spate: An R Package for Spatio-Temporal Modeling with a Stochastic Advection-Diffusion Process

Journal Article

Author(s):
Sigrist, Fabio; Künsch, Hans R.; Stahel, Werner A.

Publication date:
2015-02-13

Permanent link:
https://doi.org/10.3929/ethz-b-000099819

Originally published in:
Journal of Statistical Software 63(14), https://doi.org/10.18637/jss.v063.i14

Journal of Statistical Software

January 2015, Volume 63, Issue 14. http://www.jstatsoft.org/

spate: An R Package for Spatio-Temporal Modeling with a Stochastic Advection-Diffusion Process

Fabio Sigrist
ETH Zurich

Hans R. Künsch
ETH Zurich

Werner A. Stahel
ETH Zurich

Abstract

The R package **spate** implements methodology for modeling of large space-time data sets. A spatio-temporal Gaussian process is defined through a stochastic partial differential equation (SPDE) which is solved using spectral methods. In contrast to the traditional geostatistical way of relying on the covariance function, the spectral SPDE approach is computationally tractable and provides a realistic space-time parametrization.

This package aims at providing tools for simulating and modeling of spatio-temporal processes using an SPDE based approach. The package contains functions for obtaining parametrizations, such as propagator or innovation covariance matrices, of the spatio-temporal model. This allows for building customized hierarchical Bayesian models using the SPDE based model at the process stage. The functions of the package then provide computationally efficient algorithms needed for doing inference with the hierarchical model. Furthermore, an adaptive Markov chain Monte Carlo (MCMC) algorithm implemented in the package can be used as an algorithm for doing inference without any additional modeling. This function is flexible and allows for application specific customizing. The MCMC algorithm supports data that follow a Gaussian or a censored distribution with point mass at zero. Spatio-temporal covariates can be included in the model through a regression term.

Keywords: space-time model, large data sets, Gaussian process, physics based model, advection-diffusion equation, spectral methods, R.

1. Introduction

Increasingly larger spatio-temporal data arise in many fields and applications. For instance, data sets are obtained from remote sensing satellites or deterministic physical models such as numerical weather prediction (NWP) models. Hence, there is a growing need for methodology that can cope with such large data. See Cressie and Wikle (2011) for an introduction and an overview of spatio-temporal statistics.

Gaussian processes are often used for modeling data in space and time. A Gaussian process is defined by specifying a mean and a covariance function. However, directly working with a spatio-temporal covariance function is computationally unfeasible if data sets are large. Alternatively, a dynamic linear state space model (or a vector autoregression) can be used. The question is then how to choose the high dimensional matrices, such as propagator, innovation covariance and observation matrices, which define such a model. This is where stochastic partial differential equations (SPDE) come into play. Gaussian processes can be defined through SPDEs. The advection-diffusion SPDE is an elementary model in the spatio-temporal setting. When solving this SPDE in the spectral space, and discretizing in time and space, a linear Gaussian state space model is obtained, see Sigrist, Künsch, and Stahel (2015). The resulting linear state space model has realistic and interpretable choices for parametrizing its matrices and it is computationally tractable. Roughly speaking, the computational speed-up is due to the temporal Markov property and the fact that Fourier functions are eigenfunctions of the differential operator, from which follows that in the spectral space most of the relevant matrices are diagonal. The goals of this package is that a user can apply this model without worrying about details such as book keeping of Fourier coefficients. In statistics, using SPDEs for spatial or spatio-temporal modeling was first done by Whittle (1954); Heine (1955); Whittle (1962). Later works include Jones and Zhang (1997); Brown, Karesen, Roberts, and Tonellato (2000); Lindgren, Rue, and Lindstrom (2011). Similarly, Wikle, Milliff, Nychka, and Berliner (2001) propose a physics based model based on the shallow-water equations for modeling tropical ocean surface winds.

Several other packages exist that allow for analyzing and modeling of space-time data. The package **SpatioTemporal** (Lindstrom, Szpiro, Sampson, Bergen, and Oron 2013) provides utilities that estimate, predict and cross-validate the spatio-temporal model developed for the Multi-Ethnic Study of Atherosclerosis and Air Pollution (MESA Air). Spatio-temporal variability is modeled using spatially varying temporal basis functions. The package **gstat** (Pebesma 2004) allows for multivariate geostatistical modelling, prediction and simulation. It supports spatio-temporal variogram modelling and kriging. Both of the above packages rely on the covariance function which makes their application for large data unfeasible. **spacetime** (Pebesma 2012) is a package for storing, handling, and exploring different types of spatio-temporal data. Statistical modeling of spatio-temporal processes is not supported, though. **RandomFields** (Schlather, Malinowski, Menck, Oesting, and Strokorb 2015) provides functions for simulation and estimation of spatial and spatio-temporal random fields. Estimation of the hyper-parameters of Gaussian random fields can be done using maximum likelihood or least squares based on the variogram. **spTimer** (Bakar and Sahu 2015) allows for modeling of large data using separable spatio-temporal models. **spBayes** (Finley, Banerjee, and Carlin 2007; Finley, Banerjee, and Gelfand 2015) provides functions for modeling spatio-temporal data with Markov chain Monte Carlo (MCMC). Predictive process models (Finley, Banerjee, and Gelfand 2012) can be used for dealing with large data sets. Furthermore, the package **fields** (Nychka, Furrer, and Sain 2014) contains various tools for spatial statistics.

The package **spate** (Sigrist, Künsch, and Stahel 2013) has the following functionality. On the one hand, it provides tools for constructing customized models such as generalized linear mixed models (GLMM) or hierarchical Bayesian models (HBM, Wikle, Berliner, and Cressie 1998) using the SPDE based spatio-temporal Gaussian process at some stage, for instance, in the linear predictor. These tools include functions for obtaining spectral propagator and covariance matrices of the linear Gaussian state space model, fast calculation of the two-

dimensional real Fourier transform, reduced dimensional approximations, fast evaluation of the log-likelihood, and fast simulation from the full conditional of the Fourier coefficients using a spectral variant of the forward filtering backward sampling (FFBS) algorithm (Carter and Kohn 1994; Frühwirth-Schnatter 1994). On the other hand, the package also provides a function for doing Bayesian inference using an MCMC algorithm that is designed such that it needs as little fine tuning as possible. The MCMC algorithm can model data being normally distributed or censored data with point mass at zero following a skewed Tobit distribution. There is also a function for making probabilistic predictions. A user interested in modeling data not following one of the above two types of data distributions can modify the MCMC algorithm to allow for different distributions. In addition, functions for plotting and simulation of space-time processes are also provided.

1.1. Notation and model overview

In the following, we briefly introduce the notation used in this paper and present the SPDE based spatio-temporal model. For more details, we refer to the following sections. We assume that we observe a Gaussian process $w(t_i, \boldsymbol{s}_l)$, $i = 1, \ldots, T$, $l = 1, \ldots, N$, on a regular, rectangular grid of $n \times n = N$ spatial locations $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N$ in $[0, 1]^2$, n even, and at equidistant time points $t_1, \ldots, t_T$ with $t_i - t_{i-1} = \Delta$. These two assumptions can be easily relaxed, i.e., one can have irregular spatial locations and non-equidistant time points. The former can be achieved by adopting a data augmentation approach (implemented in `spate.mcmc`) or by using an incidence matrix (also implemented in `spate.mcmc`, see below) depending on the dimensionality of the observation process. The latter can be done by taking a time varying Δ_i. We assume that the observed process $w(t_i, \boldsymbol{s}_l)$ equals a parametric regression term plus an SPDE based Gaussian process $\xi(t_i, \boldsymbol{s}_l)$, modeling structured spatio-temporal variation, plus an unstructured term $\nu(t_i, \boldsymbol{s}_l)$ accounting for measurement errors and / or small scale variation.

In vectorized form, we write $\boldsymbol{w}(t_i) = (w(t_i, \boldsymbol{s}_1), \ldots, w(t_i, \boldsymbol{s}_N))^\top$ (where stacking is done first over the x-axis and then over the y-axis). $\boldsymbol{\xi}(t_i)$ and $\boldsymbol{\nu}(t_i)$ are defined analogously. The model we propose is then the following linear Gaussian state space model

$$\boldsymbol{w}(t_{i+1}) = \sum_{p=1}^{P} \beta_p \cdot \boldsymbol{x}_p(t_{i+1}) + \boldsymbol{\xi}(t_{i+1}) + \boldsymbol{\nu}(t_{i+1}), \quad \boldsymbol{\nu}(t_{i+1}) \sim N(\boldsymbol{0}, \tau^2 \boldsymbol{1}), \tag{1}$$

$$\boldsymbol{\xi}(t_{i+1}) = \boldsymbol{\Phi}\boldsymbol{\alpha}(t_{i+1}), \tag{2}$$

$$\boldsymbol{\alpha}(t_{i+1}) = \boldsymbol{G}\boldsymbol{\alpha}(t_i) + \widehat{\boldsymbol{\epsilon}}(t_{i+1}), \quad \widehat{\boldsymbol{\epsilon}}(t_{i+1}) \sim N(\boldsymbol{0}, \widehat{\boldsymbol{Q}}), \tag{3}$$

where $\boldsymbol{x}_p(t_{i+1}) = (x_p(t_{i+1}, \boldsymbol{s}_1), \ldots, x_p(t_{i+1}, \boldsymbol{s}_N))^\top$, $p = 1, \ldots, P$, are spatio-temporal covariates with coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_P)^\top \in \mathbb{R}^P$. For each t_i, the spatio-temporal process $\boldsymbol{\xi}(t_i) = \boldsymbol{\Phi}\boldsymbol{\alpha}(t_i)$ is the Fourier transform of Fourier coefficients $\boldsymbol{\alpha}(t_i)$ which evolve dynamically over time. The parametrizations of the propagator and innovation covariance matrices $\boldsymbol{G}$ and $\widehat{\boldsymbol{Q}}$ are defined through the spectral solution of the SPDE, see Section 2.3 for details.

If the observations are censored, to be more specific, if they follow a skewed Tobit distribution, the observed data is denoted by $y(t_i, \boldsymbol{s}_l)$. Finally, $\boldsymbol{\theta} = (\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2)^\top$ denotes the vector of all hyper-parameters used for parametrizing the spatio-temporal model.

2. Summary of methodological background

In the following, we briefly present the underlying model and methodology. For more details we refer to Sigrist *et al.* (2015). We start with a continuous space-time model that is defined through an SPDE. This is then solved in the spectral space and discretized in both space and time to obtain a vector autoregression, see Equations 2 and 3.

2.1. Space-time Gaussian process defined through an SPDE

A spatio-temporal Gaussian process $\xi(t, \boldsymbol{s})$ is defined as the solution of the stochastic advection-diffusion equation

$$\frac{\partial}{\partial t}\xi(t, \boldsymbol{s}) = -\boldsymbol{\mu}^\top \nabla \xi(t, \boldsymbol{s}) + \nabla \cdot \boldsymbol{\Sigma} \nabla \xi(t, \boldsymbol{s}) - \zeta \xi(t, \boldsymbol{s}) + \epsilon(t, \boldsymbol{s}), \tag{4}$$

where $t \geq 0$, $\boldsymbol{s} \in [0, 1]^2$ wrapped on a torus, $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)^\top$ is the gradient operator, and $\nabla \cdot \boldsymbol{F} = \frac{\partial F^x}{\partial x} + \frac{\partial F^y}{\partial y}$ is the divergence operator for $\boldsymbol{F} = (F^x, F^y)^\top$ being a vector field, $\boldsymbol{\mu} = (\mu_x, \mu_y)^\top$,

$$\boldsymbol{\Sigma}^{-1} = \frac{1}{\rho_1^2} \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\gamma \cdot \sin\alpha & \gamma \cdot \cos\alpha \end{pmatrix}^T \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\gamma \cdot \sin\alpha & \gamma \cdot \cos\alpha \end{pmatrix},$$

and where $\epsilon(t, \boldsymbol{s})$ is a Gaussian random field that is white in time and has a spatial Matérn covariance function with spectral density

$$\widehat{f}(\boldsymbol{k}) = \frac{\sigma^2}{(2\pi)^2} \left(\boldsymbol{k}^\top \boldsymbol{k} + \frac{1}{\rho_0^2}\right)^{-(v+1)}.$$

For the parameters, we have the following restrictions

$$\rho_0, \sigma, \rho_1, \gamma, \zeta, v \geq 0, \quad \mu_x, \mu_y \in [-0.5, 0.5], \quad \alpha \in [0, \pi/2].$$

2.2. Interpretation

Conceptually, the SPDE can just be seen as a tool for defining a space-time Gaussian process. The covariance function of this process is implicitly defined through the SPDE and the choice of parameters of the SPDE. The covariance function cannot be given in closed form, its Fourier transform can, though. Similarly as, e.g., a range parameter of an exponential covariance function in spatial statistics, the parameters of this Gaussian process can be interpreted.

Heuristically, the SPDE specifies what happens locally at each point in space during a small time step. The first term $\boldsymbol{\mu} \cdot \nabla \xi(t, \boldsymbol{s})$ models transport effects (called advection in weather applications), $\boldsymbol{\mu}$ being a drift or velocity vector. The second term, $\nabla \cdot \boldsymbol{\Sigma} \nabla \xi(t, \boldsymbol{s})$, is a diffusion term that can incorporate anisotropy. ρ_1 acts as a range parameter and controls the amount of diffusion. The parameters γ and α control the amount and the direction of anisotropy. With $\gamma = 1$, isotropic diffusion is obtained. Removing a certain amount of $\xi(t, \boldsymbol{s})$ at each time, $-\zeta \xi(t, \boldsymbol{s})$ accounts for damping and regulates the amount of temporal correlation. Finally, $\epsilon(t, \boldsymbol{s})$ is a source-sink or stochastic forcing term that can be interpreted as describing,

amongst others, convective phenomena in precipitation modeling applications. ρ_0 is a range parameter and σ^2 determines the marginal variance. Since in many applications the smoothness parameter v is not estimable from data, we take $v = 1$ by default, which corresponds to the Whittle covariance function.

If the terms $\boldsymbol{\mu} \cdot \nabla \xi(t, \boldsymbol{s})$ and $\nabla \cdot \boldsymbol{\Sigma} \nabla \xi(t, \boldsymbol{s})$ equal zero, the model reduces to a separable Gaussian process with a covariance function that is just the product of a temporal and a spatial one. Its discretized version is then a vector autoregression with a scalar propagator matrix. In many applications, however, separability is an assumptions that is too simple to be realistic.

2.3. Spectral solution

As is shown in Sigrist *et al.* (2015), inference can be done computationally efficiently when solving the SPDE in the spectral space. The latter means, roughly speaking, that the solution is represented as a linear combination of deterministic, real Fourier basis functions

$$\phi_j^{(c)}(\boldsymbol{s}) = \cos(\boldsymbol{k}_j^\top \boldsymbol{s}), \quad \phi_j^{(s)}(\boldsymbol{s}) = \sin(\boldsymbol{k}_j^\top \boldsymbol{s}),$$

with random coefficients $\alpha_j^c(t)$, $\alpha_j^s(t)$ that evolve dynamically over time according to a vector autoregression. See Cressie and Wikle (2011, Chapter 7) for an overview of basis function expansions in spatio-temporal statistics. Fourier functions have several advantages for solving the SPDE (4). Amongst others, Fourier functions are eigenfunctions of the spatial differential operator: differentiation in the physical space corresponds to multiplication in the spectral space. Furthermore, one can use the fast Fourier transform (FFT) (Cooley and Tukey 1965) for efficiently transforming from the physical to the spectral space, and vice versa.

While the solution of the SPDE is a continuous space-time model, in practice, one typically needs to discretize it in space and time. Solving the SPDE in the spectral space and discretizing it, we obtain the vector autoregression given in Equations 2 and 3. In Equation 2, the matrix $\boldsymbol{\Phi}$ is given by

$$\begin{aligned}
\boldsymbol{\Phi} &= \Big[\boldsymbol{\phi}(\boldsymbol{s}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_N)\Big]^\top, \\
\boldsymbol{\phi}(\boldsymbol{s}_l) &= \Big(\phi_1^{(c)}(\boldsymbol{s}_l), \ldots, \phi_4^{(c)}(\boldsymbol{s}_l), \phi_5^{(c)}(\boldsymbol{s}_l), \phi_5^{(s)}(\boldsymbol{s}_l), \ldots, \phi_{K/2+2}^{(c)}(\boldsymbol{s}_l), \phi_{K/2+2}^{(s)}(\boldsymbol{s}_l)\Big)^\top.
\end{aligned}$$

This matrix applies the discrete, real Fourier transformation to the coefficients

$$\boldsymbol{\alpha}(t) = \Big(\alpha_1^{(c)}(t), \ldots, \alpha_4^{(c)}(t), \alpha_5^{(c)}(t), \alpha_5^{(s)}(t), \ldots, \alpha_{K/2+2}^{(c)}(t), \alpha_{K/2+2}^{(s)}(t)\Big)^\top.$$

Note that the first four terms are cosine terms and, afterwards, there are cosine - sine pairs. This is a peculiarity of the real Fourier transform. It is due to the fact that for four wavenumbers $\boldsymbol{k}_j$, the sine terms equal zero on the grid. We use the real Fourier transform, instead of the complex one, in order to avoid complex numbers in the propagator matrix $\boldsymbol{G}$ and since data is usually real. Note that due to the use of Fourier functions, we assume spatial stationarity for both the solution ξ and the innovation term ϵ.

Equation 3 specifies how the random Fourier coefficients evolve dynamically over time. The propagator matrix $\boldsymbol{G}$ is a block diagonal matrix with 2×2 blocks, and the innovation covari-

ance matrix $\widehat{\boldsymbol{Q}}$ is a diagonal matrix. These two matrices are defined as follows:

$$\begin{aligned} \boldsymbol{G} &= e^{\Delta \boldsymbol{H}}, \\ [\boldsymbol{H}]_{1:4,1:4} &= \operatorname{diag}\left(-\boldsymbol{k}_j^\top \boldsymbol{\Sigma} \boldsymbol{k}_j - \zeta\right), \\ [\boldsymbol{H}]_{5:K,5:K} &= \operatorname{diag}\begin{pmatrix} -\boldsymbol{k}_j^\top \boldsymbol{\Sigma} \boldsymbol{k}_j - \zeta & -\boldsymbol{\mu} \boldsymbol{k}_j \\ \boldsymbol{\mu} \boldsymbol{k}_j & -\boldsymbol{k}_j^\top \boldsymbol{\Sigma} \boldsymbol{k}_j - \zeta \end{pmatrix}, \end{aligned} \tag{5}$$

and

$$\widehat{\boldsymbol{Q}} = \operatorname{diag}\left(\widehat{f}(\boldsymbol{k}_j) \frac{1 - e^{-2\Delta(\boldsymbol{k}_j^\top \boldsymbol{\Sigma} \boldsymbol{k}_j + \zeta)}}{2(\boldsymbol{k}_j^\top \boldsymbol{\Sigma} \boldsymbol{k}_j + \zeta)}\right). \tag{6}$$

The above result is given in vector format. For the sake of understanding, we can also write the solution as follows.

$$\begin{aligned} \xi(t, \boldsymbol{s}_l) =& \sum_{j=1}^{4} \alpha_j^{(c)}(t) \phi_j^{(c)}(\boldsymbol{s}_l) + \sum_{j=5}^{K/2+2} \alpha_j^{(c)}(t) \phi_j^{(c)}(\boldsymbol{s}_l) + \alpha_j^{(s)}(t) \phi_j^{(s)}(\boldsymbol{s}_l) \\ =& \boldsymbol{\phi}(\boldsymbol{s}_l)^\top \boldsymbol{\alpha}(t), \end{aligned} \tag{7}$$

where K denotes the number of Fourier terms, i.e., $K = N$. K however does not necessary need to equal N, see below for a discussion on dimension reduction.

The spatial wavenumbers $\boldsymbol{k}_j$ used in the real Fourier transform lie on the $n \times n$ grid $D_n = \{2\pi \cdot (i, j) : -(n/2 + 1) \leq i, j \leq n/2\} \subset 2\pi \cdot \mathbb{Z}^2$ with $n^2 = N$ and n even.

2.4. Non-Gaussian data, missing data, and non-grid data

Non-Gaussian data can be modeled, for instance, in the framework of generalized linear mixed models (GLMM) or hierarchical Bayesian models (HBM). This means that one assumes that the data follow a non-Gaussian distribution $\mathcal{F}$ conditionally on $w(t_i, \boldsymbol{s}_l)$ or conditionally on the linear predictor $\sum_{p=1}^{P} \beta_p \cdot \boldsymbol{x}_p(t_i, \boldsymbol{s}_l) + \xi(t_{i+1}, \boldsymbol{s}_l)$. HBMs can be fitted similarly as outlined in Rue and Held (2005, Chapter 4) for Gaussian Markov random fields (GMRF). Analogously as in the case of GMRFs, fast simulation and evaluation of the likelihood of the Gaussian field is crucial when modeling non Gaussian data using an HBM. Note, however, that fitting such models can be a non-trivial task and is subject to ongoing research.

Another approach, which avoids adding an additional stochastic level, is to assume that the data is a transformed version of $w(t_i, \boldsymbol{s}_l)$. For instance, if the observations follow a skewed Tobit model, then the we have the following observation relation

$$y(t_i, \boldsymbol{s}_l) = \max(0, w(t_i, \boldsymbol{s}_l))^\lambda, \tag{8}$$

where now $y(t_i, \boldsymbol{s}_l)$ denotes the observed values and $w(t_i, \boldsymbol{s}_l)$ is a latent Gaussian field. This data model is implemented in the package **spate**. Such a model is often used for modeling precipitation.

Furthermore, missing values, and the censored ones in (8), can be easily dealt with using a data augmentation approach. Roughly speaking, one adds an additional Gibbs step for simulating the missing values. See, e.g., Sigrist, Künsch, and Stahel (2012) for more details.

If the observations do not lie on a regular spatial grid, one can either include an incidence matrix $\boldsymbol{I}$ that relates the process on the grid to the observation locations (see Equation 9

below) or, depending on the number of observations, one can assign the data to a regular grid and treat the cells with no observations as missing data.

2.5. Computationally efficient inference

When doing inference, for both data models, the Gaussian one in (1) and the transformed Tobit model (8), the main difficulty consists in evaluating the likelihood $\ell(\boldsymbol{\theta}) = P[\boldsymbol{\theta}|\boldsymbol{w}]$, $\boldsymbol{\theta} = (\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2)^\top$, and in simulating from the full conditional of the coefficients $[\boldsymbol{\alpha}|\boldsymbol{w}, \boldsymbol{\theta}]$, where $\boldsymbol{w} = (\boldsymbol{w}(t_1)^\top, \ldots, \boldsymbol{w}(t_T)^\top)^\top$ and $\boldsymbol{\alpha} = (\boldsymbol{\alpha}(t_1)^\top, \ldots, \boldsymbol{\alpha}(t_T)^\top)^\top$ denote the full space-time fields. As shown in Sigrist *et al.* (2015), this can be done in $O(TN)$ time in the spectral space using the Kalman filter and a backward sampling step. The FFT can be used to transform between the physical and the spectral space. Since there are T fields of dimension N $(= n^2)$, the costs for this are $O(TN \log N)$. These T Fourier transforms can be run in parallel, though.

2.6. Dimension reduction

The total computational costs can be additionally alleviated by using a reduced dimensional Fourier basis with $K << N$ basis functions. This means that one includes only certain frequencies, e.g., low ones. The spectral filtering and sampling algorithms then require $O(KT)$ operations. For using the FFT, the frequencies being excluded are just set to zero.

Alternatively, when the observation process is irregular and low-dimensional in space, one can include an incidence matrix $\boldsymbol{I}$ that relates the process on the grid to the observation locations. Instead of (1), the observation equation, without a regression term, is then

$$\boldsymbol{w}(t_{i+1}) = \boldsymbol{I}\boldsymbol{\Phi}\boldsymbol{\alpha}(t_{i+1}) + \boldsymbol{\nu}(t_{i+1}), \quad \boldsymbol{\nu}(t_{i+1}) \sim N(\boldsymbol{0}, \tau^2 \boldsymbol{1}_K). \tag{9}$$

The FFT cannot be used anymore, and the total computational costs are $O(K^3T)$ due to the traditional FFBS.

3. Parametrization of the dynamic space-time model

3.1. Innovation spectrum and Matérn spectrum

The function `innov.spec` returns the spectrum of the integrated stochastic innovation field $\boldsymbol{\Phi}\widehat{\boldsymbol{\epsilon}}(t_{i+1})$. I.e., the function returns the diagonal entries of the covariance matrix $\widehat{\boldsymbol{Q}}$ of $\widehat{\boldsymbol{\epsilon}}(t_{i+1})$ as specified in (6). Similarly, the function `matern.spec` returns the spectrum of the Matérn covariance function. Note that the Matérn spectrum is renormalized, by dividing with the sum over all frequencies so that they sum to one. This guarantees that the parameter σ^2 is the marginal variance no matter how many wavenumbers are included, in case dimension reduction is done and some frequencies are set to zero.

The code below illustrates how these functions are used. First a vector of independent Gaussian random variables with variances according to the desired spectrum is simulated. For instance, $\widehat{\boldsymbol{\epsilon}} \sim N(0, \widehat{\boldsymbol{Q}})$. In the example, this is done for the Whittle and the integrated innovation spectrum specified in (6). Then its Fourier transform $\boldsymbol{\Phi}\widehat{\boldsymbol{\epsilon}}$ is calculated to obtain a sample from the spatial field with corresponding spectrum. See two sections below for more

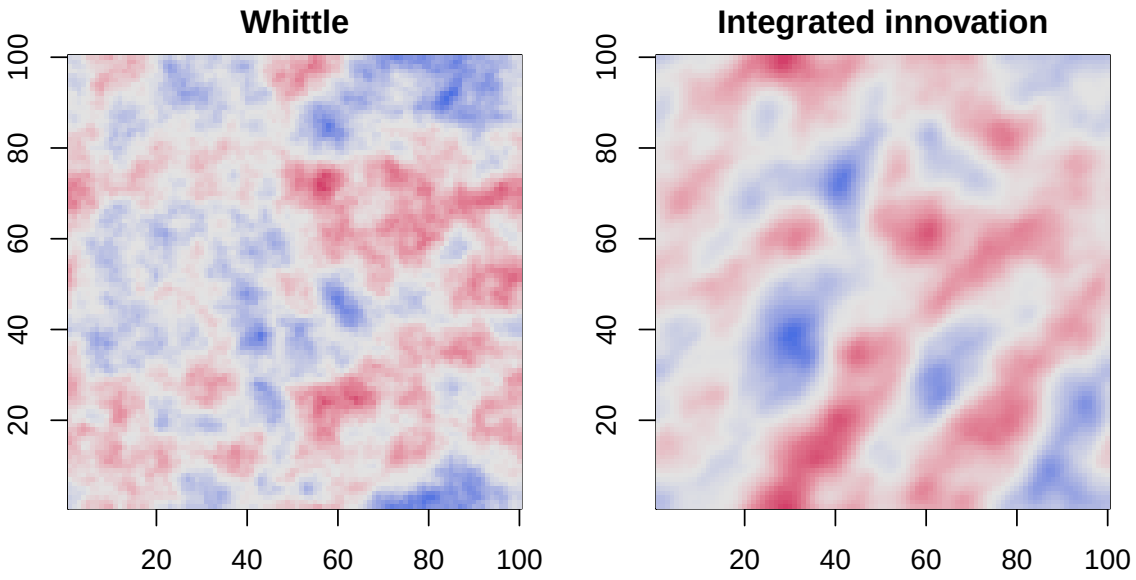

Figure 1: Samples from Gaussian processes with Whittle covariance function and the covariance function of the integrated stochastic innovation field $\mathbf{\Phi}\widehat{\boldsymbol{\epsilon}}(t_{i+1})$.

details on how to calculate the Fourier transform. Figure 1 shows sample fields from the Whittle process and from the stochastic innovation process. The color scale is generated using a palette from the package **colorspace** (Ihaka, Murrell, Hornik, Fisher, and Zeileis 2013). See also Zeileis, Hornik, and Murrell (2009).

```
R> n <- 100
R> set.seed(1)
R> wave <- spate.init(n = n, T = 1)[["wave"]]
R> matern.spec <- matern.spec(wave = wave, n = n, rho0 = 0.05, sigma2 = 1,
+    norm = TRUE)
R> matern.sim <- real.fft(sqrt(matern.spec) * rnorm(n * n), n = n,
+    inv = FALSE)
R> innov.spec <- innov.spec(wave = wave, n = n, rho0 = 0.05, sigma2 = 1,
+    zeta = 0.5, rho1 = 0.05, alpha = pi/4, gamma = 2, norm = TRUE)
R> innov.sim <- real.fft(sqrt(innov.spec) * rnorm(n * n), n = n,
+    inv = FALSE)
```

3.2. Propagator matrix

The function `get.propagator` returns the spectral propagator matrix $\boldsymbol{G}$ as defined in (5). The following code illustrates how `get.propagator` is used.

```
R> n <- 4
R> wave <- wave.numbers(n)
R> G <- get.propagator(wave = wave[["wave"]], indCos = wave[["indCos"]],
+    zeta = 0.5, rho1 = 0.1, gamma = 2, alpha = pi/4, muX = 0.2,
+    muY = -0.15)
```

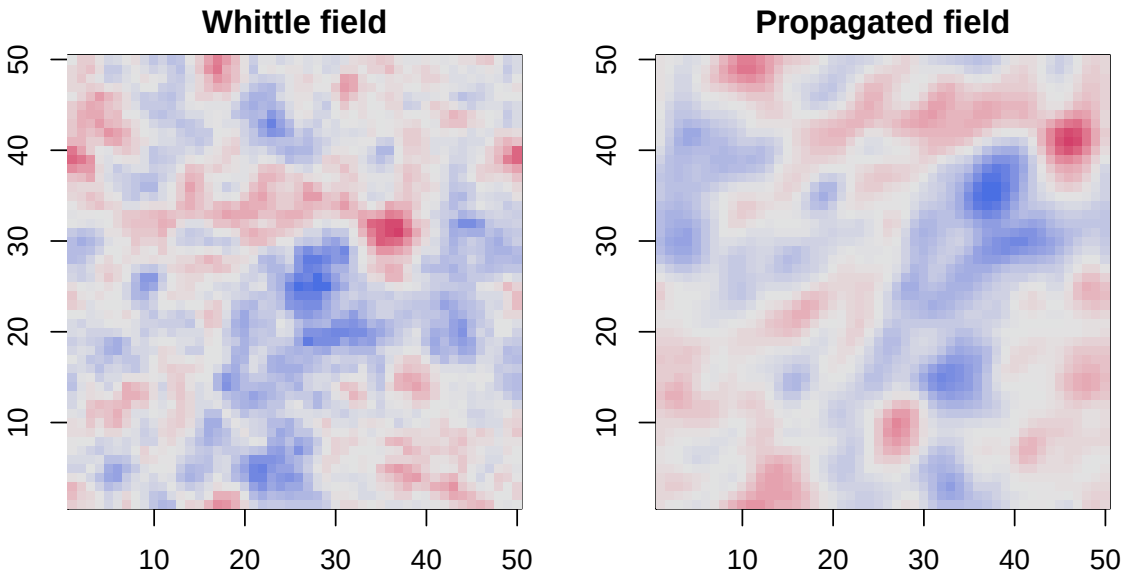

Figure 2: Illustration of spectral propagation: initial and propagated field.

Alternatively, the function `propagate.spectral` propagates a state $\boldsymbol{\alpha}(t)$ to obtain $\boldsymbol{G}\boldsymbol{\alpha}(t)$ in a computationally efficient way using the block-diagonal structure of $\boldsymbol{G}$. Note that this is a wrapper function of a C function. In general, it is preferable to use `propagate.spectral` instead of calculating a matrix multiplication with $\boldsymbol{G}$. The function `propagate.spectral` has as argument the propagator matrix $\boldsymbol{G}$ in vectorized from as obtained from the function `get.propagator.vec`. Figure 2 and the corresponding code illustrates the use of these two functions. First, we define an initial state $\boldsymbol{\alpha}(t)$, which is a sample from the process with the Whittle covariance function in this example. Then $\boldsymbol{\alpha}(t)$ is propagated forward to obtain $\boldsymbol{G}\boldsymbol{\alpha}(t)$. The code shows that actually calculating $\boldsymbol{G}\boldsymbol{\alpha}(t)$ and applying `propagate.spectral` are equivalent (see last line of code).

```
R> n <- 50
R> wave <- wave.numbers(n)
R> spec <- matern.spec(wave = wave[["wave"]], n = n, rho0 = 0.05,
+    sigma2 = 1, norm = TRUE)
R> alphat <- sqrt(spec) * rnorm(n * n)
R> G <- get.propagator(wave = wave[["wave"]], indCos = wave[["indCos"]],
+    zeta = 0.1, rho1 = 0.02, gamma = 2, alpha = pi/4, muX = 0.2,
+    muY = 0.2, dt = 1, ns = 4)
R> alphat1a <- as.vector(G %*% alphat)
R> Gvec <- get.propagator.vec(wave = wave[["wave"]],
+    indCos = wave[["indCos"]], zeta = 0.1, rho1 = 0.02, gamma = 2,
+    alpha = pi/4, muX = 0.2, muY = 0.2, dt = 1, ns = 4)
R> alphat1b <- propagate.spectral(alphat, n = n, Gvec = Gvec)
R> sum(abs(alphat1a - alphat1b))
```

```
[1] 0
```

3.3. Two-dimensional real Fourier transform

The function `real.fft` calculates the fast two-dimensional real Fourier transform. This is a wrapper function of a C function which uses the complex FFT function from the `fftw3` library. Furthermore, the function `real.fft.TS` calculates the two-dimensional real Fourier transform of a space-time field for all time points at once. To be more specific, for each time point, the corresponding spatial field is transformed. In contrast to using T times the function `real.FFT`, R needs to communicate with C only once which saves considerable computational time, depending on the data size. For an example of the use of `real.fft`, see two sections above.

The function `wave.number` returns the wavenumbers used in the real Fourier transform. In contrast to the complex Fourier transform, which uses n^2 different wavenumbers $\boldsymbol{k}_j$ on a square grid, the real Fourier transform uses $n^2/2 + 2$ different wavenumbers. As mentioned earlier, four of them have only a cosine term, and the remaining $n^2/2 - 2$ wavenumbers each have a sine and cosine term. For technical details on the real Fourier transform, we refer to Dudgeon and Mersereau (1984), Borgman, Taheri, and Hagan (1984), Royle and Wikle (2005), and Paciorek (2007).

The function `get.real.dft.mat` returns the matrix $\boldsymbol{\Phi}$ (see (2)) which applies the two-dimensional real Fourier transform. Note that, in general, it is a lot faster to use `real.fft` rather than actually multiplying with $\boldsymbol{\Phi}$. The following code shows how $\boldsymbol{\Phi}$ can be constructed using `get.real.dft.mat`.

```
R> n <- 20
R> wave <- wave.numbers(n = n)
R> Phi <- get.real.dft.mat(wave = wave[["wave"]], indCos = wave[["indCos"]],
+    n = n)
```

4. Simulation and plotting

The function `spate.sim` allows for simulating from the SPDE based spatio-temporal Gaussian process model defined through (2) and (3). The function returns a `"spateSim"` object containing the sample $\boldsymbol{\xi}$, the coefficients $\boldsymbol{\alpha}$, as well as the observed $\boldsymbol{w}$ obtained by adding a nugget effect to $\boldsymbol{\xi}$. The argument `par` is a vector of parameters $\boldsymbol{\theta}$ in the following order $\boldsymbol{\theta} = (\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2)^\top$. An initial state, or starting value, $\boldsymbol{\xi}(t_1)$ for the dynamic model can be given through the argument `StartVal`. The starting field needs to be a stacked vector of lengths n^2 (number of spatial points). Use `as.vector()` to convert a spatial matrix to a vector. `"spateSim"` objects can be plotted with the function `plot.spateSim`. The following code illustrates the use of these functions, with the plot shown in Figure 3:

```
R> StartVal <- rep(0, 100^2)
R> StartVal[75 * 100 + 75] <- 1000
R> par <- c(rho0 = 0.05, sigma2 = 0.7^2, zeta = -log(0.99), rho1 = 0.06,
+    gamma = 3, alpha = pi/4, muX = -0.1, muY = -0.1, tau2 = 0.00001)
R> spateSim <- spate.sim(par = par, n = 100, T = 5, StartVal = StartVal,
+    seed = 1)
R> plot(spateSim, mfrow = c(1, 5), mar = c(2, 2, 2, 2), indScale = TRUE,
+    cex.axis = 1.5, cex.main = 2)
```

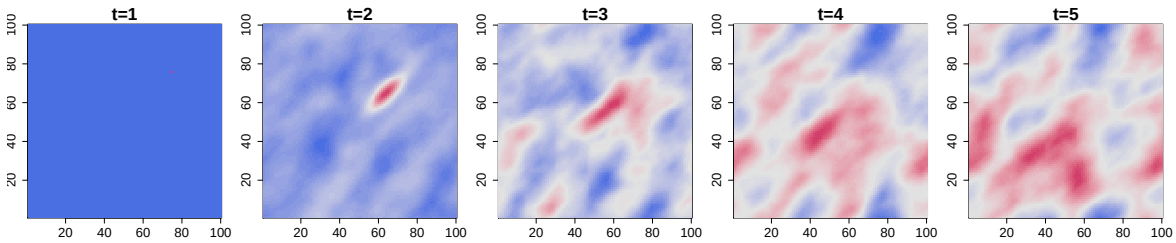

Figure 3: Simulated spatio-temporal Gaussian process as defined in (2) and (3).

Note that `indScale = TRUE` specifies that each field has its individual scale on the z-axis rather than having one common scale for all six images. Figure 3 shows one example of a simulated space-time process. The deterministic initial state in the left plot gets propagated over time and at each time step a spatially structured innovation noise is added, see Equations 2 and 3. The drift points from north-east to south-west and there is anisotropy in the same direction.

5. Log-likelihood and sampling from the full conditional

The function `ffbs.spectral` implements the computationally efficient Kalman filter and backward sampling algorithms in the spectral space for the model specified in (1), (2), and (3). The logical arguments `lglk` or `BwSp` control whether evaluation of the log-likelihood, sampling from the full conditional of the coefficients $\boldsymbol{\alpha}$, or both are done. This is a wrapper function and the actual calculation is done in C. Note that either the actual observed data $\boldsymbol{w}$ can be given or the Fourier transform $\widehat{\boldsymbol{w}}$ (`wFT`). The latter is useful if, for instance, the log-likelihood needs to be evaluated several times given the same $\boldsymbol{w}$. The Fourier transform is then calculated only once, instead of each time the function is called. `loglike` and `sample.four.coef` are wrapper functions that call `ffbs.spectral`.

5.1. Example of use of `sample.four.coef`

The following code illustrates the use of the function `sample.four.coef`. First, we simulate data $\boldsymbol{w}$, and then we sample from the full conditional of the coefficients $[\boldsymbol{\alpha}|\cdot]$ to obtain samples from the posterior of the latent process. For simplicity, the parameters $\boldsymbol{\theta}$ are fixed at their true values. In Figure 4, the results are shown. In the top plot, the simulated data is displayed and in the bottom plots the mean of full conditional of the process $\boldsymbol{\xi} = \boldsymbol{\Phi}\boldsymbol{\alpha}$. The latter is obtained by drawing 50 samples from the full conditional $[\boldsymbol{\alpha}|\cdot]$, calculating their mean, and applying the Fourier transform.

```
R> n <- 50
R> T <- 4
R> par <- c(rho0 = 0.1, sigma2 = 0.2, zeta = 0.5, rho1 = 0.1, gamma = 2,
+    alpha = pi/4, muX = 0.2, muY = -0.2, tau2 = 0.01)
R> spateSim <- spate.sim(par = par, n = n, T = T, seed = 4)
R> w <- spateSim$w
R> Nmc <- 50
```

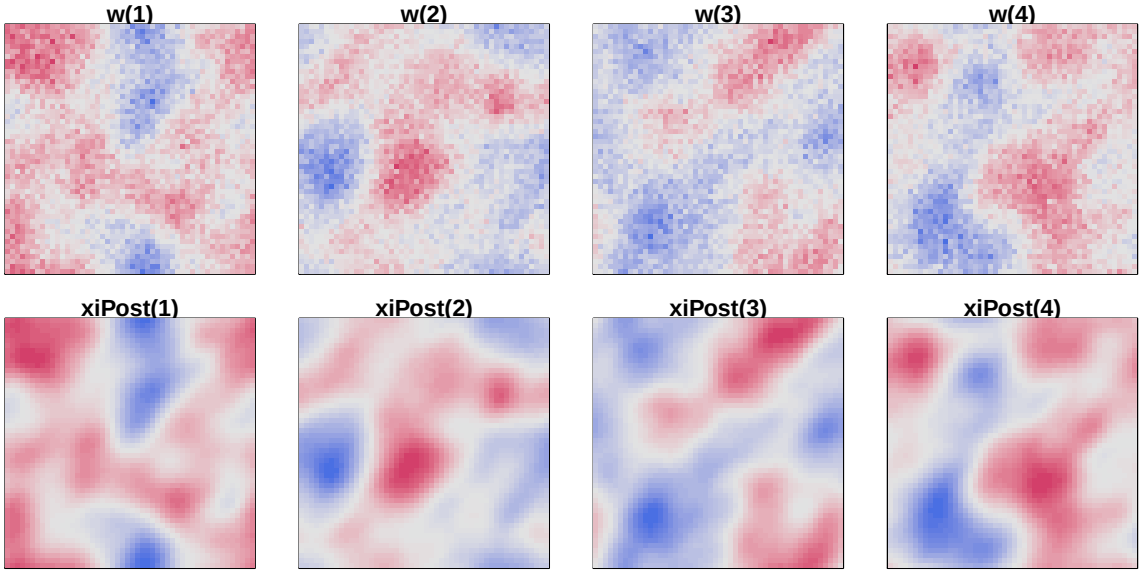

Figure 4: Sampling from the full conditional of the coefficients: comparison of observed data (top plots) and mean of full conditional of $\boldsymbol{\xi}$ (bottom plots).

```
R> alphaS <- array(0, c(T, n * n, Nmc))
R> wFT <- real.fft.TS(w, n = n, T = T)
R> for(i in 1:Nmc) { alphaS[, , i] <- sample.four.coef(wFT = wFT,
+    par = par, n = n, T = T, NF = n * n) }
R> alphaMean <- apply(alphaS, c(1, 2), mean)
R> xiMean <- real.fft.TS(alphaMean, n = n, T = T, inv = FALSE)
```

5.2. Example of use of `loglike`

The following code provides an example of the use of `loglike`. We use the same simulated data as in the previous example and evaluate the log-likelihood at the true parameter values. The code also demonstrates that the function `loglike` does the same thing whether one uses the original data $\boldsymbol{w}$ or their Fourier transform $\widehat{\boldsymbol{w}}$ =`wFT`. For an example on how to do maximum likelihood estimation, see the next section.

```
R> loglike(par = par, w = w, n = n, T = T)

[1] 7861.001

R> loglike(par = par, wFT = wFT, n = n, T = T)

[1] 7861.001
```

5.3. Maximum likelihood estimation

With the function `loglike`, one can do maximum likelihood estimation. The following code shows an example of how this can be done using a general purpose optimizer, e.g., implemented

in the R function `optim`. First, simulated data is generated. Then `optim` is used to minimize the negative log-likelihood. In the code when calling `loglike`, we set `negative = TRUE` as an argument for `loglike` so that it returns the negative log-likelihood. Further, with `logScale = TRUE` we specify that certain parameters are on the logarithmic scale to ensure positivity constraints. `logInd` is a vector of natural numbers indicating which parameters in `par` are on the logarithmic scale. Additional constraints, e.g., on the angle of the diffusion anisotropy α or on the drift terms μ_x and μ_y are set by using the L-BFGS-B algorithm called by setting `method = "L-BFGS-B"` in the `optim` function. The results show the estimated parameters, transformed back to the original scale, as well as 95% confidence intervals. Evaluating the likelihood for this 8000 dimensional Gaussian process ($20 \times 20 \times 20$) takes about 0.008 seconds on a desktop PC (AMD Athlon 64 X2 dual core processor 5600+). This is achieved without applying any dimension reduction. The entire inference takes less than 12 seconds.

```
R> n <- 20
R> T <- 20
R> par <- c(rho0 = 0.1, sigma2 = 0.2, zeta = 0.5, rho1 = 0.1,
+    gamma = 2, alpha = pi/4, muX = 0.2, muY = -0.2, tau2 = 0.01)
R> spateSim <- spate.sim(par = par, n = n, T = T, seed = 4)
R> w <- spateSim$w
R> parI <- c(rho0 = 0.2, sigma2 = 0.1, zeta = 0.25, rho1 = 0.01,
+    gamma = 1, alpha = 0.3, muX = 0, muY = 0, tau2 = 0.005)
R> logInd <- c(1, 2, 3, 4, 5, 9)
R> parI[logInd] <- log(parI[logInd])
R> wFT <- real.fft.TS(w, n = n, T = T)
R> spateMLE <- optim(par = parI, loglike, method = "L-BFGS-B",
+    control = list(trace = TRUE, maxit = 1000), wFT = wFT, negative = TRUE,
+    logScale = TRUE, logInd = c(1, 2, 3, 4, 5, 9), hessian = TRUE, n = n,
+    T = T, lower = c(-10, -10, -10, -10, -10, 0, -0.5, -0.5, -10),
+    upper = c(10, 10, 10, 10, 10, pi/2, 0.5, 0.5, 10))

iter    0 value -4612.774109
iter   10 value -4968.050499
iter   20 value -5011.362034
iter   30 value -5045.032630
iter   40 value -5045.150319
final  value -5045.150588
converged

R> mle <- spateMLE$par
R> mle[logInd] <- exp(mle[logInd])
R> sd <- sqrt(diag(solve(spateMLE$hessian)))
R> MleConfInt <- data.frame(array(0, c(4, 9)))
R> colnames(MleConfInt) <- names(par)
R> rownames(MleConfInt) <- c("True", "Estimate", "Lower", "Upper")
R> MleConfInt[1,] <- par
R> MleConfInt[2,] <- mle
R> MleConfInt[3,] <- spateMLE$par - 2 * sd
```

```
R> MleConfInt[4,] <- spateMLE$par + 2 * sd
R> MleConfInt[c(3, 4), logInd] <- exp(MleConfInt[c(3, 4), logInd])
R> round(MleConfInt, digits = 3)

     rho0 sigma2 zeta rho1 gamma alpha muX  muY  tau2
True  0.10  0.20 0.50 0.10 2.00  0.79 0.20 -0.20 0.01
Est.  0.09  0.17 0.36 0.11 2.21  0.85 0.21 -0.18 0.01
Lower 0.08  0.14 0.18 0.09 1.85  0.75 0.18 -0.21 0.01
Uppe  0.11  0.20 0.71 0.13 2.64  0.94 0.25 -0.14 0.01
```

5.4. Bayesian inference using MCMC

Using `sample.four.coef` and `loglike` an MCMC algorithm for drawing from the joint posterior of the latent process $\boldsymbol{\alpha}$, or equivalently $\boldsymbol{\xi}$, and the hyper-parameters $\boldsymbol{\theta}$ can be constructed.

One approach is to sample iteratively from $[\boldsymbol{\theta}|\cdot]$ using a Metropolis-Hastings step and from $[\boldsymbol{\alpha}|\cdot]$ with a Gibbs step. In many situations, $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ can be strongly dependent a posteriory. Consequently, if one samples successively from $[\boldsymbol{\theta}|\cdot]$ and $[\boldsymbol{\alpha}|\cdot]$, one can run into slow mixing properties. The reason is that in each step $[\boldsymbol{\theta}|\cdot]$ is constrained by the last sample of the latent process, and vice versa. To circumvent this problem, one can sample jointly from $[\boldsymbol{\theta}, \boldsymbol{\alpha}|\cdot]$. A joint proposal $(\boldsymbol{\theta}^*, \boldsymbol{\alpha}^*)$ is obtained by sampling $\boldsymbol{\theta}^*$ from a Gaussian distribution with the mean equaling the last value and an appropriately chosen covariance matrix and then sampling $\boldsymbol{\alpha}^*$ from $[\boldsymbol{\alpha}|\boldsymbol{\theta}^*, \cdot]$. The second step can be done using `sample.four.coef`. It can be shown that the acceptance probability then equals

$$\min\left(1, \frac{P[\boldsymbol{\theta}^*|\boldsymbol{w}]}{P[\boldsymbol{\theta}^{(i)}|\boldsymbol{w}]}\right), \tag{10}$$

where the likelihood $P[\boldsymbol{\theta}|\boldsymbol{w}]$ denotes the value of the density of $\boldsymbol{\theta}$ given $\boldsymbol{w}$ evaluated at $\boldsymbol{\theta}$, and where $\boldsymbol{\theta}^*$ and $\boldsymbol{\theta}^{(i)}$ denote the proposal and the last value of $\boldsymbol{\theta}$, respectively. Since this acceptance ratio does not depend on $\boldsymbol{\alpha}$, the parameters $\boldsymbol{\theta}$ can move faster in their parameter space. Note that $P[\boldsymbol{\theta}|\boldsymbol{w}]$ can be calculated using the function `loglike`.

Skewed Tobit model and missing data

For the transformed Tobit model (8), inference is done analogously. One just adds a Metropolis-Hastings step for the transformation parameter λ and a Gibbs step for the censored values $y(t, \boldsymbol{s}_l) = 0$. The latter consists in simulating from a censored normal distribution with mean $\xi^{(i)}(t, \boldsymbol{s}_l)$ and variance τ^2. See Sigrist *et al.* (2012) for more details.

As said, missing values can be dealt with by using a data augmentation approach. This means that one adds a Gibbs step consisting in simulating from a normal distribution with mean $\xi^{(i)}(t, \boldsymbol{s}_l)$ and variance $(\tau^2)^{(i)}$ for those points where $w(t, \boldsymbol{s}_l)$, or $y(t, \boldsymbol{s}_l)$, are missing.

6. An MCMC algorithm

It is well known that the performance of MCMC algorithms can be very dependent on the data, and that application specific tuning is often needed. Having this in mind, the function

`spate.mcmc` implements an MCMC algorithm that needs as little additional fine tuning as possible. It can deal with both Gaussian and skewed Tobit likelihoods through the argument `DataModel`. Sampling is done as outlined in the previous section. I.e., the coefficients $\boldsymbol{\alpha}$ and the hyper-parameters $\boldsymbol{\theta}$ are sampled together to obtain faster mixing. Further, an adaptive algorithm (Roberts and Rosenthal 2009) is used. This means that the proposal covariances `RWCov` for the Metropolis-Hastings step of $\boldsymbol{\theta}$ are successively estimated such that an optimal acceptance rate is obtained.

The function `spate.mcmc` returns an object of the class `"spateMCMC"` with, among others, samples from the posterior of the hyper-parameters stored in the matrix `Post`, the estimated proposal covariance matrix `RWCov`, and samples from the posterior of the latent process $\boldsymbol{\xi}$ in `xiPost` if `saveProcess = TRUE` was chosen. There are `plot` and `print` functions for `"spateMCMC"` objects.

6.1. Arguments of `spate.mcmc`

- The observed data is specified in the $N \times T$ matrix `y` with columns and rows corresponding to time and space (observations on a grid stacked into a vector).
- Observations in `y` can either be on a square grid or not. By default, at each time point, the observations are assumed to lie on a square grid with each axis scaled so that it has unit length. If not, the coordinates of each observation point can be specified in the $T \times N$ matrix `coord`, where $N (< n^2)$ is the number of observation stations. According to these coordinates, each observation location is then mapped to a grid cell. Alternatively to the use of `coord`, one can directly specify in the vector `Sind` the indices of the grid cells where observations are made. `Sind` needs to be of length N.
- In case `y` is not a full space time field on a rectangular grid, `n` specifies the number of points per axis of the square into which the observation points are mapped. In total, the process is modeled on a grid of size $n \times n$. `n` needs to be an even number. If the process is observed on a rectangular grid, we have by default $n \times n = N$. Otherwise, the dimension of the observed process N can be smaller than the dimension of the latent process $n \times n$.
- Spatio-temporal covariates can be specified in `x`. This is an array of dimensions $p \times T \times N$, where p denotes the number of covariates, T the number of time points, and N the number of spatial points.
- If covariates $\boldsymbol{x}$ are given, the algorithm can either sample the coefficients $\boldsymbol{\beta}$ in an additional Gibbs step from the Gaussian full conditional of the coefficients $[\boldsymbol{\beta}|\cdot]$ (`FixEffMetrop = FALSE`) or sample $\boldsymbol{\beta}$ together with $\boldsymbol{\theta}$ in the Metropolis-Hastings step (`FixEffMetrop = TRUE`). The latter is preferable since the random effects $\boldsymbol{\xi}$ and the fixed effects $\boldsymbol{x\beta}$ can be strongly dependent, which can result in very slow mixing if $\boldsymbol{\beta}$ and $\boldsymbol{\xi}$ are sampled iteratively and not jointly.
- Starting values for parameters are specified in the vector `SV`. The parameters are in the following order: $\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2$.
- Starting values for the regression coefficients can be given in the vector `betaSV`.

- The number of samples to be drawn from the Markov chain is specified in `Nmc` and the length of the burn-in in `BurnIn`.
- The covariance matrix of the proposal distribution used in the random walk Metropolis-Hastings step for the hyper-parameters can be specified in `RWCov`. If `adaptive = TRUE` is selected, this matrix is adaptively estimated and `RWCov` is only used in the initial stage. If `adaptive = TRUE`, `NCovEst` specifies the minimal number of samples to be used for estimating the proposal matrix and `BurnInCovEst` the length of the burn-in period after which the proposal matrix is estimated.
- Priors for the hyper-parameters can be specified in the functions `P.rho0`, `P.sigma2`, `P.zeta`, `P.rho1`, `P.gamma`, `P.alpha`, `P.mux`, `P.muy`, and `P.tau2`. By default, the package implements reasonable prior distributions for these parameters.
- `DataModel` specifies the data model. `"Normal"` or `"SkewTobit"` are available options.
- If the option `trace = TRUE` is selected, the MCMC algorithm prints running status messages such as acceptance rates of the hyper-parameters and estimated remaining computing time. Additionally, if choosing `plotTrace = TRUE`, running trace plots of the Markov chains are generated. Further, using `SaveToFile = TRUE`, the `"spateMCMC"` object can be successively saved in a directory specified through `path` and `file`.
- Dimension reduction can be applied by setting `DimRed = TRUE` and by specifying in `NFour` the number of Fourier functions to be used.
- If the Boolean argument `IncidenceMat` equals `TRUE`, an incidence matrix $\boldsymbol{I}$ is constructed and the model in (9) is used. In that case, dimension reduction needs to be done since one cannot use the fast spectral algorithms in combination with the FFT anymore.
- Padding can be applied by choosing `Padding = TRUE`.
- The vector of integers `indEst` specifies which parameters should be estimated and which not. By default this equals `c(1, ..., 9)`. If, for instance, one wants to fit a separable model, one can choose `indEst = c(1, 2, 3, 9)` in combination with `SV = c(0.2, 0.1, 0.25, 0, 0, 0, 0, 0, 0.001)`. The latter sets the initial values of the diffusion and drift term to zero. Since they are not sampled, they remain at zero.

`spate.mcmc` has additional functionality, see the help file of `spate.mcmc` for more details.

6.2. Additional fine tuning

In case the MCMC algorithm still needs some fine tuning, the following arguments can be varied:

- The initial covariance matrix `RWCov`.
- The burn-in length `BurnInCovEst` before starting with the adaptive estimation of `RWCov`.
- The minimal number of MCMC samples `NCovEst` required after the burn-in for estimating `RWCov`.

Due to the adaptive nature of the algorithm, the initial choice of `RWCov` is less important. However, if `RWCov` is overly large, the algorithm can have very small acceptance rates with the chain barely moving at all. On the other hand, if `RWCov` overly small, acceptance rates might be high, but the chain does not cover the parameter space.

If choosing adequate an `RWCov` turns out difficult, we propose the following strategy. For each hyper-parameter θ_i in $\boldsymbol{\theta}$, one searches for an appropriate variance σ_i^2 when fixing all other parameters. This can be done by specifying through the argument `indEst` which parameters should be estimated and which not. For instance, if `indEst = 1`, only for the first parameter ρ_0 a Markov chain is run, and the others are fixed. Using this, an appropriate σ_i^2 can be found as follows. For instance, one starts with a very low σ_i^2, and then increases it subsequently until the acceptance rate for θ_i, when fixing all other parameters, is at a reasonable level, say, around 0.4. After doing this for each parameter θ_i, `RWCov` $= \text{diag}(\sigma_i^2)$ can be used as initial covariance matrix. Note that the goal is not to find an optimal proposal covariance matrix but rather just to get a rough idea on the appropriate order of magnitude so that the algorithm is not "degenerate" from the beginning.

6.3. An example of the use of `spate.mcmc`

The following code illustrates the use of `spate.mcmc` on a simulated data set. The MCMC algorithm is run for 10000 samples with a burn-in of 2000. The burn-in for the adaptive covariance estimation is 500 and the minimal number of samples required for estimating the proposal covariance matrix of the Metropolis-Hasting step is also 500. This means that after 1000 samples, the proposal covariance matrix is first estimated. Subsequently, it is estimated every 500 samples based on the past excluding the first 500 samples from the Markov chain. Figure 5 shows trace plots of the MCMC algorithm. The vertical lines represent the burn-in period, and the horizontal lines are the true values of the parameters. Figure 5 shows how the mixing of the Markov chain improves with increasing time. Note that the number of samples, 10000, is used for illustration. In practice, more samples are needed.

```
R> par <- c(rho0 = 0.1, sigma2 = 0.2, zeta = 0.5, rho1 = 0.1,
+    gamma = 2, alpha = pi/4, muX = 0.2, muY = -0.2, tau2 = 0.01)
R> spateSim <- spate.sim(par = par, n = 20, T = 20, seed = 4)
R> w <- spateSim$w
R> spateMCMC <- spate.mcmc(y = w, x = NULL, SV = c(rho0 = 0.2,
+      sigma2 = 0.1, zeta = 0.25, rho1 = 0.2, gamma = 1, alpha = 0.3,
+      muX = 0, muY = 0, tau2 = 0.005),
+    RWCov = diag(c(5, 5, 50, 5, 5, 1, 0.2, 0.2, 0.2)/1000),
+    Nmc = 10000, BurnIn = 2000, seed = 4, NCovEst = 500,
+    BurnInCovEst = 500, trace = FALSE, Padding = FALSE)
R> spateMCMC

Posterior of parameters:
             Median        2.5 %      97.5 %
rho_0    0.09151798  0.075791669  0.10909257
sigma^2  0.17324097  0.142144435  0.22142933
zeta     0.36457843  0.112205001  0.64655163
rho_1    0.10796025  0.091442696  0.13150375
```

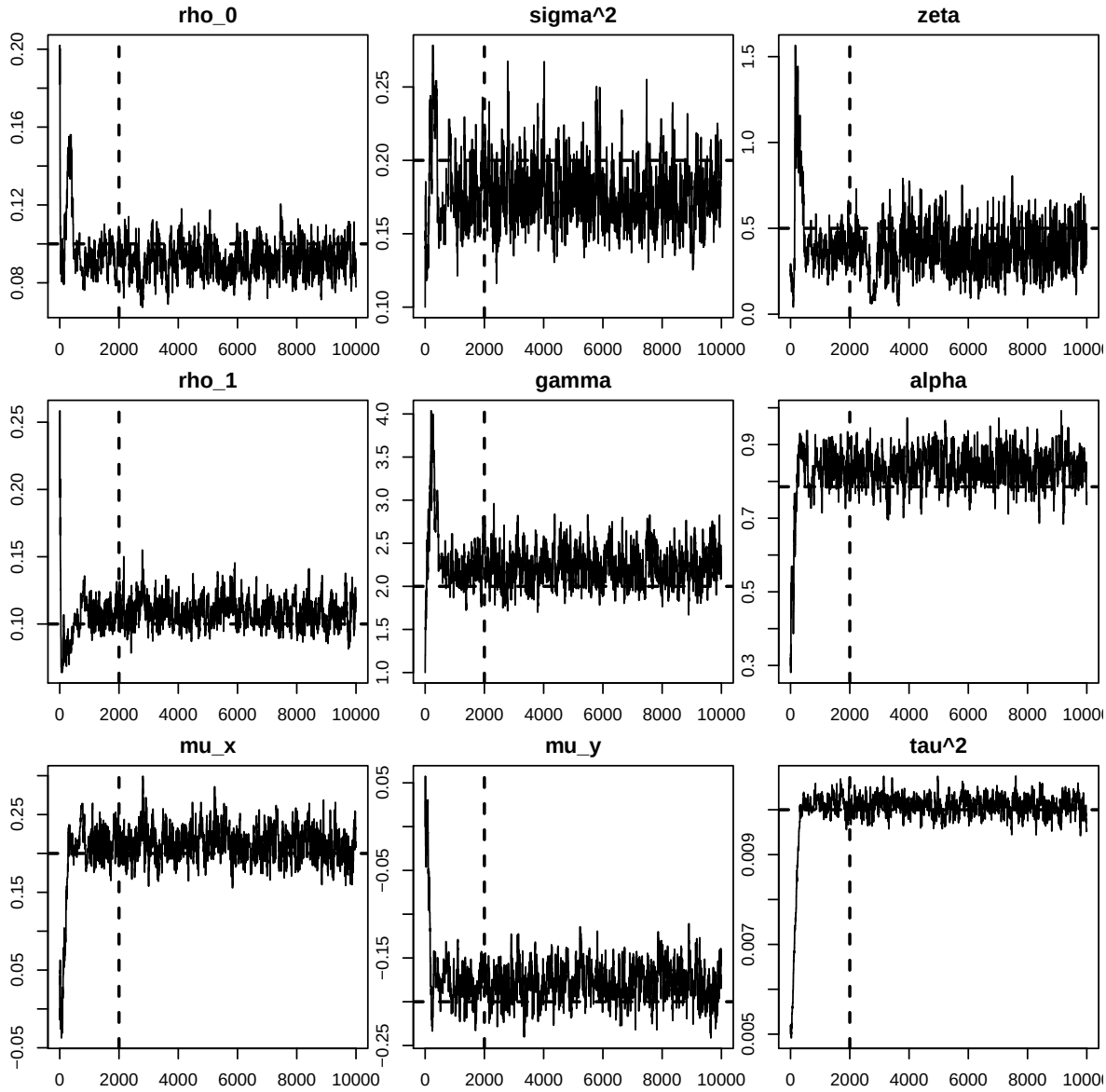

Figure 5: Trace plots from the MCMC algorithm. The vertical line shows the burn-in, and the horizontal lines are the true values of the parameters.

```
gamma     2.21058762  1.856203179  2.64283184
alpha     0.84013487  0.747082459  0.92371518
mu_x      0.21182190  0.175596745  0.25214613
mu_y     -0.17956504 -0.217171052 -0.13986582
tau^2     0.01009896  0.009687484  0.01047601

Results based on 8000 MCMC samples after a
  burn-in of 2000 samples
```

```
R> plot(spateMCMC, true = par, medianHist = FALSE, ask = FALSE)
```

The following code illustrates the use of `spate.mcmc` when an incidence matrix approach (see Equation 9) is used in combination with dimension reduction. This is the real data application used in Sigrist *et al.* (2015) where, roughly speaking, the goal is to model a spatio-temporal precipitation field. We are not showing any results here, but we only illustrate how the function `spate.mcmc` is called. For more details, we refer to Sigrist *et al.* (2015). A skewed Tobit model is used as data model. The observed data is not available on the full 100×100 grid but only at 32 observation locations. Observations are made at 720 time points. In the code below, `y` is a 720×32 matrix, and `covTS` is a $2 \times 720 \times 32$ array containing two covariates. `Sind` is a vector of length 32 indicating the grid cells in which the observation stations lie. `DataModel = "SkewTobit"` specifies that a skewed Tobit likelihood is used. `DimRed = TRUE` and `NFour=29` indicate that a reduced dimensional model consisting of 29 Fourier functions is used. By setting `IncidenceMat = TRUE`, we specify that an incidence matrix is used. Finally, `FixEffMetrop = TRUE` indicates that the coefficients of the covariates are sampled together with the hyper-parameters of the spatio-temporal model in order to avoid slow mixing due to correlations between fixed and random effects.

```
R> spateMCMC <- spate.mcmc(y = y, x = covTS, DataModel = "SkewTobit",
+    Sind = Sind, n = 100, DimRed = TRUE, NFour = 29, IncidenceMat = TRUE,
+    FixEffMetrop = TRUE, Nmc = 105000, BurnIn = 5000, Padding = TRUE,
+    NCovEst = 500, BurnInCovEst = 1000)
```

6.4. Making predictions with `spate.predict`

The function `spate.predict` allows for making probabilistic prediction. It takes a `"spateMCMC"` object containing samples from the posterior of the hyper-parameters as argument. The function then internally calls `spate.mcmc` where now the Metropolis-Hastings step for the hyper-parameters is skipped since these are given, and simulation is only done for the latent coefficients $\boldsymbol{\alpha}$. In doing so, samples from the predictive distribution are generated. The time points where predictions are to be made are specified through the argument `tPred`. Spatial points are either specified through `sPred` (grid points) or `xPred` and `yPred` (coordinates). If no spatial points are selected, predictions will be made for the entire fields at the time points chosen in `tPred`. In the example, we make predictions at time points $t = 21, 22, 23$ for the entire spatial fields using `Nsim = 100` samples. Figure 6 shows means and standard deviations of the predicted fields.

```
R> predict <- spate.predict(y = w, tPred = (21:23), spateMCMC = spateMCMC,
+    Nsim = 100, BurnIn = 10, DataModel = "Normal", seed = 4)
R> mean <- apply(predict, c(1, 2), mean)
R> sd <- apply(predict, c(1, 2), sd)
```

7. Summary

We have presented tools for modeling high-dimensional space-time processes. This is done using a spatio-temporal Gaussian process defined through an SPDE. The package is thought to be used in two different modeling situations. In the first one, the spatio-temporal model is used as a component in a customized generalized linear mixed model (GLMM) or hierarchical

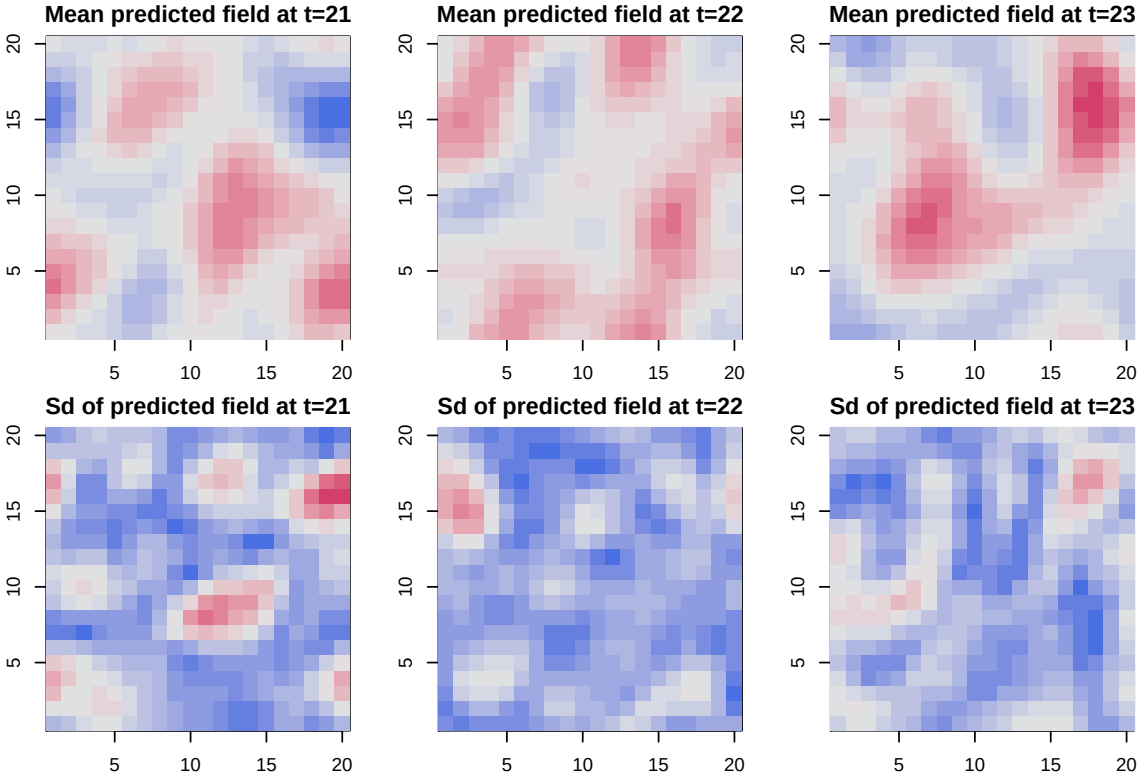

Figure 6: Means and standard deviations of predicted fields.

Bayesian model (HBM). In the second one, no additional modeling is done, and one can uses the adaptive MCMC algorithm implemented in the package. The latter is more convenient and less flexible, though.

Acknowledgments

We would like to thank Martin Mächler for his helpful support and advice. Furthermore, we are thankful for valuable comments from two anonymous referees and an editor.

References

Bakar KS, Sahu SK (2015). "**spTimer**: Spatio-Temporal Bayesian Modelling Using R." *Journal of Statistical Software*, **63**(15), 1–32. URL http://www.jstatsoft.org/v63/i15/.

Borgman L, Taheri M, Hagan R (1984). "Three-Dimensional Frequency-Domain Simulations of Geological Variables." In G Verly (ed.), *Geostatistics for Natural Resources Characterization*, pp. 517–541. D. Reidel.

Brown PE, Karesen KF, Roberts GO, Tonellato S (2000). "Blur-Generated Non-Separable Space-Time Models." *Journal of the Royal Statistical Society B*, **62**(4), 847–860.

Carter CK, Kohn R (1994). "On Gibbs Sampling for State Space Models." *Biometrika*, **81**(3), 541–553.

Cooley JW, Tukey JW (1965). "An Algorithm for the Machine Calculation of Complex Fourier Series." *Mathematics of Computation*, **19**, 297–301.

Cressie N, Wikle CK (2011). *Statistics for Spatio-Temporal Data*. John Wiley & Sons.

Dudgeon DE, Mersereau RM (1984). *Multidimensional Digital Signal Processing*. Englewood Cliffs.

Finley AO, Banerjee S, Carlin BP (2007). "**spBayes**: An R Package for Univariate and Multivariate Hierarchical Point-Referenced Spatial Models." *Journal of Statistical Software*, **19**(4), 1–24. URL http://www.jstatsoft.org/v19/i04/.

Finley AO, Banerjee S, Gelfand AE (2012). "Bayesian Dynamic Modeling for Large Space-Time Datasets Using Gaussian Predictive Processes." *Journal of Geographical Systems*, **14**(1), 29–47.

Finley AO, Banerjee S, Gelfand AE (2015). "**spBayes** for Large Univariate and Multivariate Point-Referenced Spatio-Temporal Data Models." *Journal of Statistical Software*, **63**(13), 1–28. URL http://www.jstatsoft.org/v63/i13/.

Frühwirth-Schnatter S (1994). "Data Augmentation and Dynamic Linear Models." *Journal of Time Series Analysis*, **15**, 183–202.

Heine V (1955). "Models for Two-Dimensional Stationary Stochastic Processes." *Biometrika*, **42**(1/2), 170–178.

Ihaka R, Murrell P, Hornik K, Fisher JC, Zeileis A (2013). ***colorspace**: Color Space Manipulation*. R package version 1.2-2, URL http://CRAN.R-project.org/package=colorspace.

Jones RH, Zhang Y (1997). "Models for Continuous Stationary Space-Time Processes." In TG Gregoire, DR Brillinger, PJ Diggle, E Russek-Cohen, WG Warren, RD Wolfinger (eds.), *Modelling Longitudinal and Spatially Correlated Data*, volume 122 of *Lecture Notes in Statistics*, pp. 289–298. Springer-Verlag, New-York.

Lindgren F, Rue H, Lindstrom 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.

Lindstrom J, Szpiro A, Sampson PD, Bergen S, Oron AP (2013). ***SpatioTemporal**: Spatio-Temporal Model Estimation*. R package version 1.1.7, URL http://CRAN.R-project.org/package=SpatioTemporal.

Nychka D, Furrer R, Sain S (2014). ***fields**: Tools for Spatial Data*. R package version 7.1, URL http://CRAN.R-project.org/package=fields.

Paciorek CJ (2007). "Bayesian Smoothing with Gaussian Processes Using Fourier Basis Functions in the **spectralGP** Package." *Journal of Statistical Software*, **19**(2), 1–38. URL http://www.jstatsoft.org/v19/i02/.

Pebesma E (2012). "**spacetime**: Spatio-Temporal Data in R." *Journal of Statistical Software*, **51**(7), 1–30. URL http://www.jstatsoft.org/v51/i07/.

Pebesma EJ (2004). "Multivariable Geostatistics in S: The **gstat** Package." *Computers & Geosciences*, **30**(7), 683 – 691.

Roberts GO, Rosenthal JS (2009). "Examples of Adaptive MCMC." *Journal of Computational and Graphical Statistics*, **18**(2), 349–367.

Royle JA, Wikle CK (2005). "Efficient Statistical Mapping of Avian Count Data." *Environmental and Ecological Statistics*, **12**, 225–243.

Rue H, Held L (2005). *Gaussian Markov Random Fields*, volume 104 of *Monographs on Statistics and Applied Probability*. Chapman & Hall/CRC, Boca Raton.

Schlather M, Malinowski A, Menck PJ, Oesting M, Strokorb K (2015). "Analysis, Simulation and Prediction of Multivariate Random Fields with Package **RandomFields**." *Journal of Statistical Software*, **63**(8), 1–25. URL http://www.jstatsoft.org/v63/i08/.

Sigrist F, Künsch HR, Stahel WA (2012). "A Dynamic Nonstationary Spatio-Temporal Model for Short Term Prediction of Precipitation." *Annals of Applied Statistics*, **6**(4), 1452–1477.

Sigrist F, Künsch HR, Stahel WA (2013). ***spate**: Spatio-Temporal Modeling of Large Data Using a Spectral SPDE Approach*. R package version 1.3, URL http://CRAN.R-project.org/package=spate.

Sigrist F, Künsch HR, Stahel WA (2015). "Stochastic Partial Differential Equation Based Modeling of Large Space-Time Data Sets." *Journal of the Royal Statistical Society B*, **77**(1), 3–33.

Whittle P (1954). "On Stationary Processes in the Plane." *Biometrika*, **41**(3/4), 434–449.

Whittle P (1962). "Topographic Correlation, Power-Law Covariance Functions, and Diffusion." *Biometrika*, **49**(3/4), 305–314.

Wikle CK, Berliner LM, Cressie N (1998). "Hierarchical Bayesian Space-Time Models." *Environmental and Ecological Statistics*, **5**, 117–154.

Wikle CK, Milliff RF, Nychka D, Berliner LM (2001). "Spatiotemporal Hierarchical Bayesian Modeling: Tropical Ocean Surface Winds." *Journal of the American Statistical Association*, **96**(454), 382–397.

Zeileis A, Hornik K, Murrell P (2009). "Escaping RGBland: Selecting Colors for Statistical Graphics." *Computational Statistics & Data Analysis*, **53**, 3259–3270.

Affiliation:

Fabio Sigrist
Seminar for Statistics
ETH Zurich
8092 Zurich, Switzerland
E-mail: sigrist@stat.math.ethz.ch

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 63, Issue 14 *Submitted:* 2013-04-09
January 2015 *Accepted:* 2014-10-13