappa <- sapply(resk, function(x) {
    e <- inla.emarginal(function(x) exp(x), x$marginals.hy[[3]])
    c(e, inla.emarginal(function(x) (exp(x) - e)^2, x$marginals.hy[[3]]))
  })
```

The main think in data clonning is to visualize the posterior mean and the posterior variance of the parameters. When we don't have identificability problem we see that the mean converges to the likelihood estimate and the posterior variance converges to zero.

We visualize the posterior mean and posterior variance of each parameter (β_0, σ_y^2, σ_x^2 and κ) relative to the result without clone ($k = 1$). Also, to compare, we visualize the likelihood estimate. This graphs are on the Figure 12.1 with following commands

```
par(mfrow=c(2, 2), mar=c(3,3,2,1), mgp=c(1.5,.5,0))
for (i in 1:length(r)) {
    plot(k, r[[i]][2, ]/r[[i]][2,1], type='o', ylim=c(0, 1.2),
         ylab='relative to k=1', main=names(r)[i])
    lines(k, r[[i]][1, ]/r[[i]][1,1], type='o', lty=2, col=2)
    abline(h=lk.est[i]/r[[i]][1,1], col=4, lty=3)
  }
```

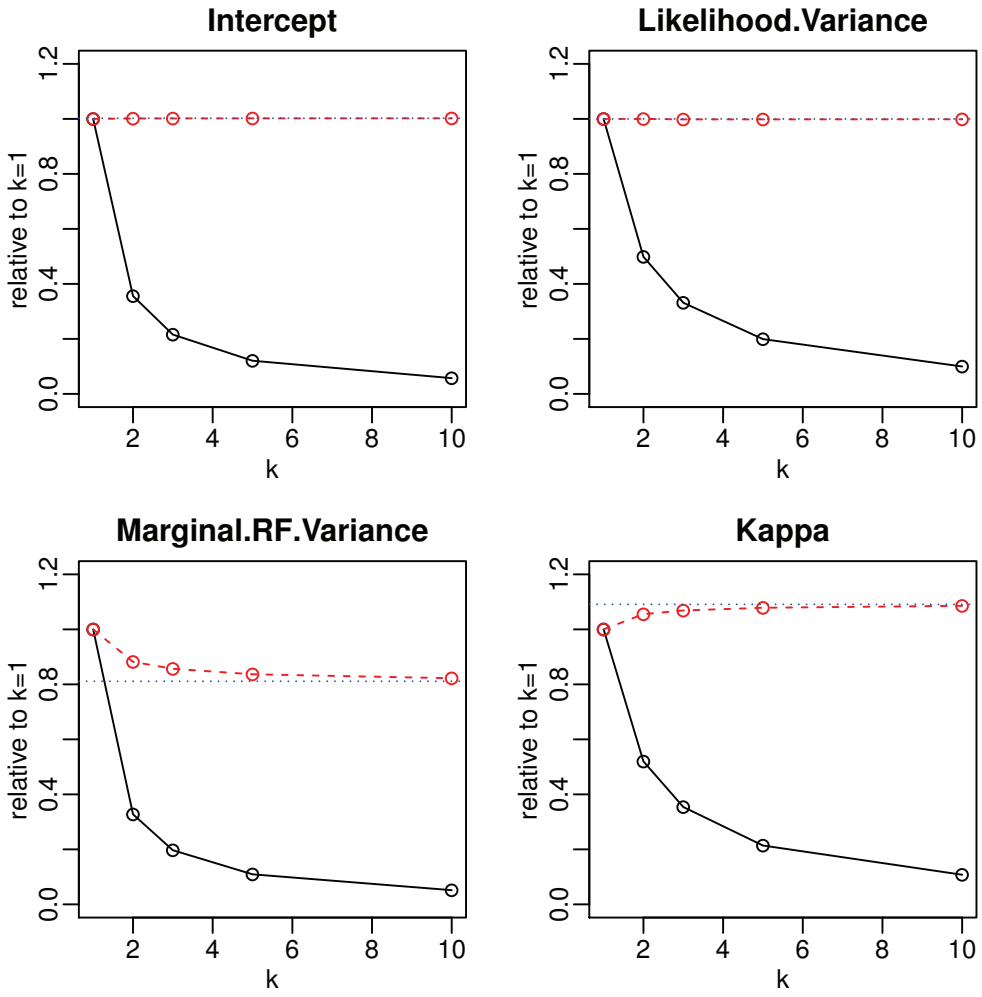

Figure 12.1: Likelihood estimate (green), posterior mean (red) and posterior variance (black), relatives to the value with $k = 1$.

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.

[Bivand and Rundel, 2013] Bivand, R. and Rundel, C. (2013). *rgeos: Interface to Geometry Engine - Open Source (GEOS)*. R package version 0.2-13.

[Bivand et al., 2012] Bivand, R., with contributions by Micah Altman, Anselin, L., ao, R. A., Berke, O., Bernat, A., Blanchet, G., Blankmeyer, E., Carvalho, M., Christensen, B., Chun, Y., Dormann, C., Dray, S., Halbersma, R., Krainski, E., Legendre, P., Lewin-Koh, N., Li, H., Ma, J., Millo, G., Mueller, W., Ono, H., Peres-Neto, P., Piras, G., Reder, M., Tiefelsdorf, M., and Yu., D. (2012). *spdep: Spatial dependence: weighting schemes, statistics and models*. R package version 0.5-55.

[Bivand et al., 2008] Bivand, R. S., Pebesma, E. J., and Gomez-Rubio, V. (2008). *Applied spatial data analysis with R*. Springer, NY.

[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.-l. (2010). Geostatistical inference under preferential sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 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.

[Illian et al., 2012] Illian, J. B., Srbye, S. H., and Rue, H. (2012). A toolbox for fitting complex spatial point process models using integrated nested laplace approximation (inla). *Annals of Applied Statistics*, 6(4):1499–1530.

[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 and Rue, 2013] Lindgren, F. and Rue, H. (2013). Bayesian spatial and spatio-temporal modelling with r-inla. *Journal of Statistical Software*.

[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 (with discussion). *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*.

[Pebesma and Bivand, 2005] Pebesma, E. J. and Bivand, R. S. (2005). Classes and methods for spatial data in R. *R News*, 5(2):9–13.

[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.

[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.

[Simpson et al., 2011] Simpson, D. P., Illian, J. B., Lindren, F., Srbye, S. H., and Rue, H. (2011). Going off grid: Computationally efficient inference for log-gaussian cox processes. *submitted*.

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

[Turner, 2014] Turner, R. (2014). *deldir: Delaunay Triangulation and Dirichlet (Voronoi) Tessellation.* R package version 0.1-5.