The R-INLA tutorial on SPDE models

Warning: work in progress...

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

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

November 20, 2017

Abstract

This tutorial will show you how to fit models that contains at least one effect specified from an SPDE using the 'R-INLA'. Up to now, it can an SPDE based model can be applied to model random effects over continuous one- or two- dimensional domains. However, the theory works for higher dimensional cases. The usual application is data whose geographical location is explicitly considered in the analysis. This tutorial explores 'R-INLA' functionalities by using examples. It starts with simple models and increases in complexity.

In Chapter 1 includes a section introducing the random field models and the Matérn class. We illustrate some features of this class in figures. The we introduce the main results in [Lindgren et al., 2011] intuitively linking to images of the matrices involved being computed for a illustrative small case with few spatial locations. We show how to fit a geostatistical model for a simulated data, the toy example, covering from the mesh building, model definition, data preparation, showing results, doing predictions and considering results from different meshes. We also show how to build a mesh considering non-convex domains, spatial polygons objects and domains with holes or physical boundaries.

In Chapter 2 we consider three examples. We consider the daily average rainfall from rainfall collected at 616 gauge stations in the Paraná state in Brasil over year 2011. For this data we show a detailed analysis including code to compute geographical covariates, smoothed regression an prediction. The second example in this Chapter consider survival analysis for the Leukaemia dataset, analysed in [Henderson et al., 2003]. We show how to consider the parametric Weibull case and also the non-parametric Cox proportional hazard case. For this case we have the implementation internally considers a new structure of the data in order to perform a Poisson regression. The last example in this Chapter considers simulated data to illustrate the approach of modeling the SPDE model parameters by a regression which is the case of having covariates in the covariance, proposed in [Ingebrigtsen et al., 2014].

In Chapter 3 we have a collection of examples were copy random fields to model two or more outcomes jointly. It includes a measurement error model in order to account for spatially structured measurement error in a covariate. A coregionalization model consider the case for three outcomes were the fist outcome is in the linear predictor for the second one and both are in the predictor for the third outcome, as proposed in [Schimdt and Gelfand, 2003]. An example considering copying a part or the entire linear predictor from one outcome in a linear predictor to another one ends this chapter. It shows a slight different way from the coregionalization model to jointly model three outcomes.

The log Cox point process model is considered in Chapter 4. In this case we show how to fit a Log-cox point process using the direct approximation for the likelihood as proposed in [Simpson et al., 2016]. We also take the opportunity to show how to consider the joint modeling of the process and the locations, under the preferential sampling as proposed in [Diggle et al., 2010].

Finally, Chapter 5 presents several cases to example analysis of space-time data. We start by an example having discrete time domain as in [Cameletti et al., 2012]. We extend this example considering the time as continuous by considering time knots along with temporal function basis functions for projection. We also extend the coregionalization example for the space-time in this Chapter. The space-time model is also applied for modeling regression coefficients in a dynamic regression example having the regression coefficients varying over space-time. We consider the space-time version of the log-Cox point process model for a dataset and also illustrates an approach to deal whit the case of having a large space-time point process data.

Since this tutorial is more a collection of examples, one should start with the tutorial marked as **Read this first!** at the tutorials link in the R-INLA web page, `http://www.r-inla.org`, more precisely at `http://www.r-inla.org/examples/tutorials/spde-tutorial-from-jss`. If you are in a rush to fit a simple geostatistical model, we made a short tutorial without the details as a vignette in the **INLA**. Thus one can have it just typing `vignette(SPDEhowto)` for a two dimensional example or `vignette(SPDE1d)` for a one dimensional example. We built a Shiny application to help one to understand the mesh building. It depends on the **shiny** package. This application opens by typing `demo(mesh2d)`.

Acknowledgments To Sarah Gallup and Helen Sofaer for valuable English review in the text. To several people who brought cool problems to the discussion forum `http://www.r-inla.org/comments-1` and to me individually.

Updates

```
2017 Oct 05: English review (thanks to SG)
2017 Mar 01: revision and new organization
2017 Feb 15: started a major revision/organization
2017 Jan 23: compile to html and small fixes
2016 May 17: fix names in copy lin. pred. example (Thx to MC) and improve text
2016 May 09: copy linear predictor example
2016 Mar 22: tiny fix of survival ex.: cor(log(sd), ...). Thx HR
2016 Feb 26: space-time (small fix) and non-stationary (s. improv.)
2016 Feb 03: small changes in dynamic and space-time simulation
2016 Feb 01: tiny fix in the rainfall example model description
2015 Dec 30: measurement error example added
2015 Sep 25: small fixes, improve dynamic example
2015 Sep 24: 1) include hyperref package. 2) improve dynamic example
2015 Sep 23: small fix on likelihood equations,
  space-time with continuous time and dynamic regression model example
2015 Aug 31: aproach for large spacetime point process
2015 Aug 26: fix the spacetime 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

1.1 Introduction and main results from [Lindgren et al., 2011]

The R source code for this introduction is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-introduction.R`

1.1.1 Spatial variation

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 longitude and latitude. Point-referenced data are common in many areas of science. This type of data appears in mining, climate modeling, ecology, agriculture and elsewhere. If we want to incorporate the influence of location in a model for the data we need a model for geo-referenced data.

A regression model can be built using the location's coordinates as a covariates. In some cases it will be necessary a complicated function based on the coordinates to adequately describe the effect of the location. For example, we can consider basis functions on the coordinates and use it as covariates in order to build a complex function. This model explictly the trend on the mean.

Alternatively, it may be more natural to model explictly the variations of the outcome considering that it maybe similar at nearby locations. The first law of geography asserts: "Everything is related to everything else, but near things are more related than distant things", [Tobler, 1970]. 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 observation that is collected from farther away. A spatially-structured random effects model incorporates such spatial dependency rather than simply one consistent spatial trend across a domain. Sometimes it is appropriate to include both terms in a model.

The models that accounts for spatial dependency may be defined depending whether the locations are areas (cities for example) or points. The latter case is usually divided when considering the locations as fixed (geostatistics or point-referenced data) or random (point process), [Cressie, 1993].

1.1.2 The Gaussian random field

To introduce some notation, let s be any location in a study area and let $U(s)$ be the random (spatial) effect at that location. $U(s)$ is a stochastic process, with $s \in \boldsymbol{D}$, where $\boldsymbol{D}$ is the domain of the study area and $\boldsymbol{D} \in \Re^d$. Suppose, for example, that $\boldsymbol{D}$ is one 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 $u(\mathbf{s}_i)$, $i = 1, 2, ..., n$, a realization of $U(s)$ in n locations. It is commonly assumed that $u(s)$ has a multivariate Gaussian distribution. If we assume that $U(s)$ is continuous over space, we have a continuously-indexed Gaussian field (GF). This implies that it is possible to collect these data at any finite set

of locations within the study region. To complete the specification of the distribution of $u(s)$, it is necessary to define its mean and covariance.

A very simple option is to define a correlation function based only on the Euclidean distance between locations. This assumes that if we have two pairs of points separated by the same distance h, both pairs have same degree of correlation. It is intuitive to choose a function that decrease with distance, h. [Abrahamsen, 1997] presents Gaussian random fields and correlation functions.

Now suppose that we have data y_i observed at locations $\mathbf{s}_i$, $i = 1, ..., n$. If an underlying GF generated these data, we can fit the parameters of this process considering $y(\mathbf{s}_i) = u(\mathbf{s}_i)$, where the observation $y(\mathbf{s}_i)$ is assumed to be a realization of the GF at the location $\mathbf{s}_i$. If we assume $y(\mathbf{s}_i) = \mu + u(\mathbf{s}_i)$, we have one more parameter to estimate. It is worth mentioning that the distribution for $u(s)$ at a finite number of points is considered a realization of a multivariate distribution. In this case, the likelihood function is the multivariate distribution with covariance Σ.

In many situations we assume that there is an underlying GF but we cannot observe direclty. Instead we observe data with a measurement error

$$y(\mathbf{s}_i) = u(\mathbf{s}_i) + e_i. \tag{1.1}$$

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 remaining noise and is called the nugget effect. The covariance of the marginal distribution of $y(s)$ at a finite number of locations is $\Sigma_y = \Sigma + \sigma_e^2\mathbf{I}$. This is a short extension of the basic GF model, and gives one additional parameter to estimate. For more about this model see [Diggle and Ribeiro Jr, 2007].

To usual way to evaluate the likelihood function, which is just a multivariate Gaussian density for the model in Eq. 1.1, usually considers a Cholesky factorization of the covariance matrix. Because this matrix is a dense, this is a operation of order $O(n^3)$, so this is a 'big n problem'. Some software for geostatistical analysis uses an empirical variogram to fit the parameters of the correlation function. However, this option does not make any assumption about a likelihood function for the data or uses a multivariate Gaussian distribution for the spatially structured random effect. A good description of these techniques is available in [Cressie, 1993].

When extending the model to deal with non-Gaussian data, it is usual to assume a likelihood for the data conditional on an unobserved random effect, which is GF to model the spatial dependence. Such spatial mixed effects model is under the the model based geostatistics approach, citediggleribeiro:2007. It is possible to describe the model in Eq. 1.1 within a larger class of models, hierarchical models. Suppose that we have observations y_i for locations $\mathbf{s}_i$, $i = 1, ..., n$. We start with

$$\begin{gathered} y_i|\theta, \beta, u_i, \mathbf{F}_i \sim P(y_i|\mu_i, \phi) \\ \mathbf{u} \sim GF(0, \Sigma) \end{gathered} \tag{1.2}$$

where $\mu_i = h(\mathbf{F}_i^T\beta + u_i)$, $\mathbf{F}$ is a matrix of covariates, $\mathbf{u}$ is the random effect, θ are parameters for the random effect, β are covariate coefficients, $h()$ is a function mapping the linear predictor $\mathbf{F}_i^T\beta + u_i$ to $\mathrm{E}(y_i) = \mu_i$ 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 $\mathbf{F}_i^T\beta$, assume a Gaussian distribution for y_i, with variance σ_e^2 and $\mathbf{u}$ as a GF. However, at times one does consider a multivariate Gaussian distribution for the random effect, it becomes impractical to use covariance directly for model-based inference.

In another area of 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) and a good reference is [Rue and Held, 2005]. It is computationally easier to make Bayesian inference when we use a GMRF than when we use a GF, because the cost in computation of working with

a sparse precision matrix in GMRF models is $O(n^{3/2})$. This makes it easier to conduct analyses with big 'n'.

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

1.1.3 The Matérn covariance

A very popular correlation function is the Matérn correlation function. It has a scale parameter $\kappa > 0$ and a smoothness parameter $\nu > 0$. For two locations $\mathbf{s}_i$ and $\mathbf{s}_j$, the stationary and isotropic Matérn correlation function is:

$$Cor_M(U(\mathbf{s}_i), U(\mathbf{s}_j)) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\kappa \parallel \mathbf{s}_i - \mathbf{s}_j \parallel)^{\nu} K_{\nu}(\kappa \parallel \mathbf{s}_i - \mathbf{s}_j \parallel) \tag{1.3}$$

where $\parallel . \parallel$ denotes the Euclidean distance and K_ν is the modified Bessel function of the second order. The Matérn covariance function is $\sigma_u Cor(U(\mathbf{s}_i), U(\mathbf{s}_j))$, where σ_u is the marginal variance of the process.

If we have a realization $u(s)$ from $U(s)$ at n locations, $\mathbf{s}_1, ..., \mathbf{s}_n$, we can define its joint covariance matrix. Each entry of this joint covariance matrix Σ is $\Sigma_{i,j} = \sigma_u Cor_M(u(\mathbf{s}_i), u(\mathbf{s}_j))$. It is common to assume that $U(.)$ has a zero mean. We have now completely defined a multivariate distribution for $u(s)$.

To gain a better feel about the Matérn correlation we can drawn samples from the RF process and look at it. A sample $\mathbf{u}$ is drawn considering $\mathbf{u} = \mathbf{Lz}$ where $\mathbf{L}$ is the Cholesky decomposition of the covariance at n locations and z is a vector with n samples drawn from a standard Gaussian distribution. It implies that $\mathrm{E}(\mathbf{u}) = \mathrm{E}(\mathbf{Lz}) = \mathbf{L}\mathrm{E}(\mathbf{z}) = 0$ and $Var(\mathbf{u}) = \mathbf{L}'\mathrm{Var}(\mathbf{z})\mathbf{L} = \mathbf{L}'\mathbf{L}$. We define functions to do the sampling bellow

```
cMatern <- function(h, nu, kappa) ### Matern correlation
    besselK(h * kappa, nu) *
        (h*kappa)^nu / (gamma(nu) * 2^(nu-1))
### function to sample from zero mean multivariate normal
rmvnorm0 <- function(n, cov, L=NULL) {
    if (is.null(L)) L <- chol(cov)
    return(crossprod(L, matrix(rnorm(n*ncol(L)), ncol(L))))
}
```

These steps are collected into the function `rMatern` that is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`

In order to simplify the visualization of the properties, we consider a set of $n = 249$ locations in the one-dimensional space from 0 to 25.

```
### define locations and distance matrix
loc <- 1:249/25
mdist <- as.matrix(dist(loc))
```

We consider four values for the smoothness parameter ν. The values for the κ parameter was determined from the practical range expression $\sqrt{8\nu}/\kappa$, which is the distance that gives correlation near 0.13. By combining the four values for ν with two values for the practical range we have eight parameter configurations.

```
### define parameters
nu <- c(0.5,1,2,5)
pract.range = c(1,4)
kappa <- c(sqrt(8*nu)/pract.range[1],
```

```
           sqrt(8*nu)/pract.range[2])
### covariance parameter scenarios
params <- cbind(nu=rep(nu, length(pract.range)),
                kappa=kappa,
                r=rep(pract.range, each=length(nu)))
```

The sample value depends on the covariance matrix and on the noise considered from the standard Gaussian distribution, z. We consider a set of five vectors of size n drawn from the standard Gaussian distribution. These five standard Gaussian were the same among the eight parameter configurations in order to keep track what the different parameter configuration are doing.

```
### sample error
set.seed(123)
z <- matrix(rnorm(nrow(mdist)*5), ncol=5)
```

Therefore, we have a set of 40 different realizations, five for each parameter configuration

```
### compute the correlated samples
yy <- lapply(1:nrow(params), function(j) { ## scenarios
    v <- cMatern(mdist, params[j,1], params[j,2])
    diag(v) <- 1 + 1e-10
    return(list(params=params[j,], ### parameter scenario
                y=crossprod(chol(v), z))) ### compute sample
})
```

These samples are shown in the eight plots in Figure 1.1.3.

```
### visualize
(ry <- range(unlist(lapply(yy, tail, 1))))
par(mfcol=c(4,2), mar=c(2,2,1,.1), mgp=c(1.5,0.7,0), las=1)
for (i in 1:length(yy)) { ### each scenario
    plot(loc, yy[[i]]$y[,1], ylim=ry,
         xlab='', ylab='', type='n',
         main=as.expression(bquote(paste(
             nu==.(yy[[i]]$params[1]), ', ',
             kappa==.(round(yy[[i]]$params[2],2)), ', ',
             r==.(yy[[i]]$params[3])))))
    for (k in 1:5)
        lines(loc, yy[[i]]$y[,k], col=k) ### each sample
}
```

One important point to observe in Figure 1.1.3 is the main feature on the samples. It almost does not depends on the smoothness parameter. To see it consider one of the five samples (one of the colors) and compare it for different smoothness. Also, image if we add a noise on a smooth process, it will becomes hard them to distinguish what is due to noise from what is due to smoothness. Therefore, in practice we usually fix the smoothness parameter and add a noise term.

1.1.4 Simulation of a toy data set

We will now draw a sample from the model in Eq. 1.1 and use in Section 1.2. We consider a set of $n = 100$ locations within a square of area one and bottom left and top right limits

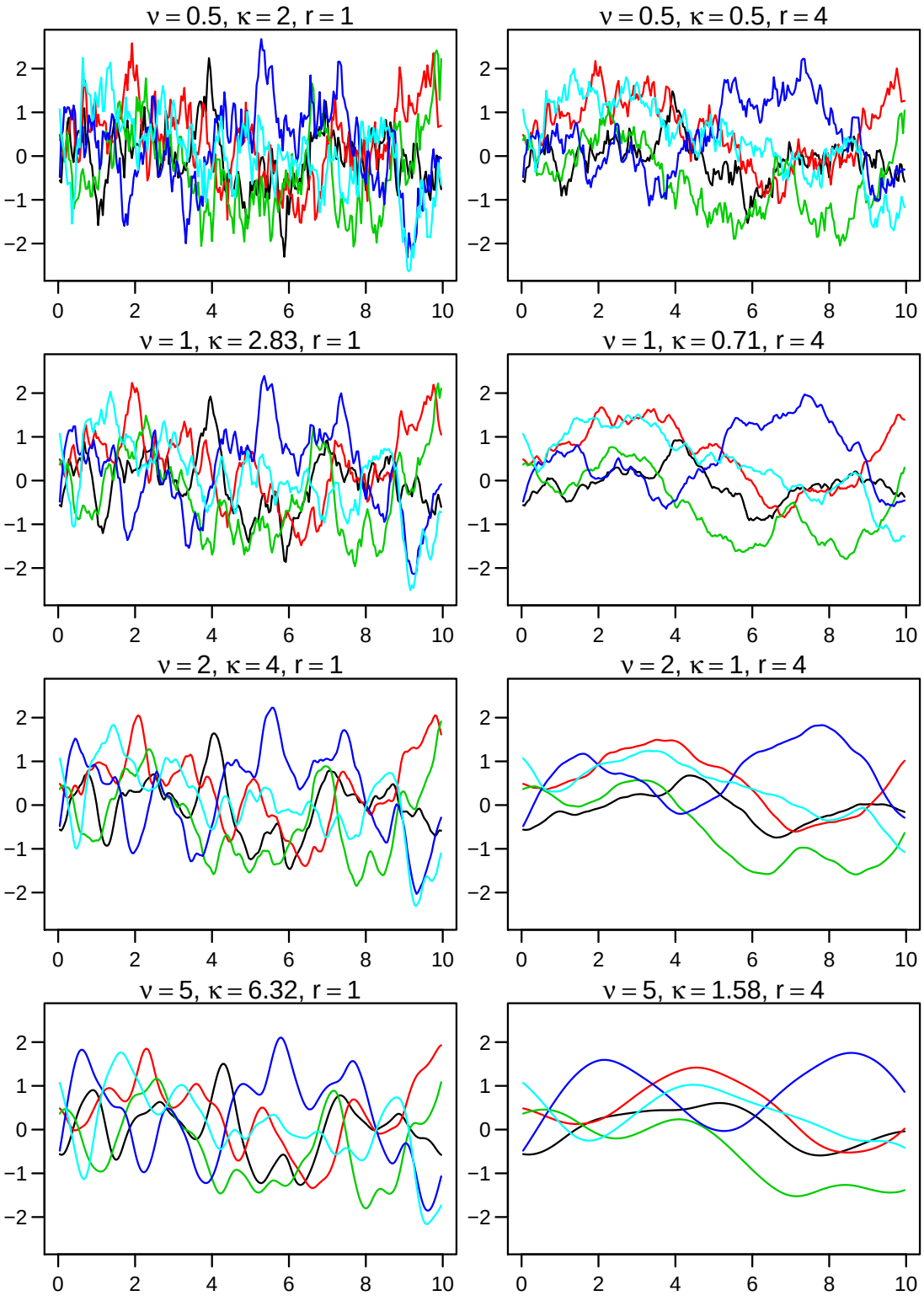

Figure 1.1: Five samples from the one-dimensional Matérn correlation function for two different ranges (each column of plots) and four different value for the smoothness parameter (each line of plots).

of (0,0) and (1,1). We choose a higher density of locations 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)
```

To get a (lower triangle) matrix of distances we call

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

We choose parameter values for the Matérn as $\sigma_u^2 = 5$, $\kappa = 7$ and $\nu = 1$. We define the mean $\beta_0 = 10$ and the nugget parameter $\sigma_e^2 = 0.3$. We declare values for these parameters using

```
beta0 <- 10; sigma2e <- 0.3; sigma2u <- 5; kappa <- 7; nu <- 1
```

Now we need to sample from a multivariate distribution with constant mean equals β_0 and covariance $\sigma_e^2 I + \Sigma$, which is the marginal covariance of the observations.

```
mcor <- as.matrix(2^(1-nu)*(kappa*dmat)^nu *
                  besselK(dmat*kappa,nu)/gamma(nu))
diag(mcor) <- 1;   mcov <- sigma2e*diag(n) + sigma2u*mcor
```

We can now sample considering the Cholesky factor times a unit variance noise and add the mean

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

We show these simulated data in a graph of the locations where the size of the points is proportional to the simulated values. Figure 1.1.4 was 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)
```

This data will be used as a toy example in this tutorial. It is available in the `R-INLA` package and can be loaded by

```
data(SPDEtoy)
```

1.1.5 The SPDE approach

In this section we summarizes the main results in [Lindgren et al., 2011]. If your purpose does not includes understanding the methodology, you can skip this section. If you keep reading this section and have difficulty, do not be discouraged. You may still be able to use INLA effectively even if you have only a limited grasp of what's 'under the hood.'

[Rue and Tjelmeland, 2002] proposed to approximate a continuous field using a Gaussian Markov Random Field - GMRF. This idea is not so strange since there are continuous random fields that are Markov. This is the case when the continuous field is a solution of a linear stochastic partial differential equation, see [Rozanov, 1977]. [Lindgren et al., 2011]

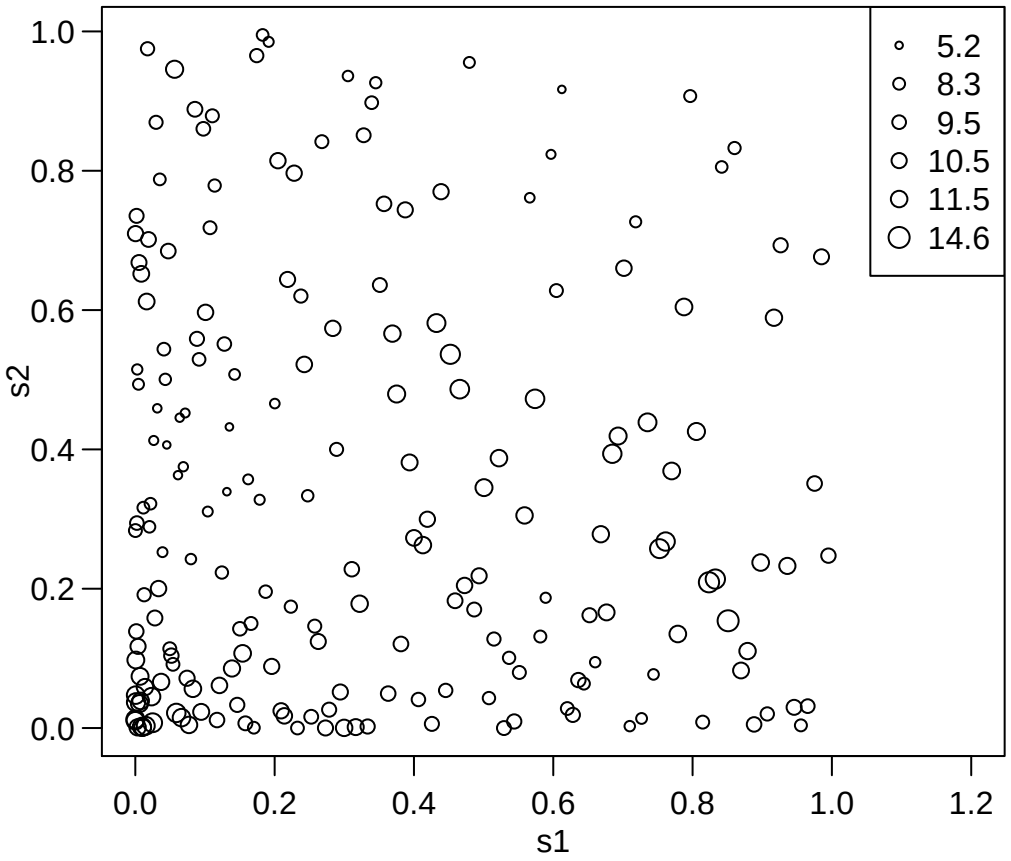

Figure 1.2: The simulated toy example data.

provided that the GRF with Matérn correlation is a solution for an stochastic partial differential equation (SPDE).

If you want to get an intuition about the approach, we tried to provide it in this section. However, if you want to know all the details, they are in the Appendix of [Lindgren et al., 2011]. In few words it uses the Finite Element Method (FEM) along with basis functions carefully chosen to preserve the sparse structure of the resulting precision matrix for the random field at a set of mesh nodes. This provides an explicit link between a continuous random field and a GMRF representation, which allows efficient computations.

First result

Lindgren's first main provided that a GF with a generalized covariance function, obtained when $\nu \to 0$ in the Matérn correlation function, is a solution of a SPDE. This extends the result obtained by [Besag, 1981]. A more statistical way of considering this result is when taking a regular two-dimensional lattice with number of sites tending to infinity. In this case the full conditional at the site ij has

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

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

$$\begin{array}{|cc} \text{-1} & \\ a & \text{-1} \end{array} \tag{1.5}$$

A GF $U(s)$ with Matérn covariance is a solution to the following linear fractional SPDE

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

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

$$\begin{array}{|ccc} 1 & & \\ \text{-2}a & 2 & \\ 4{+}a^2 & \text{-2}a & 1 \end{array} \tag{1.7}$$

and, for $\nu = 2$:

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

Interpreting the result intuitively, as the smoothness parameter ν increases the precision matrix in the GMRF representation becomes less sparse. Greater density of the matrix is because the conditional distributions depend on a wider neighborhood. The precision matrix for $\alpha = 2$, $\boldsymbol{Q}_2 = \boldsymbol{Q}_1 \boldsymbol{C}^{-1} \boldsymbol{Q}_1$, is a standardized square of the precision matrix for $\alpha = 1$, $\boldsymbol{Q}_1$.

However, the denser matrix does not imply that the conditional mean is an average over a wider neighborhood. The conceptual parallel is going from a first order random walk to a second order one. To understand this point let us consider the precision matrix for the first order random walk, its square and the precision matrix for the second order random walk.

```
(q1 <- INLA:::inla.rw1(n=5))

## 5 x 5 sparse Matrix of class "dgTMatrix"
##
## [1,]  1 -1  .  .  .
## [2,] -1  2 -1  .  .
## [3,]  . -1  2 -1  .
## [4,]  .  . -1  2 -1
## [5,]  .  .  . -1  1

crossprod(q1) ### same inner pattern as for RW2

## 5 x 5 sparse Matrix of class "dsCMatrix"
##
## [1,]  2 -3  1  .  .
## [2,] -3  6 -4  1  .
## [3,]  1 -4  6 -4  1
## [4,]  .  1 -4  6 -3
## [5,]  .  .  1 -3  2

INLA:::inla.rw2(n=5)

## 5 x 5 sparse Matrix of class "dgTMatrix"
##
## [1,]  1 -2  1  .  .
## [2,] -2  5 -4  1  .
## [3,]  1 -4  6 -4  1
## [4,]  .  1 -4  5 -2
## [5,]  .  .  1 -2  1
```

We can see that the difference is only in the corners.

Second result

It is common point data are not located on a grid, but instead are distributed irregularly. [Lindgren et al., 2011] provide a second set of results that provide a solution for the case irregular grids. They use the finite element method technique (FEM), a tool that is used widely in engineering and applied mathematics to solve differential equations.

The domain can be divided into a set of non-intersecting triangles, which may be irregular, where any two triangles meet in at most a common edge or corner. The three corners of a triangle are named vertices or nodes. The solution for the SPDE and its properties will depend on the basis functions used. [Lindgren et al., 2011] choose basis functions carefully in order to preserve the sparse structure of the resulting precision matrix.

The approximation is

$$u(\mathbf{s}) = \sum_{k=1}^{m} \psi_k(\mathbf{s}) w_k$$

where ψ_k are basis functions, w_k are Gaussian distributed weights, $k = 1, ..., m$ with m the number of vertices in the triangulation. Because ψ_k is piecewise linear within each triangle, with ψ_k is equal to 1 at vertices k and 0 at all other vertices, we w_k is the value of the field at the vertex k. A stochastic weak solution was considered to show that the joint distribution for the weights determines the full distribution in the continuous domain. These weights can be interpolated for any point inside the triangulated domain.

We will now focus on the resulting precision matrix. It does consider the triangulation and the basis functions. It matches the first result when applying for a regular grid. Consider the set of $m \times m$ matrices $\boldsymbol{C}$, $\boldsymbol{G}$ and $\boldsymbol{K}_\kappa$ with entries

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

The precision matrix $\boldsymbol{Q}_{\alpha,\kappa}$ as a function of κ^2 and α can be written as

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

The actual $\boldsymbol{K}_\kappa$ matrix consider

$$\tilde{\boldsymbol{C}}_{i,j} = \langle \psi_i, 1 \rangle$$

instead, which is common when working with FEM. Since $\tilde{\boldsymbol{C}}$ is diagonal $\boldsymbol{K}_\kappa$ is as sparse as $\boldsymbol{G}$.

The projection of the weight for any location inside the mesh domain considers a linear interpolation in 2D. It uses the barycentric coordinates of the point with respect to the coordinates of the triangle vertices. For this particular case they are known also as areal coordinates. When a point is inside a triangle we have three non-zero values in the corresponding line of $\boldsymbol{A}$. When it is along an edge, we have two non-zeros and when the point is on top of a triangle vertex we have only one non-zero which equals one.

The following code creates a set of six points, builds a mesh around them and extracts the FEM matrices (**C**, **G** and **A**):

```
s <- 3 ### this factor will only changes C, not G
pts <- rbind(c(1,1), c(2,1),
             c(2.6, 1), c(0.7,1.7), 4:5/3, c(2,1.7))*s
n <- nrow(pts)
mesh0 <- inla.mesh.2d(pts[1:2,], max.edge=3*s,
                      offset=1*s, n=6, cutoff=s*1/2)
mesh <- inla.mesh.2d(rbind(c(3.3,1)*s, c(2.4,2)*s,
                           mesh0$loc[-c(3:4),1:2]),
                     max.edge=3*s, offset=1e-5, cutoff=s*1/2, n=100)
(m <- mesh$n)
dmesh <- inla.mesh.dual(mesh)
fem <- inla.mesh.fem(mesh, order=1)
A <- inla.spde.make.A(mesh, pts)
```

We can gain intuition about this result by considering the structure of each matrix which is detailed in Appendix A.2 in [Lindgren et al., 2011]. It may be easier to understand it by considering the plots in Figure 1.1.5. In this figure we have a mesh with 8 nodes, shown in thicker border lines. The corresponding dual mesh form a collection of polygons around each mesh vertex.

```
par(mfrow=c(1,3), mar=c(2,2,1,1))
plot(mesh, asp=1, lwd=2, edge.color=1)
box(); axis(1); axis(2)
points(mesh$loc, cex=3)
text(mesh$loc[,1]-rep(c(0,0.1),c(m-1,1))*s,
     mesh$loc[,2]+.2*s, 1:m, cex=2)

plot(dmesh, asp=1, lwd=2, main='Dual mesh overlayed')
plot(mesh, add=TRUE)
```

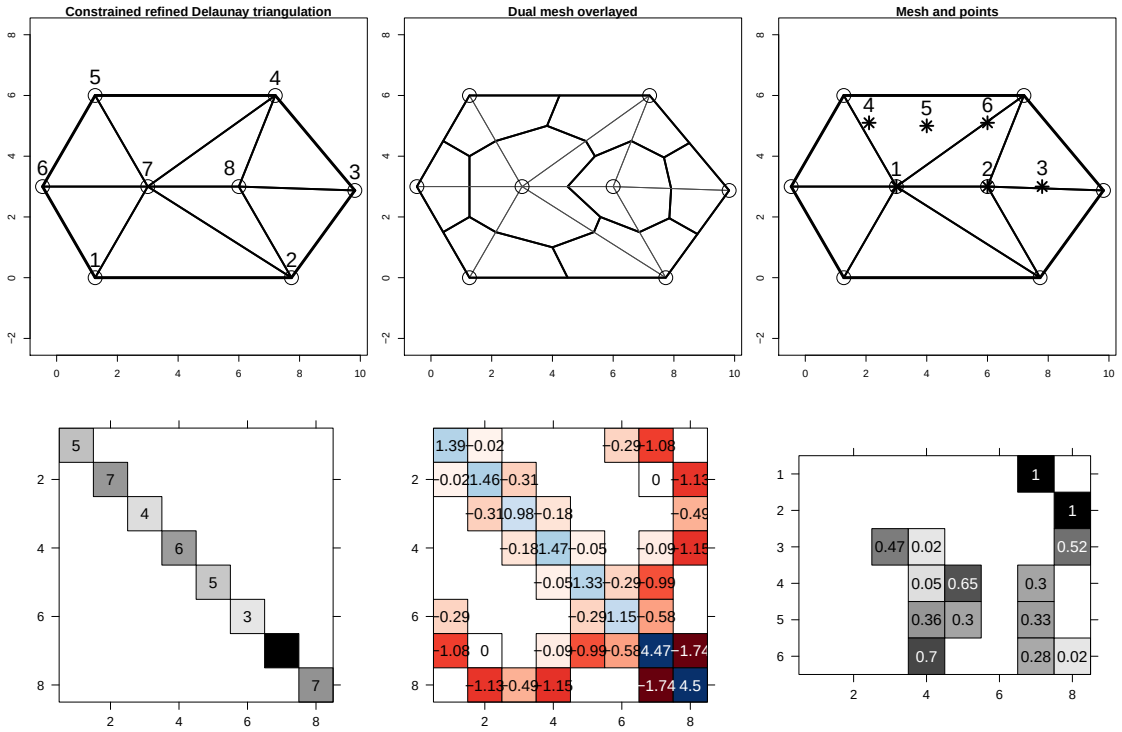

Figure 1.3: A mesh and its nodes identified (top left), the dual mesh polygons (top mid) and the mesh with some points identified (top right). The associated $\boldsymbol{C}$ matrix (bottom left), $\boldsymbol{G}$ matrix (bottom mid) and $\boldsymbol{A}$ matrix (bottom right).

```
box(); axis(1); axis(2)
points(mesh$loc, cex=3)

plot(mesh, asp=1, lwd=2, edge.color=1, main='')
title(main='Mesh and points')
box(); axis(1); axis(2)
points(mesh$loc, cex=3)
points(pts, pch=8, cex=2, lwd=2)
text(pts[,1], pts[,2]+0.2*s, 1:n, cex=2)

A <- as(Matrix(round(as.matrix(A), 10)), ### force zero as zero
        'dgTMatrix')
cc <- as(fem$c0, 'dgTMatrix')
gg <- as(fem$g1, 'dgTMatrix')
library(gridExtra)
library(latticeExtra)
grid.arrange(
    plot(cc, colorkey=FALSE, xlab='', ylab='', sub='') +
    layer(panel.text(cc@j+1L, cc@i+1L, paste0(round(cc@x)),
                     col=gray(cc@x>30))),
    plot(gg, colorkey=FALSE, xlab='', ylab='', sub='') +
    layer(panel.text(gg@j+1L, gg@i+1L, round(gg@x,2), col=gray(abs(gg@x)>1.5))),
    plot(A, colorkey=FALSE, xlab='', ylab='', sub='') +
    layer(panel.text(A@j+1L, A@i+1L, round(A@x, 2),
                     col=gray(A@x>0.5))), ncol=3)
```

The $\tilde{\boldsymbol{C}}$ matrix is diagonal for $\tilde{\boldsymbol{C}}_{ii}$ equals the area of the polygons formed from the dual

mesh. The $\tilde{\boldsymbol{C}}_{ii}$ is equal the sum of one third the area of each triangle that the vertice i is part of. Notice that each polygon around each mesh node is formed by one third of the triangles that it is part of.

The $\boldsymbol{G}$ matrix reflects the connectivity of the mesh nodes. Nodes not connected by edges have corresponding entry as zero. The values do not depend on the size of the triangles as they are scaled by the area of the triangles. For more detailed information, see A.2 in [Lindgren et al., 2011].

We have seen that the resulting precision matrix for increasing ν is a convolution of the precision matrix for $\nu - 1$ with a scaled $\boldsymbol{K}_\kappa$. It still implies denser the precision matrix when working with $\kappa\boldsymbol{C} + \boldsymbol{G}$.

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

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

For $\alpha = 1$ and $\alpha = 2$ we have (1.10). 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.

1.2 A toy example

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-toy.R`

In this example we will fit a simple geostatistical model for the toy dataset simulated in Section 1.1.4.

```
data(SPDEtoy)
```

This dataset is a three-column `data.frame`. Coordinates are in the first two clomuns and the response is in the third column.

```
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 ...
```

1.2.1 SPDE model definition

We have n observations y_i, $i = 1, ..., n$, at locations s_i and we define the following model

$$\begin{gathered} \mathbf{y}|\beta_0, \mathbf{u}, \sigma_e^2 \sim N(\beta_0 + \mathbf{A}\mathbf{u}, \sigma_e^2) \\ \mathbf{u} \sim GF(0, \Sigma) \end{gathered} \tag{1.12}$$

where β_0 is the intercept, $\boldsymbol{A}$ is the projector matrix and x is a spatial Gaussian random field.

The first step in working with SPDE is to build a mesh. We will use as a mesh a collection of triangles, formed by m vertices and the edges connecting them. This mesh

has to cover the entire spatial domain of interest. More details on the mesh building are given in section 1.3. Here we will use the fifth mesh built in Section **??** using the following code

```
pl.dom <- cbind(c(0,1,1,0.7,0), c(0,0,0.7,1,1))
mesh5 <- inla.mesh.2d(, pl.dom, max.e=c(0.092, 0.2))
```

The SPDE model in the original parameterization can be built 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.

We will choose our parameters based instead on the marginal variance and the practical range, $\sqrt{8\nu}/\kappa$. For details on this parameterization see [Lindgren, 2012]. When defining the SPDE model we will need also to set priors for both the parameters. The smoothness parameter ν is fixed as $\alpha = \nu + d/2 \in [1, 2]$. The `inla.spde2.pcmatern()` uses this parameterization to set the Penealized Complexity prior, PC-prior, as derived in [Fuglstad et al., 2017].

Our example domain is the $[0, 1] \times [0, 1]$ square. We set the prior median to 0.3. We set the probability that the marginal standard deviation exceeds 1 to 1%. The toy dataset was simulated with $\alpha = 2$ and it is considered to fit the model as well.

```
spde5 <- inla.spde2.pcmatern(
    mesh=mesh5, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.3, 0.5), ### P(practic.range<0.3)=0.5
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

1.2.2 Projector matrix

The second step in preparing to use SPDE is to build a projector matrix. This will project the random field modeled at the mesh nodes. For details, see section 1.1.5. The projector matrix can be built 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. The projection is a weighted average using weights computed by the `inla.spde.make.A()` function. Using the toy data set and example 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 times the number of vertices in the mesh

```
dim(A5)

## [1] 200 490
```

Because each point location is inside one of the triangles there are 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 elements on each line sum to one.

```
table(rowSums(A5))

##
##   1
## 200
```

The reason they sum to one is that multiplication with any vector of weights at mesh nodes times this matrix is the projection of these weights at the point locations.

There are some columns in the projector matrix all of whose elements equal zero.

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

##
## FALSE  TRUE
##   242   248
```

These columns correspond to triangles with no point location inside. These columns can be dropped. The `inla.stack()` function (section 1.2.3) does this automatically.

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 `mesh1` built in section **??**.

```
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
```

Each element equals one because in this instance the projections to the location points are equal to the weight at the corresponding node (at the same location) of the mesh.

1.2.3 The data stack

The `inla.stack()` function is usefull for organizing data, covariates, indices and projector matrices. All are relevant to SPDE models. `inla.stack()` helps to control the way effects are projected in the linear predictor. Detailed examples including one dimensional, replicated random field and spacetime models are presented in [Lindgren, 2012].

In the toy example we have a linear predictor that can be written as

$$\eta^* = \mathbf{1}\beta_0 + \mathbf{A}\mathbf{u} \, .$$

The first half of the right side represents the intercept. The term on the far right represents the spatial effect. Each component is represented as a product of a projector matrix and an effect.

The Finite Element Method solution considered for implementing the SPDE models build the model over the mesh nodes. Usually the number of nodes is not equal to the number of locations for which we have observations. The `inla.stack` function allows us to work with predictors that includes terms with different dimensions. The three main `inla.stack()` arguments are the `data` vectors list, a list of projector matrices (each related to one block effect) and the effects.

We need two projector matrices: the projector matrix for the latent field and a matrix that is a one-to-one map of the covariate and the response. The latter matrix can simply be a constant rather than a diagonal matrix. Thus 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 any column in a projector matrix that has a zero sum. It generates a new and simplified matrix. The `inla.stack.A()` extracts a simplified predictor matrix to use with the `inla()` function, while the `inla.stack.data()` function extracts the corresponding data.

The simplified projector matrix from the stack consists of the simplified projector matrices, where each column holds one effect block.

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

## [1] 200 249
```

In the toy example we have one column more than the number of columns with non-null elements in the projector matrix.

1.2.4 Model fitting and some results

To fit the model, we need to remove the intercept from the formula and add it as a covariate term, so that all the formula's covariate terms can be captured in a projector matrix. Them the matrix of predictors is passed to the `inla()` function in its `control.predictor` argument

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

The `inla()` function returns an object that is a set of several results. It includes summaries, marginal posterior densities of each parameter in the model, the regression parameters, each element that is a 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 9.558645 0.532128   8.470973 9.566697   10.60065 9.580974 1.790807e-10
```

and the summary of $1/\sigma_e^2$ by

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

##                                                mean        sd 0.025quant
## Precision for the Gaussian observations 2.720618 0.4290992   1.968994
##                                                0.5quant 0.975quant     mode
## Precision for the Gaussian observations 2.689447    3.65456 2.630198
```

A marginal distribution in `inla()` output consists of two vectors. One is a set of values on the range of the parameter space with posterior marginal density bigger than zero and another is the posterior marginal density at each one of these values. Any posterior marginal can be transformed. If we want the posterior marginal for σ_e, the square root of σ_e^2, for example, we use

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

Now we are able to summarize this distribution.

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

## [1] 0.6118776

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

## [1] 0.5237614 0.6096876 0.7116268

inla.hpdmarginal(0.95, post.se)

##                   low      high
## level:0.95 0.5200683 0.7072642

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

## [1] 0.004872462 0.959916660
```

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] 9.137271
```

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

## [1] 2.938193

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

## [1] 0.3201411
```

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

1.2.5 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 subsection 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))
```

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   0.1228 0.6367    -1.1069   0.1123     1.4146  0.0936   0
## APredictor.202   2.9738 0.8173     1.3815   2.9685     4.5972  2.9583   0
## APredictor.203 -2.7025 1.0898    -4.8525  -2.6997    -0.5685 -2.6942   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 1.3692 4.583411
```

and see that around the point (0.5,0.5) the random field has positive values, see Figure 1.2.6.

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]  0.1224874  2.9734719 -2.7021620
```

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

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

## [1]  0.1224874  2.9734719 -2.7021620
```

and see that for the mean we have similar values than those on previous subsection. Also, we can get the standard deviation

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

## [1] 0.7397400 0.9926802 1.2683836
```

and we have a little difference.

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

## [1] 0.4898804 0.6148041 0.8634632
```

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 1.2.6.

1.2.6 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 subsection, it is possible to find the marginal distribution or to make a projection of some functional of the response.

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  9.681 0.344      9.008    9.680     10.358  9.679
## fitted.APredictor.202 12.532 0.625     11.308   12.531     13.763 12.529
## fitted.APredictor.203  6.857 0.976      4.948    6.853      8.788  6.845
```

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 11.30454 13.75801
```

and see that around the point (0.5,0.5) we have the values of the response significantly larger than β_0, see Figure 1.2.6.

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]  9.681133 12.532117  6.856483
```

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.
##  0.3396  0.7899  1.1696  1.9685  2.1570  8.2790

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

## [1] 1.354064 1.510507 1.701425
```

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
##      0.32786989      9.01280379      0.08696628      9.42763996
```

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 subsection, on Figure 1.2.6 with the commands bellow

```
library(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',
```

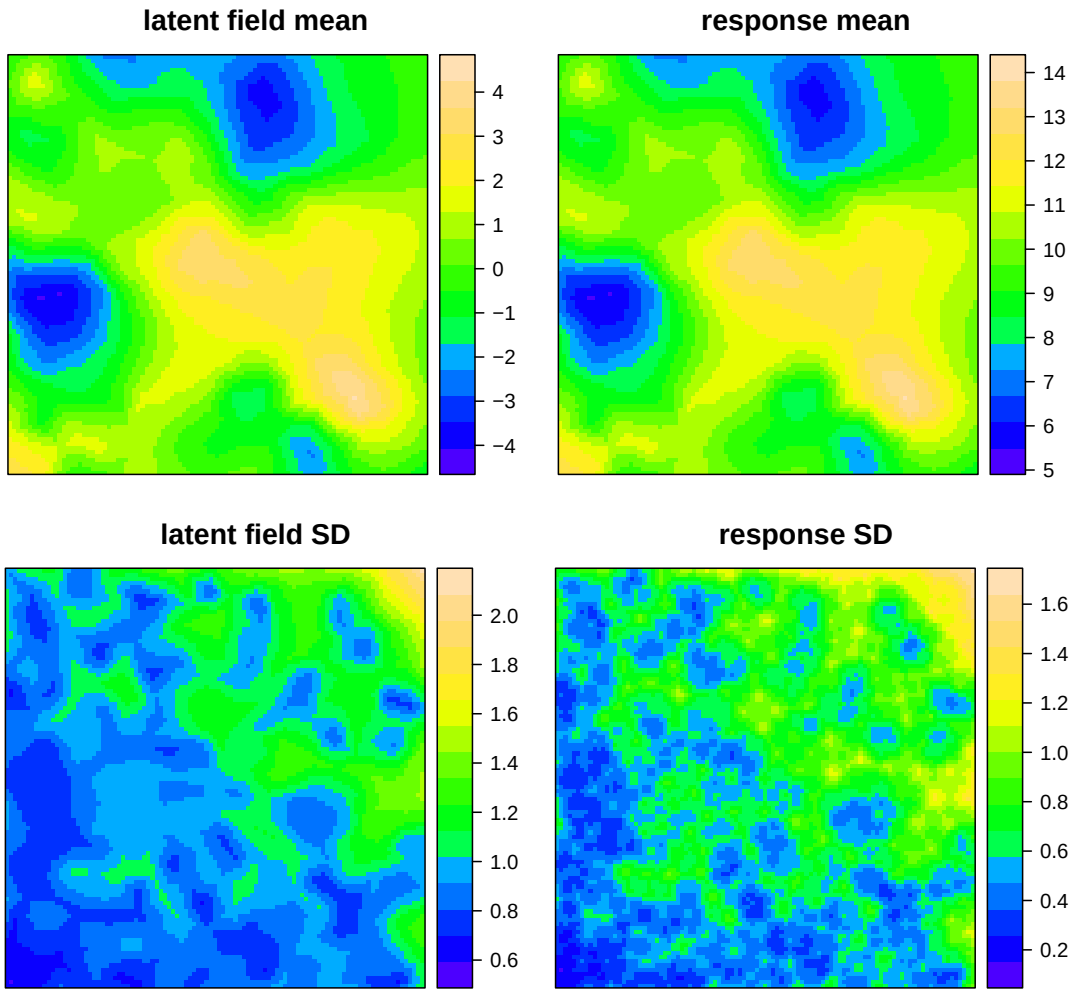

Figure 1.4: The mean and standard deviation of the random field (top left and bottom left) and the mean and standard variation of the response (top right and bottom right)

```
                        col.regions=topo.colors(99), scales=list(draw=FALSE)),
              nrow=2)
```

We see on Figure 1.2.6 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.

1.2.7 Results from different meshes

In this subsection we compare six results for the toy dataset based on the six different meshs builded on Section 1.3. 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
##        13.24         2.00         1.05        13.42         1.51
## Running inla
##         1.23
```

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; sigma2u <- 5; kappa <- 7; nu <- 1
```

and the maximum likelihood one can have using the **geoR** package, [Ribeiro Jr and Diggle, 2001], are

```
lk.est <- c(beta=9.54, s2e=0.374, s2u=3.32, range=0.336)
```

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=sigma2u, 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 1.2.7 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

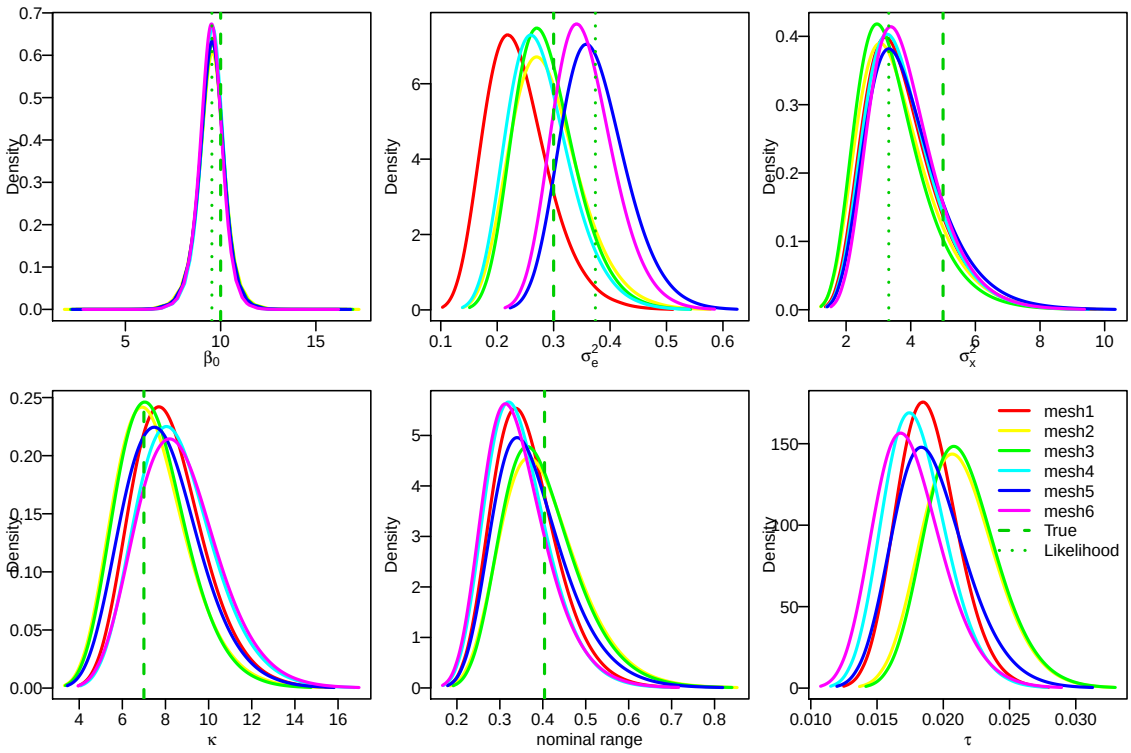

Figure 1.5: Marginal posterior distribution for β_0 (top left), σ_e^2 (top mid), σ_x^2 (top right), κ (bottom left), nominal range (bottom mid) and τ (bottom right).

estimate, the second has mode near likelihood estimate and the third large. Considering 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.

1.3 Triangulation details and examples

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-aaa-mesh.R` You can play with the shiny app typing `demo(mesh2d)`

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, 2015].

1.3.1 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 = NULL, min.angle = NULL,
##     cutoff = 1e-12, max.n.strict = NULL, max.n = NULL, plot.delay = NULL,
##     crs = 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)
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 1.3.1, 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"      "crs"
```

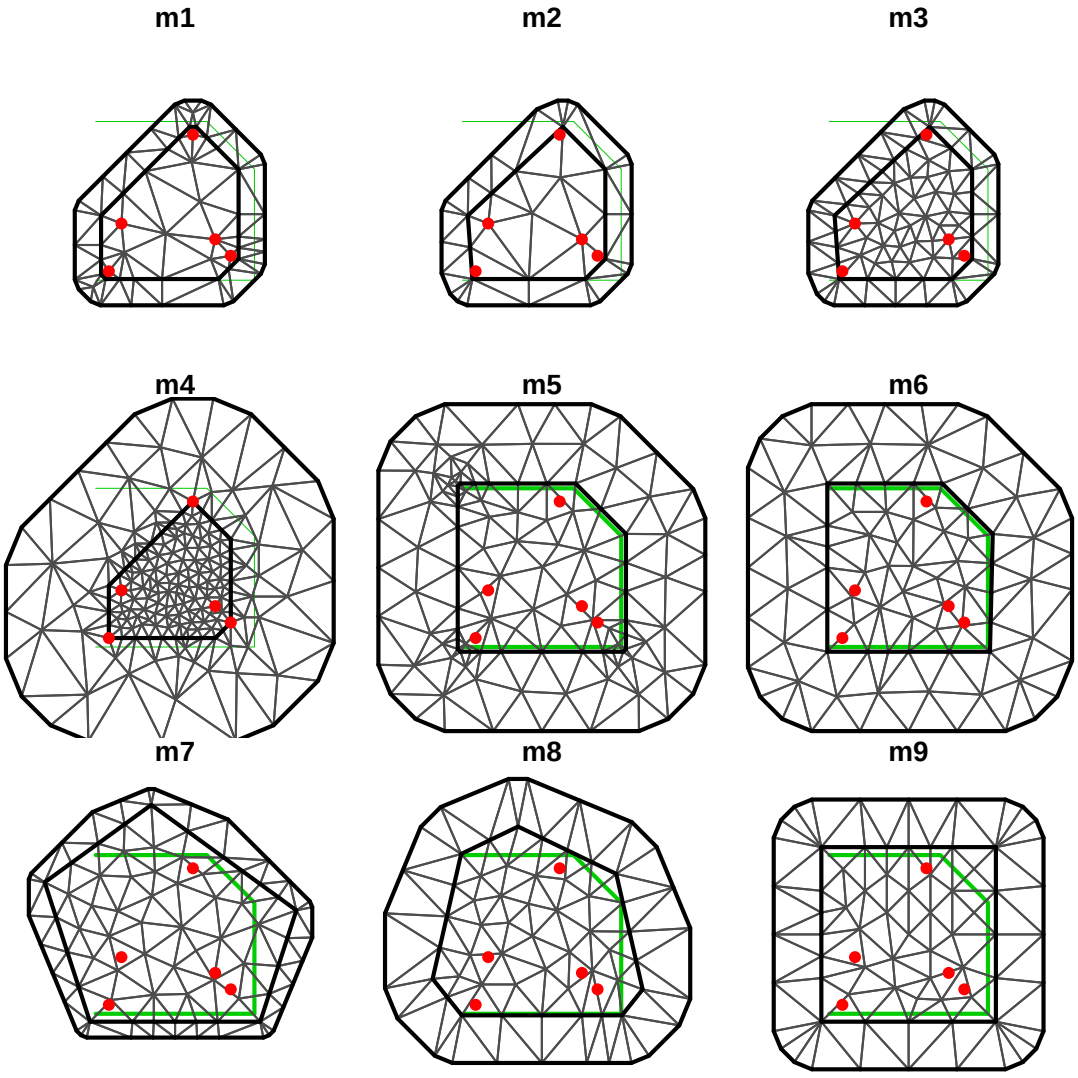

Figure 1.6: Triangulation with different restrictions.

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
```

1.3.2 Non-convex hull meshes

All the meshes in Figure 1.3.1 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, crs = 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 1.3.2 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=''))
}
```

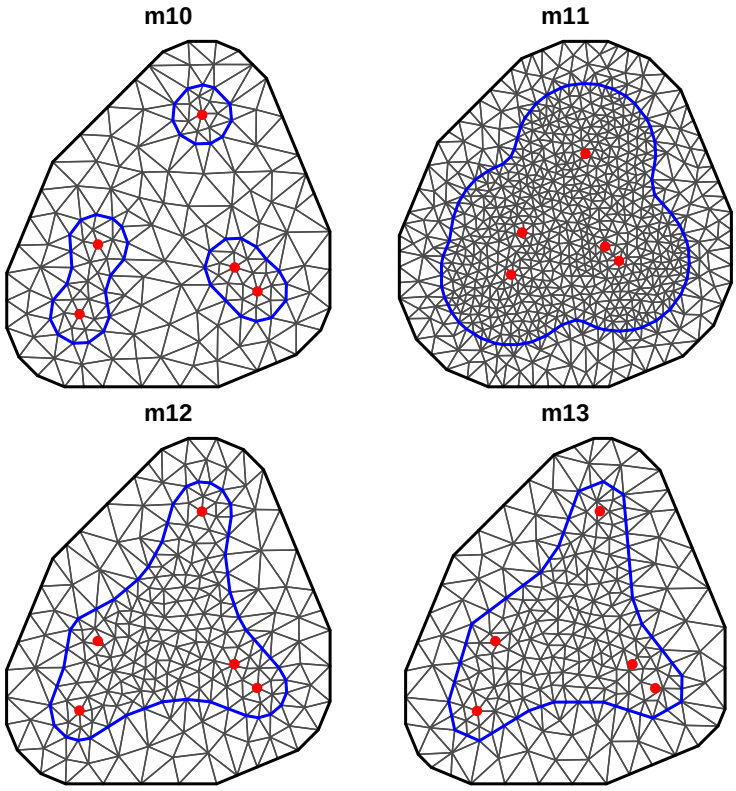

Figure 1.7: Non-convex meshes with different boundaries.

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.

1.3.3 Meshes for the toy example

To analyze the toy data set, we use six triangulation options to make comparisons in section 1.2.7. The first mesh forces the location points to be vertices of the mesh.

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

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

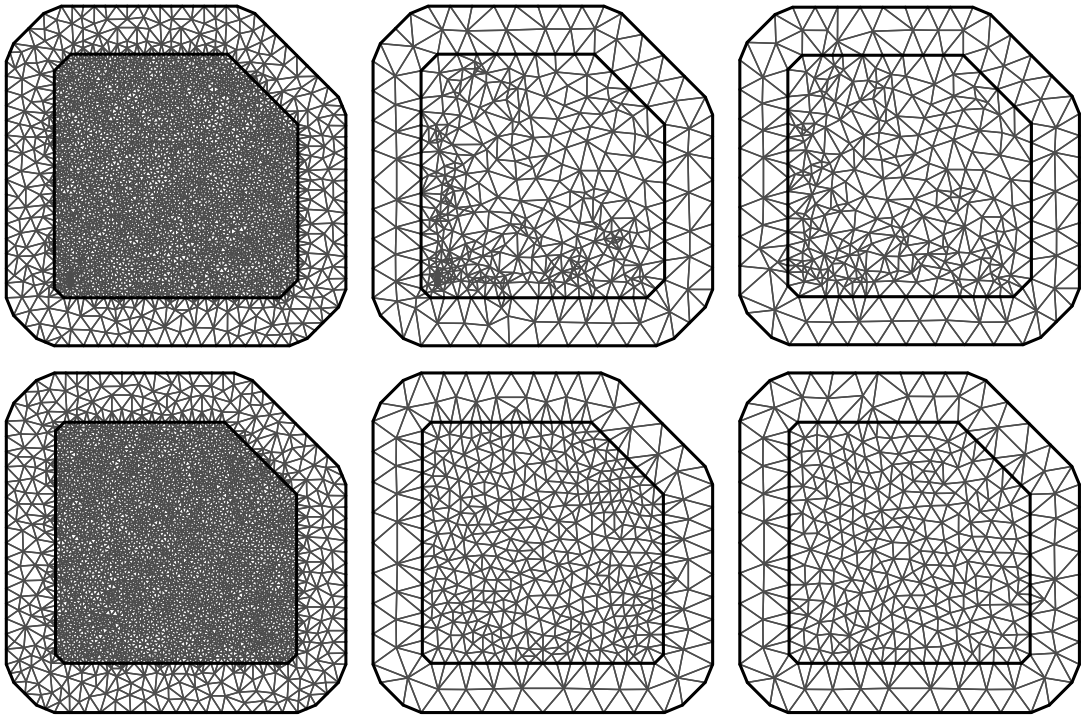

Figure 1.8: Six triangulation options for the toy example.

```
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 1.3.3 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='')
```

1.3.4 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))
```

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 1.3.4 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)
```

1.3.5 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].

```
library(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"))
```

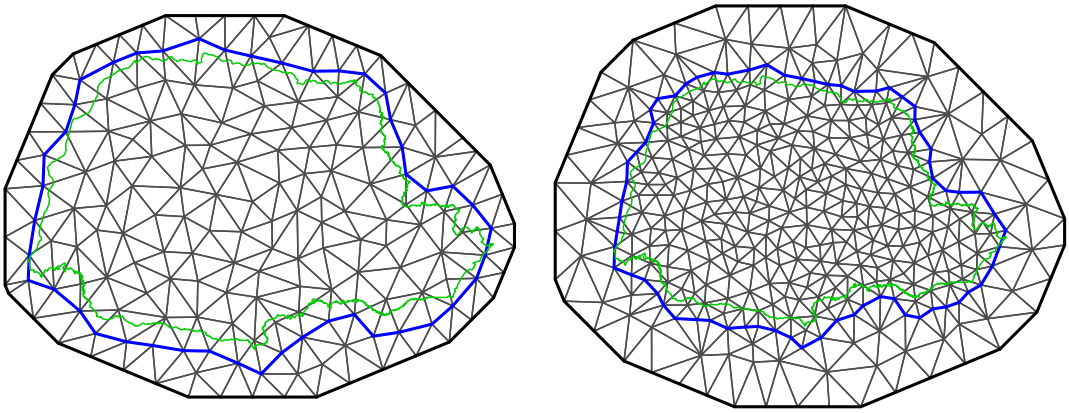

Figure 1.9: Mesh for Paraná state

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

```
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 1.3.5 with the commands bellow.

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

1.3.6 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 1.3.6 were created with the following commands:

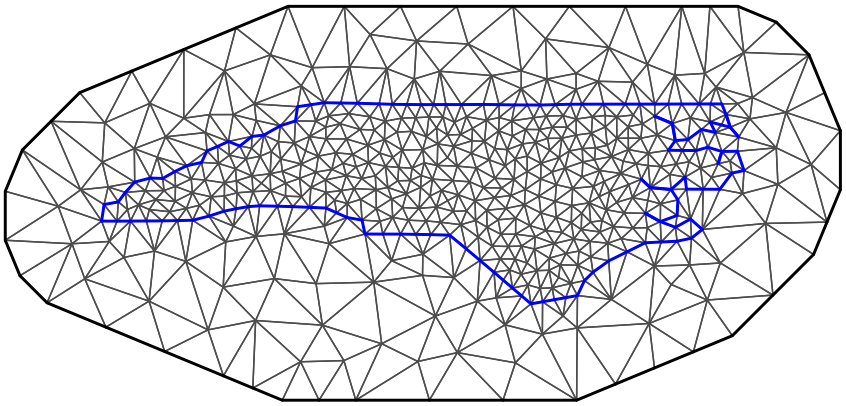

Figure 1.10: Mesh constructed using the North Carolina map

```
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.

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

The mesh is displayed in the Figure 1.3.6 using the commands below:

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.

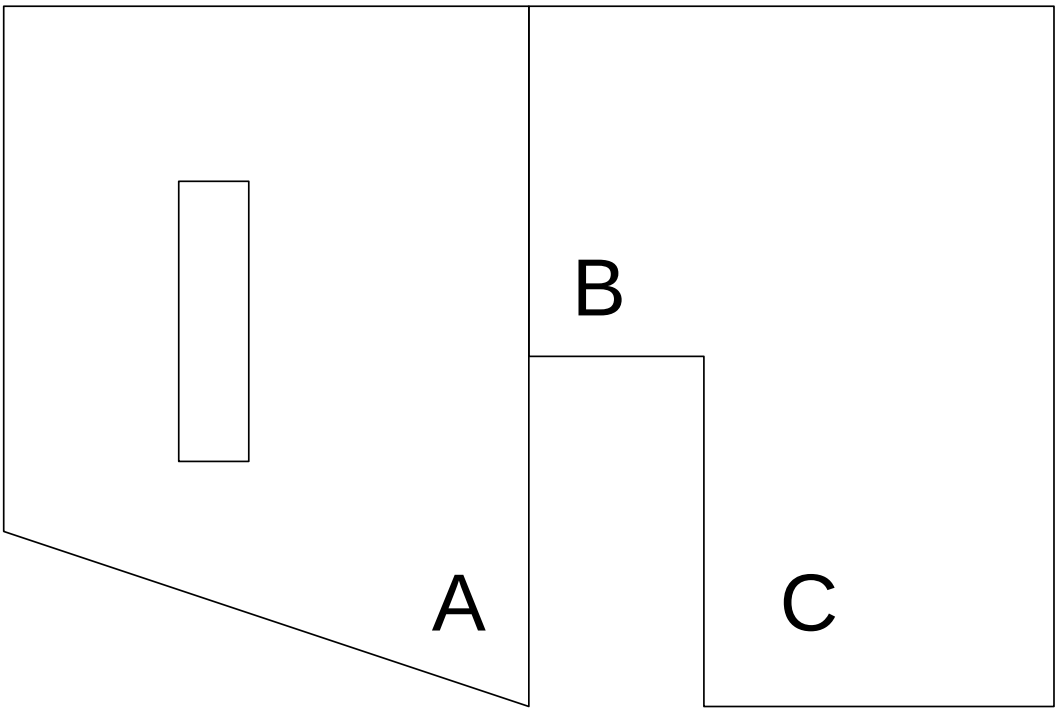

Figure 1.11: Region with a hole and non convex domain.

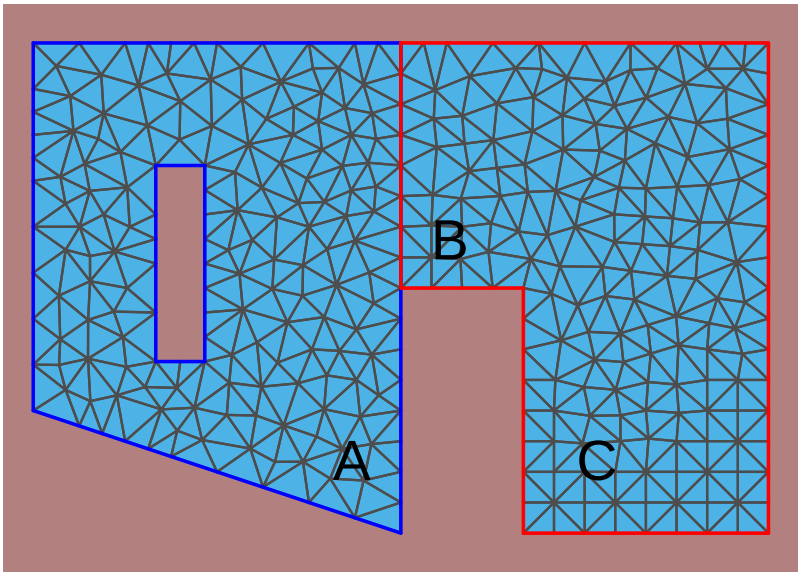

Figure 1.12: Triangulation with hole and non convex shaped region

Chapter 2

Non-Gaussian and covariates in the covariance

2.1 Non-Gaussian response: Precipitation on Paraná

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-rain.R`

A very common data in spatial statistics is the climate data. We have collected data from the National Water Agency in Brazil, in Portuguese it is *Agencia Nacional de Águas* - ANA. The ANA collect data from many locations over Brazil. All these data are freely available from the ANA website.

2.1.1 The data set

We have daily rainfall data on each day of the year 2011 at 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.

In the bellow code we show some data from 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 do visualize this four stations as red points in the right plot in Figure 2.1.1.

There are a few problems in this data set. There are seven stations with missing altitude and missing data on daily rainfall, which are displayed in red in the left plot in Figure 2.1.1. If this information is considerend when building a model it will be important to have it everywere in the state. There are digital elevation models that can be considered to find out the altitude at these locations. It can also be considered a stochastic model.

We will analyse the daily rainfall mean in January 2011. However, we do have 269 missing observations. So, we take the average over the number of days without missing data as follows

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

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.500   4.966   6.540   6.958   8.629  21.650       6

table(rowSums(is.na(PRprec[,3+1:31])))

##
##   0   2   4   8  17  26  31
## 604   1   1   1   1   2   6
```

and still have missing data for it as we do have stations without data in January 2011. We will also consider the Paraná state border

```
data(PRborder)
```

to define the study domain.
We can visualize the locations in Figure 2.1.1, 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)
lines(PRborder[1034:1078, ], col='cyan')
```

The size of the points on left graph are proportional to altitude of the locations. The cyan line in the east border is along the Atlantic Ocean. There are low altitudes near the sea, hight altitudes around 50 to 100 kilometers from this coast and from the mid of the state towards south as well. It decreases when goint towards the north and west sides of the Paraná state. The size of the points in the right plot is proportional to the daily average of the precipitation in January 2011. There are higher values near the coast.

2.1.2 The model and covariate selection

In this subsection we analise the average of the daily accumulated precipitation for each of the 31 days in January 2011. It must be a positive and we will consider a Gamma likelihood. In the Gamma likelihood we have $E(y_i) = a_i/b_i = \mu_i$ and $V(y_i) = a_i/b_i^2 = \mu_i^2/\phi$, where ϕ is a precision parameter. Then we have to define a model for the linear predictor $\eta_i = \log(\mu_i)$, depending on the covariates $\mathbf{F}$ and the spatial random field $\mathbf{x}$ as follows

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

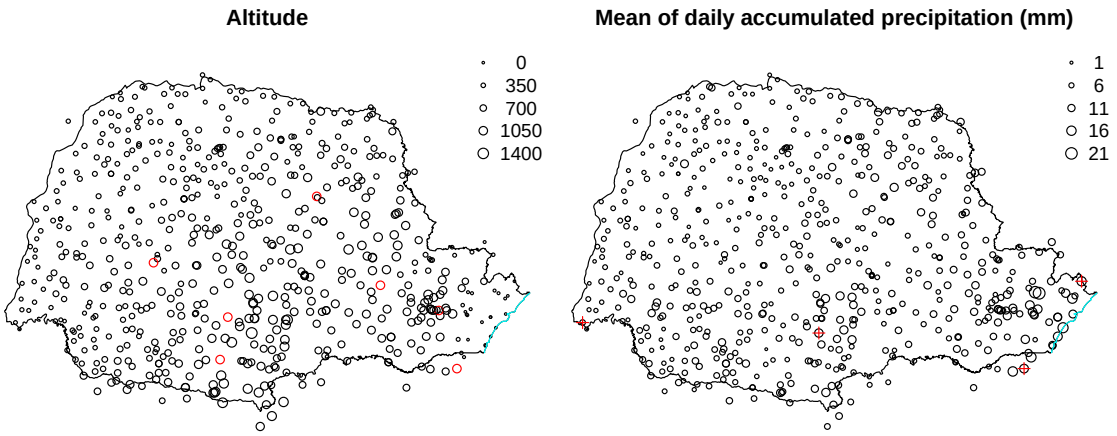

Figure 2.1: Locations of Paraná stations, altitude and average of daily accumulated precipitation (mm) in January 2011.

where, F_i is a vector of covariates (the location coordinates and altitude) tha will assumes a function detailed later and x is the spatial latent Gaussian random field. We will consider a Matérn covariance function with parameters ν, κ and σ_x^2.

Smoothed covariate effect

To make an initial exploration of the relationship between the precipitation and the covariates, we visualize some dispersion diagrams. After preliminary tests, it seems reazonable to construct a new covariate. That is the distance from each station to the Atlantic Ocean. The Paraná state border along the Atlantic Ocean is shown as cyan line in Figure 2.1.1). We can compute the distance from each station to the neighbor coordinate of this line.

To have this distance in kilometers we use the `spDists()` function from the **sp** package

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

However, this function computes the distance between each location in the first set of points to each point in the second set of points. So, we need to take the minimum along the lines of the resulting matrix of distances

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

We can see the dispersion plots in Figure 2.1.2 It seems to have a non well defined non-linear relationship with Longitude. Also, there is a similar, but inverse, relation with sea distance. We will build two models, one with longitude as covariate and another with distance to sea as covariate. We can compute fitting measures to proceed a model choise among this two options.

```
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)
```

To consider a non-linear relationship from a covariate we can set a randon walk prior over its effect. To do that we can discretize this covariate in a set of knots and place the random walk over. In this case the term in the linear predictor due to sea distance (or longitude) is discretized into m classes considering the `inla.group()` function. The model

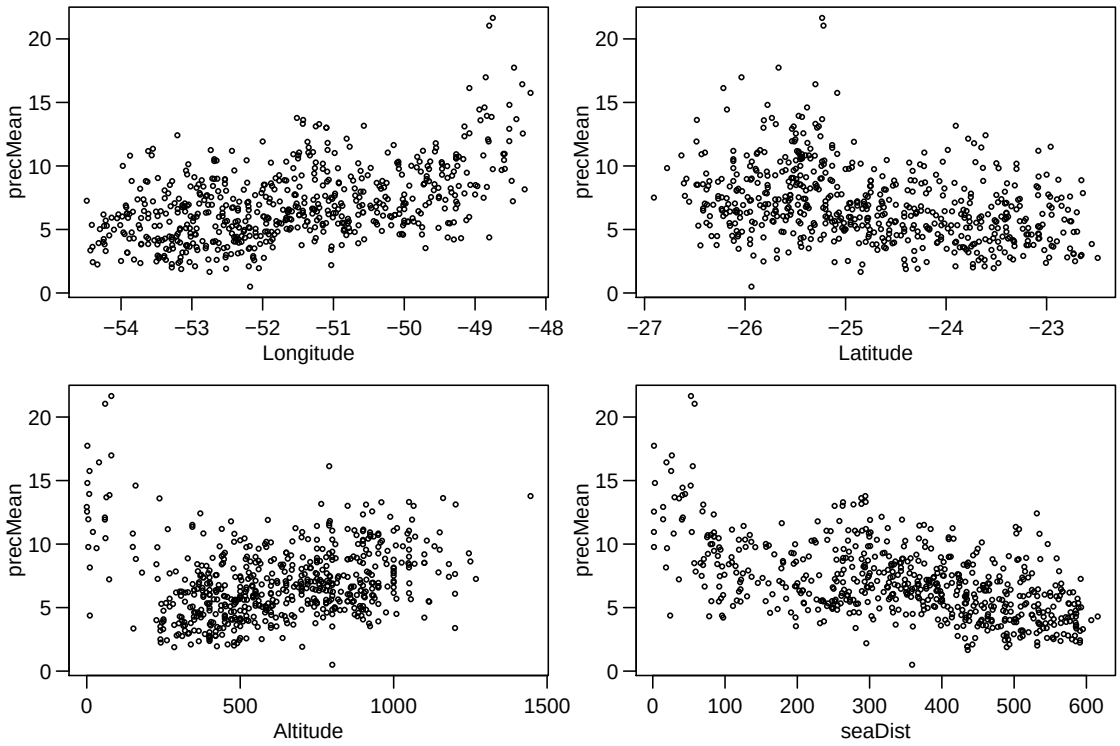

Figure 2.2: Dispersion plots of average of daily accumulated precipitation by Longitude (top left), Latitude (top right), Altitude (bottom left) and distance to sea (bottom right).

can be chosen from any one dimensional model available in INLA, from **rw1**, **rw2**, **ar1** or others. It can also be considered an one-dimensional Matérn model.

When considering intrinsic models as prior one should consider to scale it first, [Sørbye and Rue, 2014]. After it, the precision parameter can be interpreted as the inverse of the random effect marginal variance. It makes easier the process of defining a prior on it. The suggestion is to consider the PC-prior, [Simspon et al., 2017]. This can be done defining a reference standard deviation σ_0 and the right tail probability u as $P(\sigma > \sigma_0) = u$. Setting $\sigma_0 = 1$ and $u = 0.01$

```
pcprec <- list(prior='pcprec', param=c(1, 0.01))
```

Define the spatial model and prepare the data

In order to define the spatial model we need to define a mesh. We first define a boundary around the points and use it to create the mesh

```
pts.bound <- inla.nonconvex.hull(coords, 0.3, 0.3)
mesh <- inla.mesh.2d(coords, boundary=pts.bound,
                     max.edge=c(0.3,1), offset=c(1e-5,1.5), cutoff=0.1)
```

The projector matrix is computed by

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

The SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation is defined as follows

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

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

Fitting the models

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', ### first random walk prior
      scale.model=TRUE, ### scaling this prior
      hyper=list(theta=pcprec)) + ### considering the PC prior
    f(s, model=spde)
r.west <- inla(f.west, family='Gamma',
               control.compute=list(cpo=TRUE),
               data=inla.stack.data(stk.dat),
               control.predictor=list(
                   A=inla.stack.A(stk.dat), link=1))
```

We have `link=1` in the `control.predictor` to track the function link to be considered in the computation of the fitted values.

For the model with distance to sea covariate we have

```
f.seaD <- y ~ 0 + Intercept +
    f(gSeaDist, model='rw1', scale.model=TRUE,
      hyper=list(theta=pcprec)) +
    f(s, model=spde)
r.seaD <- inla(f.seaD, family='Gamma',
               control.compute=list(cpo=TRUE),
               data=inla.stack.data(stk.dat),
               control.predictor=list(
                   A=inla.stack.A(stk.dat), link=1))
```

We can see in Figure 2.1.2 that the effect from distance to sea is almost linear. We them also fit the model considering this option For the model with distance to sea covariate we have

```
f.seaD.l <- y ~ 0 + Intercept + seaDist +
    f(s, model=spde)
r.seaD.l <- inla(f.seaD.l, family='Gamma',
                 control.compute=list(cpo=TRUE),
                 data=inla.stack.data(stk.dat),
                 control.predictor=list(
                     A=inla.stack.A(stk.dat), link=1))
```

Compare the models and look at the results

We have the negated sum of the log CPO from each model with

```
slcpo <- function(m, na.rm=TRUE)
    -sum(log(m$cpo$cpo), na.rm=na.rm)
c(long=slcpo(r.west), seaD=slcpo(r.seaD),
  seaD.l=slcpo(r.seaD.l))

##     long     seaD   seaD.l
## 1278.658 1278.279 1273.842
```

wich inform that the model with distance to sea as a linear effect has a bit better fit. It was just to show one way to compare models considering how it fits the data. In this case the three models have a very similar sum of the log CPO.

We got the summary of posterior distribution of the intercept with

```
round(r.seaD.l$summary.fixed, 4)

##               mean     sd 0.025quant 0.5quant 0.975quant    mode kld
## Intercept  2.4293 0.0943     2.2473   2.4276     2.6215  2.4245   0
## seaDist   -0.0016 0.0002    -0.0020  -0.0015    -0.0011 -0.0015   0
```

Summary of the PMD for the gamma likelihood dispersion parameter

```
round(unlist(r.seaD.l$summary.hy[1,]), 4)

##       mean         sd 0.025quant   0.5quant 0.975quant       mode
##    14.8825     1.4495    12.1855    14.8333    17.8773    14.7596
```

The summary for the practical range and the standard deviation of the spatial process is

```
round(r.seaD.l$summary.hyper[-1, ], 4)

##                mean     sd 0.025quant 0.5quant 0.975quant   mode
## Range for s 0.6695 0.1437     0.4413   0.6508     1.0025 0.6132
## Stdev for s 0.2336 0.0220     0.1932   0.2327     0.2796 0.2309
```

The posterior marginal distribution for β_0, the practical range (in degrees), the standard deviation for the spatial random field,

mean and 95% credibility interval of the distance to sea effect at Figure 2.1.2. We choose $1/\kappa$ instead κ because $1/\kappa$ is the range parameter and in this case is expressed in degrees units. Figure 2.1.2 we look

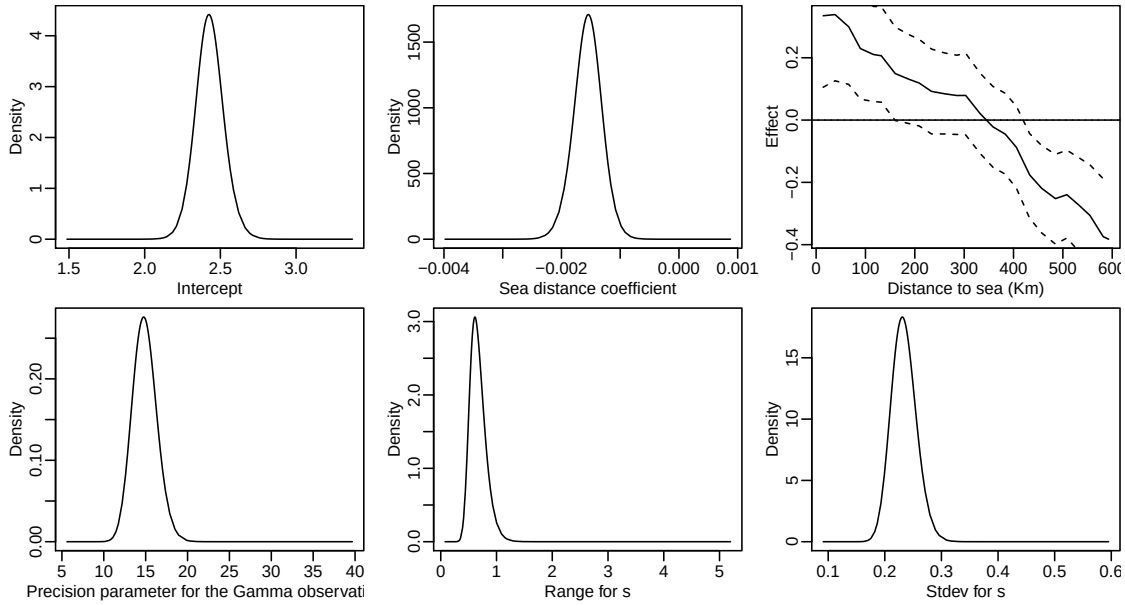

Figure 2.3: PMD for β_0 (top left), PMD for the distance to sea coefficient (top mid), posterior mean (continuous line) and 95% credibility interval (dashed lines) for the distance to sea effect (top right), PMD for the Gamma likelihood precision (bottom left), PMD for the practical range (bottom mid) and PMD for the standard deviation of the spatial field (bottom right).

```
par(mfrow=c(2,3), mar=c(3,3.5,0,0), mgp=c(1.5, .5, 0), las=0)
plot(r.seaD.1$marginals.fix[[1]], type='l',
     xlab='Intercept', ylab='Density')
plot(r.seaD.1$marginals.fix[[2]], type='l',
     xlab='Sea distance coefficient', ylab='Density')
plot(r.seaD$summary.random[[1]][,1:2], type='l',
     xlab='Distance to sea (Km)', ylab='Effect')
abline(h=0, lty=3)
for (i in c(4,6))
    lines(r.seaD$summary.random[[1]][,c(1,i)], lty=2)
abline(h=0)
for (j in 1:3)
plot(r.seaD.1$marginals.hy[[j]], type='l',
     ylab='Density', xlab=names(r.seaD.1$marginals.hy)[j])
```

Suppose that we want to test the significance of the spatial random effect component on the model. To access the significance of this effect, we can fit the model without the spatial effect and compare the sum of the log CPO with the model including the spatial effect.

```
r0.seaD.1 <- inla(y ~ 0 + Intercept + seaDist,
                  family='Gamma', control.compute=list(cpo=TRUE),
                  data=inla.stack.data(stk.dat),
                  control.predictor=list(A=inla.stack.A(stk.dat), link=1))
```

And we can compare minus the log CPO sum

```
c(seaD.l=slcpo(r.seaD.l), seaD.l0=slcpo(r0.seaD.l))

##   seaD.l  seaD.l0
## 1273.842 1371.288
```

and conclude that the spatial effect is useful.

2.1.3 Prediction of the random field

We can visualize the spatial effect projecting it on a grid. We want to have a regular grid where each pixel is a square with near 4 kilometers each side. We will use the (4/111) factor because each degree has aproximately 111 kilometers to give the size of the pixels in degrees

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

## [1] 0.03603604
```

The Paraná state shape is wider than height and we will use it to define the grid. We consider the range along each axis and consider the size of the pixels to define the number of pixels in each direction. We will divide the range along each axis by the of each coordinate and roun it

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

## [1] 183 117
```

The `inla.mesh.projector()` create a projector matrix. If a set of coordinates is not supplied it will create a grid automatically. We will change the limits and dimensions as desired

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

Them we can use it in the `inla.mesh.project()` function to do the projection for both the posterior mean and the posterior standard deviation

```
xmean <- inla.mesh.project(projgrid, r.seaD$summary.random$s$mean)
xsd <- inla.mesh.project(projgrid, r.seaD$summary.random$s$sd)
```

To improve the visualization we will assign NA for the pixels falling outside the Paraná state border. We use the function `inout()` from the **splancs** package to do that

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

##
## FALSE  TRUE
##  7865 13546

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

The posterior mean and posterior standard deviation for the spatial effect at each pixel is visualized in Figure 2.1.4. In the top left of this figure one can see that the posterior mean varies from -0.6 to 0.4. This is the variation after accounting for the distance to sea effect. When comparing it with the standard deviation image, it seems to be considerable as the standard deviations aries around 0.2. The variation in the standard deviation is mainly due to the density of the stations over the region. At the top right plot in Figure 2.1.4 the first green region from right to left is near the captital city of Curitiba where the number of stations around is relatively bigger than in other regions of the state.

2.1.4 Prediction of the response on a grid

When the objective is to to predict the response a simple approach will be to add the other terms to the spatial field projected in the previous Subsection and apply the inverse of the link function. A full Bayesian analysis for this problem is the joint prediction with the estimation process. However, it can be computationally expensive when the number of pixeis in the grid is large. At the end of this subsection we show a cheap way for a specific case.

By computation of the posterior distributions

Considering the grid from the previous subsection we can avoid to compute the posterior marginal distributions at those pixels that are not inside the Paraná border. That is we consider the corresponding lines of the projector matrix from the projector object built earlier

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

We need to have the covariate for every pixel. In order to have it we can extract these coordinates from the projector object as well

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

Computing the distance to sea for each selected pixel

```
seaDist0 <- apply(spDists(PRborder[1034:1078,],
                          prdcoo, longlat=TRUE), 2, min)
```

Suppose that our model is the one with the smoothed effect of sea distance. We have to discretize each computed distance in the same way as for the estimation data at the stations. We have to collect the knots and order it

```
seaDist.k <- sort(unique(stk.dat$effects$data$gSeaDist))
```

We them have to identify to with knot each distance to sea computed for the pixels belongs to and assign these knots for each pixel. The knots can are actually the mids of the breaks and we will consider it as follows

```
seaDist.b <- (seaDist.k[-1] + seaDist.k[length(seaDist.k)])/2
i0 <- findInterval(seaDist0, seaDist.b)+1
gSeaDist0 <- seaDist.k[i0]
```

Building the stack data with the prediction scenario and joining it with the data with the data for estimation

```
stk.prd <- inla.stack(
    data=list(y=NA), A=list(Aprd,1),
    effects=list(s=1:spde$n.spde,
                 data.frame(Intercept=1,
                            seaDist=seaDist0)), tag='prd')
stk.all <- inla.stack(stk.dat, stk.prd)
```

We can now use the mode of theta found as known values in the new `inla()` call. We can also avoid computing things not needed such as quantiles. We can also avoid non needed objects to be returned, such as the marginals distributions for the random effects and predictor. Since the number of latent variables is the main issue in this case, the Gaussian approximation will reduce a lot the computation time. It can be considered using `control.inla=list(strategy='gaussian'`.

```
r2.seaD.1 <- inla(f.seaD.1, family='Gamma',
                  data=inla.stack.data(stk.all),
                  control.predictor=list(A=inla.stack.A(stk.all),
                                         compute=TRUE, link=1),
                  quantiles=NULL,
                  control.inla=list(strategy='gaussian'),
                  control.results=list(return.marginals.random=FALSE,
                                       return.marginals.predictor=FALSE),
                  control.mode=list(theta=r.seaD.1$mode$theta,
                                    restart=FALSE))
```

We have to find the index for the predictions in the grid. It can be done using the `inla.stack.index()` function. We also have to assign it into the right positions of a matrix with the same dimension as the grid. We will do it for both, the posterior predicted mean and its standard deviation

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

This result is shown in the botton plot of Figure 2.1.4 with the following commands

```
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)))
```

The posterior mean for the expected ranifall in January 2011 was higher near the sea and lower n the north west side of the Paraná state. Since the linear effect from the distance to sea will drive this pattern, the spatial effect in this case is there to fit deviations from this pattern. That is, the spatial effect is higher near the sea and summing up with the higher effect from distance to sea fitted the higher observed values there. The spatial effect is also higher in a region of the west side causing the expected values to be not so low there.

Sampling at mesh nodes and interpolating

When all the covariates are smooth over space, it makes sence to make the predictions at the mesh nodes, where we have the spatial effect, and them project it on the grid. However,

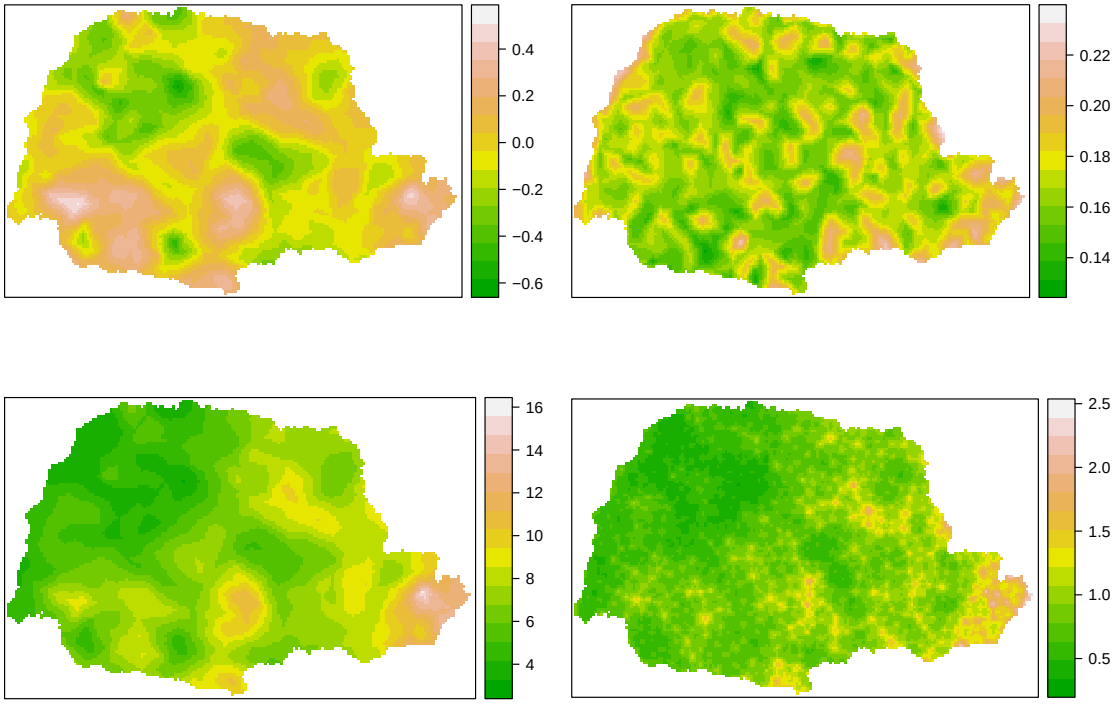

Figure 2.4: Posterior mean and standard deviation of the random field (top left and top right, respectively). Posterior mean and standard deviation for the response (bottom left and bottom right, respectively).

as long a covariate is not smooth over space this approach no longer makes sence. The advantage of this approach is that it is computationally cheap than to compute the full posterior marginals like in the previous subsection.

In our example the idea is to compute the distance to sea effect at the mesh nodes. Them, compute the linear predictor at the mesh nodes. By that, it is possible to predict the response at the mesh nodes and them do interpolate it.

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

```
seaDist.mesh <- apply(
    spDists(PRborder[1034:1078,],
            mesh$loc[,1:2], longlat=TRUE), 2, min)
```

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.

Building the stack for predict into the mesh

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

Fitting the model again and asking for the configuration, which will include in the output the precision matrix for each hyperparameter configuration needed for sampling from the joint posterior

```
rm.seaD.l <- inla(f.seaD.l, 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)) ## need to sample
```

Sampling from the model

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

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$latent[nrow(PRprec) + 1:nrow(mesh$loc)])))

## [1]  967 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] 0.9997716

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

## [1] 0.9609754
```

2.2 Survival analysis

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-survival.R`

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 2.2.1 we show how to fit a parametric survival model and in the section 2.2.2 we also show how to fit the semiparametric Cox proportional hazard model.

2.2.1 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)
```

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

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 'weibullsurv' likelihood. Hovewer, other parametric likelihoods can be used as well.

```
r <- inla(formula, family="weibullsurv", 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  -5.7182 0.2468    -6.1981  -5.7192    -5.2285 -5.7182   0
## sex  0.0719 0.0693    -0.0641   0.0719     0.2079  0.0718   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     0.0440  0.0247   0
```

and the hyperparameters with

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

##                                    mean    sd 0.025quant 0.5quant 0.975quant
## alpha parameter for weibullsurv 0.600 0.016      0.568    0.600      0.631
## Range for spatial               0.313 0.157      0.111    0.279      0.711
## Stdev for spatial               0.286 0.073      0.167    0.278      0.451
##                                    mode
## alpha parameter for weibullsurv 0.601
## Range for spatial               0.223
## Stdev for spatial               0.263
```

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 2.2.1. 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)
```

2.2.2 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

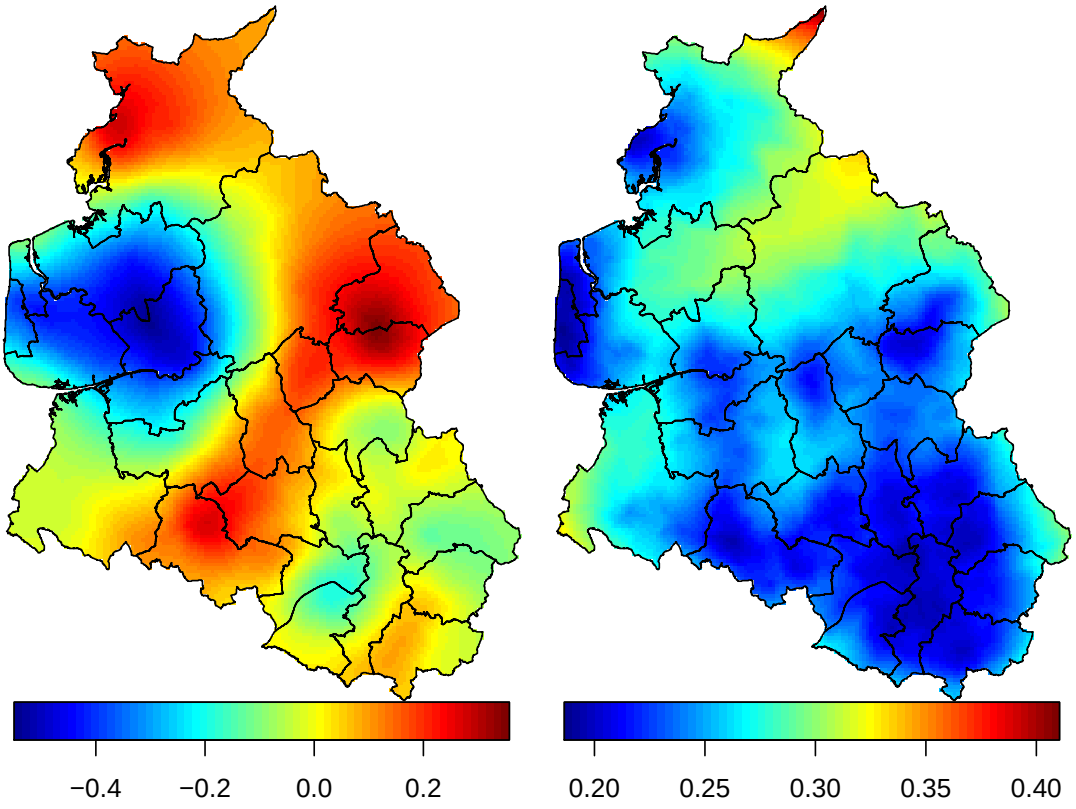

Figure 2.5: Map of the spatial effect in the Weibull survival model. Posterior mean (left) and posterior standard deviation (right).

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.

```
library(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 -10.193474960 -10.161989424
## sex 0.052175883   0.057962920   0.068750769
## age 0.029617048   0.033255809   0.034798643
## wbc 0.003072442   0.003395679   0.003427944
## tpi 0.029284096   0.034231896   0.032351156
```

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] 0.9941812

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

## [1] 0.9988743
```

2.3 Explanatory variables in the covariance

In this example we will show how an example of the model proposed in [Ingebrigtsen et al., 2014]. This is a way to include explanatory variables (covariates) in both the SPDE model parameters, the local precision and the range. The R source is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-nonstationar.R`

2.3.1 Introduction

We start to remember the definition for the precision matrix considering the equations (1.10) and (1.11). Considering $\alpha = 1$ and $\alpha = 2$ we have

- $\alpha = 1$: $\boldsymbol{Q}_{1,\kappa} = \boldsymbol{K}_\kappa = \kappa^2\boldsymbol{C} + \boldsymbol{G}$
- $\alpha = 2$: $\boldsymbol{Q}_{2,\kappa} = \boldsymbol{K}_\kappa\boldsymbol{C}^{-1}\boldsymbol{K}_\kappa = \kappa^4\boldsymbol{C} + \kappa^2\boldsymbol{G} + \kappa^2\boldsymbol{G} + \boldsymbol{G}\boldsymbol{C}^{-1}\boldsymbol{G}$

The approach is to consider a regression like model for $\log\tau$ and $\log\kappa$. In order to implement it, the precision matrix are written in a more general way as

$$\boldsymbol{Q} = \boldsymbol{D}^{(0)}(\boldsymbol{D}^{(1)}\boldsymbol{M}^{(0)}\boldsymbol{D}^{(1)} + \boldsymbol{D}^{(2)}\boldsymbol{D}^{(1)}\boldsymbol{M}^{(1)} + (\boldsymbol{M}^{(1)})^T\boldsymbol{D}^{(1)}\boldsymbol{D}^{(2)} + \boldsymbol{M}^{(2)})\boldsymbol{D}^{(0)} \tag{2.1}$$

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 $\boldsymbol{D}^{(0)}$, $\boldsymbol{D}^{(1)}$ and $\boldsymbol{D}^{(2)}$ are diagonal with elements used to describe non-stationarity. The definition of these matrices 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 $\boldsymbol{B}^{(0)} = [0\ 1\ 0]$ (one by three) matrix for the local precision parameter τ, $\boldsymbol{B}^{(1)} = [0\ 0\ 1]$ (one by three) matrix for the scaling parameter κ, and $\boldsymbol{B}^{(2)} = 1$. When these basis matrices are supplied as just one line matrix, the actual basis matrix will be formed having all lines equals to this unique line matrix.

In the next section, we add one of the location coordinates as a fourth column for $\boldsymbol{B}^{(1)}$ in order to build a non-stationary model.

2.3.2 An example

We now will define a model were the local precision depends on one of the coordinates. Note that in order to build a precision matrix defined in the equation (2.1), one also needs $\boldsymbol{M}^{(0)}$, $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ defined at the mesh nodes.

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 using this 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. We want to define a model where the local precision depends on the first coordinate. So, we have to consider a fourth column for $\boldsymbol{B}^{(1)}$, supplied as in the `B.tau` argument of the `inla.spde2.matern()` function. By doing it, we also do need to set prior θ according to its new dimension, a three length vector. The default is a Gaussian distribution and we just need to specify the mean and precision diagonal, two vectors as follows:

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

where it was set $\boldsymbol{B}^{(1)}$ to define

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

having the local precision non-constant. In this case it also implies in a marginal variance non-constant as well, as the marginal variance is:

$$\sigma^2 = (4\pi\tau^2\kappa^2)^{-1}.$$

We can have a feeling about the model just defined setting values for θ, build the covariance and look at the marginal variance (our interest). We consider two different cases:

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

The precision matrices are built with

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

As we have the x-coordinate in the (0,1) interval, and the sin function is positive and non-decreasing in this interval, the second precision matrix has larger values of τ_i, implying in a lower marginal variance.

To clarify, we compute both covariance matrix implied. The covariance matrix of

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

at the mesh nodes as the inverse of the precision matrix. The `inla.qinv()` function computes the diagonal of the covariance matrix and the covariance elements at the non-zero elements of $\boldsymbol{Q}$ efficiently.

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

A summary of the variances implied (diagonal of the covariance matrix) for both covariance matrices is obtained with

```
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
```

showing large values for the first case.

We can see the marginal variance at the mesh nodes considering both process in the Figure 2.3.2. Commands to make the figure 2.3.2:

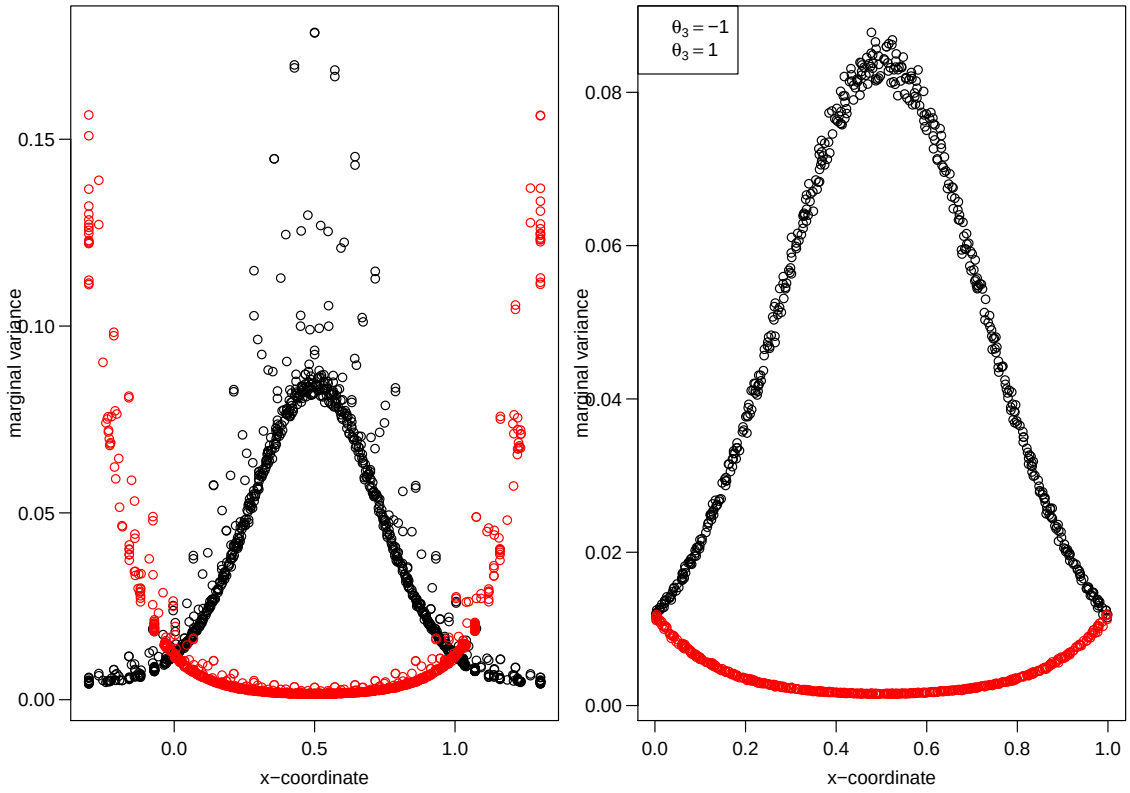

Figure 2.6: Marginal variances at mesh nodes implied by both non-stationary process defined, considering all the mesh points (left) and only those inside the region domain (right).

```
par(mfrow=c(1,2), mar=c(3,3,.5,.5), mgp=c(2, .7, 0), las=1)
plot(mesh$loc[,1], v1, ylim=range(v1,v2), las=1,
     xlab='x-coordinate', ylab='marginal variance')
points(mesh$loc[,1], v2, col=2)
i1 <- which((mesh$loc[,1]>0) & (mesh$loc[,1]<1) &
  (mesh$loc[,2]>0) & (mesh$loc[,2]<1))
plot(mesh$loc[i1,1], v1[i1], ylim=range(v1[i1],v2[i1]),
     xlab='x-coordinate', ylab='marginal variance', las=1)
points(mesh$loc[i1,1], v2[i1], col=2)
legend('topleft', as.expression(lapply(
                      c(theta1[3], theta2[3]),
                      function(x) bquote(theta[3]==.(x)))), col=1:2)
```

In the plot at left we have some marginal variances that does not follows the pattern. For example, it is clear that some points in the right plot does not follows the expected pattern when $x - axis$ is near 0.5. These points are marginal variances computed for the mesh nodes for $y - axis$ near bellow 0 or above 1, outside the domain. This is expected as the variance of the approximation happens to be bigger at the boundary of the mesh.

2.3.3 Simulation at the mesh nodes

Both precision matrix defined previously consider that the locations are the mesh nodes. So, the simulation made with it is a realization of the random field on each point of the mesh nodes. 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 opposite happens and we have larger values, as it has lower local precision.

One can see the simulated values projected to a grid in Figure 2.3.3. We use a projector matrix to project the simulated values in the grid limited in the unit square with limits (0,0) and (1,1) with

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

Simulation with linear constraint

The linear constraint is common for models intrinsic models, such the random walks in one or two dimensions. This is not the case for the models we have defined in this chapter. However, we would like to show the use of linear constraints for the SPDE models.

Because the SPDE models are based in the Finite Element Method (FEM) approximation, 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

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

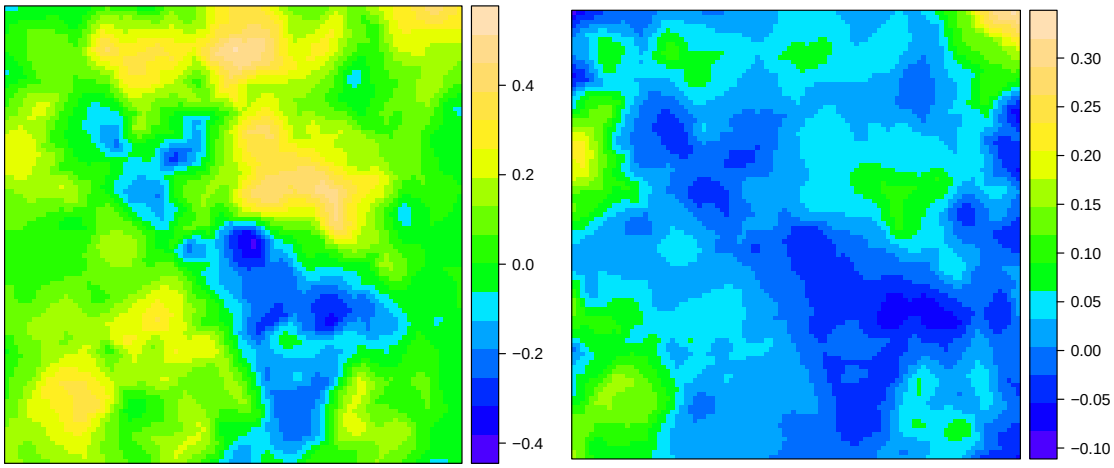

Figure 2.7: Two simulated random fields, using two diferent θ in the same basis functions.

is provided by

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

where $\boldsymbol{C}$ is the matrix used in the FEM.

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

The $\boldsymbol{C}$ 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))
```

So, 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 at mesh nodes have mean near zero.

2.3.4 Estimation with data simulated at the mesh nodes

The model can be fitted easily with the data simulated at mesh nodes. Considering that we have data exactly at each mesh node, 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 in a high 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 do not 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 -0.9621 0.0289    -1.0249  -0.9594    -0.9123 -0.9495
## Theta2 for i    2  1.9321 0.1144     1.7307   1.9228     2.1776  1.8893
## Theta3 for i   -1 -0.9589 0.0513    -1.0431  -0.9649    -0.8452 -0.9878
```

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 -1.0276 0.0335    -1.0914  -1.0285    -0.9596 -1.0317
## Theta2 for i    2  2.0958 0.1046     1.8758   2.1025     2.2847  2.1270
## Theta3 for i    1  1.0501 0.0432     0.9644   1.0503     1.1347  1.0512
```

We can see good results for both cases. We will see more results later.

2.3.5 Estimation with locations not at the mesh nodes

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

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

Now, we do the projection of the simulated data from the mesh vertices to these locations. To do it, we need a projector 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 at 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 -1.0406 0.1811    -1.4081  -1.0356    -0.6957 -1.0179
## Theta2 for i    2  2.1853 0.1866     1.8093   2.1889     2.5443  2.2020
## Theta3 for i   -1 -0.9753 0.2436    -1.4375  -0.9821    -0.4802 -1.0067

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

##               True    mean     sd 0.025quant 0.5quant 0.975quant    mode
## Theta1 for i   -1 -0.9717 0.1758    -1.3157  -0.9727    -0.6232 -0.9761
## Theta2 for i    2  2.1779 0.1880     1.7958   2.1835     2.5349  2.2031
## Theta3 for i    1  0.9291 0.2384     0.4592   0.9298     1.3959  0.9326
```

To make the visualization more good, we take the logarithm 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 in Figure 2.3.5 with commands below

```
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 in the Figure 2.3.5 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 for the first coordinate. And we see the opposite for the second random field. Also, we see that the variance of the first is greater than the second.

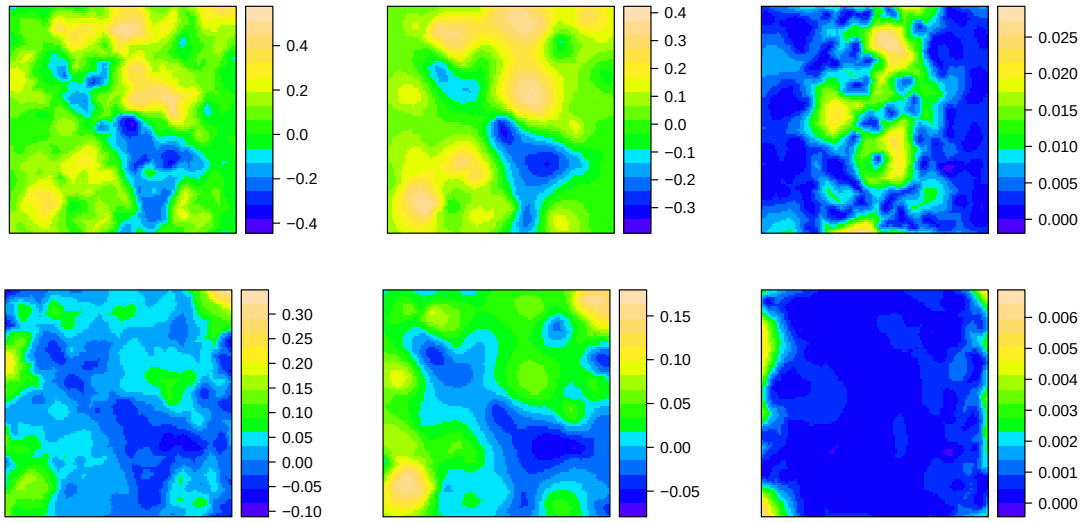

Figure 2.8: Simulated (top and bottom left), posterior mean (top and bottom mid) and the posterior variance (top and bottom right) for both random fields.

Chapter 3

Manipulating the random field and more than one likelihood

3.1 Measurement error model

Here we focus on a similar situation of the misalignement model example in the last Chapter of [Blangiardo and Cameletti, 2015]. Here we extend the model to consider a a spatially structured error. We also have a response y and a covariate c and the misalignment. The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-measurement-error.R`

3.1.1 The model

We consider the following model c

$$c_j = \alpha_c + \beta_w w_j + m_j$$

where w_i is a covariate. Considering a GF for m, we have here a kind of measurement error models, [Muff et al., 2013], where the error is assumed t o have spatial structure.

For y, we have

$$y_i \sim \alpha_y + \beta_c cc_i + x_i + e_i$$

where α_y is an intercept, β_c is the regression coefficient on the predicted value for c x_j is an zero mean random field and e_i is a error that remains unexplained on y such that $e_i \sim N(0, \sigma_y^2)$ measures the.

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.

Differently than the joint model in Chapter 8 of [Blangiardo and Cameletti, 2015] we do need to define a term that express the linear predictor of c and then copy it on the response linear predictor. We need an extra equation to do it

$$\omega_j = \alpha_c + \beta_w w_j + m_j$$

To fit ω we need o use the faked zero observations strategy, [Ruiz-Cárdenas et al., 2012]. So, we rewrite this as

$$0 = \alpha_c + \beta_w w_j + m_j - \omega_j$$

and define a Gaussian likelihood with fixed high precision to fit it.

3.1.2 Simulation from the model

We now draw a sample from this model. First, we simulate a set of locations

```
n.y <- 123;          n.c <- 234
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 <- 7;            sigma2.m <- 3
kappa.x <- 10;           sigma2.x <- 2
```

We need the simulation of m in both set of locations. To do that we use the `rMatern` function available in `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`.

```
set.seed(2)
mm <- rMatern(n=1, coords=rbind(loc.c, loc.y),
            kappa=kappa.m, variance=sigma2.m, nu=1)
xx <- rMatern(n=1, coords=loc.y, kappa=kappa.x,
              variance=sigma2.x, nu=1)
### center it to avoid confounding
mm <- mm-mean(mm)
xx <- xx-mean(xx)
```

and with the following parameters

```
alpha.c <- -5;       beta.w <- 0.5
alpha.y <- 3;        beta.c <- 2
sigma2.y <- 0.3
```

we do simulation of the covariate and response with

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

3.1.3 Fitting the model

First we build a mesh taking into account the true value of the smaller range process

```
(rmin <- min(sqrt(8)/c(kappa.m, kappa.x)))

## [1] 0.2828427

(mesh <- inla.mesh.2d(rbind(loc.c, loc.y), max.edge=rmin/c(5, 2),
                      cutoff=rmin/10, offset=rmin*c(1/2,3)))$n

## [1] 2166
```

We will use the same mesh and the index vectors for both spatial effects will 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)
```

We have to use three likelihoods. One for the response, one for the covariate c and one for the faked zero observations. It is easier to buil one stack for each one and join they to fit the model

```
stk.y <- inla.stack(data=list(y=cbind(yy, NA, NA)),
                    A=list(Ay, 1),
                    effects=list(x=1:mesh$n,
                        data.frame(
                            a.y=1, o.c=(n.c+1):(n.c+n.y))))
stk.c <- inla.stack(data=list(y=cbind(NA, cc[1:n.c], NA)),
                    A=list(Ac, 1), tag='dat.c',
                    effects=list(m=1:mesh$n,
                        data.frame(a.c=1, w=w[1:n.c])))
stk.0 <- inla.stack(data=list(y=cbind(NA, NA, rep(0, n.c + n.y))),
                    A=list(rBind(Ac,Ay), 1), tag='dat.0',
                    effects=list(m=1:mesh$n,
                        data.frame(a.c=1, w=w[1:(n.c+n.y)],
                                   o=1:(n.c+n.y),
                                   o.weig=rep(-1,n.c+n.y))))
stk <- inla.stack(stk.c, stk.y, stk.0)
```

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

For the estimation of the regression coefficient of c on y we use the copy feature. We set a $N(0, 5)$ prior to β_c when defining the model as

```
form <- y ~  0 + a.c + a.y + w +
    f(m, model=spde) + f(x, model=spde) +
    f(o, o.weig, model='iid',
      hyper=list(theta=list(initial=-20, fixed=TRUE))) +
    f(o.c, copy='o', fixed=FALSE,
      hyper=list(theta=list(param=c(0,5))))
```

and fit the model with

```
pcprec <- list(prior='pcprec', param=c(1, 0.01))
res <- inla(form, data=inla.stack.data(stk), family=rep('gaussian',3),
            control.predictor=list(compute=TRUE, A=inla.stack.A(stk)),
            control.family=list(list(hyper=list(theta=pcprec)),
                list(hyper=list(theta=list(initial=20, fixed=TRUE))),
                list(hyper=list(theta=list(initial=20, fixed=TRUE)))))
```

3.1.4 The results

The true values of the intercepts and the regression coefficient of w on c and the summary of its posterior marginal distributions

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

##     True    mean     sd 0.025quant 0.5quant 0.975quant    mode kld
## a.c -5.0 -4.8424 0.3441    -5.5299  -4.8404    -4.1676 -4.8367   0
## a.y  3.0  1.7832 1.0967    -0.5613   1.8145     3.7797  1.7545   0
## w    0.5  0.2454 0.0791     0.0901   0.2454     0.4008  0.2453   0
```

The true values of the precision of y and the summary of the posterior marginal distribution

```
round(c(True=1/sigma2.y, unlist(res$summary.hy[1,])), 4)

##       True       mean         sd 0.025quant   0.5quant 0.975quant
##     3.3333     2.2633     0.7755     1.1304     2.1335     4.1359
##       mode
##     1.8980
```

The summary for the random field parameters and the regression parameter of c on y is shown by

```
round(cbind(True=c(sqrt(8)/kappa.m, sigma2.m,
                   sqrt(8)/kappa.x, sigma2.x, beta.c),
            res$summary.hyperpar[-1,]), 3)

##                True  mean    sd 0.025quant 0.5quant 0.975quant  mode
## Range for m   0.404 0.195 0.021      0.159    0.194      0.241 0.189
## Stdev for m   3.000 1.627 0.129      1.398    1.618      1.904 1.596
## Range for x   0.283 0.385 0.102      0.219    0.374      0.618 0.353
## Stdev for x   2.000 1.754 0.322      1.195    1.730      2.454 1.687
## Beta for o.c 2.000 1.731 0.174      1.394    1.729      2.075 1.723
```

We see the posterior distribution of regression parameters on Figure 3.1.4 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$marginals.fix[[1]], type='l',
     xlab=expression(alpha[c]), ylab='')
abline(v=alpha.c, col=4)
plot(res$marginals.fix[[2]], type='l',
     xlab=expression(alpha[y]), ylab='')
abline(v=alpha.y, col=4)
plot(res$marginals.fix[[3]], type='l',
     xlab=expression(beta[w]), ylab='')
abline(v=beta.w, col=4)
plot(res$marginals.hy[[6]], type='l',
     xlab=expression(beta[c]), ylab='')
abline(v=beta.c, col=4)
```

We see on the Figure 3.1.4 that the posterior distribution covers the true values of all the parameters.

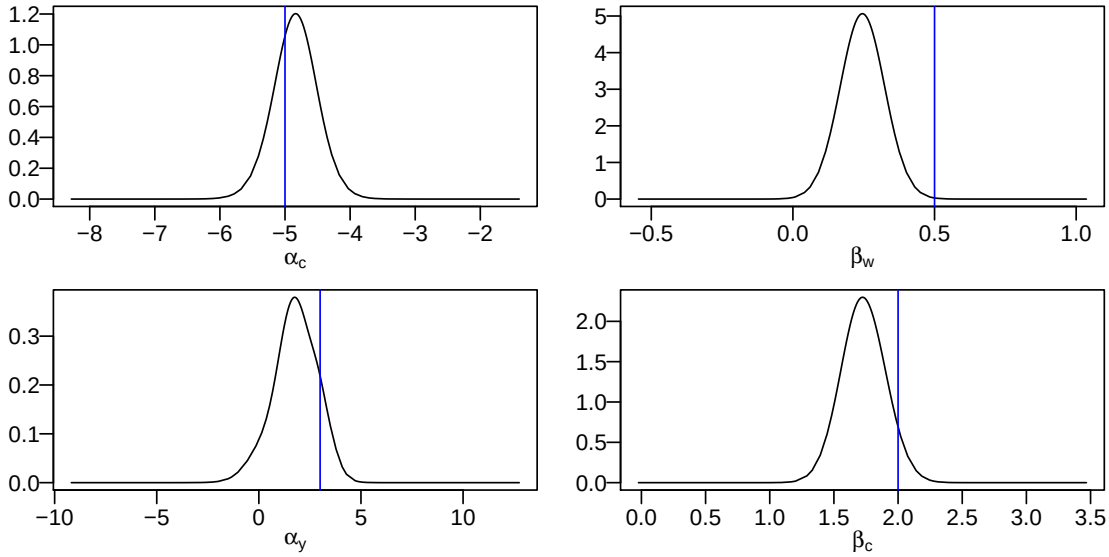

Figure 3.1: Posterior distribution of the likelihood parameters.

The posterior marginals for the random fields is shown in Figure 3.1.4 generated with comands below

```
par(mfcol=c(2,2), mar=c(3,3,.1,.3), mgp=c(1.5,.5,0), las=1)
for (j in 2:5) {
    plot(res$marginals.hyperpar[[j]], type='l',
         xlab=names(res$marginals.hyperpar)[j], ylab='Density')
    abline(v=c(sqrt(8)/kappa.m, sqrt(sigma2.m),
               sqrt(8)/kappa.x, sqrt(sigma2.x), beta.c)[j-1], col=4)
}
```

We see on Figure 3.1.4 that the posterior marginal distribution of the all parameters of both spatial process cover the true values well.

Another interesting result is the prediction of the covariate on the response locations. 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

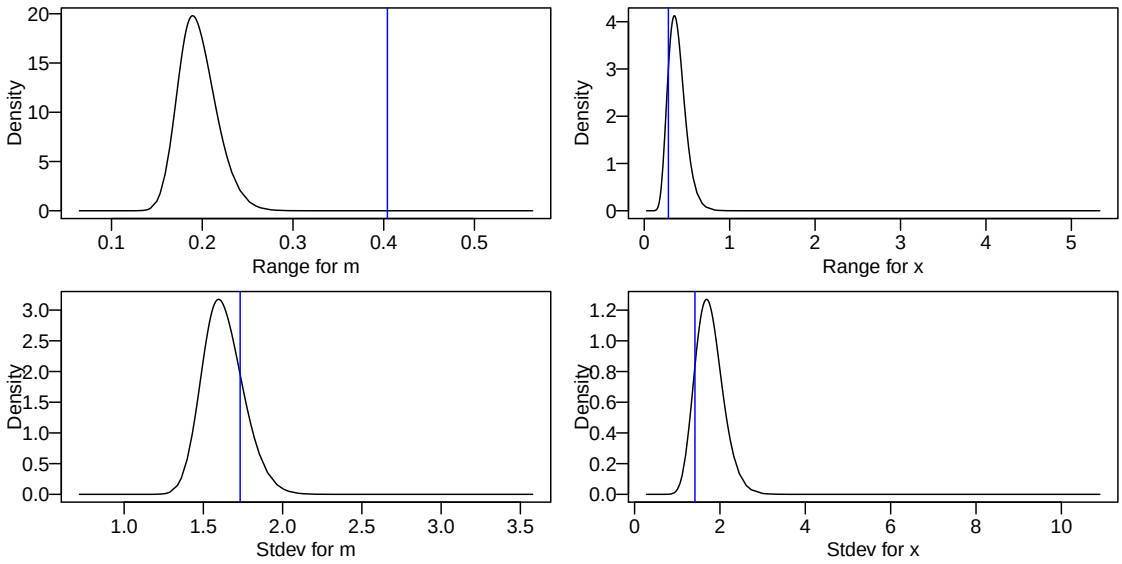

Figure 3.2: Posterior marginal distributions the hyperparameters parameters of both random fields.

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

on the Figure 3.1.4. The blue line represents the situation where predicted is equal to simulated.

3.2 Coregionalization model

The R source code for this example is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-coregionalization.R`

In this Chapter we present a way to fit the Bayesian coregionalization model proposed by [Schimdt and Gelfand, 2003]. A particular case was considered as a covariate joint modeling in Chapter 8 of [Blangiardo and Cameletti, 2015]. Later in this tutorial we do consider a generalization for the space-time case, see Section 5.4. Also, the approach implemented in `R-INLA` allows completelly missalignment for all the outcomes, it only need the same domain.

3.2.1 The model and parametrization

The case of three outcomes is defined considering the following equations

$$y_1(s) = \alpha_1 + z_1(s) + e_1(s)$$
$$y_2(s) = \alpha_2 + \lambda_1 y_1(s) + z_2(s) + e_2(s)$$
$$y_3(s) = \alpha_3 + \lambda_2 y_1(s) + \lambda_3 y_2(s) + z_3(s) + e_3(s)$$

where the $z_k(s)$ are spacetime correlated processes and $e_k(s)$ are uncorrelated error terms, $k = 1, 2, 3$.

In order to fit this model in `R-INLA` we consider a reparametrization. This reparametrization is to change the second equation as follows

$$\begin{aligned} y_2(s) &= \alpha_2 + \lambda_1[\alpha_1 + z_1(s) + e_1(z)] + z_2(s) + e_2(s) \\ &= (\alpha_2 + \lambda_1\alpha_1) + \lambda_1[z_1(s) + e_1(s)] + z_2(s) + e_2(s) \end{aligned}$$

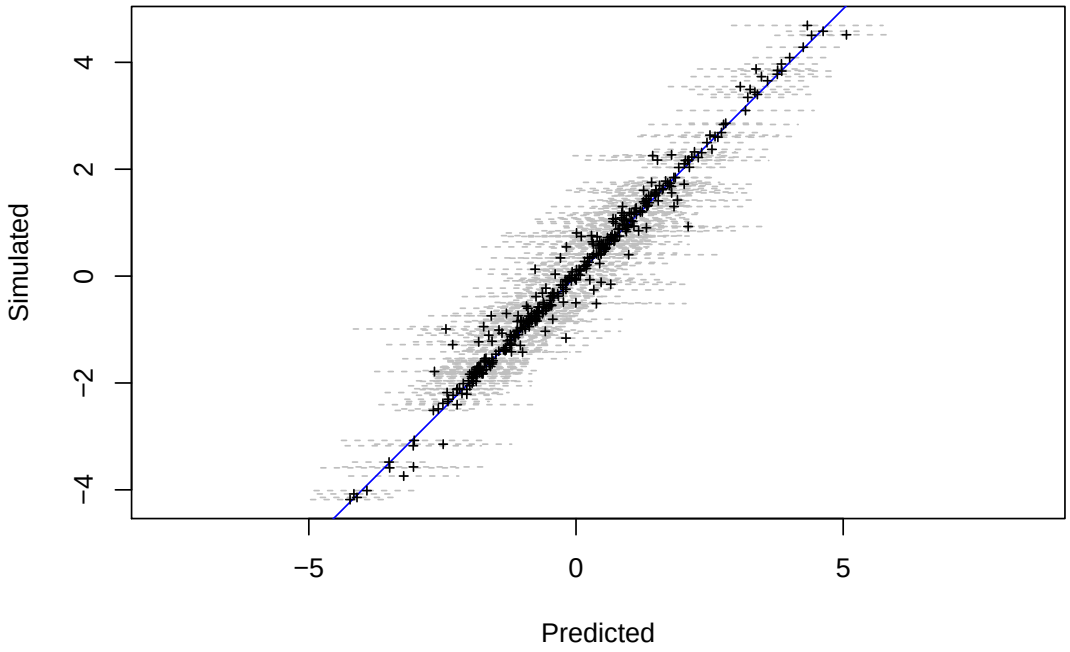

Figure 3.3: Simulated versus predicted values of m (+) and the approximated credibility intervals.

and the third equation as follows

$$\begin{aligned} y_3(s) &= \alpha_3 + \lambda_2(\alpha_2 + \lambda_1\alpha_1) + \lambda_2\lambda_1[z_1(s) + e_1(s)] + \lambda_3\{\alpha_2 + \lambda_1\alpha_1 + \lambda_1[z_1(s) + e_1(s)] + z_2(s) + e_2(s)\} \\ &= [\alpha_3 + \lambda_2\alpha_1 + \lambda_3(\alpha_2 + \lambda_1\alpha_1)] + \\ &\quad (\lambda_2 + \lambda_3\lambda_1)[z_1(s) + e_1(s)] + \lambda_3[z_2(s) + e_2(s)] + z_3(s) + e_3(s) \end{aligned}$$

We have then two new intercepts $\alpha_2^* = \alpha_2 + \lambda_1\alpha_1$ and $\alpha_3^* = \alpha_3 + \lambda_2(\alpha_2 + \lambda_1\lambda_1) + \lambda_3(\alpha_2 + \lambda_1\alpha_1)$. We also have one new regression coeffiecient $\lambda_2^* = \lambda_2 + \lambda_3\lambda_1$.

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.

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 $\lambda_3 = \beta_3$.

3.2.2 Data simulation

Parameter setting

```
alpha <- c(-5, 3, 10) ### intercept on reparametrized model
m.var <- (3:5)/10 ### random field marginal variances
kappa <- c(12, 10, 7) ### GRF scales: inverse range parameters
beta <- c(.7, .5, -.5) ### copy par.: reparam. coregionalization par.
n1 <- 99; n2 <- n1+1; n3 <- n2+1 ### number of spatial locations
```

It is not required to the spatial locations to be the same for each process to fit this model in `R-INLA`. We will consider a different set of locations for each outcome.

```
loc1 <- cbind(runif(n1), runif(n1))
loc2 <- cbind(runif(n2), runif(n2))
loc3 <- cbind(runif(n3), runif(n3))
```

We can use the `rMatern()` function to simulate independent random field realizations for each time. This function is available in the file at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`.

In order to build the second and third outcomes, we do consider samples for the random field in the first outcome at all the locations and for the random field on the second outcome in the second and third set of locations.

```
z1 <- rMatern(1, rbind(loc1, loc2, loc3), kappa[1], m.var[1])
z2 <- rMatern(1, rbind(loc2, loc3), kappa[2], m.var[2])
z3 <- rMatern(1, loc3, kappa[3], m.var[3])
```

Then we define the observation samples

```
e.sd <- c(0.3, 0.2, 0.15)
y1 <- alpha[1] + z1[1:n1] + rnorm(n1, 0, e.sd[1])
y2 <- alpha[2] + beta[1] * z1[n1+1:n2] + z2[1:n2] +
    rnorm(n2, 0, e.sd[2])
y3 <- alpha[3] + beta[2] * z1[n1+n2+1:n3] +
    beta[3] * z2[n2+1:n3] + z3 + rnorm(n3, 0, e.sd[3])
```

3.2.3 Model fitting

We wil build only one mesh to fit all the three spatial random fields. This makes easier to link it across different outcomes at diferent spatial locations. We will use all the locations

```
mesh <- inla.mesh.2d(rbind(loc1, loc2, loc3), ###loc.domain=locd,
                     max.edge=c(0.05, 0.2),
                     offset=c(0.05, 0.3), cutoff=0.01)
```

One can also use only the domain to build the mesh. However, since the sparsity of the resulting total precision matrix also depends on the sparsity of the projector matrices, having the points as nodes in the mesh will gives a projector matrix a bit sparser.

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

This object model can be considered for all the random effects, if we do want to consider the same prior for the hyperparameters.

For each of the the copy parameters we have a $N(0, 10)$ prior distribution

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

Define the formula including all the terms in the model.

```
form <- y ~ 0 + intercept1 + intercept2 + intercept3 +
  f(s1, model=spde) + f(s2, model=spde) + f(s3, model=spde) +
  f(s12, copy="s1", fixed=FALSE, hyper=hc1) +
  f(s13, copy="s1", fixed=FALSE, hyper=hc2) +
  f(s23, copy="s2", fixed=FALSE, hyper=hc3)
```

Define the projector matrix for each set of locations

```
A1 <- inla.spde.make.A(mesh, loc1)
A2 <- inla.spde.make.A(mesh, loc2)
A3 <- inla.spde.make.A(mesh, loc3)
```

Organize the data in three data stack and join it

```
stack1 <- inla.stack(
  data=list(y=cbind(as.vector(y1), NA, NA)), A=list(A1),
  effects=list(list(intercept1=1, s1=1:spde$n.spde)))
stack2 <- inla.stack(
  data=list(y=cbind(NA, as.vector(y2), NA)), A=list(A2),
  effects=list(list(intercept2=1, s2=1:spde$n.spde,
                    s12=1:spde$n.spde)))
stack3 <- inla.stack(
  data=list(y=cbind(NA, NA, as.vector(y3))), A=list(A3),
  effects=list(list(intercept3=1, s3=1:spde$n.spde,
                    s13=1:spde$n.spde,
                    s23=1:spde$n.spde)))
stack <- inla.stack(stack1, stack2, stack3)
```

We consider a penalized complexity prior for the errors precision, [Simspon et al., 2017],

```
eprec <- list(hyper=list(theta=list(prior='pc.prec',
                                    param=c(1, 0.01))))
```

We have two hyperparameters for each spatial effect, one for each likelihood and three copy parameters, which is also considered as hyperparametes. That is 12 hyperparameters in total. To make the optimization process fast, we use the parameter values used in the simulation as the initial values

```
theta.ini <- c(log(1/e.sd^2),
               c(log(sqrt(8)/kappa), log(sqrt(m.var))
                 )[c(1,4, 2,5, 3,6)], beta)
```

We will consider the empirical Bayes approach instead of integrating over the hyperparameters. It is just to avoind to do computations over the 281 configurations of the hyperparameters in the CCD integration strategy. It will saves a bit less of one minute in computational time when using 6 threads. Fitting the model

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

```
##  Pre-processing    Running inla Post-processing           Total
##       1.2334726     111.9322116       0.9429719     114.1086562
```

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

## [1] "Number of function evaluations = 458"

round(result$mode$theta, 2)

##    Log precision for the Gaussian observations
##                                           2.52
## Log precision for the Gaussian observations[2]
##                                           3.45
## Log precision for the Gaussian observations[3]
##                                           4.40
##                                log(Range) for s1
##                                           -1.53
##                                log(Stdev) for s1
##                                           -0.61
##                                log(Range) for s2
##                                           -1.33
##                                log(Stdev) for s2
##                                           -0.41
##                                log(Range) for s3
##                                           -1.17
##                                log(Stdev) for s3
##                                           -0.36
##                             Beta_intern for s12
##                                            0.56
##                             Beta_intern for s13
##                                            0.44
##                             Beta_intern for s23
##                                           -0.19
```

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 -4.75 0.14      -5.02    -4.75      -4.48 -4.75   0
## intercept2    3  3.11 0.20       2.72     3.11       3.50  3.11   0
## intercept3   10  9.90 0.22       9.47     9.90      10.33  9.90   0
```

Posterior marginal for the errors precision

```
round(cbind(true=c(e=e.sd^-2), result$summary.hy[1:3, ]), 4)

##       true     mean       sd 0.025quant 0.5quant 0.975quant    mode
## e1 11.1111  15.2507   7.0640     6.5099  13.6029    33.4431 11.0530
## e2 25.0000  50.0045  40.8152    12.7767  38.0575   157.6448 24.8732
## e3 44.4444 126.2301 114.0612    25.3297  92.8388   428.1342 56.0944
```

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  0.5625 0.1952     0.1812   0.5616     0.9486  0.5585
## Beta for s13  0.5  0.4525 0.2085     0.0493   0.4493     0.8692  0.4379
## Beta for s23 -0.5 -0.2011 0.1803    -0.5602  -0.1994     0.1493 -0.1931
```

Look for the random field parameters for each field. The practical range for each random field

```
round(cbind(true=sqrt(8)/kappa, result$summary.hy[c(4,6,8),]), 3)

##               true  mean    sd 0.025quant 0.5quant 0.975quant  mode
## Range for s1 0.236 0.218 0.058      0.122    0.212      0.350 0.200
## Range for s2 0.283 0.268 0.059      0.170    0.262      0.401 0.251
## Range for s3 0.404 0.329 0.077      0.208    0.318      0.510 0.297
```

The standard deviation for each random field

```
round(cbind(true=m.var^0.5, result$summary.hy[c(5,7,9),]), 3)

##               true  mean    sd 0.025quant 0.5quant 0.975quant  mode
## Stdev for s1 0.548 0.553 0.077      0.419    0.547      0.719 0.535
## Stdev for s2 0.632 0.673 0.098      0.504    0.665      0.886 0.649
## Stdev for s3 0.707 0.710 0.115      0.512    0.700      0.963 0.681
```

We can plot the posterior mean for each random field projected at the data locations. We can see it in Figure 3.2.3. It seems that the method was reazonable well having covered the parameter values used to simulate the data.

```
par(mfrow=c(2,3), mar=c(2.5,2.5,1.5,0.5), mgp=c(1.5,0.5,0))
plot(drop(A1%*%result$summary.ran$s1$mean), z1[1:n1],
     xlab='Posterior mean', ylab='Simulated',
     asp=1, main='z1 in y1'); abline(0:1)
plot(drop(A2%*%result$summary.ran$s1$mean), z1[n1+1:n2],
     xlab='Posterior mean', ylab='Simulated',
     asp=1, main='z1 in y2'); abline(0:1)
plot(drop(A3%*%result$summary.ran$s1$mean), z1[n1+n2+1:n3],
     xlab='Posterior mean', ylab='Simulated',
     asp=1, main='z1 in y3'); abline(0:1)
plot(drop(A2%*%result$summary.ran$s2$mean), z2[1:n2],
     xlab='Posterior mean', ylab='Simulated',
     asp=1, main='z2 in y2'); abline(0:1)
plot(drop(A3%*%result$summary.ran$s2$mean), z2[n2+1:n3],
     xlab='Posterior mean', ylab='Simulated',
     asp=1, main='z2 in y3'); abline(0:1)
plot(drop(A3%*%result$summary.ran$s3$mean), z3[1:n3],
     xlab='Posterior mean', ylab='Simulated',
     asp=1, main='z3 in y3'); abline(0:1)
```

3.3 Copying part or the entire linear predictor

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-linear-predictor-copy.R`

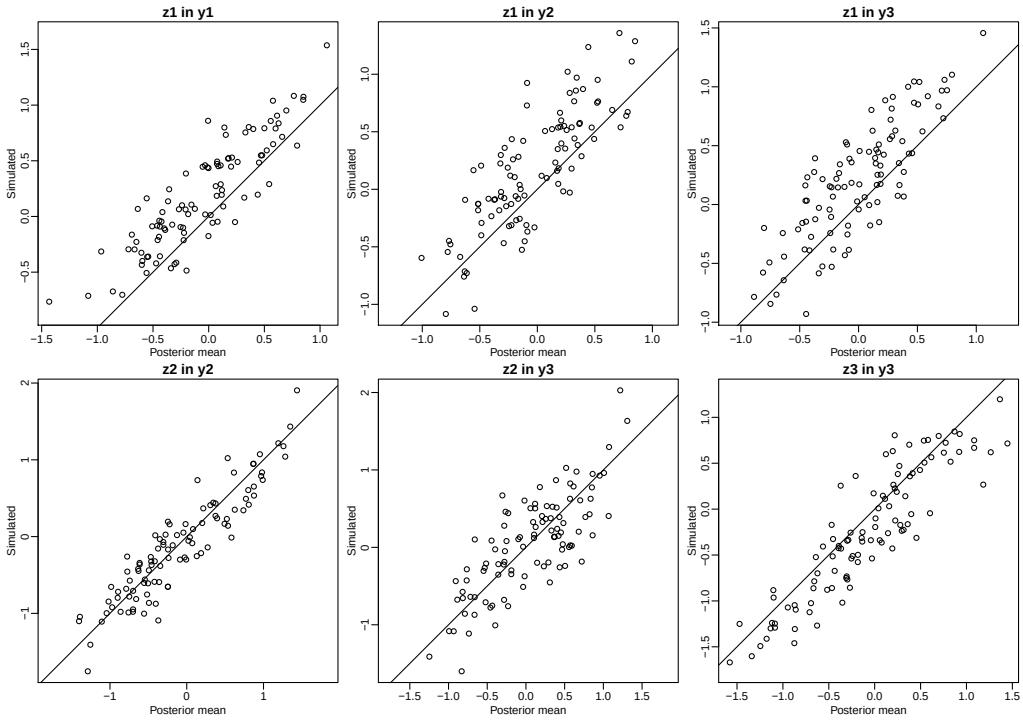

Figure 3.4: Simulated and posterior mean fitted for the random fields.

In this Chapter we show how to copy part of a linear predictor. That is two or more outcomes (or same outcome from different sources) are modeled jointly. In this case there are effects which are common in the linear predictor for more than one outcome.

Suppose we have data collected at the locations $\boldsymbol{s}$, $\boldsymbol{y}1(\boldsymbol{s})$, $\boldsymbol{y}2(\boldsymbol{s})$ and $\boldsymbol{y}3(\boldsymbol{s})$. Consider a case were we have the following three observation models

$$\begin{aligned} \boldsymbol{y}1(\boldsymbol{s}) &= \beta_0 + \beta_1 \boldsymbol{x}(\boldsymbol{s}) + \boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)\boldsymbol{b}(\boldsymbol{s}0) + \boldsymbol{\epsilon}1(\boldsymbol{s}) \quad (3.2) \\ \boldsymbol{y}2(\boldsymbol{s}) &= \beta_2(\beta_0 + \beta_1 \boldsymbol{x}(\boldsymbol{s})) + \boldsymbol{\epsilon}2(\boldsymbol{s}) \quad (3.3) \\ \boldsymbol{y}3(\boldsymbol{s}) &= \beta_3(\beta_0 + \beta_1 \boldsymbol{x}(\boldsymbol{s}) + \boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)\boldsymbol{b}(\boldsymbol{s}0)) + \boldsymbol{\epsilon}3(\boldsymbol{s}) \quad (3.4) \end{aligned}$$

where we have a SPDE model at the mesh nodes $\boldsymbol{b}(\boldsymbol{s}0)$ with $\boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)$ being the projector matrix, $\boldsymbol{\epsilon}j$, j=1,2,3, are observation errors considered as zero mean Gaussian with variance σ_j^2. By this setting we have a linear model for each outcome. We can see that a common effect is scaled from one linear preditor into another, where β_2 and β_3 are the scaling parameters.

We can define the following model terms

- $\boldsymbol{\eta}0(\boldsymbol{s}) = \beta_0 + \beta_1 \boldsymbol{x}(\boldsymbol{s})$
- $\boldsymbol{\eta}1(\boldsymbol{s}) = \boldsymbol{\eta}0(\boldsymbol{s}) + \boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)\boldsymbol{b}(\boldsymbol{s}0)$
- $\boldsymbol{\eta}2(\boldsymbol{s}) = \beta_2 \boldsymbol{\eta}0(\boldsymbol{s})$
- $\boldsymbol{\eta}3(\boldsymbol{s}) = \beta_3 \boldsymbol{\eta}1(\boldsymbol{s})$

By this, this example is about showing how to copy $\boldsymbol{\eta}0$ into $\boldsymbol{\eta}2$ and $\boldsymbol{\eta}1$ into $\boldsymbol{\eta}3$ in order to estimate β_2 and β_3.

We have assumed all the three observation vectors, $\boldsymbol{y}1$, $\boldsymbol{y}2$ and $\boldsymbol{y}3$ to be observed at the same locations.

3.3.1 Generating data

First of all we define the set of parameters in the model. for β_j, j=0,2,...,3 as follows

```
beta0 = 5
beta1 = 1
beta2 = 0.5
beta3 = 2
```

Then, we define the variance errors as

```
s123 <- c(0.1, 0.05, 0.15)
```

For the $b(s)$ process, we consider a Matérn covariance function with κ_b, σ_b^2 and $\nu = 1$ (fixed).

```
kappab <- 10
sigma2b <- 1
```

To realize the process we consider a set of locations as follows

```
n <- 50
loc <- cbind(runif(n), runif(n))
```

and draw one realization of the Matérn process considering the `rMatern()` function available in the file at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`

```
b <- rMatern(n=1, coords=loc, kappa=10, variance=1)
```

This sample can be visualized using

```
par(mar=c(0,0,0,0))
plot(loc, asp=1, cex=0.3+2*(b-min(b))/diff(range(b)),
     pch=19, axes=FALSE); box()
```

Additionally we have to define a covariate. We just sample it as follows

```
x <- runif(n, -1, 1)*sqrt(3)
```

Them we build the linear predictors as follows

```
eta1 <- beta0 + beta1*x + b
eta2 <- beta2*(beta0 + beta1*x)
eta3 <- beta3*eta1
```

and have the observations as follows

```
y1 <- rnorm(n, eta1, s123[1])
y2 <- rnorm(n, eta2, s123[2])
y3 <- rnorm(n, eta3, s123[3])
```

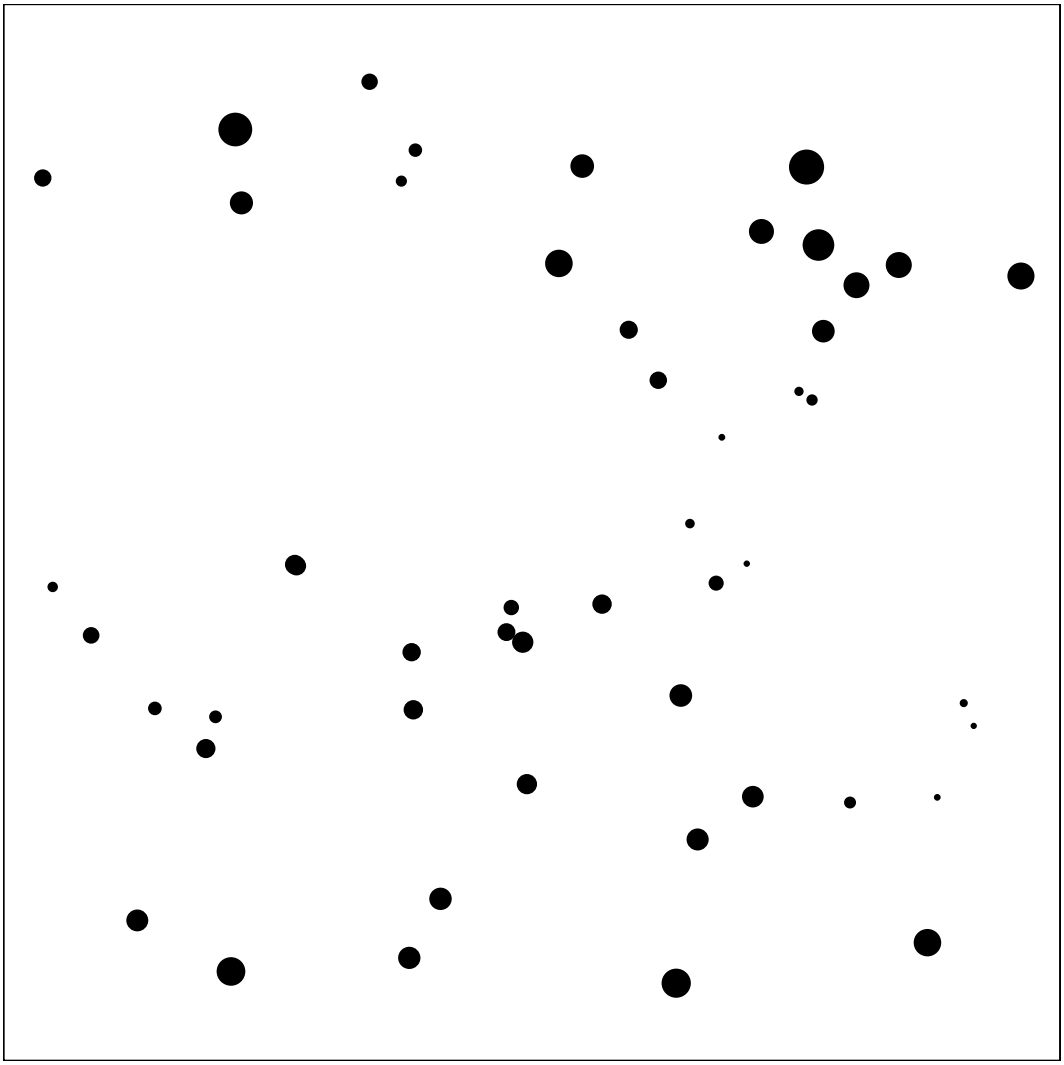

Figure 3.5: Locations

3.3.2 Setting the way to fit the model

There are more than one way to fit this model using the `R-INLA` package. The main point is the need to compute $\boldsymbol{\eta}0(\boldsymbol{s}) = \beta_0 + \beta_1\boldsymbol{x}(\boldsymbol{s})$ and $\boldsymbol{\eta}1(\boldsymbol{s}) = \beta_0 + \beta_1\boldsymbol{x}(\boldsymbol{s}) + \boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)\boldsymbol{b}(\boldsymbol{s})$ from the first observation equation in order to copy it to the second and third observation equation. So, one has to define a model that computes $\boldsymbol{\eta}0(\boldsymbol{s})$ and $\boldsymbol{\eta}1(\boldsymbol{s})$ explicitelly.

The way we choose is to minimize the size of the graph generated by the model, [Rue et al., 2017]. First, we have that

$$\mathbf{0}(\boldsymbol{s}) = \boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)\boldsymbol{b}(\boldsymbol{s}0) + \boldsymbol{\eta}0(\boldsymbol{s}) + \boldsymbol{\epsilon}1(\boldsymbol{s}) - \boldsymbol{y}\mathbf{1}(\boldsymbol{s}) \quad (3.5)$$

$$\mathbf{0}(\boldsymbol{s}) = \boldsymbol{\eta}1(\boldsymbol{s}) + \boldsymbol{\epsilon}1(\boldsymbol{s}) - \boldsymbol{y}1(\boldsymbol{s}) \quad (3.6)$$

where only $\boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)$, $\boldsymbol{x}(\boldsymbol{s})$ and $\boldsymbol{y}1(\boldsymbol{s})$ are known. For the $\boldsymbol{\eta}0(\boldsymbol{s})$ and $\boldsymbol{\eta}2(\boldsymbol{s})$ terms we consider an independent and identically distributed ('iid') model with low fixed precision. With this high fixed hight variance, each element in $\boldsymbol{\eta}0(\boldsymbol{s})$ and $\boldsymbol{\eta}1(\boldsymbol{s})$ can assume any value. However, it will be forced these values to be $\beta_0 + \beta_1\boldsymbol{x}(\boldsymbol{s})$ and $\beta_0 + \beta_1\boldsymbol{x}(\boldsymbol{s}) + \boldsymbol{A}(\boldsymbol{s}, \boldsymbol{s}0)\boldsymbol{b}(\boldsymbol{s}0)$ by considering a Gaussian likelihood for the 'faked zero' observations with a hight fixed precision value (lower fixed likelihood variance). For details and examples of this approach see [Ruiz-Cárdenas et al., 2012], [Martins et al., 2013] and Chapter 8 of [Blangiardo and Cameletti, 2015].

Second, since we have only Gaussian likelihood for the observed data, one can can include $\boldsymbol{\epsilon}j$ (j=1,2,3) in the linear predictor and fix a high likehihood precision. By this setting, we have only Gaussian likelihood with high fixed precision: only one likelihood.

3.3.3 Fitting the model

In order to fit the $b(\boldsymbol{s})$ term we have to set a SPDE Matérn model. For that, we set a mesh

```
mesh <- inla.mesh.2d(loc.domain=cbind(c(0,1,1,0), c(0,0,1,1)),
                     max.edge=c(0.1, 0.3),
                     offset=c(0.05, 0.35), cutoff=0.05)
```

define the projector matrix

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

and the SPDE model

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

to be used.

The data has to be organized using the `inla.stack()` function. The data stack for the first observation vector is just

```
stack1 <- inla.stack(tag='y1',
                     data=list(y=y1),
                     effects=list(
                         data.frame(beta0=1, beta1=x),
                         s=1:spde$n.spde,
                         e1=1:n),
                     A=list(1, As, 1))
```

where the `e1` term will be used to fit $\boldsymbol{\epsilon}1$.

Them the stack data for the first 'facked zero' observations

```
stack01 <- inla.stack(tag='eta1',
                      data=list(y=rep(0,n), offset=-y1),
                      effects=list(
                          s=1:spde$n.spde,
                          list(e1=1:n,
                               eta1=1:n)),
                      A=list(As, 1))
```

which has the negated fist observation vector as offset.
The stack for the second 'facked zero' observation is

```
stack02 <- inla.stack(tag='eta2',
                      data=list(y=rep(0,n), offset=-y1),
                      effects=list(list(e1=1:n, eta2=1:n)),
                      A=list(1))
```

The stack for the second observation vector now considers an index set to compute the $\boldsymbol{\eta}1$ copied from the first 'facked zero' observations.

```
stack2 <- inla.stack(tag='y2',
                     data=list(y=y2),
                     effects=list(list(eta1c=1:n, e2=1:n)),
                     A=list(1))
```

In similar way, we have the third observation stack including an index set to compute the $\boldsymbol{\eta}2$ copied from the second 'facked zero' observations.

```
stack3 <- inla.stack(tag='y3',
                     data=list(y=y3),
                     effects=list(list(eta2c=1:n, e3=1:n)),
                     A=list(1))
```

To fit the model we join all the data

```
stack <- inla.stack(stack1, stack01, stack02, stack2, stack3)
```

The prior distributions will be the default for most of the parameters. For the three variance errors in the observations we have set the PC-prior as follows

```
pcprec <- list(theta=list(prior='pcprec', param=c(0.5, 0.1)))
```

And consider it when defining the model formula

```
formula123 <- y ~ 0 + beta0 + beta1 +
    f(s, model=spde) + f(e1, model='iid', hyper=pcprec) +
    f(eta1, model='iid',
      hyper=list(theta=list(initial=-10, fixed=TRUE))) +
    f(eta2, model='iid',
      hyper=list(theta=list(initial=-10, fixed=TRUE))) +
    f(eta1c, copy='eta1', fixed=FALSE) +
    f(e2, model='iid', hyper=pcprec) +
    f(eta2c, copy='eta2', fixed=FALSE) +
    f(e3, model='iid', hyper=pcprec)
```

The model is them fitted with

```
res123 <- inla(formula123,
               data=inla.stack.data(stack),
               offset=offset,
               control.family=list(list(
                   hyper=list(theta=list(initial=10, fixed=TRUE)))),
               control.predictor=list(A=inla.stack.A(stack)))
```

3.3.4 Model results

We can see the fixed effects β_0 and β_1 with

```
round(cbind(true=c(beta0,beta1),res123$summary.fixed), 4)

##       true   mean     sd 0.025quant 0.5quant 0.975quant   mode   kld
## beta0    5 5.1284 0.1672     4.7778   5.1504     5.3974 5.1427 4e-04
## beta1    1 0.9779 0.0311     0.9125   0.9820     1.0326 0.9855 2e-04
```

The β_2 and β_3 parameters are

```
i.b <- match(paste0('Beta for eta', 1:2, 'c'),
             rownames(res123$summary.hyper))
round(cbind(true=c(beta2, beta3), res123$summary.hy[i.b,]), 4)

##                 true   mean     sd 0.025quant 0.5quant 0.975quant   mode
## Beta for eta1c  0.5 0.4704 0.0084     0.4488   0.4723     0.4845 0.4728
## Beta for eta2c  2.0 2.0038 0.0022     1.9984   2.0042     2.1041 2.0049
```

The posterior marginal distribution of the standard deviation for $\boldsymbol{\epsilon}1$, $\boldsymbol{\epsilon}2$ and $\boldsymbol{\epsilon}1$ can be visualized with

```
i.e123 <- match(paste0('Precision for e', 1:3),
                names(res123$marginals.hyper))
par(mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(1.5, 0.5, 0))
for (j in 1:3) {
    plot(inla.tmarginal(function(x) 1/sqrt(x),
                        res123$marginals.hyperpar[[i.e123[j]]]),
         type='l', lwd=2, xlab=bquote(sigma[.(j)]^2),
         ylab='Posterior marginal density')
    abline(v=s123[j], lwd=2)
}
```

Finally, the random field parameters posterior marginals can be visualized with

```
rfparams <- inla.spde2.result(res123, 's', spde)
par(mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(1.5, 0.5, 0))
for (j in 15:17) {
    plot(rfparams[[j]][[1]], xlab=names(rfparams)[j],
         type='l', lwd=2, ylab='Posterior marginal density')
    abline(v=c(10, 1, sqrt(8)/10)[j-14], lwd=2)
}
```

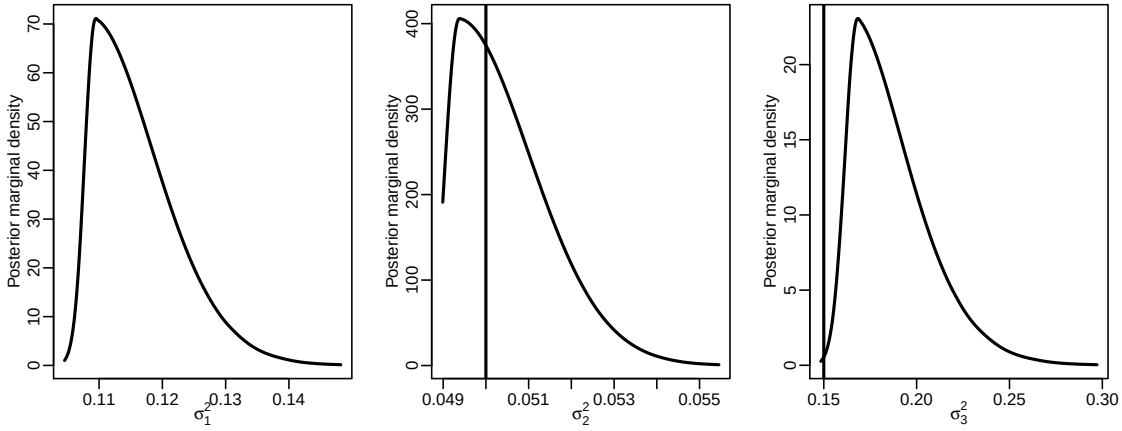

Figure 3.6: Observation error standard deviations.

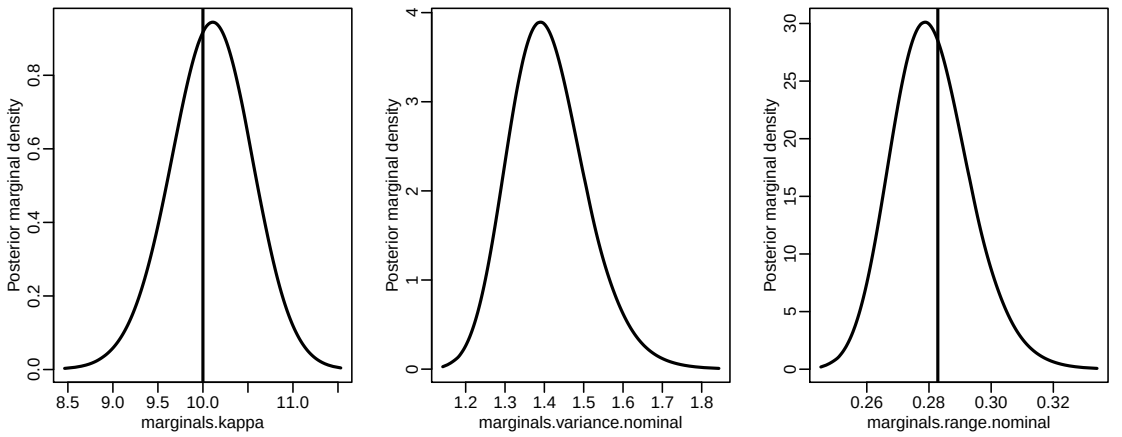

Figure 3.7: Posterior marginal distributions for the random field parameters

Chapter 4

Log-Cox point process and preferential sampling

The R source for the three examples in this Chapter is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-prefsampl.R`

4.1 The [Simpson et al., 2016] approach example

Under the log-Cox model assumption, there is a latent Gaussian Random Field (LGRF) and the inference can be done using **INLA**, [Møller et al., 1998]. 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, [Møller and Waagepetersen, 2003]. One can consider this counts and a Poisson likelihood conditional to a LGRF and use INLA to fit the model, [Illian et al., 2012].

An approach considering the SPDE approach was considerd in [Simpson et al., 2016]. This approach has a nice theoretical justification and considers a direct approximation of the log-Cox point process model likelihood. The data is modeled considering its exact location instead of binning it into cells. Along with the flexibility for defining a mesh, it can handle non-rectangular areas avoiding to waste computational effort.

4.1.1 Data simulation

We will use the data simulated here later in Section 4.2 and in Section 4.3. To sample from a log-Cox point process we will use the `rLGCP()` function from the **spatstat R** package. 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.

```
library(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 Shapter 4.2.

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

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

Doing the simulation

```
library(RandomFields)
set.seed(1)
lg.s <- rLGCP('matern', beta0,
              var=sigma2x, scale=1/kappa, nu=1, win=win)
```

Both the LGRF and the point pattern are returned. We collect the point pattern locations coordinates with

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

and the number of points is

```
(n <- nrow(xy))

## [1] 186
```

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 4.1.1 produced with the following commands

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

4.1.2 Inference

Following [Simpson et al., 2016] we can estimate the parameters of the log-Cox point process model using few command lines.

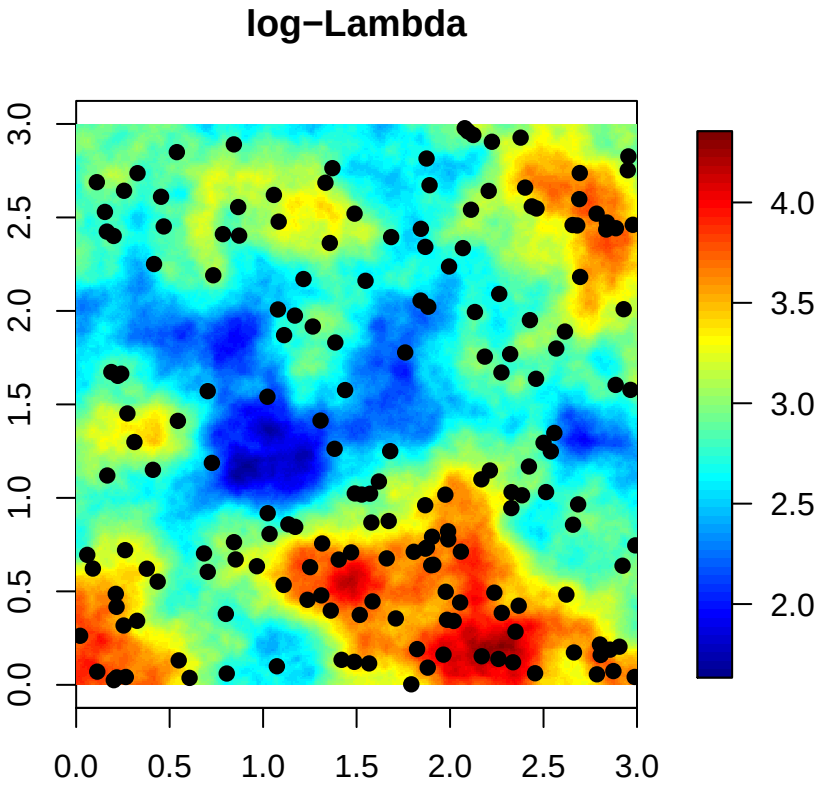

Figure 4.1: Simulated intensity of the point process (left), simulated point pattern (right).

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 1.2.7. 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.domain=loc.d, offset=c(.3, 1),
            max.edge=c(.3, 0.7), cutoff=.05))$n)

## [1] 485
```

which is visualized at Figure 4.1.2 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 considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

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. The method in [Simpson et al., 2016] defines the expected number of events to be proportional to the polygons volume of the dual mesh. 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 for every mesh node. However, the mesh has nodes out of the domain implying

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

## [1] 30.06906
```

to be larger than the domain area.

We can use these values for the nodes on the inner domain and compute the intersection between the dual mesh polygons and the study domain polygon. To do that we have the function `inla.mesh.dual()`, available in `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R` This function considers a mesh and return the dual mesh in the `SpatialPolygons` class.

```
dmesh <- inla.mesh.dual(mesh)
```

We can see it in Figure 4.1.2.

We can convert the domain polygon into a `SpatialPolygons` class with

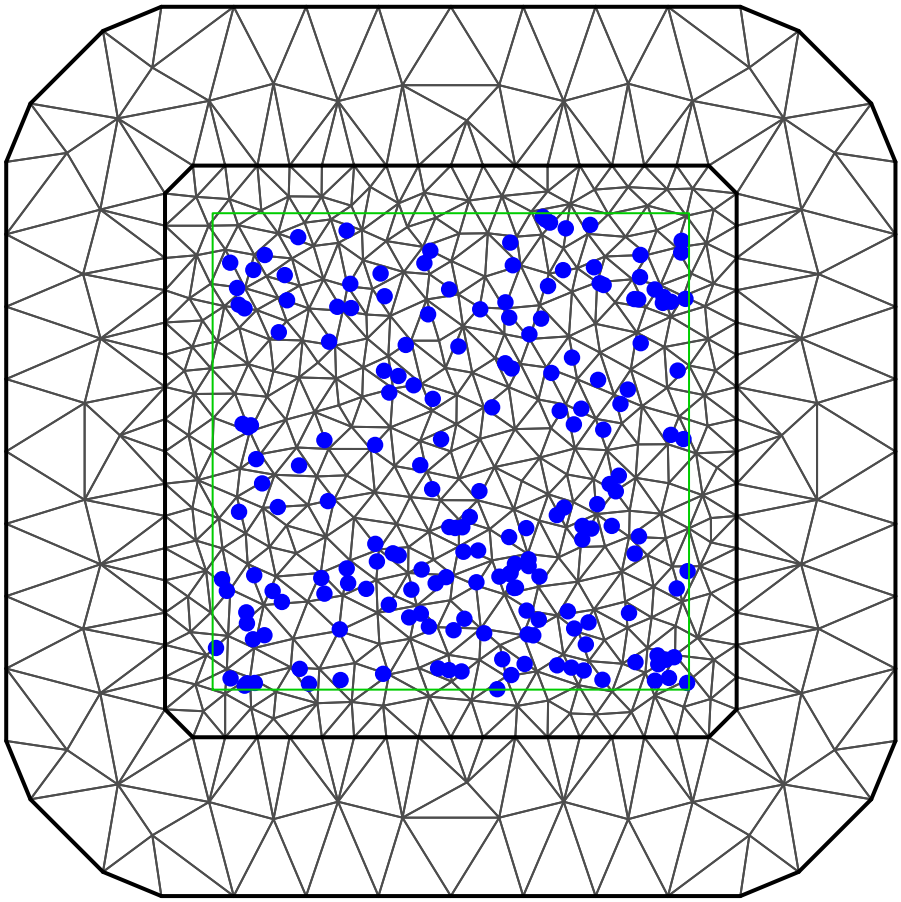

Figure 4.2: Mesh used to inference for the log-Cox process.

```
domainSP <- SpatialPolygons(list(Polygons(
    list(Polygon(loc.d)), '0')))
```

Them, we can have the intersection with each polygon from the mesh dual using the functions `gIntersection()` from the **rgeos** package

```
library(rgeos)
sum(w <- sapply(1:length(dmesh), function(i) {
    if (gIntersects(dmesh[i,], domainSP))
        return(gArea(gIntersection(dmesh[i,], domainSP)))
    else return(0)
}))

## [1] 9
```

And we chan check that there are integration points with zero weight.

```
table(w>0)

##
## FALSE  TRUE
##   197   288
```

These are identified in red in Figure 4.1.2.

```
par(mar=c(2,2,1,1), mgp=2:0)
plot(mesh$loc, asp=1, col=(w==0)+1, pch=19, xlab='', ylab='')
plot(dmesh, add=TRUE)
lines(loc.d, col=3)
```

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.

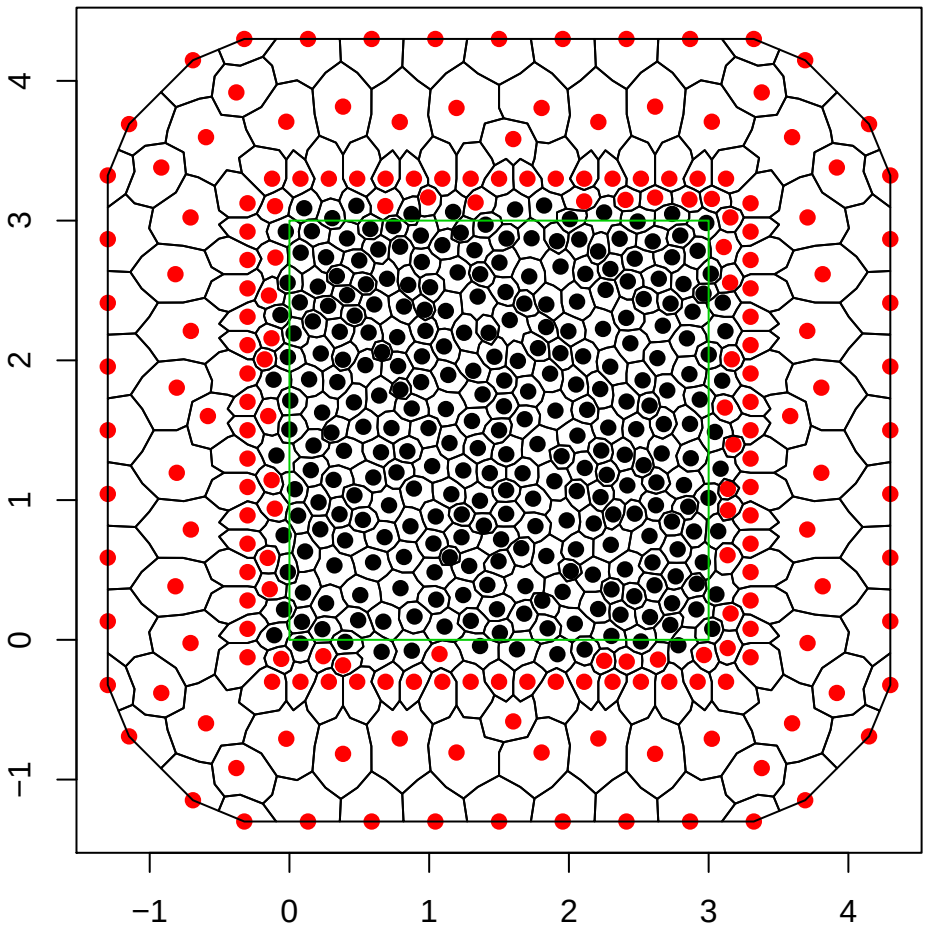

Figure 4.3: Voronoy polygons for the mesh used to inference for the log-Cox process.

```
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)))
```

Posterior marginals

The posterior marginals for the parameters are obtained with

```
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)
```

We can see the summary for the practical range and standard deviation of the lateng Gaussian random field with

```
round(pp.res$summary.hyperpar, 4)
```

```
##                mean     sd 0.025quant 0.5quant 0.975quant   mode
## Range for i 2.7830 2.4332     0.5556   2.0805     9.1818 1.2626
## Stdev for i 0.3069 0.1117     0.1339   0.2926     0.5660 0.2628
```

The posterior distribution of the log-Cox model parameters are visualized on the Figure 4.1.2.

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

4.2 Including a covariate on the log-Cox process

In the previous example 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.

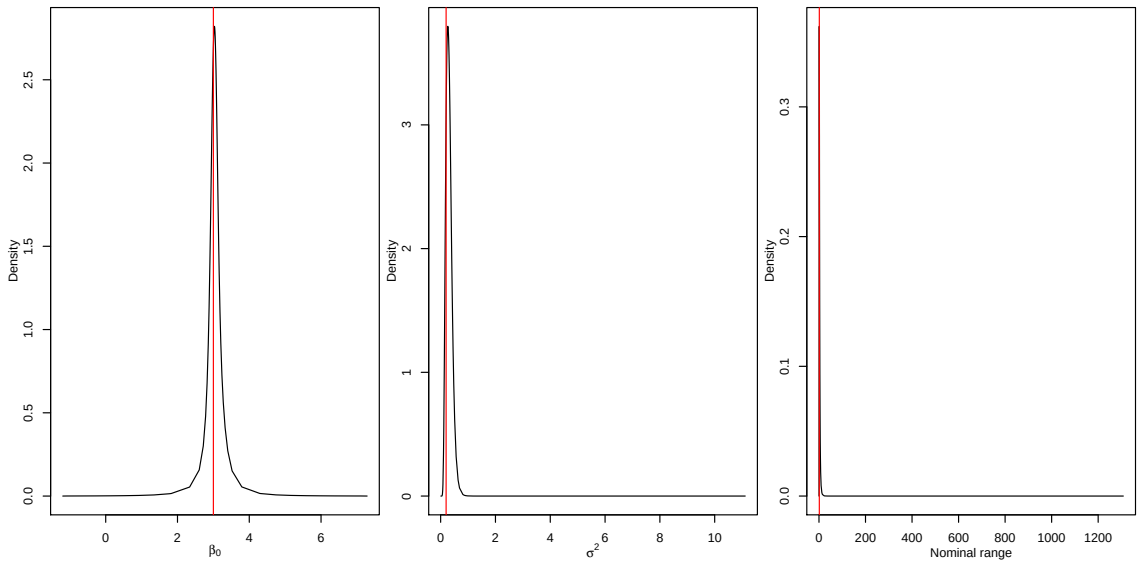

Figure 4.4: Posterior distribution for the parameters of the log-Cox model σ^2 (left), κ (mid) and the nominal range (right)

4.2.1 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),
              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] 174
```

We can see the covariate values and the simulated LGRF over the grid in Figure 4.2.1 with the following commands

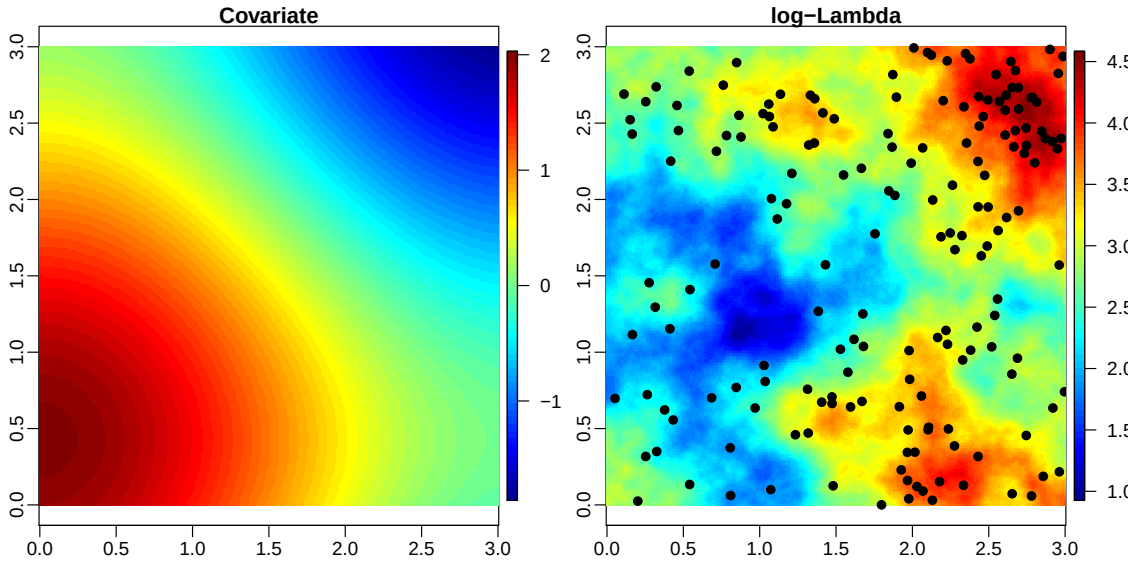

Figure 4.5: Covariate (left), simulated intensity of the point process (mid), simulated point pattern (right).

```
library(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)
```

4.2.2 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)))
```

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)
```

Summary of the model parameters

```
round(pp.c.res$summary.hyperpar, 4)

##                mean     sd 0.025quant 0.5quant 0.975quant   mode
## Range for i 1.9827 1.3095     0.5612   1.6386     5.4285 1.1653
## Stdev for i 0.3984 0.1299     0.1936   0.3826     0.6989 0.3519
```

The posterior distribution of the log-Cox model parameters are visualized on the Figure 4.2.2.

```
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.c.res$marginals.hyperpar[[2]], type='l', ylab='Density',
     xlab=expression(sigma)); abline(v=sigma2x^0.5, col=2)
plot(pp.c.res$marginals.hyperpar[[1]], type='l', ylab='Density',
     xlab=expression(sqrt(8)/kappa)); abline(v=kappa, col=2)
```

4.3 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.

This approach assumes a linear predictor for the point process as

$$\beta_0 + \text{random field}$$

and for the observations a linear predictor which is function of the latent Gaussian random field for the point process

$$\beta_0^y + \beta\text{random field}$$

where β_0^y is an intercept for the observations and β is a sharing parameter.

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

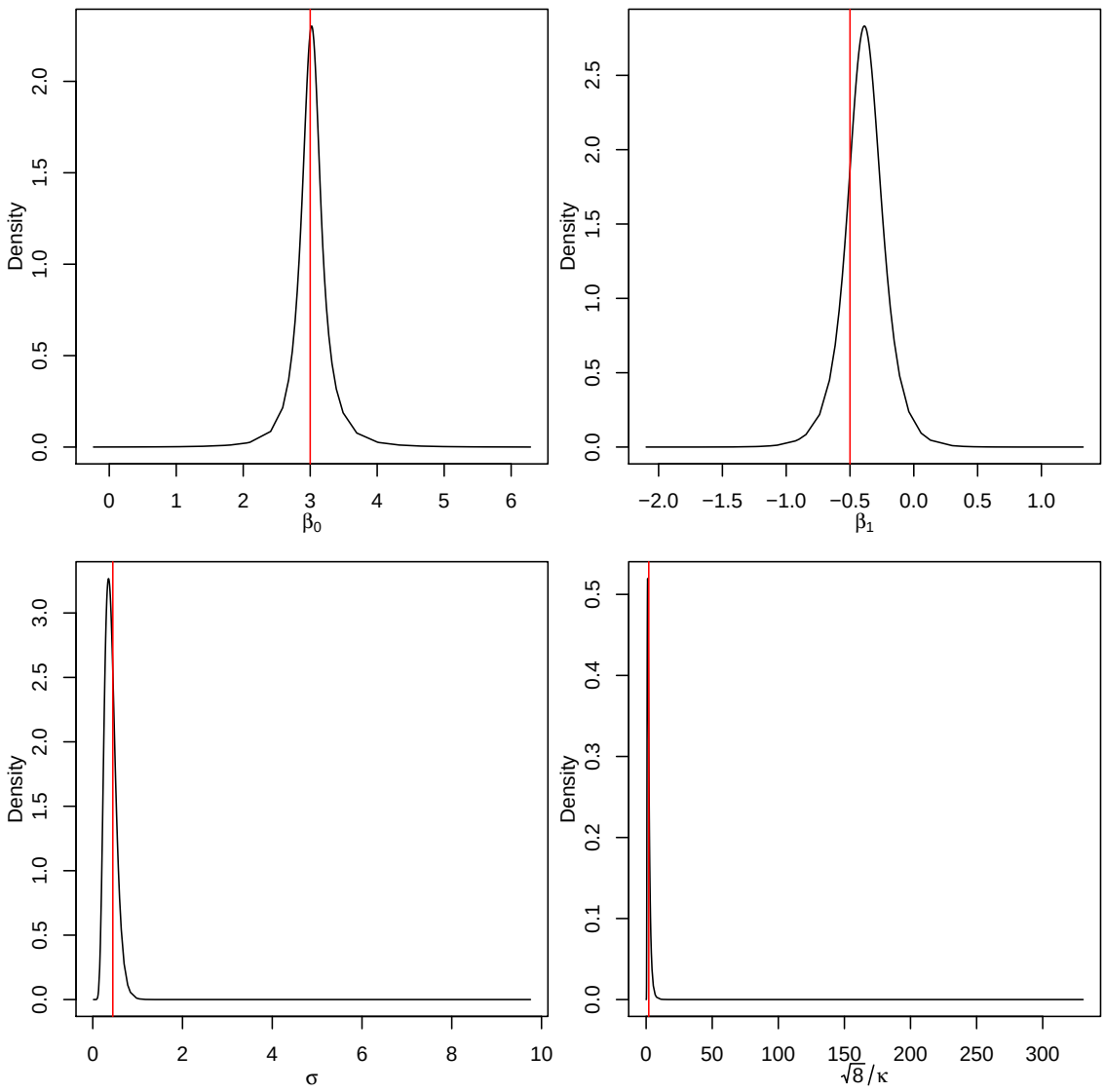

Figure 4.6: Posterior distribution for the intercept (top left), coefficient of the covariate (top right) and the parameters of the log-Cox model σ^2 (bottom left), κ (bottom right)

for the latent random field. Here, we show geoestatistical inference under preferencial sampling using SPDE.

We now will use the values of the latent GRF considered in the simulation of the point process to define an outcome at the location points. 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.
##   2.069   2.833   3.093   3.170   3.580   4.216
```

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.

```
beta0.y <- 10;   beta <- -2;   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.
##   8.900   9.626   9.929   9.919  10.240  10.710
```

Considering $\beta < 0$, it means that the response values is inversly proportional to the point process density.

4.3.1 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.

```
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  9.9163 0.1556     9.5712    9.923    10.2201  9.9315
## prec.y   16 14.1906 1.7138    11.0600   14.113    17.7855 13.9836
```

We can see the posterior distribution marginals for the model parameters in the Figure 4.3.1, produced with the following code

```
par(mfrow=c(1,3), mar=c(3, 3, 0.3, 0.3), mgp=c(2,1,0))
plot(inla.tmarginal(function(x) sqrt(1/x),
                    u.res$marginals.hyperpar[[1]]),
     type='l', ylab='Density', xlab=expression(sigma))
abline(v=prec.y^-0.5, col=2)
```

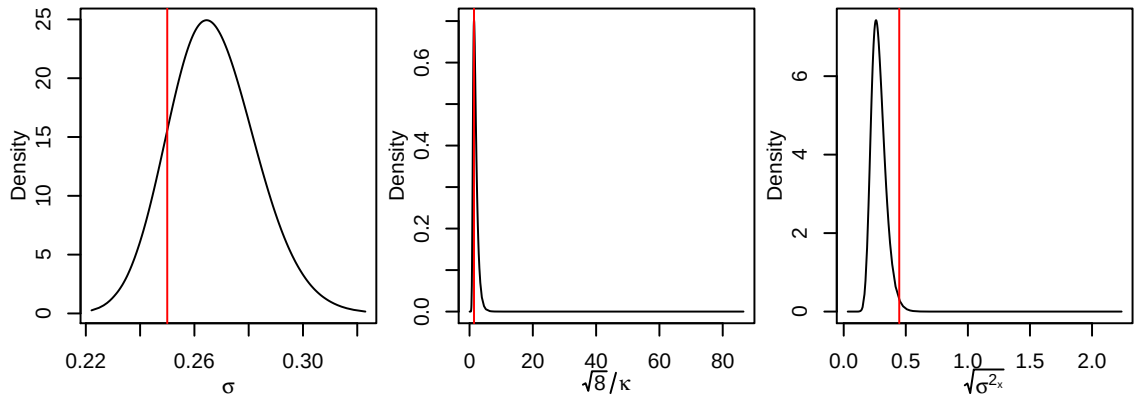

Figure 4.7: Posterior distribution for σ^2, κ and the nominal range just using the response.

```
plot(u.res$marginals.hyperpar[[2]], type='l', ylab='Density',
     xlab=expression(sqrt(8)/kappa)); abline(v=sqrt(8)/kappa, col=2)
plot(u.res$marginals.hyperpar[[3]], type='l',
     xlab=expression(sqrt(sigma^2[x])), ylab='Density')
abline(v=sigma2x^0.5, col=2)
```

4.3.2 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 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)))
```

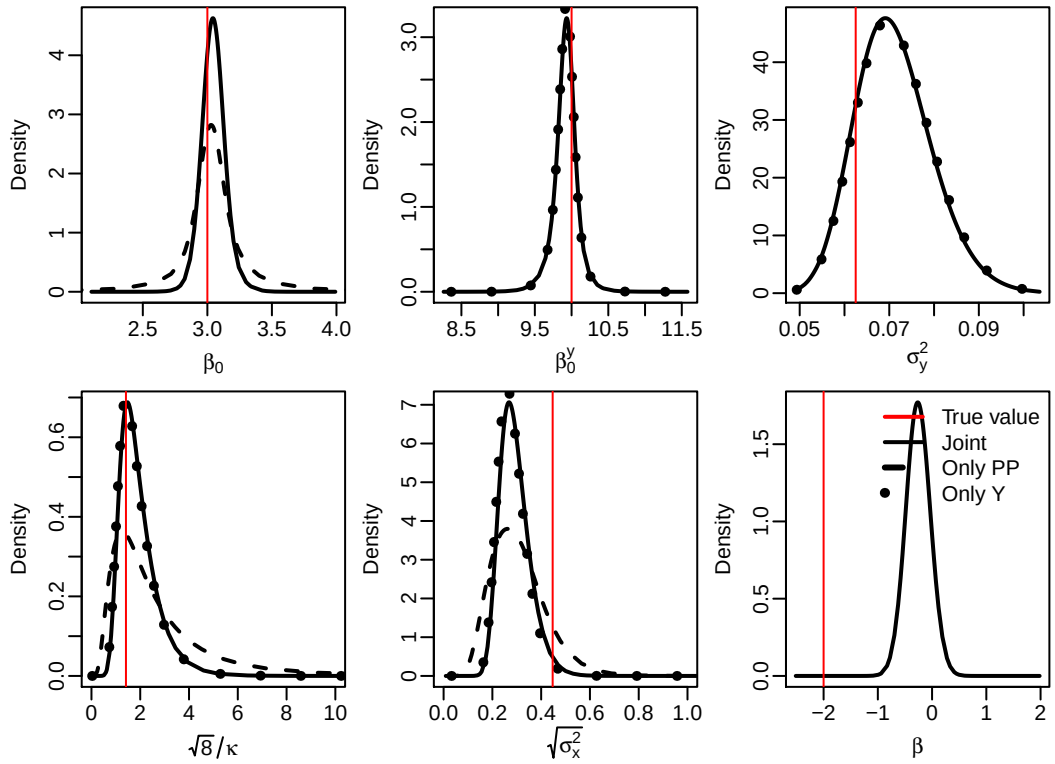

Figure 4.8: Posterior marginal distribution for: intercept for the point process β_0, intercept for the observations β_0^y, noise variance in the observations σ_y^2, the practical range parametrized as $\sqrt{8}/\kappa$, the marginal standard deviation for the random field σ_x^2 and the sharing coefficient β.

```
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 3.0468 0.0945     2.8660   3.0448     3.2436 3.0430   0
## b0.y    10 9.9178 0.1660     9.5548   9.9246    10.2402 9.9334   0
```

We can visualize the posterior marginal distributions for the model parameters from the result considering only the point process (PP), only the observations/marks ($\mathbf{Y}$) and jointly in Figure 4.3.2. Notice that for the β_0 parameter we only have results considering only the PP and joint, for β_o^y we only have results considering only $\mathbf{Y}$ and joint, and for β (fitted using copy) we only have result from the joint model. Only for the random field parameters we do have results from the three results.

Chapter 5

Space-time examples

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-spacetime.R`

In this chapter we show an example on fitting a space-time 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 an AR(1) model for the time dimention. The space-time separable model is defined by the kronecker product between the precision of these two models.

We provide two examples, one for discrete time domain and another when the time is discretized over a set of knots. Basically the difference appears only in the simulation process, wich is not that important. The main difference in the fitting process is that in the continuous time case we have to select time knots and build the projector matrix considering it. However, both cases allows to have different locations at different times.

5.1 Discrete time domain

In this section we show how to fit a space-time separable model, as in [Cameletti et al., 2012]. Additionally, we show the use of a categorical covariate.

5.1.1 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 in Section 1.3.

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. This last option is easy as we can simulate one realization of a spatial random field for each time.

First we set $k = 12$, the time dimention

```
k <- 12
```

and consider the location points from the `PRprec` data in a random order

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

In the following simulation step we will use the `rspde()` function available in the file at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`.

The k independent realizations can be done by

```
params <- c(variance=1, kappa=1)
set.seed(1)
x.k <- rspde(coords, kappa=params[2], variance=params[1], 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 correlated sample over time using

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

where the $\sqrt{(1-\rho^2)}$ term is added as we would like to consider that the innovation noise follows the stationary distribution, see [Rue and Held, 2005] and [Cameletti et al., 2012].

We can visualize the realization at the figure 5.1.1 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

```
sd.y <- 0.1
y <- beta[unclass(ccov)] + x + rnorm(n*k, 0, sd.y)
tapply(y, ccov, mean)

##          A          B          C
## -1.4042972 -0.4275658  0.5918728
```

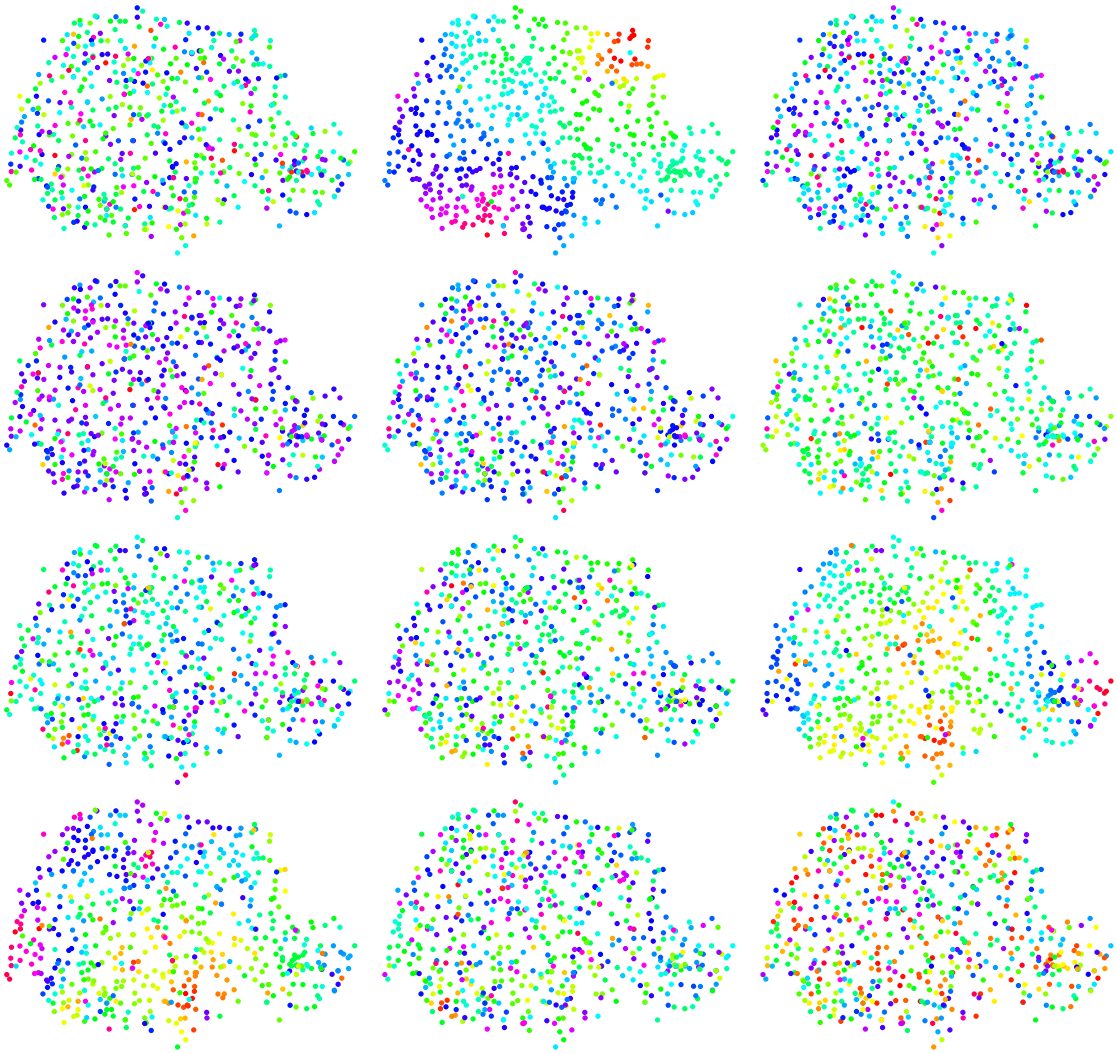

Figure 5.1: Realization of the space-time random field.

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.

5.1.2 Data stack preparation

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=prmesh1, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.5, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

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

```
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))
```

5.1.3 Fitting the model and some results

We set the PC-prior for the temporal autoregressive parameter with $P(cor > 0) = 0.9$

```
h.spec <- list(theta=list(prior='pccor1', param=c(0, 0.9)))
```

The likelihood hyperparameter is fixed on a hight precision, just because we haven't noise. 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))
prec.prior <- list(prior='pc.prec', param=c(1, 0.01))
res <- inla(formulae,  data=inla.stack.data(sdat),
            control.predictor=list(compute=TRUE, A=inla.stack.A(sdat)),
            control.family=list(hyper=list(theta=prec.prior)),
            control.fixed=list(expand.factor.strategy='inla'))
```

Summary for the trhee intercepts (and 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  -1.4466 -1.4423 0.4908    -2.4140  -1.4420    -0.4740 -1.4413   0
## B  -0.4021 -0.4386 0.4908    -1.4104  -0.4383     0.5297 -0.4376   0
## C   0.6265  0.5643 0.4908    -0.4075   0.5646     1.5326  0.5653   0
```

Look a the posterior marginal distributions for the random field parameters and the marginal ditribution for the temporal correlation, on the Figure 5.1.3 with the commands bellow

```
par(mfrow=c(2,2), mar=c(3,3,1,0.1), mgp=2:0)
for (j in 1:4) {
    plot(res$marginals.hyper[[j]], type='l',
         xlab=names(res$marginals.hyper)[j], ylab='Density')
    abline(v=c(1/sd.y^2, sqrt(8)/params[1],
               params[2]^0.5, rho)[j], col=2)
}
```

5.1.4 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.

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 ...
```

The correlation between the simulated data response and the posterior mean of the predicted values (there is no error term in the model):

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

## [1] 0.998647
```

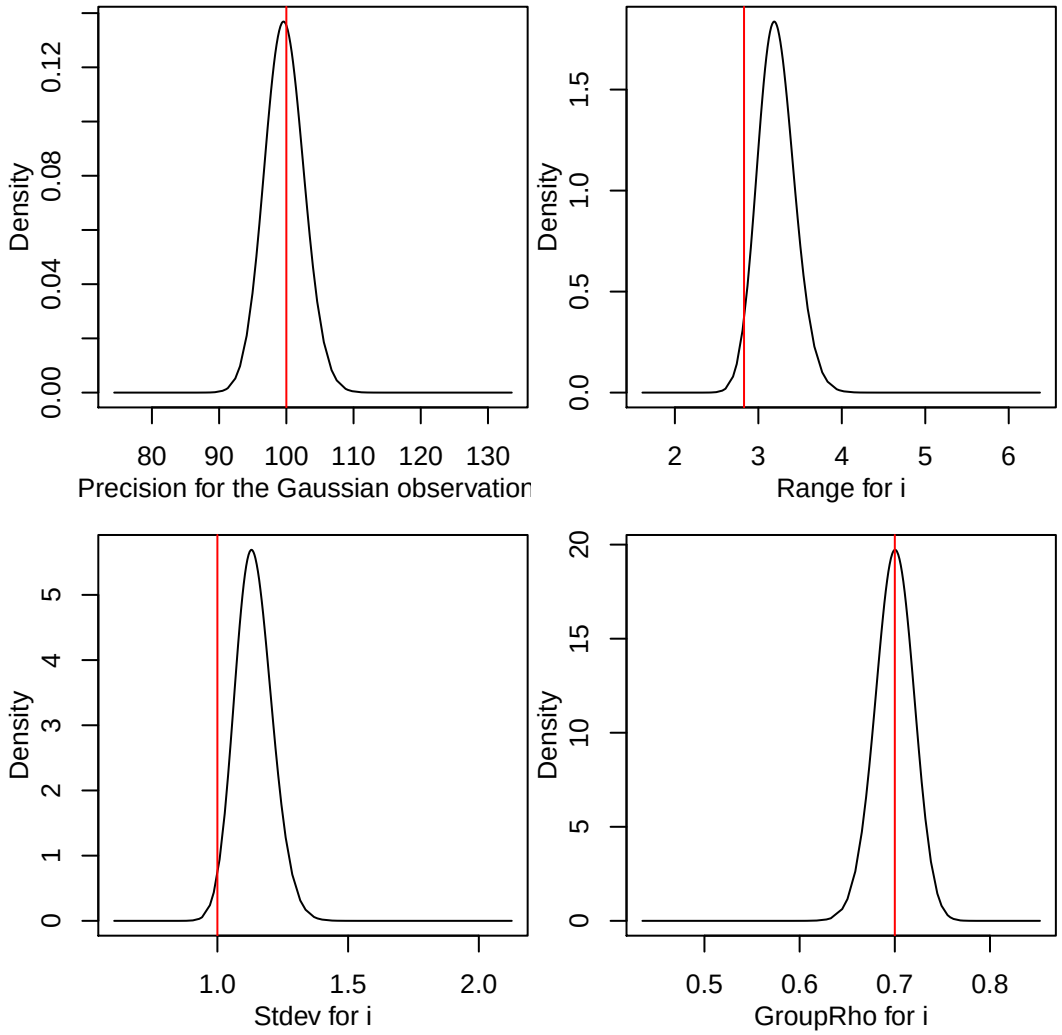

Figure 5.2: Marginal posterior distribution for the practical range (left), standard deviation of the field (mid) and the temporal corrlation (right). The red vertical lines are placed at true value.

We also can do prediction for each time and visualize it. First, we define the grid in the same way as in the rainfall example in Section 2.1.

```
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])
```

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

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

To plot, 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 5.1.4 can be made by the comands bellow

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

5.1.5 Validation

The results on previous section are done using part of the simulated data. This part of the simulated data is now used as a validation data. So, we prepare another data stack to compute posterior distributions to this part of the data:

```
vdat <- data.frame(r=as.vector(y), w=ccov, t=rep(1:k, each=n),
                   x=rep(coords[,1], k), y=rep(coords[,2], k))[-isel, ]
Aval <- inla.spde.make.A(prmesh1, loc=cbind(vdat$x, vdat$y), group=vdat$t)
stval <- inla.stack(tag='stval', data=list(y=NA), ### set NA in order to pre-
dict
                    A=list(Aval,1),  effects=list(iset, w=vdat$w))
```

Now, we just use a full data stack to fit the model and consider the hyperparameters values fitted before

```
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(hyper=list(theta=prec.prior)),
             control.fixed=list(expand.factor.strategy='inla'),
             control.mode=list(theta=res$mode$theta, restart=FALSE))
```

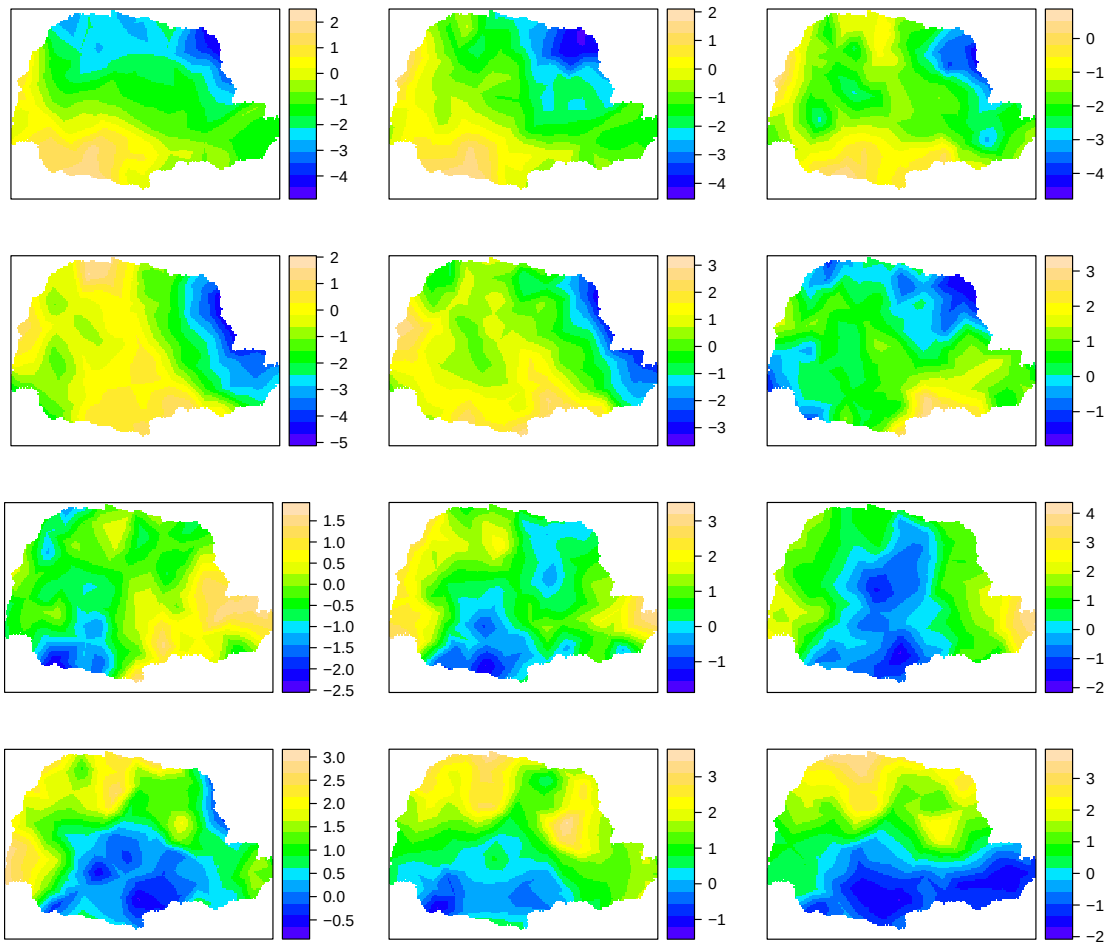

Figure 5.3: Visualization of the posterior mean of the space-time random field.

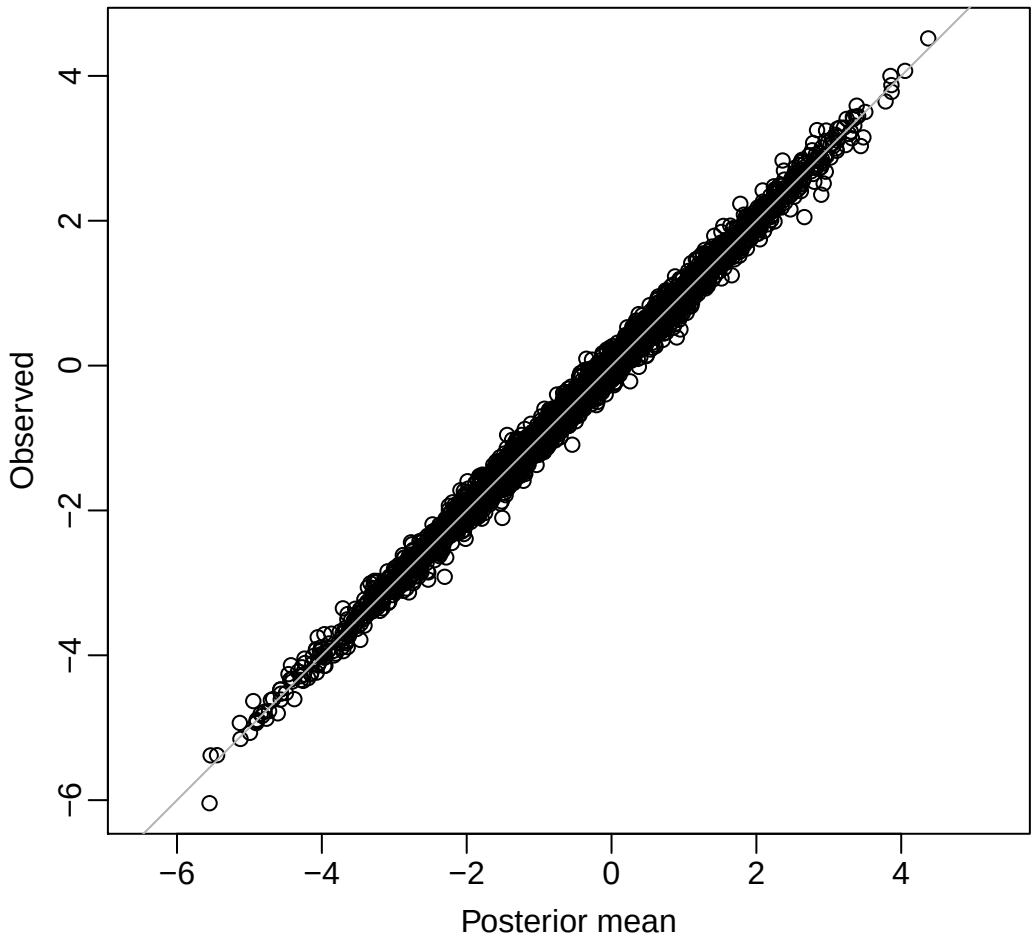

Figure 5.4: Validation: Observed versus posterior mean.

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 5.1.5.

```
par(mfrow=c(1,1), mar=c(3,3,0.5,0.5), mgp=c(1.75,0.5,0))
plot(vres$summary.fitted.values$mean[ival], vdat$r,
     asp=1, xlab='Posterior mean', ylab='Observed')
abline(0:1, col=gray(.7))
```

5.2 Continuous time domain

We now suppose that we have that the observations are not collected over discrete time points. This is the case for fishing data and space-time point process in general. Similar to the Finite Method approach for the space, we can use piecewise linear basis function at a set of time knots, as we have in some other spacetime examples.

5.2.1 Data simulation

We now sample some locations over space and time points as well.

```
n <- nrow(loc <- unique(as.matrix(PRprec[,1:2])))
time <- sort(runif(n, 0, 1))
```

To sample from the model, we define a space-time separable covariance function, which is Matérn in space and Exponential over time:

```
stcov <- function(coords, time, kappa.s, kappa.t, variance=1, nu=1) {
    s <- as.matrix(dist(coords))
    t <- as.matrix(dist(time))
    scorr <- exp((1-nu)*log(2) + nu*log(s*kappa.s) - lgamma(nu)) *
        besselK(s*kappa.s, nu)
    diag(scorr) <- 1
    return(variance * scorr * exp(-t*kappa.t))
}
```

and use it to sample from the model

```
kappa.s <- 1; kappa.t <- 5; s2 <- 1/2
xx <- crossprod(chol(stcov(loc, time, kappa.s, kappa.t, s2)),
                rnorm(n))
beta0 <- -3; tau.error <- 3
y <- beta0 + xx + rnorm(n, 0, sqrt(1/tau.error))
```

5.2.2 Data stack preparation

To fit the space-time continuous model we need to determine the time knots and the temporal mesh

```
k <- 10
(mesh.t <- inla.mesh.1d(seq(0+0.5/k, 1-0.5/k, length=k)))$loc

##  [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95
```

Consider the the low resolution mesh for Paraná state border created in Section 1.3, used in the previous example and the SPDE model also defined in the previous example.

Building the index set

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

The projector matrix consider the spatial and time projection. So, it needs the spatial mesh and the spatial locations, the time points and the temporal mesh

```
A <- inla.spde.make.A(mesh=prmesh1, loc=loc,
                      group=time, group.mesh=mesh.t)
```

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=y),
                   A=list(A,1),  effects=list(iset, list(b0=rep(1,n))))
```

5.2.3 Fitting the model and some results

We used an Exponential correlation function for time with parameter κ as the inverse range parameter. It gives a correlation between time knots equals to

```
exp(-kappa.t*diff(mesh.t$loc[1:2]))

## [1] 0.6065307
```

Fitting the model considering a AR1 temporal correlation over the time knots

```
formulae <- y ~ 0 + b0 +
    f(i, model=spde, group=i.group,
      control.group=list(model='ar1', hyper=h.spec))
res <- inla(formulae, data=inla.stack.data(sdat),
            control.family=list(hyper=list(theta=prec.prior)),
            control.predictor=list(compute=TRUE, A=inla.stack.A(sdat)))
```

Look at the summary of the posterior marginal distributions for the likelihood precision and the random field parameters:

```
round(res$summary.hyper, 4)

##                                             mean     sd 0.025quant 0.5quant
## Precision for the Gaussian observations 2.8535 0.1903     2.4917   2.8496
## Range for i                             2.4527 0.4363     1.7246   2.4070
## Stdev for i                             0.6753 0.0725     0.5423   0.6721
## GroupRho for i                          0.5124 0.1464     0.1879   0.5270
##                                         0.975quant   mode
## Precision for the Gaussian observations     3.2401 2.8449
## Range for i                                 3.4360 2.3133
## Stdev for i                                 0.8274 0.6664
## GroupRho for i                              0.7559 0.5585
```

These distributions are showed in Figure 5.2.3, as well also the marginal ditribution for the intercept, error precision, spatial range, standard deviation and temporal correlation in the spacetime field with the commands bellow

```
par(mfrow=c(2,3), mar=c(3,3,1,0.1), mgp=2:0)
plot(res$marginals.fixed[[1]], type='l',
     xlab=expression(beta[0]), ylab='Density')
abline(v=beta0, col=2)
for (j in 1:4) {
    plot(res$marginals.hyper[[j]], type='l',
         xlab=names(res$marginals.hyper)[j], ylab='Density')
    abline(v=c(tau.error, sqrt(8)/kappa.s, sqrt(s2),
               exp(-kappa.t*diff(mesh.t$loc[1:2])))[j], col=2)
}
```

5.3 Lowering resolution of a spatio-temporal model

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-lower-spatio-temporal.R`

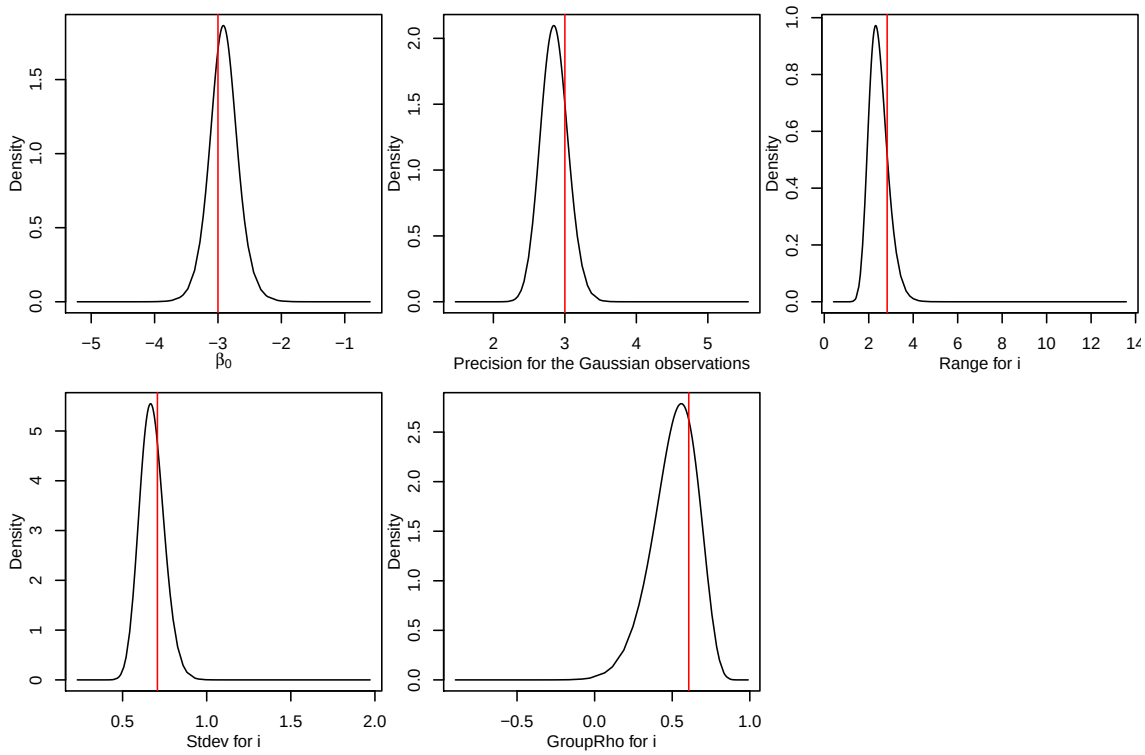

Figure 5.5: Marginal posterior distribution for the intercept, likelihood precision and the parameters in the space-time process.

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(1e-5,0.7), cutoff=0.5)
spde.s <- inla.spde2.matern(mesh.s)
```

5.3.1 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
```

5.3.2 Lowering temporal model resolution

This approach can be seen from the template code in Section 3.2 of [Lindgren and Rue, 2015] and was also considered in the last example of the INLA book, [Blangiardo and Cameletti, 2015]. The main idea is to place some knots over the time window and define the model on such knots. Them define the projection from the time knots as we do for the spatial case with the mesh.

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 5.3.2 were made using the following commands

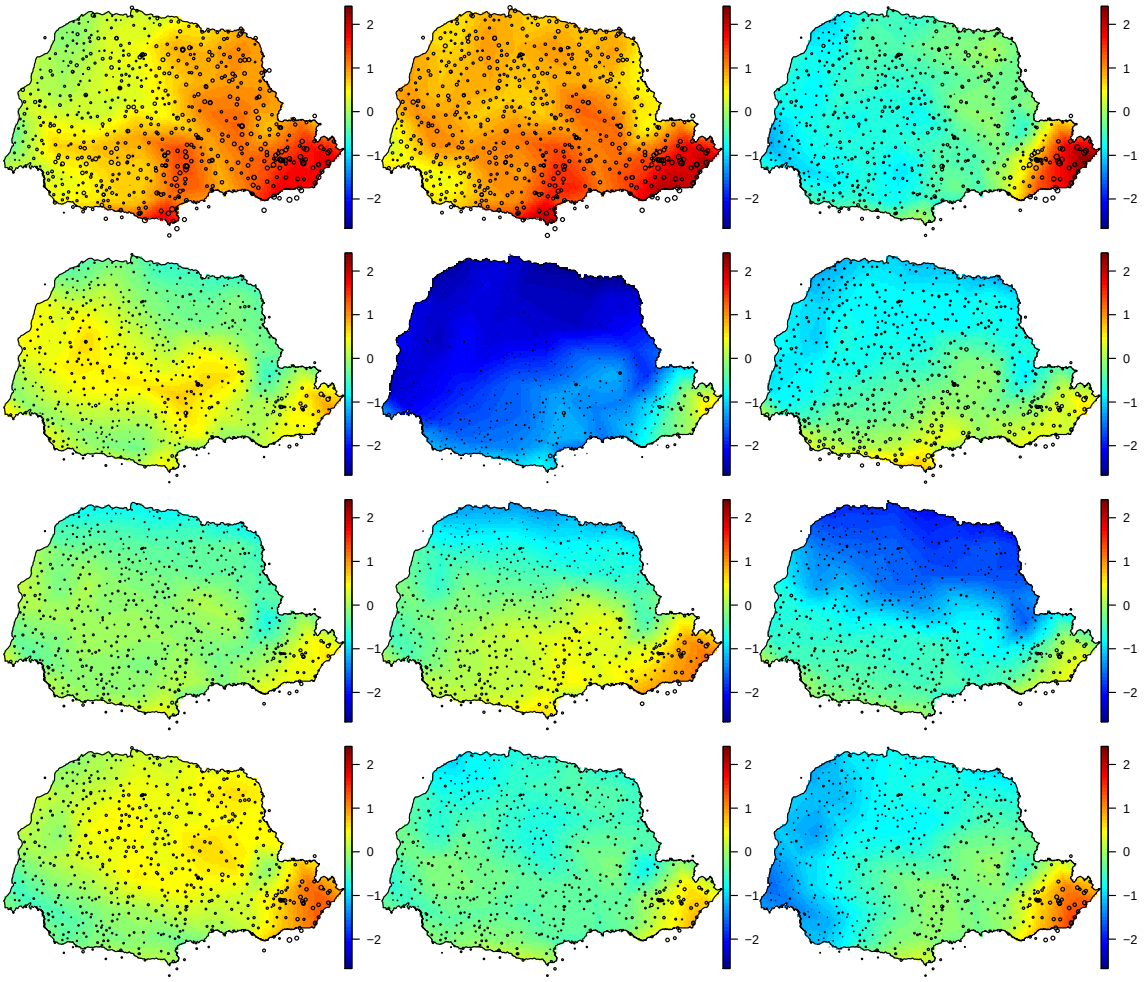

Figure 5.6: Spatial effect at each time knots.

```
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]))
}
```

5.4 Space-time coregionalization model

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-stcoregionalization.R`

```
## Loading required package:  sp
## Loading required package:  Matrix
## This is INLA_17.07.20-1 built 2017-07-20 20:57:50 UTC.
## See www.r-inla.org/contact-us for how to get help.
```

In this Chapter we present a way to fit a spacetime version of the Bayesian spatial coregionalization model proposed by [Schimdt and Gelfand, 2003]. Because we do the

modeling with SPDE that consider the model on a mesh and it can be considered projections for other points in the spacetime domain. This is an important point as we can have the outcomes measured at different points in space and time. The only need is to have the data in the same spacetime domain.

WARNING: a crude mesh and empirical Bayes is used in order to run this example in a short time.

5.4.1 The model and parametrization

The case of three outcomes is defined considering the following equations

$$y_1(s,t) = \alpha_1 + z_1(s,t) + e_1(s,t)$$

$$y_2(s,t) = \alpha_2 + \lambda_1 y_1(s,t) + z_2(s,t) + e_2(s,t)$$

$$y_3(s,t) = \alpha_3 + \lambda_2 y_1(s,t) + \lambda_3 y_2(s,t) + z_3(s,t) + e_3(s,t)$$

where the $z_k(s,t)$ are spacetime correlated processes and $e_k(s,t)$ are uncorrelated error terms, $k = 1, 2, 3$.

In order to fit this model in `R-INLA` we consider a reparametrization. This reparametrization is to change the second equation as follows

$$\begin{aligned} y_2(s,t) &= \alpha_2 + \lambda_1[\alpha_1 + z_1(s,t) + e_1(z,t)] + z_2(s,t) + e_2(s,t) \\ &= (\alpha_2 + \lambda_1\alpha_1) + \lambda_1[z_1(s,t) + e_1(s,t)] + z_2(s,t) + e_2(s,t) \end{aligned}$$

and the third equation as follows

$$\begin{aligned} y_3(s,t) &= \alpha_3 + \lambda_2(\alpha_2 + \lambda_1\alpha_1) + \lambda_2\lambda_1[z_1(s,t) + e_1(s,t)] + \lambda_3\{\alpha_2 + \lambda_1\alpha_1 + \lambda_1[z_1(s,t) + e_1(s,t)] + z_2(s, \\ &= [\alpha_3 + \lambda_2\alpha_1 + \lambda_3(\alpha_2 + \lambda_1\alpha_1)] + \\ &\quad (\lambda_2 + \lambda_3\lambda_1)[z_1(s,t) + e_1(s,t)] + \lambda_3[z_2(s,t) + e_2(s,t)] + z_3(s,t) + e_3(s,t) \end{aligned}$$

We have then two new intercepts $\alpha_2^* = \alpha_2 + \lambda_1\alpha_1$ and $\alpha_3^* = \alpha_3 + \lambda_2(\alpha_2 + \lambda_1\lambda_1) + \lambda_3(\alpha_2 + \lambda_1\alpha_1)$. We also have one new regression coeffiecient $\lambda_2^* = \lambda_2 + \lambda_3\lambda_1$.

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.

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 $\lambda_3 = \beta_3$.

5.4.2 Data simulation

Parameter setting

```
alpha <- c(-5, 3, 10) ### intercept on reparametrized model
m.var <- (3:5)/10 ### random field marginal variances
kappa <- c(12, 10, 7) ### GRF scales: inverse range parameters
beta <- c(.7, .5, -.5) ### copy par.: reparam. coregionalization par.
rho <- c(.7, .8, .9) ### temporal correlations
n <- 300;  k <- 9  ### number of spatial locations and time points
```

When working with SPDE models is not required for the spatial locations to be the same for each process to fit this model in `R-INLA`, as shown in the Chapter 8 of [Blangiardo and Cameletti, 2015] and in the measurement error example in Section 3.1. As we define the model over a set of time knots to fit a spacetime continuous random field, it is also not required for the spacetime coordinates from each outcome to be the same. 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 can use the `rMatern()` function defined in the section 1.1.4 to simulate independent random field realizations for each time. This function is available in the file at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`

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

The time evolution will follows an autoregressive first order process as we used in Chapter 5.

```
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

```
e.sd <- c(0.3, 0.2, 0.15)
y1 <- alpha[1] + z1 + rnorm(n, 0, e.sd[1])
y2 <- alpha[2] + beta[1] * z1 + z2 + rnorm(n, 0, e.sd[2])
y3 <- alpha[3] + beta[2] * z1 + beta[3] * z2 + z3 +
    rnorm(n, 0, e.sd[3])
```

5.4.3 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=0.2,
                     offset=0.1, cutoff=0.1)
```

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

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., 2017]

```
rho1p <- list(theta=list(prior='pccor1', 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)))
```

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 a penalized complexity prior for the errors precision, [Simspon et al., 2017],

```
eprec <- list(hyper=list(theta=list(prior='pc.prec',
                                    param=c(1, 0.01))))
```

We have 15 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(1/e.sd^2),
               c(log(sqrt(8)/kappa), log(sqrt(m.var)),
                 qlogis(rho))[c(1,4,7, 2,5,8, 3,6,9)], beta)
```

With 15 hyperparameters in the model and the CCD strategy will use 287 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, setting `int.strategy='eb'` and these marginals will consider only the modal configuration for θ.

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

## Note:  method with signature 'Matrix#numLike' chosen for function '%*%',
##  target signature 'dgTMatrix#numeric'.
##  "TsparseMatrix#ANY" would also be valid
## Note:  method with signature 'sparseMatrix#matrix' chosen for function '%*%',
##  target signature 'dgTMatrix#matrix'.
##  "TsparseMatrix#ANY" would also be valid
```

```
##  Pre-processing    Running inla Post-processing           Total
##       1.3002827     743.4713945       0.7511568     745.5228341
```

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

## [1] "Number of function evaluations = 1063"

round(result$mode$theta, 2)

##    Log precision for the Gaussian observations
##                                           2.03
## Log precision for the Gaussian observations[2]
##                                           2.23
## Log precision for the Gaussian observations[3]
##                                           2.50
##                                log(Range) for s1
##                                          -1.33
##                                log(Stdev) for s1
##                                          -0.42
##                           Group rho_intern for s1
##                                           2.38
##                                log(Range) for s2
##                                          -1.06
##                                log(Stdev) for s2
##                                          -0.36
##                           Group rho_intern for s2
##                                           2.91
##                                log(Range) for s3
##                                          -0.81
##                                log(Stdev) for s3
##                                          -0.45
##                           Group rho_intern for s3
```

```
##                                          3.60
##                            Beta_intern for s12
##                                          0.78
##                            Beta_intern for s13
##                                          0.59
##                            Beta_intern for s23
##                                         -0.60
```

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 -4.90 0.14      -5.18    -4.90      -4.62 -4.90   0
## intercept2    3  3.28 0.25       2.79     3.28       3.76  3.28   0
## intercept3   10  9.97 0.32       9.35     9.97      10.59  9.97   0
```

Posterior marginal for the errors precision

```
round(cbind(true=c(e=e.sd^-2), result$summary.hy[1:3, ]), 4)

##       true    mean     sd 0.025quant 0.5quant 0.975quant    mode
## e1 11.1111  7.6471 0.2234     7.2155   7.6444     8.0948  7.6401
## e2 25.0000  9.3143 0.2777     8.7767   9.3114     9.8702  9.3074
## e3 44.4444 12.2251 0.3627    11.5293  12.2192    12.9551 12.2067
```

Summary of the posterior marginal density for the temporal correlations:

```
round(cbind(true=rho, result$summary.hy[c(6,9,12),]), 4)

##                  true   mean     sd 0.025quant 0.5quant 0.975quant   mode
## GroupRho for s1   0.7 0.8316 0.0226     0.7841   0.8326     0.8731 0.8341
## GroupRho for s2   0.8 0.8942 0.0166     0.8576   0.8956     0.9230 0.8984
## GroupRho for s3   0.9 0.9480 0.0116     0.9230   0.9488     0.9682 0.9501
```

Summary of the posterior marginal density for the copy parameters:

```
round(cbind(true=beta, result$summary.hy[13:15,]), 4)

##              true    mean     sd 0.025quant 0.5quant 0.975quant    mode
## Beta for s12  0.7  0.7761 0.0499     0.6778   0.7762     0.8743  0.7764
## Beta for s13  0.5  0.5935 0.0447     0.5056   0.5935     0.6813  0.5936
## Beta for s23 -0.5 -0.5986 0.0506    -0.6981  -0.5986    -0.4990 -0.5986
```

Look for the random field parameters for each field. The practical range for each random field

```
round(cbind(true=sqrt(8)/kappa, result$summary.hy[c(4, 7, 10),]), 3)

##               true  mean    sd 0.025quant 0.5quant 0.975quant  mode
## Range for s1 0.236 0.263 0.024      0.219    0.263      0.313 0.261
## Range for s2 0.283 0.350 0.034      0.290    0.348      0.421 0.344
## Range for s3 0.404 0.451 0.053      0.359    0.447      0.566 0.438
```

The standard deviation for each random field

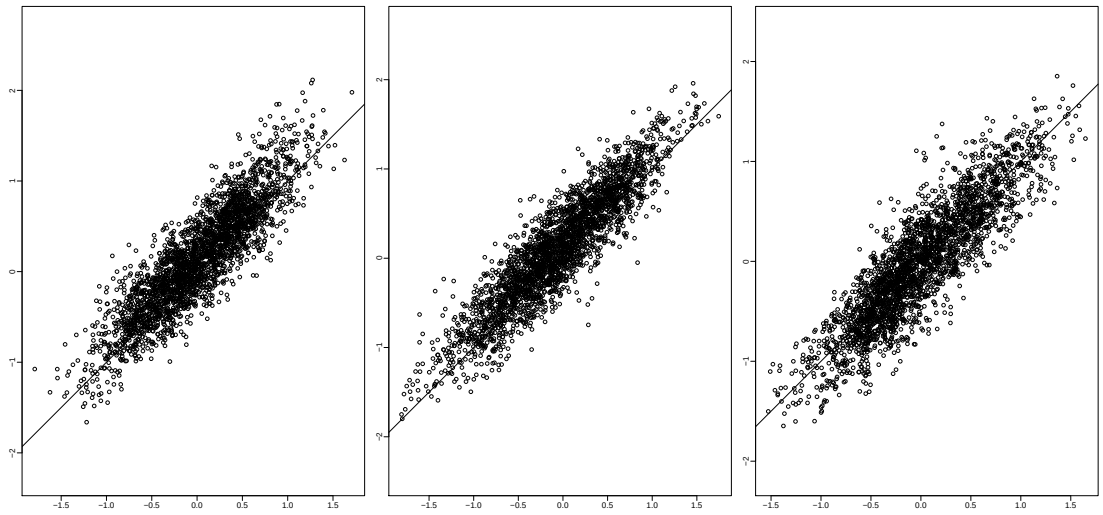

Figure 5.7: True and fitted random field values.

```
round(cbind(true=m.var^0.5, result$summary.hy[c(5, 8, 11),]), 3)
```

```
##                 true  mean    sd 0.025quant 0.5quant 0.975quant  mode
## Stdev for s1 0.548 0.662 0.042      0.585    0.660      0.749 0.656
## Stdev for s2 0.632 0.701 0.051      0.604    0.699      0.806 0.696
## Stdev for s3 0.707 0.647 0.065      0.533    0.642      0.788 0.629
```

The posterior mean for each random field is projected to the observation locations and shown against the simulated correspondent fields in Figure 5.4.3 with the code bellow.

```
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)
```

Remember that the crude approximation for the covariance and the simplifications on the inference procedure is not recommended to use in practice. It can be considered for having initial results. Even thou, it seems that the method was reazonable well having covered the parameter values used to simulate the data.

5.5 Dynamic regression example

The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-dynamic.R`

There is a large literature about dynamic models with also some books about it, from [West and Harrison, 1997] to [Petris et al., 2009]. These models basically defines an hierarchical framework for a class of time series models. A particular case is the dynamic regression model, were the regression coefficients are modeled as time series. That is the case when the regression coefficients vary smoothly over time.

5.5.1 Dynamic space-time regression

The specific class of models for spatially structured time series was proposed by [Gelfand et al., 2003], where the regression coefficients varies smoothly over time and space. For the areal data

case, the use of proper Gaussian Markov random fields (PGMRF) over space as proposed by [Vivar and Ferreira, 2009]. There exists a particular class of such models called "spatially varying coefficient models" were the regression coefficients veries over space, [Assunção et al., 1999], [Assunção et al., 2002], [Gamerman et al., 2003].

In [Gelfand et al., 2003] the Gibbs sampler were used for inference and it was claimed that better algorithms is needed due to strong autocorrelations. In [Vivar and Ferreira, 2009] the use of forward information filtering and backward sampling (FIFBS) recursions were proposed. Both MCMC algorithms are computationally expensive.

One can avoid the FFBS algorithm as a relation between the Kalman-filter and the Cholesky factorization is provided in [Knorr-Held and Rue, 2002]. The Cholesky fator is more general and has superior performance when using sparse matrix methods, [Rue and Held, 2005, p. 149]. Additionally, the restriction that the prior for the latent field has to be proper can be avoided.

When the likelihood is Gaussian, there is no approximation needed in the inference process since the distribution of the latent field given the data and the hyperparameters is Gaussian. So, the main task is to perform inference for the hyperparameters in the model. For this, the mode and curvature around can be found without any sampling method. For the class of models in [Vivar and Ferreira, 2009] it is natural to use INLA, as shown in [Ruiz-Cárdenas et al., 2012], and for the models in [Gelfand et al., 2003] we can use the SPDE approach when considering the Matérn covariance for the spatial part.

In this example we will show how to fit the space-time dynamic regression model as in [Gelfand et al., 2003], considering the Matérn spatial covariance and the AR(1) model for time which corresponds to the exponential correlation function. This particular covariance choise correspond to the model in [**?**], where only the intercept is dynamic. Here, we show the case when we have a dynamic intercept and a dynamic regression coefficient for an harmonic over time.

5.5.2 Simulation from the model

We can start on defining the spatial locations:

```
n <- 150;   set.seed(1);  coo <- matrix(runif(2*n), n)
```

To sample from a random field on a set of location, we can use the `rMatern()` function defined in the Section 1.1.4 to simulate independent random field realizations for each time. This function is available in the file at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`

We draw k (number of time points) samples from the random field. Then, we make it temporally correlated considering the time autoregression

```
kappa <- c(10, 12);    sigma2 <- c(1/2, 1/4)
k <- 15;  rho <- c(0.7, 0.5)
set.seed(2); beta0 <- rMatern(k, coo, kappa[1], sigma2[1])
set.seed(3); beta1 <- rMatern(k, coo, kappa[2], sigma2[2])
beta0[,1] <- beta0[,1] / (1-rho[1]^2)
beta1[,1] <- beta1[,1] / (1-rho[2]^2)
for (j in 2:k) {
    beta0[, j] <- beta0[,j-1]*rho[1] + beta0[,j] * (1-rho[1]^2)
    beta1[, j] <- beta1[,j-1]*rho[2] + beta1[,j] * (1-rho[2]^2)
}
```

where the $(1 - \rho_j^2)$ term is in accord to the parametrization of the AR(1) model in INLA.

To get the response, we define the harmonic as a function over time, compute the mean and add an error term

```
set.seed(4); hh <- runif(n*k) ### simlate the covariate values
mu.beta <- c(-5, 1);   taue <- 20
set.seed(5); error <- rnorm(n*k, 0, sqrt(1/taue)) ### error in the observa-
tion
length(y <- (mu.beta[1] + beta0) + (mu.beta[2]+beta1)*hh + ### dynamic regres-
sion part
           error)

## [1] 2250
```

5.5.3 Fitting the model

We have two space-time terms on the model, each one with three hyperparameters: precision, spatial scale, temporal scale (or temporal correlation). So, considering the likelihood, 7 hyperparameters in total. To perform fast inference, we choose to have a crude mesh with with small number of vertices.

```
(mesh <- inla.mesh.2d(coo, max.edge=c(0.25), ### coarse mesh
                      offset=c(0.15), cutoff=0.05))$n

## [1] 142
```

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
```

We do need one set of index for each call of the `f()` function, no matter if they are the same, so:

```
i0 <- inla.spde.make.index('i0', spde$n.spde, n.group=k)
i1 <- inla.spde.make.index('i1', spde$n.spde, n.group=k)
```

In the SPDE approach, the space-time model is defined in a set of mesh nodes. As we have considered continuous time, it is also defined on a set of time knots. So, we have to deal with the projection from the model domain (nodes, knots) to the space-time data locations. For the intercept it is the same way as in the other examples. For the regression coefficients, all we need to multiply the projector matrix by the covariate vector columnwie, i. e., each column of the projector matrix is multiplyed by the covariate vector. It can be seen from the following

$$\begin{aligned} \boldsymbol{\eta} &= \mu_{\beta_0} + \mu_{\beta_2}\boldsymbol{h} + \boldsymbol{A}\boldsymbol{\beta}_0 + (\boldsymbol{A}\boldsymbol{\beta}_1)\boldsymbol{h} \\ &= \mu_{\beta_0} + \mu_{\beta_1}\boldsymbol{h} + \boldsymbol{A}\boldsymbol{\beta}_0 + (\boldsymbol{A} \oplus (\boldsymbol{h}\boldsymbol{1}^{'}))\boldsymbol{\beta}_1 \end{aligned} \tag{5.2}$$

where $\boldsymbol{A} \oplus (\boldsymbol{h}\boldsymbol{1}^{'})$ is the row-wise Kronecker product between $\boldsymbol{A}$ and a the vector $\boldsymbol{h}$ (with length equal the number of rows in $\boldsymbol{A}$) expressed as the Kronecker sum of $\boldsymbol{A}$ and $\boldsymbol{h}\boldsymbol{1}^{1}$. This operation can be performed usind the `inla.row.kron()` function and is done internally in the function `inla.spde.make.A()` when supplying a vector in the `weights` argument.

The space-time projector matrix $\boldsymbol{A}$ is defined as follows:

```
A0 <- inla.spde.make.A(mesh, cbind(rep(coo[,1], k), rep(coo[,2], k)),
                       group=rep(1:k, each=n))
A1 <- inla.spde.make.A(mesh, cbind(rep(coo[,1], k), rep(coo[,2], k)),
                       group=rep(1:k, each=n), weights=hh)
```

The data stack is as follows

```
stk.y <- inla.stack(data=list(y=as.vector(y)), tag='y',
                    A=list(A0, A1, 1),
                    effects=list(i0, i1,
                        data.frame(mu1=1, h=hh)))
```

where `i0` is similar to `i1` and the elements `mu1` and `h` in the second element of the effects `data.frame` is for $\boldsymbol{\mu}_\xi$.

The formula take these things into account

```
form <- y ~ 0 + mu1 + h + ### to fit mu_beta
    f(i0, model=spde, group=i0.group, control.group=list(model='ar1')) +
        f(i1, model=spde, group=i1.group, control.group=list(model='ar1'))
```

As we have Gaussian likelihood there is no approximation in the fitting process. The first step of the INLA algorithm is the optimization to find the mode of the 7 hyperparameters in the model. By choosing good starting values it will be needed less iteractions in this optimization process. Below, we define starting values for the hyperparameters in the internal scale considering the values used to simute the data

```
(theta.ini <- c(log(taue), ## likelihood log precision
                log(sqrt(8)/kappa[1]), ## log range 1
                log(sqrt(sigma2[1])), ## log stdev 1
                log((1+rho[1])/(1-rho[1])), ## inv.logit rho 1
                log(sqrt(8)/kappa[2]), ## log range 1
                log(sqrt(sigma2[2])), ## log stdev 1
                log((1+rho[2])/(1-rho[2]))))## inv.logit rho 2

## [1]  2.9957323 -1.2628643 -0.3465736  1.7346011 -1.4451859 -0.6931472
## [7]  1.0986123
```

This step takes around few minutes to fit, and with bigger `tolerance` value in `inla.control`, it will makes fewer posterior evaluations.

The integration step when using the CCD strategy, will integrates over 79 hyperparameter configurations, as we have 7 hyperparameters. However, in the following `inla()` call we avoid it.

Fitting the model considering the initial values defined above

```
(res <- inla(form, family='gaussian', data=inla.stack.data(stk.y),
             control.predictor=list(A=inla.stack.A(stk.y)),
             control.inla=list(int.strategy='eb'), ### no integration wr theta
             control.mode=list(theta=theta.ini, ### initial theta value
                               restart=TRUE)))$cpu

##  Pre-processing    Running inla Post-processing           Total
##       0.8508792     145.7084601       0.4307158     146.9900551
```

Summary of the μ_β:

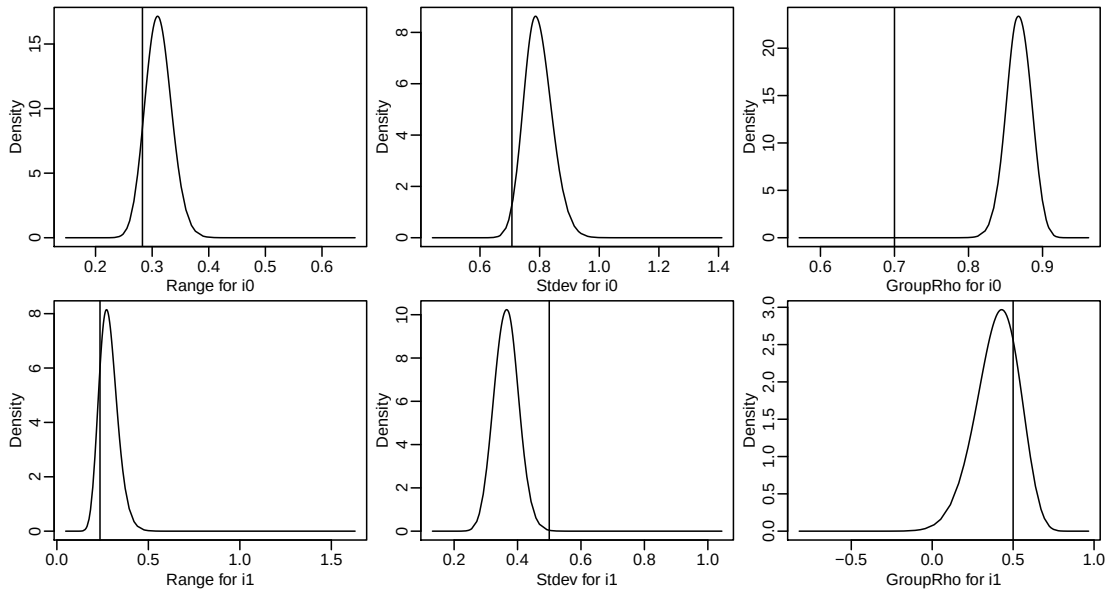

Figure 5.8: Posterior marginal distributions for the hyperparameters of the spacetime fields.

```
round(cbind(true=mu.beta, res$summary.fix), 4)

##     true    mean     sd 0.025quant 0.5quant 0.975quant    mode kld
## mu1   -5 -4.7481 0.1818    -5.1051  -4.7481    -4.3915 -4.7481   0
## h      1  0.9428 0.0531     0.8386   0.9428     1.0469  0.9428   0
```

Summary for the likelihood precision

```
round(c(true=taue, unlist(res$summary.hy[1,])), 3)

##       true       mean         sd 0.025quant   0.5quant 0.975quant
##     20.000     11.484      0.536     10.458     11.474     12.569
##       mode
##     11.458
```

We can see the posterior marginal distributions for the range and standard deviation for each spatio-temporal process in Figure 5.5.3.

```
par(mfrow=c(2, 3), mar=c(2.5,2.5,0.3,0.3), mgp=c(1.5,0.5,0))
for (j in 2:7) {
    plot(res$marginals.hy[[j]], type='l',
         xlab=names(res$marginals.hyperpar)[j], ylab='Density')
    abline(v=c(sqrt(8)/kappa[1], sigma2[1]^0.5, rho[1],
               sqrt(8)/kappa[2], sigma2[2]^0.5, rho[2])[j-1])
}
```

We can have a look over the posterior mean of the dynamic coefficients. We compute the correlation between the simulated and the posterior mean ones by

```
c(beta0=cor(as.vector(beta0), drop(A0%*%res$summary.ran$i0$mean)),
  beta1=cor(as.vector(beta1),
      drop(A0%*%res$summary.ran$i1$mean))) ## using A0 to account only for the co-
eff.
```

```
##     beta0     beta1
## 0.9452990 0.6163353
```

5.6 Space-time point process: Burkitt example

In this example we show hot to fit a space-time point process using the `burkitt` dataset from the **splancs R** package. The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-burkitt.R`

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 5.6.

```
n <- nrow(burkitt)
par(mfrow=c(1,1), mar=c(1.5,.1,.1,.1), mgp=c(2,0.7,0))
plot(burkitt$t, rep(1,n), type='h', ylim=0:1, axes=FALSE, xlab='', ylab='')
box(); axis(1)
```

We have to define a set of knots over time in order to fit SPDE spatio temporal model. It is then used to built a temporal mesh

```
k <- 6
tknots <- seq(min(burkitt$t), max(burkitt$t), length=k)
abline(v=tknots, lwd=4, col=4) ## add to plot
mesh.t <- inla.mesh.1d(tknots)
```

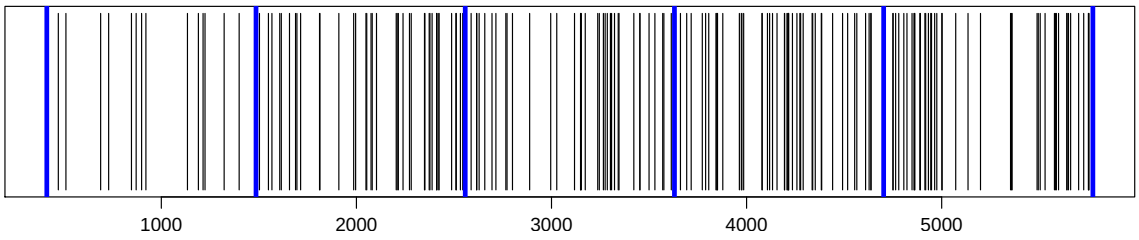

Figure 5.9: Time when each event occurred (black) and knots used for inference (blue).

The spatial mesh can be done using the polygon of the region as a boundary. We can convert the domain polygon into a `SpatialPolygons` class with

```
domainSP <- SpatialPolygons(list(Polygons(
    list(Polygon(burbdy)), '0')))
```

and the use it as a boundary

```
mesh.s <- inla.mesh.2d(burpts, boundary=inla.sp2segment(domainSP),
                       max.edge=c(10, 25), cutoff=3) ### just a crude mesh
```

Defining the SPDE model considering the PC-prior derived in [Fuglstad et al., 2017] for the model parameters as the practical range, $\sqrt{8\nu}/\kappa$, and the marginal standard deviation.

```
spde <- inla.spde2.pcmatern(
    mesh=mesh.s, alpha=2, ### mesh and smoothness parameter
    prior.range=c(0.05, 0.01), ### P(practic.range<0.05)=0.01
    prior.sigma=c(1, 0.01)) ### P(sigma>1)=0.01
m <- spde$n.spde
```

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

```
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 3234
```

Internally `inla.spde.make.A` function makes a row Kronecker product (see `inla.row.kron`) between the spatial projector matrix and the group (temporal in our case) projector one. This matrix has number of columns equals to the number of nodes in the mesh times the number of groups.

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. For the integration points it is the spacetime volume computed for each mesh node and time knot, considering the spatial area of the dual mesh polygons, as in Chapter 4, times the the length of the time window at each time point. For the data locations it is zero as for a point the expectation is zero, in accord to the likelihood approximation proposed by [Simpson et al., 2016].

The dual mesh is extracted considering the function `inla.mesh.dual()`, available in `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-functions.R`

```
source('R/spde-tutorial-functions.R')
dmesh <- inla.mesh.dual(mesh.s)
```

Them, we compute the intersection with each polygon from the mesh dual using the functions `gIntersection()` from the **rgeos** package (show the sum of the intersection polygons areas):

```
library(rgeos)
sum(w <- sapply(1:length(dmesh), function(i) {
    if (gIntersects(dmesh[i,], domainSP))
        return(gArea(gIntersection(dmesh[i,], 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
```

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')

```
pcrho <- list(prior='pccor1', param=c(0.7, 0.7))
form <- y ~ 0 + a0 +
    f(s, model=spde, group=s.group,
      control.group=list(model='ar1', hyper=list(theta=pcrho)))
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 188.0629
```

We can plot the posterior marginal distributions for the intercept and parameters, in Figure 5.6, with

```
par(mfrow=c(2,2), mar=c(3,3,1,1), mgp=c(1.7,0.7,0))
plot(burk.res$marginals.fix[[1]], type='l', xlab='Intercept')
plot(burk.res$marginals.hy[[1]], type='l',
     xlim=c(0, 10), xlab='Practical range')
plot(burk.res$marginals.hy[[2]], type='l',
     xlim=c(0, 3), xlab='Standard deviation')
plot(burk.res$marginals.hy[[3]], type='l',
     xlim=c(0, 1), xlab='time correlation')
```

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

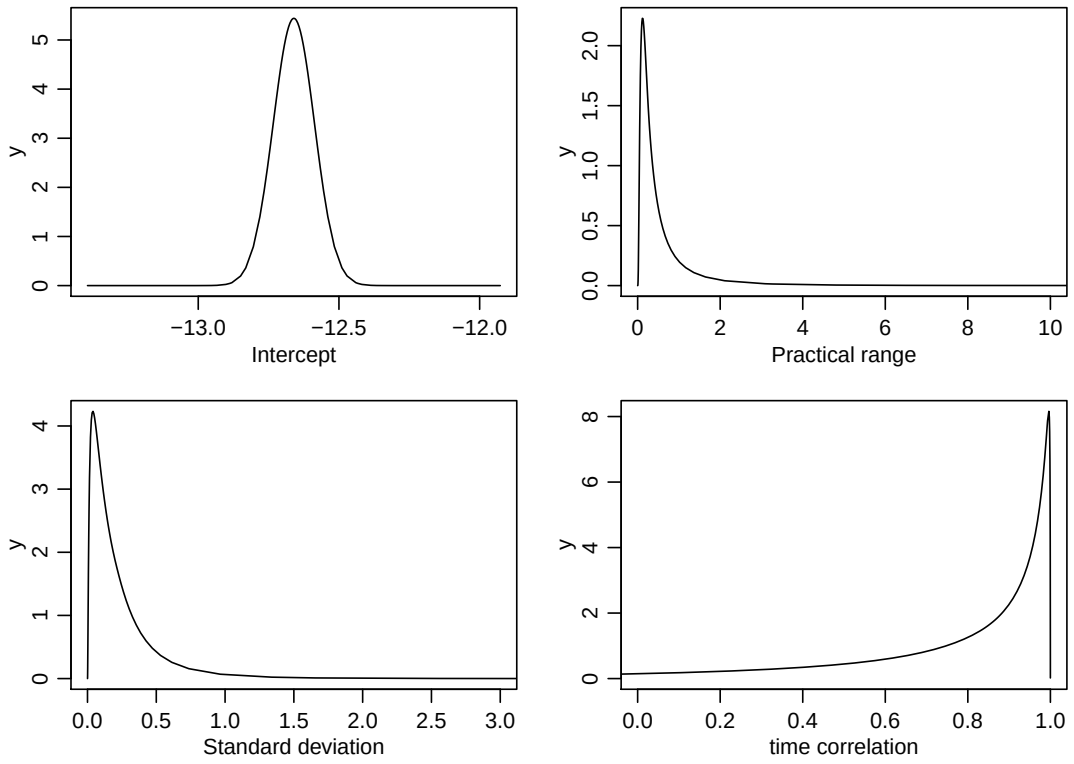

Figure 5.10: Intercept and Random Field parameters posterior marginal distributions.

```
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 5.6, 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)
```

5.7 Large point process data set

In this chapter we show how an approach to fit a spatio temporal log-Cox point process model for a large data sets. We are going to drawn samples from a separable space time intensity function. The R source for this file is available at `http://www.math.ntnu.no/inla/r-inla.org/tutorials/spde/R/spde-tutorial-stpp.R`

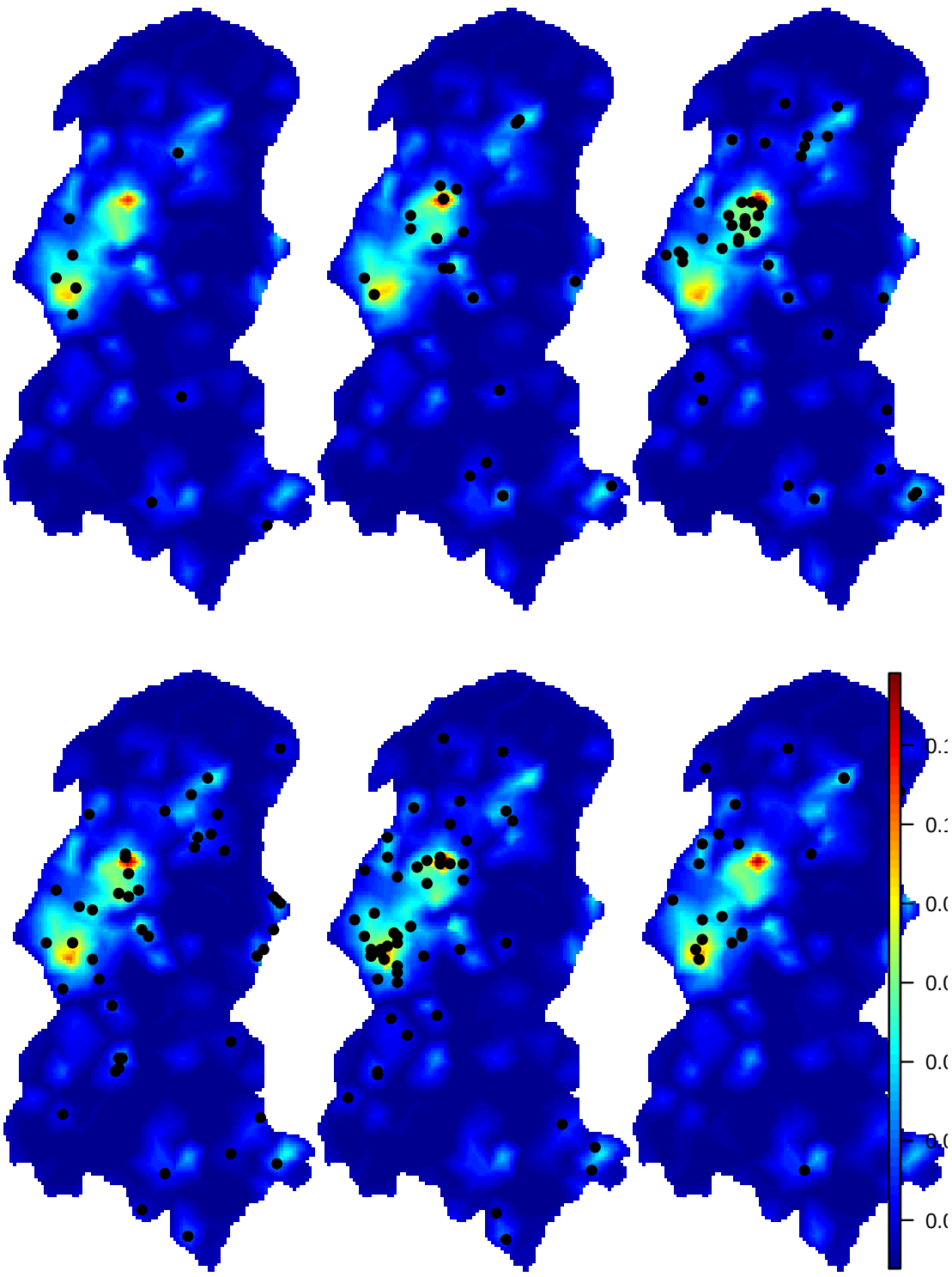

Figure 5.11: Fitted latent field at each time knot overlayed by the points closer in time.

First we define the spatial domain as follows

```
x0 <- seq(0, 4*pi, length=15)
domain <- data.frame(x=c(x0, rev(x0), 0))
domain$y <- c(sin(x0/2)-2, sin(rev(x0/2))+2, sin(0)-2)
```

and convert it into the `SpatialPolygons` class

```
library(sp)
domainSP <- SpatialPolygons(list(Polygons(list(Polygon(domain)), '0')))
```

We choose to sample a dataset using the **lgcp**, [Taylor et al., 2013], package as follows

```
library(lgcp)
ndays <- 15
n <- (xyt <- lgcpSim(
    owin=spatstat:::owin(poly=domain), tlim=c(0,ndays),
    model.parameters=lgcppars(1,0.5,0.1,0,0.5), cellwidth=0.1,
    spatial.covmodel='matern', covpars=c(nu=1)))$n
```

In order to fit the model, we do need to define a discretization over space and over time. For the time domain, we define a temporal mesh based on a number of time knots:

```
k <- 7; tmesh <- inla.mesh.1d(seq(0, ndays, length=k))
```

The spatial mesh is defined using the domain polygon:

```
smesh <- inla.mesh.2d(boundary=inla.sp2segment(domainSP),
                      max.edge=1, cutoff=0.3)
```

We can have a look in Figure 5.7 to see a plot of a sample of the data over time, the time knots and over space and the spatial mesh as well with the commands below

```
par(mfrow=c(2,1), mar=c(1.5,0,0,0), mgp=c(1,0.5,0))
plot(sample(xyt$t,500), rep(1,500), type='h', ylim=0:1,
     xlab='Day', ylab='', axes=FALSE); box(); axis(1)
abline(v=tmesh$loc, col=4, lwd=3)
par(mar=c(0,0,0,0))
plot(smesh, asp=1, main='')
points(xyt$x, xyt$y, cex=0.5, pch=3)
```

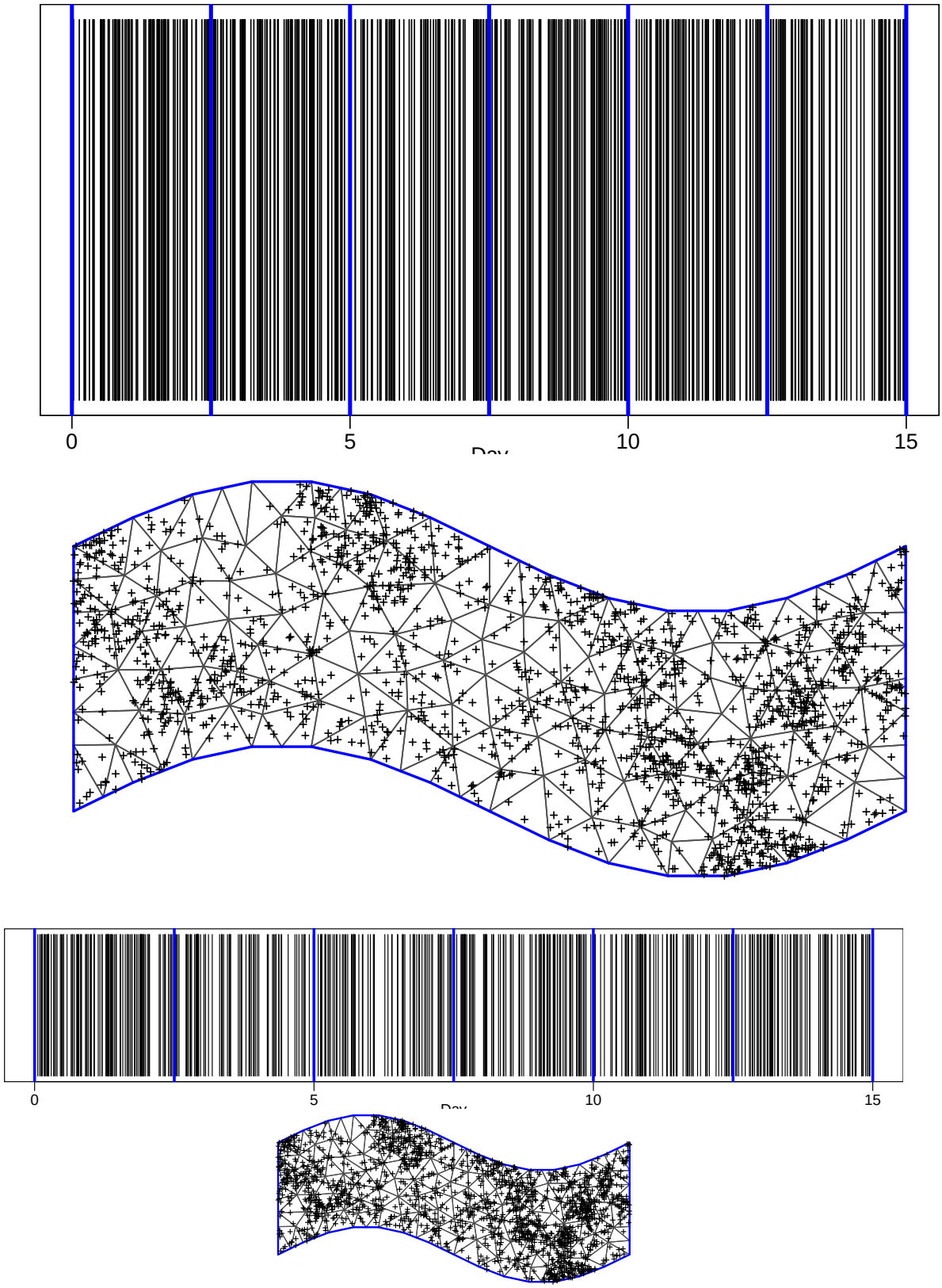

Figure 5.12: Time for a sample of the events (black), time knots (blue) in the upper plot. Spatial locations of a sample on the spatial domain (botton plot).

5.7.1 Space-time aggregation

For large datasets it can be computationally demanding to fit the model. The problem is because the dimention of the model would be $n + m * k$, where n is the number of data

points, m is the number of nodes in the mesh, k is the number of time knots. In this section we choose to aggregate the data in a way that we have a problem with dimention $2 * m * k$. So, this approach really makes sence when $n >> m * k$.

We choose to aggregate the data in accord to the integration points to make the fitting process easier. We also consider the dual mesh polygons, as shown in Chapter 4.

So, firt we find the Voronoi polygons for the mesh nodes

```
library(deldir)
dd <- deldir(smesh$loc[,1], smesh$loc[,2])
tiles <- tile.list(dd)
```

Convert it into `SpatialPolygons`:

```
polys <- SpatialPolygons(lapply(1:length(tiles), function(i)
    { p <- cbind(tiles[[i]]$x, tiles[[i]]$y)
      n <- nrow(p)
      Polygons(list(Polygon(p[c(1:n, 1),])), i)
  }))
```

Find to which polygon belongs each data point:

```
area <- factor(over(SpatialPoints(cbind(xyt$x, xyt$y)),
                    polys), levels=1:length(polys))
```

Fint to which part of the time mesh belongs each data point:

```
t.breaks <- sort(c(tmesh$loc[c(1,k)],
                   tmesh$loc[2:k-1]/2 + tmesh$loc[2:k]/2))
table(time <- factor(findInterval(xyt$t, t.breaks),
                     levels=1:(length(t.breaks)-1)))

##
##   1   2   3   4   5   6   7
## 183 327 260 282 321 288 146
```

Use these both identification index sets to aggregate the data

```
agg.dat <- as.data.frame(table(area, time))
for(j in 1:2) ### set time and area as integer
    agg.dat[[j]] <- as.integer(as.character(agg.dat[[j]]))
str(agg.dat)

## 'data.frame': 1064 obs. of  3 variables:
##  $ area: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ time: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Freq: int  1 2 0 0 0 0 0 0 0 3 ...
```

We need to define the expected number of cases (at least) proportional to the area of the Polygons times the width length of the time knots. Compute the intersection area of each polygon with the domain (show the sum).

```
library(rgeos)
sum(w.areas <- sapply(1:length(tiles), function(i)
    { p <- cbind(tiles[[i]]$x, tiles[[i]]$y)
```

```
      n <- nrow(p)
      pl <- SpatialPolygons(list(Polygons(list(Polygon(p[c(1:n, 1),])), i)))
      if (gIntersects(pl, domainSP))
          return(gArea(gIntersection(pl, domainSP)))
      else return(0)
  }))

## [1] 50.26548
```

A summary of the polygons area is

```
summary(w.areas)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.06293 0.21780 0.35040 0.33070 0.41160 0.69310
```

and the area of the spatial domain is

```
gArea(domainSP)

## [1] 50.26548
```

The time length (domain) is 365 and the width of each knot is

```
(w.t <- diag(inla.mesh.fem(tmesh)$c0))

## [1] 1.25 2.50 2.50 2.50 2.50 2.50 1.25
```

where the knots at boundary are with less width than the internal ones.

Since the intensity function is the number of cases per volumn unit, with n cases the intensity varies around the average number of cases (intensity) by unit volumn

```
(i0 <- n / (gArea(domainSP) * diff(range(tmesh$loc))))

## [1] 2.396608
```

and this value is related to an intercept in the model we fit below. The space-time volumn (area unit per time unit) at each polygon and time knot is

```
summary(e0 <- w.areas[agg.dat$area] * (w.t[agg.dat$time]))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.07866 0.45690 0.64780 0.70860 0.97130 1.73300
```

5.7.2 Model fit

The projector matrix, SPDE model object and the space-time index set definition:

```
A.st <- inla.spde.make.A(smesh, smesh$loc[agg.dat$area,],
                         group=agg.dat$time, mesh.group=tmesh)
spde <- inla.spde2.matern(smesh)
idx <- inla.spde.make.index('s', spde$n.spde, n.group=k)
```

Defining the data stack

```
stk <- inla.stack(data=list(y=agg.dat$Freq, exposure=e0),
                  A=list(A.st, 1),
                  effects=list(idx,
                      list(b0=rep(1, nrow(agg.dat)))))
```

the formula

```
formula <- y ~ 0 + b0 +
    f(s, model=spde, group=s.group, control.group=list(model='ar1'))
```

and fitting the model

```
res <- inla(formula, family='poisson',
            data=inla.stack.data(stk), E=exposure,
            control.predictor=list(A=inla.stack.A(stk)),
            control.inla=list(strategy='gaussian'))
```

The log of the average intensity and the intercept summary:

```
round(cbind(true=log(i0), res$summary.fixed),4)

##      true  mean     sd 0.025quant 0.5quant 0.975quant  mode kld
## b0 0.8741 0.702 0.1553     0.3933   0.7019     1.0107 0.702   0
```

The expected number of cases at each integration point can be used to compute the total expected number of cases

```
eta.i <- res$summary.fix[1,1] + res$summary.ran$s$mean
c(n=xyt$n, 'E(n)'=sum(rep(w.areas, k)*rep(w.t, each=smesh$n)*exp(eta.i)))

##        n     E(n)
## 1807.000 1805.903
```

The spatial surface at each time knot can be computed by

```
r0 <- diff(range(domain[,1]))/diff(range(domain[,2]))
prj <- inla.mesh.projector(smesh, xlim=bbox(domainSP)[1,],
                           ylim=bbox(domainSP)[2,], dims=c(r0*200, 200))
g.no.in <- is.na(over(SpatialPoints(prj$lattice$loc), domainSP))
t.mean <- lapply(1:k, function(j) {
    z <- inla.mesh.project(prj, res$summary.ran$s$mean[idx$s.group==j])
    z[g.no.in] <- NA
    return(z)
})
```

and is visualized in Figure 5.7.2 is visualized by

```
zlims <- range(unlist(t.mean), na.rm=TRUE)
library(fields)
par(mfrow=c(4,2), mar=c(0.1,0.1,0.1,0.1))
for (j in 1:k) {
    image(prj$x, prj$y, t.mean[[j]],
          axes=FALSE, zlim=zlims, col=tim.colors(30))
    points(xyt$x[time==j], xyt$y[time==j], cex=0.1)
```

```
}
image.plot(prj$x, prj$y, t.mean[[j]]+1e9, axes=FALSE, zlim=zlims, xlab='',
           legend.mar=10, legend.width=5, col=tim.colors(30), horizontal=T)
```

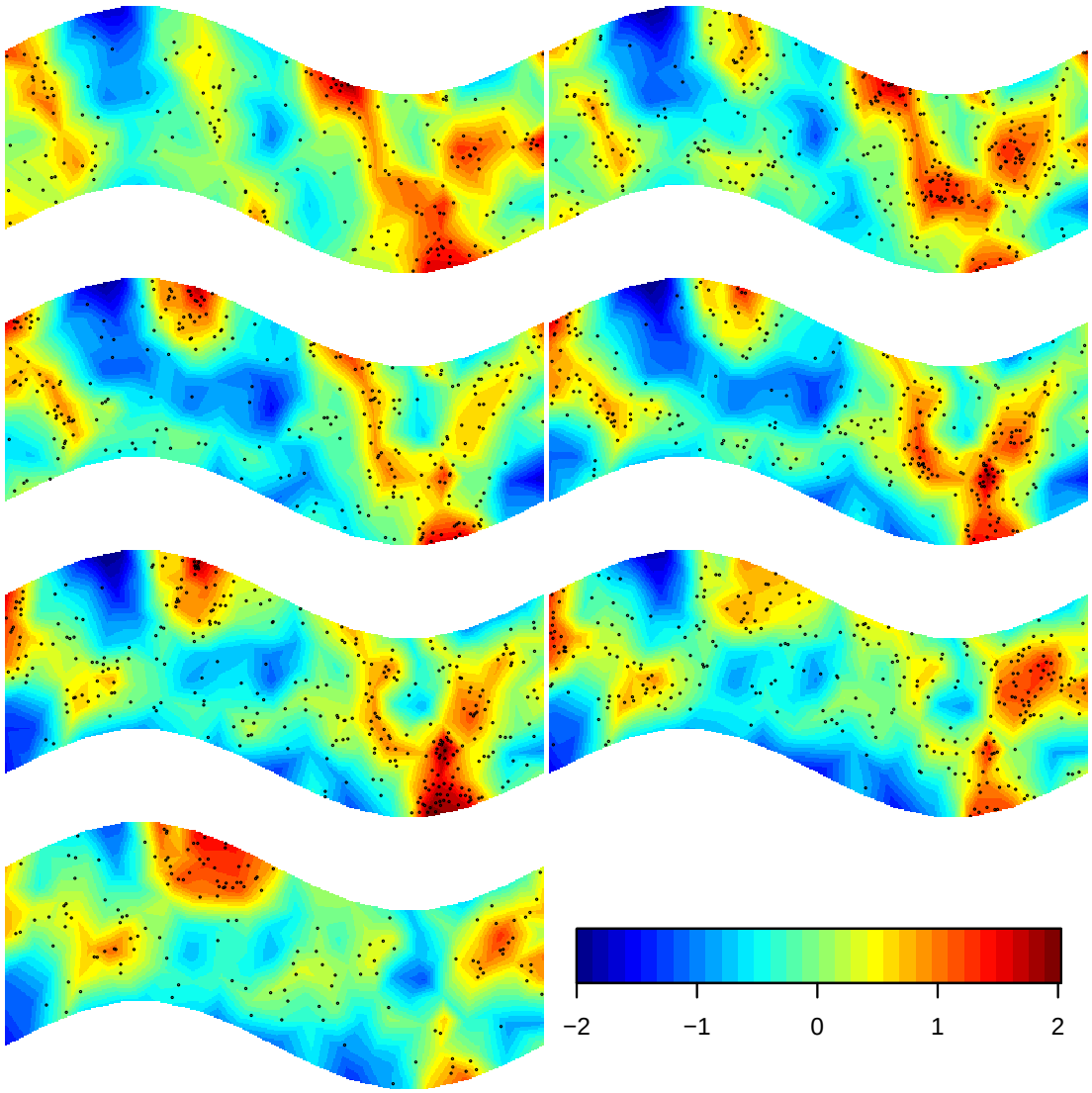

Figure 5.13: Spatial surface fitted at each time knot overlayed by the point pattern formed by the points nearest to each time knot.

Bibliography

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

[Assunção et al., 1999] Assunção, J. J., Gamerman, D., and Assunção, R. M. (1999). Regional differences in factor productivities of brazilian agriculture: A space-varying parameter approach. Technical report, Universidade Federal do Rio de Janeiro, Statistical Laboratory.

[Assunção et al., 2002] Assunção, R. M., Potter, J. E., and Cavenaghi, S. M. (2002). A bayesian space varying parameter model applied to estimating fertility schedules. *Statistics in Medicine*, 21:2057–2075.

[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 SpatioTemporal Bayesian models with RINLA*. 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*, 97(2):109–131.

[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.

[Fuglstad et al., 2017] Fuglstad, G., Simpson, D., Lindgren, F., and Rue, H. (2017). Constructing priors that penalize the complexity of gaussian random fields. *submitted*.

[Gamerman et al., 2003] Gamerman, D., Moreira, A. R. B., and Rue, H. (2003). Space-varying regression models: specifications and simulation. *Computational Statistics & Data Analysis - Special issue: Computational econometrics*, 42(3):513–533.

[Gelfand et al., 2003] Gelfand, A. E., Kim, H., Sirmans, C. F., and Banerjee, S. (2003). Spatial modeling with spatially varying coefficient processes. *Journal of the American Statistical Association*, 98(462):387–396.

[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.

[Knorr-Held and Rue, 2002] Knorr-Held, L. and Rue, H. (2002). On block updating in markov random field models for disease mapping. *Scandinavian Journal of Statistics*, 20:597–614.

[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, 2015] Lindgren, F. and Rue, H. (2015). Bayesian spatial and spatio-temporal modelling with R-INLA. *Journal of Statistical Software*, 63(19).

[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.

[Martins et al., 2013] Martins, T. G., Simpson, D., Lindgren, F., and Rue, H. (2013). Bayesian computing with INLA: New features. *Computational Statistics & Data Analysis*, 67(0):68–83.

[Møller et al., 1998] Møller, J., Syversveen, A. R., and Waagepetersen, R. P. (1998). Log gaussian cox processes. *Scandinavian Journal of Statistics*, 25:451–482.

[Møller and Waagepetersen, 2003] Møller, J. and Waagepetersen, R. P. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, Boca Raton.

[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.

[Petris et al., 2009] Petris, G., Petroni, S., and Campagnoli, P. (2009). *Dynamic Linear Models with R*. Springer.

[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.

[Rozanov, 1977] Rozanov, J. A. (1977). Markov random fields and stochastic partial differential equations. *Math. USSR Sbornik*, 32(4)):515–534.

[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., 2017] Rue, H., Riebler, A. I., Sørbye, S. H., Illian, J. B., Simpson, D. P., and Lindgren, F. K. (2017). Bayesian computing with inla: A review. *Annual Review of Statistics and Its Application*, 4:395–421.

[Rue and Tjelmeland, 2002] Rue, H. and Tjelmeland, H. (2002). Fitting gaussian markov random fields to gaussian fields. *Scandinavian Journal of Statistics*, 29(1):31–49.

[Ruiz-Cárdenas et al., 2012] Ruiz-Cárdenas, R., Krainski, E. T., and Rue, H. (2012). Direct fitting of dynamic models using integrated nested laplace approximations — {INLA}. *Computational Statistics & Data Analysis*, 56(6):1808 – 1828.

[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., 2016] Simpson, D. P., Illian, J. B., Lindren, F., Sørbye, S. H., and Rue, H. (2016). Going off grid: computationally efficient inference for log-Gaussian Cox processes. *Biometrika*, 103(1):49–70.

[Simspon et al., 2017] Simspon, D. P., Rue, H., Riebler, A., Martins, T. G., and Sørbye, S. H. (2017). Penalising model component complexity: A principled, practical approach to constructing priors. *Statistical Science*.

[Sørbye and Rue, 2014] Sørbye, S. and Rue, H. (2014). Scaling intrinsic gaussian markov random field priors in spatial modelling. *Spatial Statistics*, 8.

[Taylor et al., 2013] Taylor, B. M., Davies, T. M., S., R. B., and Diggle, P. J. (2013). lgcp: An R package for inference with spatial and spatio-temporal log-Gaussian Cox processes. *Journal of Statistical Software*, 52(4):1–40.

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

[Vivar and Ferreira, 2009] Vivar, J. C. and Ferreira, M. A. R. (2009). Spatiotemporal models for gaussian areal data. *Journal of Computational and Graphical Statistics*, 18(3):658–674.

[West and Harrison, 1997] West, M. and Harrison, J. (1997). *Bayesian Forecasting and Dynamic Models*. Springer.