`spate`: an R Package for Statistical Modeling with SPDE Based Spatio-Temporal Gaussian Processes

Fabio Sigrist, Hans R. Künsch, Werner A. Stahel
Seminar for Statistics, Department of Mathematics, ETH Zürich
8092 Zürich, Switzerland

November 29, 2012

Abstract

The R package `spate` provides tools 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 two different modeling approaches. First, the SPDE based spatio-temporal model can be used as a component in a customized hierarchical Bayesian model (HBM) or generlized linear mixed model (GLMM). The functions of the package then provide parametrizations of the process part of the model as well as computationally efficient algorithms needed for doing inference with the hierarchical model. Alternatively, the adaptive MCMC algorithm implemented in the package can be used as an algorithm for doing inference without any additional modeling. 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.

Contents

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 infeasible if data sets are large. This is due to the fact that for doing inference, frequentist or Bayesian, covariance matrices need to be factorized which is computationally expensive. Alternatively, a Gaussian process can be specified through a stochastic partial differential equation (SPDE), which implicitly also gives a covariance function. 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 which is computationally advantageous (see Sigrist et al. (2012)). 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. This package implements the methodology presented in Sigrist et al. (2012).

The package `spate` 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 et al., 1998) using a 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 algorithm. On the other hand, the package also provides a function for Bayesian inference using a Monte Carlo Markov chain (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. Finally, functions for plotting and simulation of space-time processes are also provided.

1.1 Notation

Since some readers might skip the next section, we start by establishing the principal notation. The Gaussian process modeling structured variation in space

and time is denoted by $\xi(t_i, \boldsymbol{s}_l)$, $i = 1, \ldots, T$, $l = 1, \ldots, N$. In vectorized form, we write $\boldsymbol{\xi}(t_i) = (\xi(t_i, \boldsymbol{s}_1), \ldots, \xi(t_i, \boldsymbol{s}_N))'$ (where stacking is done first over the x-axis and then over the y-axis), and $\boldsymbol{\xi} = (\boldsymbol{\xi}(t_1)', \ldots, \boldsymbol{\xi}(t_T)')'$. For each t_i, $\boldsymbol{\xi}(t_i) = \boldsymbol{\Phi}\boldsymbol{\alpha}(t_i)$ is the Fourier transform of the Fourier coefficients $\boldsymbol{\alpha}(t_i)$, $\boldsymbol{\alpha} = (\boldsymbol{\alpha}(t_1)', \ldots, \boldsymbol{\alpha}(t_T)')'$. The observed Gaussian process $w(t_i, \boldsymbol{s}_l)$ equals the latent $\xi(t_i, \boldsymbol{s}_l)$ plus a measurement error. $\boldsymbol{w}(t_i)$ and $\boldsymbol{w}$ are defined analogously. If the observations are censored, i.e., if they follow a skewed Tobit distribution, the observed data is denoted by $y(t_i, \boldsymbol{s}_l)$. Finally, $\boldsymbol{\theta} = (\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2)'$ denotes the vector of hyper-parameters.

We assume that we model the process on a regular, rectangular grid of $n \times n = N$ spatial locations $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N$ in $[0, 1]^2$ 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.

2 Summary of modeling background

In the following, we briefly recall the underlying model and methodology. For more details we refer to Sigrist et al. (2012).

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

$$\frac{\partial}{\partial t}\xi(t, \boldsymbol{s}) = -\boldsymbol{\mu} \cdot \nabla\xi(t, \boldsymbol{s}) + \nabla \cdot \boldsymbol{\Sigma}\nabla\xi(t, \boldsymbol{s}) - \zeta\xi(t, \boldsymbol{s}) + \epsilon(t, \boldsymbol{s}), \tag{1}$$

where $t \geq 0$, $\boldsymbol{s} \in [0, 1]^2$ wrapped on a torus, $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)'$ 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 $\boldsymbol{F} = (F^x, F^y)'$ being a vector field, $\boldsymbol{\mu} = (\mu_x, \mu_y)'$,

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

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

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

For the parameters, we have the following restrictions

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

The parameters are interpreted as follows. 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 ν is not estimable from data, we take $\nu = 1$ by default, which corresponds to the Whittle covariance function.

2.2 Spectral solution

As is shown in Sigrist et al. (2012), inference can be done computationally efficiently when solving the SPDE in the spectral space. The latter means, roughly speaking, that the solution is represented a linear combination of deterministic, real Fourier basis functions

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

with random coefficients $\alpha_j^c(t)$, $\alpha_j^s(t)$ that evolve dynamically over time according to a vector autoregression. Fourier functions have several advantages for solving the SPDE (1). 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 FFT for efficiently transforming from the physical to the spectral space, and vice versa.

As is customary for spatial and spatio-temporal models, we add an non-structured Gaussian term $\nu(t, \boldsymbol{s}) \sim N(0, \tau^2)$ that is white in time and space to $\xi(t, \boldsymbol{s})$. This term accounts for small scale variation and / or measurement errors (nugget effect). Solving the SPDE in the spectral space and discretizing in time and space, we obtain the following linear Gaussian state space model:

$$\boldsymbol{w}(t_{i+1}) = \boldsymbol{\xi}(t_{i+1}) + \boldsymbol{\nu}(t_{i+1}), \boldsymbol{\nu}(t_{i+1}) \sim N(0, \tau^2 \boldsymbol{1}), \tag{2}$$

$$\boldsymbol{\xi}(t_{i+1}) = \boldsymbol{\Phi} \boldsymbol{\alpha}(t_{i+1}), \tag{3}$$

$$\boldsymbol{\alpha}(t_{i+1}) = \boldsymbol{G} \boldsymbol{\alpha}(t_i) + \widehat{\boldsymbol{\epsilon}}(t_{i+1}), \quad \widehat{\boldsymbol{\epsilon}}(t_{i+1}) \sim N(0, \widehat{\boldsymbol{Q}}). \tag{4}$$

The observation equation (2) specifies how the observation field $\boldsymbol{w}(t_{i+1}) = (w(t_i, \boldsymbol{s}_1), \ldots, w(t_i, \boldsymbol{s}_N))'$ at time t_{i+1} is related to the spatio-temporal process $\boldsymbol{\xi}(t_{i+1})$. See below on how to handle non-Gaussian data. In the second equation (3), the matrix $\boldsymbol{\Phi}$,

$$\boldsymbol{\Phi} = \Big[\boldsymbol{\phi}(\boldsymbol{s}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_N)\Big]'$$

$$\boldsymbol{\phi}(\boldsymbol{s}_l) = \Big(\phi_1^{(c)}(\boldsymbol{s}_l), \ldots, \phi_4^{(c)}(\boldsymbol{s}_l), \phi_5^{(c)}(\boldsymbol{s}_l), \phi_5^{(s)}(\boldsymbol{s}_l), \ldots, \phi_{(K-4)/2}^{(c)}(\boldsymbol{s}_l), \phi_{(K-4)/2}^{(s)}(\boldsymbol{s}_l)\Big)',$$

applies the discrete, real Fourier transformation to the coefficients

$$\boldsymbol{\alpha}(t) = \left(\alpha_1^{(c)}(t), \ldots, \alpha_4^{(c)}(t), \alpha_5^{(c)}(t), \alpha_5^{(s)}(t), \ldots, \alpha_{(K-4)/2}^{(c)}(t), \alpha_{(K-4)/2}^{(s)}(t)\right)'.$$

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 (see Figure 1). 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.

The third equation (4) 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 covariance matrix $\widehat{\boldsymbol{Q}}$ is a diagonal matrix. These two matrices are defined as follows:

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

and

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

Figure 4 shows an example of a propagator matrix $\boldsymbol{G}$.

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

$$\begin{aligned} \xi(t, \boldsymbol{s}_l) =& \sum_{l=1}^{4} \alpha_j^{(c)}(t)\phi_j^{(c)}(\boldsymbol{s}_l) \\ &+ \sum_{l=5}^{K/2+2} \alpha_j^{(c)}(t)\phi_j^{(c)}(\boldsymbol{s}_l) + \alpha_j^{(s)}(t)\phi_j^{(s)}(\boldsymbol{s}_l) \\ =& \boldsymbol{\phi}(\boldsymbol{s}_l)'\boldsymbol{\alpha}(t), \end{aligned} \tag{7}$$

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 $n^2 = N$. Figure 1 illustrates the spatial wavenumbers on a 20×20 grid. The red crosses mark the first four spatial wavenumbers having only a cosine term. The remaining dots with a circle around represent the wavenumbers used by the cosine - sine pairs in the real Fourier transform.

To get an idea how the basis functions $\cos(\boldsymbol{k}_j'\boldsymbol{s})$ and $\sin(\boldsymbol{k}_j'\boldsymbol{s})$ look like, we plot in Figure 2 twelve low-frequency basis functions corresponding to the six spatial frequencies closest to the origin $\mathbf{0}$ (see Figure 1).

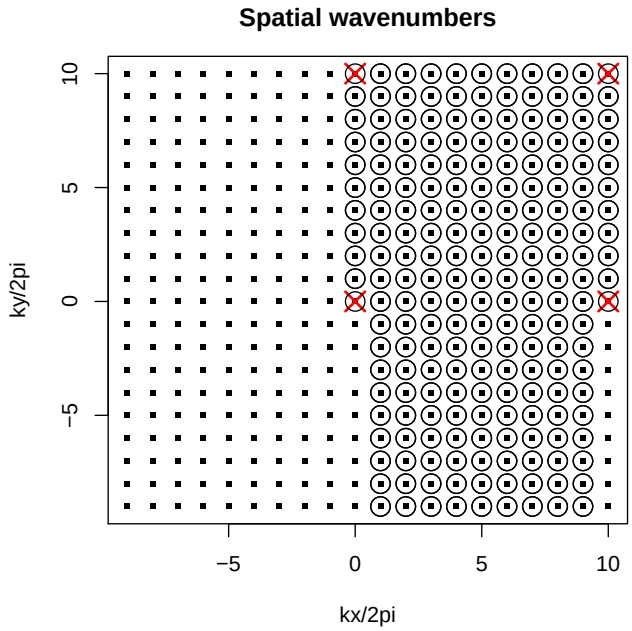

Figure 1: Illustration of wavenumbers used in the two-dimensional discrete real Fourier transform with $n^2 = 400$ grid points.

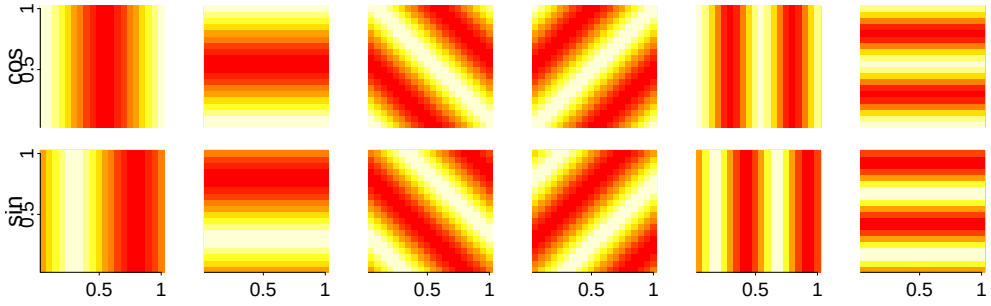

Figure 2: Illustration of two dimensional Fourier basis functions used in the discrete real Fourier transform. On the x- and y-axis are the coordinates of $\boldsymbol{s} = (s_x, s_y)'$.

2.3 Non-Gaussian data, covariates, and missing data

Spatio-temporal covariates $x_p(t_i, \boldsymbol{s}_l)$, $p = 1, \ldots, P$ can easily be included in the model by adding a regression term to the equation (2):

$$w(t_i, \boldsymbol{s}_l) = \sum_{p=1}^{P} \beta_p \cdot x_p(t_i, \boldsymbol{s}_l) + \xi(t_{i+1}, \boldsymbol{s}_l) + \nu(t_{i+1}, \boldsymbol{s}_l). \tag{8}$$

Non-Gaussian data can be modeled, for instance, in the framework of generalized linear mixed models (GLMM) or hierarchical Bayesian models. 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 $\sum_{p=1}^{P} \beta_p \cdot x_p(t_i, \boldsymbol{s}_l) +$

$\xi(t_{i+1}, \boldsymbol{s}_l)$. Note that fitting such models is a non-trivial task and 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

$$y(t_i, \boldsymbol{s}_l) = \max(0, w(t_i, \boldsymbol{s}_l))^\lambda, \tag{9}$$

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 (9), can be easily dealt with using a data augmentation approach. See, e.g., Sigrist et al. (2012) for more details. In particular, if the observations do not lie on a regular spatial grid, the grid cells where no observations are made can be assumed to have missing data.

2.4 Computationally efficient frequentist and Bayesian inference

When doing inference, for both data models, the Gaussian one in (2) and the transformed Tobit model (9), the main difficulty consists in evaluating the likelihood $\ell(\boldsymbol{\theta}) = P[\boldsymbol{\theta}|\boldsymbol{w}]$, $\boldsymbol{\theta} = (\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2)'$, and in simulating from the full conditional of the coefficients $[\boldsymbol{\alpha}|\boldsymbol{w}, \boldsymbol{\theta}]$, where $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ denote the full space-time fields. As shown in Sigrist et al. (2012), this can be done in $O(TN)$ time in the spectral space using the Kalman filter and a backward sampling step. The fast Fourier transform (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 $O(TN \log N)$.

2.5 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 (2), the observation equation is then

$$\boldsymbol{w}(t_{i+1}) = \boldsymbol{I}\boldsymbol{\Phi}\boldsymbol{\alpha}(t_{i+1}) + \boldsymbol{\nu}(t_{i+1}), \quad \boldsymbol{\nu}(t_{i+1}) \sim N(0, \tau^2 \mathbf{1}_K). \tag{10}$$

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

3 Parametrization of the dynamic space-time model

3.1 Innovation spectrum $\widehat{Q}$ and Matérn spectrum

The function `innov.spec` returns the spectrum $\widehat{\boldsymbol{Q}}$ of the integrated stochastic innovation field $\widehat{\boldsymbol{\epsilon}}(t_{i+1})$ 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 details on how to calculate the Fourier transform. Figure 3 shows sample fields from the Whittle process and from the stochastic innovation process.

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

3.2 Propagator matrix G

The function `get.propagator` returns the spectral propagator matrix $\boldsymbol{G}$ as defined in (5). Figure 4 shows an example of a propagator matrix $\boldsymbol{G}$. The code before the figure illustrates how `get.propagator` is used.

```
> n <- 4
> wave <- wave.numbers(n)
> 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)
```

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

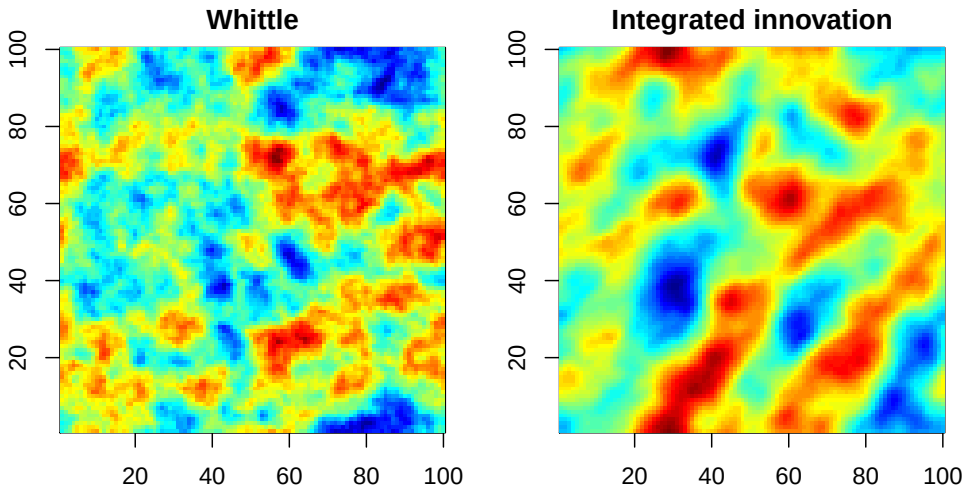

Figure 3: Samples from Gaussian processes with Whittle covariance function and the covariance function of the integrated stochastic innovation field $\widehat{\boldsymbol{\epsilon}}(t_{i+1})$.

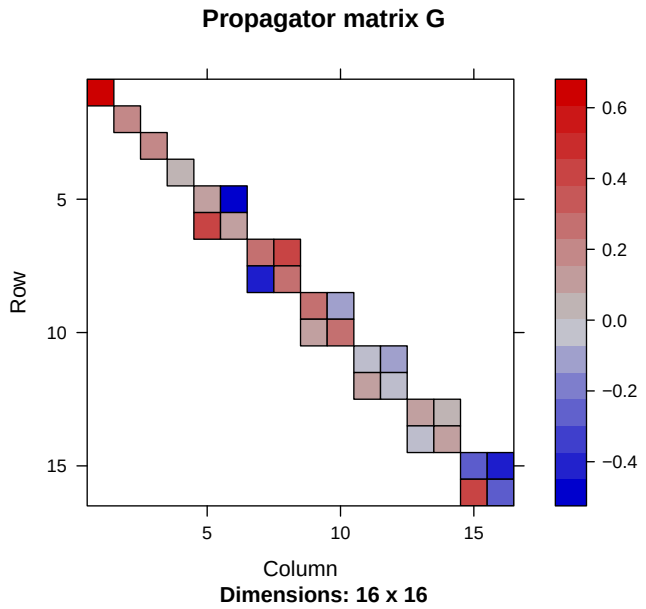

Figure 4: Illustration of propagator matrix $\boldsymbol{G}$.

the propagator matrix $\boldsymbol{G}$ in vectorized from as obtained from the function `get.propagator.vec`. Figure 5 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 $\boldsymbol{G\alpha}(t)$. The code shows that actually calculating $\boldsymbol{G\alpha}(t)$ and applying `propagate.spectral` are equivalent.

```
> n <- 50
> wave <- wave.numbers(n)
> spec <- matern.spec(wave=wave[["wave"]],n=n,
+                     rho0=0.05,sigma2=1,norm=TRUE)
> ## Initial state
> alphat <- sqrt(spec)*rnorm(n*n)
> ## Propagate state
> 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)
> alphat1a <- as.vector(G%*%alphat)
> 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)
> alphat1b <- propagate.spectral(alphat,n=n,Gvec=Gvec)
> ## Both methods do the same thing:
> sum(abs(alphat1a-alphat1b))

[1] 0
```

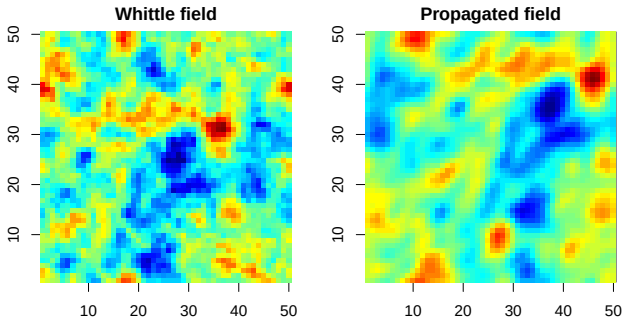

Figure 5: Illustration of spectral propagation: initial and propagated field.

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 et al. (1984), Royle and Wikle (2005), and Paciorek (2007).

The function `get.real.dft.mat` returns the matrix $\mathbf{\Phi}$ (see (3)) 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 $\mathbf{\Phi}$. The following code shows how $\mathbf{\Phi}$ can be constructed using `get.real.dft.mat`.

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

As another example of the use of the two-dimensional real Fourier transform, the following code shows how an image can be reconstructed with varying resolution. In the code, we first define a two-dimensional image on a 50×50 grid. We then construct three different $\mathbf{\Phi}_i$s using the function `get.real.dft.mat`. Dimension reduction is done using the function `spate.init`. The argument `NF` specifies the number of Fourier functions. Since the image is defined on a 50×50 grid, the total number of Fourier terms is 2500. As can be seen in the code, we construct reduced dimensional $\mathbf{\Phi}_i$s with `NF=45` and `NF=101`. Reduced dimensional reconstructions of the image $\mathbf{\Psi}$ are the obtained by calculating $\mathbf{\Phi}_i\mathbf{\Phi}_i'\mathbf{\Psi}$. Figure 6 shows the results.

```
> ## Example: reduced dimensional image reconstruction
> n <- 50
> ## Define image
> image <- rep(0,n*n)
> for(i in 1:n){
+   for(j in 1:n){
+     image[(i-1)*n+j] <- cos(5*(i-n/2)/n*pi)*sin(5*(j)/n*pi)*
+       (1-abs(i/n-1/2)-abs(j/n-1/2))
+   }
+ }
> ## Low-dimensional: only 45 (of potentially 2500) Fourier functions
> spateObj <- spate.init(n=n,T=17,NF=45)
> Phi.LD <- get.real.dft.mat(wave=spateObj$wave, indCos=spateObj$indCos,
+                   ns=spateObj$ns, n=n)
> ## Mid-dimensional: 545 (of potentially 2500) Fourier functions
> spateObj <- spate.init(n=n,T=17,NF=101)
> Phi.MD <- get.real.dft.mat(wave=spateObj$wave, indCos=spateObj$indCos,
+                   ns=spateObj$ns, n=n)
> ## High-dimensional: all 2500 Fourier functions
> spateObj <- spate.init(n=n,T=17,NF=2500)
> Phi.HD <- get.real.dft.mat(wave=spateObj$wave, indCos=spateObj$indCos,
+                   ns=spateObj$ns, n=n)
> ## Aply inverse Fourier transform, dimension reduction,
> ## and then Fourier transform
> image.LD <- Phi.LD %*% (t(Phi.LD) %*% image)
```

```
> image.MD <- Phi.MD %*% (t(Phi.MD) %*% image)
> image.HD <- Phi.HD %*% (t(Phi.HD) %*% image)
```

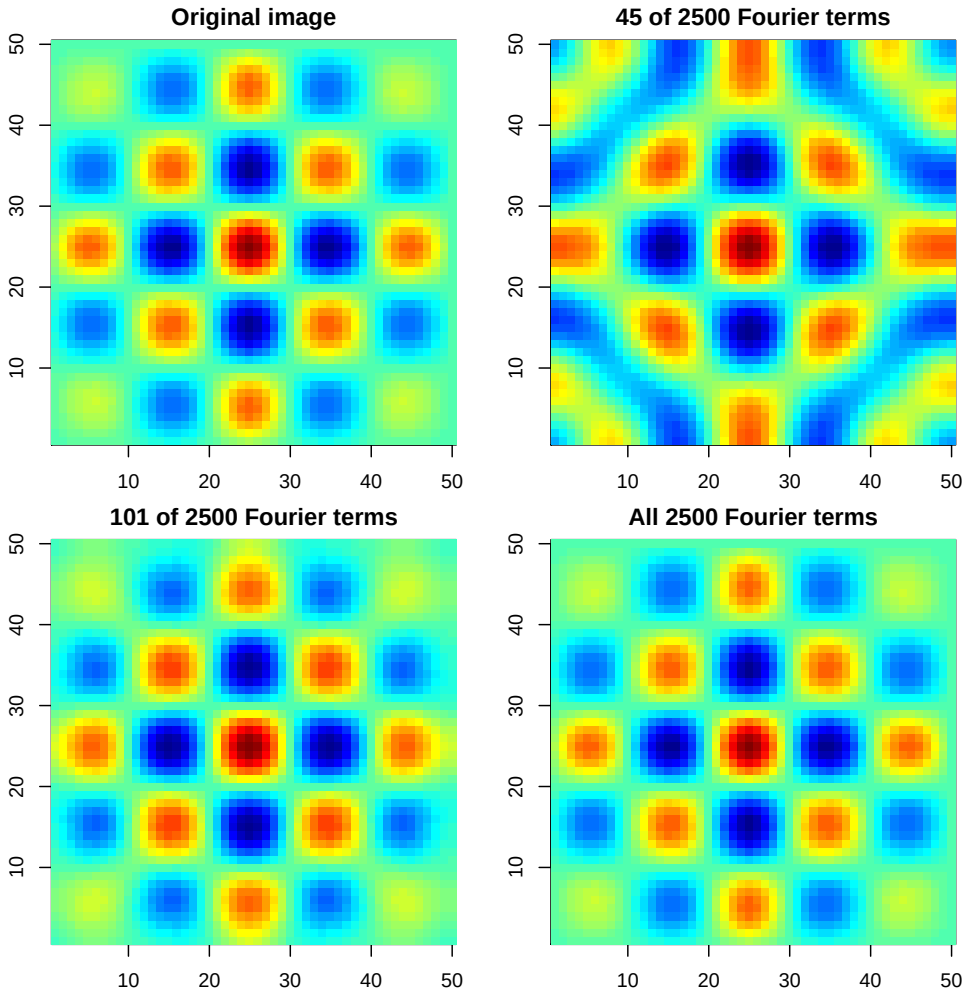

Figure 6: Example of use of Fourier transform: reduced dimensional image reconstruction

4 Simulation and plotting

The function `spate.sim` allows for simulating from the SPDE based spatio-temporal Gaussian process model defined through (3) and (4). 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 $\boldsymbol{\theta} = (\rho_0, \sigma^2, \zeta, \rho_1, \gamma, \alpha, \mu_x, \mu_y, \tau^2)'$. 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 code below illustrates the use of these functions. 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 7 shows one example of a simulated space-time process.

```
> StartVal <- rep(0,100^2)
> StartVal[75*100+75] <- 1000
> 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)
> spateSim <- spate.sim(par=par,n=100,T=5,StartVal=StartVal,seed=1)
> plot(spateSim,mfrow=c(1,5),mar=c(2,2,2,2),indScale=TRUE,
+      cex.axis=1.5,cex.main=2)
```

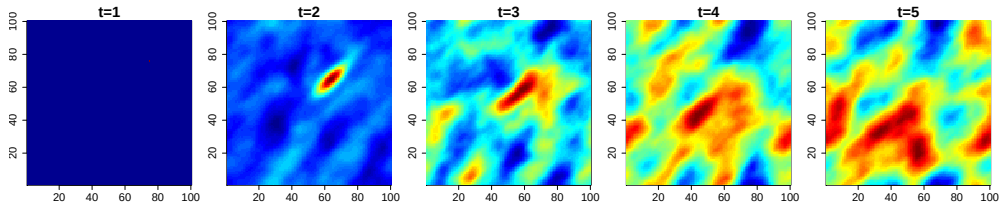

Figure 7: Simulated spatio-temporal Gaussian process as defined in (3) and (4)

5 Inference: log-likelihood evaluation 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 (2), (3), and (4). 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 8, 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.

```
> ## Example of use of 'sample.four.coef'
> ## Simulate data
> n <- 50
> T <- 4
> 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)
> spateSim <- spate.sim(par=par,n=n,T=T,seed=4)
> w <- spateSim$w
> ## Sample from full conditional
> Nmc <- 50
> alphaS <- array(0,c(T,n*n,Nmc))
> wFT <- real.fft.TS(w,n=n,T=T)
> for(i in 1:Nmc){
+   alphaS[,,i] <- sample.four.coef(wFT=wFT,par=par,n=n,T=T,NF=n*n)
+ }
> ## Mean from full conditional
> alphaMean <- apply(alphaS,c(1,2),mean)
> xiMean <- real.fft.TS(alphaMean,n=n,T=T,inv=FALSE)
```

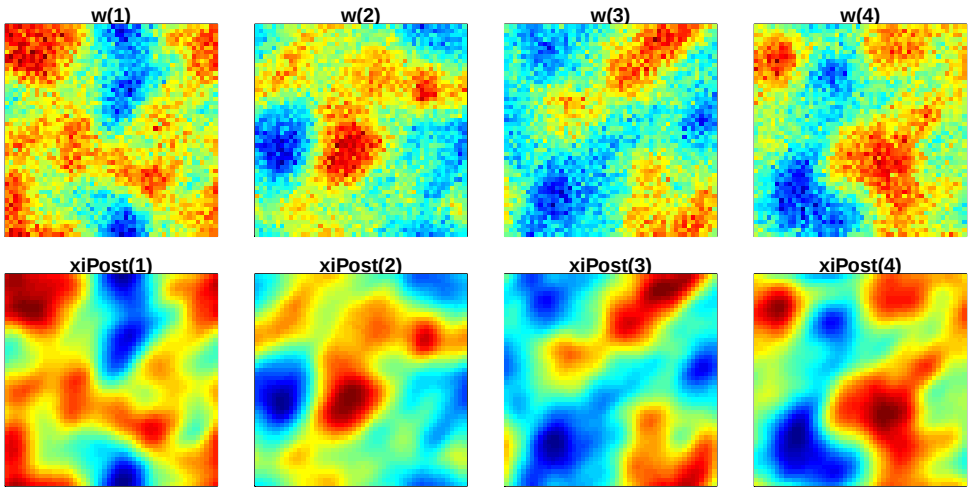

Figure 8: Sampling from the full conditional of the coefficients: comparison of observed data (top plots) and mean of full conditional of $\boldsymbol{\xi}$ (bottom plots).

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.

```
> ## Evaluation of log-likelihood
> loglike(par=par,w=w,n=n,T=T)

[1] 7861.001

> ## Equivalently, one can use the Fourier transformed data 'wFT'
> 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(tm) 64 X2 Dual Core Processor 5600+). This is achieved without applying any dimension reduction. The entire inference takes less than 12 seconds.

```
> ## Simulate data
> n <- 20
> T <- 20
> 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)
> spateSim <- spate.sim(par=par,n=n,T=T,seed=4)
> w <- spateSim$w
> ## Initial values for optim
> 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)
> ## Transform to log-scale
> logInd=c(1,2,3,4,5,9)
> parI[logInd] <- log(parI[logInd])
> ## Maximum likelihood estimation using optim
> wFT <- real.fft.TS(w,n=n,T=T)
> spateMLE <- optim(par=parI,loglike,control=list(trace=TRUE,maxit=1000),
+                   wFT=wFT,method="L-BFGS-B",
+                   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),
+                    negative=TRUE,logScale=TRUE,
+                    logInd=c(1,2,3,4,5,9),hessian=TRUE,n=n,T=T)

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

> mle <- spateMLE$par
> mle[logInd] <- exp(mle[logInd])
> sd=sqrt(diag(solve(spateMLE$hessian)))
> ## Calculate confidence intervals
> MleConfInt <- data.frame(array(0,c(4,9)))
> colnames(MleConfInt) <- names(par)
> rownames(MleConfInt) <- c("True","Estimate","Lower","Upper")
> MleConfInt[1,] <- par
> MleConfInt[2,] <- mle
> MleConfInt[3,] <- spateMLE$par-2*sd
> MleConfInt[4,] <- spateMLE$par+2*sd
> MleConfInt[c(3,4),logInd] <- exp(MleConfInt[c(3,4),logInd])
> ## Results: estimates and confidence intervals
> round(MleConfInt,digits=3)

          rho0 sigma2  zeta  rho1 gamma alpha   muX    muY  tau2
True     0.100  0.200 0.500 0.100 2.000 0.785 0.200 -0.200 0.010
Estimate 0.092  0.166 0.357 0.105 2.209 0.845 0.213 -0.177 0.010
Lower    0.077  0.136 0.180 0.088 1.847 0.753 0.177 -0.214 0.010
Upper    0.111  0.203 0.710 0.126 2.643 0.937 0.249 -0.139 0.011
```

5.4 Bayesian inference using MCMC

Using `sample.four.coef` and `loglike` a Markov chain Monte Carlo (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 reasons 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

$$\min\left(1, \frac{P[\boldsymbol{\theta}^*|\boldsymbol{w}]}{P[\boldsymbol{\theta}^{(i)}|\boldsymbol{w}]}\right), \tag{11}$$

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`.

5.4.1 Skewed Tobit model and missing data

For the transformed Tobit model (9), 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 given data, and that data 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`

- 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.

- The number of samples to be drawn from the Markov chain is specified in `Nmc` and the length of the burn-in in `BurnIn`.
- 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 specifying through `NFour` the number of Fourier functions to be used.
- If the observations `y` are not on a grid, `y` can be a $T \times N$ matrix where $N(< n^2)$ is the number of observation stations, and the coordinates of the stations can be specified in the $N \times 2$ matrix `coord`. Alternatively, one can specify through the vector `Sind` at which grid point each observation lies.
- If the boolean argument `IncidenceMat` equals `TRUE`, an incidence matrix $\boldsymbol{I}$ is constructed and the model in (10) 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=(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.

For more details and explanations on, e.g., starting values, specification of prior distributions, selection of output for monitoring the MCMC algorithm, etc., see the help information of `spate.mcmc`.

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 specifing 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`=diag(σ_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 9 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. The figure 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.

```
> ## Simulate data
> 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)
> spateSim <- spate.sim(par=par,n=20,T=20,seed=4)
> w <- spateSim$w
> ## This is an example to illustrate the use of the MCMC algorithm.
> ## In practice, more samples (Nmc) are needed for a sufficiently
> ## large effective sample size.
> 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(0.005,0.005,0.05,0.005,
+                           0.005,0.001,0.0002,0.0002,0.0002)),
+                         Nmc=10000,BurnIn=2000,seed=4,NCovEst=500,
+                         BurnInCovEst=500,trace=FALSE)

> 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
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
```

The following code illustrates the use of `spate.mcmc` when an incidence matrix approach (see (10)) is used in combination with dimension reduction. This is the real data application used in Sigrist et al. (2012) 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. (2012). 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 consiting 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.

```
> 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-

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

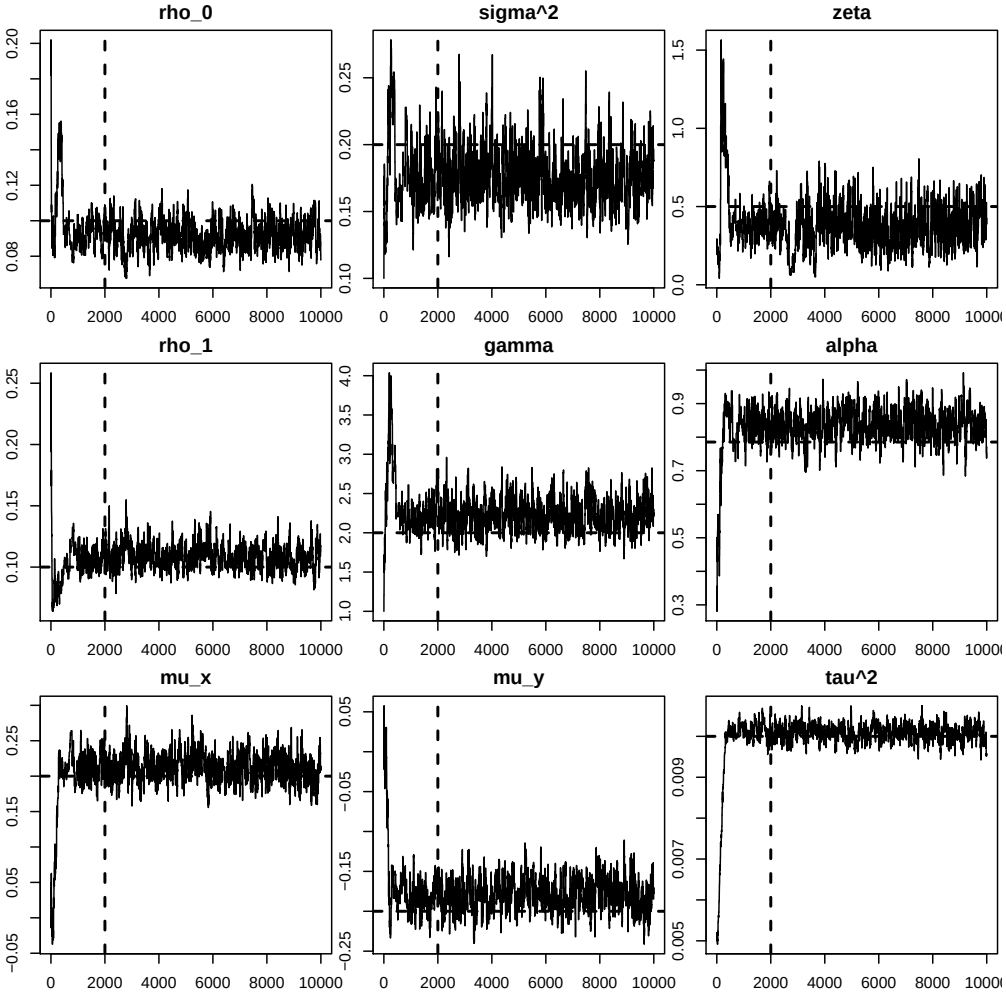

Figure 9: Trace plots from the MCMC algorithm. The vertical line shows the burn-in, and the horizontal lines are the true values of the parameters.

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 10 shows means and standard deviations of the predicted fields.

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

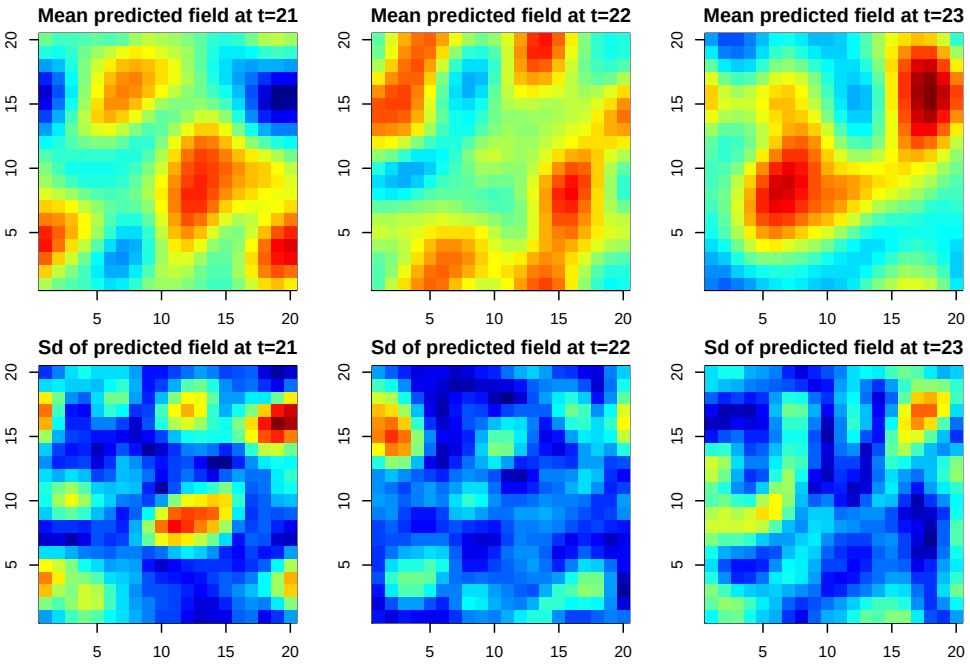

Figure 10: Means and standard deviations of predicted fields.

Acknowledgments

We would like to thank Martin Mächler for his helpful support and advice.

References

Borgman, L., Taheri, M., and Hagan, R. (1984), "Three-dimensional frequency-domain simulations of geological variables," in *Geostatistics for Natural Resources Characterization*, ed. Verly, G., D. Reidel, pp. 517–541.

Cressie, N. and Wikle, C. K. (2011), *Statistics for spatio-temporal data*, Wiley Series in Probability and Statistics, John Wiley & Sons, Inc.

Dudgeon, D. E. and Mersereau, R. M. (1984), *Multidimensional digital signal processing*, Prentice-Hall.

Paciorek, C. J. (2007), "Bayesian smoothing with Gaussian processes using Fourier basis functions in the spectralGP package," *Journal of Statistical Software*, 19, 1–38.

Roberts, G. O. and Rosenthal, J. S. (2009), "Examples of adaptive MCMC," *Journal of Computational and Graphical Statistics*, 18, 349–367.

Royle, J. A. and Wikle, C. K. (2005), "Efficient statistical mapping of avian count data," *Environmental and Ecological Statistics*, 12, 225–243.

Sigrist, F., Künsch, H. R., and Stahel, W. A. (2012), "A Dynamic Nonstationary Spatio-temporal Model for Short Term Prediction of Precipitation," *Annals of Applied Statistics (to appear)*, 6, –.

Sigrist, F., Künsch, H. R., and Stahel, W. A. (2012), "SPDE based modeling of large space-time data sets," *Preprint (http://arxiv.org/abs/1204.6118)*.

Wikle, C. K., Berliner, L. M., and Cressie, N. (1998), "Hierarchical Bayesian space-time models," *Environmental and Ecological Statistics*, 5, 117–154.