Fixed Rank Kriging: The `R` package

Andrew Zammit-Mangion and Noel Cressie

February 18, 2022

Abstract

FRK is an R software package for spatial/spatio-temporal modelling and prediction with large datasets. It facilitates optimal spatial prediction (kriging) on the most commonly used manifolds (in Euclidean space and on the surface of the sphere), for both spatial and spatio-temporal fields. It differs from existing packages for spatial modelling and prediction by avoiding stationary and isotropic covariance and variogram models, instead constructing a spatial random effects (SRE) model on a discretised spatial domain. The discrete element is known as a basic areal unit (BAU), whose introduction in the software leads to several practical advantages. The software can be used to (i) integrate multiple observations with different supports with relative ease; (ii) obtain exact predictions at millions of prediction locations with the use of sparse linear algebraic techniques (without conditional simulation); and (iii) distinguish between measurement error and fine-scale variation at the resolution of the BAU, thereby allowing for improved uncertainty quantification when compared to related packages. The temporal component is included by adding another dimension. A key component of the SRE model is the specification of spatial or spatio-temporal basis functions; they can be generated automatically or by the user. The package also offers automatic BAU construction, an Expectation Maximisation (EM) algorithm for parameter estimation, and functionality for prediction over any user-specified polygons or BAUs. Use of the package is illustrated on several large spatial and spatio-temporal datasets.

Contents

1 Introduction

Fixed Rank Kriging (FRK) is a spatial/spatio-temporal modelling and prediction framework that is scaleable, works well with large datasets, and can change spatial support easily. FRK hinges on the use of a spatial random effects (SRE) model, in which a spatially correlated mean-zero random process is decomposed using a linear combination of spatial basis functions with random weights plus a term that captures the random process' fine-scale variation. Dimensionality reduction through a relatively small number of basis functions ensures computationally efficient prediction, while the reconstructed spatial process is, in general, non-stationary. The SRE model has a spatial covariance function that is always nonnegative-definite, and, because any (possibly non-orthogonal) basis functions can be used, it can be constructed so as to approximate standard families of covariance functions [Kang and Cressie, 2011]. For a detailed treatment of FRK, see Cressie and Johannesson [2006, 2008], Shi and Cressie [2007], and Nguyen et al. [2012].

There are numerous R packages available for modelling and prediction with spatial or spatio-temporal data,[1] although relatively few of these make use of a model with spatial basis functions. However, a few variants of FRK have been developed to date, and the one that comes closest to the present software is **LatticeKrig** [Nychka et al., 2015]. **LatticeKrig** uses Wendland basis functions (that have compact support) to decompose the spatially correlated process, and it also has a Markov assumption to construct a precision matrix (the matrix $\mathbf{K}^{-1}$ in Section 2.1) to describe the dependence between the coefficients of these basis functions. It does not cater for what we term fine-scale-process variation, and instead the finest scale of the process is limited to the finest resolution of the basis functions used. However, this scale can be relatively fine due to the computationally motivated sparsity imposed on $\mathbf{K}^{-1}$. **LatticeKrig**'s underlying model makes use of sparse precision matrices constructed using Gaussian Markov random field (GMRF) assumptions, which results in efficient computations and the potential use of a large number ($> 10,000$) of basis functions.

The package **INLA** is a general-purpose package for model fitting and prediction. When using **INLA** for spatial and spatio-temporal modelling, the prevalent approach is to assume that basis functions are triangular 'tent' functions and that the coefficients are normally distributed with a sparse precision matrix, such that the covariance function of the resulting Gaussian process is approximately a spatial covariance function from the Matérn class [see Lindgren and Rue, 2015, for details on software implementation]. **INLA**'s approach thus shares many of the features of **LatticeKrig**. A key advantage of **INLA** is that once the spatial or spatio-temporal model is constructed, one has access to all the approximate-inference machinery and likelihood models available within the package.

Kang and Cressie [2011] develop Bayesian FRK; they keep the spatial basis functions fixed and put a prior distribution on $\mathbf{K}$. The predictive-process approach of Banerjee et al. [2008] can also be seen as a type of Bayesian FRK, where the basis functions are constructed from the postulated covariance function of the spatial random effects and hence depend on parameters [see Katzfuss and Hammerling, 2017, for an equivalence argument]. An R package that implements predictive processes is **spBayes** [Finley et al., 2007]. It allows for multivariate spatial or spatio-temporal processes, and Bayesian inference is carried out using Markov chain Monte Carlo (MCMC), thus allowing for a variety of likelihood models. Because the implied basis functions are constructed based on a parametric covariance model, a prior distribution on parameters reults in new basis functions generated at each MCMC iteration. Since this can slow down the computation, the number of knots used in predictive processes needs to be small.

Our software package **FRK** differs from spatial prediction packages currently available by constructing an SRE model on a discretised domain, where the discrete element is known as a basic areal unit [BAU; see, e.g., Nguyen et al., 2012]. Reverting to discretised spatial processes might appear to be counter-intuitive, given all the theory and efficient approaches available for continuous-domain processes. However, BAUs allow one to easily combine multiple observations with different supports, which is common when working with, for example, remote sensing datasets. Further, the consideration of a discrete element allows one to distinguish between measurement error and fine-scale variation at the resolution of the discrete element which leads to better uncertainty quantification. The BAUs need to be 'small,' in the sense that they should be able to reconstruct the (undiscretised) process with minimal error, but **FRK** implements functions to predict over any arbitrary user-defined polygons.

In the standard "flavour" of **FRK** [Cressie and Johannesson, 2008], which we term *vanilla* FRK (FRK-V), there is an explicit reliance on multi-resolution basis functions to give complex non-stationary spatial

[1] see `https://cran.r-project.org/web/views/Spatial.html`.

patterns at the cost of not imposing any structure on $\mathbf{K}$, the covariance matrix of the basis function weights. This can result in identifiability issues and hence can result in over-fitting the data when $\mathbf{K}$ is estimated using standard likelihood methods [e.g., Nguyen et al., 2014], especially in regions of data paucity. Therefore, in **FRK** we also implement a model (FRK-M) where a parametric structure is imposed on $\mathbf{K}$ [e.g., Stein, 2008, Nychka et al., 2015]. The main aim of the package **FRK** is to facilitate spatial and spatio-temporal analysis and prediction for large datasets, where multiple observatons come with different spatial supports. We see that in 'big data' scenarios, lack of consideration of fine-scale variation may lead to over-confident predictions, irrespective of the number of basis functions adopted.

In this vignette we illustrate how to use **FRK** on spatial and spatio-temporal datasets with differing supports and on different manifolds. In Section 2 we first present the model, the estimation approach and the prediction equations. In Sections 3 and 4 we consider examples of spatial and spatio-temporal data, respectively. In Section 5 we discuss some additional functionality (e.g., modelling of anisotropic fields) and in Section 6 we discuss package limitations and opportunities for further development.

2 Outline of Fixed Rank Kriging: Modelling, estimation and prediction

In this section we present the theory behind the operations in **FRK**. In Section 2.1 we introduce the SRE model, in Section 2.2 we discuss the EM algorithm for parameter estimation, and in Section 2.3 we present the prediction equations.

2.1 The SRE model

Denote the spatial process of interest as $\{Y(\mathbf{s}) : \mathbf{s} \in D\}$, where $\mathbf{s}$ indexes the location of $Y(\mathbf{s})$ in our domain of interest D. In what follows, we assume that D is a spatial domain but extensions to spatio-temporal domains are natural within the framework (Section 4). Consider the classical spatial statistical model,

$$Y(\mathbf{s}) = \mathbf{t}(\mathbf{s})^\top \boldsymbol{\alpha} + \upsilon(\mathbf{s}) + \xi(\mathbf{s}); \quad \mathbf{s} \in D,$$

where, for $\mathbf{s} \in D$, $\mathbf{t}(\mathbf{s})$ is a vector of spatially referenced covariates, $\boldsymbol{\alpha}$ is a vector of regression coefficients, $\upsilon(\mathbf{s})$ is a small-scale, spatially correlated random effect, and $\xi(\mathbf{s})$ is a fine-scale random effect that is 'almost' spatially uncorrelated. It is natural to let $\mathrm{E}(\upsilon(\cdot)) = \mathrm{E}(\xi(\cdot)) = 0$. Define $\lambda(\cdot) \equiv \upsilon(\cdot)+\xi(\cdot)$, so that $\mathbb{E}(\lambda(\cdot)) = 0$. It is the structure of the process $\upsilon(\cdot)$ in terms of a linear combination of a fixed number of spatial basis functions that defines the SRE model for $\lambda(\cdot)$:

$$\lambda(\mathbf{s}) = \sum_{l=1}^{r} \phi_l(\mathbf{s})\eta_l + \xi(\mathbf{s}); \quad \mathbf{s} \in D,$$

where $\boldsymbol{\eta} \equiv (\eta_1, \ldots, \eta_r)^\top$ is an r-variate random vector, and $\boldsymbol{\phi}(\cdot) \equiv (\phi_1(\cdot), \ldots, \phi_r(\cdot))^\top$ is an r-dimensional vector of pre-specified spatial basis functions. Sometimes, $\boldsymbol{\phi}(\cdot)$ contains basis functions of multiple resolutions (e.g., wavelets), they may or may not be orthogonal, and they may or may not have compact support. The basis functions chosen should be able to adequately reconstruct realisations of $Y(\cdot)$; an empirical spectral-based approach that can ensure this is discussed in Zammit-Mangion et al. [2012].

In order to cater for different observation supports $\{B_j\}$ (defined below), it is convenient to assume a discretised domain of interest $D^G \equiv \{A_i \subset D : i = 1, \ldots, N\}$ that is made up of N small, non-overlapping basic areal units or BAUs [Nguyen et al., 2012], and $D = \bigcup_{i=1}^{N} A_i$. The set D^G of BAUs is a discretisation, or 'tiling,' of the original domain D, and typically $N \gg r$. The process $\{Y(\mathbf{s}) : \mathbf{s} \in D\}$ is then averaged over the BAUs, giving the vector $\mathbf{Y} = (Y_i : i = 1, \ldots, N)^\top$, where

$$Y_i \equiv \frac{1}{|A_i|} \int_{A_i} Y(\mathbf{s}) \mathrm{d}\mathbf{s}; \quad i = 1, \ldots, N, \tag{1}$$

and N is the number of BAUs. At this BAU level,

$$Y_i = \mathbf{t}_i^\top \boldsymbol{\alpha} + \upsilon_i + \xi_i, \tag{2}$$

where for $i = 1, \ldots, N$, $\mathbf{t}_i \equiv \frac{1}{|A_i|}\int_{A_i} \mathbf{t}(\mathbf{s})\mathrm{d}\mathbf{s}$, $\upsilon_i \equiv \frac{1}{|A_i|}\int_{A_i} \upsilon(\mathbf{s})\mathrm{d}\mathbf{s}$, and ξ_i is specified below. The SRE model specifies that the small-scale random variation is $\upsilon(\cdot) = \boldsymbol{\phi}(\cdot)^\top \boldsymbol{\eta}$, and hence in terms of the discretisation onto D^G,

$$\upsilon_i = \left(\frac{1}{|A_i|}\int_{A_i} \boldsymbol{\phi}(\mathbf{s})\mathrm{d}\mathbf{s}\right)^\top \boldsymbol{\eta}; \quad i = 1, \ldots, N,$$

so that $\boldsymbol{\upsilon} = \mathbf{S}\boldsymbol{\eta}$, where $\mathbf{S}$ is the $N \times r$ matrix defined as follows:

$$\mathbf{S} \equiv \left(\frac{1}{|A_i|}\int_{A_i} \boldsymbol{\phi}(\mathbf{s})\mathrm{d}\mathbf{s} : i = 1, \ldots, N\right)^\top. \tag{3}$$

In **FRK**, we assume that $\boldsymbol{\eta}$ is an r-dimensional Gaussian vector with mean zero and $r \times r$ covariance matrix $\mathbf{K}$, and estimation of $\mathbf{K}$ is based on likelihood methods; we denote this variant of FRK as FRK-V (where recall that 'V' stands for 'vanilla'). If some structure is imposed on $\mathrm{var}(\boldsymbol{\eta})$ in terms of parameters $\boldsymbol{\vartheta}$, then $\mathbf{K} = \mathbf{K}_\circ(\boldsymbol{\vartheta})$ and $\boldsymbol{\vartheta}$ needs to be estimated; we denote this variant as FRK-M (where recall that 'M' stands for 'model'). Frequently, the resolution of the BAUs is sufficiently fine, and the basis functions are sufficiently smooth, so that $\mathbf{S}$ can be approximated:

$$\mathbf{S} \approx (\boldsymbol{\phi}(\mathbf{s}_i) : i = 1, \ldots, N)^\top, \tag{4}$$

where $\{\mathbf{s}_i : i = 1, \ldots, N\}$ are the centroids of the BAUs. Since small BAUs are always assumed, this approximation is used throughout **FRK**.

In **FRK**, we do not directly model $\xi(\mathbf{s})$, since we are only interested in its discretised version. Rather, we assume that $\xi_i \equiv \frac{1}{|A_i|}\int_{A_i} \xi(\mathbf{s})\mathrm{d}\mathbf{s}$ has a Gaussian distribution with mean zero and variance

$$\mathrm{var}(\xi_i) = \sigma^2_\xi \nu_{\xi,i},$$

where σ^2_ξ is a parameter to be estimated, and the weights $\{\nu_{\xi,1}, \ldots, \nu_{\xi,N}\}$ are known and represent heteroscedasticity. These weights are typically generated from domain knowledge; they may, for example, correspond to topographical features such as terrain roughness [Zammit-Mangion et al., 2015]. Since we specified $\xi(\cdot)$ to be 'almost' spatially uncorrelated, it is reasonable to assume that the variables representing the discretised fine-scale variation, $\{\xi_i : i = 1, \ldots, N\}$, are uncorrelated. From (2), we can write

$$\mathbf{Y} = \mathbf{T}\boldsymbol{\alpha} + \mathbf{S}\boldsymbol{\eta} + \boldsymbol{\xi}, \tag{5}$$

where $\mathbf{T} \equiv (\mathbf{t}_i : i = 1, \ldots, N)^\top$, $\boldsymbol{\xi} \equiv (\xi_i : i = 1, \ldots, N)^\top$, and $\mathrm{var}(\boldsymbol{\xi}) \equiv \sigma^2_\xi \mathbf{V}_\xi$, for known $\mathbf{V}_\xi \equiv \mathrm{diag}(\nu_{\xi,1}, \ldots, \nu_{\xi,N})$.

We now assume that the hidden (or latent) process, $Y(\cdot)$, is observed with m footprints (possibly overlapping) spanning one or more BAUs, where typically $m \gg r$ (note that both $m > N$ and $N \geq m$ are possible). We thus define the observation domain as $D^O \equiv \{\cup_{i \in c_j} A_i : j = 1, \ldots, m\}$, where c_j is a non-empty set in $2^{\{1,\ldots,N\}}$, the power set of $\{1, \ldots, N\}$, and $m = |D^O|$. For illustration, consider the simple case of the discretised domain being made up of three BAUs. Then $D^G = \{A_1, A_2, A_3\}$ and, for example, $D^O = \{B_1, B_2\}$, where $B_1 = A_1 \cup A_2$ (i.e., $c_1 = \{1, 2\}$) and $B_2 = A_3$ (i.e., $c_2 = \{3\}$). Catering for different footprints is important for remote sensing applications in which satellite-instrument footprints can widely differ [e.g., Zammit-Mangion et al., 2015].

Each $B_j \in D^O$ is either a BAU or a union of BAUs. Measurement of $\mathbf{Y}$ is imperfect: We define the measurement process as noisy measurements of the process averaged over the footprints

$$Z_j \equiv Z(B_j) = \left(\frac{\sum_{i=1}^N Y_i w_{ij}}{\sum_{i=1}^N w_{ij}}\right) + \left(\frac{\sum_{i=1}^N \delta_i w_{ij}}{\sum_{i=1}^N w_{ij}}\right) + \epsilon_j; \quad B_j \in D^O, \tag{6}$$

where the weights,

$$w_{ij} = |A_i|\mathbb{I}(A_i \subset B_j); \quad i = 1, \ldots, N; \;\; j = 1, \ldots, m; \;\; B_j \in D^O,$$

depend on the areas of the BAUs, and $\mathbb{I}(\cdot)$ is the indicator function. Currently, in **FRK**, BAUs of equal area are assumed, but we give (6) in its most general form. The random quantities $\{\delta_i\}$ and $\{\epsilon_i\}$ capture the imperfections of the measurement. Better known is the measurement-error component ϵ_i, which is assumed to be mean-zero Gaussian distributed. The component δ_i captures any bias in the measurement at the BAU level, which has the interpretation of an intra-BAU systematic error. These systematic errors are BAU-specific, that is, the $\{\delta_i\}$ are uncorrelated with mean zero and variance

$$\text{var}(\delta_i) = \sigma^2_\delta v_{\delta,i},$$

where σ^2_δ is a parameter to be estimated, and $\{v_{\delta,1}, \ldots, v_{\delta,N}\}$ represent known heteroscedasticity.

We assume that $\mathbf{Y}$ and $\boldsymbol{\delta}$ are independent. We also assume that the observations are conditionally independent, when conditioned on $\mathbf{Y}$ and $\boldsymbol{\delta}$. Equivalently, we assume that the measurement errors $\{\epsilon_j : j = 1, \ldots, m\}$ are independent with $\text{var}(\epsilon_i) = \sigma^2_\epsilon v_{\epsilon,i}$.

We represent the data as $\mathbf{Z} \equiv (Z_j : j = 1, \ldots, m)^\top$. Then, since each element in D^O is the union of subsets of D^G, one can construct a matrix

$$\mathbf{C}_Z \equiv \left(\frac{w_{ij}}{\sum_{l=1}^N w_{lj}} : i = 1, \ldots, N; j = 1, \ldots, m \right),$$

such that

$$\mathbf{Z} = \mathbf{C}_Z \mathbf{Y} + \mathbf{C}_Z \boldsymbol{\delta} + \boldsymbol{\varepsilon},$$

where the three components are independent, $\boldsymbol{\varepsilon} \equiv (\epsilon_j : j = 1, \ldots, m)^\top$, and $\text{var}(\boldsymbol{\varepsilon}) = \boldsymbol{\Sigma}_\epsilon \equiv \sigma^2_\epsilon \mathbf{V}_\epsilon \equiv \sigma^2_\epsilon \text{diag}(v_{\epsilon,1}, \ldots, v_{\epsilon,m})$ is an $m \times m$ diagonal covariance matrix. The matrix $\boldsymbol{\Sigma}_\epsilon$ is assumed known from the properties of the measurement. If it is not known, $\mathbf{V}_\epsilon$ is fixed to $\mathbf{I}$ and σ^2_ϵ is estimated using variogram techniques [Kang et al., 2009]. Notice that the rows of the matrix $\mathbf{C}_Z$ sum to 1.

It will be convenient to re-write

$$\mathbf{Z} = \mathbf{T}_Z \boldsymbol{\alpha} + \mathbf{S}_Z \boldsymbol{\eta} + \boldsymbol{\xi}_Z + \boldsymbol{\delta}_Z + \boldsymbol{\varepsilon}, \tag{7}$$

where $\mathbf{T}_Z \equiv \mathbf{C}_Z \mathbf{T}$, $\mathbf{S}_Z \equiv \mathbf{C}_Z \mathbf{S}$, $\boldsymbol{\xi}_Z \equiv \mathbf{C}_Z \boldsymbol{\xi}$, $\boldsymbol{\delta}_Z \equiv \mathbf{C}_Z \boldsymbol{\delta}$, $\text{var}(\boldsymbol{\xi}_Z) = \sigma^2_\xi \mathbf{V}_{\xi,Z} \equiv \sigma^2_\xi \mathbf{C}_Z \mathbf{V}_\xi \mathbf{C}_Z^\top$, $\text{var}(\boldsymbol{\delta}_Z) = \sigma^2_\delta \mathbf{V}_{\delta,Z} \equiv \sigma^2_\delta \mathbf{C}_Z \mathbf{V}_\delta \mathbf{C}_Z^\top$, and where $\mathbf{V}_\delta \equiv \text{diag}(v_{\delta,1}, \ldots, v_{\delta,N})$ is known. Then, recalling that $\mathbb{E}(\boldsymbol{\eta}) = \mathbf{0}$ and $\mathbb{E}(\boldsymbol{\xi}_Z) = \mathbb{E}(\boldsymbol{\delta}_Z) = \mathbb{E}(\boldsymbol{\varepsilon}) = \mathbf{0}$,

$$\begin{aligned}
\mathbb{E}(\mathbf{Z}) &= \mathbf{T}_Z \boldsymbol{\alpha}, \\
\text{var}(\mathbf{Z}) &= \mathbf{S}_Z \mathbf{K} \mathbf{S}_Z^\top + \sigma^2_\xi \mathbf{C}_Z \mathbf{V}_\xi \mathbf{C}_Z^\top + \sigma^2_\delta \mathbf{C}_Z \mathbf{V}_\delta \mathbf{C}_Z^\top + \sigma^2_\epsilon \mathbf{V}_\epsilon.
\end{aligned}$$

In practice, it is not always possible for each B_j to include entire BAUs. For simplicity, in **FRK** we assume that the observation footprint overlaps a BAU if and only if the BAU centroid lies within the footprint. Frequently, point-referenced data is included in $\mathbf{Z}$. In this case, each data point is attributed to a specific BAU and it is possible to have multiple observations of the process defined on the same BAU.

We collect the unknown parameters in the set $\boldsymbol{\theta} \equiv \{\boldsymbol{\alpha}, \sigma^2_\xi, \sigma^2_\delta, \mathbf{K}\}$ for FRK-V and $\boldsymbol{\theta}_\circ \equiv \{\boldsymbol{\alpha}, \sigma^2_\xi, \sigma^2_\delta, \boldsymbol{\vartheta}\}$ for FRK-M for which $\mathbf{K} = \mathbf{K}_\circ(\boldsymbol{\vartheta})$; their estimation is the subject of Section 2.2. If the parameters in $\boldsymbol{\theta}$ or $\boldsymbol{\theta}_\circ$ are known, an inversion that uses the Sherman–Woodbury identity [Henderson and Searle, 1981] allows spatial prediction at any BAU in D^G. Estimates of $\boldsymbol{\theta}$ are substituted into these spatial predictors to yield FRK-V. Similarly, estimates of $\boldsymbol{\theta}_\circ$ substituted into the spatial-prediction equations yield FRK-M.

In **FRK**, we allow the prediction set D^P to be as flexible as D^O; specifically, $D^P \subset \{\cup_{i \in \tilde{c}_k} A_i : k = 1, \ldots, N_P\}$, where $\tilde{c}_k$ is a non-empty set in $2^{\{1,\ldots,N\}}$ and N_P is the number of prediction areas. We can thus predict both at the individual BAU level or averages over an area spanning multiple BAUs, and these prediction regions may overlap. This is an important change-of-support feature of **FRK**. We provide the FRK equations in Section 2.3.

2.2 Parameter estimation using an EM algorithm

In all its generality, parameter estimation with the model of Section 2.1 is problematic due to confounding between $\boldsymbol{\delta}$ and $\boldsymbol{\xi}$. In **FRK**, the user thus needs to choose between modelling the intra-BAU systematic errors

(in which case σ_ξ^2 is fixed to 0) or the process' fine-scale variation (in which case σ_δ^2 is fixed to 0). We describe below the estimation procedure for the latter case; due to symmetry, the estimation equations of the former case can be simply obtained by replacing the subscript ξ with δ. However, which case is chosen by the user has a considerable impact on the prediction equations for Y (Section 2.3). Recall that the measurement-error covariance matrix $\mathbf{\Sigma}_\epsilon$ is assumed known from measurement characteristics, or estimated using variogram techniques prior to estimating the remaining parameters described below. For conciseness, in this section we use $\boldsymbol{\theta}$ to denote the parameters in both FRK-V and FRK-M, only distinguishing when necessary.

We carry out parameter estimation using an expectation maximisation (EM) algorithm [similar to Katzfuss and Cressie, 2011, Nguyen et al., 2014] with (7) as our model. Define the complete-data likelihood $L_c(\boldsymbol{\theta}) \equiv [\boldsymbol{\eta}, \mathbf{Z} \mid \boldsymbol{\theta}]$ (with $\boldsymbol{\xi}_Z$ integrated out), where $[\ \cdot\ ]$ denotes the probability distribution of its argument. The EM algorithm proceeds by first computing the conditional expectation (conditional on the data) of the complete-data log-likelihood at the current parameter estimates (the E-step) and, second, maximising this function with respect to the parameters (the M-step). In mathematical notation, in the E-step the function

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(l)}) \equiv \mathbb{E}(\ln L_c(\boldsymbol{\theta}) \mid \mathbf{Z}, \boldsymbol{\theta}^{(l)}),$$

is found for some current estimate $\boldsymbol{\theta}^{(l)}$. In the M-step, the updated parameter estimates

$$\boldsymbol{\theta}^{(l+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(l)}),$$

are found.

The E-step boils down to finding the conditional distribution of $\boldsymbol{\eta}$ at the current parameter estimates. One can use standard results in Gaussian conditioning [e.g., Rasmussen and Williams, 2006, Appendix A] to show from the joint distribution, $[\boldsymbol{\eta}, \mathbf{Z} \mid \boldsymbol{\theta}^{(l)}]$, that

$$\boldsymbol{\eta} \mid \mathbf{Z}, \boldsymbol{\theta}^{(l)} \sim \text{Gau}(\boldsymbol{\mu}_\eta^{(l)}, \mathbf{\Sigma}_\eta^{(l)}),$$

where

$$\begin{aligned}
\boldsymbol{\mu}_\eta^{(l)} &= \mathbf{\Sigma}_\eta^{(l)} \mathbf{S}_Z^\top \left(\mathbf{D}_Z^{(l)}\right)^{-1} \left(\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l)}\right), \\
\mathbf{\Sigma}_\eta^{(l)} &= \left(\mathbf{S}_Z^\top \left(\mathbf{D}_Z^{(l)}\right)^{-1} \mathbf{S}_Z + \left(\mathbf{K}^{(l)}\right)^{-1}\right)^{-1},
\end{aligned}$$

where $\mathbf{D}_Z^{(l)} \equiv (\sigma_\xi^2)^{(l)} \mathbf{V}_{\xi,Z} + \mathbf{\Sigma}_\epsilon$, and where $\mathbf{K}^{(l)}$ is defined below.

The update for $\boldsymbol{\alpha}$ is

$$\boldsymbol{\alpha}^{(l+1)} = \left(\mathbf{T}_Z^\top \left(\mathbf{D}_Z^{(l+1)}\right)^{-1} \mathbf{T}_Z\right)^{-1} \mathbf{T}_Z^\top \left(\mathbf{D}_Z^{(l+1)}\right)^{-1} \left(\mathbf{Z} - \mathbf{S}_Z \boldsymbol{\mu}_\eta^{(l)}\right). \tag{8}$$

In FRK-V, the update for $\mathbf{K}^{(l+1)}$ is

$$\mathbf{K}^{(l+1)} = \mathbf{\Sigma}_\eta^{(l)} + \boldsymbol{\mu}_\eta^{(l)} \boldsymbol{\mu}_\eta^{(l)^\top},$$

while in FRK-M, where recall that $\mathbf{K} = \mathbf{K}_\circ(\boldsymbol{\vartheta})$, the update is

$$\boldsymbol{\vartheta}^{(l+1)} = \arg\max_{\boldsymbol{\vartheta}} \ln \left|\mathbf{K}_\circ(\boldsymbol{\vartheta})^{-1}\right| - \text{tr}\left(\mathbf{K}_\circ(\boldsymbol{\vartheta})^{-1} \left(\mathbf{\Sigma}_\eta^{(l)} + \boldsymbol{\mu}_\eta^{(l)} \boldsymbol{\mu}_\eta^{(l)^\top}\right)\right),$$

which is numerically optimised using the function `optim` with $\boldsymbol{\vartheta}^{(l)}$ as the initial vector.

The update for σ_ξ^2 requires the solution to

$$\text{tr}((\mathbf{\Sigma}_\epsilon + (\sigma_\xi^2)^{(l+1)} \mathbf{V}_{\xi,Z})^{-1} \mathbf{V}_{\xi,Z}) = \text{tr}((\mathbf{\Sigma}_\epsilon + (\sigma_\xi^2)^{(l+1)} \mathbf{V}_{\xi,Z})^{-1} \mathbf{V}_{\xi,Z} (\mathbf{\Sigma}_\epsilon + (\sigma_\xi^2)^{(l+1)} \mathbf{V}_{\xi,Z})^{-1} \mathbf{\Omega}), \tag{9}$$

where

$$\mathbf{\Omega} \equiv \mathbf{S}_Z \mathbf{\Sigma}_\eta^{(l)} \mathbf{S}_Z^\top + \mathbf{S}_Z \boldsymbol{\mu}_\eta^{(l)} \boldsymbol{\mu}_\eta^{(l)^\top} \mathbf{S}_Z^\top - 2\mathbf{S}_Z \boldsymbol{\mu}_\eta^{(l)} (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l+1)})^\top + (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l+1)})(\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l+1)})^\top. \tag{10}$$

The solution to (9), namely $(\sigma_\xi^2)^{(l+1)}$, is found numerically using `uniroot` after (8) is substituted into (10). Then $\boldsymbol{\alpha}^{(l+1)}$ is found by substituting $(\sigma_\xi^2)^{(l+1)}$ into (8). Computational simplifications are possible when $\mathbf{V}_{\xi,Z}$ and $\boldsymbol{\Sigma}_\epsilon$ are diagonal, since then only the diagonal of $\boldsymbol{\Omega}$ needs to be computed. Further simplifications are possible when $\mathbf{V}_{\xi,Z}$ and $\boldsymbol{\Sigma}_\epsilon$ are proportional to the identity matrix, with constants of proportionality γ_1 and γ_2, respectively. In this case,

$$(\sigma_\xi^2)^{(l+1)} = \frac{1}{\gamma_1}\left(\frac{\text{tr}(\boldsymbol{\Omega})}{m} - \gamma_2\right),$$

where recall that m is the dimension of the data vector $\mathbf{Z}$ and $\boldsymbol{\alpha}^{(l+1)}$ is, in this special case, the ordinary-least-squares estimate given $\boldsymbol{\mu}_\eta^{(l)}$ (see (8)). These simplifications are used by **FRK** whenever possible.

Convergence of the EM algorithm is assessed using the (incomplete-data) log-likelihood function at each iteration,

$$\ln\left[\mathbf{Z} \mid \boldsymbol{\alpha}^{(l)}, \mathbf{K}^{(l)}, (\sigma_\xi^2)^{(l)}\right] = -\frac{m}{2}\ln 2\pi - \frac{1}{2}\ln\left|\boldsymbol{\Sigma}_Z^{(l)}\right| - \frac{1}{2}(\mathbf{Z} - \mathbf{T}_Z\boldsymbol{\alpha}^{(l)})^\top(\boldsymbol{\Sigma}_Z^{(l)})^{-1}(\mathbf{Z} - \mathbf{T}_Z\boldsymbol{\alpha}^{(l)}),$$

where

$$\boldsymbol{\Sigma}_Z^{(l)} = \mathbf{S}_Z\mathbf{K}^{(l)}\mathbf{S}_Z^\top + \mathbf{D}_Z^{(l)},$$

and recall that $\mathbf{D}_Z^{(l)} \equiv (\sigma_\xi^2)^{(l)}\mathbf{V}_{\xi,Z} + \boldsymbol{\Sigma}_\epsilon$. Efficient computation of the log-likelihood is facilitated through the use of the Sherman–Morrison–Woodbury matrix identity and a matrix-determinant lemma [e.g., Henderson and Searle, 1981]. Specifically, the operations

$$\left(\boldsymbol{\Sigma}_Z^{(l)}\right)^{-1} = \left(\mathbf{D}_Z^{(l)}\right)^{-1} - \left(\mathbf{D}_Z^{(l)}\right)^{-1}\mathbf{S}_Z\left[\left(\mathbf{K}^{(l)}\right)^{-1} + \mathbf{S}_Z^\top\left(\mathbf{D}_Z^{(l)}\right)^{-1}\mathbf{S}_Z\right]^{-1}\mathbf{S}_Z^\top\left(\mathbf{D}_Z^{(l)}\right)^{-1},$$

$$\left|\boldsymbol{\Sigma}_Z^{(l)}\right| = \left|\left(\mathbf{K}^{(l)}\right)^{-1} + \mathbf{S}_Z^\top\left(\mathbf{D}_Z^{(l)}\right)^{-1}\mathbf{S}_Z\right|\left|\mathbf{K}^{(l)}\right|\left|\mathbf{D}_Z^{(l)}\right|,$$

ensure that we only deal with vectors of length m and matrices of size $r \times r$, where typically the fixed rank $r \ll m$, the dataset size.

2.3 Prediction

The prediction task is to make inference on the hidden Y-process over a set of prediction regions D^P. Consider the process $\{Y_P(\tilde{B}_k) : k = 1, \ldots, N_P\}$, which is derived from the Y process and, similar to the observations, is constructed using the BAUs $\{A_i : i = 1, \ldots, N\}$. Here, N_P is the number of areas at which spatial prediction takes place, and is equal to $|D^P|$. Then,

$$Y_{P,k} \equiv Y_P(\tilde{B}_k) = \left(\frac{\sum_{i=1}^N Y_i\tilde{w}_{ik}}{\sum_{i=1}^N \tilde{w}_{ik}}\right); \quad \tilde{B}_k \in D^P,$$

where the weights are

$$\tilde{w}_{ik} = |A_i|\mathbb{I}(A_i \subset \tilde{B}_k); \quad i = 1, \ldots, N; \quad k = 1, \ldots, N_P; \quad \tilde{B}_k \in D^P.$$

Define $\mathbf{Y}_P \equiv (Y_{P,k} : k = 1, \ldots, N_P)^\top$. Then, since each element in D^P is the union of subsets of D^G, one can construct a matrix,

$$\mathbf{C}_P \equiv \left(\frac{\tilde{w}_{ik}}{\sum_{l=1}^N \tilde{w}_{lk}} : i = 1, \ldots, N; k = 1, \ldots, N_P\right), \tag{11}$$

the rows of which sum to 1, such that

$$\mathbf{Y}_P = \mathbf{C}_P\mathbf{Y} = \mathbf{T}_P\boldsymbol{\alpha} + \mathbf{S}_P\boldsymbol{\eta} + \boldsymbol{\xi}_P,$$

where $\mathbf{T}_P \equiv \mathbf{C}_P\mathbf{T}$, $\mathbf{S}_P \equiv \mathbf{C}_P\mathbf{S}$, $\boldsymbol{\xi}_P \equiv \mathbf{C}_P\boldsymbol{\xi}$ and $\text{var}(\boldsymbol{\xi}_P) = \sigma_\xi^2\mathbf{V}_{\xi,P} \equiv \sigma_\xi^2\mathbf{C}_P\mathbf{V}_\xi\mathbf{C}_P^\top$. As with the observations, the prediction regions $\{\tilde{B}_k\}$ may overlap. In practice, it may not always be possible for each $\tilde{B}_k$ to include

entire BAUs. In this case, we assume that a prediction region contains a BAU if and only if the BAU centroid lies within the region.

Let l^* denote the EM iteration number at which convergence is deemed to have been reached. The final estimates are then

$$\widehat{\boldsymbol{\mu}}_\eta \equiv \boldsymbol{\mu}_\eta^{(l^*)}, \quad \widehat{\boldsymbol{\Sigma}}_\eta \equiv \boldsymbol{\Sigma}_\eta^{(l^*)}, \quad \widehat{\boldsymbol{\alpha}} \equiv \boldsymbol{\alpha}^{(l^*)}, \quad \widehat{\mathbf{K}} \equiv \mathbf{K}^{(l^*)}, \quad \widehat{\sigma}_\xi^2 \equiv (\sigma_\xi^2)^{(l^*)}, \quad \text{and} \quad \widehat{\sigma}_\delta^2 \equiv (\sigma_\delta^2)^{(l^*)}.$$

Recall from Section 2.2 that the user needs to attribute fine-scale variation at the BAU level to either the measurement process or the hidden process Y. This leads to the following two cases.

Case 1: $\sigma_\xi^2 = 0$ and estimate σ_δ^2. The prediction vector $\widehat{\mathbf{Y}}_P$ and covariance matrix $\boldsymbol{\Sigma}_{Y_P|Z}$, corresponding to the first two moments from the predictive distribution $[\mathbf{Y}_P \mid \mathbf{Z}]$ when $\sigma_\xi^2 = 0$, are

$$\begin{aligned}\widehat{\mathbf{Y}}_P &\equiv \mathbb{E}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{T}_P\widehat{\boldsymbol{\alpha}} + \mathbf{S}_P\widehat{\boldsymbol{\mu}}_\eta,\\ \boldsymbol{\Sigma}_{Y_P|Z} &\equiv \mathrm{var}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{S}_P\widehat{\boldsymbol{\Sigma}}_\eta\mathbf{S}_P^\top.\end{aligned}$$

Under the assumptions taken, $[\mathbf{Y}_P \mid \mathbf{Z}]$ is a $\mathrm{Gau}(\widehat{\mathbf{Y}}_P, \boldsymbol{\Sigma}_{Y_P|Z})$ distribution. Note that all calculations are made after substituting in the EM-estimated parameters, and that $\widehat{\sigma}_\delta^2$ is present in the estimated parameters.

Case 2: $\sigma_\delta^2 = 0$ and estimate σ_ξ^2 (Default). To cater for arbitrary observation and prediction support, we predict $\mathbf{Y}_P$ by first carrying out prediction over the full vector $\mathbf{Y}$, that is, at the BAU level, and then transforming linearly to obtain $\widehat{\mathbf{Y}}_P$ through the use of the matrix $\mathbf{C}_P$. It is easy to see that if $\widehat{\mathbf{Y}}$ is an optimal (squared-error-loss matrix criterion) predictor of $\mathbf{Y}$, then $\mathbf{A}\widehat{\mathbf{Y}}$ is an optimal predictor of $\mathbf{AY}$, where $\mathbf{A}$ is any matrix with N columns.

Let $\mathbf{W} \equiv (\boldsymbol{\eta}^\top, \boldsymbol{\xi}^\top)^\top$ and $\boldsymbol{\Pi} \equiv (\mathbf{S}, \mathbf{I})$. Then (5) can be re-written as $\mathbf{Y} = \mathbf{T}\boldsymbol{\alpha} + \boldsymbol{\Pi}\mathbf{W}$, and

$$\begin{aligned}\widehat{\mathbf{Y}} &\equiv \mathbb{E}(\mathbf{Y} \mid \mathbf{Z}) = \mathbf{T}\widehat{\boldsymbol{\alpha}} + \boldsymbol{\Pi}\widehat{\mathbf{W}},\\ \boldsymbol{\Sigma}_{Y|Z} &\equiv \mathrm{var}(\mathbf{Y} \mid \mathbf{Z}) = \boldsymbol{\Pi}\boldsymbol{\Sigma}_W\boldsymbol{\Pi}^\top,\end{aligned} \tag{12}$$

for

$$\begin{aligned}\widehat{\mathbf{W}} &\equiv \boldsymbol{\Sigma}_W\boldsymbol{\Pi}^\top\mathbf{C}_Z^\top\boldsymbol{\Sigma}_\epsilon^{-1}(\mathbf{Z} - \mathbf{T}_Z\widehat{\boldsymbol{\alpha}}),\\ \boldsymbol{\Sigma}_W &\equiv \left(\boldsymbol{\Pi}^\top\mathbf{C}_Z^\top\boldsymbol{\Sigma}_\epsilon^{-1}\mathbf{C}_Z\boldsymbol{\Pi} + \boldsymbol{\Lambda}^{-1}\right)^{-1},\end{aligned}$$

and the block-diagonal matrix $\boldsymbol{\Lambda} \equiv \mathrm{bdiag}(\widehat{\mathbf{K}}, \widehat{\sigma}_\xi^2\mathbf{V}_\xi)$, where $\mathrm{bdiag}(\cdot)$ returns a block diagonal matrix of its matrix arguments. Note that all calculations are made after substituting in the EM-estimated parameters.

For both Cases 1 and 2 it follows that $\widehat{\mathbf{Y}}_P = \mathbb{E}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{C}_P\widehat{\mathbf{Y}}$ and

$$\boldsymbol{\Sigma}_{Y_P|Z} = \mathrm{var}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{C}_P\boldsymbol{\Sigma}_{Y|Z}\mathbf{C}_P^\top. \tag{13}$$

Note that for Case 2 we need to obtain predictions for $\boldsymbol{\xi}_P$ which, unlike those for $\boldsymbol{\eta}$, are not a by-product of the EM algorithm of 2.2. Sparse-matrix operations are used to facilitate the computation of (13) when possible.

3 Fixed Rank Kriging on $\mathbb{R}^2$ or $\mathbb{S}^2$

In this part of the vignette we apply **FRK** to the case when we have spatial data, either on the plane or on the surface of a sphere. For 2D data on the plane, we consider the `meuse` data, which can be found in the package **sp**. For data on the sphere we will use readings taken between May 01 2003 and May 03 2003 (inclusive) by the Atmospheric InfraRed Sounder (AIRS) on board the Aqua satellite [e.g., Chahine et al., 2006]. For spatial modelling of the data we need to load the following packages

```
library(sp)         # for defining points/polygons
library(ggplot2)    # for plotting
library(dplyr)      # for easy data manipulation
library(FRK)        # for carrying out FRK
```

and, to keep the document tidy, we will set the `progress` package option to `FALSE`. Parallelisation is frequently used in **FRK**, but for the purposes of this document we will set the `parallel` option to 0 as well.

```
opts_FRK$set("progress",FALSE)  # no progress bars
opts_FRK$set("parallel",0L)     # no parallelisation
```

In this vignette we go through the 'expert' way of using **FRK**. There is also a simple way through the command `FRK` which serves as a wrapper for, and masks, several of the steps below; see `help(FRK)` for details. Usage of `FRK` is only recommended once the steps below are understood.

3.1 The `meuse` dataset

The `meuse` dataset contains readings of heavy-metal abundance in a region of The Netherlands along the river Meuse. For more details on the dataset see the vignette titled 'gstat' in the package **gstat**. The aim of this vignette is to analyse the spatial distribution of zinc-concentration from spatially sparse readings using FRK.

Step 1: We first load the `meuse` data:

```
data(meuse)             # load meuse data
print(class(meuse))     # print class of meuse data

## [1] "data.frame"
```

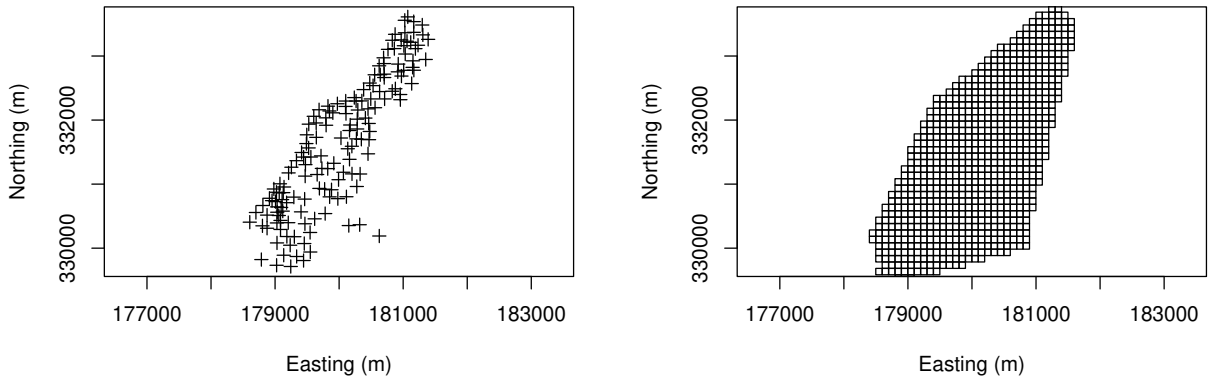
The `meuse` data is of class `data.frame`. However, **FRK** needs all spatial objects to be of class `SpatialPointsDataFrame` or `SpatialPolygonsDataFrame`, depending on whether the dataset is point-referenced of area-referenced. The `meuse` data is point referenced, and we therefore cast it into a `SpatialPointsDataFrame` by applying the `coordinates` function as follows:

```
coordinates(meuse) = ~x+y      # change into an sp object
```

Step 2: Based on the data we now generate BAUs. For this, we can use the helper function `auto_BAUs`:

```
set.seed(1)
GridBAUs1 <- auto_BAUs(manifold = plane(),     # 2D plane
                       cellsize = c(100,100),  # BAU cellsize
                       type = "grid",          # grid (not hex)
                       data = meuse,           # data around which to create BAUs
                       convex=-0.05,           # border buffer factor
                       nonconvex_hull=FALSE)   # convex hull
```

The `auto_BAUs` function takes several arguments (see `help(auto_BAUs)` for details). Above, we instruct the helper function to construct BAUs on the plane, centred around the data `meuse` with each BAU of size 100×100 (with units in m since the data is supplied with x-y coordinates in m). The `type="grid"` input instructs that we want a rectangular grid and not a hexagonal lattice (use `"hex"` for a hexagonal lattice), and `convex=-0.05` is a specific parameter controlling the buffer-width of the spatial-domain boundary. The name 'convex' was chosen as it is also used to control the buffer in case a non-convex hull is desired by setting `nonconvex_hull=TRUE` (see `INLA::inla.nonconvex.hull` for more details and note that **INLA** needs to be installed for this option to be set). For the ith BAU, we also need to attribute the element v_i that describes



(a) (b)

Figure 1: (a) Locations of the `meuse` data. (b) BAUs for Fixed Rank Kriging with the `meuse` dataset.

the hetereoscedascity of the fine-scale variation for that BAU. As described in Section 2.1, this component encompasses all process variation that occurs at the BAU scale and only needs to be known up to a constant of proportionality, σ^2_ξ or σ^2_δ (depending on the chosen model); this constant is estimated using maximum likelihood with `SRE.fit` using the EM algorithm of Section 2.2. Typically, geographic features such as altitude are appropriate, but in this case we will just set this parameter to unity. It is important that this field is labelled 'fs':

```
GridBAUs1$fs <- 1   # fine-scale variation at BAU level
```

The data and BAUs are illustrated using the `plot` function in Fig. 1.

Step 3: FRK decomposes the spatial process as a sum of basis functions that may either be user-specified (see Section 5.3) or constructed using helper functions. To create spatial basis functions we use the helper function `auto_basis` as follows:

```
G <- auto_basis(manifold = plane(),   # 2D plane
                data = meuse,         # meuse data
                nres = 2,             # number of resolutions
                type = "Gaussian",    # type of basis function
                regular = 1)          # place regularly in domain
```

The argument `nres = 3` indicates how many resolutions we wish, while `type = "Gaussian"` indicates that the basis set we want is composed of Gaussian functions. Other built-in functions that can be used are `"exp"` (the exponential covariance function), `"bisquare"` (the bisquare function), and `"Matern32"` (the Matérn covariance function with smoothness parameter equal to 1.5). The argument `regular` indicates that we want to place the basis functions regularly in the domain. Usually better results can be achieved by placing them irregularly in the domain. For this functionality the mesher in the package **INLA** is used and thus **INLA** needs to be installed when `regular = 0`. The basis can be visualised using `show_basis`, see Fig. 2.

```
show_basis(G) +              # illustrate basis functions
    coord_fixed() +          # fix aspect ratio
    xlab("Easting (m)") +    # x-label
    ylab("Northing (m)")     # y-label
```

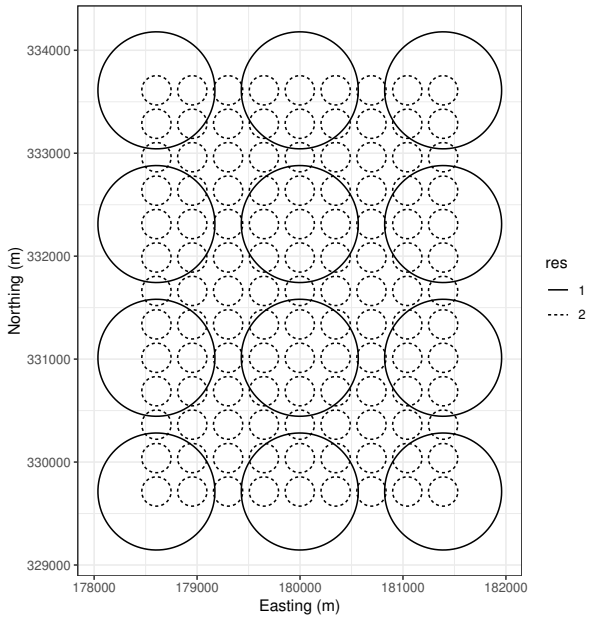


Figure 2: Basis functions automatically generated for the meuse dataset with 2 resolutions. The interpretation of the circles change with the domain and basis. For Gaussian functions on the plane, each circle is centred at the basis function centre, and has a radius equal to 1σ. Type `help(auto_basis)` for details.

Step 4: With the BAUs and the basis functions specified, we can construct the SRE model. For fixed effects, we just use an intercept; if we wish to use covariates, one must make sure that they are also specified at the BAU level (and hence attributed to `GridBAUs1`). The fixed effects are supplied in a usual R formula, which we store in `f`:

```
f <- log(zinc) ~ 1    # formula for SRE model
```

The Spatial Random Effects model is then constructed using the function `SRE`, which essentially bins the data in the BAUs, constructs all the matrices required for estimation, and provides initial guesses for the quantities that need to be estimated.

```
S <- SRE(f = f,                    # formula
         data = list(meuse),       # list of datasets
         BAUs = GridBAUs1,         # BAUs
         basis = G,                # basis functions
         est_error = TRUE,         # estimation measurement error
         average_in_BAU = FALSE)  # do not average data over BAUs
```

The function `SRE` takes as arguments the formula; the data (as a list that can include additional datasets); the BAUs; the basis functions; a flag; `est_error`; and a flag `average_in_BAU`. The flag `est_error` indicates whether we wish to attempt to estimate the measurement-error variance $\mathbf{\Sigma}_\epsilon \equiv \sigma^2_\epsilon \mathbf{I}$ or not using variogram methods [Kang et al., 2009]. Currently, `est_error = TRUE` is only allowed with spatial data. When not set, the dataset needs to also contain a field `std`, the standard deviation of the measurement error.

In practice, several datasets (such as the `meuse` dataset) are point-referenced. Since **FRK** is built on the concept of a Basic Areal Unit, the smallest footprint of an observation has to be equal to that of a BAU. If multiple point-referenced observations fall within the same BAU, then these are assumed to be readings of the same random variable (hence, the fine-scale variation is not a nugget in the classical sense). When multiple data points can fall into the same BAU, the matrix $\mathbf{V}_Z$ is not diagonal; this increases computational time considerably. For large point-referenced datasets, such as the `AIRS` dataset considered in Section 3.2,

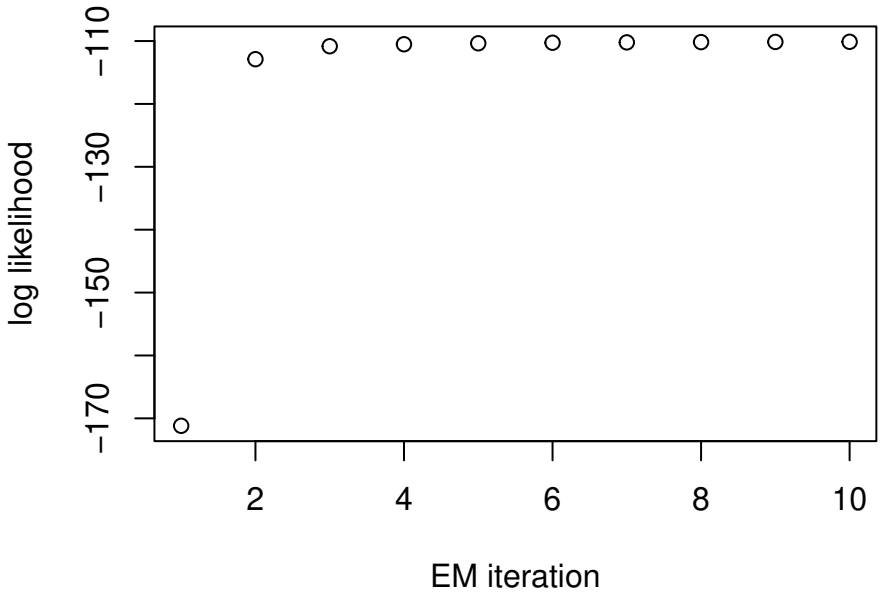


Figure 3: Convergence of the EM algorithm when using `FRK` with the `meuse` dataset.

one can use the argument `average_in_BAU = TRUE` to indicate that one wishes to summarise the data at the BAU level. When this flag is set, all data falling into one BAU is averaged; the measurement error of the averaged data point is then taken to be the average measurement error of the individual data points (i.e., measurement error is assumed to be perfectly correlated within the BAU). Consequently, the dataset is thinned; this can be used to obtain quick results prior to a more detailed analysis.

Step 5: The SRE model is fitted using the function `SRE.fit`. Maximum likelihood is carried out using the EM algorithm of Section 2.2, which is assumed to have converged either when `n_EM` is exceeded, or when the likelihood across subsequent steps does not change by more than `tol`. In this example, the EM algorithm would converge in 30 iterations but we limit the maximum number of iterations to 10 to minimise compilation-time of this vignette; see Fig. 3.

```
S <- SRE.fit(S,                 # SRE model
             n_EM = 10,         # max. no. of EM iterations
             tol = 0.01,        # tolerance at which EM is assumed to have converged
             print_lik=TRUE)    # print log-likelihood at each iteration
```

Step 6: Finally, we predict at all the BAUs with the fitted model. This is done using the function `predict`. The argument `obs_fs` dictates whether we attribute the fine-scale variation to the process model or the observation model (in which case it takes the role of systematicerror). Below, we allocate it to the process model.

```
GridBAUs1 <- predict(S, obs_fs = FALSE)
```

The object `GridBAUs1` now contains the prediction vector and the square of the prediction standard error at the BAU level in the fields `mu` and `var`, respectively. These can be plotted using the standard plotting

commands, such as those in **sp** or **ggplot2**. To use the latter, we first need to convert the `Spatial` object to a data frame as follows:

```
BAUs_df <- as(GridBAUs1,"data.frame")
```

The function `SpatialPolygonsDataFrame_to_df` takes as argument the BAUs and the variables we wish to extract from the BAUs. Now **ggplot2** can be used to plot the observations, the predictions, and the standard errors; for example, the following code yields the plots in Fig. 4.

```
g1 <- ggplot() +                            # Use a plain theme
    geom_tile(data=BAUs_df ,                    # Draw BAUs
                aes(x,y,fill=mu),       # Colour <-> Mean
                colour="light grey") +          # Border is light grey
    scale_fill_distiller(palette="Spectral",    # Spectral palette
                         name="pred.") +        # legend name
    geom_point(data=data.frame(meuse),          # Plot data
               aes(x,y,fill=log(zinc)),         # Colour <-> log(zinc)
               colour="black",                  # point outer colour
               pch=21, size=3) +                # size of point
    coord_fixed() +                             # fix aspect ratio
    xlab("Easting (m)") + ylab("Northing (m)") + # axes labels
    theme_bw()


g2 <- ggplot() +                            # Similar to above but with s.e.
    geom_tile(data=BAUs_df,
                aes(x,y,fill=sqrt(var)),
                colour="light grey") +
    scale_fill_distiller(palette="BrBG",
                         name = "s.e.",
                         guide = guide_legend(title="se")) +
    coord_fixed() +
    xlab("Easting (m)") + ylab("Northing (m)") + theme_bw()
```

Now, assume that we wish to predict over regions encompassing several BAUs such that the matrix $\mathbf{C}_P$ containes multiple non-zeros per row. Then we need to set the `newdata` argument in the function `auto_BAUs`. First, we create this larger regionalisation as follows

```
Pred_regions <- auto_BAUs(manifold = plane(),       # model on the 2D plane
                          cellsize = c(600,600),    # choose a large grid size
                          type = "grid",            # use a grid (not hex)
                          data = meuse,             # the dataset on which to center cells
                          convex=-0.05,             # border buffer factor
                          nonconvex_hull=FALSE)     # convex hull
```

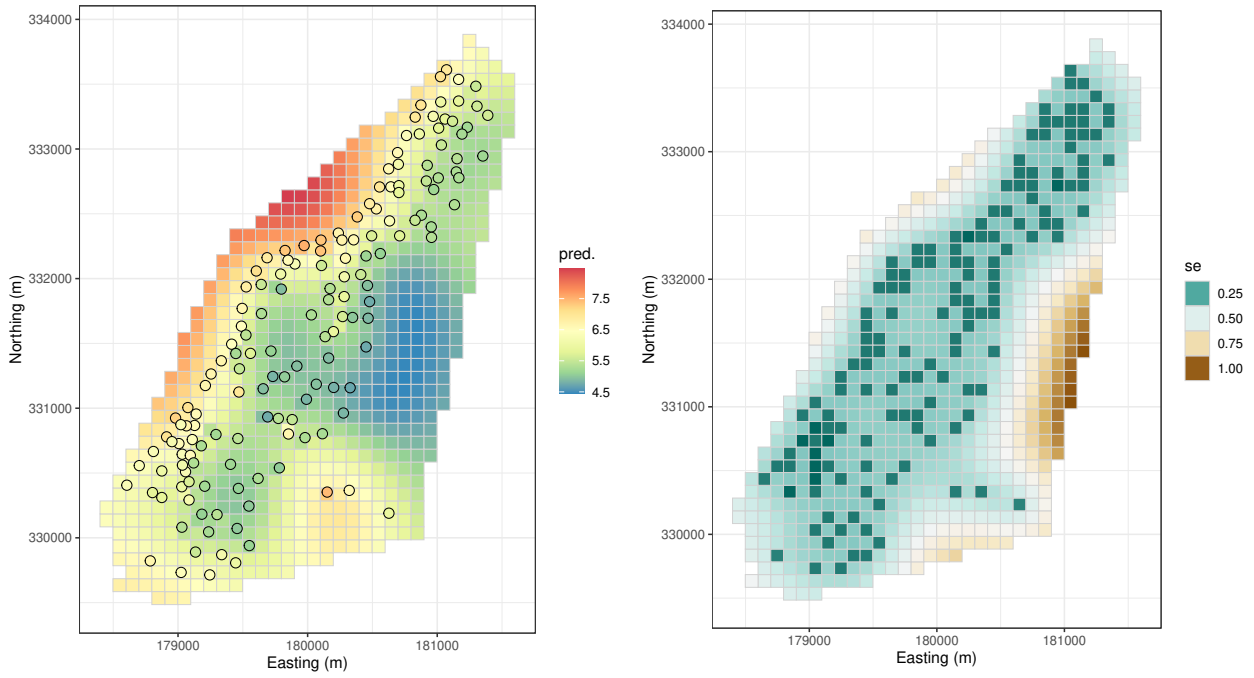
and carry out prediction on the larger polygons:

```
Pred_regions <- predict(S, newdata = Pred_regions)    # prediction polygons
```

The prediction and its standard error can be visualised as before. These plots are shown in Fig. 5.

Point-level data and predictions

In many cases, the user has one data object or data frame containing both observations and prediction locations with accompanying covariates. Missing observations are then usually denoted as `NA`. Since in **FRK**



(a) (b)

Figure 4: Inference at the BAU level using FRK with the `meuse` dataset. (a) FRK prediction. (b) FRK prediction standard error.

all covariates are associated with the process, and not the data, the data frame needs to be used to construct (i) a data object without missing values and one that does not contain covariates, and (ii) BAUs at both the observation and prediction locations containing all the covariate locations.

For example, assume we are missing the first 10 points in the `meuse` dataset.

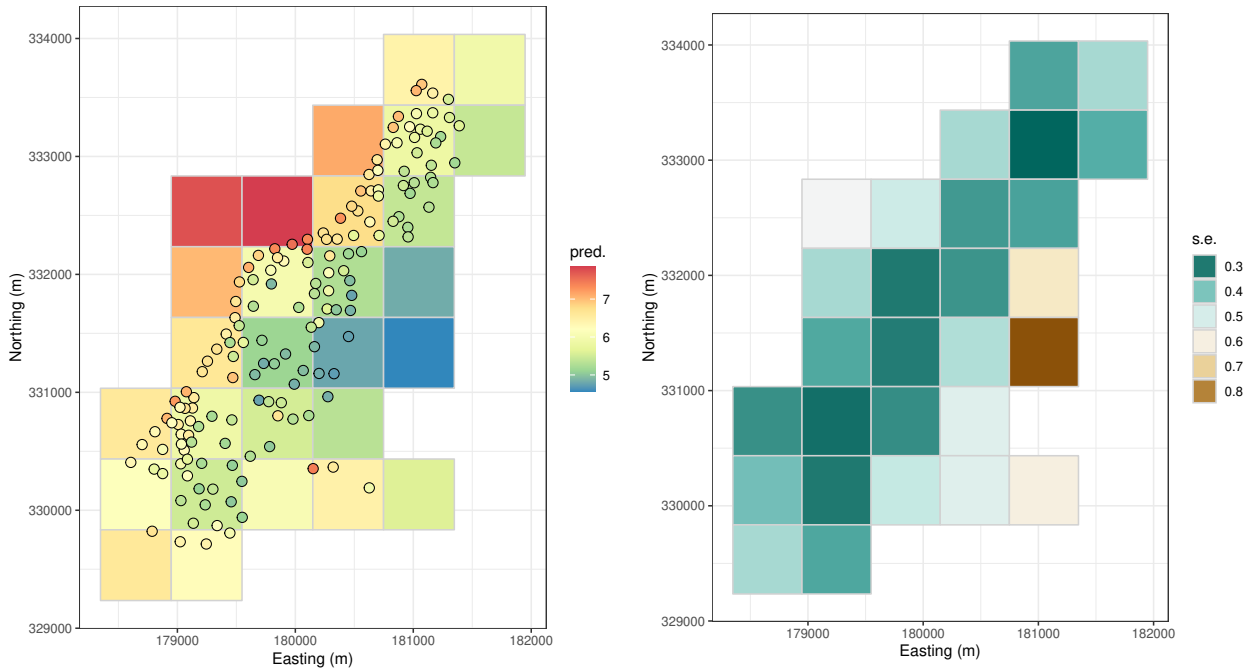
```
data(meuse)
meuse[1:10,"zinc"] <- NA
```

The data object we should use with **FRK** must not contain the missing values, nor the covariates we will use in the model. Once the data frame is appropriately subsetted, it is then cast as a `SpatialPointsDataFrame` as usual.

```
meuse2 <- subset(meuse,!is.na(zinc))
meuse2 <- meuse2[,c("x","y","zinc")]
coordinates(meuse2) <- ~x+y
```

The BAUs, on the other hand, should contain all the data and prediction locations, but not the data itself. Their construction is facilitated by the function `BAUs_from_points` which constructs tiny BAUs around the data and prediction locations.

```
meuse$zinc <- NULL
coordinates(meuse) <- c("x","y")
meuse.grid2 <- BAUs_from_points(meuse)
```



(a) (b)

Figure 5: Prediction and prediction standard error obtained with FRK from the `meuse` dataset over arbitrary polygons. Both quantities are logs of ppm.

Once the complete-data data frame and the BAUs are constructed from the original data frame, FRK may proceed as shown above. Note that predictions are made at both the unobserved and unobserved locations.

3.2 The `AIRS` dataset

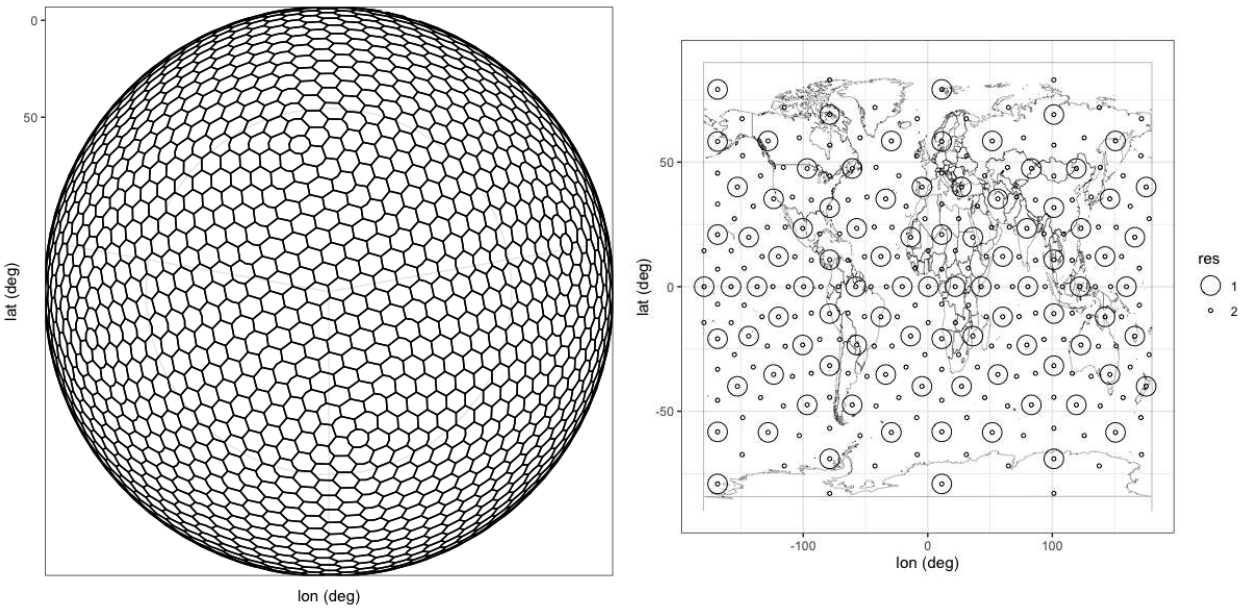
Modelling on the sphere proceeds in a very similar fashion to the plane, except that a coordinate reference system (CRS) on the surface of the sphere needs to be declared for the data. This is implemented using a `CRS` object with string `"+proj=longlat +ellps=sphere"`.

Step 1: Fifteen days of `AIRS` data in May 2003 are included with `FRK` and these can be loaded through the `data` command:

```
data(AIRS_05_2003)                                    ## Load data
```

We next subset the data to include only the first three days, rename `co2std` to `std` (since this is what is required by **FRK** to identify the standard deviation of the measurement error), and select the columns that are relevant for the study. Finally we assign the CRS object:

```
AIRS_05_2003 <-
    dplyr::filter(AIRS_05_2003,day %in% 1:3) %>%     # only first three days
    dplyr::mutate(std=co2std) %>%                    # change std to have suitable name
    dplyr::select(lon,lat,co2avgret,std)             # select columns we actually need
coordinates(AIRS_05_2003) = ~lon+lat                 # change into an sp object
proj4string(AIRS_05_2003) =
```



(a) (b)

Figure 6: BAUs and basis functions used in modelling and predicting with the `AIRS` data. (a) BAUs are the ISEA3H hexagons at resolution 5. (b) Basis function centroids constructed using the function `auto_basis`.

```
            CRS("+proj=longlat +ellps=sphere")      # unprojected coordinates on sphere
```

Step 2: The next step is to create BAUs on the sphere. This is done, again, using the `auto_BAUs` function but this time with the manifold specified to be the sphere. We also specify that we wish the BAUs to form an ISEA Aperture 3 Hexagon (ISEA3H) discrete global grid (DGG) at resolution 6. Resolutions 0–6 are included with **FRK**; for higher resolutions please install the package **dggrids** from `https://github.com/andrewzm/dggrids`. By default, this will create a hexagonal grid on the sphere. However, it is possible to have a rectangular lattice by using `type = "grid"` and specifying the `cellsize` as in Section 3.1; see Section 4.2. An example of an ISEA3H grid, at resolution 5, is shown in Fig. 6.

```
isea3h_sp_poldf <- auto_BAUs(manifold   = sphere(),  # model on sphere
                             isea3h_res = 6,         # isea3h resolution 6 BAUs
                             type = "hex",           # hexagonal grid
                             data = AIRS_05_2003)    # remove BAUs where there is not data
isea3h_sp_poldf$fs = 1                               # fine-scale component
```

Step 3: Now we construct the basis functions, this time of type `"bisquare"` with two resolutions (three would be better, but for computational reasons we leave it at two).

```
G <- auto_basis(manifold = sphere(), # basis functions on the sphere
                data=AIRS_05_2003,   # AIRS data
                nres = 2,            # number of resolutions
                type = "bisquare")   # bisquare function
```

Steps 4–5: Since CO_2 mole fraction has a latitudinal gradient, we use latitude as a covariate in our model. The SRE object is then constructed in the same way as Section 3.1, but this time we set `est_error = FALSE`

since the measurement error is supplied with the data. When multiple data points fall into the same BAU, we assume that each of these data points are conditionally independent readings of the process in the BAU. The matrix $\mathbf{V}_Z$ is therefore not diagonal and this increases computational time considerably. For large point-referenced datasets, such as the `AIRS` dataset, one can leave the argument `average_in_BAU = TRUE` set (by default) to indicate that one wishes to summarise the data at the BAU level. Below, and in all subsequent analyses, we set the maximum number of EM iterations `n_EM = 1` so as to reduce the computational time of the vignette:

```
f <- co2avgret ~ lat + 1          # formula for fixed effects
S <- SRE(f = f,                   # formula for fixed effects
         list(AIRS_05_2003),      # list of data objects
         basis = G,               # basis functions
         BAUs = isea3h_sp_poldf, # BAUs
         est_error = FALSE,       # do not estimate meas. error
         average_in_BAU = TRUE)   # summarise data

S <- SRE.fit(S,                 # SRE model
             n_EM = 1,          # max. no. of EM iterations
             tol = 0.01,        # tolerance at which EM is assumed to have converged
             print_lik=FALSE)   # do not print log-likelihood at each iteration
```

Step 6: We now predict at the BAU level but this time ensure that `obs_fs = TRUE`, which indicates that we are forcing σ^2_ξ to be zero and that the observations have systematic error. Maps of the FRK prediction generated with this flag set are rather smooth (since the basis functions adopted tend to be smooth) and the prediction standard error tends to be higher compared to what one would obtain when setting `obs_fs = FALSE`.

```
isea3h_sp_poldf <- predict(S)          # fs variation is in the observation model
```

The prediction and prediction standard error maps, together with the observation data, are shown in Figs. 7–9.

4 Fixed Rank Kriging in space and time

Although **FRK** is primarily designed for spatial data, it also has functionality for modelling and predicting with spatio-temporal data. The functionality in the software is limited in some respects; in particular, temporal change of support is not possible, and estimation of the standard deviation of the measurement error is not implemented. These features will be implemented in future revisions.

Fixed Rank Kriging is space and time is different from Fixed Rank Filtering [Cressie et al., 2010] where a temporal auto-regressive structure is imposed on the basis-function weights $\{\boldsymbol{\eta}_t\}$, and where subsequently Kalman filtering and Rauch-Tung-Striebel smoothing are used for inference on $\{\boldsymbol{\eta}_t\}$. In FRK, the basis functions also have a temporal dimension; the only new aspect is specifying these space-time basis functions.

We illustrate FRK in space and time using two datasets. The first dataset we consider was obtained from the National Oceanic and Atmospheric Administration (NOAA), and we will term it the `NOAA` dataset. This dataset is included in **FRK**, and it contains daily observations of maximum temperature (`Tmax`) in degrees Fahrenheit at 138 stations in the US between between 32N–46N and 80W–100W, recorded between the years 1990 and 1993 (inclusive); see Fig. 10. We will only consider the 31 maximum temperatures recorded in July 1993 in this vignette. The second dataset we use is the same `AIRS` dataset referred to in Section 3.2. In Section 3.2 only the first 3 days were used to illustrate spatial-only FRK; we now use all 15 days to illustrate spatio-temporal FRK.

For creating spatio-temporal objects used by **FRK** we need to load the **spacetime** package:

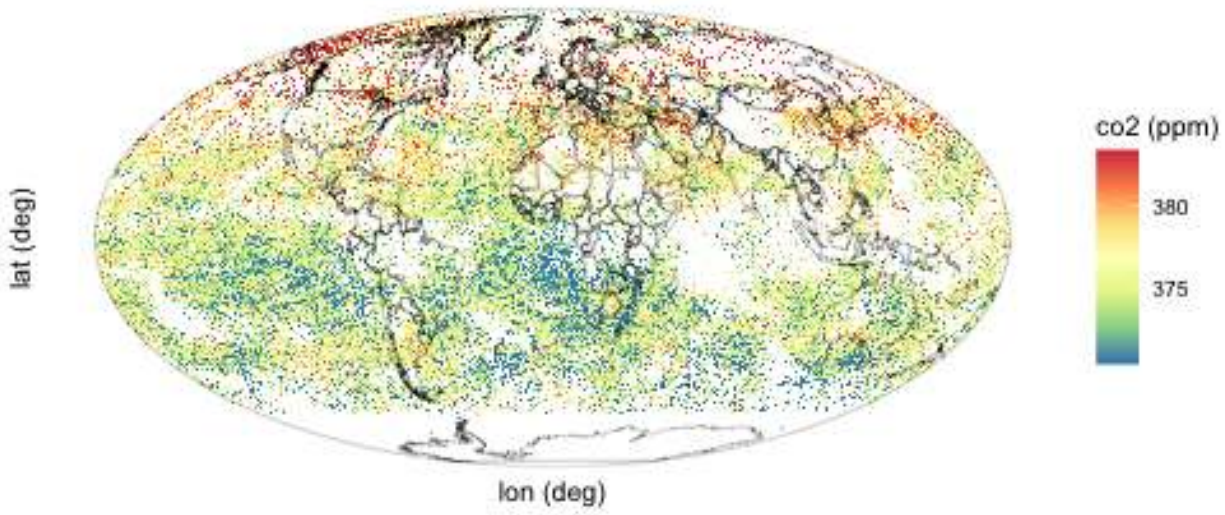


Figure 7: CO_2 mole-fraction readings in ppm from the `AIRS`.

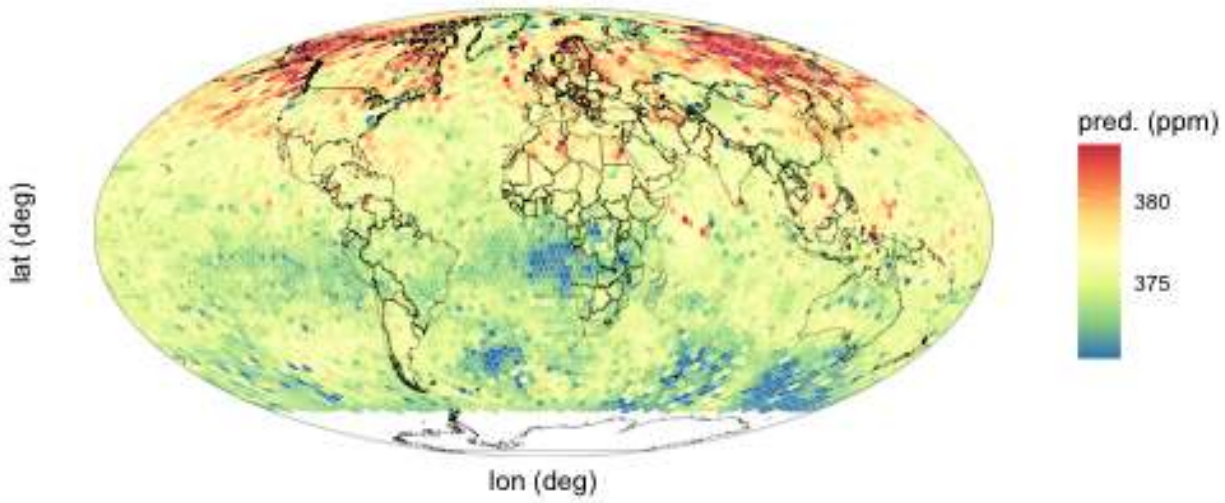


Figure 8: Prediction of $\mathbf{Y}_P$ in ppm following FRK on the `AIRS` data.

```
library(spacetime)
```

4.1 The `NOAA` dataset

Step 1: We load the dataset and extract the data for July 1993 using the commands

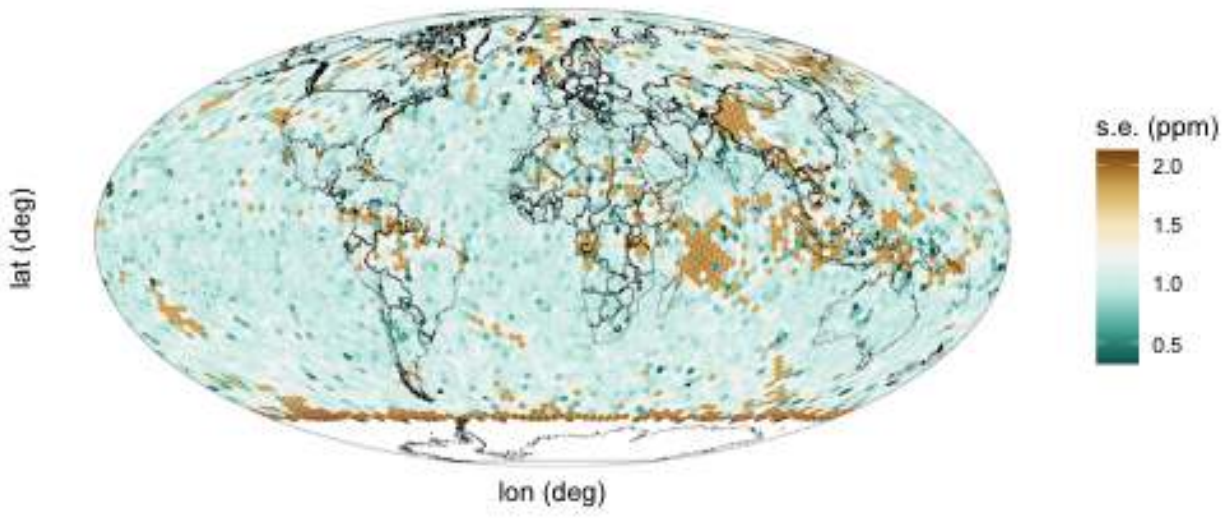


Figure 9: Prediction standard error of $\mathbf{Y}_P$ in ppm following FRK on the `AIRS` data.

```
data("NOAA_df_1990")              # load data
Tmax <- subset(NOAA_df_1990,      # subset the data
               month %in% 7 &     # May to July
               year == 1993)      # year of 1993
```

To construct a spatio-temporal object, one must first define the temporal component as a `Date` object by stringing the year, month and day together:

```
Tmax <- within(Tmax,
               {time = as.Date(paste(year,month,day,sep="-"))})  # create Date field
```

Since the data is point-referenced, we need to cast our data into a 'spatio-temporal irregular data frame', `STIDF`; refer to the vignette `JSS816` for various ways to do this. One of the most straightforward approaches is to use the function `stConstruct` in the package **spacetime**. The function needs to be supplied along with the data, the names of the spatial coordinates field, the name of the Date field, and a flag indicating whether the data can be treated as having been recorded over the temporal interval and not at the specific instant recorded in `time` (in our case `interval=TRUE`).

```
STObj <- stConstruct(x = Tmax,                    # dataset
                     space = c("lon","lat"),      # spatial fields
                     time="time",                 # time field
                     interval=TRUE)               # time reflects an interval
```

Unlike for the spatial-only case, the standard deviation of the measurement error needs to be specified. In this case, we conservatively set it to be 2 degrees Fahrenheit, although it is likely to be much less in practice. We also treat the data as being on $\mathbb{R}^2$ (that is, where space is the plane; we consider space-time data where space is the surface of a sphere in Section 4.2):

```
STObj$std <- 2
```

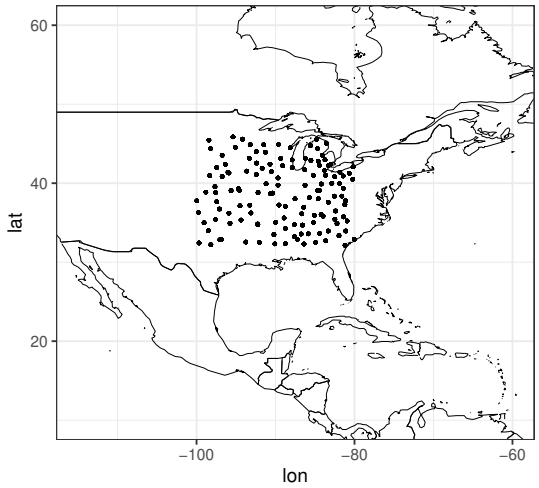


Figure 10: Station locations from which the maximum temperature readings in the `NOAA` dataset were obtained.

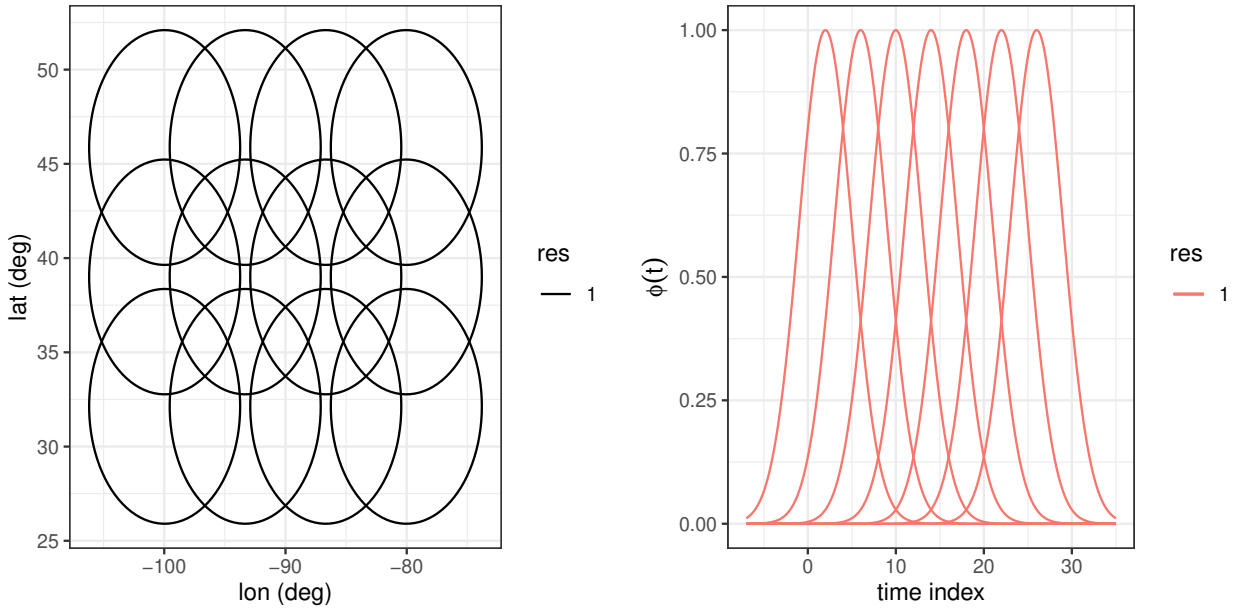
Step 2: When dealing with spatio-temporal data, the BAUs are space-time regular lattices, which are classified in **spacetime** as a 'spatio-temporal fixed data frame', `STFDF`; type `help(STFDF)` for details. `STFDF` objects may be constructed manually or by using the helper function `auto_BAUs`. In the following, the helper function is used to construct BAUs in a space-time cube, centred around the data `STObj`, with each BAU of size 1 deg. latitude $\times$ 1 deg. longitude $\times$ 1 day. The new arguments here are `manifold = STplane()`, which indicates that we are going to model a spatio-temporal field on the 2D plane, and `tunit = "days"`, which indicates that each BAU has a temporal 'width' equal to one day. Once again, we specify the fine-scale component to be homoscedastic:

```
grid_BAUs <- auto_BAUs(manifold=STplane(),     # spatio-temporal process on the plane
                       data=STObj,             # data
                       cellsize = c(1,1,1),    # BAU cell size
                       type="grid",            # grid or hex?
                       convex=-0.1,            # parameter for hull construction
                       tunit="days",           # time unit
                       nonconvex_hull=FALSE)   # convex hull
grid_BAUs$fs = 1                          # fine-scale variation
```

Step 3: The simplest way to construct spatio-temporal basis functions is to first construct spatial basis functions, then temporal basis functions, and then combine them by taking their tensor product. To construct spatial basis functions, we first project the spatio-temporal data onto the spatial domain (collapse out time) using `as(STObj,"Spatial")`, and then construct spatial basis function using `auto_basis`:

```
G_spatial <- auto_basis(manifold = plane(),           # spatial functions on the plane
                        data=as(STObj,"Spatial"),     # remove the temporal dimension
                        nres = 1,                     # three resolutions
                        type = "bisquare",            # bisquare basis functions
                        regular = 1)                  # regular basis functions
```

For the temporal basis functions, we use the function `local_basis`, which gives the user more control over the location parameters and the scale parameters of the basis functions. In this case we specify that we want basis functions on the real line, located between $t = 2$ and $t = 28$ at an interval spacing of 4. Here, each location represents a temporal interval used in the construction of `grid_BAUs`; for example, $t = 1$ corresponds to 1993-07-01, $t = 5$ to 1993-07-05, and so on.



(a) (b)

Figure 11: Spatial and temporal basis functions used to construct the spatio-temporal basis functions. (a) Spatial support of the bisquare spatial basis functions. (b) The temporal basis functions.

```
print(head(grid_BAUs@time))                                  # show time indices

##            timeIndex
## 1993-07-01         1
## 1993-07-02         2
## 1993-07-03         3
## 1993-07-04         4
## 1993-07-05         5
## 1993-07-06         6

G_temporal <- local_basis(manifold = real_line(),           # functions on the real line
                          type = "Gaussian",                # Gaussian functions
                          loc = matrix(seq(2,28,by=4)),     # locations of functions
                          scale = rep(3,7))                 # scales of functions
```

The basis functions can be visualised using `show_basis`. The generated basis functions are shown in Figure 11:

```
basis_s_plot <- show_basis(G_spatial) + xlab("lon (deg)") + ylab("lat (deg)")
basis_t_plot <- show_basis(G_temporal) + xlab("time index") + ylab(expression(phi(t)))
```

The spatio-temporal basis functions are then constructed using the function `TensorP` as follows:

```
G <- TensorP(G_spatial,G_temporal)          # take the tensor product
```

Steps 4–6: We next construct the SRE model, using an intercept and latitude as fixed effects, `STObj` as the data, `G` as the set of basis functions, and `grid_BAUs` as the BAUs. We also specify `est_error = FALSE`

since this functionality is currently not implemented for spatio-temporal data. The SRE model is then fitted using the familiar command `SRE.fit` and prediction is carried out using `predict`:

```
f <- z ~ 1 + lat                  # fixed effects part
S <- SRE(f = f,                   # formula
         data = list(STObj),      # data (can have a list of data)
         basis = G,               # basis functions
         BAUs = grid_BAUs,        # BAUs
         est_error = FALSE)       # do not estimate measurement-error variance

S <- SRE.fit(S,                  # estimate parameters in the SRE model S
             n_EM = 1,           # maximum no. of EM iterations
             tol = 0.1,          # tolerance on log-likelihood
             print_lik=FALSE)    # print log-likelihood trace

grid_BAUs <- predict(S, obs_fs = FALSE)
```

Plotting proceeds precisely the same way as in Section 3.1, however now we need to convert the spatial polygons at multiple time points to data frames. This can be done by simply iterating through the time points we wish to visualise. In the following, we extract a data frame for the BAUs on days 1, 4, 8, 12, 16, and 20. The prediction and the prediction standard error are shown in Figs. 12 and 13, respectively. Note how the prediction has both smooth and fine-scale components. This is expected, since fine-scale variation was, this time, included in the prediction. Note also that the BAUs we used are not located everywhere within the square domain of interest. This is because the `auto_BAUs` function carefully chooses a (non-square) domain in an attempt to minimise the number of BAUs needed. This can be adjusted by changing the `convex` parameter in `auto_BAUs`.

```
analyse_days <- c(1,4,8,12,16,20)  # analyse only a few days
df_st <- lapply(analyse_days,      # for each day
       function(i)
           as(grid_BAUs[,i],"data.frame") %>%
           cbind(day = i))         # add day number to df
df_st <- do.call("rbind",df_st)    # append all dfs together
```

In order to model the `NOAA` dataset on a subset of the sphere, we first need to associate an appropriate Coordinate Reference System with `STObj`,

```
proj4string(STObj) <- "+proj=longlat +ellps=sphere"
```

and then adjust the BAUs and basis functions used. This entails using `STsphere()` instead of `STplane()` in BAU construcation and `sphere()` instead of `plane()` in spatial-basis-function construction.

```
grid_BAUs <- auto_BAUs(manifold=STsphere(),       # spatio-temporal process on the sphere
                       data=STObj,                # data
                       cellsize = c(1,1,1),       # BAU cell size
                       type="grid",               # grid or hex?
                       convex=-0.1,               # parameter for hull construction
                       tunit="days")              # time unit

G_spatial <- auto_basis(manifold = sphere(),      # spatial functions on the plane
                        data=as(STObj,"Spatial"), # remove the temporal dimension
                        nres = 2,                 # two resolutions of DGG
                        type = "bisquare",        # bisquare basis functions
                        prune=15,                 # prune basis functions
                        isea3h_lo = 4)            # but remove those lower than res 4
```

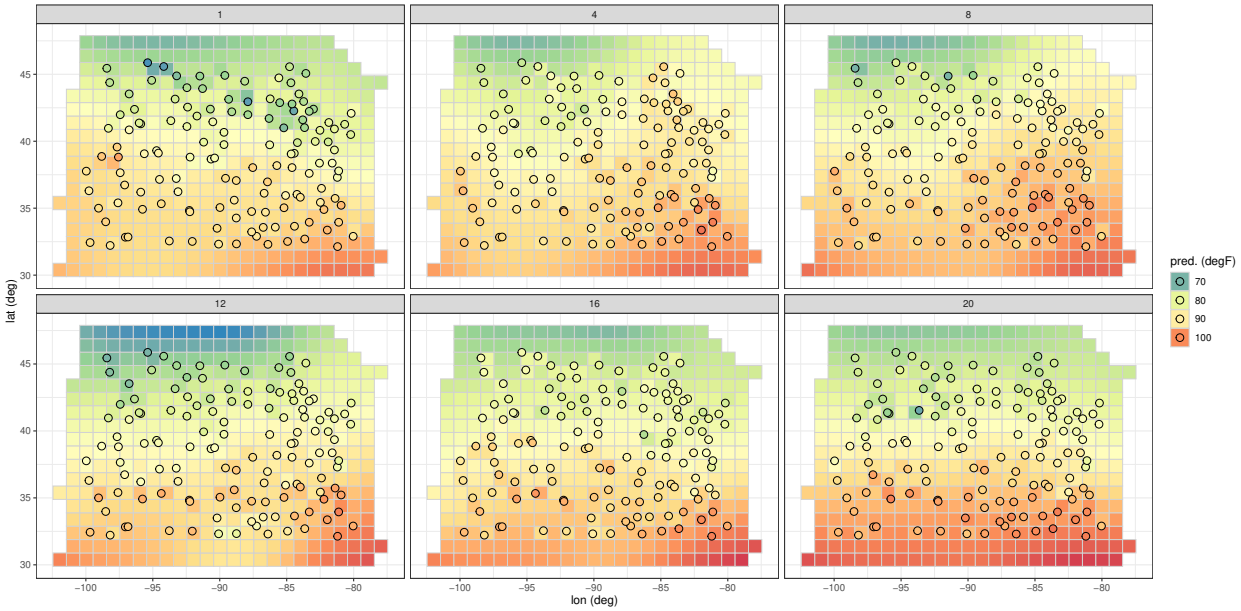


Figure 12: Spatio-temporal FRK prediction of `Tmax` on the plane in degrees Fahrenheit within a domain enclosing the region of interest for six selected days spanning the temporal window of the data: 01 July 1993 – 20 July 2003.

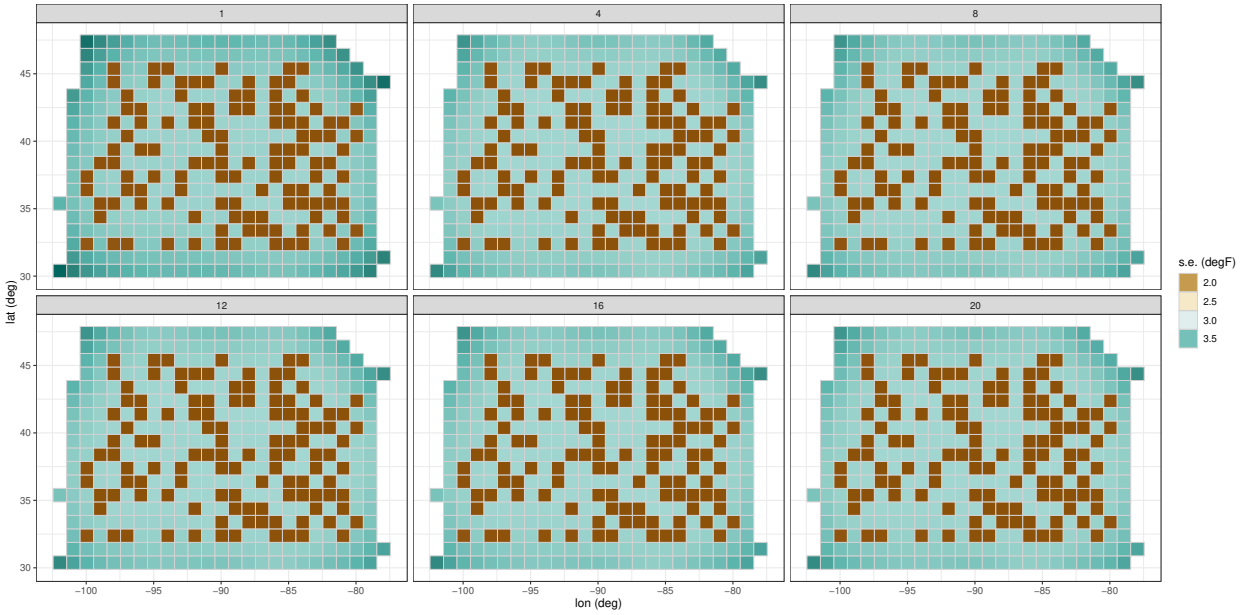


Figure 13: Spatio-temporal FRK prediction standard error of `Tmax` on the plane in degrees Fahrenheit within a domain enclosing the region of interest for the same six days selected in Fig. 12 and spanning the temporal window of the data, 01 July 1993 – 20 July 2003.

Recall that when calling `auto_basis` on the sphere, basis functions are automatically constructed at locations specified by the DGGs. In the code given above, we use the first six resolutions (resolutions 0 – 5) of the DGGs but discard resolutions less than 4 by using the argument `isea3h_lo = 4`. The `prune=15` argument behaves as above on the basis functions at these higher resolutions. The basis functions constructed

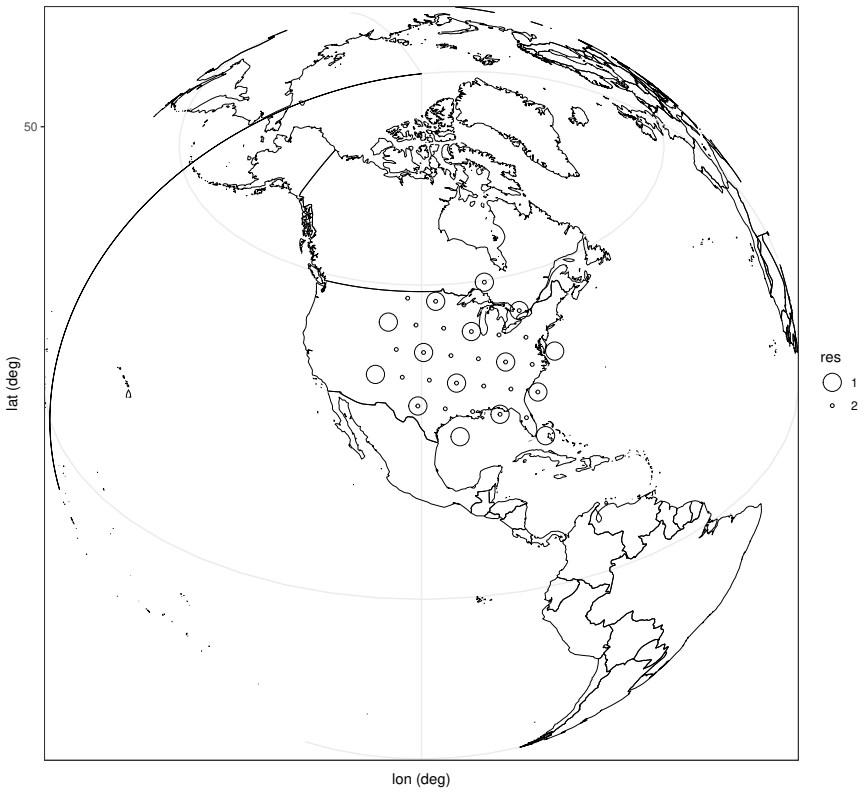


Figure 14: Basis functions for FRK on the sphere with the `NOAA` dataset using two ISEA3H DGGs for location parameters of the basis functions.

using this code are shown in Fig. 14. We provide more details on FRK on the sphere in Section 4.2.

4.2 The `AIRS` dataset

In this section we use spatio-temporal FRK on the sphere to obtain FRK predictions and prediction standard errors of CO_2, in ppm, between 01 May 2003 and 15 May 2003 (inclusive). To keep the package size small, the dataset `AIRS_05_2003` only contains data in these first 15 days of May 2003.

Step 1: First we load the dataset that is available with **FRK**:

```
data(AIRS_05_2003)   # load AIRS data
```

and then rename `co2std` to `std` and attribute the time index `t` to the day number. We also use 20000 data points chosen at random between 01 May 2003 and 15 May 2003 (inclusive) in order to keep the compilation time of the vignette low.

```
set.seed(1)
AIRS_05_2003 <- mutate(AIRS_05_2003,           # take the data
                       std=co2std) %>%         # rename std
                sample_n(20000)                # sample 20000 points
```

As with the `NOAA` dataset, we create a date field using `as.Date`

```
AIRS_05_2003 <- within(AIRS_05_2003,
               {time = as.Date(paste(year,month,day,sep="-"))})  # create Date field
```

and construct the spatio-temporal object (`STIDF`) using `stConstruct`:

```
STObj <- stConstruct(x = AIRS_05_2003,              # dataset
                 space = c("lon","lat"),            # spatial fields
                 time ="time",                      # time field
                 crs = CRS("+proj=longlat +ellps=sphere"),  # CRS
                 interval=TRUE)                     # time reflects an interval
```

Step 2: We next construct the BAUs. This time we specify `STsphere()` for the manifold, and for illustration we discretise the sphere using a regular grid rather than a hexagonal lattice. To do this we set a `cellsize` and specify `type="grid"`. We also supply `time(STObj)` so that the BAUs are constructed around the time of the data; if we supply `STObj` instead, then BAUs are pruned spatially. We show the BAUs generated using `time(STObj)` and `STObj` in Fig. 15.

```
## Prediction (BAU) grid
grid_BAUs <- auto_BAUs(manifold=STsphere(),          # space-time field on sphere
                           data=time(STObj),         # temporal part of the data
                           cellsize = c(5,5,1),      # cellsize (5 deg x 5 deg x 1 day)
                           type="grid",              # grid (not hex)
                           tunit = "days")           # time spacing in days
grid_BAUs$fs = 1
```

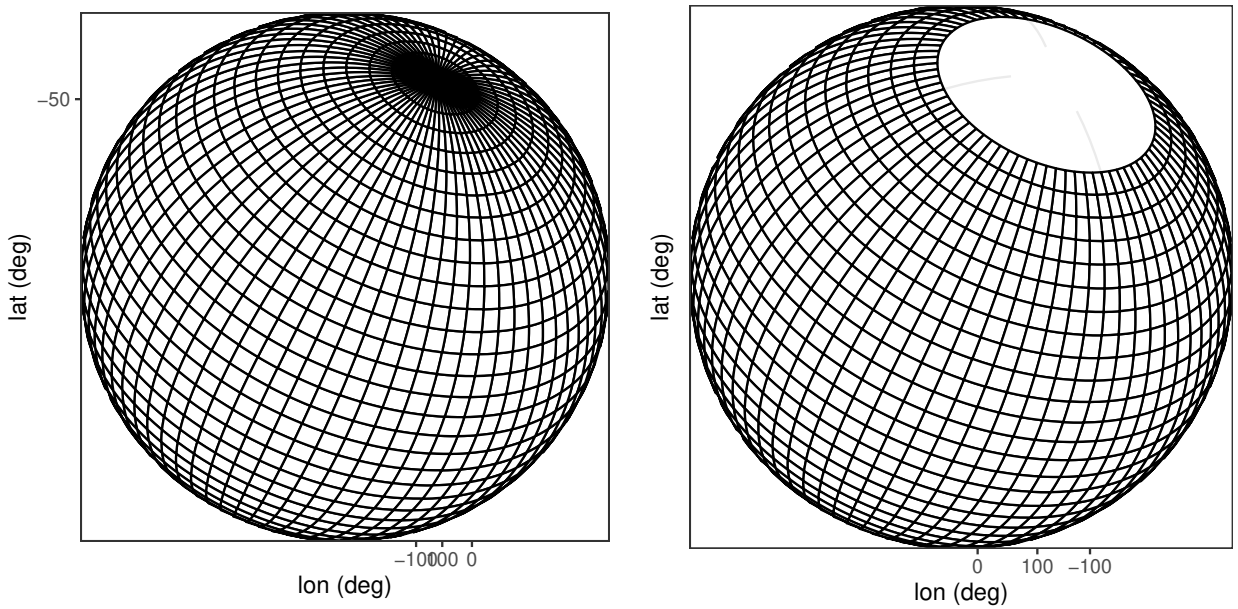
Step 3: We next construct the spatio-temporal basis functions. This proceeds in exactly the same way as in the `NOAA` dataset: We first construct spatial basis functions, then temporal basis functions, and then we find their tensor product.

Since by default basis functions on the sphere are set the cover the whole globe, a restriction to the area of interest needs to be enforced bt 'pruning' basis functions outside this region. Pruning is controlled by the parameter `prune`. Another argument `subsamp` then dictates how many data points (chosen at random) should be used when carrying out the pruning. In general, the higher `nres`, the higher `subsamp` should be in order to ensure that high resolution basis functions are not omitted where data is actually available. The argument `subsamp` need only be used when pruning with the entire dataset consumes a lot of resources. See `help(auto_basis)` for details.

```
G_spatial <- auto_basis(manifold = sphere(),       # functions on sphere
                        data=as(STObj,"Spatial"), # collapse time out
                        nres = 1,                 # use three DGGRID resolutions
                        prune= 15,                 # prune basis functions
                        type = "bisquare",        # bisquare basis functions
                        subsamp = 2000,          # use only 2000 data points for pruning
                        isea3h_lo = 2)            # start from isea3h res 2

G_temporal <- local_basis(manifold=real_line(),      # functions on real line
                          loc = matrix(c(2,7,12)),   # location parameter
                          scale = rep(3,3),          # scale parameter
                          type = "Gaussian")
G_spacetime <- TensorP(G_spatial,G_temporal)
```

Steps 4–6: Finally, the SRE model is constructed and fitted as in the other examples. Recall that `est_error = FALSE` is required with spatio-temporal data and, since we have multiple data per BAU we also set



(a) (b)

Figure 15: Gridded BAUs on the sphere used for modelling and predicting with the `AIRS` data. (a) BAUs constructed when supplying only the temporal indices of the data (the entire sphere is covered with BAUs within the specified time period). (b) BAUs constructed when supplying the entire dataset. The view of the sphere is from the bottom; in this case there is no data below 60°S and thus BAUs have been omitted from this region.

`average_in_BAU = TRUE`. For predicting, we use the `pred_time` flag to indicate at which time points we wish to predict; here the numbers correspond to the time indices of the BAUs. We specify `pred_time = c(4,8,12)`, which indicates that we want to predict on the 4, 8 and 12 May 2003. The data, prediction, and prediction standard error for these days are given in Figs. 16–18.

```
f <- co2avgret ~ lat +1            # formula for fixed effects
S <- SRE(f = f,                    # formula
         data = list(STObj),       # spatio-temporal object
         basis = G_spacetime,      # space-time basis functions
         BAUs = grid_BAUs,         # space-time BAUs
         est_error = FALSE,        # do not estimate measurement error
         average_in_BAU = TRUE)    # average data that fall inside BAUs

S <- SRE.fit(S,                    # SRE model
             n_EM = 1,             # max. EM iterations
             tol = 0.01)           # convergence criteria

grid_BAUs <- predict(S, obs_fs = TRUE,          # fs variation is in obs. model
                     pred_time = c(4L,8L,12L)) # predict only at select days
```

5 Other topics

Sections 1–4 have introduced the core functionality of **FRK**. The purpose of this section is to present additional functionality that may be of use to the analyst.

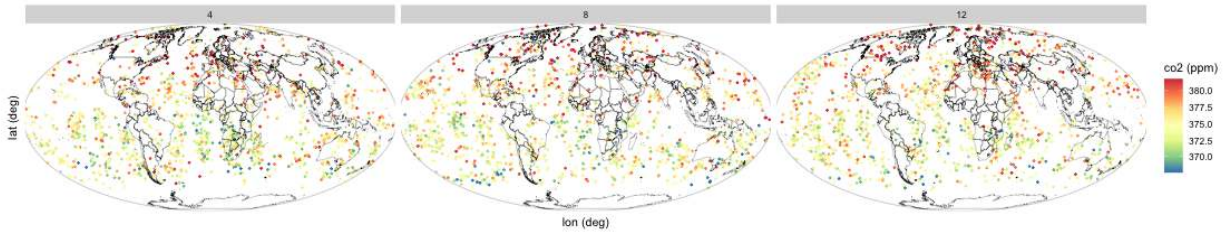


Figure 16: CO_2 readings taken from the `AIRS` on the 04, 08 and 12 May 2003 in ppm.

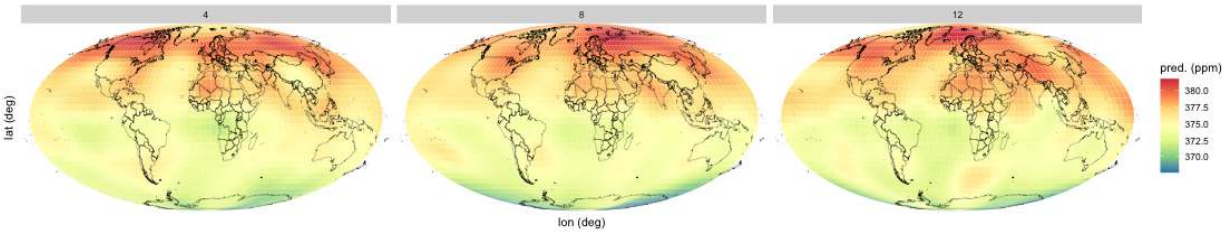


Figure 17: Prediction of $\mathbf{Y}_P$ in ppm on 04, 08, and 12 May 2003 obtained with **FRK** on the `AIRS` data.

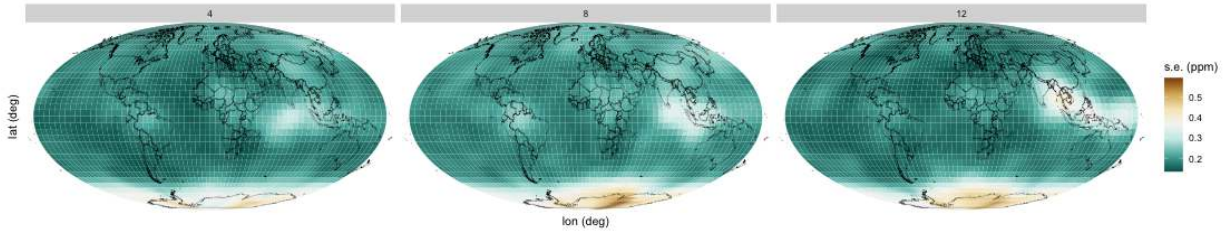


Figure 18: Prediction standard error of $\mathbf{Y}_P$ in ppm on 04, 08 and 12 May 2003 obtained with **FRK** on the `AIRS` data.

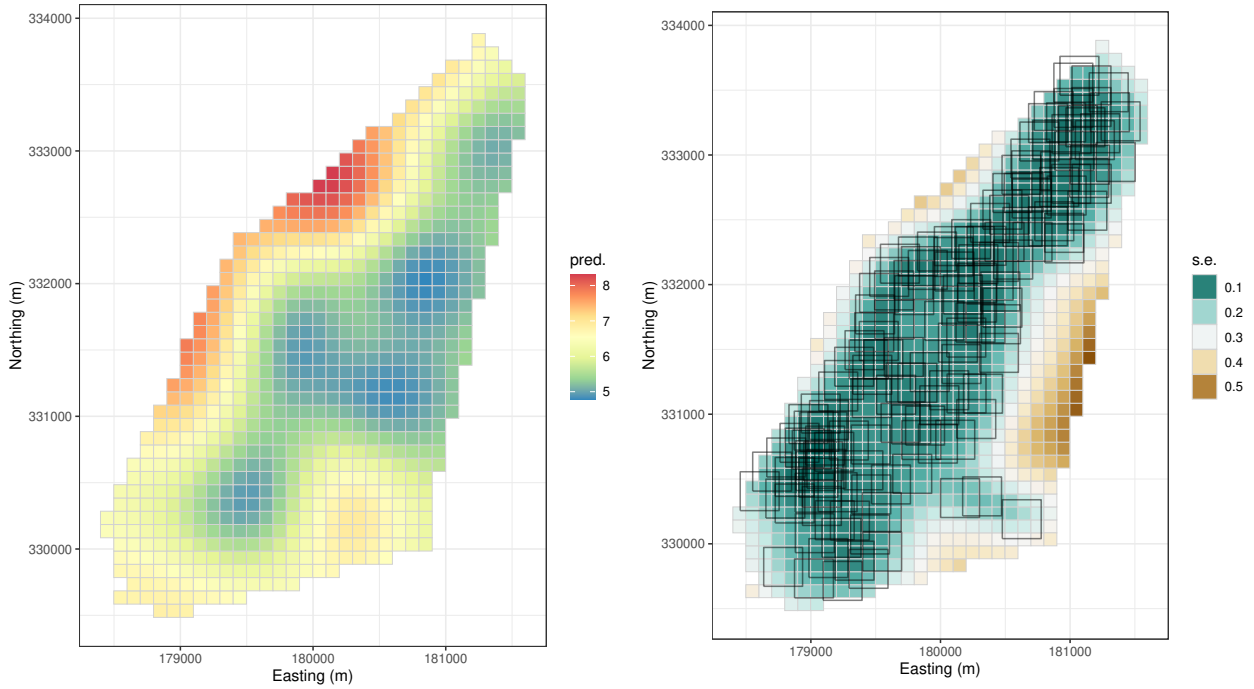
5.1 Multiple observations with different supports

The main advantage of using BAUs is that one can make use of multiple datasets with different spatial supports without any added difficulty. Consider the `meuse` dataset. We synthesise observations with a large support by changing the `meuse` object into a `SpatialPolygonsDataFrame`, where each polygon is a square of size 300 m $\times$ 300 m centred around the original `meuse` data point, and with the value of log-zinc concentration taken as the block-average of the prediction in the previous analysis. Once this object is set up, which we name `meuse_pols`, the analysis proceeds in precisely the same way as in Section 3.1, but with `meuse_pols` used instead of `meuse`.

The prediction and the prediction standard error using `meuse_pols` are shown in Fig. 19. In Fig. 19 (b) we also overlay the footprints of the observations. Note how the observations affect the prediction standard error in the BAUs and how BAUs which are observed by overlapping observations have lower prediction error than those that are only observed once. As expected, the prediction standard error is, overall, considerably higher than that in Fig. 4. Note also that the supports of the observations and the BAUs do not precisely overlap. For simplicity, we assumed that an observation "influences" a BAU only if the centroid of the BAU lies within the observation footprint. Refining this will require a more detailed consideration of the BAU and observation footprint geometry and it will be considered in future revision.

5.2 Anisotropy: Changing the distance measure

So far we have only considered isotropic fields. Anisotropy can be easily introduced by changing the distance measure associated with the manifold. To illustrate this, we simulate below a highly anisotropic, noisy,



(a) (b)

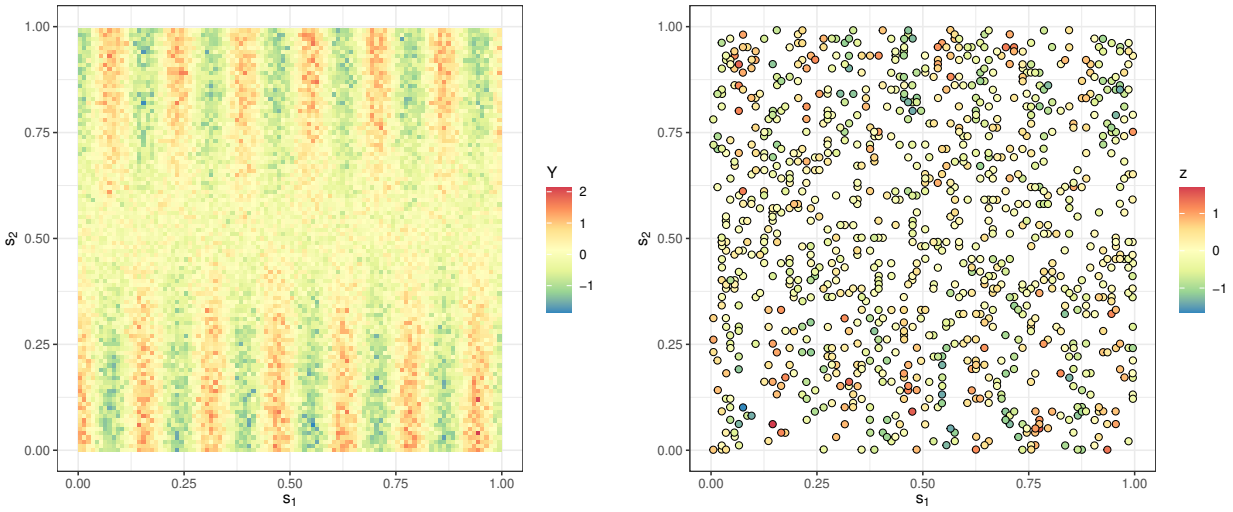
Figure 19: Prediction and prediction standard error obtained with FRK using the `meuse` dataset where each observation is assuming to have a spatial footprint of 300 m × 300m. (a) FRK prediction at the BAU level. (b) FRK prediction standard error at the BAU level. The black hexagons outline the spatial footprints of the data.

spatio-temporal process on a fine grid in $D = [0, 1] \times [0, 1]$ and sample 1000 points chosen at random from it. The process and the sampled data are shown in Fig. 20.

```
set.seed(1)
N <- 50
sim_process <- expand.grid(x = seq(0.005,0.995,by=0.01),        # x grid
                           y = seq(0.001,0.995,by=0.01)) %>%    # y grid
    mutate(proc = cos(x*40)*cos(y*3) + 0.3*rnorm(length(x)))    # anisotropic function

sim_data <- sample_n(sim_process,1000) %>%                      # sample data from field
    mutate(z = proc + 0.1*rnorm(length(x)),                     # add noise
           std = 0.1,                                           # with 0.1 std
           x = x + runif(1000)*0.001,                           # jitter x locations
           y = y + runif(1000)*0.001)                           # jitter y locations
coordinates(sim_data) = ~x + y                                  # change into SpatialPoints
```

To create the modified distance measure, we note that the spatial frequency in x is approximately four times that in y. Therefore, in order to generate anisotropy, we use a measure that scales x by 4. In **FRK**, a `measure` object requires a distance function, and the dimension of the manifold on which it is used, as follows:



(a) (b)

Figure 20: FRK with anisotropic fields. (a) Simulated process. (b) Observed data.

```
scaler <- diag(c(4,1))                                  # scale x by 4
asymm_measure <- new("measure",                         # new measure object
                     dist=function(x1,x2=x1)               # new distance function
                           FRK:::distR(x1 %*% scaler,    # scaling of first point
                                       x2 %*% scaler),   # scaling of second point
                     dim=2L)                            # in 2D
```

The distance function used on the plane can be changed by assigning the object `asymm_measure` to the manifold:

```
TwoD_manifold <- plane()                  # Create R2 plane
TwoD_manifold@measure <- asymm_measure    # Assign measure
```

We now generate a grid of basis functions (at a single resolution) manually. First, we create a 5×14 grid on D, which we will use as centres for the basis functions. We then call the function `local_basis` to construct bisquare basis functions centred at these locations with a range parameter (i.e., the radius in the case of a bisquare) of 0.4. Due to the scaling used, this implies a range of 0.1 in x and a range of 0.4 in y. Basis function number 23 is illustrated in Fig. 21.

```
basis_locs <- seq(0,1,length=14) %>%                 # x locations
   expand.grid(seq(0,1,length=5)) %>%                # y locations
   as.matrix()                                       # convert to matrix
G <-  local_basis(manifold = TwoD_manifold,          # 2D plane
                  loc=basis_locs,                    # basis locations
                  scale=rep(0.4,nrow(basis_locs)),   # scale parameters
                  type="bisquare")                   # type of function
```

From here on, the analysis proceeds in exactly the same way as shown in all the other examples. The prediction and prediction standard error are shown in Fig. 22.

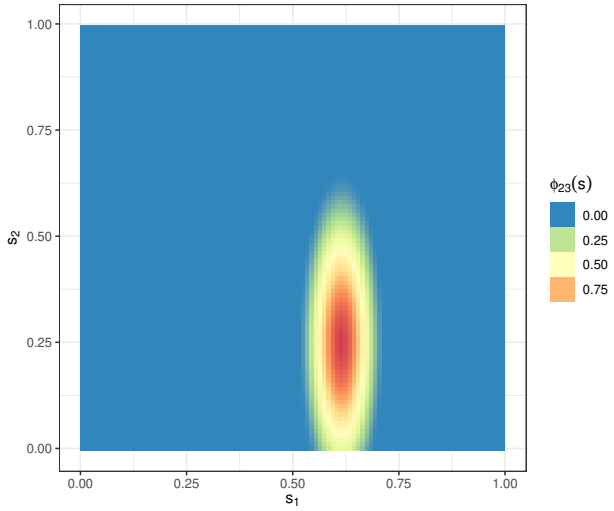


Figure 21: Basis function 23 of the 75 constructed to fit an anisotropic spatial field. Anisotropy is obtained by changing the `measure` object of the manifold on which the basis function is constructed.

5.3 Customised basis functions and Basic Areal Units (BAUs)

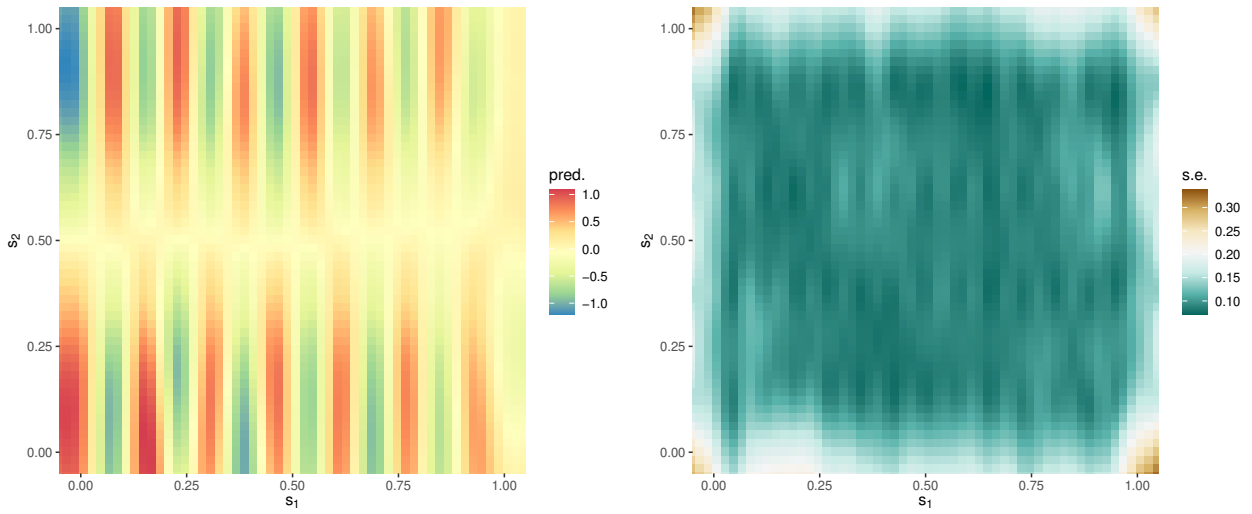
The package **FRK** provides the functions `auto_BAUs` and `auto_basis` to help the user construct the BAUs and basis functions based on the supplied data. These, however, could be done manually. When doing so it is important that some rules are adhered to: The object containing the basis functions needs to be of class `Basis`. This class contains 5 slots:

- **dim**: The dimension of the manifold.
- **fn**: A list of functions. By default, distances in these functions are attributed with a manifold, but arbitrary distances can be used.
- **pars**: A list of parameters associated with each basis function. For the local basis functions used in this vignette (constructed using `auto_basis` or `local_basis`), each list item is a list with fields `loc` and `scale` where `length(loc)` is equal to the dimension of the manifold and `length(scale) = 1`.
- **df**: A data frame with number of rows equalling the number of basis functions, and containing auxiliary information about the basis functions (e.g., resolution number).
- **n**: An integer equalling the number of basis functions.

There is no constructor yet for `Basis`, and the R command `new` needs to be used to create this object from scratch.

There are less restrictions for constructing BAUs; they need to be stored as a `SpatialPolygonsDataFrame` object, and the `data` slot of this object must contain

- All covariates used in the model.
- A field `fs` denoting the fine-scale variation.
- Fields that can be used to summarise the BAU as a point, typically the centroid of each polygon. The names of these fields need to equal those of the `coordnames(BAUs)` (typically `c("x","y")` or `c("lon","lat")`).



(a) (b)

Figure 22: FRK using data generated by an anisotropic field. (a) FRK prediction. (b) FRK prediction standard error.

6 Future work

There are a number of important features that remain to be implemented in future revisions, some of which are listed below:

- Currently, **FRK** is designed to work with local basis functions with analytic form. However, the package can also accommodate basis functions that have no known functional form, such as empirical orthogonal functions (EOFs) and classes of wavelets defined iteratively; future work will attempt to incorporate the use of such basis functions. Vanilla FRK (FRK-V), where the entire positive-definite matrix $\mathbf{K}$ is estimated, is particularly suited to the former (EOF) case where one has very few basis functions that explain a considerable amount of observed variability.

- There is currently no component of the model that caters for sub-BAU process variation, and each datum that is point-referenced is mapped onto a BAU. Going below the BAU scale is possible, and intra-BAU correlation can be incorporated if the covariance function of the process at the sub-BAU scale is known [Wikle and Berliner, 2005].

- Most work and testing in **FRK** has been done on the real line, the 2D plane and the surface of the sphere ($\mathbb{S}^2$). Other manifolds can be implemented since the SRE model always yields a valid spatial covariance function, no matter the manifold. Some, such as the 3D hyperplane, are not too difficult to construct. Ultimately, it would be ideal if the user can specify his/her own manifold, along with a function that can compute the appropriate distances on the manifold.

- Although designed for very large data, **FRK** begins to slow down when several hundreds of thousands of data points are used. The flag `average_in_BAU` can be used to summarise the data and hence reduce the size of the dataset, however it is not always obvious how the data should be summarised (and whether one should summarise it in the first place). Future work will focus on providing the user with different options for summarising the data.

- Currently all BAUs are assumed to be of equal area. This is not problematic in our case, since we use equal-area icosahedral grids on the surface of the sphere, and regular grids on the real line and the plane. However, a regular grid on the surface of the sphere, for example that shown in Figure 6, right panel, is not an equal area grid and appropriate weighing should be used in this case when aggregating to arbitrary polygons.

In summary, the package **FRK** is designed to address the majority of needs for spatial and spatio-temporal prediction. In separate tests we found that the low-rank model used by the package has validity (accurate coverage) in a big-data scenario when compared to high-rank models implemented by other packages such as **LatticeKrig** and **INLA**. However, it is less efficient (larger root mean squared prediction errors) when data density is high and the basis functions are unable to capture the spatial variability.

The development page of **FRK** is `https://github.com/andrewzm/FRK`. Users are encouraged to report any bugs or issues relating to the package on this page.

Acknowledgements

Package development was facilitated with **devtools**; this paper was compiled using **knitr**, and package testing was carried out using **testthat** and **covr**. We thank Jonathan Rougier, Christopher Wikle, Clint Shumack, Enki Yoo and Behzad Kianian for providing valuable feedback on the package.

References

S. Banerjee, A.E. Gelfand, A.O. Finley, and H. Sang. Gaussian predictive process models for large spatial data sets. *Journal of the Royal Statistical Society B*, 70:825–848, 2008.

M.T. Chahine, T.S. Pagano, H.H. Aumann, R. Atlas, C. Barnet, J. Blaisdell, L. Chen, M. Divakarla, E.J. Fetzer, M. Goldberg, C. Gautier, S. Granger, S. Hannon, F.W. Irion, R. Kakar, E. Kalnay, B.H. Lambrigtsen, S.-Y. Lee, J.L. Marshall, W.W. McMillan, L. McMillin, E.T. Olsen, H. Revercomb, P. Rosenkranz, W.L. Smith, D. Staelin, L. Larrabee Strow, J. Susskind, D. Tobin, W. Wolf, and L. Zhou. AIRS: Improving weather forecasting and providing new data on greenhouse gases. *Bulletin of the American Meteorological Society*, 87:911–926, 2006.

N. Cressie and G. Johannesson. Spatial prediction for massive data sets. In *Mastering the Data Explosion in the Earth and Environmental Sciences: Proceedings of the Australian Academy of Science Elizabeth and Frederick White Conference*, pages 1–11, Canberra, Australia, 2006. Australian Academy of Science.

N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society B*, 70:209–226, 2008.

N. Cressie, T. Shi, and E.L. Kang. Fixed rank filtering for spatio-temporal data. *Journal of Computational and Graphical Statistics*, 19:724–745, 2010.

A.O. Finley, S. Banerjee, and B.P. Carlin. spBayes: an R package for univariate and multivariate hierarchical point-referenced spatial models. *Journal of Statistical Software*, 19:1–24, 2007.

H.V. Henderson and S.R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23:53–60, 1981.

E.L. Kang and N. Cressie. Bayesian inference for the spatial random effects model. *Journal of the American Statistical Association*, 106:972–983, 2011.

E.L. Kang, D. Liu, and N. Cressie. Statistical analysis of small-area data based on independence, spatial, non-hierarchical, and hierarchical models. *Computational Statistics & Data Analysis*, 53:3016–3032, 2009.

M. Katzfuss and N. Cressie. Spatio-temporal smoothing and EM estimation for massive remote-sensing data sets. *Journal of Time Series Analysis*, 32:430–446, 2011.

M. Katzfuss and D. Hammerling. Parallel inference for massive distributed spatial data using low-rank models. *Statistics and Computing*, 27:363–375, 2017.

F. Lindgren and H. Rue. Bayesian spatial modelling with R-INLA. *Journal of Statistical Software*, 63:1–25, 2015.

H. Nguyen, N. Cressie, and A. Braverman. Spatial statistical data fusion for remote sensing applications. *Journal of the American Statistical Association*, 107:1004–1018, 2012.

H. Nguyen, M. Katzfuss, N. Cressie, and A. Braverman. Spatio-temporal data fusion for very large remote sensing datasets. *Technometrics*, 56:174–185, 2014.

D. Nychka, S. Bandyopadhyay, D. Hammerling, F. Lindgren, and S. Sain. A multiresolution Gaussian process model for the analysis of large spatial datasets. *Journal of Computational and Graphical Statistics*, 24: 579–599, 2015.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, MA, 2006.

T. Shi and N. Cressie. Global statistical analysis of MISR aerosol data: a massive data product from NASA's Terra satellite. *Environmetrics*, 18:665–680, 2007.

M.L. Stein. A modeling approach for large spatial datasets. *Journal of the Korean Statistical Society*, 37: 3–10, 2008.

C. K. Wikle and L. M. Berliner. Combining information across spatial scales. *Technometrics*, 47:80–91, 2005.

A. Zammit-Mangion, G. Sanguinetti, and V. Kadirkamanathan. Variational estimation in spatiotemporal systems from continuous and point-process observations. *IEEE Transactions on Signal Processing*, 60: 3449–3459, 2012.

A. Zammit-Mangion, J. Rougier, N. Schön, F. Lindgren, and J. Bamber. Multivariate spatio-temporal modelling for assessing Antarctica's present-day contribution to sea-level rise. *Environmetrics*, 26:159–177, 2015.