# Fixed Rank Kriging: The `R` package

Andrew Zammit-Mangion and Noel Cressie

February 5, 2017

**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 (**?**). For a detailed treatment of FRK, see **???**, and **?**.

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** (**?**). **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 **?**, 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.

**?** develop Bayesian FRK; they keep the spatial basis functions fixed and put a prior distribution on $\mathbf{K}$. The predictive-process approach of **?** 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 **?**, for an equivalence argument). An R package that implements predictive processes is **spBayes** (**?**). 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., **?**). 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. We also find that when using sparse partial-matrix inversion approaches outlined in **?**, we are able to carry out exact predictions (without conditional simulation) at millions of prediction locations with relative ease. 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, as will be seen in this article, 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** (**?**), which we term *vanilla* FRK (FRK-V), there is an explicit reliance on multi-resolution basis functions to give complex non-stationary spatial patterns at the cost of not imposing

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

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., **?**), 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., **??**). 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 $D$ is our domain of interest. In the following, 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,

$$Y(\mathbf{s}) = \mathbf{t}(\mathbf{s})'\boldsymbol{\alpha} + \upsilon(\mathbf{s}) + \xi(\mathbf{s}); \quad \mathbf{s} \in D, \tag{1}$$

where, for $\mathbf{s} \in D$, $\mathbf{t}(\mathbf{s})$ is a vector of spatially-referenced covariates, $\boldsymbol{\alpha}$ are regression coefficients, $\upsilon(\mathbf{s})$ is a small-scale, spatially correlated random effect, and $\xi(\mathbf{s})$ is a fine-scale, uncorrelated, random effect. It is natural to let $\mathrm{E}(\upsilon(\cdot)) = \mathrm{E}(\xi(\cdot)) = 0$. Define $\lambda(\cdot) \equiv \upsilon(\cdot) + \xi(\cdot)$. 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 $\boldsymbol{\eta} \equiv (\eta_1, \ldots, \eta_r)'$ is an $r$-variate random vector, and $\boldsymbol{\phi}(\cdot) \equiv (\phi_1(\cdot), \ldots, \phi_r(\cdot))'$ is an $r$-dimensional vector of pre-specified spatial basis functions. Sometimes, $\boldsymbol{\phi}(\cdot)$ contains basis functions and multiple resolutions (e.g., wavelets) and its elements may or may not have compact support. The basis chosen should be able to adequately reconstruct realisations of $Y(\cdot)$; an empirical spectral-based approach that can ensure this is discussed in **?**.

In order to cater for different observation supports, it is convenient to assume a domain of interest $D^G \equiv \{A_i \subset D : i = 1, \ldots, N\}$ that contains $N$ small, non-overlapping BAUs (**?**), and $D = \bigcup_{i=1}^{N} A_i$. The set $D^G$ of BAUs could be thought of as 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 $\mathbf{Y} = (Y_i : i = 1, \ldots, N)'$, where

$$Y_i \equiv \frac{1}{|A_i|} \int_{A_i} Y(\mathbf{s}) \mathrm{d}\mathbf{s}; \quad i = 1, \ldots, N, \tag{2}$$

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

$$Y_i = \mathbf{t}_i'\boldsymbol{\alpha} + \upsilon_i + \xi_i, \tag{3}$$

where $\mathbf{t}_i \equiv \frac{1}{|A_i|} \int_{A_i} \mathbf{t}(\mathbf{s}) \mathrm{d}\mathbf{s}$, $\upsilon_i \equiv \frac{1}{|A_i|} \int_{A_i} \upsilon(\mathbf{s}) \mathrm{d}\mathbf{s}$, and $\xi_i$ is specified below. The SRE model specifies that $\upsilon(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})'\boldsymbol{\eta}; \mathbf{s} \in D$, and hence in terms of the discretisation onto $D^G$,

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

so that $\boldsymbol{v} = \mathbf{S}\boldsymbol{\eta}$, where $\mathbf{S}$ is the $N \times r$ matrix

$$\mathbf{S} \equiv \left( \frac{1}{|A_i|} \int_{A_i} \boldsymbol{\phi}(\mathbf{s}) \mathrm{d}\mathbf{s} : i = 1, \ldots, N \right)'. \tag{4}$$

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. 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. Frequently, the resolution of the BAUs is sufficiently fine, and the basis functions are sufficiently smooth, so that $\mathbf{S}$ can be approximated:

$$\mathbf{S} \approx (\boldsymbol{\phi}(\mathbf{s}_i) : i = 1, \ldots, N)', \tag{5}$$

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

$$\mathrm{var}(\xi_i) = \sigma^2_\xi v_{\xi,i}, \tag{6}$$

where $\sigma^2_\xi$ is a parameter to be estimated, and $\{v_{\xi,1}, \ldots, v_{\xi,N}\}$ are known and represent heteroscedasticity. We further assume that the variables representing the discretised fine-scale variation, $\{\xi_i : i = 1, \ldots, N\}$, are independent. From (3), we can write

$$\mathbf{Y} = \mathbf{T}\boldsymbol{\alpha} + \mathbf{S}\boldsymbol{\eta} + \boldsymbol{\xi}, \tag{7}$$

where $\mathbf{T} \equiv (\mathbf{t}_i : i = 1, \ldots, N)'$, $\boldsymbol{\xi} \equiv (\xi_i : i = 1, \ldots, N)'$, and $\mathrm{var}(\boldsymbol{\xi}) \equiv \sigma^2_\xi \mathbf{V}_\xi$, where $\mathbf{V}_\xi \equiv \mathrm{diag}(v_{\xi,1}, \ldots, v_{\xi,N})$ and known.

We now assume that the unobserved 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 $N > m$ 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 subset of $2^{\{1,\ldots,N\}}$, the power set of $\{1, \ldots, N\}$. 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 $B_1 = A_1 \cup A_2$ and $B_2 = A_3$. Catering for different footprints is important for remote sensing applications in which satellite-instrument footprints can widely differ (e.g., **??**).

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

$$Z_j \equiv Z(B_j) = \left( \frac{\sum_{i=1}^N Y_i w_{ij}}{\sum_{i=1}^N w_{ij}} \right) + \left( \frac{\sum_{i=1}^N \delta_i w_{ij}}{\sum_{i=1}^N w_{ij}} \right) + \epsilon_j; \quad B_j \in D^O, \tag{8}$$

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 BAUs of equal area are assumed. The random quantities $\{\delta_i\}$ and $\{\epsilon_i\}$ capture the imperfections of the measurement. Better known is the measurement error $\epsilon_i$, which is assumed to be zero-mean Gaussian distributed. The component $\delta_i$ captures any bias in the measurement at the BAU-level, which has the interpretation of a systematic error, and $\{\delta_i\}$ are independent.

$$\mathrm{var}(\delta_i) = \sigma^2_\delta v_{\delta,i}, \tag{9}$$

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

We assume that the observations are conditionally independent, when conditioned on $\mathbf{Y}$ *and* $\boldsymbol{\delta}$. Equivalently the measurement errors $\{\epsilon_j : j = 1, \ldots, m\}$, where $m \equiv |D^O|$ is the number of observations, are

independent. Represent the data as $\mathbf{Z} \equiv (Z_j : j = 1, \ldots, m)'$. 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

$$\mathbf{Z} = \mathbf{C}_Z \mathbf{Y} + \mathbf{C}_Z \boldsymbol{\delta} + \boldsymbol{\varepsilon}, \tag{10}$$

where $\boldsymbol{\varepsilon} \equiv (\epsilon_j : j = 1, \ldots, m)'$, and $\text{var}(\boldsymbol{\varepsilon}) = \boldsymbol{\Sigma}_\epsilon \equiv \sigma^2_\epsilon \mathbf{V}_\epsilon$ is a diagonal covariance matrix. The matrix $\boldsymbol{\Sigma}_\epsilon$ is assumed known from the properties of the measurement process. If it is not known, $\mathbf{V}_\epsilon$ is fixed to $\mathbf{I}$ and $\sigma^2_\epsilon$ is estimated using variogram techniques (**?**).

The rows of the matrix $\mathbf{C}_Z$ sum to 1. It will be convenient, as a consequence of conditional independence, to re-write

$$\mathbf{Z} = \mathbf{T}_Z \boldsymbol{\alpha} + \mathbf{S}_Z \boldsymbol{\eta} + \boldsymbol{\xi}_Z + \boldsymbol{\delta}_Z + \boldsymbol{\varepsilon}, \tag{11}$$

where $\mathbf{T}_Z \equiv \mathbf{C}_Z \mathbf{T}$, $\mathbf{S}_Z \equiv \mathbf{C}_Z \mathbf{S}$, $\boldsymbol{\xi}_Z \equiv \mathbf{C}_Z \boldsymbol{\xi}$, $\boldsymbol{\delta}_Z \equiv \mathbf{C}_Z \boldsymbol{\delta}$, and where $\text{var}(\boldsymbol{\xi}_Z) = \sigma^2_\xi \mathbf{V}_{\xi,Z} \equiv \sigma^2_\xi \mathbf{C}_Z \mathbf{V}_\xi \mathbf{C}'_Z$ and $\text{var}(\boldsymbol{\delta}_Z) = \sigma^2_\delta \mathbf{V}_{\delta,Z} \equiv \sigma^2_\delta \mathbf{C}_Z \mathbf{V}_\delta \mathbf{C}'_Z$. Then

$$\begin{aligned} \mathbb{E}(\mathbf{Z}) &= \mathbf{T}_Z \boldsymbol{\alpha} + \mathbf{S}_Z \boldsymbol{\eta}, \\ \text{var}(\mathbf{Z}) &= \mathbf{S}_Z \mathbf{K} \mathbf{S}'_Z + \sigma^2_\xi \mathbf{C}_Z \mathbf{V}_\xi \mathbf{C}'_Z + \sigma^2_\delta \mathbf{C}_Z \mathbf{V}_\delta \mathbf{C}'_Z + \sigma^2_\epsilon \mathbf{V}_\epsilon \ . \end{aligned}$$

In practice, it is not always possible for each $B_i$ to include entire BAUs. For simplicity, 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 also supplied. In this case each data point is attributed to a specific BAU and it is possible to have multiple observations of 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 when $\mathbf{K} = \mathbf{K}_\circ(\boldsymbol{\vartheta})$; their estimation is the subject of Section 2.2. If the parameters in $\boldsymbol{\theta}$ are known, a straightforward inversion 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 $\boldsymbol{\theta}_\circ$ are substituted in to 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 subset of $2^{\{1,\ldots,N\}}$ 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. 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 where the fine-scale variation sits, either in the observation model (8) (in which case $\sigma^2_\xi$ is fixed to 0) or in the process model (3) (in which case $\sigma^2_\delta$ 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 obtained by simply replacing the subscript $\xi$ with $\delta$. However, which case is chosen has a considerable impact on the prediction equations (Section 2.3). Recall that the measurement-error covariance matrix $\boldsymbol{\Sigma}_\epsilon$ is assumed known from measurement characteristics, or estimated using variogram techniques prior to estimating the remaining parameters described below.

Consider the case where $\sigma^2_\delta = 0$. We carry out parameter estimation using an EM algorithm (**??**) with (11) 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. In the E-step, the function

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(l)}) \equiv \mathbb{E}(\ln L_c(\boldsymbol{\theta}) \mid \mathbf{Z}, \boldsymbol{\theta}^{(l)}), \tag{12}$$

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

$$\boldsymbol{\theta}^{(l+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(l)}), \tag{13}$$

is found. It is straightforward to show that, conditionally,

$$\boldsymbol{\eta} \mid \mathbf{Z}, \boldsymbol{\theta}^{(l)} \sim \text{Gau}(\boldsymbol{\mu}_\eta^{(l)}, \boldsymbol{\Sigma}_\eta^{(l)}), \tag{14}$$

where

$$\boldsymbol{\mu}_\eta^{(l)} = \boldsymbol{\Sigma}_\eta^{(l)} \mathbf{S}_Z' (\mathbf{D}_Z^{-1})^{(l)} (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l)}), \tag{15}$$

$$\boldsymbol{\Sigma}_\eta^{(l)} = (\mathbf{S}_Z' (\mathbf{D}_Z^{-1})^{(l)} \mathbf{S}_Z + (\mathbf{K}^{-1})^{(l)})^{-1}, \tag{16}$$

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

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

$$\boldsymbol{\alpha}^{(l+1)} = (\mathbf{T}_Z' (\mathbf{D}_Z^{-1})^{(l+1)} \mathbf{T}_Z)^{-1} \mathbf{T}_Z' (\mathbf{D}_Z^{-1})^{(l+1)} (\mathbf{Z} - \mathbf{S}_Z \boldsymbol{\mu}_\eta^{(l)}). \tag{17}$$

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

$$\mathbf{K}^{(l+1)} = \boldsymbol{\Sigma}_\eta^{(l)} + \boldsymbol{\mu}_\eta^{(l)} \boldsymbol{\mu}_\eta^{(l)'}, \tag{18}$$

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

$$\boldsymbol{\vartheta}^{(l+1)} = \arg\max_{\boldsymbol{\vartheta}} \ln |\mathbf{K}_\circ(\boldsymbol{\vartheta})^{-1}| - \text{tr}(\mathbf{K}_\circ(\boldsymbol{\vartheta})^{-1} (\boldsymbol{\Sigma}_\eta^{(l)} + \boldsymbol{\mu}_\eta^{(l)} \boldsymbol{\mu}_\eta^{(l)'})). \tag{19}$$

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

The update for $\sigma_\xi^2$ requires the solution to

$$\text{tr}((\boldsymbol{\Sigma}_\epsilon + (\sigma_\xi^2)^{(l+1)} \mathbf{V}_{\xi,Z})^{-1} \mathbf{V}_{\xi,Z}) = \text{tr}((\boldsymbol{\Sigma}_\epsilon + (\sigma_\xi^2)^{(l+1)} \mathbf{V}_{\xi,Z})^{-1} \mathbf{V}_{\xi,Z} (\boldsymbol{\Sigma}_\epsilon + (\sigma_\xi^2)^{(l+1)} \mathbf{V}_{\xi,Z})^{-1} \boldsymbol{\Omega}), \tag{20}$$

where

$$\boldsymbol{\Omega} \equiv \mathbf{S}_Z \boldsymbol{\Sigma}_\eta^{(l)} \mathbf{S}_Z' + \mathbf{S}_Z \boldsymbol{\mu}_\eta^{(l)} \boldsymbol{\mu}_\eta^{(l)'} \mathbf{S}_Z' - 2 \mathbf{S}_Z \boldsymbol{\mu}_\eta^{(l)} (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l+1)})' + (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l+1)}) (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l+1)})'. \tag{21}$$

The solution to (20) is also found numerically using `optim` after the expression for $\boldsymbol{\alpha}^{(l+1)}$ is substituted into (21). After $(\sigma^2)^{(l+1)}$ is found, $\boldsymbol{\alpha}^{(l+1)}$ is found from (17). 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 $\mathbf{V}_Z$ and $\boldsymbol{\Sigma}_\epsilon$ are proportional to the identity matrix, with constants of proportionality $\gamma_1$ and $\gamma_2$, respectively. In this case,

$$(\sigma_\xi^2)^{(l+1)} = \frac{1}{\gamma_1} \left( \frac{\text{tr}(\boldsymbol{\Omega})}{m} - \gamma_2 \right), \tag{22}$$

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 $\boldsymbol{\mu}_\eta^{(l)}$. These simplifications are used by **FRK** when possible.

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

$$\ln[\mathbf{Z} \mid \boldsymbol{\alpha}^{(l)}, \mathbf{K}^{(l)}, (\sigma_\xi^2)^{(l)}] = -\frac{m}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{\Sigma}_Z^{(l)}| - \frac{1}{2} (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l)})' (\boldsymbol{\Sigma}_Z^{-1})^{(l)} (\mathbf{Z} - \mathbf{T}_Z \boldsymbol{\alpha}^{(l)}), \tag{23}$$

where

$$\boldsymbol{\Sigma}_Z^{(l)} = \mathbf{S}_Z \mathbf{K}^{(l)} \mathbf{S}_Z' + \mathbf{D}_Z^{(l)}, \tag{24}$$

and recall that $\mathbf{D}_{\xi,Z}^{(l)} \equiv (\sigma_\xi^2)^{(l)} \mathbf{V}_Z + \boldsymbol{\Sigma}_\epsilon$. 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., **?**). Specifically, the operations

$$(\boldsymbol{\Sigma}_Z^{-1})^{(l)} = (\mathbf{D}_Z^{-1})^{(l)} - (\mathbf{D}_Z^{-1})^{(l)} \mathbf{S}_Z ((\mathbf{K}^{-1})^{(l)} + \mathbf{S}_Z' (\mathbf{D}_Z^{-1})^{(l)} \mathbf{S}_Z)^{-1} \mathbf{S}_Z' (\mathbf{D}_Z^{-1})^{(l)}, \tag{25}$$

$$|\boldsymbol{\Sigma}_Z^{(l)}| = |(\mathbf{K}^{-1})^{(l)} + \mathbf{S}_Z' (\mathbf{D}_Z^{-1})^{(l)} \mathbf{S}_Z| |\mathbf{K}^{(l)}| |\mathbf{D}_Z^{(l)}|, \tag{26}$$

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

## 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 $\{Y_P(B_k) : k = 1, \ldots, N_P\}$, which, 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 places, and is equal to $|D^P|$. Then,

$$Y_{P,k} \equiv Y_P(B_k) = \left( \frac{\sum_{i=1}^N Y_i \tilde{w}_{ik}}{\sum_{i=1}^N \tilde{w}_{kj}} \right); \quad B_k \in D^P. \tag{27}$$

where the weights,

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

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

$$\mathbf{C}_P \equiv \left( \frac{\tilde{w}_{ik}}{\sum_{l=1}^N \tilde{w}_{lk}} : i = 1, \ldots, N; k = 1, \ldots, N_P \right), \tag{28}$$

the rows of which sum to one, such that

$$\mathbf{Y}_P = \mathbf{C}_P \mathbf{Y} = \mathbf{T}_P \boldsymbol{\alpha} + \mathbf{S}_P \boldsymbol{\eta} + \boldsymbol{\xi}_P, \tag{29}$$

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 $\text{var}(\boldsymbol{\xi}_P) = \sigma_\xi^2 \mathbf{C}_P \mathbf{V}_\xi \mathbf{C}_P' \equiv \sigma_\xi^2 \mathbf{V}_{\xi,P}$. As with the observations, these prediction regions may overlap. In practice, it is not always possible for each $B_i$ 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 prediction area.

Let $l^*$ denote the EM iteration number at which convergence was reached. The final estimates are then $\widehat{\boldsymbol{\mu}}_\eta \equiv \boldsymbol{\mu}_\eta^{(l^*)}, \widehat{\boldsymbol{\Sigma}}_\eta \equiv \boldsymbol{\Sigma}_\eta^{(l^*)}, \widehat{\boldsymbol{\alpha}} \equiv \boldsymbol{\alpha}^{(l^*)}, \widehat{\mathbf{K}} \equiv \mathbf{K}^{(l^*)}, \widehat{\sigma}_\xi^2 \equiv (\sigma_\xi^2)^{(l^*)}$, and $\widehat{\sigma}_\delta^2 \equiv (\sigma_\delta^2)^{(l^*)}$. Recall from Section 2.2 that the user needs to attribute excess fine-scale variation to either the measurement process or the physical process. This leads to the following two cases.

**Case 1: $\sigma_\xi^2 = 0$.** 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

$$\widehat{\mathbf{Y}}_P \equiv \mathbb{E}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{T}_P \widehat{\boldsymbol{\alpha}} + \mathbf{S}_P \widehat{\boldsymbol{\mu}}_\eta, \tag{30}$$

$$\boldsymbol{\Sigma}_{Y_P|Z} \equiv \text{var}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{S}_P \widehat{\boldsymbol{\Sigma}}_\eta \mathbf{S}_P'. \tag{31}$$

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

**Case 2: $\sigma_\delta^2 = 0$ (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 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 $\mathbf{W} \equiv (\boldsymbol{\eta}', \boldsymbol{\xi}')'$ and $\boldsymbol{\Pi} \equiv (\mathbf{S}, \mathbf{I})$. Then (7) can be re-written as $\mathbf{Y} = \mathbf{T}\boldsymbol{\alpha} + \boldsymbol{\Pi}\mathbf{W}$ and

$$\widehat{\mathbf{Y}} \equiv \mathbb{E}(\mathbf{Y} \mid \mathbf{Z}) = \mathbf{T}\widehat{\boldsymbol{\alpha}} + \boldsymbol{\Pi}\widehat{\mathbf{W}}, \tag{32}$$

$$\boldsymbol{\Sigma}_{Y|Z} \equiv \text{var}(\mathbf{Y} \mid \mathbf{Z}) = \boldsymbol{\Pi}\boldsymbol{\Sigma}_W \boldsymbol{\Pi}', \tag{33}$$

where

$$\widehat{\mathbf{W}} \equiv \boldsymbol{\Sigma}_W \boldsymbol{\Pi}' \mathbf{C}_Z' \boldsymbol{\Sigma}_\epsilon^{-1} (\mathbf{Z} - \mathbf{T}_Z \widehat{\boldsymbol{\alpha}}), \tag{34}$$

$$\boldsymbol{\Sigma}_W \equiv (\boldsymbol{\Pi}' \mathbf{C}_Z' \boldsymbol{\Sigma}_\epsilon^{-1} \mathbf{C}_Z \boldsymbol{\Pi} + \boldsymbol{\Lambda}^{-1})^{-1}, \tag{35}$$

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

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

$$\text{var}(\mathbf{Y}_P \mid \mathbf{Z}) = \mathbf{C}_P\boldsymbol{\Sigma}_{Y|Z}\mathbf{C}_P'. \tag{36}$$

Note that for Case 2. we need to predict $\boldsymbol{\xi}_P$, which is not available at this stage. Since we generally wish to predict at locations other than where we have observations, this quantity is rarely equal to $\boldsymbol{\xi}_Z$; for this reason, $\boldsymbol{\xi}_Z$ was not estimated in Section 2.2.

By carrying out all intensive linear algebraic operations on small, dense matrices, up to this point all equations have moderate computational and memory complexity. On the other hand, (36) indicates that all of $\boldsymbol{\Sigma}_{Y|Z}$ (and hence $\boldsymbol{\Sigma}_W$) needs to be computed and stored. In reality, only a few elements of $\boldsymbol{\Sigma}_{Y|Z}$ need to be known and, thus, computational simplifications are possible by using sparse inverse methods outlined in **?**, which are used in **FRK**. However, the larger the number of BAUs needed for aggregation (encoded in $\mathbf{C}_P$), the more the elements that need to be computed in $\boldsymbol{\Sigma}_{Y|Z}$ and the slower the prediction.

# 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., **?**). 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
```

The mesher in **INLA** is also required for this vignette. Please install **INLA** by visiting `http://www.r-inla.org/download`.

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.

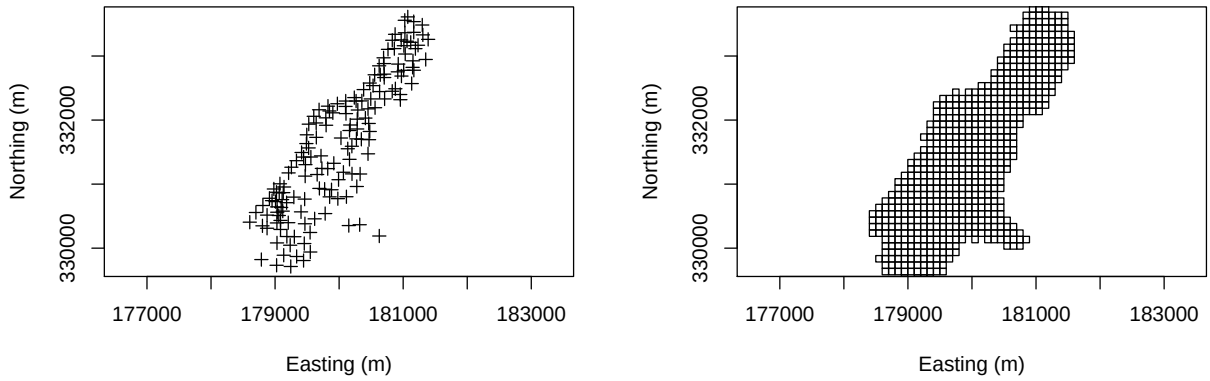
## 3.1 The `meuse` dataset

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:




(a) (b)

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

```
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
```

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 $100 \times 100$ (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 spatial-domain boundary (see `INLA::inla.nonconvex.hull` for more details). For the $i$-th BAU, we also need to attribute the element $v_i$ that describes the hetereoscedasticity 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, $\sigma^2_\xi$ or $\sigma^2_\delta$ (depending on the chosen model); this constant is estimated using maximum likelihood with `SRE.predict` 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:

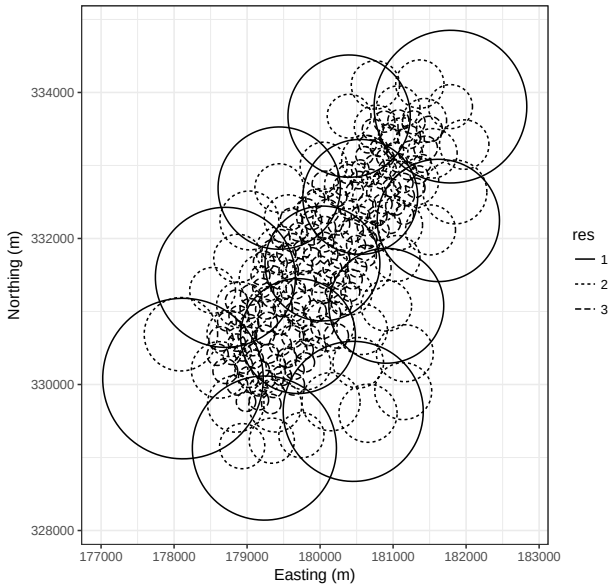



Figure 2: Basis functions automatically generated for the meuse dataset with 2 resolutions. The interpretation of the circles change with the domain and basis. For Gaussian functions on the plane, each circle is centred at the basis function centre, and has a radius equal to $1\sigma$. Type `help(auto_basis)`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

```
G <- auto_basis(manifold = plane(),    # 2D plane
                data=meuse,            # meuse data
                nres = 3,              # number of resolutions
                prune = 5,             # prune threshold
                type = "Gaussian",     # type of basis function
                regular = 0)           # place irregularly 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). Basis functions do not have to be positive-definite and are constructed such that the process is accurately represented where data is present, and not represented otherwise. This is controlled by the parameter `prune`, see `help(auto_basis)` for details. 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

## Note:  show_basis assumes spherical distance functions when plotting
```

**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

## Loading required namespace:  gstat
```

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 (**?**). 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, 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 5 to minimise compilation-time of this vignette; see Fig. 3.

```
S <- SRE.fit(SRE_model = S,     # SRE model
             n_EM = 5,          # 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

## [1] "Maximum EM iterations reached"
```

**Step 6:** Finally, we predict at all the BAUs with the fitted model. This is done using the function `SRE.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 <- SRE.predict(SRE_model = S,          # SRE model
                         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

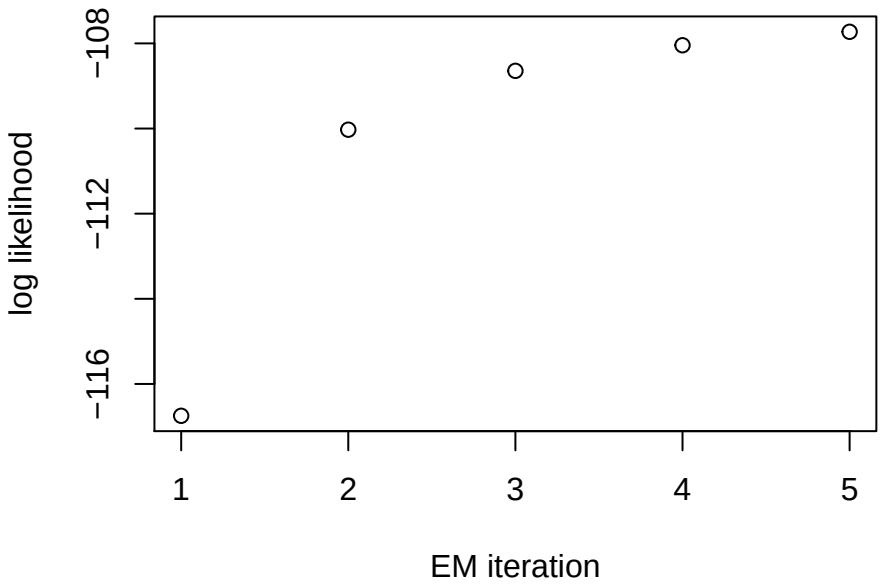


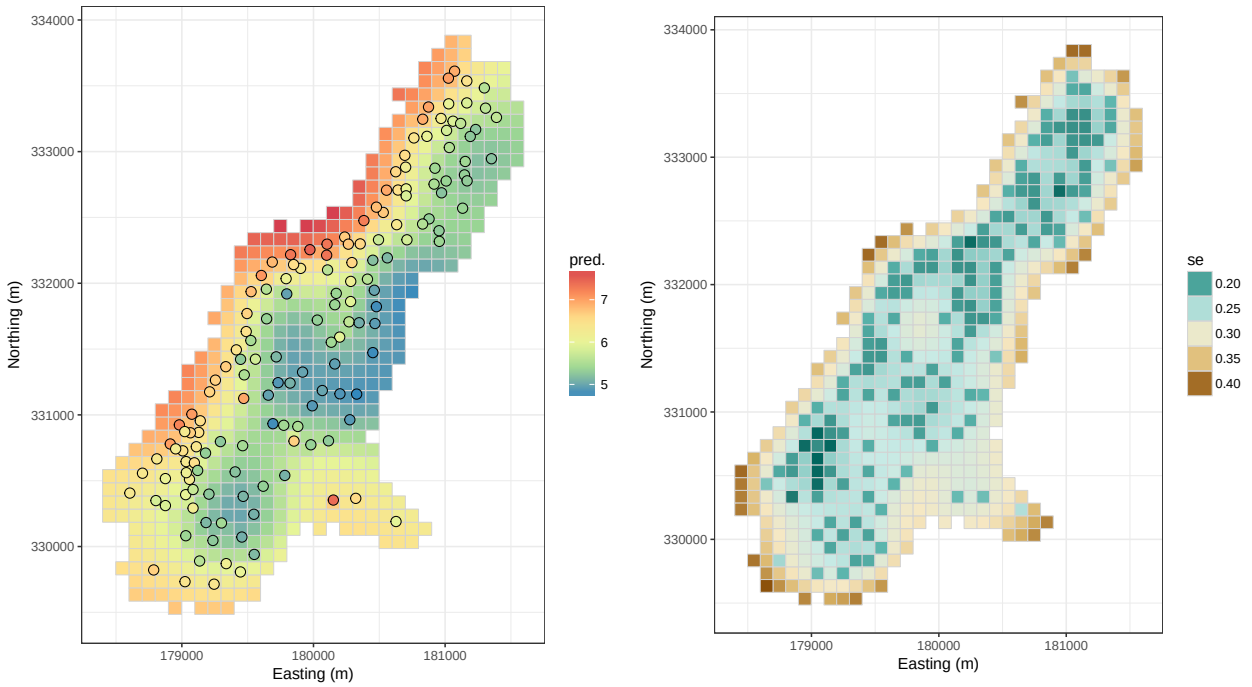

Figure 3: Convergence of the EM algorithm when using `FRK` with the `meuse` dataset.

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)),
```




(a) (b)

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

```
                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()
```

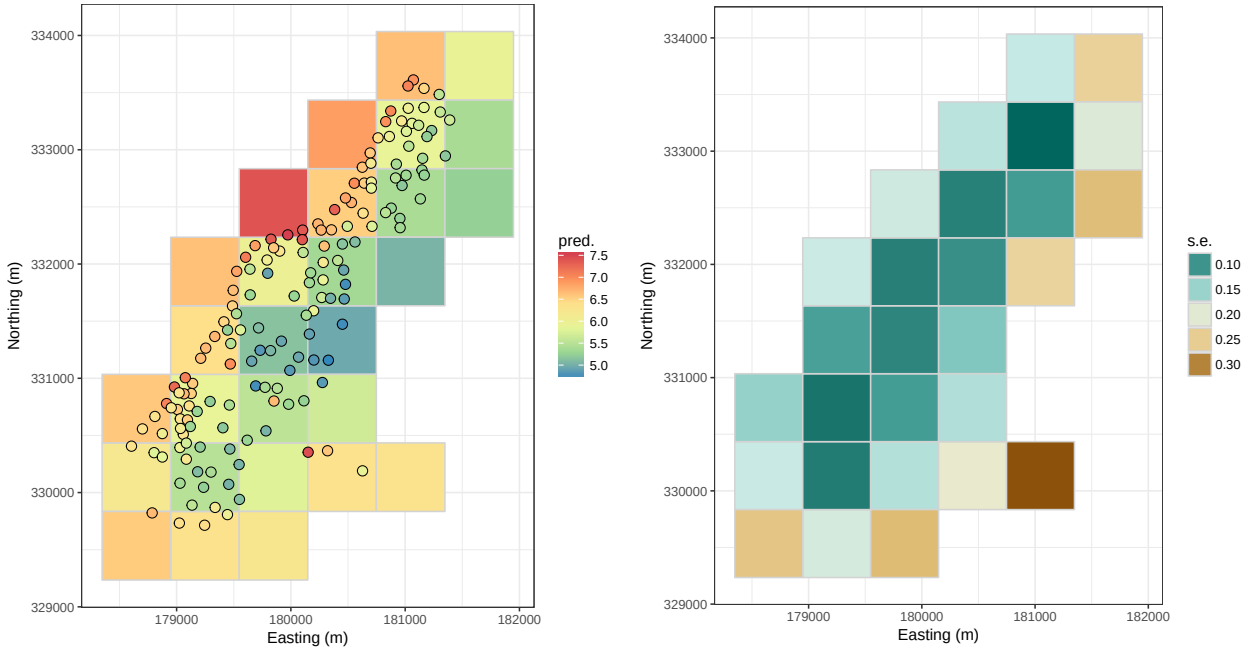
Now, assume that we wish to predict over regions encompassing several BAUs such that the matrix $\mathbf{C}_P$ containes multiple non-zeros per row. Then we need to set the `pred_polys` argument in the function `auto_BAUs`. First, we create this larger regionalisation as follows

```
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
```

and carry out prediction on the larger polygons:

```
Pred_regions <- SRE.predict(SRE_model = S,                # SRE model
                            pred_polys = Pred_regions,    # prediction polygons
                            obs_fs = FALSE)
```

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




(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.

```
## Joining, by = "id"
```

## 3.2 The `AIRS` dataset

Modelling on the sphere proceeds in a very similar fashion to the plane, except that an unprojected coordinate reference system (CRS) for the data needs to be declared on the sphere. 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) =
            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.

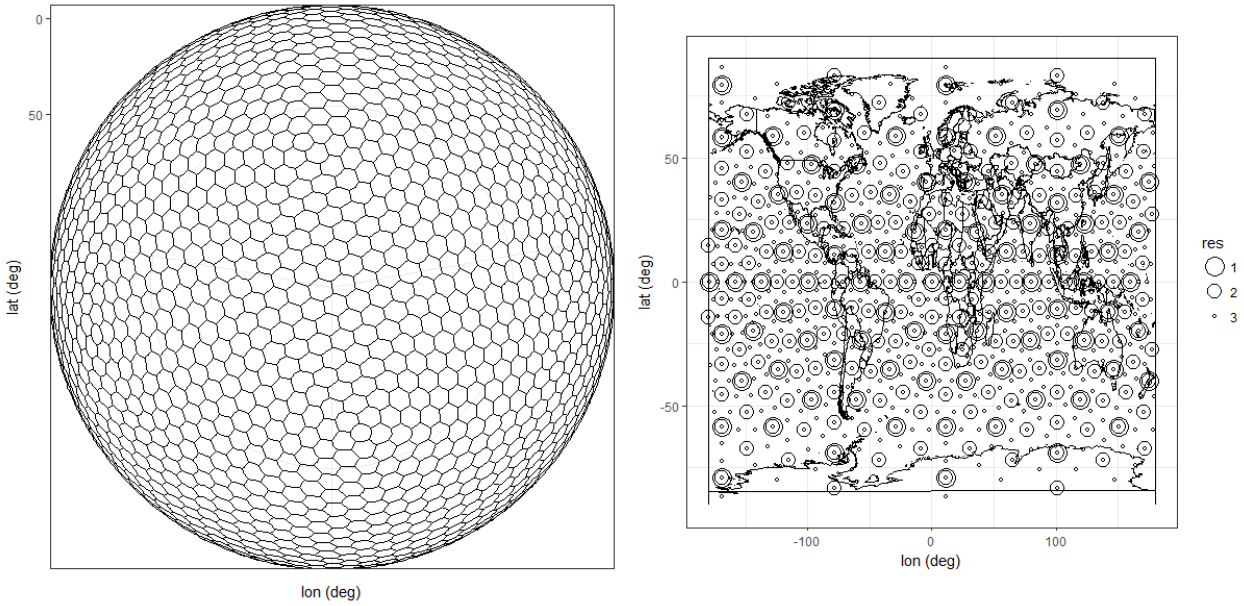
```
isea3h_sp_poldf <- auto_BAUs(manifold   = sphere(),  # model on sphere
                             isea3h_res = 6,         # isea3h resolution 6 BAUs
                             type = "hex",           # hexagonal grid
                             data = AIRS_05_2003)    # remove BAUs where there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

## Error in resolve_vars(new_groups, tbl_vars(.data)):  unknown variable to group by :  id

isea3h_sp_poldf$fs = 1                               # fine-scale component

## Error in isea3h_sp_poldf$fs = 1:  object 'isea3h_sp_poldf' not found
```

**Step 3:** Now we construct the basis functions, this time of type `"bisquare"` with three resolutions. We supply the data and the argument `prune` so that basis functions in regions where there are no data are omitted from the final set. We also introduce a new argument, `subsamp`. This argument 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.

```
G <- auto_basis(manifold = sphere(), # basis functions on the sphere
                data=AIRS_05_2003,   # AIRS data
                nres = 3,            # number of resolutions
                prune = 0,            # prune threshold
                type = "bisquare",   # bisquare function
                subsamp = 20000)     # prune using 20000 data points (at random)
```

**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 = 4` 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
```




(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`.

```
         average_in_BAU = TRUE)  # summarise data

## Error in is(BAUs, "Spatial"):  object 'isea3h_sp_poldf' not found

S <- SRE.fit(SRE_model = S,     # SRE model
             n_EM = 2,          # 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

## [1] "Maximum EM iterations reached"
```

**Step 6:** We now predict at the BAU level but this time ensure that `obs_fs = TRUE`, which indicates that we are forcing $\sigma^2_\xi$ 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 <- SRE.predict(SRE_model = 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–??.

```
## Error in cbind(sp_polys@polygons[[i]]@Polygons[[1]]@coords, id = polynames[i]):  no slot
of name "polygons" for this object of class "SpatialPixelsDataFrame"
## Error in eval(expr, envir, enclos):  object 'X' not found
## Error in eval(expr, envir, enclos):  object 'X' not found
```

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

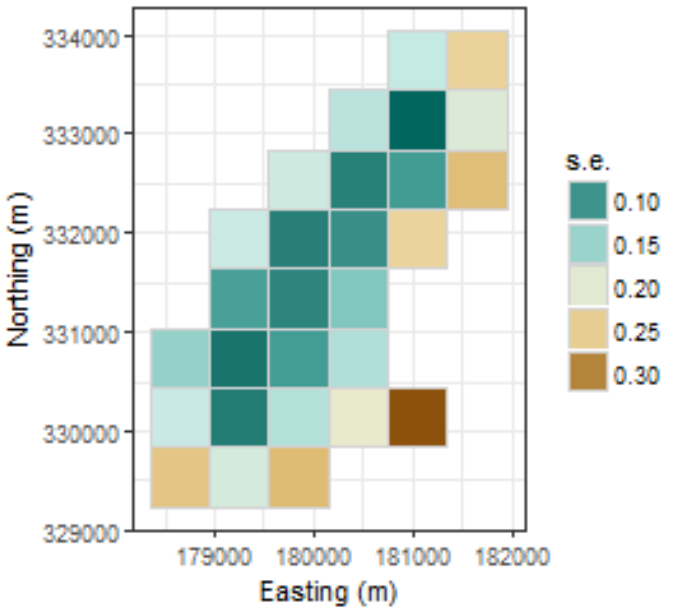



Figure 8: Prediction of $\mathbf{Y}_P$ in ppm following FRK on the `AIRS` data.

```
## Error in pmax(co2avgret, mumin):  object 'mumin' not found
```

```
## Error in fortify(data):  object 'X' not found
```

```
## Error in pmax(sqrt(X$var), 0.32):  object 'X' not found
## Error in fortify(data):  object 'X' not found
```

```
## Error in print(g3):  object 'g3' not found
```

# 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, only point-referenced spatio-temporal data can be used (that will be subsequently mapped to BAUs), 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 (**?**) 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. 9. 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:

```
library(spacetime)
```

## 4.1 The `NOAA` dataset

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

```
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
```

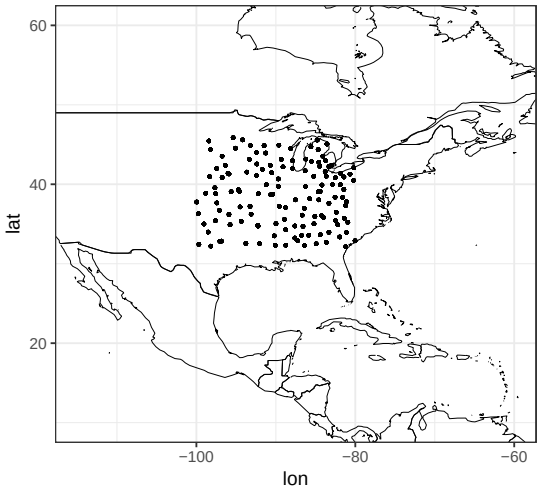



Figure 9: Station locations from which the maximum temperature readings in the `NOAA` dataset were obtained.

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):

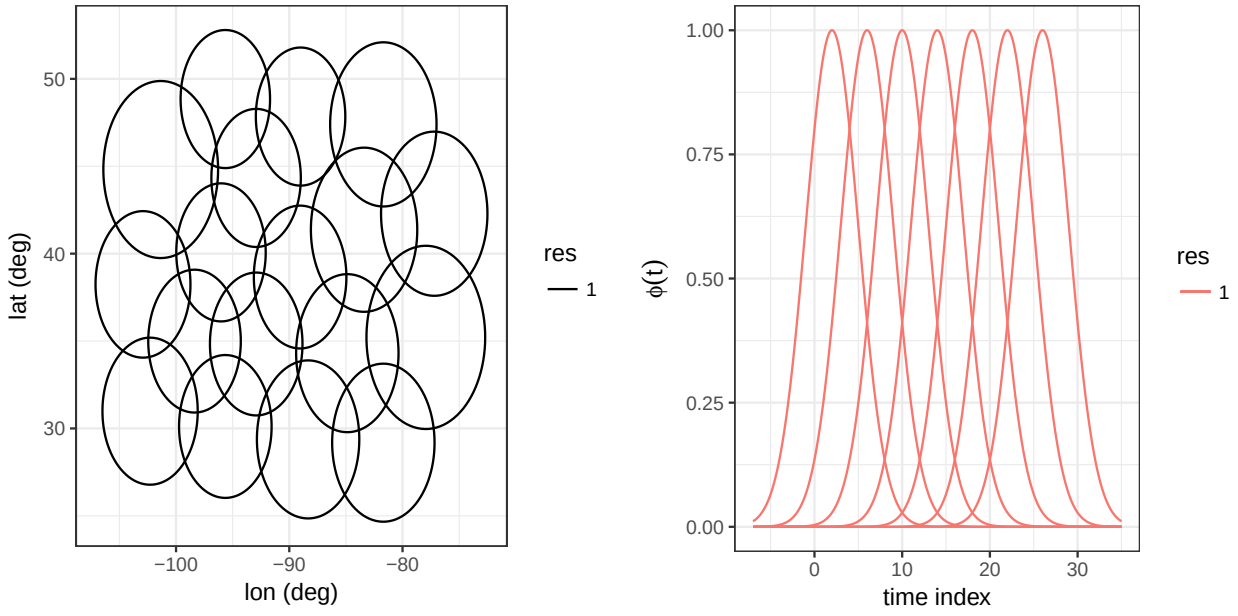
```
STObj$std <- 2
```

**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 × 1 deg. longitude × 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
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 = 0)
```




(a) (b)

Figure 10: 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.

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.

```
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 10:

```
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 `SRE.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(SRE_model = S,     # estimate parameters in the SRE model S
             n_EM = 2,          # maximum no. of EM iterations
             tol = 0.1,         # tolerance on log-likelihood
             print_lik=FALSE)   # print log-likelihood trace

grid_BAUs <- SRE.predict(SRE_model = S,         # SRE model
                         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. 11 and 12, 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
```

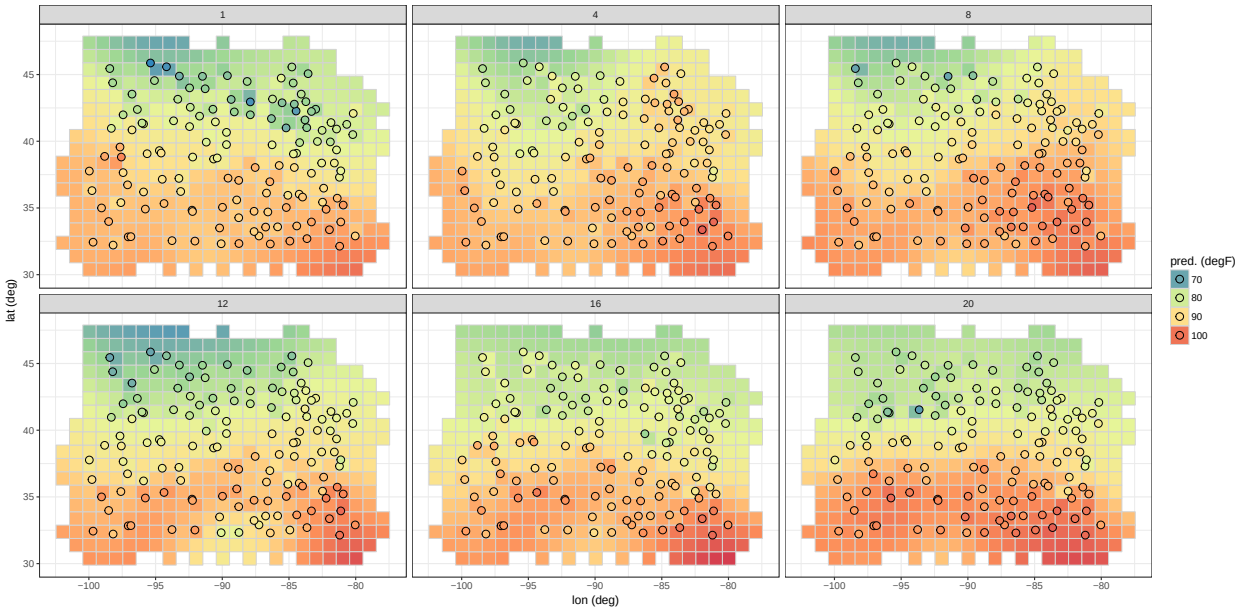



Figure 11: Spatio-temporal FRK prediction of `Tmax` on the plane in degrees Fahrenheit within a domain enclosing the region of interest for six selected days spanning the temporal window of the data: 01 July 1993 – 20 July 2003.

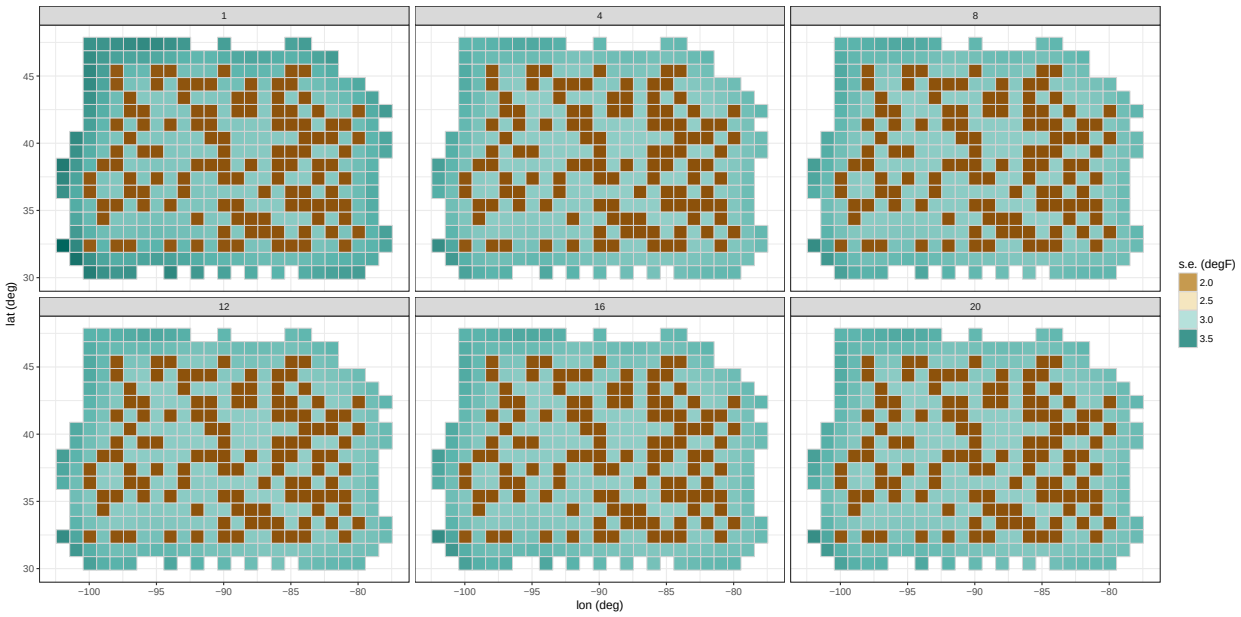



Figure 12: Spatio-temporal FRK prediction standard error of `Tmax` on the plane in degrees Fahrenheit within a domain enclosing the region of interest for the same six days selected in Fig. 11 and spanning the temporal window of the data, 01 July 1993 – 20 July 2003.

```
## Error in over(sphere_BAUs, conv_hull):  package rgeos is required for additional over methods
```

```
G_spatial <- auto_basis(manifold = sphere(),      # spatial functions on the plane
                        data=as(STObj,"Spatial"),  # remove the temporal dimension
```

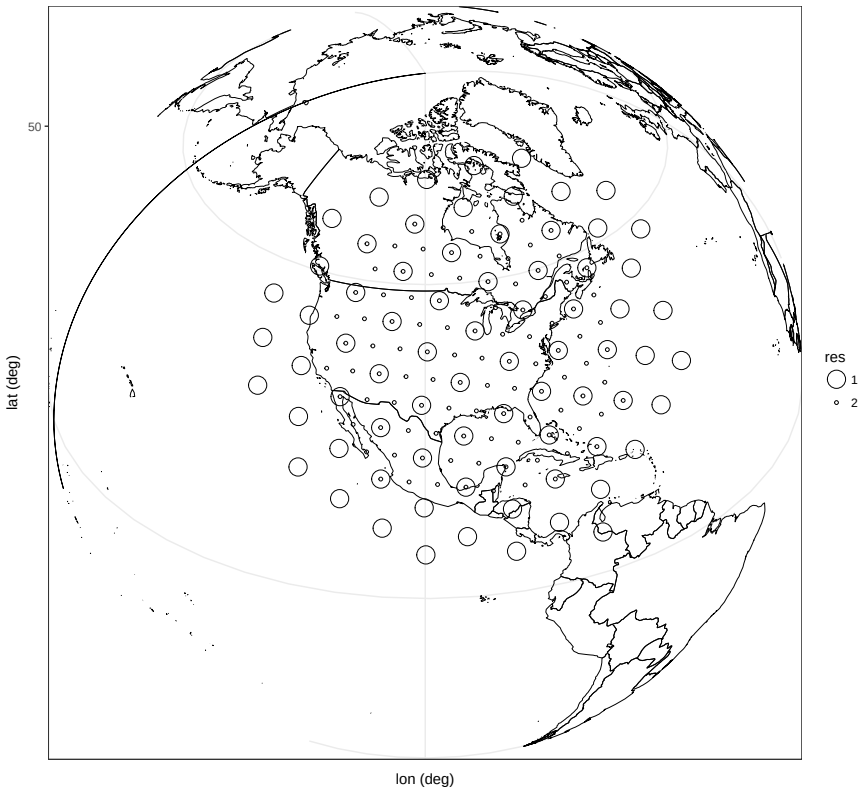



Figure 13: Basis functions for FRK on the sphere with the `NOAA` dataset using two ISEA3H DGGs for location parameters of the basis functions.

```
                    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
```

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=1` argument behaves as above on the basis functions at these higher resolutions. The basis functions constructed using this code are shown in Fig. 13. 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
                       t = day)   %>%          # generate time index
                sample_n(20000)                # sample 2000 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. 14.

```
## 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
```

```
## Error in over(sphere_BAUs, conv_hull):  package rgeos is required for additional over methods
## Error in SpatialPolygonsDataFrame_to_df(grid_BAUs2[, 1], "n"):  object 'grid_BAUs2' not found
## Error in ggplot(X2):  object 'X2' not found
```

**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:

```
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
```

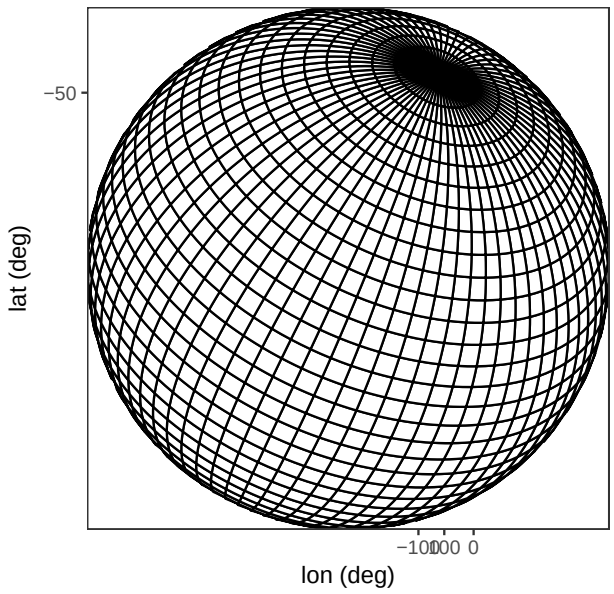



Figure 14: Gridded BAUs on the sphere used for modelling and predicting with the `AIRS` data. (a) BAUs constructed when supplying only the temporal indices of the data (the entire sphere is covered with BAUs within the specified time period). (b) BAUs constructed when supplying the entire dataset. The 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.

```
                          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 `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. 15–17.

```
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(SRE_model = S,        # SRE model
             n_EM = 4,           # max. EM iterations
             tol = 0.01)           # convergence criteria

grid_BAUs <- SRE.predict(SRE_model = S,          # SRE model
                         obs_fs = TRUE,          # fs variation is in obs. model
```

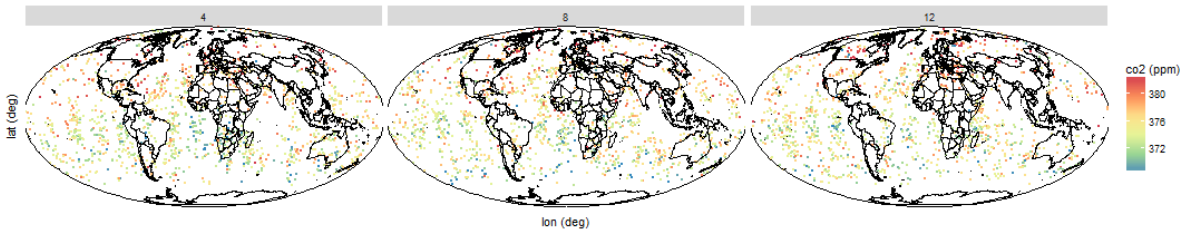



Figure 15: $CO_2$ readings taken from the `AIRS` on the 04, 08 and 12 May 2003 in ppm.

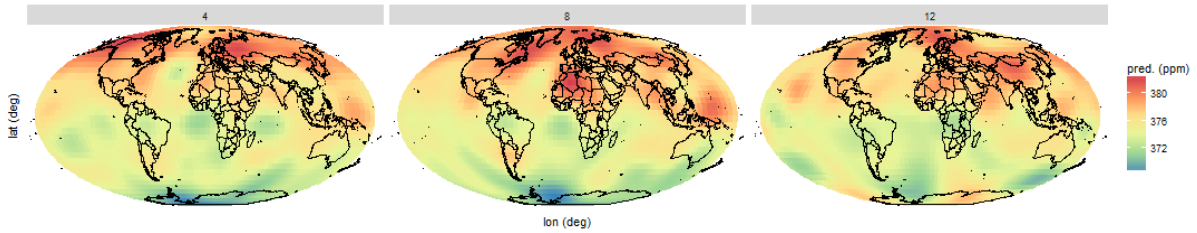



Figure 16: Prediction of $\mathbf{Y}_P$ in ppm on 04, 08, and 12 May 2003 obtained with **FRK** on the `AIRS` data.

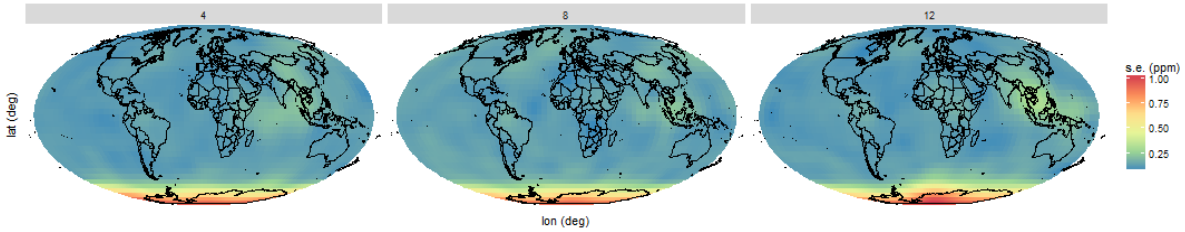



Figure 17: Prediction standard error of $\mathbf{Y}_P$ in ppm on 04, 08 and 12 May 2003 obtained with **FRK** on the `AIRS` data.

```
                    pred_time = c(4L,8L,12L)) # predict only at select days
```

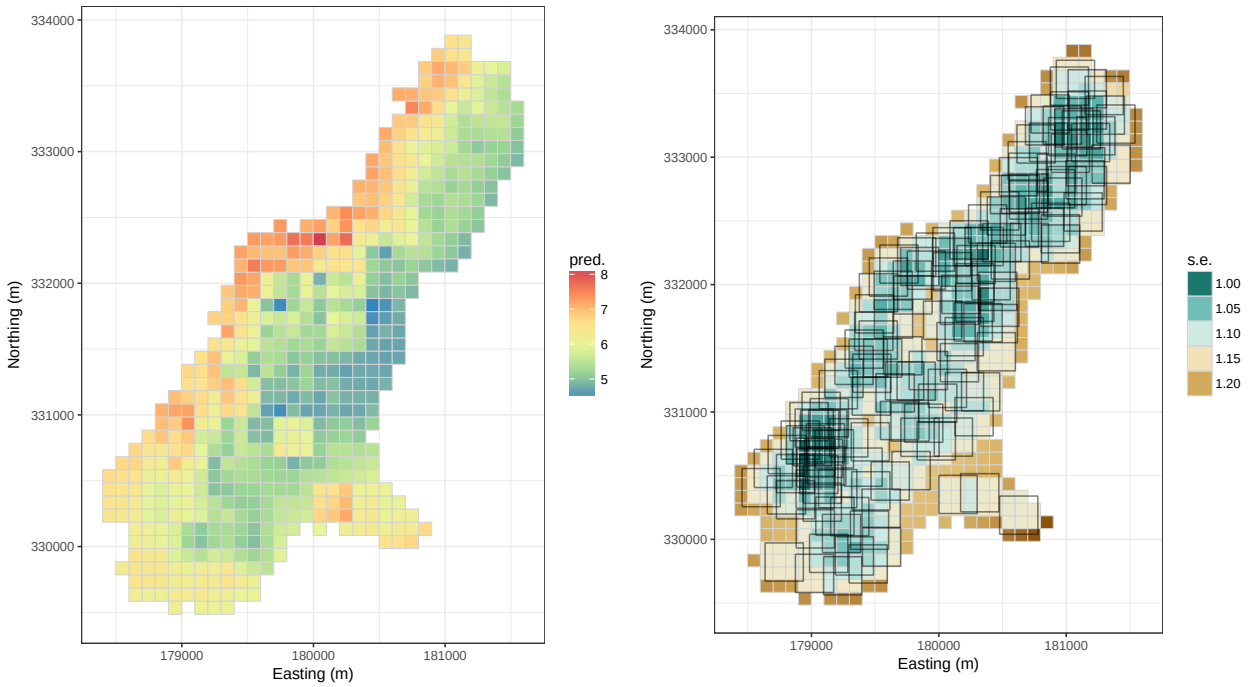
# 5 Other topics

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.

## 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 original value of zinc concentration assigned to it. 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. 18. In Fig. 18 (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




(a) (b)

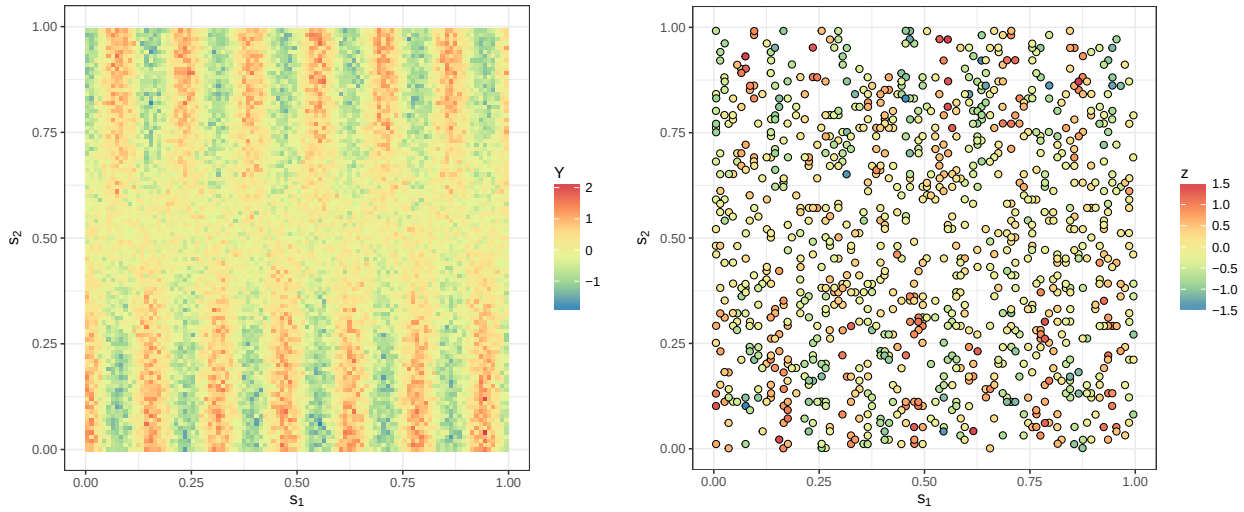
Figure 18: Prediction and prediction standard error obtained with FRK using the `meuse` dataset where each observation is assuming to have a spatial footprint of 300 m × 300m. (a) FRK prediction at the BAU level. (b) FRK prediction standard error at the BAU level. The black hexagons outline the spatial footprints of the data.

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, 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. 19.

```
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
```




(a) (b)

Figure 19: FRK with anisotropic fields. (a) Simulated process. (b) Observed data.

```
          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:

```
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 $5 \times 14$ 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. 20.

```
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
```

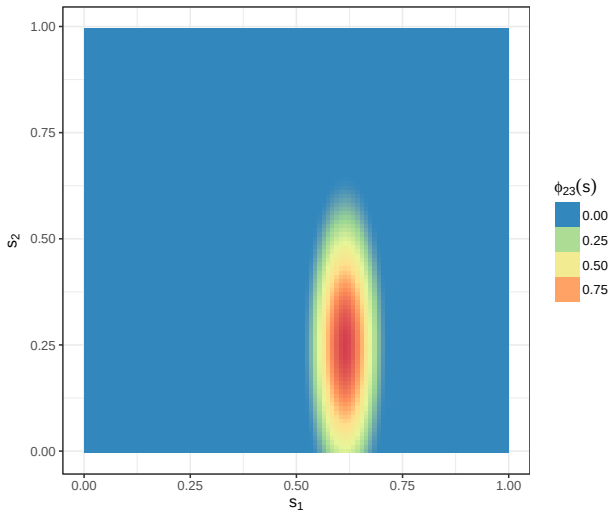



Figure 20: Basis function 23 of the 75 constructed to fit an anisotropic spatial field. Anisotropy is obtained by changing the `measure` object of the manifold on which the basis function is constructed.

```
                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. 21.

## 5.3 Customised basis functions and Basic Areal Units (BAUs)

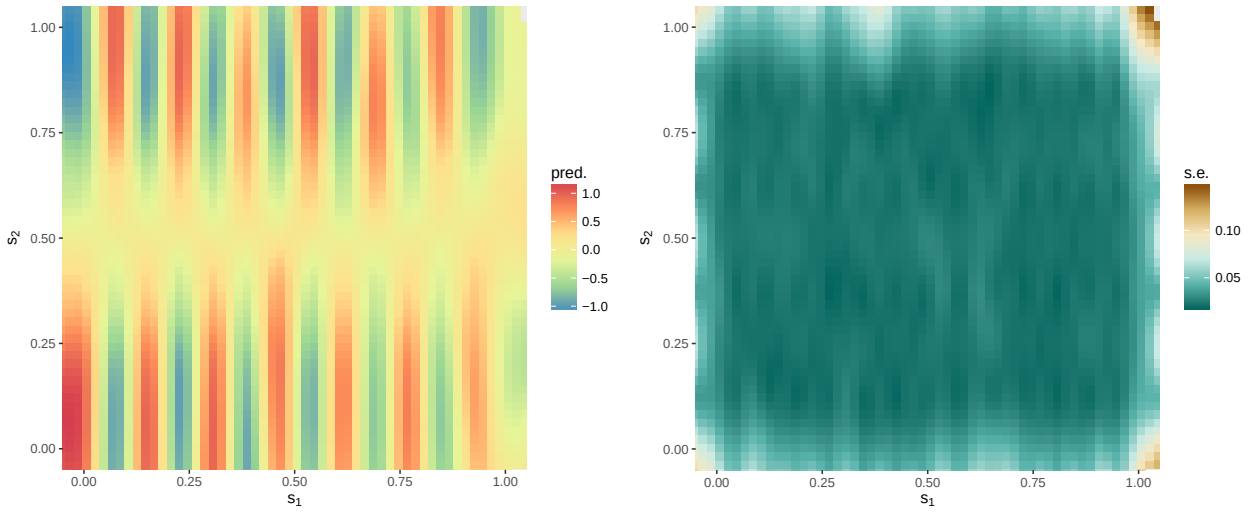
The package **FRK** provides the functions `auto_BAUs` and `auto_basis` to help the user construct the BAUs and basis functions based on the supplied data. These, however, could be done manually. When doing so it is important that some rules are adhered to: The object containing the basis functions needs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

Figure 21: FRK using data generated by an anisotropic field. (a) FRK prediction. (b) FRK prediction standard error.

- 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")`).

# 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 (**?**).

- 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. 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**. The package includes within it some data manipulation functions from **Hmisc**. Some sparse-matrix operations are facilitated using C code from the software package **SuiteSparse** (**?**). We thank Jonathan Rougier for helpful comments on the manuscript, Chris Wikle for discussions on the importance of the fine-scale variation component of a spatial statistical model, Clint Shumack for using **FRK** to analyse the OCO-2 data, and Enki Yoo for providing useful feedback on the package.