Bayesian Spatial and Spatio-temporal Modelling with R-INLA

Finn Lindgren
University of Bath, United Kingdom

Håvard Rue
Norwegian University of Science and Technology, Norway

Abstract

The principles behind the interface to continuous domain spatial models in the **R-INLA** software package for R are described. The Integrated Nested Laplace Approximation (INLA) approach proposed by Rue et al. (2009) is a computationally effective alternative to MCMC for Bayesian inference. INLA is designed for latent Gaussian models, a very wide and flexible class of models ranging from (generalized) linear mixed to spatial and spatio-temporal models. Combined with the Stochastic Partial Differential Equation approach (SPDE, Lindgren et al., 2011), one can easily accomodate all kinds of geographically referenced data, including areal and geostatistical ones, as well as spatial point process data. The implementation interface covers stationary spatial models, non-stationary spatial models, and also spatio-temporal models, and is applicable in epidemiology, ecology, environmental risk assessment, as well as general geostatistics.

Keywords: Bayesian inference, Gaussian Markov random fields, stochastic partial differential equations, Laplace approximation, R.

Traditionally, Markov models in image analysis and spatial statistics have been largely confined to discrete spatial domains, such as lattices and regional adjacency graphs. However, as discussed in Lindgren, Rue, and Lindström (2011), one can express a large class of random field models as solutions to continuous domain stochastic partial differential equations (SPDEs), and write down explicit links between the parameters of each SPDE and the elements of precision matrices for weights in a discrete basis function representation. As shown by Whittle (1963), such models include those with Matérn covariance functions, which are ubiquitous in traditional spatial statistics, but in contrast to covariance based models it is far easier to introduce non-stationarity into the SPDE models. This is because the differential operators act locally, similarly to local increments in Gibbs-specifications of Markov models, and only mild regularity conditions are required. The practical significance of this is that classical Gaussian random fields can be merged with methods based on the Markov property, providing continuous domain models that are computationally efficient, and where the parameters can be specified locally without having to worry about positive definiteness of covariance functions.

The following sections present the basic ingredients of the link between continuous domains and Markov models and related simulation free Bayesian inference methods (Section 1), and

describe the structure of the interface to using such models in the **R-INLA** software package (Section 2 and 3). Special emphasis is placed on the abstractions necessary to simplify the practical bookkeeping for the users of the software.

1. Spatial modelling and inference

This section describes the basic principles of the continuous domain spatial models and Bayesian inference methods in the **R-INLA** package (Rue, Martino, Lindgren, Simpson, and Riebler 2013b).

1.1. Continuous domain spatial Markov random fields

When building and using hierarchical models with latent random fields it is important to remember that the latent fields often represent real-world phenomena that exist independently of whether they are observed in a given location or not. Thus, we are not building models solely for *discretely observed data*, but for approximations of *entire processes* defined on continuous domains. For a spatial field $x(\boldsymbol{s})$, while the data likelihood typically depends only on the values at a finite set of locations, $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$, the model itself defines the joint behaviour for all locations, typically $\boldsymbol{s} \in \mathbb{R}^2$ or $\boldsymbol{s} \in \mathbb{S}^2$ (a sphere/globe). In the case of lattice data, the discretisation typically happens in the observation stage, such as integration over grid boxes (e.g., photon collection in a camera sensor). Often, this is approximated by point-wise evaluation, but there is nothing apart from computational challenges preventing other observation models.

As discussed in the introduction, an alternative to traditional covariance based modelling is to use SPDEs, but carry out the practical computations using Gaussian Markov random field (GMRF) representations. This is done by approximating the full set of spatial random functions with weighted sums of simple basis functions, which allows us to hold on to the continuous interpretation of space, while the computational algorithms only see discrete structures with Markov properties. Beyond the main paper Lindgren *et al.* (2011), this is further discussed by Simpson, Lindgren, and Rue (2012a,b).

Stationary Matérn fields

The simplest model for $x(\boldsymbol{s})$ currently implemented in **R-INLA** is the SPDE/GMRF version of the stationary Matérn family, obtained as the stationary solutions to

$$(\kappa^2 - \Delta)^{\alpha/2}(\tau x(\boldsymbol{s})) = \mathcal{W}(\boldsymbol{s}), \quad \boldsymbol{s} \in \Omega,$$

where Δ is the Laplacian, κ is the spatial scale parameter, α controls the smoothness of the realisations, τ controls the variance, and Ω is the spatial domain. The right-hand side of the equation, $\mathcal{W}(\boldsymbol{s})$ is Gaussian spatial white noise process. As noted by Whittle (1954, 1963), the stationary solutions on $\mathbb{R}^d$ have Matérn covariances,

$$\mathsf{Cov}(x(\mathbf{0}), x(\boldsymbol{s})) = \frac{\sigma^2}{2^{\nu-1}\Gamma(\nu)} (\kappa\|s\|)^\nu K_\nu(\kappa\|s\|). \tag{1}$$

The parameters in the two formulations are coupled so that the Matérn smoothness is $\nu =$

$\alpha - d/2$ and the marginal variance is

$$\sigma^2 = \frac{\Gamma(\nu)}{\Gamma(\alpha)(4\pi)^{d/2}\kappa^{2\nu}\tau^2}. \tag{2}$$

From this we can identify the exponential covariance with $\nu = 1/2$ and $\alpha = 3/2$, and note that fields with $\alpha \leq 1$ give $\nu \leq 0$ and that such fields have no point-wise interpretation (but do have well-defined integration properties). From spectral theory one can show that integer values for α gives continuous domain Markov fields (Rozanov 1982), and these are the easiest for which to provide discrete basis representations. In **R-INLA**, the default value is $\alpha = 2$, but $0 \leq \alpha < 2$ are also available, though not as extensively tested. For the non-integer α values the approximation method introduced in the authors' discussion response in Lindgren *et al.* (2011) is used.

The models discussed in Lindgren *et al.* (2011) and implemented in **R-INLA** are built on a basis representation

$$x(\boldsymbol{s}) = \sum_{k=1}^{n} \psi_k(\boldsymbol{s}) x_k, \tag{3}$$

where the joint distribution of $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ is chosen so that the distribution of the functions $x(\boldsymbol{s})$ approximates the distribution of solutions to the SPDE on the domain. To obtain a Markov structure, and to preserve it when conditioning on local observations, we use basis function with compact support. The construction is done by projecting the SPDE onto the basis representation in what is essentially a Finite Element method.

To allow easy and explicit evaluation, for two-dimensional domains we use piece-wise linear basis functions defined by a triangulation of the domain of interest. For one-dimensional domains, B-splines of degrees 1 (piecewise linear) and 2 (piecewise quadratic) are supported. This yields sparse matrices $\boldsymbol{C}$, $\boldsymbol{G}_1$, and $\boldsymbol{G}_2$ such that the appropriate precision matrix for the weights is given by

$$\boldsymbol{Q} = \tau^2(\kappa^4\boldsymbol{C} + 2\kappa^2\boldsymbol{G}_1 + \boldsymbol{G}_2)$$

for the default case $\alpha = 2$, so that the elements of $\boldsymbol{Q}$ have explicit expressions as functions of κ and τ. Assigning the Gaussian distribution $\boldsymbol{x} \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{Q}^{-1})$ now generates continuously defined functions $x(\boldsymbol{s})$ that are approximative solutions to the SPDE (in a stochastically weak sense).

The simplest internal representation of the parameters in the model interface is $\log(\tau) = \theta_1$ and $\log(\kappa) = \theta_2$, where θ_1 and θ_2 are assigned a joint normal prior distribution. Since τ and κ has a joint influence on the marginal variances of the resulting field, it is often more natural to construct the parameter model using the standard deviation σ and *range* ρ, where $\rho = (8\nu)^{1/2}/\kappa$ is the distance for with the correlation functions has fallen to approximately 0.13, for all $\nu > 1/2$. Translating this into τ and κ yields

$$\log\tau = \frac{1}{2}\log\left(\frac{\Gamma(\nu)}{\Gamma(\alpha)(4\pi)^{d/2}}\right) - \log\sigma - \nu\log\kappa, \tag{4}$$

$$\log\kappa = \frac{\log(8\nu)}{2} - \log\rho. \tag{5}$$

The internal parameterisation is then obtained by setting

$$\log \sigma = \log \sigma_0 + \theta_1, \tag{6}$$
$$\log \rho = \log \rho_0 + \theta_2. \tag{7}$$

Non-stationary fields

There is a vast range of possible extensions to the stationary SPDE described in the previous section, including non-stationary versions (see Lindgren *et al.* 2011; Bolin and Lindgren 2011, for examples). In the current version of the package, a non-stationary model defined via spatially varying $\kappa(\boldsymbol{s})$ and $\tau(\boldsymbol{s})$ is available for the case $\alpha = 2$. The SPDE is defined as

$$(\kappa(\boldsymbol{s})^2 - \Delta)(\tau(\boldsymbol{s})x(\boldsymbol{s})) = \mathcal{W}(\boldsymbol{s}), \quad \boldsymbol{s} \in \Omega,$$

and $\log \kappa(\boldsymbol{s})$ and $\log \tau(\boldsymbol{s})$ are defined as linear combinations of basis functions,

$$\log(\tau(\boldsymbol{s})) = b_0^\tau(\boldsymbol{s}) + \sum_{k=1}^{p} b_k^\tau(\boldsymbol{s})\theta_k,$$
$$\log(\kappa(\boldsymbol{s})) = b_0^\kappa(\boldsymbol{s}) + \sum_{k=1}^{p} b_k^\kappa(\boldsymbol{s})\theta_k.$$

The precision matrix is a simple modification of the stationary one, with the parameter fields (evaluated at the mesh discretisation points) entering via diagonal matrices:

$$\boldsymbol{T} = \mathrm{diag}(\tau(\boldsymbol{s_i})), \quad \boldsymbol{K} = \mathrm{diag}(\kappa(\boldsymbol{s_i})),$$
$$\boldsymbol{Q} = \boldsymbol{T}(\boldsymbol{K}^2\boldsymbol{C}\boldsymbol{K}^2 + \boldsymbol{K}^2\boldsymbol{G}_1 + \boldsymbol{G}_1^\top\boldsymbol{K}^2 + \boldsymbol{G}_2)\boldsymbol{T}.$$

Just as in the stationary case, the model can be reparameterised using Equation 4 and 5, where

$$\log(\sigma(\boldsymbol{s})) = b_0^\sigma(\boldsymbol{s}) + \sum_{k=1}^{p} b_k^\sigma(\boldsymbol{s})\theta_k,$$
$$\log(\rho(\boldsymbol{s})) = b_0^\rho(\boldsymbol{s}) + \sum_{k=1}^{p} b_k^\rho(\boldsymbol{s})\theta_k,$$

and $\sigma(\boldsymbol{s})$ and $\rho(\boldsymbol{s})$ are the *nominal* local standard deviations and correlation ranges. There are no explicit expressions for the actual values, since they depend on the entire parameter functions in a non-trivial way. For given values of $\boldsymbol{\theta}$, the marginal variances can be efficiently calculated using the discretised GMRF representation, see Section 2.3.

Given the offsets, $b_0^\sigma(\boldsymbol{s})$ and $b_0^\kappa(\boldsymbol{s})$, and basis functions. $b_k^\sigma(\boldsymbol{s})$ and $b_k^\kappa(\boldsymbol{s})$, for the $\log(\sigma(\boldsymbol{s}))$ and $\log(\rho(\boldsymbol{s}))$ parameter fields, the internal model representation can be constructed using the following identities:

$$b_0^\kappa(\boldsymbol{s}) = \frac{\log(8\nu)}{2} - b_0^\rho(\boldsymbol{s}), \tag{8}$$
$$b_k^\kappa(\boldsymbol{s}) = -b_k^\rho(\boldsymbol{s}), \tag{9}$$
$$b_0^\tau(\boldsymbol{s}) = \frac{1}{2}\log\left(\frac{\Gamma(\nu)}{\Gamma(\alpha)(4\pi)^{d/2}}\right) - b_0^\sigma(\boldsymbol{s}) - \nu b_0^\kappa(\boldsymbol{s}), \tag{10}$$
$$b_k^\tau(\boldsymbol{s}) = -b_k^\sigma(\boldsymbol{s}) - \nu b_k^\kappa(\boldsymbol{s}). \tag{11}$$

The constant $\Gamma(\nu)/(\Gamma(\alpha)(4\pi)^{d/2})$ is $1/2$ and $1/4$ for $d = 1$, $\alpha = 1$ and 2. For $d = 2$ and $\alpha = 2$ it is $1/(4\pi)$. There is experimental support for constructing basis functions that reduces the influence of the range on the variance for cases where the basis functions for $\log \rho(\boldsymbol{s})$ have rapid changes.

Boundary effects

When constructing solutions to the SPDEs on bounded domains, boundary conditions are imposed, but how to construct practical and proper stochastic boundary conditions for these models is an open research problem. In the current version of the package, all 2D models are restricted to deterministic Neumann boundaries (zero normal-derivatives), as this is easy to construct, has well defined physical interpretation in terms of *reflection*, and has an effect on the covariances that is easy to quantify. As a rule of thumb, the boundary effect is negligible at a distance ρ from the boundary, and the variance is inflated near the boundary by a factor 2 along straight boundaries, and by a factor 4 near right-angled corners. In practice one can therefore avoid the boundary effect by extending the domain of interest by a distance at least ρ, as well as avoid sharp corners. The built-in mesh generation routines (see Section 2.1) are designed to do this. For one-dimensional models, the boundaries can also be defined as Dirichlet (value zero at the boundary), free, or cyclic.

Space-time models

While no space-time models are currently implemented explicitly, it is possible to construct such models using general `code()` features. The most important method is to construct a Kronecker product model. Starting from a basis representation

$$x(\boldsymbol{s}, t) = \sum_k \psi_k(\boldsymbol{s}, t) x_k,$$

where each basis function is the product of a spatial and a temporal basi function, $\psi_k(\boldsymbol{s}, t) = \psi_i^{\boldsymbol{s}}(\boldsymbol{s})\psi_j^t(t)$, the space-time SPDE

$$\frac{\partial}{\partial t}(\kappa(\boldsymbol{s})^2 - \Delta)^{\alpha/2} (\tau(\boldsymbol{s})x(\boldsymbol{s}, t)) = \mathcal{W}(\boldsymbol{s}, t), \quad (\boldsymbol{s}, t) \in \Omega \times \mathbb{R}$$

generates a precision matrix for the weight vector $\boldsymbol{x}$ as $\boldsymbol{Q} = \boldsymbol{Q}_t \otimes \boldsymbol{Q}_{\boldsymbol{s}}$, where $\boldsymbol{Q}_{\boldsymbol{s}}$ is the precision for the previous purely spatial model, and $\boldsymbol{Q}_t$ is the precision corresponding to a one-dimensional random walk. Any temporal GMRF model can be used in this construction, and Section 3 contains examples for how to specify such models in **R-INLA**. See Cameletti, Lindgren, Simpson, and Rue (2012) for a case-study using a Kronecker-model based on a temporal AR(1) process, including the full `R` code (although note that the interface has evolved slightly since the case-study was implemented).

Kronecker models generate separable covariance functions. The internal representation of the SPDE precision structures however also permits construction of non-separable models, as long as the unknown parameters appear in the appropriate places. Non-separable models that can be constructed in this way includes special cases of the stochastic heat equation. Wrapper functions for constructing such models are expected to be added in the future.

1.2. Bayesian inference

The **R-INLA** package (Rue *et al.* 2013b) implements the Integrated Nested Laplace Approximation (INLA) method introduced by Rue, Martino, and Chopin (2009). This method performs direct numerical calculation of posterior densities in a large Latent Gaussian (LGM) sub-class of Bayesian hierarchical models, avoiding time-consuming Markov chain Monte Carlo simulations. The implementation covers models of the follwing form,

$$\begin{aligned}(\boldsymbol{\theta}) &\sim p(\boldsymbol{\theta})\\ (\boldsymbol{x} \mid \boldsymbol{\theta}) &\sim \mathrm{N}(\mathbf{0}, \boldsymbol{Q}(\boldsymbol{\theta})^{-1})\\ \eta_i &= \sum_j a_{ij} x_j\\ (y_i \mid \boldsymbol{x}, \boldsymbol{\theta}) &\sim p(y_i \mid \eta_i, \boldsymbol{\theta})\end{aligned}$$

where $\boldsymbol{y}$ is a data vector, $\boldsymbol{\eta}$ is a linear predictor, $\boldsymbol{x}$ is a latent Gaussian field, and $\boldsymbol{\theta}$ are (hyper)parameters.

The basic principle is to approximate the posterior density for $(\boldsymbol{\theta} \mid \boldsymbol{y})$ using a Gaussian approximation $\widetilde{p}(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{y})$ for the posterior for the latent field, evaluated at the posterior mode, $\boldsymbol{x}^*(\boldsymbol{\theta}) = \underset{\boldsymbol{x}}{\operatorname{argmax}}\, p(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{y})$,

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}) \propto \left.\frac{p(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{y})}\right|_{\boldsymbol{x}=\boldsymbol{x}^*(\boldsymbol{\theta})} \approx \left.\frac{p(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})}{\widetilde{p}(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{y})}\right|_{\boldsymbol{x}=\boldsymbol{x}^*(\boldsymbol{\theta})},$$

which is called a Laplace approximation. This allows approximate evaluation of the (unnormalised) posterior density for $\boldsymbol{\theta}$ at any point. The algorithm uses numerical optimisation to find the mode of the posterior. The marginal posteriors for each θ_i and x_i are then calculated using numerical integration over $\boldsymbol{\theta}$, with another Laplace approximation involved in the latent field marginal posterior calculations:

$$\begin{aligned}p(\theta_i \mid \boldsymbol{y}) &\approx \int \widetilde{p}(\boldsymbol{\theta} \mid \boldsymbol{y})\, \mathrm{d}\boldsymbol{\theta}_{-i},\\ p(x_i \mid \boldsymbol{y}) &\approx \int \widetilde{p}(x_i \mid \boldsymbol{\theta}, \boldsymbol{y})\, \widetilde{p}(\boldsymbol{\theta} \mid \boldsymbol{y})\, \mathrm{d}\boldsymbol{\theta}\end{aligned}$$

For more details and information, see Martins, Simpson, Lindgren, and Rue (2013).

The linear predictor

An important aspect of the package interface is how to specify the connection between the latent field $\boldsymbol{x}$ and the linear predictor $\boldsymbol{\eta}$. **R-INLA** uses a formula definitions similar to the standard one used for linear model estimation with `lm()`. The main difference is that random effects of all kinds, including smooth nonlinear effects, structured graph effects and spatial effects, are specified using terms `f()`, where the user specifies the properties of such effects. The first argument of each `f()` specifies what element of the latent effect should apply to each observation, either as a scalar location of covariate value (for smooth nonlinear effects) or as a direct component index.

The linear predictor $\eta_i = \mathrm{fixed}_i\beta + f(\mathrm{time}_i) + \mathrm{random}_i$ can be constructed using the formula

```
~ -1 + fixed + f(time, model="rw2") + f(random, model="iid")
```

Let $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ denote the covariate values for for the `fixed` and `time` effects, and let $\boldsymbol{z}^3$ denote random effect indices. We can then rewrite the model using mapping functions $h_k(\cdot)$ that denote the mapping from covariates or indices to the actual latent value for formula component k,

$$\begin{aligned} h_1(z_i^1) &= z_i^1 \beta \\ h_2(z_i^2) &= \text{smooth effect evaluated at } z_i^2 \\ h_3(z_i^3) &= \text{random effect component number } z_i^3 \\ \eta_i &= \sum_k h_k(z_i^k) \end{aligned}$$

The latent field is the joint vector of all latent Gaussian variables, including the linear covariate effect coefficient β. For missing values in the z-vectors, the h functions are defined to be zero.

Since this construction only allows each observation to directly depend on a single element from each $h_k(\cdot)$ effect, this does not cover the case when an effect is defined using a basis expansion such as Equation 3. To solve this, **R-INLA** can apply a second layer of linear combinations to the $\boldsymbol{\eta}$ predictor,

$$\boldsymbol{\eta}^* = \boldsymbol{A}\boldsymbol{\eta}, \tag{12}$$

where $\boldsymbol{A}$ is a user-defined sparse matrix. This allows the SPDE models to be treated as indexed random effects, and the mapping between the basis weights and function values is done by placing appropriate $\psi_j(\boldsymbol{s})$ values in the $\boldsymbol{A}$ matrix. See Section 2.2 for how to construct the part of the $\boldsymbol{A}$ matrix needed for an SPDE model, and Section 2.5 for how to set up the joint matrices needed for general models.

2. R interface

The **R-INLA** interface to the SPDE models described in the previous section is divided into five basic categories: 1) Mesh construction, 2) space mapping, 3) SPDE model construction, 4) plotting, and 5) INLA input and output structure bookkeeping.

Due mostly to the the complexity of building the binary executables that form the computational backbone of the **R-INLA** package, it is not available to install from CRAN, but can still be easily installed directly from its webpage (Rue *et al.* 2013b) from within R:

```
R> source("http://www.math.ntnu.no/inla/givemeINLA-testing.R")
R> library("INLA")
```

The package can later be upgraded to the latest development version with

```
R> inla.upgrade(testing=TRUE)
```

which contains the latest features and bug fixes. The non-testing version is updated less frequently. The package website `http://www.r-inla.org` contains more documentation, as well as a discussion forum. The recommended way to access the full source code is to clone the repository located at `http://code.google.com/p/inla/` (Rue, Martino, Lindgren, Simpson, and Riebler 2013a).

The major challenge when designing a general software package for practical use of the SPDE/GMRF models is that of *bookkeeping*, i.e. how to assist the user in keeping track of the links between continuous and discrete representations, in a way that frees the user from having to know the details of the implementation and internal storage. To solve this, a bit of abstraction is needed to avoid cluttering the interface with those details. Thus, instead of visibly keeping track of mappings between triangle mesh node indices and data locations, the user can use sparse matrices to encode these relationships, and wrapper functions are provided to manipulate these matrices and associated index and covariate vectors in ways suitable for the intended usage.

2.1. Mesh construction

The basic tools for building basis function representations are provided by the low level function `inla.mesh.create()` and the three high level functions `inla.nonconvex.hull(loc, ...)`, `inla.mesh.2d()` and `inla.mesh.1d()`. The latter function defines B-spline basis representations in one dimension (see Section 3.1 for an example). The remainder of this section gives a brief introduction to mesh generation for two-dimensional domains.

The aim is to create the triangulated mesh on top of which the SPDE/GMRF representation is to be built. The example here illustrates a common usage case, which is to have semi-randomly scattered observation locations in a region of space such that there is no physical boundary, just an limited observation region. When dealing with only covariances between data points, this distinction is often unimportant, but here it becomes a possibly vital part of the model, since the SPDE will exhibit boundary effects. In the **R-INLA** implementation, Neumann boundaries are used, which increases the variance near the boundary. If we intend to model a stationary field across the entire domain of observations, we must therefore extend the model domain far enough so that the boundary effects don't influence the observations. However, note that the reverse is also true: if there *is* a physical boundary, the boundary effects may actually be desirable. The function `inla.mesh.2d()` allows us to create a mesh with small triangles in the domain of interest, and use larger triangles in the extension used to avoid boundary effects. This minimises the extra computational work needed due to the extension.

```
R> m = 50
R> points = matrix(runif(m*2), m, 2)
R> mesh = inla.mesh.2d(
+    loc=points,
+    cutoff=0.05,
+    offset=c(0.1, 0.4),
+    max.edge=c(0.05, 0.5) )
```

The `cutoff` parameter is used to avoid building many small triangles around clustered input locations, `offset` specifies the size of the inner and outer extensions around the data locations, and `max.edge` specifies the maximum allowed triangle edge lengths in the inner domain and in the outer extension. The overall effect of the triangulation construction is that, if desired, one can have smaller triangles, and hence higher accuracy of the field representation, where the observation locations are dense, larger triangles where data is more sparse (and hence provides less detailed information), and large triangles where there is no data and spending

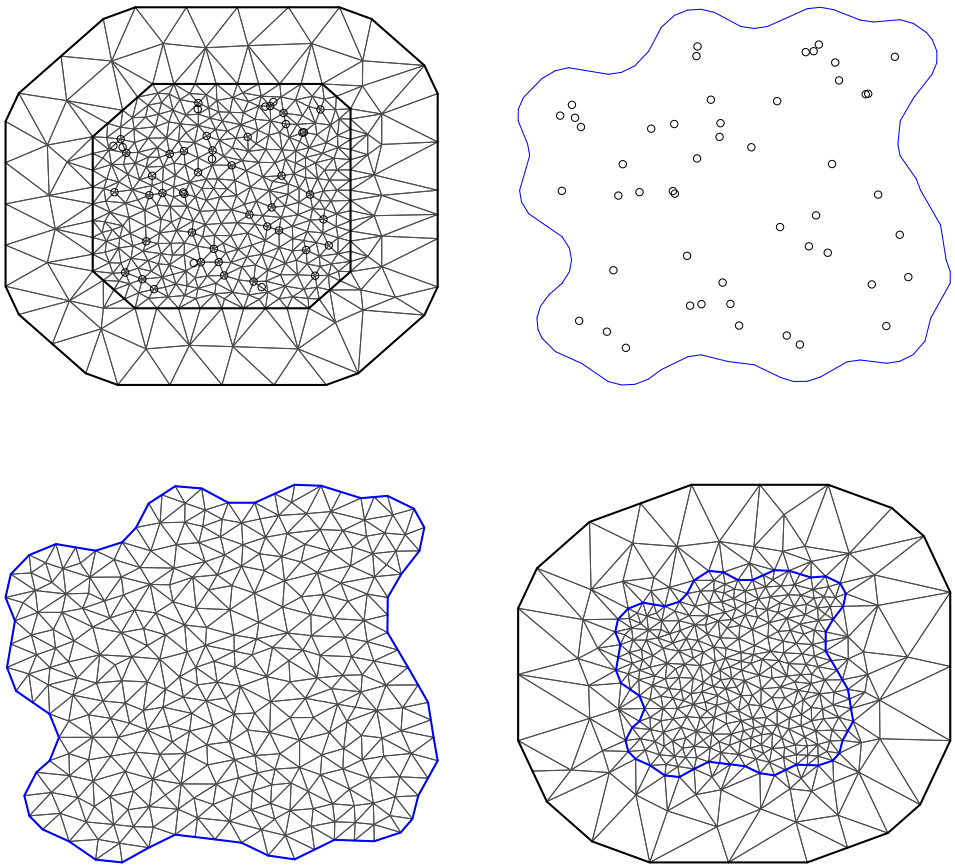

Figure 1: Illustrations of meshes constructed based on a common set of domain definition points but different mesh generation parameters.

computational resources would be wasteful. However, note that there is neither any guarantee nor any requirement that the observation locations are included as nodes in the mesh. If one so desires, the mesh can be designed from different principles, such as lattice points with no relation to the precise measurement locations. This emphasises the decoupling between the continuous domain of the field model and the discrete data locations.

A new feature is the option to compute a non-convex covering to use as boundary information,

```
R> bnd = inla.nonconvex.hull(points, convex=0.12)
R> mesh = inla.mesh.2d(
+    boundary=bnd,
+    cutoff=0.05,
+    offset = c(1, 0.5),
+    max.edge=c(0.1, 0.5) )
```

The resulting mesh is shown in the lower left panel of Figure 1.

2.2. Mapping between meshes and continuous space

One of the most important features is the `inla.spde.make.A()` functions, which computes the sparse weight matrices needed to map between the internal representation of weights for basis functions and the values of the resulting functions and fields. The basic syntax is

```
R> A = inla.spde.make.A(mesh, loc=points)
```

which produces a matrix with $A_{ij} = \psi_j(\boldsymbol{s}_i)$ for all points $\boldsymbol{s}_i$ in `points`.

Models in **R-INLA** can have several replicates, as well as being *grouped*, which corresponds to Kronecker product models. In order to obtain the correct `A` matrix for such models, the user can specify indices for the parameters `group` and `repl`. A recently introduced feature also allows specifying a one dimensional `group.mesh`, which is then interpreted as defining a Kronecker product basis, such as for the space-time models mentioned in Section 1.1, and an example is given in Section 3.2.

2.3. SPDE model construction

As the theory and practice evolves, new internal SPDE representation models are implemented. The current development focuses on models with internal name `spde2`. To find out what the currently available user-accessible models are, use the function `inla.spde.models()`.

Defining an SPDE model object can now be as simple as

```
R> spde = inla.spde2.matern(mesh, alpha=2)
```

but in practice we need to also specify the prior distribution for the parameters, and/or modify the parameterisation to suit the specific situation. This is true in particular when the models are used as simple smoothers, as there is then rarely enough information in the likelihood to fully identify the parameters, giving more importance to the prior distributions.

Using the theory from Section 1.1, the empirically derived range expression $\rho = \sqrt{8\nu}/\kappa$ allows for construction of a model with known range and variance $(= 1)$ for $(\theta_1, \theta_2) = (0, 0)$, via

```
R> sigma0 = 1
R> size = min(c(diff(range(mesh$loc[,1])), diff(range(mesh$loc[,2]))))
R> range0 = size/5
R> kappa0 = sqrt(8)/range0
R> tau0 = 1/(sqrt(4*pi)*kappa0*sigma0)
R> spde = inla.spde2.matern(mesh,
+    B.tau=cbind(log(tau0), -1, +1),
+    B.kappa=cbind(log(kappa0), 0, -1),
+    theta.prior.mean=c(0,0),
+    theta.prior.prec=c(0.1, 1) )
```

Here, `sigma0` is the field standard deviation and `range0` is the spatial range for $\boldsymbol{\theta} = \mathbf{0}$, and `B.tau` and `B.kappa` are matrices storing the parameter basis functions introduced in Section 1.1. For stationary models, only the first matrix row needs to be supplied. In this example, the prior median for the spatial range is chosen heuristically to be a fifth of the approximate domain diameter.

Setting suitable priors for $\boldsymbol{\theta}$ in these models in general a difficult problem. The heuristic used above is to specify a fairly vague prior for θ_1 which controls the variance, with σ_0^2 being

the median prior variance, and a larger prior precision for θ_2. When `range0` is a fifth of the domain size, the precision 1 for θ_2 gives an approximate 95% prior probability for the range being shorter than the domain size. Experimental helper functions for constructing parametrisations and priors are included in the package.

Models with range larger than the domain size are usually indistinguishable from intrinsic random fields, which can be modelled by fixing κ to zero (or rather some small positive value) with `B.tau=cbind(log(tau0),1)` and `B.kappa=cbind(log(small),0)`. Note that the sum-to-zero constraints often used for lattice based intrinsic Markov models is inappropriate due to the irregular mesh structure, and a *weighted* sum-to-zero constraint is needed to reproduce such models. The option `constr=TRUE` to the `inla.spde.matern()` call to apply an integrate-to-zero constraint. Further integration constraints can be specified using `extraconstr.int=list(A=A, e=e)` option, which implements constraints of the form

$$\boldsymbol{ACx} = \boldsymbol{e},$$

where $\boldsymbol{C}$ is the sparse matrix with elements $C_{ij} = \int_\Omega \psi_i(\boldsymbol{s})\psi_j(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}$. Non-integration constraints can be supplied with `extraconstr`, and all constraints will be passed on automatically to the `inla()` call later.

Properties and sampling

There are several helper functions for querying properties about `spde` model objects, the most important one being `inla.spde.precision()`. To obtain the precision for the constructed model, with standard deviation a factor 3 larger than the prior median value, and range equal to the prior median, use

```
R> Q = inla.spde.precision(spde, theta=c(log(3), 0))
```

The following code then generates two samples from the model,

```
R> x = inla.qsample(n=2, Q)
```

and the resulting fields are shown in Figure 2. To take any constraints specified in the `spde` object into account when sampling, use

```
R> x = inla.qsample(n=2, Q, constr=spde$f$extraconstr)
```

Obtaining covariances is a much more costly operation, but the function `inla.qinv(Q)` can quickly calculate all covariances between neighbours (in the Markov sense), including the marginal variances. Finally, `inla.qsolve(Q,b)` uses the internal **R-INLA** methods for solving a linear system involving $\boldsymbol{Q}$.

2.4. Plotting

Mesh structure

The interface supports a `plot()` function aimed at plotting the basic structure of a triangulation mesh. By specifying `rgl=TRUE`, the **rgl** plotting system is used, which is useful in particular for spherical domains. Variations of the following commands were used to produce Figure 1:

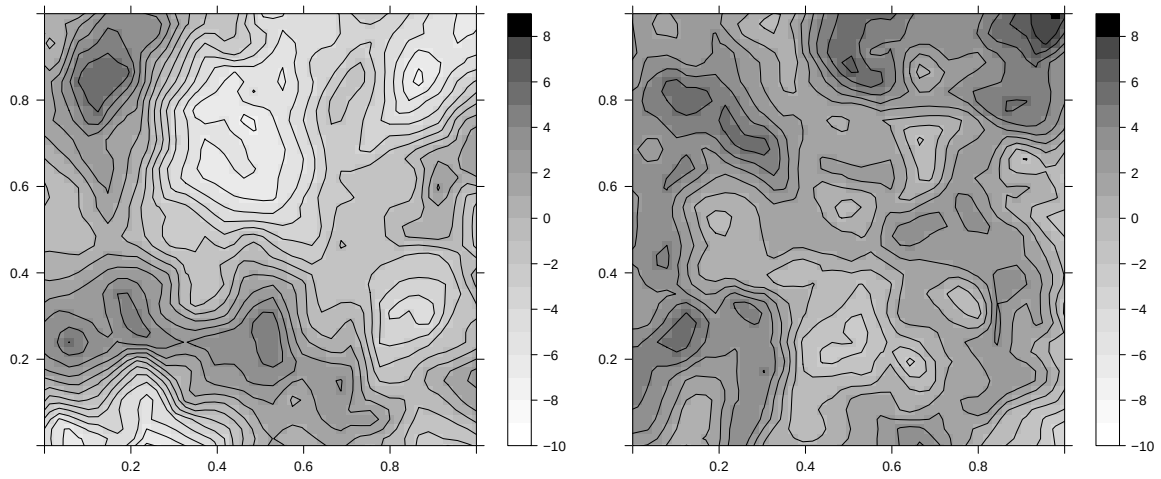

Figure 2: The two random field samples, with only the domain of interest, $[0,1]\times[0,1]$, shown.

```
R> plot(mesh)
R> plot(mesh, rgl=TRUE)
R> lines(mesh$segm$bnd, mesh$loc, add=FALSE)
```

Spatial fields

For plotting fields defined on meshes, one option is to use the `rgl` option, which supports specifying color values for the nodes, producing an interpolated field plot, and optionally draw triangle edges and vertices in the same plot:

```
R> plot(mesh, rgl=TRUE, col=x[,1],
+       color.palette=function(n) grey.colors(n, 1, 0),
+       draw.edges=FALSE, draw.segments=TRUE, draw.vertices=FALSE)
```

The more common option is to explicitly evaluate the field on a regular lattice, and use any matrix-based plotting mechanism, such as `image()`:

```
R> proj = inla.mesh.projector(mesh, dims=c(100, 100))
R> image(proj$x, proj$y, inla.mesh.project(proj, field=x[,1]))
```

All the figures showing fields have been drawn using a wrapper around the `levelplot()` from the **lattice** package, and which is available in the supplementary material.

The `inla.mesh.project/or()` functions are here used to map between the basis function weights for the mesh nodes and points on a regular grid, by default a 100×100 lattice covering the mesh domain. As illustrated in Figure 3, the functions also support several types of projections for spherical domains.

2.5. Advanced predictor manipulation

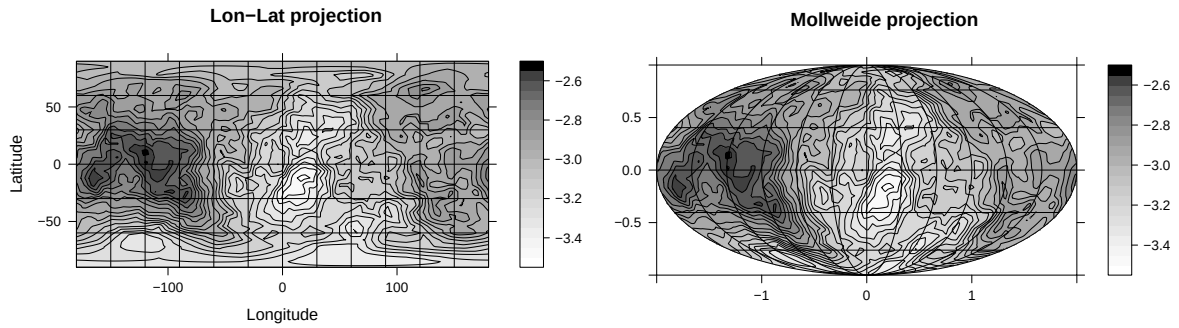

Figure 3: Projections of a sample from an SPDE model on a spherical domain. The left panel uses longitude-latitude projection, and the right hand panel uses the equal area Mollweide projection.

To aid the user in setting up an appropriate $\boldsymbol{A}$ for a whole model, the function `inla.stack()` can be used. The function is meant to hide most of the tedious vector and manipulation that is necessary, and from the users point of view can be seen as implementing a few abstract operations on effect and predictor definitions. Section 2.6 contains a practical example showing the practical use of this feature, while here we describe the abstract operations.

First, all effects are conceptually joined into a compact matrix-like notation:

$$\begin{aligned}
\boldsymbol{Z} &= \left[\boldsymbol{z}^1 \quad \dots \quad \boldsymbol{z}^K\right], \\
\boldsymbol{\eta} &= H(\boldsymbol{Z}) = \sum_k h_k(\boldsymbol{z}^k), \\
\boldsymbol{\eta}^* &= \boldsymbol{A}\, H(\boldsymbol{Z}),
\end{aligned}$$

where $h_k(\cdot)$ are the effect mapping functions defined in Section 1.2 and specified using a model `formula`. The effects are treated as *named* vectors, regardless of the ordering. Any effect known to H but not present in a particular $\boldsymbol{Z}$ is treated as "no effect", which is the same effect as when providing `NA` values.

The first operation is to construct *sums* of predictors (shown here only for two predictors):

$$\begin{aligned}
\boldsymbol{\eta}^* &= \boldsymbol{A}_1\, H(\boldsymbol{Z}_1) + \boldsymbol{A}_2\, H(\boldsymbol{Z}_2) = \widetilde{\boldsymbol{A}}\, H(\widetilde{\boldsymbol{Z}}), \\
\widetilde{\boldsymbol{Z}} &= \begin{bmatrix} \boldsymbol{Z}_1 \\ \boldsymbol{Z}_2 \end{bmatrix}, \\
\widetilde{\boldsymbol{A}} &= \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \end{bmatrix}.
\end{aligned}$$

The joining of the $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ effect collections is performed by matching vector names, adding `NA` for any missing components.

The second operation is to *join* predictors in sequence (only two predictors shown):

$$\begin{aligned}
\boldsymbol{\eta}_1^* &= \boldsymbol{A}_1\, H(\boldsymbol{Z}_1),\\
\boldsymbol{\eta}_2^* &= \boldsymbol{A}_2\, H(\boldsymbol{Z}_2),\\
\widetilde{\boldsymbol{\eta}}^* &= (\boldsymbol{\eta}_1^*, \boldsymbol{\eta}_2^*) = \widetilde{\boldsymbol{A}}\, H(\widetilde{\boldsymbol{Z}}),\\
\widetilde{\boldsymbol{Z}} &= \begin{bmatrix} \boldsymbol{Z}_1 \\ \boldsymbol{Z}_2 \end{bmatrix},\\
\widetilde{\boldsymbol{A}} &= \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_2 \end{bmatrix}.
\end{aligned}$$

As a post-processing step for both operations, the new covariate matrix $\widetilde{\boldsymbol{Z}}$ and matrix $\widetilde{\boldsymbol{A}}$ are analysed to detect any duplicate rows in $\widetilde{\boldsymbol{Z}}$ or any all-zero columns in $\widetilde{\boldsymbol{A}}$, and those are by default removed in order to minimise the internal size of the model representation.

The syntax for a *sum* operation is

```
R> stack = inla.stack(data=list(...),
+                     A=list(A1, A2, ...),
+                     effects=list(list(...),
+                                  list(...),
+                                  ...),
+                     tag=...)
```

where each `A` matrix has an associated list of effects. A *join* operation is performed by supplying two or more previously generated stack objects,

```
R> stack = inla.stack(stack1, stack2, ...)
```

Any vectors specified in the `data` list, most importantly the response variable vector itself, should be the same length as the predictor itself (scalars are replicated to the appropriate length). With the help of the name-`tag` it also keeps track of the indices needed to map from the original inputs into the resulting *stacked* representation. See Section 2.6 for an illustrating example of using `inla.stack()`.

Note that $H(\cdot)$ is conceptually defined by the model `formula`, which needs to mention every covariate component present in $\boldsymbol{Z}$ and that is meant to be used.

2.6. Bayesian inference

In this section we will look at a simple example of how to use the SPDE models in latent Gaussian models when doing direct Bayesian inference based on integrated nested Laplace approximation method described in Section 1.2.

Consider a simple Gaussian linear model involving two independent realisations (replicates) of a latent spatial field $x(\boldsymbol{s})$, observed at the same m locations, $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$, for each replicate. For each $i = 1, \ldots, m$,

$$\begin{aligned}
y_i &= \beta_0 + c_i\beta_c + x_1(\boldsymbol{s}_i) + e_i,\\
y_{i+m} &= \beta_0 + c_{i+m}\beta_c + x_2(\boldsymbol{s}_i) + e_{i+m},
\end{aligned}$$

where c_i is an observation-specific covariate, e_i is measurement noise, and $x_1(\cdot)$ and $x_2(\cdot)$ are the two field replicates. Note that the intercept, β_0, can be interpreted as a spatial covariate effect, constant over the domain.

We use the basis function representation of $x(\cdot)$ to define a sparse matrix of weights $\boldsymbol{A}$ such that $x(\boldsymbol{s}_i) = \sum_j A_{ij} x_j$, where $\{x_j\}$ is the joint set of weights for the two replicate fields. If we only had one replicate, we would have $A_{ij} = \psi_j(\boldsymbol{s}_i)$. The matrix can be generated by `inla.spde.make.A()`, which locates the points in the mesh and organises the evaluated values of the basis functions for the two replicates:

```
R> A = inla.spde.make.A(mesh,
+                       loc = points,
+                       index = rep(1:m, times=2),
+                       repl = rep(1:2, each=m) )
```

For each observation, `index` gives the corresponding index into the matrix of measurement locations, and `repl` determines the corresponding replicate index. In case of missing observations, one can either keep this `A`-matrix while setting the corresponding elements of the data vector `y` to `NA`, or omit the corresponding elements from `y` as well as from the `index` and `repl` parameters above. Also note that the row-sums of $\boldsymbol{A}$ are 1, since the piece-wise linear basis functions we use sum to 1 at each location.

Rewriting the observation model on vector form gives

$$\begin{aligned}\boldsymbol{y} &= \mathbf{1}\beta_0 + \boldsymbol{c}\beta_c + \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e} \\ &= \boldsymbol{A}(\boldsymbol{x} + \mathbf{1}\beta_0) + \boldsymbol{c}\beta_c + \boldsymbol{e}\end{aligned}$$

Using the helper functions, we can generate data using our two previously simulated model replicates,

```
R> x = as.vector(x)
R> covariate = rnorm(m*2)
R> y = 5 + covariate*2 + as.vector(A %*% x) + rnorm(m*2)*0.1
```

The `formula` in `inla()` defines a linear predictor $\boldsymbol{\eta}$ as the sum of all effects, and an `NA` in a covariate or index vector is interpreted as *no effect*. To accommodate predictors that involve more than one element of a random effect, one can specify a sparse matrix of weights defining an arbitrary linear combination of the elements of $\boldsymbol{\eta}$, giving a new predictor vector $\boldsymbol{\eta}^*$. The linear predictor output from `inla()` then contains the joint vector $(\boldsymbol{\eta}^*, \boldsymbol{\eta})$. To implement our model, we separate the spatial effects from the covariate by defining

$$\boldsymbol{\eta}_e = \begin{bmatrix} \boldsymbol{x} + \mathbf{1}\beta_0 \\ \boldsymbol{c}\beta_c \end{bmatrix},$$

and construct the predictor as

$$\begin{aligned}\boldsymbol{\eta}_e^* &= \boldsymbol{A}(\boldsymbol{x} + \mathbf{1}\beta_0) + \boldsymbol{c}\beta_c \\ &= \begin{bmatrix}\boldsymbol{A} & \boldsymbol{I}\end{bmatrix} \boldsymbol{\eta}_e = \boldsymbol{A}_e \boldsymbol{\eta}_e\end{aligned}$$

so that now $\mathsf{E}(\boldsymbol{y} \mid \boldsymbol{\eta}_e) = \boldsymbol{\eta}^*_e$. The bookkeeping required to describe this to `inla()` involves concatenating matrices and adding `NA` elements to the covariates and index vectors:

$$\begin{aligned} \boldsymbol{A}_e &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{I}_{2m} \end{bmatrix} \\ \texttt{field0} &= (1, \ldots, n, 1, \ldots, n) \\ \texttt{field} &= (\texttt{field0}, \texttt{NA}, \ldots, \texttt{NA}) \\ \texttt{intercept} &= (1, \ldots, 1, \texttt{NA}, \ldots, \texttt{NA}) \\ \texttt{cov} &= (\texttt{NA}, \ldots, \texttt{NA}, c_1, \ldots, c_{2m}) \end{aligned}$$

Doing this by hand with `cBind()`, `c()`, and `rep()` quickly becomes tedious and error-prone, so one can instead use the helper function `inla.stack()`, which takes blocks of data, weight matrices, and effects and joins them, adding `NA` where needed. Identity matrices and constant covariates can be abbreviated to scalars, with a complaint being issued if the input is inconsistent or ambiguous.

We also need to keep track of the two field replicates, and use `inla.spde.make.index()`, which gives a list of index vectors for indexing the full mesh and its replicates (it can also be used for indexing Kronecker product `group` models, e.g., in simple multivariate and spatio-temporal models). The code

```
R> mesh.index = inla.spde.make.index(name="field",
+                                    n.spde=spde$n.spde,
+                                    n.repl=2)
```

generates a list `mesh.index` with three index vectors,

$$\begin{aligned} \texttt{field} &= (1, \ldots, n, 1, \ldots, n), \\ \texttt{field.repl} &= (1, \ldots, 1, 2, \ldots, 2), \\ \texttt{field.group} &= (1, \ldots, 1, 1, \ldots, 1). \end{aligned}$$

The predictor information for the observed data can now be collected, using

```
R> st.est = inla.stack(data=list(y=y),
+                      A=list(A, 1),
+                      effects=list(c(mesh.index, list(intercept=1)),
+                                   list(cov=covariate)),
+                      tag="est")
```

where the `tag` identifier can later be used for identifying the correct indexing into the `inla()` output. As discussed in Section 2.5, each "`A`" matrix must have an associated list of "`effects`", in this case `A`:(`field`, `field.repl`, `field.group`, `intercept`) and `1`:(`cov`). The `data` list may contain anything associated with the "left hand side" of the model, such as exposure `E` for Poisson likelihoods. By default, duplicates in the effects are identified and replaced by single copies (`compress=TRUE`), and effects that do not affect $\boldsymbol{\eta}^*$ are removed completely (`remove.unused=TRUE`), so that each column of the resulting `A` matrix has a least one non-zero element.

If we want to obtain the posterior prediction of the combined spatial effects at the mesh nodes, $x(\boldsymbol{s}_i) + \beta_0$, we can define

$$\begin{aligned}\boldsymbol{\eta}_p &= \boldsymbol{x} + \mathbf{1}\beta_0 \\ \boldsymbol{\eta}_p^* &= \boldsymbol{I}\boldsymbol{\eta}_p = \boldsymbol{A}_p\boldsymbol{\eta}_p\end{aligned}$$

and construct the corresponding information stack with

```
R> st.pred = inla.stack(data=list(y=NA),
+                       A=list(1),
+                       effects=list(c(mesh.index, list(intercept=1))),
+                       tag="pred")
```

We can now join the estimation and prediction stack into a single stack,

```
R> stack = inla.stack(st.est, st.pred)
```

and the result is simplified by removing duplicated effects:

$$\begin{aligned}\boldsymbol{\eta}_s^* &= \begin{bmatrix}\boldsymbol{A}_e & \mathbf{0} \\ \mathbf{0} & \boldsymbol{A}_p\end{bmatrix}\begin{bmatrix}\boldsymbol{\eta}_e \\ \boldsymbol{\eta}_p\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{A} & \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{x} + \mathbf{1}\beta_0 \\ \boldsymbol{c}\beta_c \\ \boldsymbol{x} + \mathbf{1}\beta_0\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{A} & \boldsymbol{I} \\ \boldsymbol{I} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{x} + \mathbf{1}\beta_0 \\ \boldsymbol{c}\beta_c\end{bmatrix} \\ &= \boldsymbol{A}_s\boldsymbol{\eta}_s\end{aligned}$$

In this simple example, the second block row of $\boldsymbol{A}_s$ (generating $\boldsymbol{x} + \mathbf{1}\beta_0$) isn't strictly needed, since the same information would be available in $\boldsymbol{\eta}_s$ itself if we specified `remove.unused=FALSE` when constructing `stack.pred` and `stack`, but in general such special cases can be hard to keep track of.

We are now ready to do the actual estimation. Note that we must explicitly remove the default intercept from the $\boldsymbol{\eta}$-model, since that would otherwise be applied twice in the construction of $\boldsymbol{\eta}^*$, and the constant covariate `intercept` is used instead:

```
R> formula =
+    y ~ -1 + intercept + cov + f(field, model=spde, replicate=field.repl)
R> inla.result = inla(formula,
+                     data=inla.stack.data(stack, spde=spde),
+                     family="normal",
+                     control.predictor=list(A=inla.stack.A(stack),
+                                            compute=TRUE))
```

The function `inla.stack.data()` produces the list of variables needed to evaluate the formula and `inla.stack.A()` extracts the $\boldsymbol{A}_s$ matrix.

Since the SPDE-related contents of `inla.result` can be hard to interpret, the helper function `inla.spde2.result()` can be used to extract the relevant bits and transform them into more user-friendly information, such as posteriors for range and variance instead of raw distributions for θ:

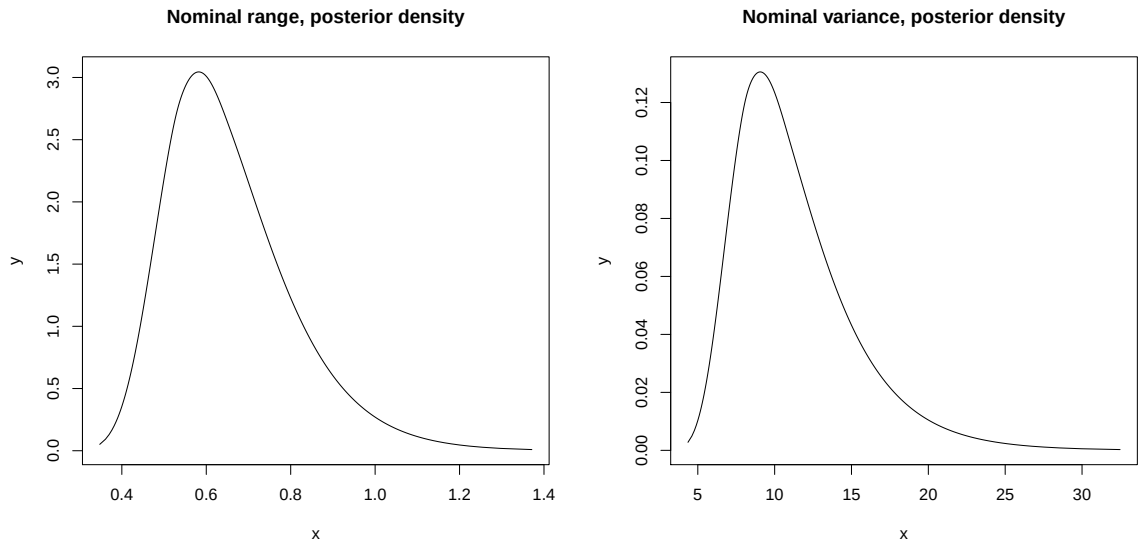

Figure 4: Posterior densities for nominal range and variance. The true values were 0.44 and 9.

```
R> result = inla.spde2.result(inla.result, "field", spde)
R> plot(result[["marginals.range.nominal"]][[1]], type="l",
+       main="Nominal range, posterior density")
```

The posterior means and standard deviations for the latent fields can be extracted and plotted as follows, where `inla.stack.index()` provides the necessary mappings between the `inla()` output and the original data stack specifications:

```
R> index = inla.stack.index(stack, "pred")$data
R> linpred.mean = inla.result[["summary.linear.predictor"]]$mean
R> image(proj$x, proj$y, inla.mesh.project(proj,
+        linpred.mean[index[mesh.index$field.repl==1]]))
```

3. Further examples

3.1. Non-Gaussian data: Tokyo rainfall

This is a simple example for using a 1D SPDE model with a 2nd order B-spline basis representation. The desired model is an intrinsic 2nd order random walk, but since there is no `inla.spde2.intrinsic()`, the details are setup explicitly.

First, a 2nd order B-spline basis mesh with 24 basis functions and cyclic boundary conditions is defined, and an appropriate `spde` model is constructed. The mesh is specified to have cyclic boundary conditions,

```
R> data("Tokyo")
```

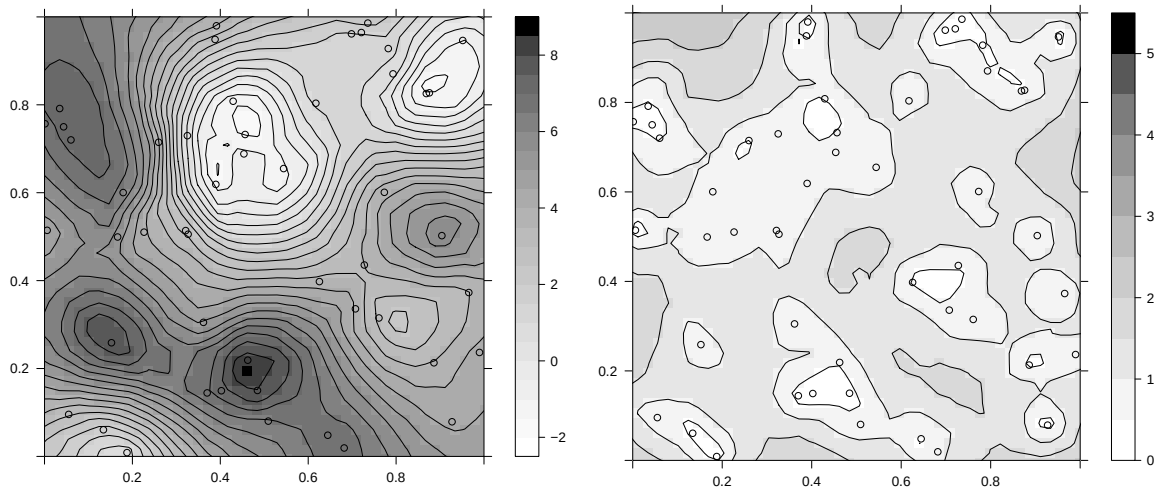

Figure 5: Posterior mean (left) and standard deviation (right) for $\beta_0 + x_1(\boldsymbol{s})$.

```
R> knots = seq(1, 367, length=25)
R> mesh = inla.mesh.1d(knots, interval=c(1, 367), degree=2, boundary="cyclic")
```

and the prior median for *tau* is calculated to give a specified prior median standard deviation when κ is fixed to a small value,

```
R> sigma0 = 1
R> kappa0 = 1e-3
R> tau0 = 1/(4*kappa0^3*sigma0^2)^0.5
```

and finally an `inla.spde2` object is created:

```
R> spde = inla.spde2.matern(mesh, constr=FALSE,
+                           B.tau = cbind(log(tau0), 1),
+                           B.kappa = cbind(log(kappa0), 0),
+                           theta.prior.prec = 1e-4)
```

Next, the data is organised using `inla.stack()`:

```
R> A = inla.spde.make.A(mesh, loc=Tokyo$time)
R> time.index = inla.spde.make.index("time", n.spde=spde$n.spde)
R> stack = inla.stack(data=list(y = Tokyo$y, link=1, Ntrials=Tokyo$n),
+                     A=list(A),
+                     effects=list(time.index),
+                     tag="est")
R> formula = y ~ -1 + f(time, model=spde)
R> data = inla.stack.data(stack)
R> result = inla(formula, family="binomial", data=data,
```

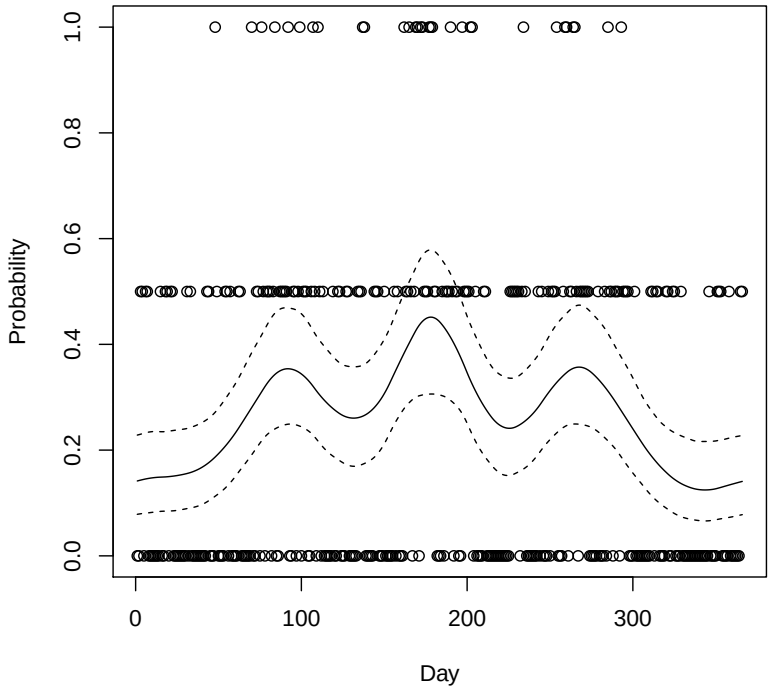

Figure 6: Empirical and model based Binomial probability estimates for the Tokyo rainfall data set, with 95% posterior predictive bounds.

```
+                Ntrials=data$Ntrials,
+                control.predictor=list(A=inla.stack.A(stack),
+                                       link=data$link,
+                                       compute=TRUE))
```

Finally, the results can be plotted, as shown in Figure 6:

```
R> time = 1:366
R> index = inla.stack.index(stack, "est")$data
R> plot(Tokyo$time, Tokyo$y/Tokyo$n, xlab="Day", ylab="Probability")
R> lines(time, result$summary.fitted.values$mean[index])
R> lines(time, result$summary.fitted.values$"0.025quant"[index], lty=2)
R> lines(time, result$summary.fitted.values$"0.975quant"[index], lty=2)
```

3.2. Kronecker product models for space-time

The following illustrates the principles for how to set up a kronecker product space-time model where both space and time are treated continuously, with a Matérn model in space and a stationary AR(1) model on coefficients for 2nd order B-splines in time. Assumed inputs are

```
  y           : measurements
  station.loc : coordinates for measurement stations
  station.id  : for each measurement, which station index?
  time        : for each measurement, what time?
```

The time interval for the model is $[0, 100]$.

```
R> mesh1 = inla.mesh.1d(loc=seq(0, 100, length=11), degree=2, boundary="free")
R> mesh2 = inla.mesh.2d(...)
R> spde = inla.spde2.matern(mesh2, alpha=2, ...)
R> index = inla.spde.make.index("space", n.spde=spde$n.spde, n.group=mesh1$m)
R> formula = y ~ -1 + f(space, model=spde, group=space.group,
+                       control.group=list(model="ar1"))
R> A = inla.spde.make.A(mesh2, loc=station.loc,
+                       index=station.id, group=time,
+                       group.mesh=mesh1)
R> stack = inla.stack(data = list(y=y), A = list(A), effects = list(index))
```

3.3. Multiple likelihoods

For models with data from more than one likelihood, `inla.stack()` can help in structuring the inputs, greatly reducing the amount of hand-tooled `rep()` statements needed to setup the input data structures. The following example shows how to setup a model where `y1` are Binomial and `y2` are Poisson, but the formula connects them via the linear predictor effects:

```
R> stack1 = inla.stack(data=list(Y=cbind(y1, NA), link=1, N=N1), ...)
R> stack2 = inla.stack(data=list(Y=cbind(NA, y2), link=2, E=E2), ...)
R> stack = inla.stack(stack1, stack2)
R> data = inla.stack.data(stack)
R> formula = Y ~ ...
R> result = inla(formula, data=data, family = c("binomial", "poisson"),
+                Ntrials = data$N,
+                E = data$E,
+                control.predictor(A=inla.stack.A(stack),
+                                  link=data$link,
+                                  compute=TRUE))
```

4. Closing

As a final note, the **R-INLA** package is in constant development, with new models added as they are needed and developed. The current work for the SPDE models is focusing on construction of parameter basis functions and priors for non-stationary models. An associated package for computing level excursion sets with joint excursion probabilities, as well as credible regions for contour curves, is being prepared for inclusion in CRAN (Bolin and Lindgren 2013).

Acknowledgements

The authors wish to thank their collaborators David Bolin, Michela Cameletti, Janine Illian, Johan Lindström, Thiago Martins, Daniel Simpson, Sigrunn Sørbye, Elias Krainski, and Ryan Yue, who have all contributed with ideas and suggestions for the development of the spatial model interface.

References

Bolin D, Lindgren F (2011). "Spatial models generated by nested stochastic partial differential equations, with an application to global ozone mapping." *The Annals of Applied Statistics*, **5**(1), 523–550. ISSN 1932-6157.

Bolin D, Lindgren F (2013). "Excursion and contour uncertainty regions for latent Gaussian models." Accepted, to appear. Submitted version at arXiv:1211.3946 [stat.ME].

Cameletti M, Lindgren F, Simpson D, Rue H (2012). "Spatio-temporal modeling of particulate matter concentration through the SPDE approach." *AStA Advances in Statistical Analysis*. ISSN 1863-8171.

Lindgren F, Rue H, Lindström J (2011). "An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach." *Journal of the Royal Statistical Society B*, **73**(4), 423–498. ISSN 1369-7412.

Martins T, Simpson D, Lindgren F, Rue H (2013). "Bayesian computing with INLA: new features." *Computational Statistics and Data Analysis*. Accepted for publication.

Rozanov A (1982). *Markov Random Fields*. Springer Verlag, New York.

Rue H, Martino S, Chopin N (2009). "Approximate Bayesian Inference for Latent Gaussian Models using Integrated Nested Laplace Approximations (with discussion)." **71**, 319–392.

Rue H, Martino S, Lindgren F, Simpson D, Riebler A (2013a). *Code repository for **R-INLA**, Approximate Bayesian inference using integrated nested Laplace approximations*. URL http://code.google.com/p/inla/.

Rue H, Martino S, Lindgren F, Simpson D, Riebler A (2013b). ***R-INLA**, Approximate Bayesian inference using integrated nested Laplace approximations*. Trondheim, Norway. URL http://www.r-inla.org/.

Simpson D, Lindgren F, Rue H (2012a). "In order to make spatial statistics computationally feasible, we need to forget about the covariance function." *Environmetrics*, **23**(1), 65–74. ISSN 1180-4009.

Simpson D, Lindgren F, Rue H (2012b). "Think continuous: Markovian Gaussian models in spatial statistics." *Spatial Statistics*, **1**, 16–29. ISSN 2211-6753.

Whittle P (1954). "On stationary processes in the plane." *Biometrika*, **41**(3/4), 434–449.

Whittle P (1963). "Stochastic processes in several dimensions." *Bull. Inst. Internat. Statist.*, **40**, 974–994.

Affiliation:

Finn Lindgren
Department of Mathematical Sciences
University of Bath
Claverton Down
BA2 7AY, Bath, United Kingdom
E-mail: f.lindgren@bath.ac.uk
URL: http://people.bath.ac.uk/fl353/