The R-INLA tutorial on SPDE models

Warning: work in progress...

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

Elias T. Krainski, Finn Lindgren, Daniel Simpson and Håvard Rue

August 26, 2015

Abstract

In this tutorial we present how to fit models to spatial point-referenced data, the so-called geostatistical model, using INLA and SPDE. After a fast introduction to such models, we start with the mesh building (necessary to fit the models) and show the application to a toy example with details. In the following examples we presents some funcionalities to fit more complex models, including non-Gaussian models and those with two likelihoods, such as semi-continuous, missaligned and preferential sampling, where we joingly model the point pattern process. Also, we introduce non-stationary SPDE models, and show hot wo fit some space-time models, including point process, and lowering time dimention. We also briefly present the data-cloning approach.

0.1 Acknowledgments

To Helen Sofaer for valuable English review in Chapters 1 and 2.

0.2 Updates

```
2015 Aug 26: fix the spatio temporal point process example
2015 Jul 17: small fixes and interpolate lin. pred. samples (Non-Gaus.)
2015 May 25: include space-time coregionalization model and tiny fix
2015 May 7: English review in Chapters 1 and 2 (thanks to HS) and
  space-time point process, space time lowering dimension and survival
2014 March 15: log-Cox example: fix weights, add covariate example
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

A point-referenced dataset is made up of any data measured at known locations. These locations may be in any coordinate reference system, most often the longitude and latitude coordinates. Point-referenced data are very common in many areas of science. This type of data appears in mining, climate modeling, ecology, agriculture and other areas. If we want to model the data while incorporating the information about where the data are from, we need a model for point referenced data.

It is possible to build a regression model using each coordinate as a covariate. But in some cases it is necessary to include a very complicated function based on the coordinates to get an adequate description of the mean. For example, we may need complex non-linear function or a non-parametric function. This type of model only incorporates a trend on the mean based on the coordinates. Also, this type of model is a fixed effect model.

Instead, it is more natural for a model to measure the first law of geography 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 that incorporates the property that an observation is more correlated with an observation collected at a neighboring location than with another that is collected at more distant location. One option to model this dependence is to use a spatially-structured random effect. This type of model incorporates spatial dependency, rather than simply the spatial trend. However, it is possible to include both terms in a model. Spatial dependency can be accounted for within more general models using spatially structured random effects.

In spatial statistics, different models are used to incorporate spatial dependency depending on whether the locations are areas (states, cities, etc.) or whether the locations are points. In the latter case, the locations can be fixed or random. Models of point-referenced data that include a spatially-structured random effect are commonly called geostatistical models. Geostatistics is the specific area of spatial statistics that these models. See [Cressie, 1993] for a good introduction to spatial statistics.

1.1 The Gaussian random field

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

Suppose 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 commonlly assumed that $x(s)$ has a multivariate Gaussian distribution. Also, if we assume that $X(s)$ is continuous over space, we have a continuously indexed Gaussian field (GF). This implies that it is possible to collect these data at any location within the study region. To complete the specification of the distribution of $x(s)$, is necessary to define its mean and covariance.

A very simple option is to define a correlation function based only on euclidean distance between locations. This assumes that if we have two pairs of points separated by the same distance h, both pairs have same correlation. It 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, which 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 the second order. The Matérn covariance function is $\sigma_x Cor(X(s_i), X(s_j))$, where σ_x is the marginal variance of the process.

If we have a realization $x(s)$ at n locations, we can define its joint covariance matrix. Each entry of this joint covariance matrix Σ is $\Sigma_{i,j} = \sigma_x Cor_M(X(s_i), X(s_j))$. It is common to assume that $X(x)$ has a zero mean. We have now completely defined a multivariate distribution for $x(s)$.

Now, suppose that we have data y_i observed at locations s_i, $i = 1, ..., n$. If an underlying GF generated these data, we can fit the parameters of this process, based on the identity $y(s_i) = x(s_i)$, where $y(s_i)$ is a realization of the GF. In this case, the likelihood function is 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 underlying GF but cannot directly observe it and instead observe data with measurement error, i.e., $y(s_i) = x(s_i) + e_i$. It is common to assume that e_i is independent of e_j for all $i \neq j$ and $e_i \sim N(0, \sigma_e)$. This additional parameter, σ_e, measures the noise, called the nugget effect. In this case the covariance of the marginal distribution of $y(s)$ is $\sigma_e^2 I + \Sigma$. This model is a short extension of the basic GF model, and in this case, we have one additional parameter to estimate. To look more about this model see [Diggle and Ribeiro Jr, 2007].

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

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

where $\mu_i = h(F_i^T\beta + x_i)$, F is a matrix of covariates, x is the random effect, θ are parameters of the random effect, 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, which is assumed for y_i. To write the GF with a nugget effect, we replace β_0 with $F_i^T\beta$, assume a Gaussian distribution for y_i, with variance σ_e^2 and x as a GF.

We can extend this basic hierarchical model in many ways, and we return to some extensions in later sections. If we know the properties of the GF, we can study all the practical models that contain, or are based on, this random effect.

It is worth mentioning that the data, or the random effect, on a finite number of n points where we have observed data are considered a realization of a multivariate Gaussian distribution. Therefore, to evaluate the likelihood function, or the random effect distribution, we need to compute the multivariate Gaussian density. So, we have, in the log scale, the expression

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

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 this is a 'big n problem'.

An alternative used in some software for geostatistical analysis is to use the empirical variogram to fit the parameters of the correlation function. This option does not use any likelihood for the data or uses a multivariate Gaussian distribution for the spatially structured random effect. A good description of these techniques is made in [Cressie, 1993].

However, it is adequate to assume a likelihood for the data and a GF for the spatial dependence as in the model based approach proposed for geostatistics, [Diggle and Ribeiro Jr, 2007]. At times we need to use the multivariate Gaussian distribution for the random effect. But, if the dimension of the GF is big, it becomes impractical for model-based inference to directy use the covariance as defined previously.

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

1.2 Simulation of a data set

First we look at the model with a parametrization used in a commonly used software program for analyzing point-referenced data in **R**, the **geoR** package, [Ribeiro Jr and Diggle, 2001]. In this section we show how we simulate a dataset with this parametrization.

Recall that one realization of a GF is just one realization of a multivariate Gaussian distribution with an appropriate covariance matrix. To specify this matrix, we need a set of locations and the matrix of the distances between each point and all others. Based on this $n \times n$ distance matrix, 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 a higher density in the bottom left corner than in the top right corner. The **R** code to do this 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 to 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 values to these parameters using

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

We first consider y as mean β_0 and with covariance $\sigma_e^2 I + \Sigma$. We get the covariance using

```
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, them $L^T L = \Sigma$. In **R** we use:

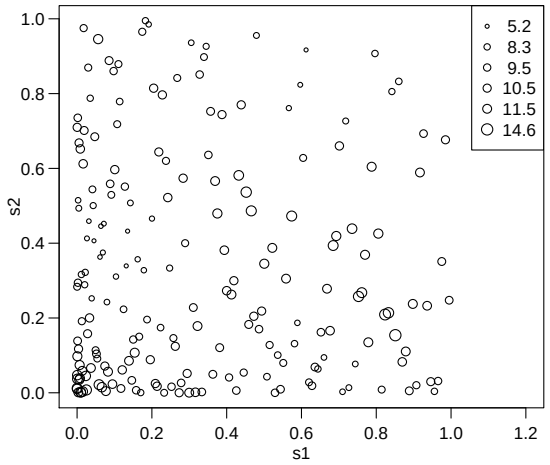

Figure 1.1: Visualization of the simulated data

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

We show these simulated data in a graph of the locations where the size of the points is proportional to the simulated values in Figure 1.1, produced with the 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 perform maximum likelihood estimation for the parameters of the model used to simulate the data in the previous section. It is useful to use the partial derivatives with respect to each parameter to compute the Fisher information matrix. We can then use it in the Newton-Raphson algorithm or in the Fisher scoring algorithm. It is even better if there exists closed form expressions for the maximum likelihood estimators to some of the parameters.

Under a Gaussian likelihood we have that the mean is orthogonal to the variance. This implies we can use closed form expressions to estimate the mean, or (in general) the regression parameters. Additionally, we can re-write the covariance and derive a closed form expression to estimate σ_x^2. Then an optimization algorithm can be used for the remaining parameters.

Deriving with respect to β we can compute its maximum likelihood estimator solving

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

where $\boldsymbol{W}$ is the inverse of the covariance matrix. Writing the variance parameter of the noise, σ_e^2, in terms of the relative one, $r = \sigma_e^2/\sigma_x^2$, we have

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

where $\boldsymbol{C}$ is the correlation matrix. This implies that $\boldsymbol{W} = (r\boldsymbol{I} + \boldsymbol{C})^{-1}$. Deriving with respect to σ_x^2 we can find

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

Now, we can work with a concentrated log-likelihood, that is a function of the relative variance of the noise 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 quasi-Newton methods to find the maximum likelihood estimates of these three parameters.

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

We use the negative of the logarithm of the likelihood function because, by default, the algorithm finds 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 a test, we calculate 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 the 'L-BFGS-B' method available through the `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 get all the estimated parameters by evaluating our concentrated log-likelihood again to get all estimates

```
(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 via maximum likelihood is to show how it works. The **geoR** package includes functions to do simulations and also to perform likelihood estimation. However, the Matérn correlation function is parametrized differently in the **geoR** package, which uses $\phi = 1/\kappa$ for the scale parameter and it uses kappa for the smoothness parameter. Also σ_e^2 is called `tausq`.

The `grf()` function can be used to get samples of a geostatistical model with several options for the correlation function. 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)
```

We can use the `likfit()` function to perform the maximum likelihood estimation. With this function we obtain the maximum likelihood estimates for the smoothness parameter using

```
(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 of other parameters using

```
(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 the likelihood here has a similar value as the previous one, because the previously estimated value of ν is close to the fixed value here.

Chapter 2

The SPDE approach

In the literature there are some ideas to fit GF by an approximation of the GF to any GMRF. A very good way is to use an explicit link between the precision matrix from the solution to a stochastic partial differential equation (SPDE) and the Matérn Gaussian random fields, [Lindgren et al., 2011]. This approach provide an explicit link between GF and GMRF. It is common to use the the Finite Element Method (FEM) to estimate the solution to an SPDE. This is particularly convenient in the case where we have a set of irregular locations.

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 a 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 has

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

and $Var(x_{ij}|x_{-ij}) = 1/a$ for $|a| > 4$. In the representation using a 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 $\nu = 2$:

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

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, the precision matrix has values over a larger neighbourhood areas. Hovewer, it does not means that one can just averages over larger neighboorhod to make it smooth. However, it does not means averaging over larger neighboorhod to make it smoother. This is like when going from first order random walk to the second order one: to square the precision matrix.

If the spatial locations are on a irregular grid, it is necessary to use the second result in [Lindgren et al., 2011]. To extend the first result, the finite element method (FEM) is used 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 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:

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

Again, increasing ν denser the 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 done VERY CAREFULLY. It is similar to choosing the integration points on a numeric integration algorithm. Should the boints be regular? How many points are needed?

Additionally, we need to add, ALSO CAREFULLY, additional points around the boundary, the outer extension. This is necessary to avoid a boundary effect where we have a variance twice as larger at the border than within the domain [Lindgren, 2012]. For more about it please see [Lindgren and Rue, 2013].

2.2.1 Getting started

For a 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:

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

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

We need some reference about the study region, which can be provided by the location points or just a domain. The location, supplied on the `loc` argument, are used as initial triangulation nodes. A single polygon can be supplied to determine the domain extent on the `loc.domain` argument. If we supply the point locations, or the domain is supplied using the `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, where each element of this list is of `inla.mesh.segment()` class. So, one of these three options is mandatory.

The other 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 numeric on the **SAME SCALE UNIT** as the coordinates.

The another arguments are used to specify additional conditions. The `offset` is a numeric, or length two vector. If negative it is interpreted as a factor relative to the approximate data diameter. If positive it 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 needs to have triangles as regular as possible in size and shape. To help this requirement in addition to `max.edge`, we have the `min.angle` argument, which 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. Values up to 21 guarantee the convergence of the algorithm.

To further control the shape of the triangles, we also have the `cutoff` argument, which is the minimum allowed distance between points. It means that points at a closer distance than the supplied value are replaced by a single vertex. So, it avoids small triangles and must be a positive number, and is critical when we have some very close points, either for point locations or on the domain boundary.

To understand how this function works, we apply it while varying some arguments to the 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 and 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)
```

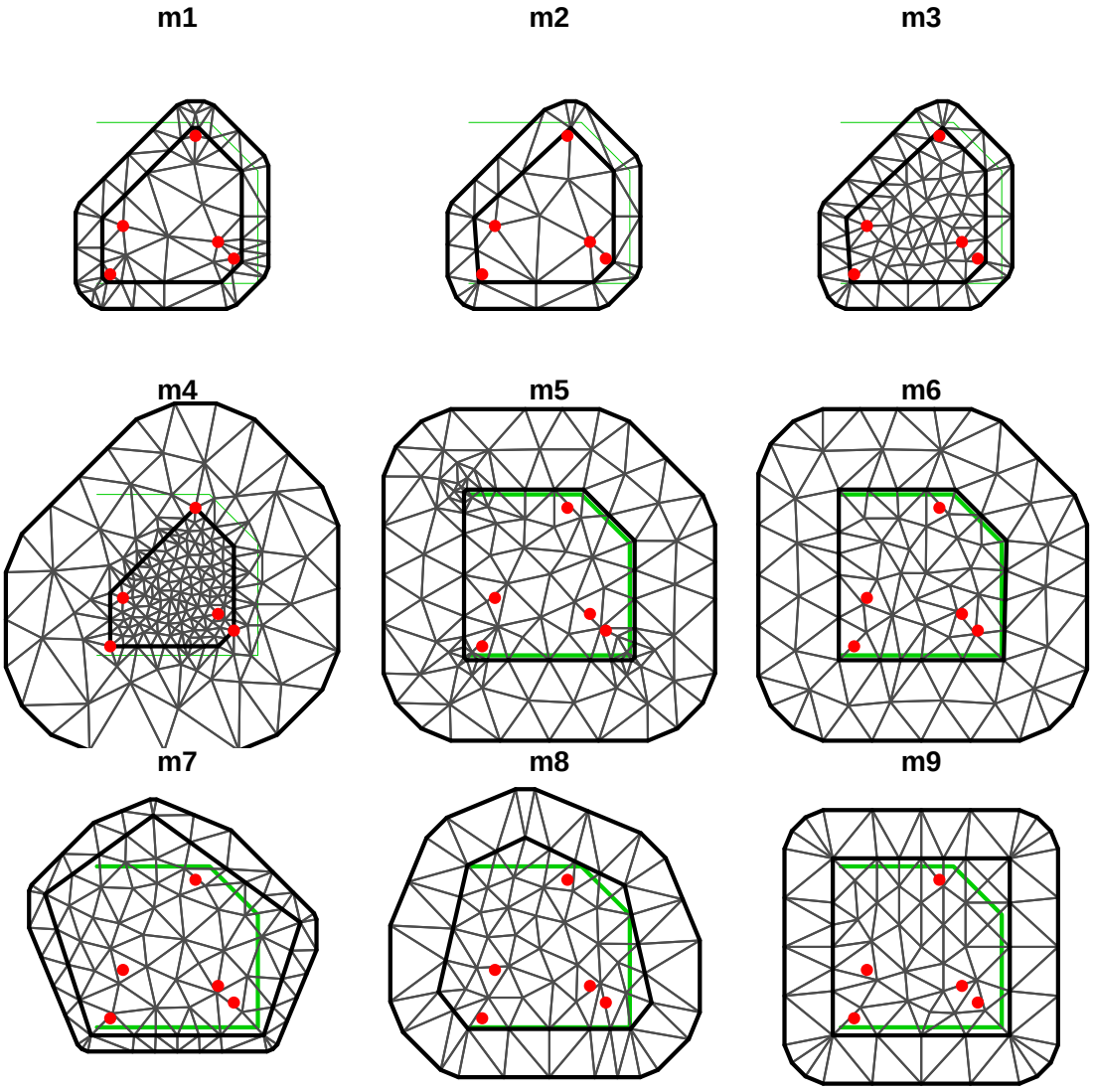

Figure 2.1: Triangulation with different restrictions.

```
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))
```

We visualize these meshes in Figure 2.1, produced 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)
  }
```

The `m1` mesh has two main problems: 1) some triangles with small inner angles, 2) some large triangles in the inner domain. In the `m2` mesh, we relax the restriction on the

locations, because points with distance less than the cutoff are considered a single vertex. This avoids some of the triangles (at bottom right side) with small angles on the previous mesh. So the **cutoff is a VERY GOOD idea**! Each inner triangle in the `m3` mesh on the top right had edge length less than 0.1 and this mesh looks better than the two previous ones.

The `m4` was made without first building a convex hull extension around the points. It has just the second outer bondary. In this case, the length of inner triangles does not work (first value on `max.edge` argument) and we have triangles with edge lengths up to 0.5. The shape of the triangles looks good, except for these ones with vertices including the two points at the bottom right side.

The `m5` mesh was 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 due the fact that is was built without specifying a `cutoff`. Also, we have a (relatively) small first extension and a (relatively) large second one. On the `m6` mesh we have added the cutoff and got a better mesh than the previous one.

In the last tree meshes we change the initial number of extension points. It can be useful to change in some situations to get convergence. Here we show the shape of the mesh that we got with, for example, `n=5`, in the `m7` mesh. This number produces a mesh that seems inadequate for this domain because we have a non uniform exension behind the border. The `m9` mesh has very bad triangles shapes.

The object returned by the `inla.mesh.2d()` function is of 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  79 195 117  88  87  69  72
```

The 'graph' element represents the CRDT obtained. In addition, the 'graph' element contains the matrix that represents the graph of the 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 in the 'idx' element

```
 m1$idx$loc
```

```
[1] 24 25 26 27 28
```

2.2.2 Non-convex hull meshes

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

We can also 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 and set some constraint. We can control the shape of the boundary including its convexity, concavity and resolution. Here, we make some boundaries and build a mesh with each one to better understand 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 in 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 built with a boundary that we got using default arguments in the `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 a larger convex value, like the one used to generate `m11`, we get a larger boundary. It's because all circles with a centre on each point and a radius less than the convex value are inside the boundary. When we choose a larger concave value, as in the boundary used for 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 get 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 triangulation options to make comparisons in section 3.5. The first mesh forces the location points to be vertices of the mesh.

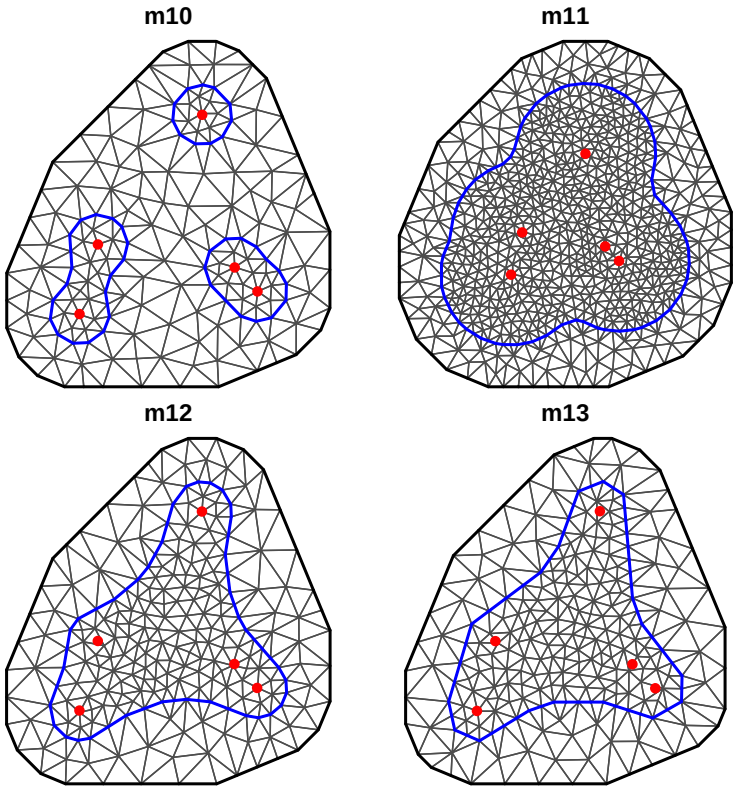

Figure 2.2: Non-convex meshes with different boundaries.

```
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 a cutoff greather than zero to avoid small triangles in regions where we have dense 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 built to have approximately the same number of vertices as the previous ones

```
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 in each one of these meshes is

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

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

These six meshes are shown in 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='')
```

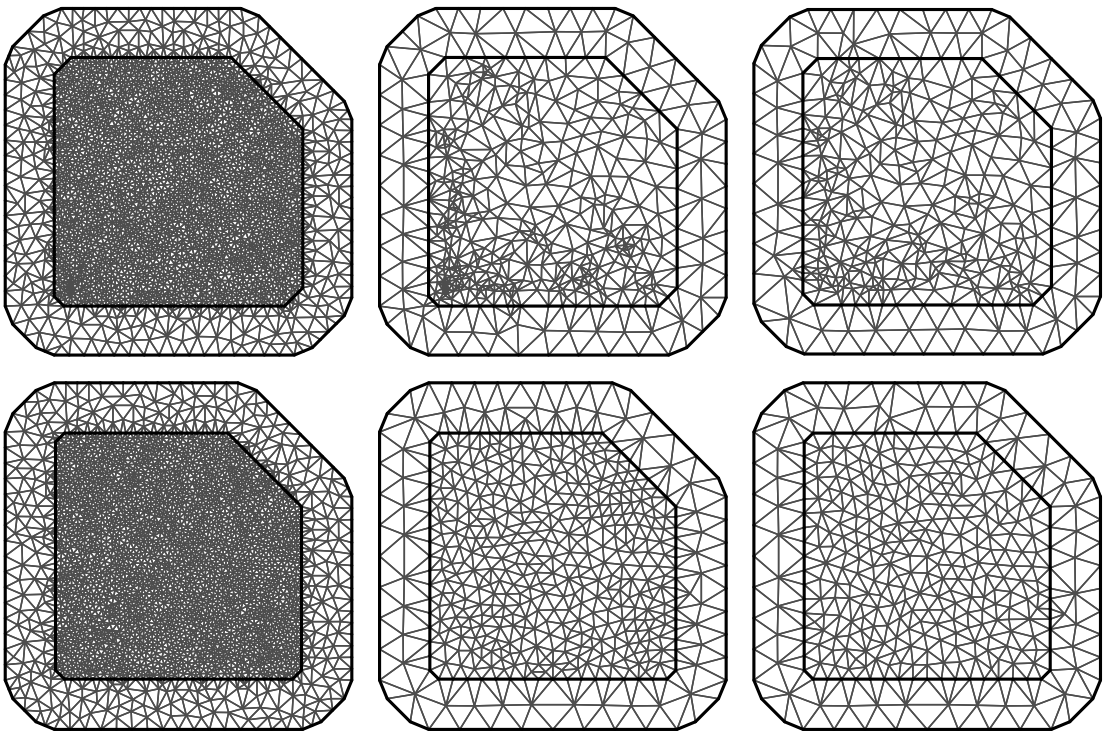

Figure 2.3: Six triangulation options for the toy example.

2.2.4 Meshes for Paraná state

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

We have the daily rainfall data

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

```
[1] 616 368
```

```
PRprec[1:2, 1:10]
```

```
  Longitude Latitude Altitude d0101 d0102 d0103 d0104 d0105 d0106 d0107
1  -50.8744 -22.8511      365     0     0     0     0     0     0   2.5
3  -50.7711 -22.9597      344     0     1     0     0     0     0   6.0
```

that consists of the daily rainfall data from 616 stations for each day of the 2011 year. The coordinates (two first colums) are on the latlong projection.

Also, we have the Paraná state polygon with

```
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 by building a non-convex domain with

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

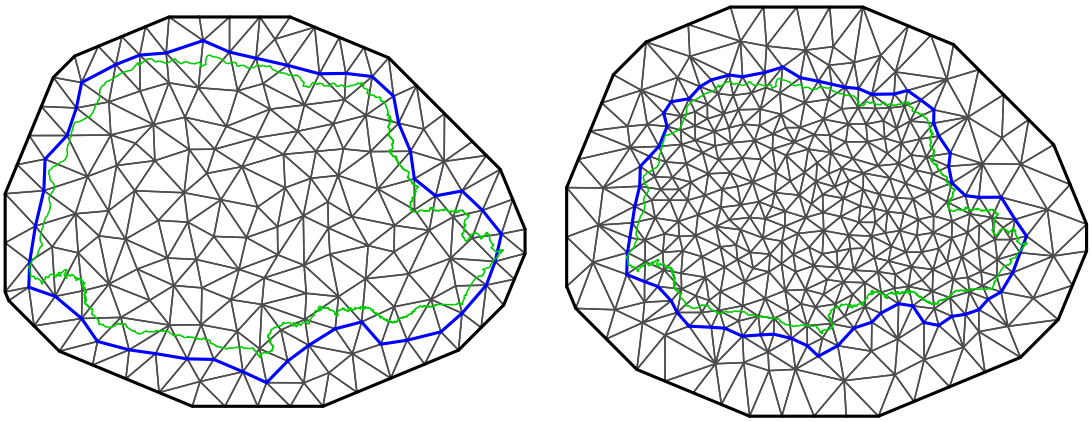

Figure 2.4: Mesh for Paraná state

with this defined domain 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 both meshes on the Figure 2.4 with commands below

```
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 a spatial object is made using object classes in the **sp** package, see [Pebesma and Bivand, 2005] and [Bivand et al., 2008]. To show an aplication in this case, we use the North Carolina map, in package **spdep**, [Bivand et al., 2012].

```
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 by 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 instead, but the we do need to set a permission

```
gpclibPermit()
```

before working with this library

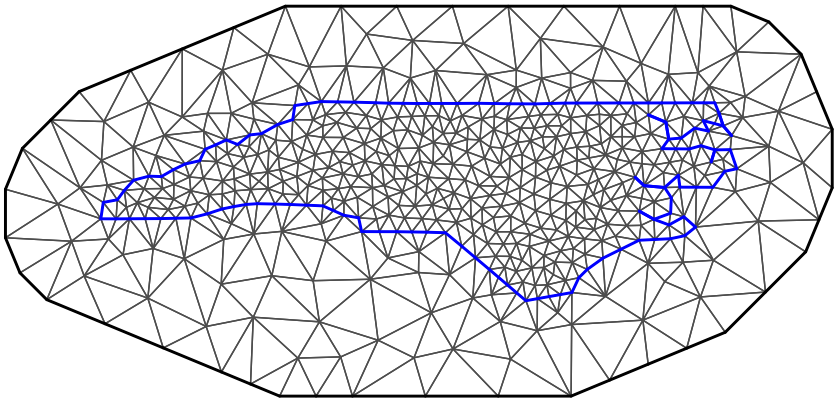

Figure 2.5: Mesh constructed using the North Carolina map

```
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

```
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 Mesh with holes and physical boundaries

Sometimes we need to deal with physical boundaries. It can be when there is a hole inside the domain or when the domain shape is not convex. An example of application is when modelling fish and we have to consider the inland as a physical barrier and, sometimes, islands inside the domain.

The polygons in the Figure 2.6 were created with the following commands:

```
pl1 <- Polygon(cbind(c(0,15,15,0,0), c(5,0,20,20,5)), hole=FALSE)
h1 <- Polygon(cbind(c(5,7,7,5,5), c(7,7,15,15,7)), hole=TRUE)
pl2 <- Polygon(cbind(c(15,20,20,30,30,15,15), c(10,10,0,0,20,20,10)), hole=FALSE)
sp <- SpatialPolygons(list(Polygons(list(pl1, h1), '0'), Polygons(list(pl2), '1')))
par(mar=c(0,0,0,0)); plot(sp) ### to visualize it
text(c(13, 17, 23), c(3, 12, 3), LETTERS[1:3], cex=3)
```

We have two neighbour regions, one with a hole and one with no convex shape. Suppose that it is necessary to avoid correlation between near regions separated by land. For example, suppose that we want to make sure that the correlation between A and C is smaller than between A and B or between B and C.

In this example we do not want additional points outer the domain. To have it, we have to supply a length one value for `max.edge`. The following code is to prepare the boundary and built the mesh.

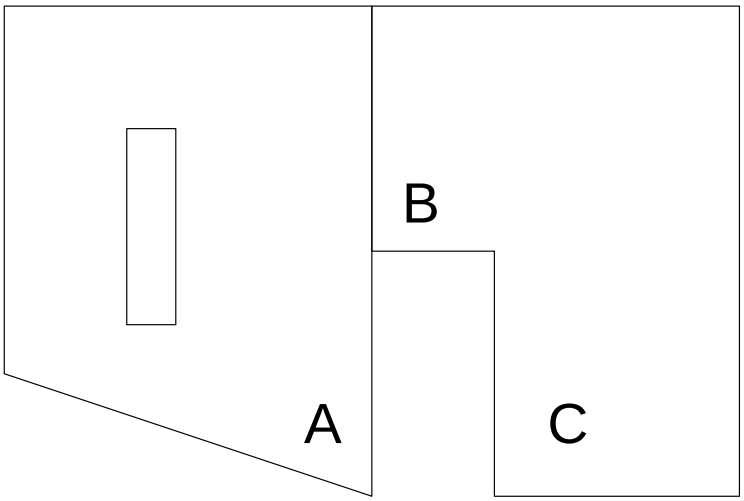

Figure 2.6: Region with a hole and non convex domain.

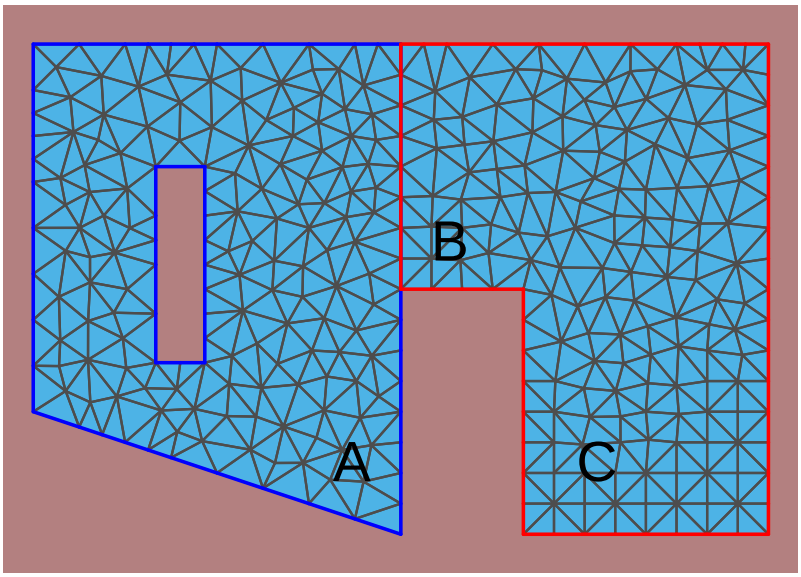

Figure 2.7: Triangulation with hole and non convex shaped region

```
bound <- inla.sp2segment(sp)
mesh <- inla.mesh.2d(boundary=bound, max.edge=2)
```

The mesh is displayed in the Figure 2.7 using the commands below:

```
par(mar=c(0,0,0,0), bg=rgb(.7,.5,.5))
plot(sp, col=rgb(.3,.7,.9))
plot(mesh, add=TRUE, lwd=2)
text(c(13, 17, 23), c(3, 12, 3), LETTERS[1:3], cex=3)
```

Notice that when building the SPDE model, the neighborhood structure of the mesh is taken into account. So, it is easier to reach B from A than C on the related graph.

2.4 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 dimension 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 projector matrix $\boldsymbol{A}$ to project the process at the mesh vertices to the locations response. The projector matrix can be builded with the `inla.spde.make.A` function. Considering that each mesh vertex 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 mesh number five we have

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

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

```
dim(A5)
```

```
[1] 200 490
```

We have exactly three non-zero elements on each line

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

```
  3
200
```

because each point location is inside one of the triangles. These three non-zero elements on each line sums to one

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

```
  1
200
```

because multiplication with any vector of 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
```

These columns correspond the triangles without any point locations inside. These columns can be droped and the stack functionality (section 3.1) automatically handles this situation.

When we have a mesh where every pont location is on a mesh vertex, each line on the projector matrix has exacty one nonzero element. This is the case for the `m1`

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

```
  1
200
```

and all these elements are equal to one

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

```
  1
200
```

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

2.5 GRF sampling through SPDE

The SPDE approach builds the model on the mesh vertices and we need to simulate values at the set of location points. We start by defining the spde model

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

and build the associated precision matrix.

This can be done with the `inla.spde2.precision()` function. To use this function, we have to provide the SPDE model and the parameters θ for 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 write a more general function to generate one or more samples, by defining a function that needs 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, mesh,
    verbose = FALSE, seed, return.attributes = FALSE)
{
    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$Q, seed = ifelse(missing(seed),
        0, seed), constr = attributes$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 million of locations in a matter of seconds

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

Some summary

```
summary(x1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.1790  0.1325  0.8425  0.7754  1.5030  2.9680
```

```
table(x1>0)
```

```
 FALSE   TRUE
186421 813579
```

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
1.943298 1.410463 3.353761

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

[1] 482

 dim(attr(x1, 'Q'))

[1] 482 482

 dim(attr(x1, 'A'))

[1] 1000000     482
```

This function is used in some of next Chapters.

2.6 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 $r = \frac{\sigma_\epsilon^2}{\sigma_x^2}$ 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 a relative variance, instead of the noise 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 a quasi-Newton optimization algorithim to find the maximum likelihood estimate of each the two parameters in the concentrated likelihood. These parameters are the ones in the $\boldsymbol{W}$ matrix and are parametrized as log(r) and log(κ).

To evaluate the likelihood function efficiently we need to compute the determinant $|\boldsymbol{W}|$ and $\boldsymbol{W}$ itself. Both computations needs care. We have that $\boldsymbol{W} = (r\boldsymbol{I} + Q(\tau^{'}, \kappa))$.

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 $\log(\sigma_y^2)$ as a 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 below recieves θ, $\log(\sigma_y^2)$ and β (regression parameters) and returns 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 from the `geoR` package

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

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, n.iid.group = 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 2.4 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 9.515322 0.732109   7.977745 9.530563    10.9655 9.555449 4.337963e-10
```

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

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

```
                                                mean        sd 0.025quant
Precision for the Gaussian observations 2.740861 0.4294079   1.982832
                                        0.5quant 0.975quant    mode
Precision for the Gaussian observations 2.712378   3.664665 2.65995
```

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] 0.6095435
```

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

```
[1] 0.5229415 0.6070405 0.7087953
```

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

```
                low      high
level:0.95 0.5190218 0.7040731
```

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

```
[1] 0.004652622 0.964102733
```

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] 7.898451
```

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

```
[1] 3.872936
```

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

```
[1] 0.3773445
```

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  0.1683 0.8119    -1.4021   0.1480     1.8621  0.1143   0
Apredictor.202  3.0480 0.9610     1.1764   3.0367     4.9886  3.0168   0
Apredictor.203 -2.7852 1.2086    -5.1885  -2.7782    -0.4225 -2.7652   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 1.153087 4.962877
```

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]  0.1677608  3.0473044 -2.7852276
```

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

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

```
[1]  0.1677608  3.0473044 -2.7852276
```

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] 0.8947857 1.1135312 1.3737750
```

and we have a little difference.

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

```
[1] 0.5897517 0.6849483 0.9346851
```

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 0.989      4.794    6.726      8.687  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 11.33272 13.79105
```

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]  9.683083 12.562626  6.730094
```

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.
 0.5925  1.0370  1.4300  2.5170  2.6080 11.0800
```

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

```
[1] 1.529301 1.669212 1.850958
```

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
    0.22797728      8.45248699      0.08040404      8.76086831
```

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.

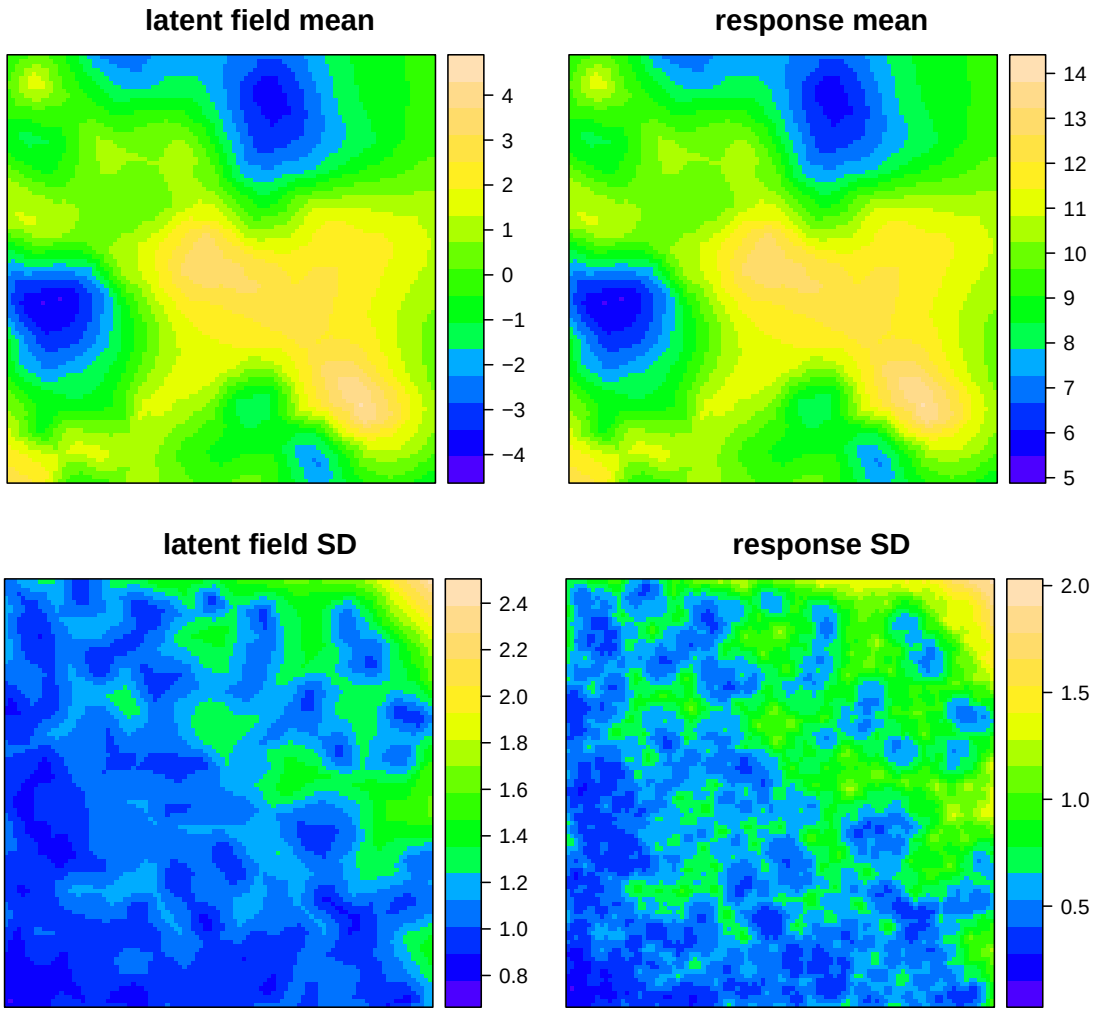

Figure 3.1: The mean and standard deviation of the random field (top left and bottom left) and the mean and standard variation of the response (top right and bottom right)

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
       12.37         1.70         0.94        11.77         1.36
Running inla
        1.12
```

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

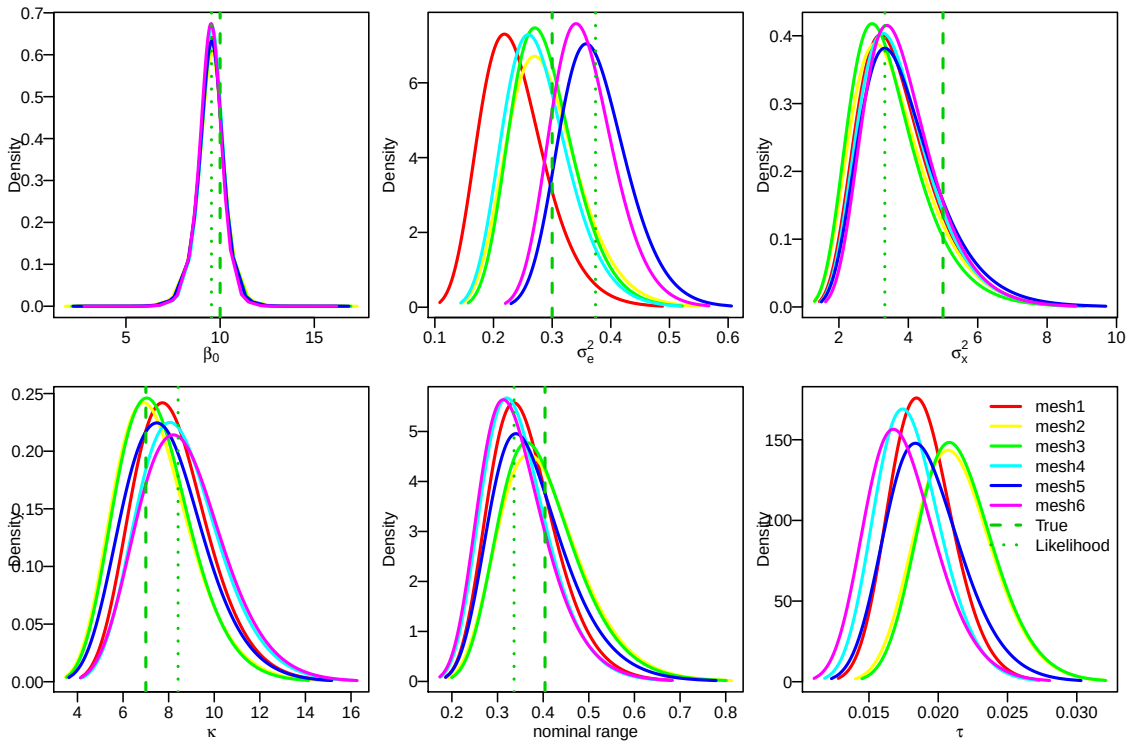

Figure 3.2: Marginal posterior distribution for β_0 (top left), σ_e^2 (top mid), σ_x^2 (top right), κ (bottom left), nominal range (bottom mid) and τ (bottom right).

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,

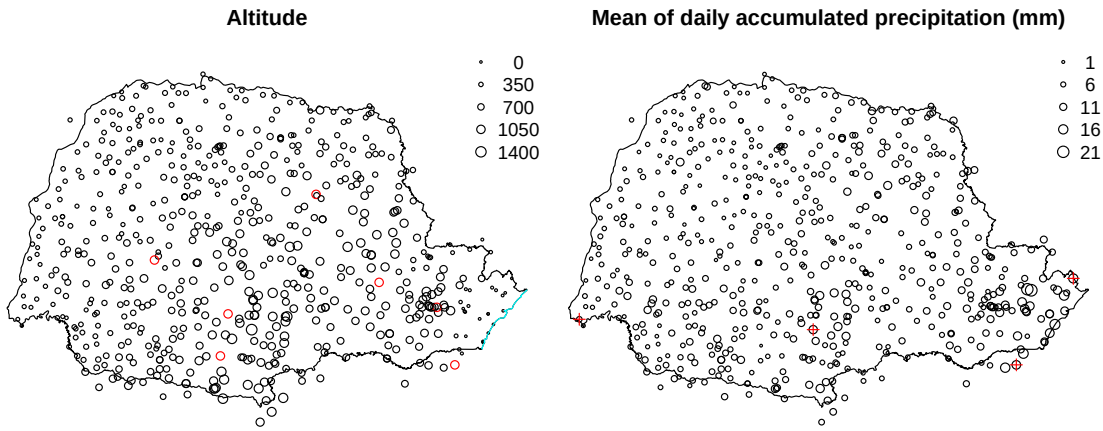

Figure 4.1: Locations of Paraná stations, altitude and average of daily accumulated precipitation (mm) on January.

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

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

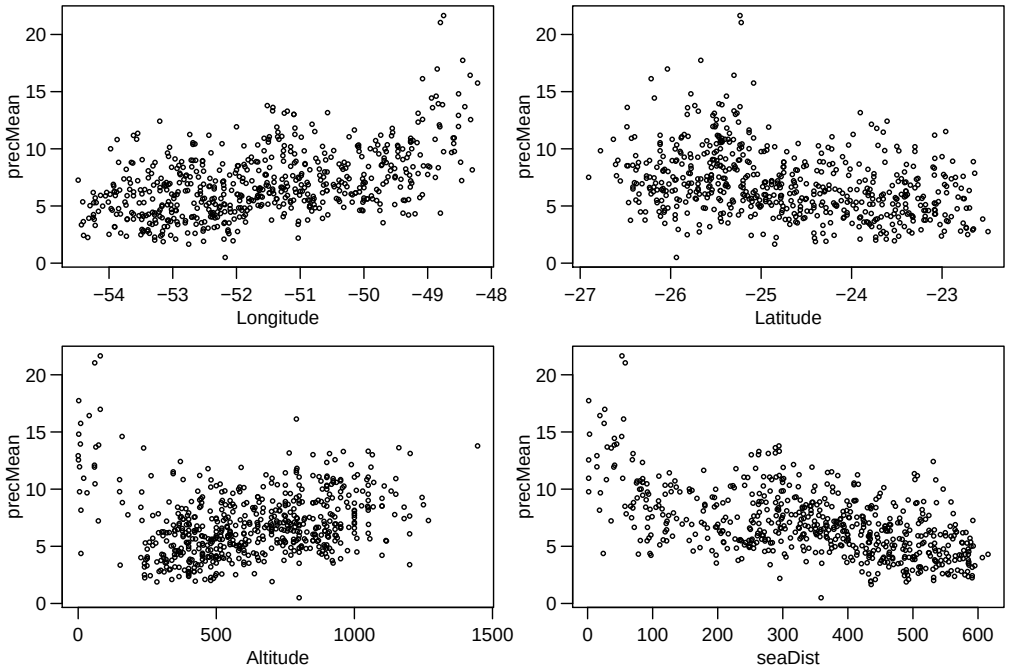

Figure 4.2: Dispersion plots of average of daily accumulated precipitation mean by Longitude (top left), Latitude (top right), Altitude (bottom left) and distance to sea (bottom right).

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

$$log(\mu_i) = \alpha + \sum_{k=1}^{m} w_k \tilde{F}_i + x_i$$

were F is the distance to sea or the longitude and $\tilde{F}_i$ is a discretized version of F using a set of knots (choose on range of the covariate values with the `inla.group()` function) and w_k are a set of serially correlated 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] 2568.442 2566.289
```

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 1.942816 0.05249861   1.836541 1.942832   2.048692 1.942671
                   kld
Intercept 2.037912e-10
```

To dispersion parameter of the gamma likelihood we have

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

```
                                                    mean        sd
Precision parameter for the Gamma observations 12.57355 0.8834257
                                                0.025quant 0.5quant
Precision parameter for the Gamma observations   10.90862 12.55004
                                                0.975quant     mode
Precision parameter for the Gamma observations   14.38037 12.51045
```

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
Precision for gSeaDist 8707.347 5718.572    2435.63 7221.915   23713.61
                           mode
Precision for gSeaDist 5131.485
```

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

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

```
               mean        sd 0.025quant 0.5quant 0.975quant     mode
Theta2 for i 1.36601 0.3133459  0.7379381 1.371543   1.967842 1.388107
```

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] 0.06143308
```

And to the practical range of the process we have

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

```
[1] 0.7573185
```

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')
```

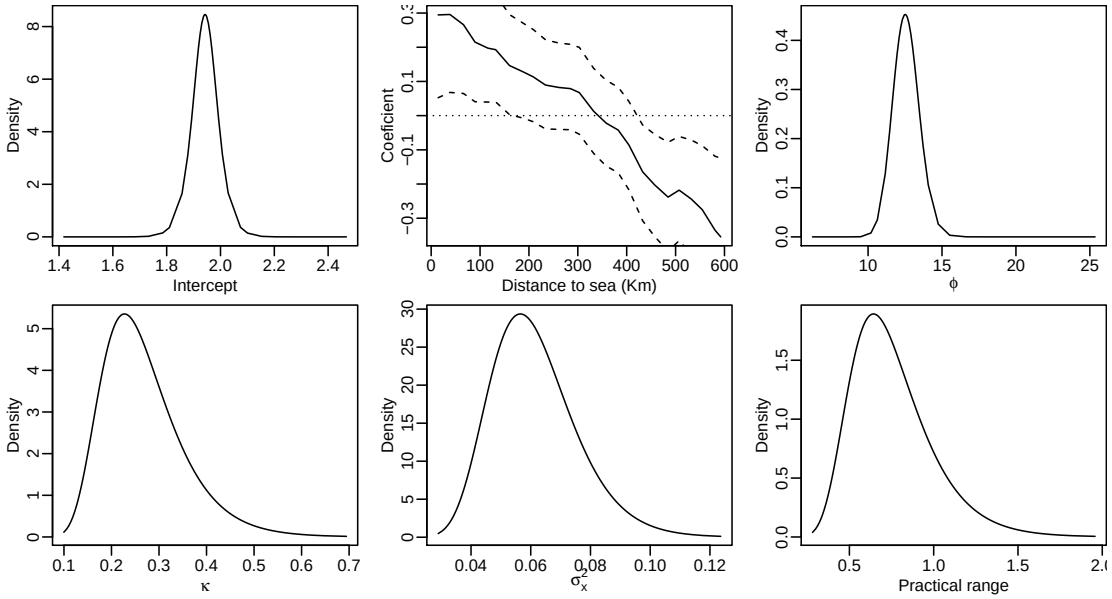

Figure 4.3: Posterior marginal distribution of β_0 (top left), the posterior mean (continuous line) and 95% credibility interval (dashed lines) to effect of distance to sea (top mid), posterior marginal distribution to ϕ (top right), posterior marginal distribution of κ (bottom left), posterior marginal 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] 2721.492 2566.289
```

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. At the end of this section we show a cheap way for a specific case.

4.5.1 By computation of the posterior distributions

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, link=1), 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.

4.5.2 By sampling and interpolating the linear predictor

The approach in this section does not always makes sense! However, in some SPECIFIC cases it makes sense to interpolate the linear predictor. This cases happens when whe only have the spatial effect and enviromental covariates which are smooth over space. The advantage is that is computationally cheap than to compute the full posterior marginals like in the previous section.

In our example the idea is to compute the effect of the distance to sea at the mesh nodes. Them, compute the linear predictor at the mesh nodes. To predict the response

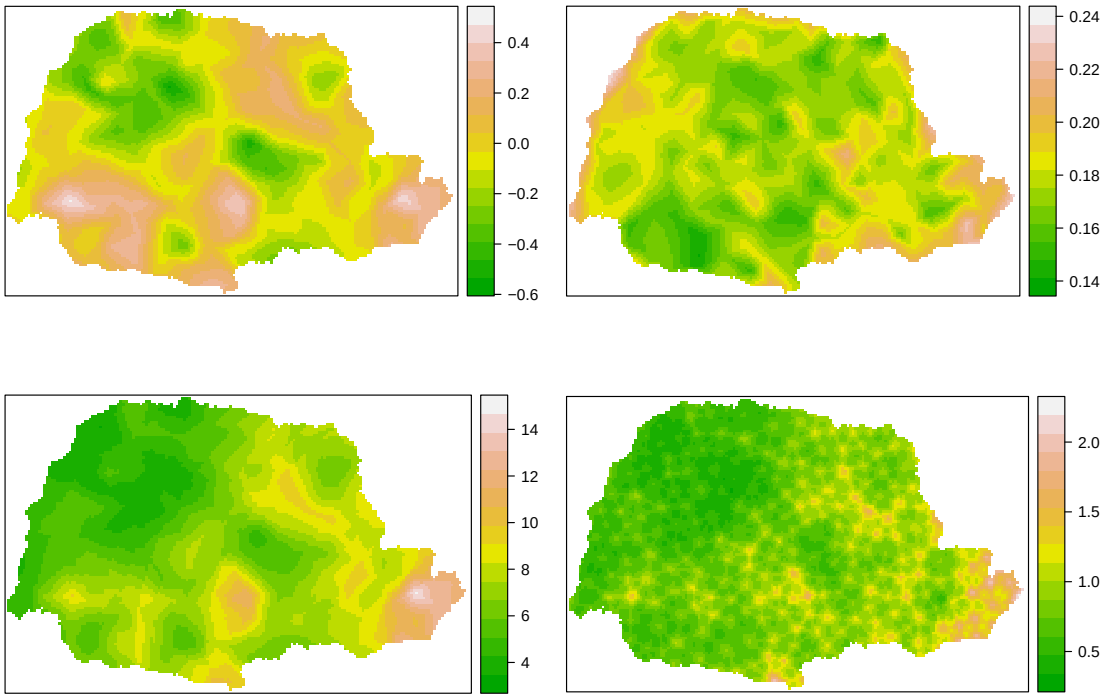

Figure 4.4: Posterior mean and standard deviation of the random field (top left and right respectively). Posterior mean and standard deviation of the response (top left and right respectively).

on gridn one can interpolate it. In our example the link is log and to aproximate also the standard error on the response, we use the sampling approach.

It consists in geting samples from the linear predictor at the mesh nodes. Them, interpolate it to grid and compute the expected value, in the response scale by applying the inverse link. When we do it for each sample, we can have any functional, for example, the mean and standard error.

The first step is to have the enviromental covariate at the mesh node

```
dsmesh <- apply(spDists(PRborder[1034:1078,],
                        prmesh2$loc[,1:2], longlat=TRUE), 2, min)
imesh0 <- apply(abs(outer(ug.seadist, dsmesh, '-')), 2, which.min)
mesh.seadist <- ug.seadist[imesh0]
```

Building the stack for predict into the mesh

```
stk.mesh <- inla.stack(data=list(y=NA), A=list(1,1),
                       effects=list(i=1:spde$n.spde,
                           data.frame(Intercept=1,
                                      gSeaDist=mesh.seadist)), tag='mesh')
stk.b <- inla.stack(stk.dat, stk.mesh)
```

Fitting the model again and ask for the configuration, needed for sampling from the model

```
rm.sead <- inla(f.sead, family='Gamma', data=inla.stack.data(stk.b),
                control.predictor=list(A=inla.stack.A(stk.b),
                    compute=TRUE, link=1), quantiles=NULL,
                control.results=list(return.marginals.random=FALSE,
```

```
                    return.marginals.predictor=FALSE),
                control.compute=list(config=TRUE))
```

Sampling from the model

```
sampl <- inla.posterior.sample(n=1000, result=rm.sead)
```

The first n elements of the latent field are the linear predictor for the observed data and the next m elements are for the location at the mesh nodes.

```
dim(pred.nodes <- exp(sapply(sampl, function(x)
      x$latent[nrow(PRprec) + 1:nrow(prmesh2$loc)])))
```

```
[1]  382 1000
```

Computing the mean and standard deviation over the samples and projecting it

```
sd.prd.s <- m.prd.s <- matrix(NA, nxy[1], nxy[2])
m.prd.s[xy.in] <- drop(Aprd %*% rowMeans(pred.nodes))
sd.prd.s[xy.in] <- drop(Aprd %*% apply(pred.nodes, 1, sd))
```

and we can see that it is similar to the computed analitically

```
cor(as.vector(m.prd.s), as.vector(m.prd), use='p')
```

```
[1] 0.9984864
```

```
cor(as.vector(sd.prd.s), as.vector(sd.prd), use='p')
```

```
[1] 0.9237277
```

Chapter 5

Survival analysis

In this chapter we shown how to fit a survival model using a continuous spatial random effect modeled throught the SPDE approach. We use the data presented in [Henderson et al., 2003]. The original code for the analysis in [Lindgren et al., 2011] is adapted here to use the stack functionality. In the section 5.1 we show how to fit a parametric survival model and in the section 5.2 we also show how to fit the semiparametric Cox proportional hazard model.

5.1 Parametric survival model

We load the Leukaemia survival data using

```
data(Leuk);    sapply(Leuk, summary)
```

```
        time   cens xcoord ycoord   age    sex    wbc     tpi district
Min.       1 0.0000 0.0000 0.0000 14.00 0.0000   0.00 -6.0900     1.00
1st Qu.   41 1.0000 0.2244 0.1792 49.00 0.0000   1.80 -2.7050     7.00
Median   185 1.0000 0.4642 0.2900 65.00 1.0000   7.90 -0.3700    16.00
Mean     533 0.8428 0.4044 0.3566 60.73 0.5244  38.59  0.3398    13.79
3rd Qu.  536 1.0000 0.6014 0.5259 74.00 1.0000  38.65  2.9300    20.00
Max.    4977 1.0000 0.7740 1.0000 92.00 1.0000 500.00  9.5500    24.00
```

The mesh is builded using the following code

```
loc <- cbind(Leuk$xcoord, Leuk$ycoord)
bnd1 <- inla.nonconvex.hull(loc, convex=0.05)
bnd2 <- inla.nonconvex.hull(loc, convex=0.25)
mesh <- inla.mesh.2d(loc, boundary=list(bnd1, bnd2),
      max.edge=c(0.05, 0.2), cutoff=0.005)
```

The projector matrix is obtained with

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

The SPDE model is created with

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

The model formula, including the intercept, covariates and the SPDE model is

```
formula <- inla.surv(time, cens) ~ 0 + a0 +
      sex + age + wbc + tpi +
      f(spatial, model=spde)
```

The trick for building the data stack is to include all the variables needed to the formula. So, for the response we have the `time` and the censoring pattern `cens`. The spatial stuff, the intercept and the covariates are included like in the other models.

```
stk <- inla.stack(data=list(time=Leuk$time, cens=Leuk$cens),
                  A=list(A, 1),
                  effect=list(
                      list(spatial=1:spde$n.spde),
                      data.frame(a0=1, Leuk[,-c(1:4)])))
```

Now, we just had to fit the model. In this example we use the 'weibull' likelihood. Hovewer, other parametric likelihoods can be used as well.

```
r <- inla(formula, family="weibull", data=inla.stack.data(stk),
          control.predictor=list(A=inla.stack.A(stk)))
```

The intercept and the covariate effects can be extracted with

```
round(r$summary.fix, 4)
```

```
      mean     sd 0.025quant 0.5quant 0.975quant    mode kld
a0  -5.7152 0.2582    -6.2165  -5.7182    -5.1956 -5.7236   0
sex  0.0720 0.0693    -0.0640   0.0720     0.2081  0.0720   0
age  0.0328 0.0022     0.0285   0.0328     0.0372  0.0328   0
wbc  0.0031 0.0005     0.0022   0.0031     0.0040  0.0031   0
tpi  0.0247 0.0099     0.0053   0.0247     0.0441  0.0247   0
```

and the hyperparameters with

```
round(r$summary.hy, 3)
```

```
                                mean    sd 0.025quant 0.5quant 0.975quant
alpha parameter for weibull    0.600 0.016      0.569    0.600      0.632
Theta1 for spatial            -2.441 0.539     -3.535   -2.426     -1.419
Theta2 for spatial             2.405 0.562      1.343    2.389      3.548
                                mode
alpha parameter for weibull    0.601
Theta1 for spatial            -2.379
Theta2 for spatial             2.338
```

We visualize the spatial effect into the map. The map of the districts is also available into the INLA package. First, we define a projection from the mesh into a grid

```
r0 <- diff(range(bbox(nwEngland)[1,]))/diff(range(bbox(nwEngland)[2,]))
prj <- inla.mesh.projector(mesh, xlim=bbox(nwEngland)[1,],
                           ylim=bbox(nwEngland)[2,],
                           dims=c(200*r0, 200))
```

then we interplate it and assign NA for grid points not inside the map

```
m.spat <- inla.mesh.project(prj, r$summary.ran$spatial$mean)
sd.spat <- inla.mesh.project(prj, r$summary.ran$spatial$sd)
ov <- over(SpatialPoints(prj$lattice$loc), nwEngland)
sd.spat[is.na(ov)] <- m.spat[is.na(ov)] <- NA
```

The posterior mean and standard deviation are in Figure 5.1. As a result, the spatial effect has continuous variation along the region, rather than constant inside each district.

```
par(mfrow=c(1,2), mar=c(0,0,0,0))
image.plot(x=prj$x, y=prj$y, z=m.spat, asp=1,
           xlab='', ylab='', axes=FALSE, horizontal=TRUE)
plot(nwEngland, add=TRUE)
image.plot(x=prj$x, y=prj$y, z=sd.spat, asp=1,
           xlab='', ylab='', axes=FALSE, horizontal=TRUE)
plot(nwEngland, add=TRUE)
```

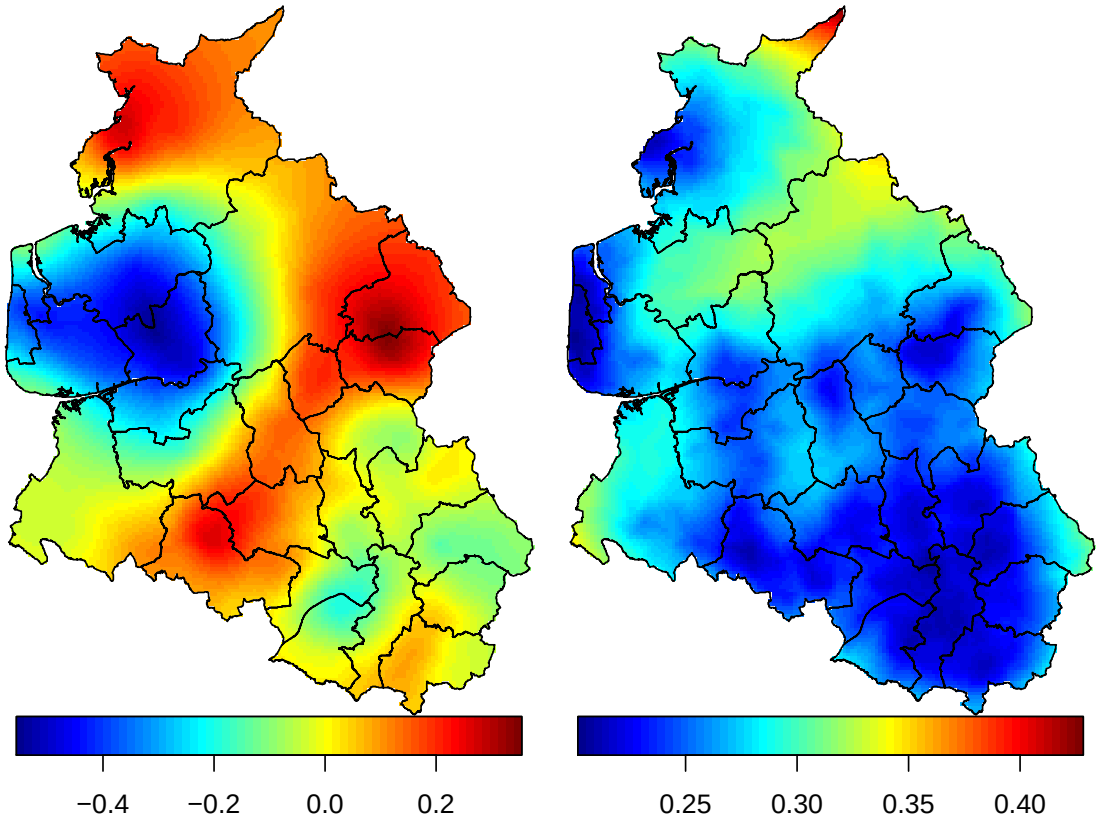

Figure 5.1: Map of the spatial effect in the Weibull survival model. Posterior mean (left) and posterior standard deviation (right).

5.2 Cox proportional hazard survival model

The Cox proportinal hazard (coxph) survival model can be written as a Poisson regression. In `R-INLA` it is done internally using the `inla.coxph()` function. We need this function to pre-prepare the data before using it as input for the `inla.stack()` function. So, we have to supply the data converted into the Poisson regression data to `inla.stack()` to prepare then the data stack in order to have the SPDE model included in the model.

We first define a formula without the spatial effect to have the data extended to the Poisson model.

```
formula0 <- inla.surv(time, cens) ~ 0 + a0 + sex + age + wbc + tpi
cph.leuk <- inla.coxph(formula0, data=data.frame(a0=1, Leuk[, c(1:8)]),
                       control.hazard=list(n.intervals=25))
```

For comparison pourpose we can fit this model using

```
cph.res0 <- inla(formula0, 'coxph', data=data.frame(a0=1,Leuk))
```

Then, we have to include the spatial effect in the formula

```
cph.formula <- update(cph.leuk$formula,
                      '. ~ . + f(spatial, model=spde)')
```

The projector matrix can be built with

```
cph.A <- inla.spde.make.A(mesh, loc=cbind(
    cph.leuk$data$xcoord, cph.leuk$data$ycoord))
```

And the stack is built considering the relevant data from the output of the `inla.coxph()` function

```
cph.stk <- inla.stack(data=c(list(E=cph.leuk$E),
                             cph.leuk$data[c('y..coxph')]),
                      A=list(cph.A, 1),
                      effects=list(
                          list(spatial=1:spde$n.spde),
                          cph.leuk$data[c('baseline.hazard', 'a0',
                                          'age', 'sex', 'wbc', 'tpi')]))
cph.data <- c(inla.stack.data(cph.stk), cph.leuk$data.list)
```

Then, we only had to use it considering the Poisson likelihood

```
cph.res <- inla(cph.formula, family='Poisson',
                data=cph.data, E=cph.data$E,
                control.predictor=list(A=inla.stack.A(cph.stk)))
```

We can compare the results with the result from the **survival** package.

```
require(survival)
m0 <- coxph(Surv(time, cens) ~ sex + age + wbc + tpi, Leuk)
cbind(survival=c(NA, coef(m0)),
        r0=cph.res0$summary.fix[,1], r1=cph.res$summary.fix[,1])
```

```
      survival            r0            r1
            NA -10.190761490 -10.157180989
sex 0.052175883   0.057953204   0.068903586
age 0.029617048   0.033258544   0.034806023
wbc 0.003072442   0.003395713   0.003429396
tpi 0.029284096   0.034230354   0.032323052
```

The spatial effect fitted very is similar to that from the Weibull model

```
cor(as.vector(m.spat),
      as.vector(inla.mesh.project(
          prj, cph.res$summary.ran$spatial$mean)), use='p')
```

```
[1] 0.9939685
```

```
cor(as.vector(sd.spat),
      as.vector(inla.mesh.project(
          prj, cph.res$summary.ran$spatial$sd)), use='p')
```

```
[1] 0.9972132
```

Chapter 6

Semicontinuous model to daily rainfall

This chapter is better presented in the last chapter of [Blangiardo and Cameletti, 2015].

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.

6.1 The hurdle continuous model

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.

6.2 The daily rainfall data

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 2011. 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 for the first seven days 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 6.1. Also, the histogram of the positive precipitation are on the rigth graph of this Figure. Both produced by the code below

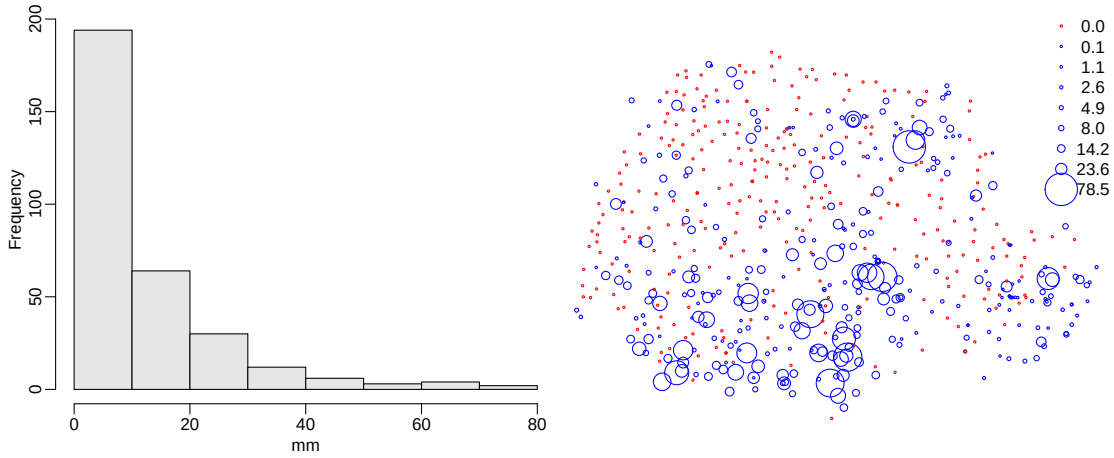

Figure 6.1: Histogram of rainfall amount and map of precipitation.

```
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)))
```

6.3 Model fitting and some results

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

```
[1] 871
```

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 0.3493 0.0792     0.1949   0.3488     0.5059 0.3474
beta.yz 1.7688 0.2165     1.3539   1.7646     2.2041 1.7520
```

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      703.5910 2080.197
joint.zy 704.7750 2130.293
joint.yz 714.5314 2120.913
```

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
sep      0.2304 0.8510    -1.5061   0.2011     2.1153 0.1634   0
joint.zy 0.0681 0.4959    -0.9324   0.0627     1.1003 0.0545   0
joint.yz 0.0257 0.3342    -0.6510   0.0267     0.6979 0.0290   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
0.3772628 0.4440634 0.4672347
```

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
sep      2.1347 0.1296     1.8781   2.1346     2.3917 2.1345   0
joint.zy 2.0848 0.2053     1.6743   2.0869     2.4901 2.0971   0
joint.yz 1.9044 0.2224     1.4657   1.9035     2.3511 1.9042   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] 0.5170283 8.0431451
```

and it looks similar to observed:

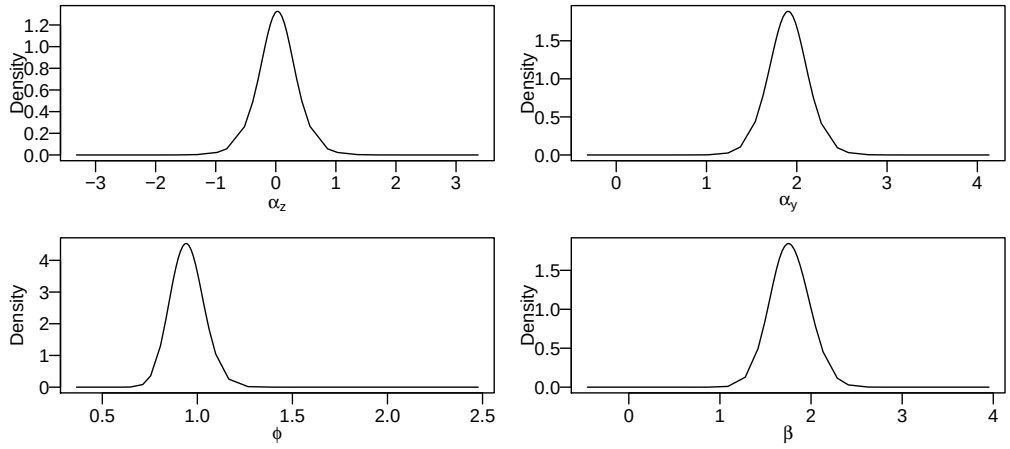

Figure 6.2: Posterior marginal distributions of α_z (top left), α_y (top right), ϕ (bottom left) and β (bottom right).

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

```
      occ       rain
0.5172414 11.4606349
```

The summary for the posterior marginal distribution of Gamma precision parameter is

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

```
                                                      mean         sd
Precision parameter for the Gamma observations[2] 0.9468243 0.08812462
                                                  0.025quant  0.5quant
Precision parameter for the Gamma observations[2]  0.7817749 0.9443022
                                                  0.975quant      mode
Precision parameter for the Gamma observations[2]   1.127976 0.9407197
```

We look the posterior marginal distributions of α_z, α_y, ϕ and β on Figure 6.2 with commands below

```
par(mfrow=c(2,2), mar=c(3,3,.5,.5), mgp=c(1.5,.5,0), las=1)
plot(res.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 6.3 the mean and this quantiles for each x in order of the posterior mean, with code below

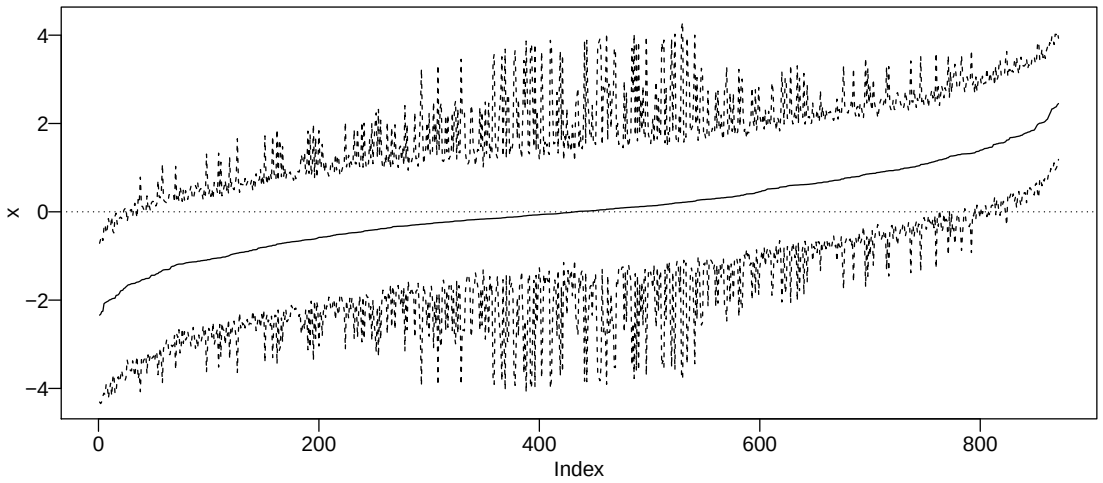

Figure 6.3: Posterior man of each x_i (continuous line) and its 2.5% and 97.5% quantiles of the posterior marginal distributions (dashed lines).

```
ordx <- order(res.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

```
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 714.5314 2120.913
```

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

6.4 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

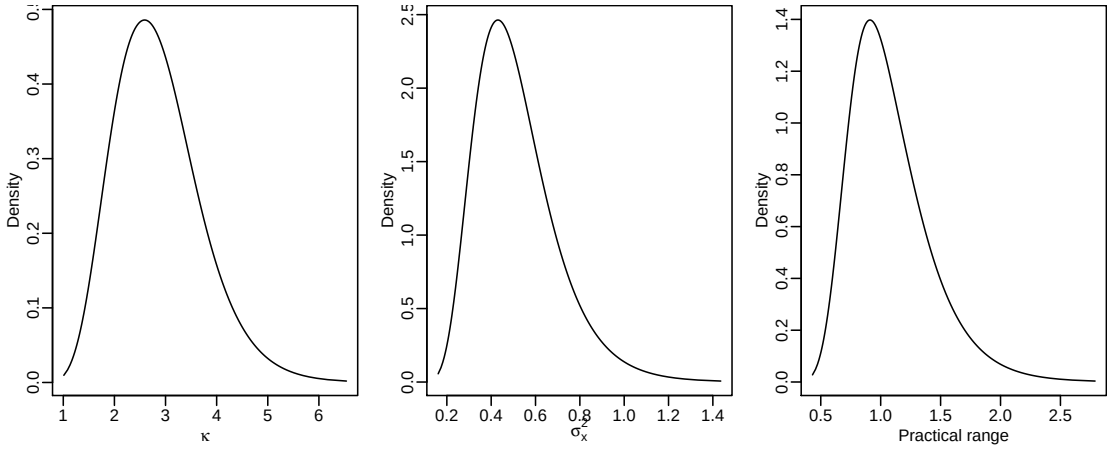

Figure 6.4: Posterior marginal distribution of κ (left), σ_x^2 (mid) and of the practical range (right).

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

```
[1] 0.5158874
```

And to the practical range of the process we have

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

```
[1] 1.081832
```

At the Figure 6.4 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)
```

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

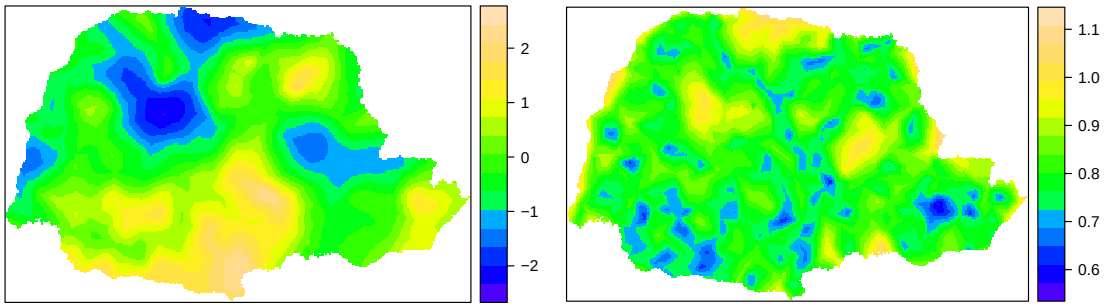

Figure 6.5: Posterior mean (left) and standard deviation (right) of the random field.

```
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 6.5. with code below

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

Comparing the left graph on Figure 6.5 with the right graph on the Figure 6.1 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 7

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.

7.1 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.

7.1.1 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))
```

7.2 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=.05,
                                    max.edge=c(.1,.2)))$n
```

```
[1] 436
```

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)))
```

7.3 The results

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

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

```
    True   mean     sd 0.025quant 0.5quant 0.975quant    mode kld
a.y   -5 -4.499 1.3429    -7.2823  -4.5146    -1.6223 -4.5372   0
```

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 5.9444 0.7889     4.5300   5.8987     7.6261 5.8138
Prec.y    2 1.1638 0.2399     0.7593   1.1414     1.6982 1.0992
```

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

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

```
             True    mean    sd 0.025quant 0.5quant 0.975quant    mode
Beta for c.y   -3 -2.7505 0.157    -3.0624  -2.7491    -2.4454 -2.7449
```

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 4.5472 1.5590 8.5388
s2x    3 4.6872 1.2814 9.2317
```

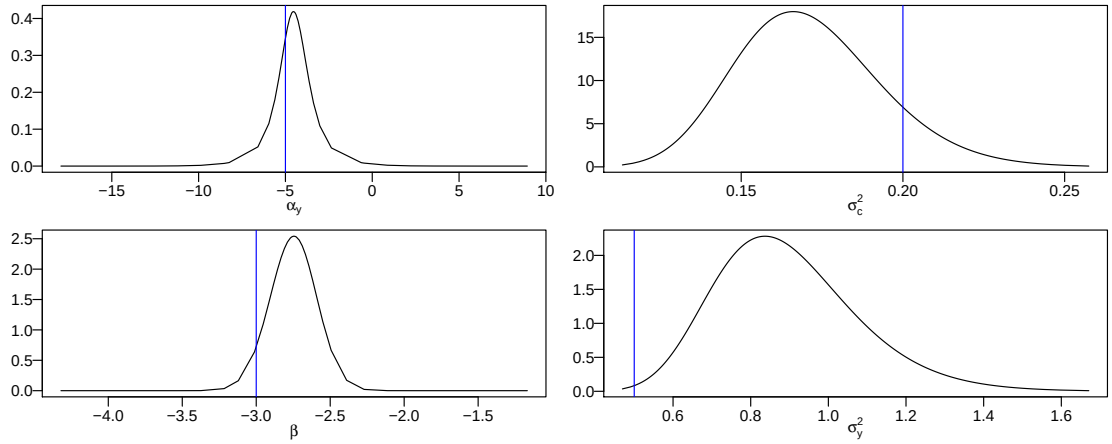

Figure 7.1: Posterior distribution of the likelihood parameters.

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 3.9368 2.1204  5.8355
kappa.x    7 5.9639 2.2872 10.2242
```

We see the posterior distribution of likelihood parameters on Figure 7.1 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 7.1 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 7.2 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)
```

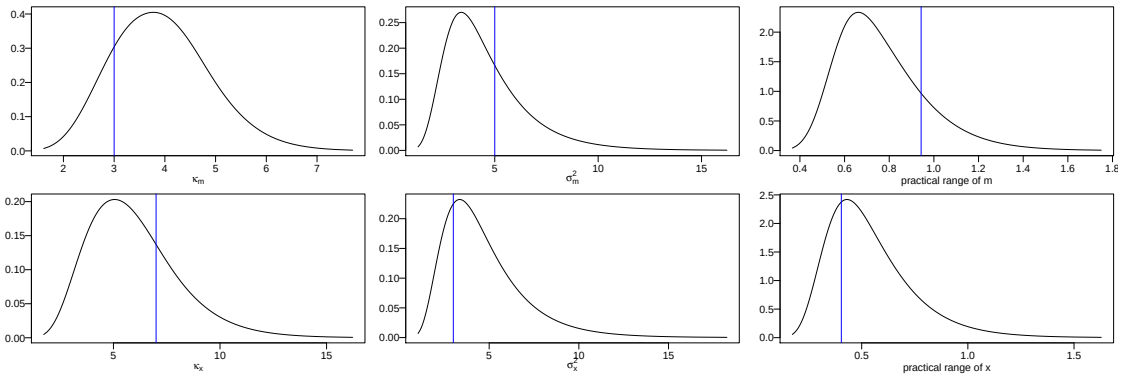

Figure 7.2: Posterior distribution of all the parameters of both random field.

```
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 7.2 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,
```

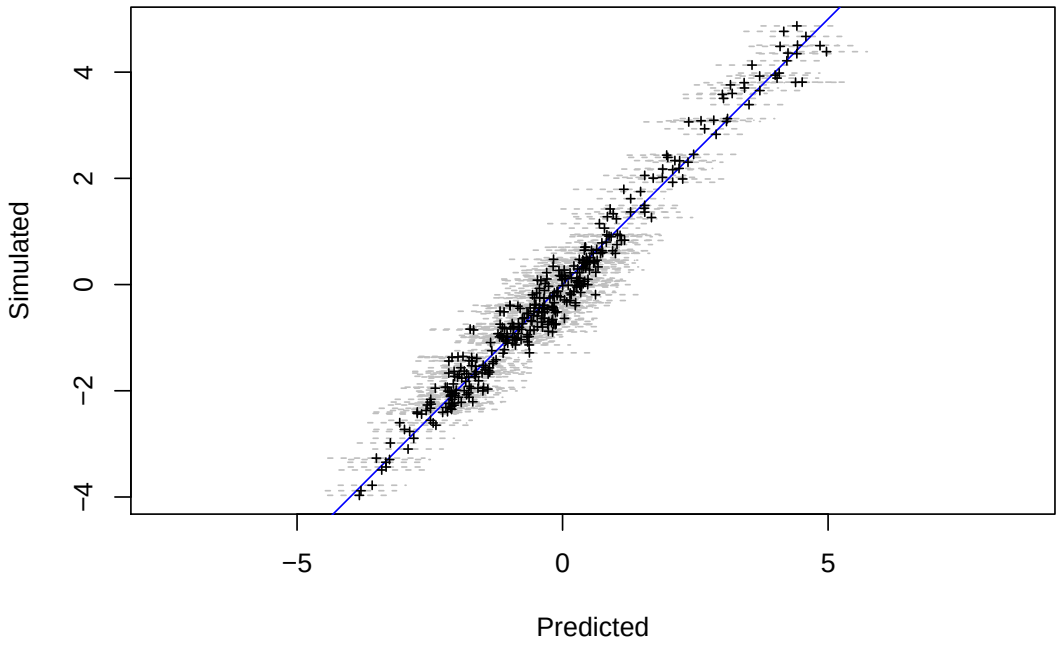

Figure 7.3: Simulated versus predicted values of m (+) and the approximated credibility intervals.

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

on the Figure 7.3. The blue line represents the situation where predicted is equal to simulated. We see that the prediction are wery well.

Chapter 8

Explanatory variables in the covariance

8.1 Introduction

In this Chapter we present the model proposed in [Ingebrigtsen et al., 2014]. To introduce variables in the covariance matrix 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).

These matrices are written in a more general way

$$\boldsymbol{Q} = \boldsymbol{D}^{(0)}(\boldsymbol{D}^{(1)}\boldsymbol{M}^{(0)}\boldsymbol{D}^{(1)} + \boldsymbol{D}^{(2)}\boldsymbol{D}^{(1)}\boldsymbol{M}^{(1)} + (\boldsymbol{M}^{(1)})^T\boldsymbol{D}^{(1)}\boldsymbol{D}^{(2)} + \boldsymbol{M}^{(2)})\boldsymbol{D}^{(0)} \tag{8.1}$$

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 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. This is a kind of non stationarity that allows variation in the variance and range over the domain.

8.2 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 (8.1) 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
```

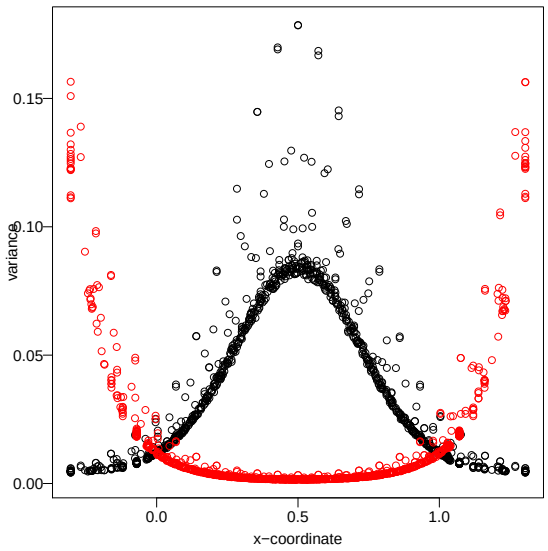

Figure 8.1: Variances implied by both non stationary process defined.

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 8.1 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)
```

8.3 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 8.2. 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

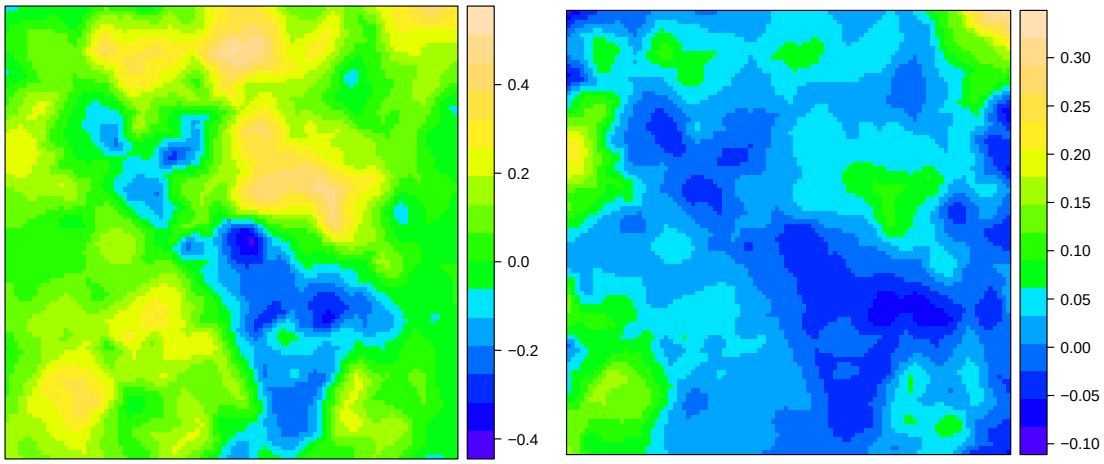

Figure 8.2: Two simulated random fields, using two diferent θ on same basis functions.

```
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)
```

8.3.1 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.

8.4 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     1.6894   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   1.0671     1.1552  1.0661
```

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

8.5 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 8.3 with commands below

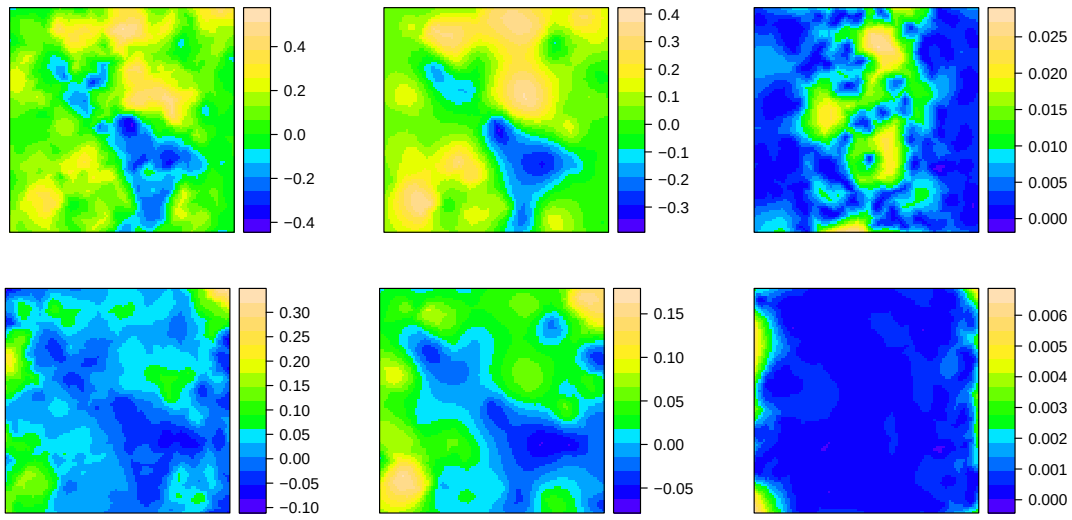

Figure 8.3: Simulated (top and bottom left), posterior mean (top and bottom mid) and the posterior variance (top and bottom right) for both random fields.

```
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 8.3 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 9

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.

9.1 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 2.5 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, return.attributes=TRUE)
dim(x.k)
```

```
[1] 616  12
```

Now, we define the autoregressive parameter ρ

```
rho <- 0.7
```

and get the sample

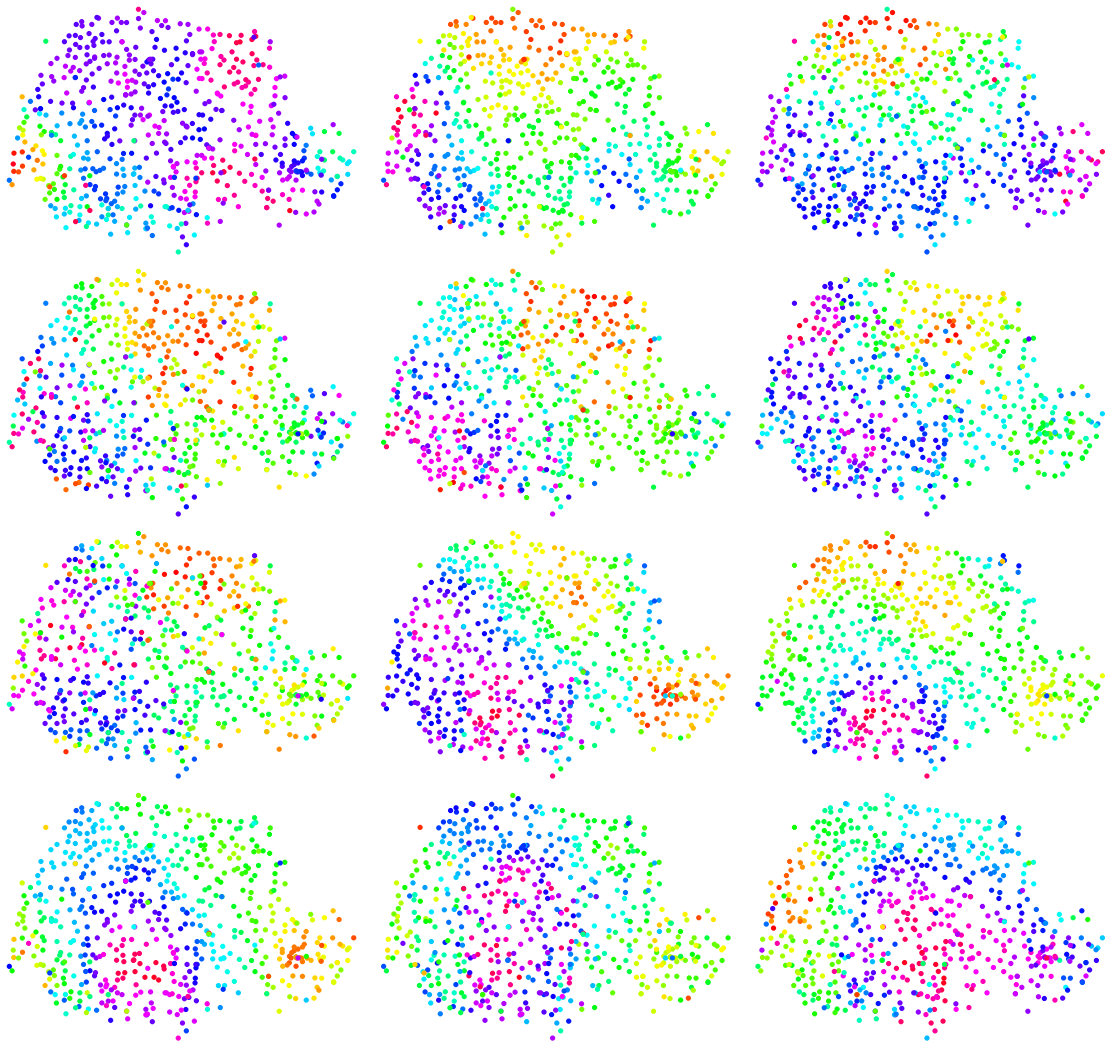

Figure 9.1: Realization of the spatio temporal random field.

```
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 9.1 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
-2.6264642 -1.5358513 -0.6016356
```

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.

9.2 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))
```

9.3 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  -2.6211 -2.5707 0.7663    -4.0931  -2.5711    -1.0483 -2.5716   0
B  -1.5821 -1.5707 0.7663    -3.0931  -1.5711    -0.0483 -1.5716   0
C  -0.5949 -0.5707 0.7663    -2.0931  -0.5711     0.9517 -0.5716   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
GroupRho for i 0.65199 0.01801    0.61591   0.6522    0.68662 0.65247
```

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 9.2 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)
```

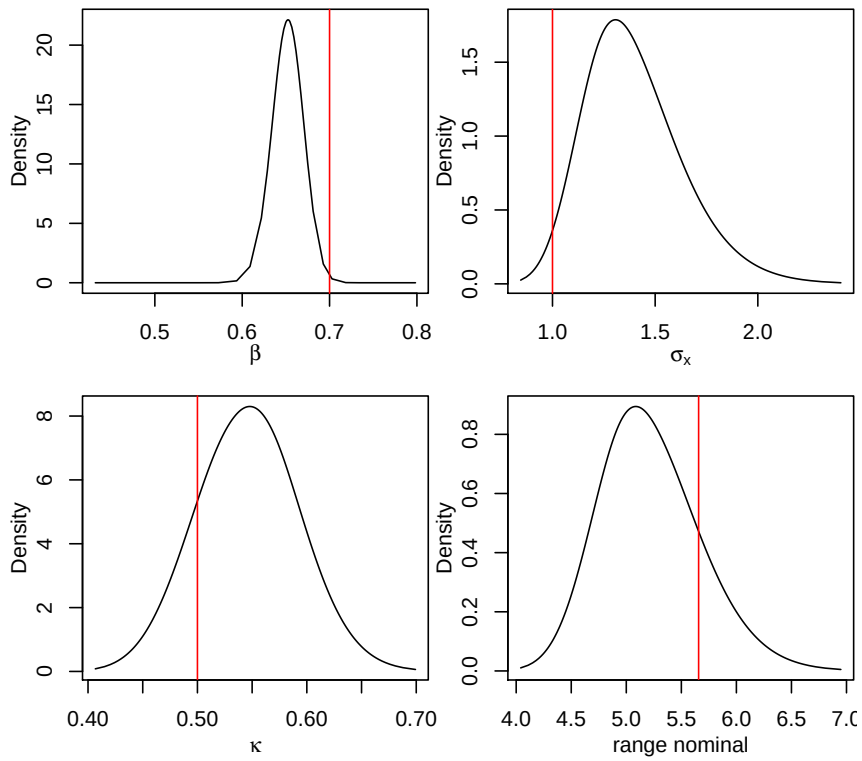

Figure 9.2: Marginal posterior distribution for β (top left), σ_x (top right), κ (bottom left) and the nominal range (bottom right). The red vertical lines are placed at true value.

9.4 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])
```

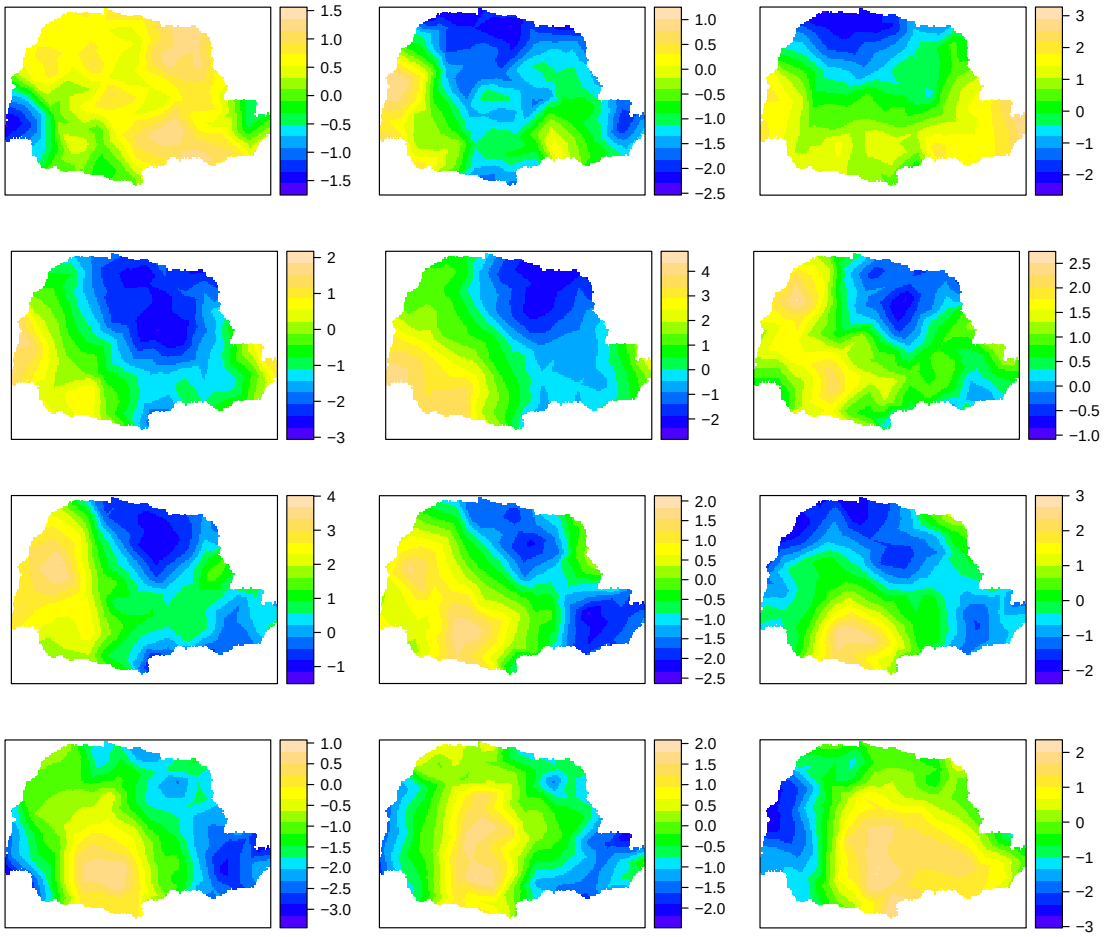

Figure 9.3: Visualization of the posterior mean of the spatio temporal random field.

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 9.3 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)))
```

9.5 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, ]
```

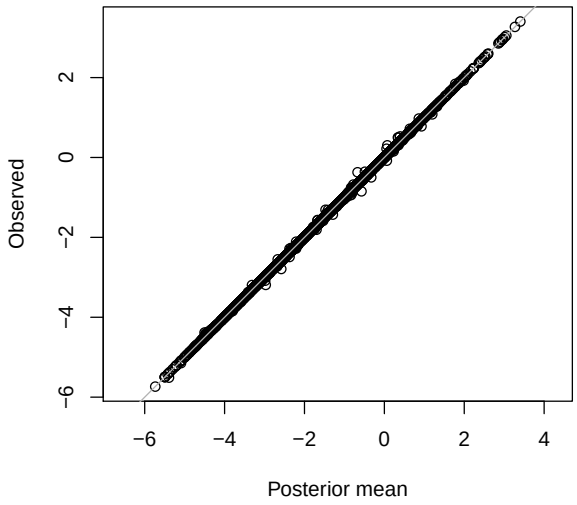

Figure 9.4: Validation: Observed versus posterior mean.

```
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 9.4.

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

Chapter 10

Lowering resolution of a spatio-temporal model

It can be challenging when dealing with large data sets. In this chapter we want to show how to fit a model using some dimention reduction.

Before starting, the spatial mesh and the SPDE model is built with the following code.

```
data(PRprec)
bound <- inla.nonconvex.hull(as.matrix(PRprec[,1:2]), .2, .2, resol=50)
mesh.s <- inla.mesh.2d(bound=bound, max.edge=c(1,2),
                       offset=c(.0001,0.7), cutoff=0.5)
spde.s <- inla.spde2.matern(mesh.s)
```

10.1 Data temporal aggregation

The data we are going to analyse is the daily rainfall in Paraná. We have rainfall at 616 location points observed over 365 days.

```
dim(PRprec)
```

```
[1] 616 368
```

```
PRprec[1:2, 1:7]
```

```
  Longitude Latitude Altitude d0101 d0102 d0103 d0104
1  -50.8744 -22.8511      365     0     0     0     0
3  -50.7711 -22.9597      344     0     1     0     0
```

To this example we are going to analyse the probability of rain. So we only consider if the value where bigger than 0.1 or not.

To reduce the time dimension of the data, we aggregate it summing every five days. At end we have two data matrix, one with the number of days without NA in each station and another with the number of raining days on such stations.

```
table(table(id5 <- 0:364%/%5 + 1))
```

```
 5
73
```

```
n5 <- t(apply(!is.na(PRprec[,3+1:365]), 1, tapply, id5, sum))
y5 <- t(apply(PRprec[,3+1:365]>0.1, 1, tapply, id5, sum, na.rm=TRUE))
k <- ncol(n5);       table(as.vector(n5))
```

```
   0    1    2    3    4    5
3563   77   72   95  172 40989
```

From now, our data has 73 time points.

From the above table, we can see that there were 3563 periods of five days with no data recorded. The first approach can be removing such pairs data, both y and n. If we do not remove it, we have to assign NA to y when $n = 0$. However, we have to assign a positive value, five for example, for such n and it will be treated as a prediction scenario.

```
y5[n5==0] <- NA;     n5[n5==0] <- 5
```

10.2 Lowering temporal model resolution

This approach is better expained in[Blangiardo and Cameletti, 2015]. The main idea is to place some knots over the time window and define a model on such knots. Them project the model into the data time points as we do using the Finite Element Method in the SPDE approach.

We choose to place knots at each 6 time points of the temporally aggregated data, which has 73 time points. So, we en up with only 12 knots over time.

```
bt <- 6;   gtime <- seq(1+bt, k, length=round(k/bt))-bt/2
mesh.t <- inla.mesh.1d(gtime, degree=1)
table(igr <- apply(abs(outer(mesh.t$loc, 1:k, '-')), 2, which.min))
```

```
1  2  3  4  5  6  7  8  9 10 11 12
7  6  6  6  6  6  6  6  6  6  6  6
```

The fist knot is closer to 7 time blocks and the others to 6.

The model dimention is then

```
spde.s$n.spde*mesh.t$n
```

```
[1] 1152
```

To built the spatial projector matrix, we need to replicate the spatial coordinates as

```
n <- nrow(PRprec)
st.sloc <- cbind(rep(PRprec[,1], k), rep(PRprec[,2], k))
```

and then to consider the temporal mesh considering the group index in the scale of the data to be analised.

```
Ast <- inla.spde.make.A(mesh=mesh.s, loc=st.sloc,
                        group.mesh=mesh.t, group=rep(1:k, each=n))
```

The index set and the stack is built as usual

```
idx.st <- inla.spde.make.index('i', n.spde=spde.s$n.spde,
                               n.group=mesh.t$n)
dat <- inla.stack(data=list(yy=as.vector(y5), nn=as.vector(n5)),
                  A=list(Ast, 1),
                  effects=list(idx.st,
                      data.frame(mu0=1,
                                 altitude=rep(PRprec$Alt/1e3, k))))
```

The formula is also as the usual for the separable spatio temporal model

```
form <- yy ~ 0 + mu0 + altitude +
      f(i, model=spde.s, group=i.group,
        control.group=list(model='ar1'))
```

To "fit" the model as fast as possible, we use the 'gaussian' approximation and the Empirical Bayes ('eb') integration strategy over the hyperparameters. We also fixed the mode at the values we have find in previous analisys.

```
result <- inla(form, 'binomial', data=inla.stack.data(dat),
                 Ntrials=inla.stack.data(dat)$nn,
                 control.predictor=list(A=inla.stack.A(dat)),
                 control.mode=list(theta=c(-0.48, -0.9, 2.52)), ###restart=TRUE),
                 control.inla=list(strategy='gaussian', int.strategy='eb'))
```

We can plot the fitted spatial effect for each temporal knot and overlay the proportion raining days considering the data closest to the time knots.

Defining a grid to project

```
data(PRborder)
r0 <- diff(range(PRborder[,1]))/diff(range(PRborder[,2]))
prj <- inla.mesh.projector(mesh.s, xlim=range(PRborder[,1]),
                           ylim=range(PRborder[,2]), dims=c(100*r0, 100))
in.pr <- inout(prj$lattice$loc, PRborder)
```

Project the posterior mean fitted at each time knot

```
mu.spat <- lapply(1:mesh.t$n, function(j) {
    r <- inla.mesh.project(prj, field=result$summary.ran$i$mean[
                          1:spde.s$n.spde + (j-1)*spde.s$n.spde])
    r[!in.pr] <- NA;   return(r)})
```

The images in Figure 10.1 were made using the following commands

```
par(mfrow=c(4,3), mar=c(0,0,0,0))
zlm <- range(unlist(mu.spat), na.rm=TRUE)
for (j in 1:mesh.t$n) {
      image.plot(x=prj$x, y=prj$y, z=mu.spat[[j]], asp=1, axes=FALSE, zlim=zlm)
      lines(PRborder)
      points(PRprec[, 1:2],
             cex=rowSums(y5[, j==igr], na.rm=TRUE)/rowSums(n5[,j==igr]))
  }
```

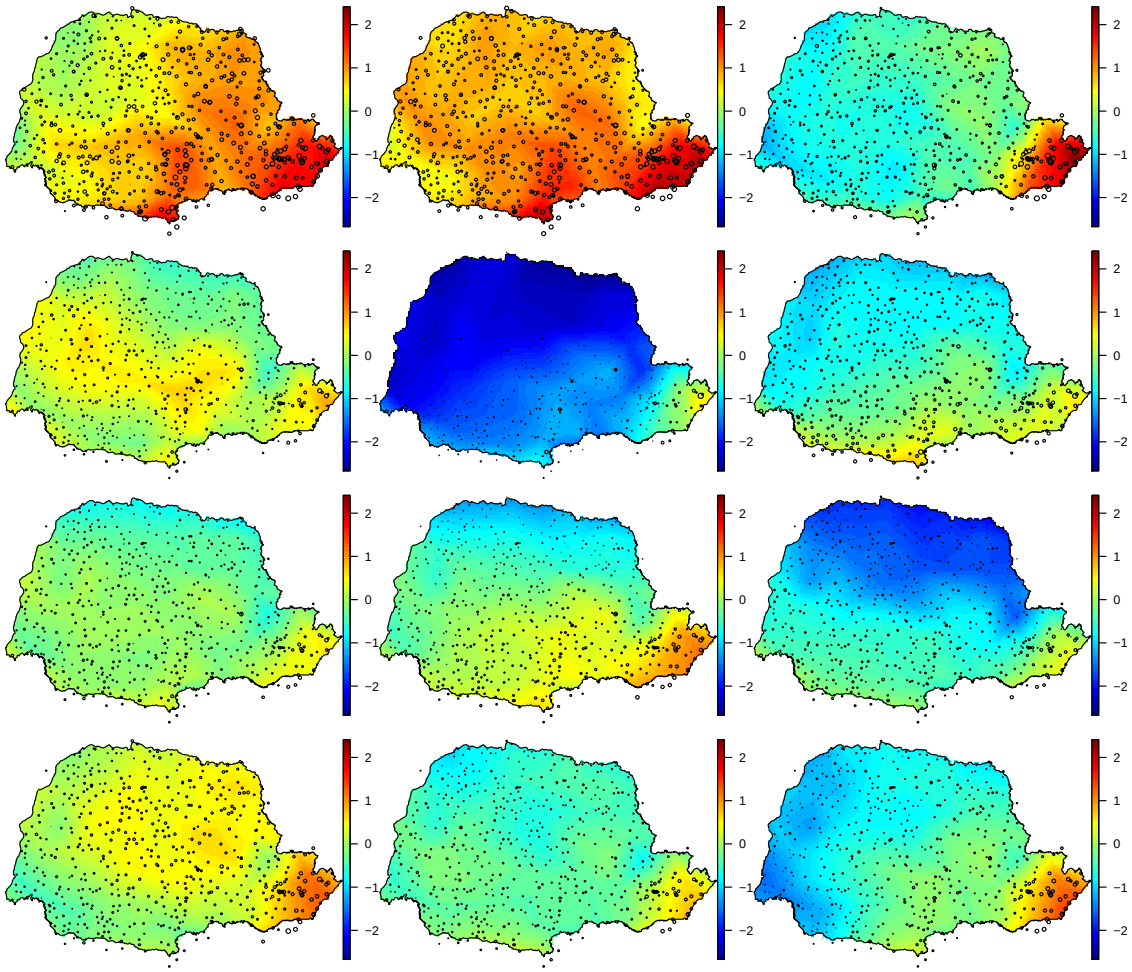

Figure 10.1: Spatial effect at each time knots.

Chapter 11

Space-time coregionalization model

In this Chapter we present a way to fit the Bayesian coregionalization model proposed by [Schimdt and Gelfand, 2003]. We consider a generalization for the space-time case. Also, the approach implemented in R-INLA allows completelly missalignment on space and time for all the outcomes, it only need the same space-time observation window.

WARNING: a crude mesh is used and simplifications in the fitting process are made to run this example in a short time.

11.1 The model and parametrization

An example for the particular case of three outcomes defined as folowing

$$y_1(s,t) = \alpha_0 + z_1(s,t)$$

$$y_2(s,t) = \alpha_1 + z_2(s,t) + \lambda_1 y_1(s,t)$$

$$y_3(s,t) = \alpha_2 + z_3(s,t) + \lambda_2 y_1(s,t) + \lambda_3 y_2(s,t)$$

This model can be fitted in R-INLA using the copy feature. In the parametrization above it is needed to copy the linear predictor in the first equation to the second and the linear predictor in the second equation to the third. However, this model can be reparametrized to make the fitting process easier. This reparametrization is to change the second and the third equation as follows

$$y_2(s,t) = \alpha_1 + \lambda_1\alpha_1 + \lambda_1 z_1(s,t) + z_2(s,t) + e_2(s,t)$$

$$y_3(s,t) = \alpha_2 + (\lambda_2 + \lambda_3\lambda_1)\alpha_1 + \lambda_3\alpha_2 + (\lambda_2 + \lambda_3\lambda_1)z_1(s,t) + \lambda_3 z_2(s,t) + z_3(s,t) + e_3(s,t)$$

We have then two new intercepts $\alpha_1^* = \alpha_1 + \lambda_1\alpha_1$ and $\alpha_2^* = \alpha_2 + (\lambda_2 + \lambda_3\lambda_1)\alpha_1 + \lambda_3\alpha_2$.

We will use the copy feature to fit $\lambda_1 = \beta_1$. In the second equation and $\lambda_2 + \lambda_3\lambda_1 = \beta_2$ will be the first copy parameter in the third equation. A second copy will be used in the third equation to fit $\lambda_3 = \beta_3$.

11.2 Data simulation

Parameter setting

```
alpha <- c(-5, 3, 10) ### intercept on reparametrized model
m.var <- (3:5)/10 ### random field marginal variances
kappa <- c(10, 7, 5) ### GRF scales: inverse range parameters
beta <- c(.7, .5, -.5) ### copy par.: reparam. coregionalization par.
rho <- c(.7, .8, .9) ### temporal correlations
n <- 50;  k <- 15  ### number of spatial locations and time points
```

It is not required to the spatial locations to be the same for each process to fit this model in R-INLA, as shown in the Chapter 7. It is also not required for the time points to be the same, as we can define the model on a set of time knots, see Chapter 10. However, to simplify the code, we just use the same spatial locations and the same time points for all three processes.

```
loc <- cbind(runif(n), runif(n))
```

We use the function defined in the section 2.5 to simulate independent random field realizations for each time.

```
x1 <- rspde(loc, kappa[1], m.var[1], n=k, seed=1)
x2 <- rspde(loc, kappa[2], m.var[2], n=k, seed=2)
x3 <- rspde(loc, kappa[3], m.var[3], n=k, seed=3)
```

The time evolution will follows an autoregressive first order process as we used in Chapter 9.

```
z1 <- x1; z2 <- x2; z3 <- x3
for (j in 2:k) {
      z1[, j] <- rho[1] * z1[,j-1] + sqrt(1-rho[1]^2) * x1[,j]
      z2[, j] <- rho[2] * z2[,j-1] + sqrt(1-rho[2]^2) * x2[,j]
      z3[, j] <- rho[3] * z3[,j-1] + sqrt(1-rho[3]^2) * x3[,j]
  }
```

The term $\sqrt{(1-\rho^2)}$ is because we are sampling from the stationary distribution, and is in accord to the first order autoregressive process parametrization implemented in R-INLA.

Then we define the observation samples

```
y1 <- alpha[1] + z1
y2 <- alpha[2] + beta[1] * z1 + z2
y3 <- alpha[3] + beta[2] * z1 + beta[3] * z2 + z3
```

11.3 Model fitting

Build the mesh to use in the fitting process (this is a crude mesh used here for short computational time pourpose)

```
mesh <- inla.mesh.2d(loc, max.edge=0.3, offset=0.2, cutoff=0.15)
```

Define the object that includes the SPDE stuff

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

Defining all the index set for the space-time fields and the for the copies. As we have the same mesh, they are the same.

```
s1 = s2 = s3 = s12 = s13 = s23 = rep(1:spde$n.spde, times=k)
g1 = g2 = g3 = g12 = g13 = g23 = rep(1:k, each=spde$n.spde)
```

Prior for ρ_j is chosen as the Penalized Complexity prior, [Simspon et al., 2015]

```
rho1p <- list(theta=list(prior='pc.rho1', param=c(0, 0.9)))
ctr.g <- list(model='ar1', hyper=rho1p)
```

Ther prior chosen above consider $P(\rho > 0) = 0.9$.

Priors for each of the the copy parameters $N(0, 10)$

```
hc3 <- hc2 <- hc1 <- list(theta=list(prior='normal', param=c(0,10)))
```

The priors for the fields are the default ones, described in [Lindgren and Rue, 2013].
Define the formula including all the terms in the model.

```
form <- y ~ 0 + intercept1 + intercept2 + intercept3 +
    f(s1, model=spde, ngroup=k, group=g1, control.group=ctr.g) +
    f(s2, model=spde, ngroup=k, group=g2, control.group=ctr.g) +
    f(s3, model=spde, ngroup=k, group=g3, control.group=ctr.g) +
    f(s12, copy="s1", group=g12, fixed=FALSE, hyper=hc1) +
    f(s13, copy="s1", group=g13, fixed=FALSE, hyper=hc2) +
    f(s23, copy="s2", group=g23, fixed=FALSE, hyper=hc3)
```

Define the projector matrix (all they are equal in this example, but it can be different)

```
stloc <- kronecker(matrix(1,k,1), loc) ### rep. coordinates each time
A <- inla.spde.make.A(mesh, stloc, n.group=k, group=rep(1:k, each=n))
```

Organize the data in three data stack and join it

```
stack1 <- inla.stack(
    data=list(y=cbind(as.vector(y1), NA, NA)), A=list(A),
    effects=list(list(intercept1=1, s1=s1, g1=g1)))
stack2 <- inla.stack(
    data=list(y=cbind(NA, as.vector(y2), NA)), A=list(A),
    effects=list(list(intercept2=1, s2=s2, g2=g2,
                      s12=s12, g12=g12)))
stack3 <- inla.stack(
    data=list(y=cbind(NA, NA, as.vector(y3))), A=list(A),
    effects=list(list(intercept3=1, s3=s3, g3=g3,
                      s13=s13, g13=g13, s23=s23, g23=g23)))
stack <- inla.stack(stack1, stack2, stack3)
```

We consider that there is no nugget effect in all the three outcomes. So the likelihood precision parameter for each likelihood are set fixed as a high value:

```
hfix <- list(hyper=list(theta=list(initial=5, fixed=TRUE)))
```

Precision equals to exp(5) is not so high in this case and we chose this value to add some 'flexibility' to help the fitting process.

We have 12 hyperparameters in the model. To make the optimization process fast, we use the parameter values used in the simulation as the initial values

```
theta.ini <- c(-log(4*pi*m.var*kappa^2)/2,
                log(kappa), binomial()$linkfun(rho), beta
                )[c(1,4, 7, 2,5, 8, 3,6, 9, 10:12)]
```

With 12 hyperparameters the CCD strategy uses 281 integration points to compute

$$\pi(x_i|y) = \int \pi(y|x)\pi(x|\theta)\pi(\theta)d\theta$$

We avoid it using the Empirical Bayes strategy approaching such marginals using only the modal configuration of θ.

```
(result <- inla(form, rep('gaussian', 3), data=inla.stack.data(stack),
                control.family=list(hfix, hfix, hfix),
                control.mode=list(theta=theta.ini, restart=TRUE),
                control.inla=list(int.strategy='eb'),##avoid integration
                control.predictor=list(A=inla.stack.A(stack))))$cpu
```

```
 Pre-processing    Running inla Post-processing           Total
      0.7594135     157.3445377       0.2745566     158.3785079
```

```
 result$logfile[grep('Number of function evaluations', result$logfile)]
```

```
[1] "Number of function evaluations = 835"
```

```
 round(result$misc$theta.mode, 2)
```

```
 [1] -3.59  2.58  1.75 -3.25  2.07  2.80 -2.67  1.50  3.88  0.65  0.58 -0.55
```

Summary of the posterior marginal density for the intercepts

```
 round(cbind(true=alpha, result$summary.fix), 2)
```

```
           true  mean   sd 0.025quant 0.5quant 0.975quant  mode kld
intercept1   -5 -5.14 0.10      -5.33    -5.14      -4.95 -5.14   0
intercept2    3  2.89 0.26       2.39     2.89       3.39  2.89   0
intercept3   10 10.26 0.54       9.20    10.26      11.32 10.26   0
```

Summary of the posterior marginal density for the temporal correlations:

```
 round(cbind(true=rho, result$summary.hy[c(3,6,9),]), 4)
```

```
                true   mean     sd 0.025quant 0.5quant 0.975quant   mode
GroupRho for s1  0.7 0.7113 0.0367     0.6387   0.7113     0.7820 0.7095
GroupRho for s2  0.8 0.8807 0.0256     0.8227   0.8836     0.9226 0.8894
GroupRho for s3  0.9 0.9618 0.0107     0.9387   0.9625     0.9800 0.9638
```

Summary of the posterior marginal density for the copy parameters:

```
 round(cbind(true=beta, result$summary.hy[10:12,]), 4)
```

```
             true    mean     sd 0.025quant 0.5quant 0.975quant    mode
Beta for s12  0.7  0.6463 0.0483     0.5521   0.6460     0.7418  0.6453
Beta for s13  0.5  0.5836 0.0384     0.5086   0.5834     0.6593  0.5829
Beta for s23 -0.5 -0.5529 0.0361    -0.6246  -0.5526    -0.4825 -0.5519
```

Computing the random field parameters for each field

```
 rf1 <- inla.spde2.result(result, 's1', spde, do.transf=TRUE)
 rf2 <- inla.spde2.result(result, 's2', spde, do.transf=TRUE)
 rf3 <- inla.spde2.result(result, 's3', spde, do.transf=TRUE)
```

The marginal variance for each random field

```
 round(cbind(true=m.var, t(sapply(list(rf1, rf2, rf3), function(rf)
      unlist(inla.zmarginal(rf$marginals.variance.nominal[[1]],
                            silent=TRUE))))), 3)
```

```
     true  mean    sd quant0.025 quant0.25 quant0.5 quant0.75 quant0.975
[1,]  0.3 0.625 0.090      0.474     0.561    0.616     0.680      0.828
[2,]  0.4 0.842 0.168      0.554     0.721    0.827     0.946      1.213
[3,]  0.5 0.941 0.275      0.545     0.743    0.889     1.085      1.618
```

Scale parameter for each random field

```
 round(cbind(true=kappa, t(sapply(list(rf1, rf2, rf3), function(rf)
      unlist(inla.zmarginal(rf$marginals.kappa[[1]], silent=TRUE))))), 3)
```

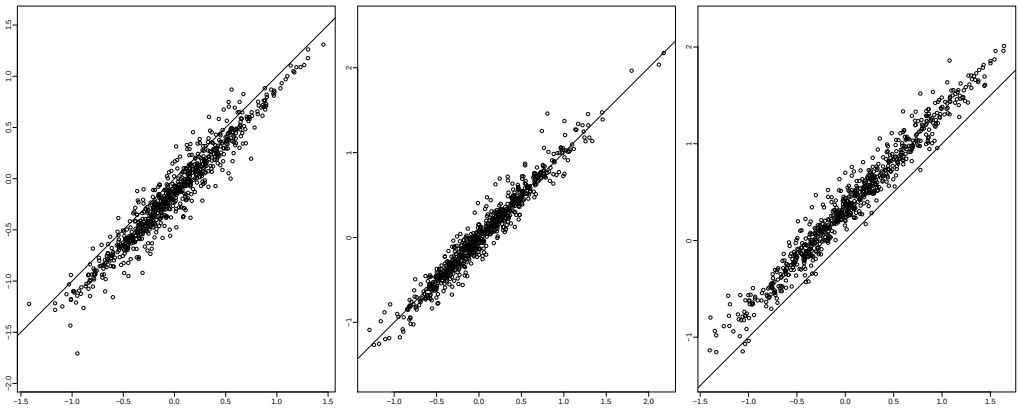

Figure 11.1: True and fitted random field values.

```
     true   mean    sd quant0.025 quant0.25 quant0.5 quant0.75 quant0.975
[1,]   10 13.594 1.853     10.477    12.262   13.395    14.723     17.741
[2,]    7  8.069 0.801      6.655     7.499    8.007     8.575      9.798
[3,]    5  4.546 0.442      3.766     4.231    4.511     4.825      5.501
```

Spatial range nominal for each random field

```
round(cbind(true=sqrt(8)/kappa, t(sapply(list(rf1, rf2, rf3),
                  function(rf) unlist(inla.zmarginal(
                      rf$marginals.range.nom[[1]], silent=TRUE))))), 3)
```

```
      true  mean    sd quant0.025 quant0.25 quant0.5 quant0.75 quant0.975
[1,] 0.283 0.212 0.028      0.159     0.192    0.211     0.230      0.270
[2,] 0.404 0.354 0.035      0.288     0.330    0.353     0.377      0.425
[3,] 0.566 0.628 0.060      0.514     0.586    0.627     0.668      0.751
```

```
par(mfrow=c(1,3), mar=c(2,2,0.5,0.5), mgp=c(1.5,0.5,0))
plot(drop(A%*%result$summary.ran$s1$mean), as.vector(z1),
       xlab='', ylab='', asp=1); abline(0:1)
plot(drop(A%*%result$summary.ran$s2$mean), as.vector(z2),
       xlab='', ylab='', asp=1); abline(0:1)
plot(drop(A%*%result$summary.ran$s3$mean), as.vector(z3),
       xlab='', ylab='', asp=1); abline(0:1)
```

The simplifications for a fast inference may caused that the posterior mean for the marginal variances and the scale parameters are not close to the values used to simulate the data. However, the nominal range is not so far and the posterior marginals cover the value used for simulation in two of the three fields. Also, we can see in Figure 11.1 that the posterior mean of each random field at each space and time location are well correlated with the simulated ones.

Chapter 12

Point process: inference for 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.

12.1 Data simulation

The data simulated on this section is used also on the Chapter 14.

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 13.

In 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

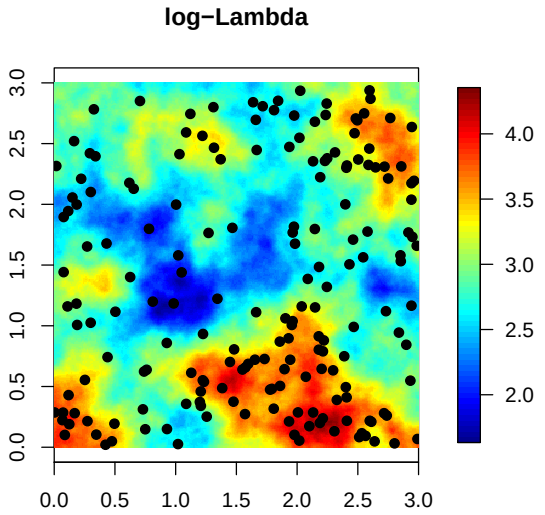

Figure 12.1: Simulated intensity of the point process (left), simulated point pattern (right).

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

Doing the simulation

```
require(RandomFields)
set.seed(1)
lg.s <- rLGCP('matern', beta0,
  ##              c(0, variance=sigma2x, nugget=0, scale=1/kappa, nu=1), win=win)
                var=sigma2x, 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 $log(\Lambda)$ below

```
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
```

We can see the simulated LGRF over the grid and the point pattern simulated in Figure 12.1 produced with the following commands

```
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)
```

12.2 Inference

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

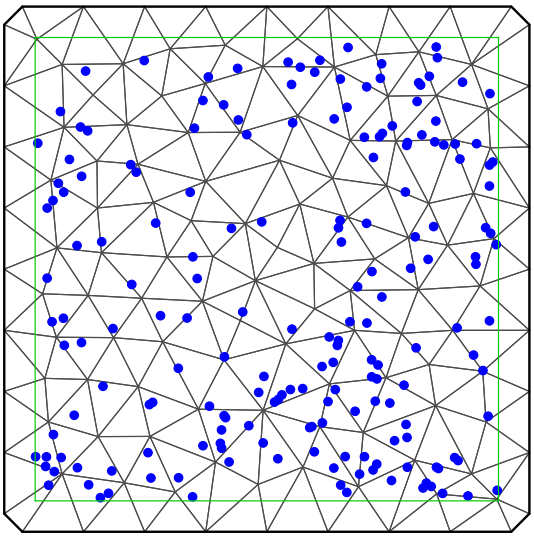

Figure 12.2: Mesh used to inference for the log-Cox process.

12.2.1 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 below).

```
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 12.2 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 12.3.

Third, we get the interection polygons between these polygons and the study region polygon, using functions from the **rgeos** package. First, we define a function to convert from simple coordinates defining a polygon to the `SpatialPolygons` object

```
require(rgeos)
coo2sp <- function(coo) {
      n <- nrow(coo)
      if (any(coo[1,]!=coo[n,]))
          coo <- coo[c(1:n,1),]
      SpatialPolygons(list(Polygons(list(Polygon(coo)), '0')))
  }
```

As an example, we apply it to coordinates that defines the domain area and compute its area

```
gArea(pl.study <- coo2sp(loc.d))
```

```
[1] 9
```

and them compute the intersection areas with

```
sum(w <- sapply(tiles, function(p) {
      pl <- coo2sp(cbind(p$x, p$y))
      if (gIntersects(pl, pl.study))
          return(gArea(gIntersection(pl, pl.study)))
      else return(0)
  }))
```

```
[1] 9
```

and we have some points without any weights (the red ones on Figure 12.3).

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

```
FALSE  TRUE
   19   113
```

At Figure 12.3 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)
```

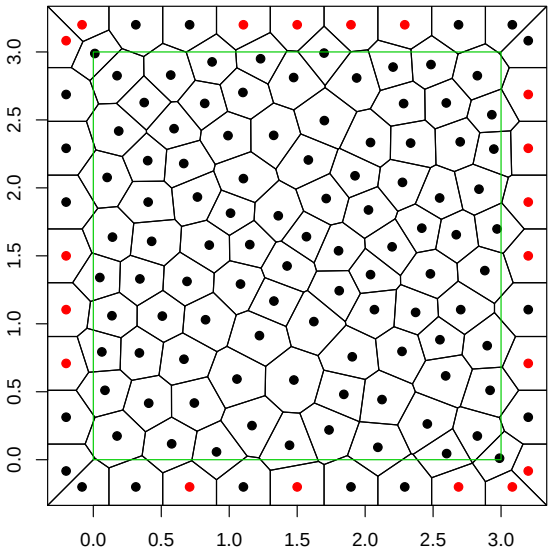

Figure 12.3: Voronoy polygons for the mesh used to inference for the log-Cox process.

12.2.2 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

```
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)))
```

12.2.3 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)
```

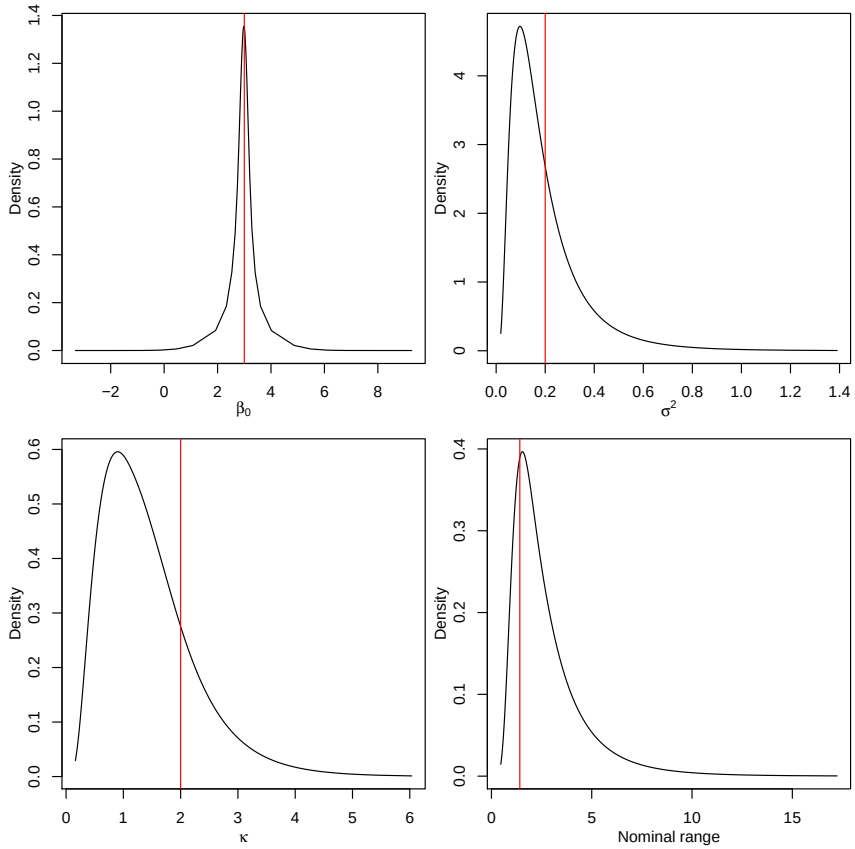

Figure 12.4: Posterior distribution for the parameters of the log-Cox model σ^2 (left), κ (mid) and the nominal range (right)

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 12.4.

```
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 13

Including a covariate on the log-Cox process

In Chapter 12 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.

13.1 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)
                var=sigma2x, 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
```

We can see the covariate values and the simulated LGRF over the grid in Figure 13.1 with the following commands

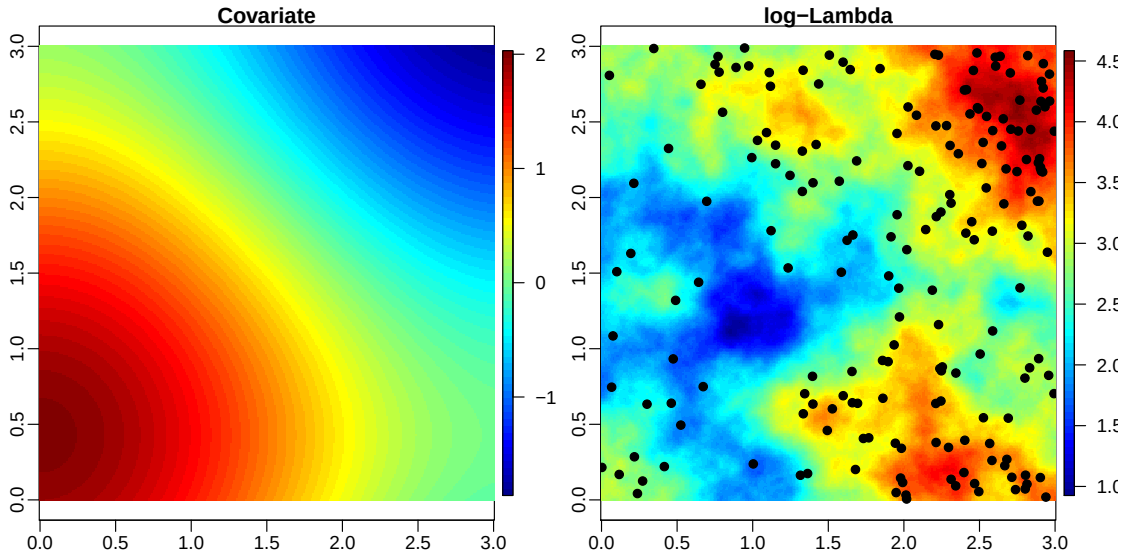

Figure 13.1: Covariate (left), simulated intensity of the point process (mid), simulated point pattern (right).

```
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)
```

13.2 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 below

```
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)))
```

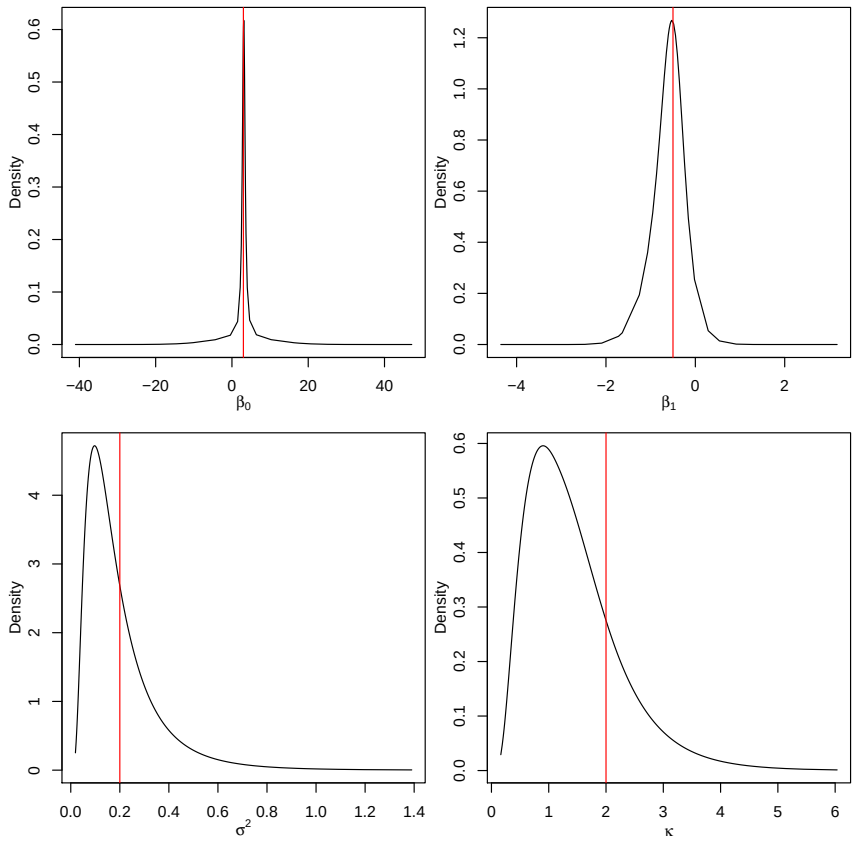

Figure 13.2: Posterior distribution for the intercept (top left), coefficient of the covariate (top right) and the parameters of the log-Cox model σ^2 (bottom left), κ (bottom right)

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

```
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 13.2.

```
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 14

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 12 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
```

14.1 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.

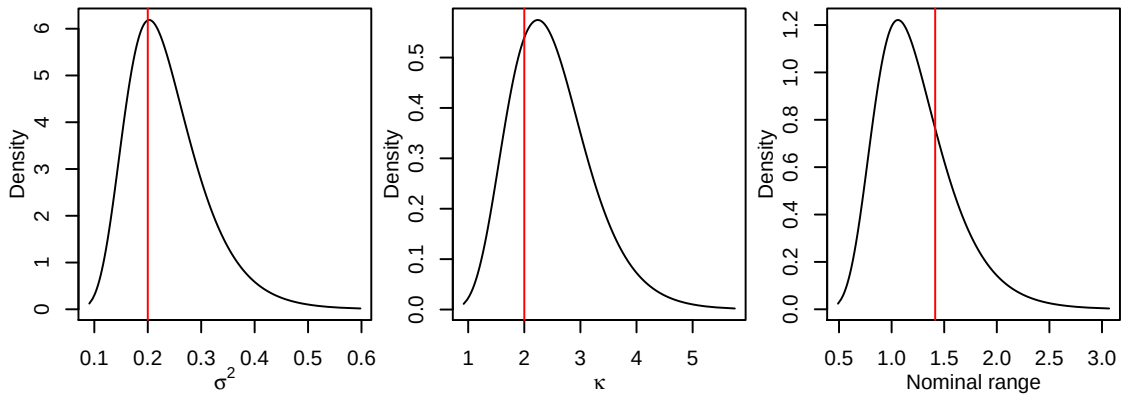

Figure 14.1: Posterior distribution for σ^2, κ and the nominal range just using the response.

```
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)))
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 0.2568     9.4249   9.9627    10.4966  9.9633
prec.y   16 11.1309 1.4337     8.5523  11.0507    14.1771 10.9022
```

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)
```

We can see the marginals posterior distributions for the model parameters in the Figure 14.1, produced with the following code

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

14.2 Model fitting 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

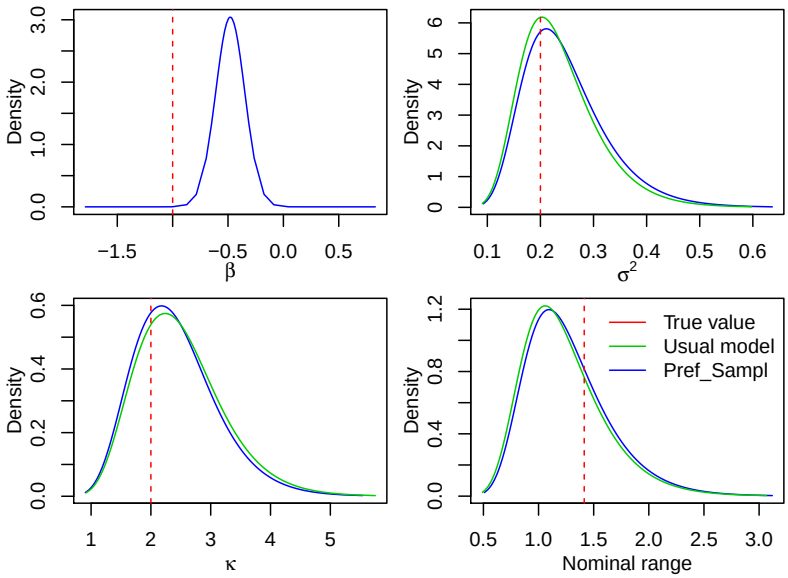

Figure 14.2: Posterior marginal distribution for β_0, σ^2, κ and the nominal range under preferential sampling.

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)))
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.

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

```
      True   mean     sd 0.025quant 0.5quant 0.975quant   mode kld
b0.pp    3 3.0161 0.1541     2.6973   3.0180     3.3255 3.0216   0
b0.y    10 9.9813 0.2739     9.4088   9.9819    10.5497 9.9826   0
```

Computing the marginals posterior distributions in user-scale random field parameters:

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

We can visualize these posterior marginal distributions in Figure 14.2. 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.

Chapter 15

Spatio temporal point process

In this chapter we are going to show how to fit a spatio temporal log-Cox point process model. We use the `burkitt` data set from the **splancs** package.

```
data('burkitt', package='splancs')
t(sapply(burkitt[, 1:3], summary))
```

```
  Min. 1st Qu. Median   Mean 3rd Qu. Max.
x  255   269.0  282.5  286.3   300.2  335
y  247   326.8  344.5  338.8   362.0  399
t  413  2412.0 3704.0 3530.0  4700.0 5775
```

The folowing commands shows the time when each event occurred, Figure 15.1.

```
n <- nrow(burkitt)
par(mfrow=c(1,1), mar=c(3,.1,.1,.1), mgp=c(2,1,0))
plot(burkitt$t, rep(1,n), type='h', xlim=c(0,max(burkitt$t)),
       ylim=0:1, axes=FALSE, xlab='time', ylab='')
box(); axis(1)
```

We have to define a set of knots over time to define the SPDE spatio temporal model. And use it then to built a temporal mesh to project the field into the data time location

```
tknots <- seq(0, 1, 0.2)*max(burkitt$t)
abline(v=tknots, lwd=4, col=4) ## add to plot
k <- (mesh.t <- inla.mesh.1d(tknots))$n
```

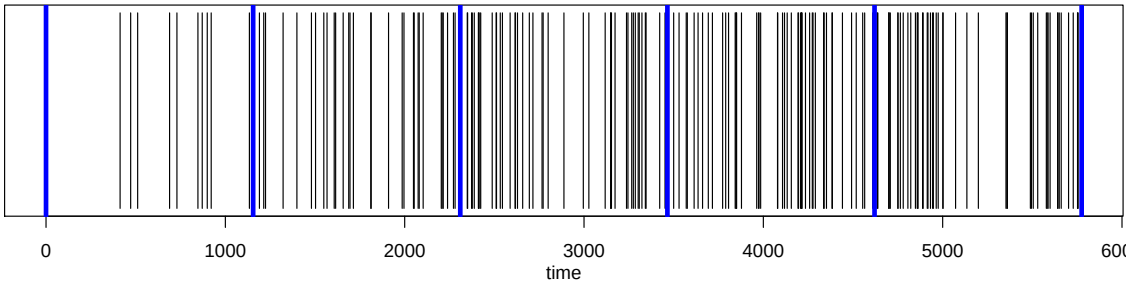

Figure 15.1: Time when each event occurred.

To work with the space, we first define a function to convert a polygon defined from a set of coordinates to an `SpatialPolygons` object:

```
coo2sp <- function(coo) {
      n <- nrow(coo)
```

```
    if (any(coo[1,]!=coo[n,]))
        coo <- coo[c(1:n,1),]
    SpatialPolygons(list(Polygons(list(Polygon(coo)), '0')))
}
```

The spatial mesh can be done using the polygon of the region as a boundary

```
domainSP <- coo2sp(burbdy)
m <- (mesh.s <- inla.mesh.2d(burpts, boundary=inla.sp2segment(domainSP),
                             max.edge=c(15, 50), cutoff=7))$n
```

The spatio temporal projector matrix is made considering both spatial and temporal locations and both meshes, spatial and temporal.

```
dim(Ast <- inla.spde.make.A(mesh=mesh.s, loc=burpts, n.group=length(mesh.t$n),
                            group=burkitt$t, group.mesh=mesh.t))
```

```
[1]  188 1326
```

This matrix has number of columns equals to the number of nodes in the mesh times the number of groups.

The SPDE model is defined using the spatial mesh

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

The index set is made considering the group feature:

```
idx <- inla.spde.make.index('s', spde$n.spde, n.group=mesh.t$n)
```

The data stack can be made considering the ideas for the purerly spatial model. So, we do need to consider the expected number of cases at the 1) integration points and 2) data locations. The first is now the spacetime volume at the mesh node and time knot, which is the area of each Voronoi polygon around the mesh location kronecker the length of the time window at each time point. The second is zero as for a point the expectation is zero, in accord to the likelihood approximation proposed by [Simpson et al., 2011].

We first need to build the Voronoi polygons. We choose to use a function from the **deldir** package:

```
require(deldir)
dd <- deldir(mesh.s$loc[,1], mesh.s$loc[,2])
tiles <- tile.list(dd)
```

The area of the intersection between each Voronoi polygons and the domain region can be done using the `gIntersects` function from **rgeos** package. To use it we define a function to convert each Voronoi polygon into a `SpatialPolygons` and use the `gIntersects` function. It is done in the code below (the sum of the intersection polygons areas is showed at end):

```
require(rgeos)
sum(w <- sapply(tiles, function(p) {
                    pl <- coo2sp(cbind(p$x, p$y))
                    if (gIntersects(pl, domainSP))
                        return(gArea(gIntersection(pl, domainSP)))
                    else return(0)
                }))
```

```
[1] 11035.01
```

We can see that it sum up the same as the domain area:

```
gArea(domainSP)
```

```
[1] 11035.01
```

So, we have computed the area of the intersection between each Voronoi polygon (one for each location of the spatial mesh) and the study domain. The spatial mesh vertices far out the study domain will have no intersection. The spatio temporal volume is the product of these values and the time window length of each time knot.

```
st.vol <- rep(w, k) * rep(diag(inla.mesh.fem(mesh.t)$c0), m)
```

The data stack is built using

```
y <- rep(0:1, c(k * m, n))
expected <- c(st.vol, rep(0, n))
stk <- inla.stack(data=list(y=y, expect=expected),
      A=list(rBind(Diagonal(n=k*m), Ast), 1),
      effects=list(idx, list(a0=rep(1, k*m + n))))
```

Model fitting (using the cruder approximation: 'gaussian')

```
form <- y ~ 0 + a0 +
      f(s, model=spde, group=s.group, control.group=list(model='ar1'))
burk.res <- inla(form, family='poisson',
                  data=inla.stack.data(stk), E=expect,
                  control.predictor=list(A=inla.stack.A(stk)),
                  control.inla=list(strategy='gaussian'))
```

The exponential of the intercept plus the random effect at each spacetime integration point is the relative risk at each these points. This relative risk times the spacetime volume will give the expected number of points at each these spacetime locations. Summing it will approaches the number of observations:

```
eta.at.integration.points <- burk.res$summary.fix[1,1] + burk.res$summary.ran$s$mean
c(n=n, 'E(n)'=sum(st.vol*exp(eta.at.integration.points)))
```

```
       n     E(n)
188.0000 187.1864
```

The projection over a grid for each time knot can be done with

```
r0 <- diff(range(burbdy[,1]))/diff(range(burbdy[,2]))
prj <- inla.mesh.projector(mesh.s, xlim=range(burbdy[,1]),
                           ylim=range(burbdy[,2]), dims=c(100, 100/r0))
ov <- over(SpatialPoints(prj$lattice$loc), domainSP)
m.prj <- lapply(1:k, function(j) {
    r <- inla.mesh.project(prj, burk.res$summary.ran$s$mean[1:m+(j-1)*m])
    r[is.na(ov)] <- NA;   return(r)
  })
```

The fitted latent field at each time knot is in Figure 15.2, produced with the code below. It can also be done for the standard deviation.

```
igr <- apply(abs(outer(burkitt$t, mesh.t$loc, '-')), 1, which.min)
zlm <- range(unlist(m.prj), na.rm=TRUE)
par(mfrow=c(2,3), mar=c(0,0,0,0))
for (j in 1:k) {
      image(x=prj$x, y=prj$y, z=m.prj[[j]], asp=1,
            xlab='', zlim=zlm, axes=FALSE, col=tim.colors(64))
      points(burkitt[igr==j, 1:2], pch=19)
  }; image.plot(legend.only=TRUE, zlim=zlm, legend.mar=5)
```

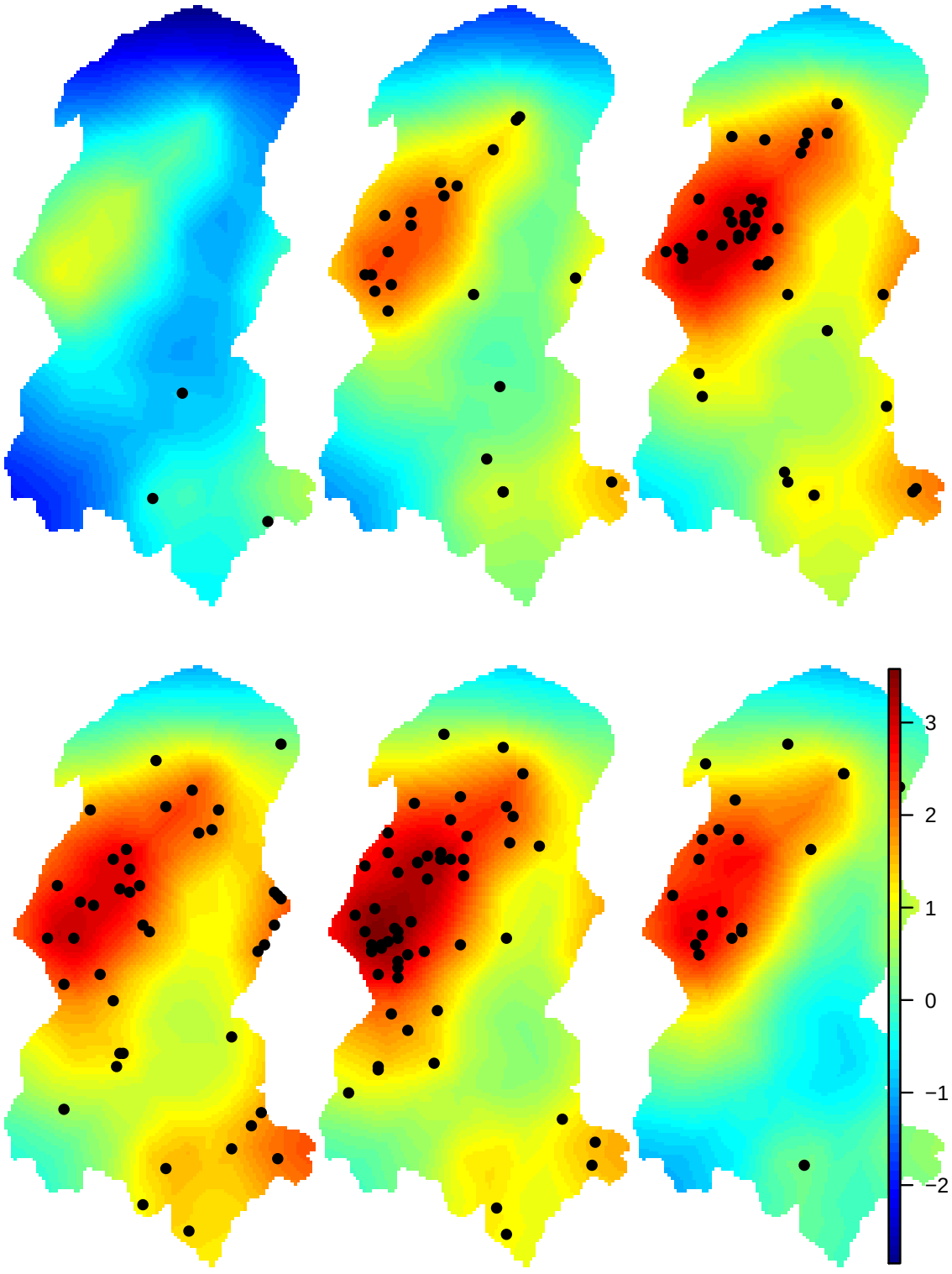

Figure 15.2: Fitted latent field at each time knot overlayed by the points closer in time.

Chapter 16

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

We use the toy data set and built the mesh and SPDE model as follows:

```
data(SPDEtoy);    coords <- as.matrix(SPDEtoy[,1:2])
mesh <- inla.mesh.2d(coords, cutoff=0.05, max.edge=1:2/10)
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.

The following code loops on the number of 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))
  }
```

The posterior marginal distributions for the marginal variance and practical range:

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

Collecting the results of the posterior mean and posterior variance of each parameter:

```
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]])
```

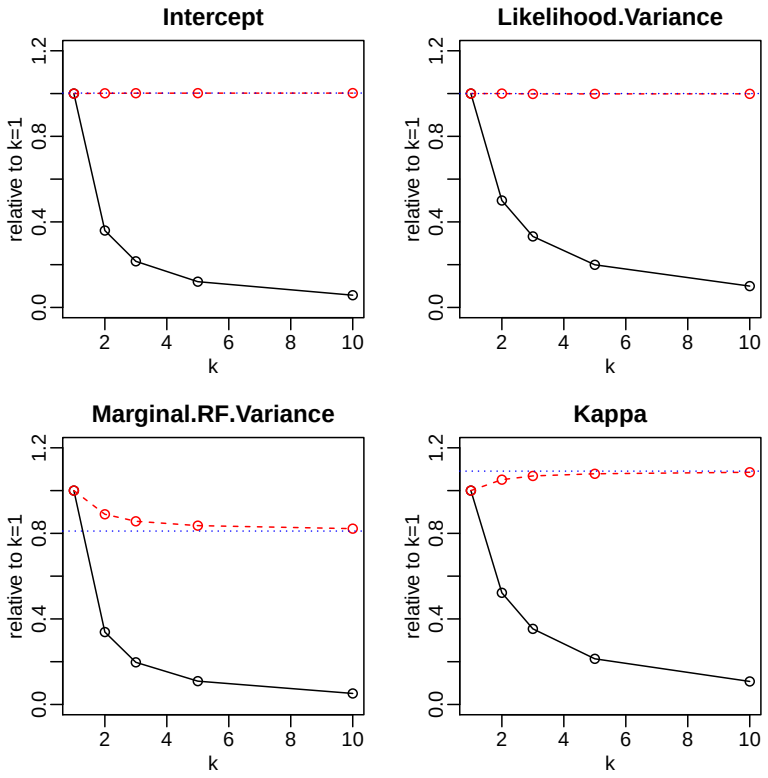

Figure 16.1: Likelihood estimate (green), posterior mean (red) and posterior variance (black), relatives to the value with $k = 1$.

```
    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. These plots are shown in Figure 16.1 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)
  }
```

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.

[Blangiardo and Cameletti, 2015] Blangiardo, M. and Cameletti, M. (2015). *Spatial and Spatio-Temporal Bayesian models with R-INLA*. Wiley.

[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.

[Henderson et al., 2003] Henderson, R., Shimakura, S., and Gorst, D. (2003). Modeling spatial variation in leukemia survival data. *JASA*, 97(460):965–972.

[Illian et al., 2012] Illian, J. B., Sørbye, 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.

[Ingebrigtsen et al., 2014] Ingebrigtsen, R., Lindgren, F., and Steinsland, I. (2014). Spatial models with explanatory variables in the dependence structure. *Spatial Statistics*, 8:20–38.

[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.

[Schimdt and Gelfand, 2003] Schimdt, A. M. and Gelfand, A. E. (2003). A bayesian coregionalization approach for multivariate pollutant data. *Journal of Geographysical Research*, 108(D24).

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

[Simspon et al., 2015] Simspon, D. P., Martins, T. G., Riebler, A., Fuglstad, G.-A., Rue, H., and Sørbye, S. H. (2015). Penalising model component complexity: A principled, practical approach to constructing priors. *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.