Predictive Methods for Large Geostatistical Data

Jingzhe Wang
School of Statistics, University of Minnesota

Abstract

Geostatistical predictive analysis conducts spatial interpolations to unobserved location points based on the observed attribute values at other locations. As large amounts of geospatial data become more accessible and an increasing demand for large spatial data analysis has been realized, classical methods for geostatistical analysis are challenged by the greater computational complexity. Heaton et al. (2018) have gathered twelve different approaches for large geostatistical interpolations in a case study competition and compared their prediction performances as well as required computational resources. This study aimed to (1) provide a brief review of the twelve predictive methods from Heaton et al. (2018), (2) replicate the twelve methods on a simulated satellite image used by Heaton et al. (2018), (3) interpolate the missing values of a new large spatial dataset using two of the methods, Nearest Neighbor Gaussian Processes - Conjugate (NNGP Conjugate) and Fixed Rank Kriging (FRK), and (4) provide a further step of evaluating the prediction performances of NNGP Conjugate at some hold-out locations in the new dataset.

1. Introduction

Geostatistical predictions, originated from the field of mining (Krige, 1951; Schabenberger and Gotway, 2005), have been widely used nowadays in many different problem solving contexts. Applications of geostatistical methods have been found in environmental and ecological science (Hefley et al., 2017), agriculture (Oliver, 2013), real estates (Kuntz and Helbich, 2014) and even the financial market (Arbia and Marcantonio, 2015). Computational intractability, as a well-known issue already, has been exacerbated by the ease of obtaining a tremendous amount of data and the pursuit of flexible solutions. In the case of spatial data analysis, Heaton et al. (2018) conducted a study to summarize and compare different geostatistical prediction methods on large satellite data, with an emphasis on their capability to handle the heavy computational burden.

This paper provided a review of the twelve methods participated in the case study competition of Heaton et al. (2018) as well as their predictive performance scores derived from a replication study on a simulated MODIS satellite image. The actual computing time for each method in the replication study appeared to be very different, from several minutes to over 100 hours. Among the methods that have completed the predictions for 44,431 locations with missing temperature values, consistent prediction accuracy as well as prediction uncertainty were observed, compared with the results from Heaton et al. (2018). Several spatial interpolation analyses were then conducted on the NOAA Sea Level Pressure (SLP) monthly means dataset using Nearest Neighbor Gaussian Processes - Conjugate (NNGP Conjugate) and Fixed Rank Kriging (FRK).

The prediction performances of NNGP Conjugate were then evaluated at some randomly chosen hold-out locations with observed SLPs in the original data.

The following parts in this section will provide a brief introduction to the common geostatistics concepts involved in the twelve methods. Section 2 gives a summary of the twelve methods. Section 3 displays the predictive performance scores from the replication study on the simulated dataset. Section 4 describes an application of NNGP Conjugate and FRK approaches as well as a further prediction evaluation step on the NOAA SLP dataset. Section 5 discusses and concludes the key findings in the replication study and the set of predictive analyses on the new dataset. Code for replication study is attached in Appendix A. Code for analyses on the NOAA SLP dataset is attached in Appendix B.

1.1. Data under Spatial Context

Intuitively speaking, spatial data can refer to any geographically or location-referenced attributes. For example, the humidity value sampled at any weather station across the country. For the simple informative purpose, specific spatial data categories are usually overlooked because the general audience are mostly not interested in knowing about them. Cartographers, however, would require basic data type information such as data dimension (point, line, area or volume) as well as Stevens'(1946) data level (nominal, ordinal, interval or ratio) so that to select the corresponding map type as well as visual variables for effective visualizations (Slocum et al., 2008). When it comes to the phase of sophisticated spatial data modeling, the spatial data type subcategories can play a vital role in leading the analysis directions as well as determining the appropriate model class (Schabenberger and Gotway, 2005). In spatial statistics, spatial data could be categorized based on the characteristics of their spatial domain (Cressie, 1993; Schabenberger and Gotway, 2005).

The spatial domain defines the overall extent of locations where the spatial process of interest may occur. According to Schabenberger and Gotway (2005), a spatial process is a stochastic process with a collection of georeferenced random variables indexed by the location points in a spatial domain. It can also be regarded as a random field because each location point is defined by a pair of longitude and latitude coordinates, which has a dimension greater than one. The spatial process may be expressed by (Schabenberger and Gotway, 2005):

$$\{Y(\boldsymbol{s}) : \boldsymbol{s} \in D \subset \mathbb{R}^2\}. \tag{1}$$

From (1), $\boldsymbol{s}$ represents the two-dimensional location points within the spatial domain D. $Y(\cdot)$ denotes the field values at all observed locations within the spatial domain which altogether constitute a realization of a random experiment.

Back to the discussion of spatial data subcategories, the spatial data can be classified as geostatistical data, lattice/areal data and point pattern data (Schabenberger and Gotway, 2005). This classification is based on whether the data have continuous (non-countable)

spatial domain D and whether the spatial domain D has a fixed (non-stochastic) set of locations (Cressie, 1993; Schabenberger and Gotway, 2005). Both the simulated and real datasets used by Heaton et al. (2018) contain geostatistical data points that have continuous and fixed spatial domain. And predictions can be made at any point location within the specified spatial domain.

1.2. The Gaussian Process Assumption

According to Heaton et al. (2018), the observations in the simulated dataset is a realization of the true Gaussian process. What does it mean by Gaussian process? Based on Heaton et al. (2018) and Wikle, Zammit-Mangion and Cressie (2019), Gaussian process assumes a stochastic process to have Gaussian distributions for all its finite-dimensional distributions. A Gaussian process on geostatistical data can be expressed as (Wikle, Zammit-Mangion and Cressie, 2019):

$$\boldsymbol{Y}(\boldsymbol{s}) \sim GP(\mu(\boldsymbol{s}), c(\boldsymbol{s}, \boldsymbol{s}')), \boldsymbol{s} \in \mathcal{D} \subset \mathbb{R}^2, \tag{2}$$

where $\boldsymbol{Y}(\cdot)$ represents the stochastic process under Gaussian process assumption with mean function $\mu(\cdot)$ and covariance function $c(\cdot, \cdot)$; $\boldsymbol{s}$ represents any two-dimensional location point in the spatial domain $\mathcal{D}$ and $\boldsymbol{s}'$ represents a location that is different from $\boldsymbol{s}$. One big advantage of Gaussian process assumption is that it allows inferences and predictions based on multivariate Gaussian distributions which can form the foundation of analysis method such as kriging (Heaton et al., 2018; Wikle, Zammit-Mangion and Cressie, 2019).

Furthermore, with the simulated observations sampled from the true Gaussian process, they can serve as a good reference to assess how well each of the predictive methods approximates the true Gaussian process. More information about the simulated dataset will be discussed in Section 3.

1.3. Basis Functions

Before going into the details of each method, one needs to first have a grasp of the concept of basis functions. In statistical modeling, a basis function can be defined as a mathematical function that transforms a covariate (Hefley et al., 2017). According to Wikle, Zammit-Mangion and Cressie (2019), one common application of the basis functions is to form the Generalized Additive Models (GAMs). In GAMs, basis functions serve as functions that transform the covariate variables and thus to add some layers of flexibility by taking care of the non-linearity in data. In the spatial context, basis functions can also be functions about locations. The basis functions, weighted by random coefficients, are usually used for representing the complicated curve or surface in space (Wikle, Zammit-Mangion and Cressie, 2019). They could add more flexibility to the model and account for small-scale variations within observed data that are not explained by the general trend derived from predictor variables. Given Tobler's First Law of Geography (1970): "everything is related to everything else, but near things are more

related than distant things", special attention should be given to spatial data modeling as there are usually positive spatial autocorrelations. In other words, the classical simple linear regression model should be used with caution because of its limited ability to accurately generalize the complex spatial process. Basis functions thus could be regarded as a rescue when they help account for the spatial autocorrelations (Hefley et al., 2017).

In geostatistics, the linear combination of basis functions and a set of weights (or basis coefficients) can form the spatial random effect term so that to model the spatial autocorrelations. Using the matrix notations from Hefley et al. (2017), this spatial random effect term is expressed as $\boldsymbol{\eta}$ and $\boldsymbol{\eta} = \boldsymbol{Z}\boldsymbol{\alpha}$, where $\boldsymbol{Z}$ is called the basis expansion which is a matrix with each column representing a basis vector and $\boldsymbol{\alpha}$ denotes a vector of basis coefficients.

In addition to maintaining the flexibility of the statistical models, basis functions with appropriate specifications can also assist in model interpretation, relieving computational burden and improve numerical stability in the parameter estimations (Hefley et al., 2017).

1.4. Hierarchical Model Structure

As we jump into the discussion about any sophisticated methods in Section 2, we shall see a common hierarchical model structure being repeatedly employed. This hierarchical model structure usually has at least two stages: the observation stage and the true latent process stage (Wikle, Zammit-Mangion and Cressie, 2019). They are defined as (Wikle, Zammit-Mangion and Cressie, 2019):

$$\begin{aligned} \text{observations} &= \text{true latent process} + \text{observation error}, \\ \text{true latent process} &= \text{regression component} + \text{dependent spatial random process}. \end{aligned} \tag{3}$$

The true latent process stage is always what we are the most interested about during the analysis. The basis functions introduced in the subsection above are usually embedded in this stage, either by transforming the covariates in the regression component, accounting for autocorrelations throughout the spatial domain, or both. When the spatial random process exists, the spatial dependence is commonly modeled by the parameters in the first or second moment of the assumed probability distributions. It is possible to make inferences about the random effect related parameters from the classical approach or the Bayesian approach. With the classical approach, we treat the random effect coefficients (e.g. basis coefficients) as random variables with constant probability distribution parameters. With the Bayesian approach, however, we treat the random effect coefficients as random variables and further specify prior distributions to their probability distribution parameters.

Berliner (1996) named the Bayesian approach applied to spatial context as Bayesian Hierarchical Model (BHM) (Wikle, Zammit-Mangion and Cressie, 2019). The overall complex dependent process is decomposed into three levels, one for data, one for the

underlying spatial process and one for the unknown parameters. Using the notation from Berliner (1996) and Wikle, Zammit-Mangion and Cressie (2019), the joint distribution for data, spatial process and parameters can be written as

$$\begin{aligned}[\text{data, spatial process, parameters}] = &[\text{data} \mid \text{spatial process, parameters}] \\ &\times [\text{spatial process} \mid \text{parameters}] \\ &\times [\text{parameters}].\end{aligned} \tag{4}$$

Using Bayes' rule, it is possible to make inferences about the posterior distribution for the underlying spatial process, as is illustrated below (Wikle, Zammit-Mangion and Cressie, 2019):

$$\begin{aligned}[\text{spatial process} \mid \text{data, parameters}] \propto &[\text{data} \mid \text{spatial process, parameters}] \\ &\times [\text{spatial process} \mid \text{parameters}] \\ &\times [\text{parameters}].\end{aligned} \tag{5}$$

When the prior knowledge about the parameters is not known, these parameters can be estimated based on data and the model will then become Empirical Hierarchical Model (EHM). In either the case of BHM or EHM, when estimating the unknown parameters, it is not always easy to obtain a closed-form solution when the model is applied to the real world scenarios, especially for large dataset. Therefore, some iterative numerical solutions are needed (Wikle, Zammit-Mangion and Cressie, 2019).

1.5. First-order and Second-order Model Specifications

When modeling the true latent process that takes both fixed and spatial random effects into account under the Gaussian process assumption, there are usually two types of model specifications: first-order specification and second-order specification (Hefley et al., 2017). In first-order specification, the autocorrelation is explicitly included in the mean term of the conditional probability distribution of true spatial process $\boldsymbol{y}$ given the random basis coefficients $\boldsymbol{\alpha}$, for example, $\boldsymbol{y}|\boldsymbol{\alpha} \sim \mathrm{N}(\boldsymbol{X\beta} + \boldsymbol{Z\alpha}, \boldsymbol{\sigma_\varepsilon^2}\mathbf{I})$, with $\boldsymbol{X}$ as a matrix of covariates, $\boldsymbol{\beta}$ as a vector of fixed unknown coefficients, $\boldsymbol{Z}$ as the basis expansion and $\boldsymbol{\sigma_\varepsilon^2}$ as the variance of the error term. Note the random basis coefficients $\boldsymbol{\alpha}$ also follow the Gaussian process assumption and $\boldsymbol{\alpha} \sim \mathrm{N}(\mathbf{0}, \Sigma_{\boldsymbol{\alpha}})$, where $\Sigma_{\boldsymbol{\alpha}}$ is the covariance matrix. The second-order specification is expressed in the marginal distribution of process $\boldsymbol{y}$ and $\boldsymbol{y} \sim \mathrm{N}(\boldsymbol{X\beta}, \boldsymbol{\sigma_\varepsilon^2}\mathbf{I} + \boldsymbol{Z}\Sigma_{\boldsymbol{\alpha}}\boldsymbol{Z}')$, where the spatial autocorrelation structure is included in the covariance matrix of the marginal probability distribution. To choose between these two model specifications, it is always a good practice to think about the overall model structure, size of data, potential implementation issues as well as the targeted questions to be answered by the study. As Wikle and Hooten (2010), Hefley et al. (2017) pointed out, the first-order specification can better illustrate the nature of a hierarchical model and is easier to understand. In contrast, the second-order specification may be a little

obscure but it allows the implementations of more computationally efficient algorithms (Hefley et al., 2017). Many of the methods in Section 2 will demonstrate these types of model specifications.

1.6. Kriging

In spatio-temporal data analysis, kriging is a classical statistical modeling technique that is typically used for value interpolations (Cressie and Johannesson, 2008; Wikle, Zammit-Mangion and Cressie, 2019). Moreover, according to Cressie and Johannesson (2008), kriging can generate the best linear unbiased predictors (BLUP) which minimizes the mean squared prediction error (MSPE).

In kriging, the observed complicated process $Z(\cdot)$ could be expressed by a hierarchical model structure (Cressie and Johannesson, 2008):

$$Z(s) = Y(s) + \varepsilon(s),\ s \in D \subset \mathbb{R}^d, \tag{6}$$

where $\varepsilon(s)$ represents the measurement errors with mean 0, variance $\sigma_\varepsilon^2 v(s) > 0$ and with $v(s)$ known. $Y(s)$ is the underlying spatial process. s represents any location in the spatial domain D (a subset of d-dimensional Euclidean space and for spatial data d is usually 2). $Y(s)$ can be further expressed by (Cressie and Johannesson, 2008; Heaton et al., 2018):

$$Y(s) = \mu(s) + \omega(s) + \xi(s),\ s \in D \subset \mathbb{R}^d, \tag{7}$$

where $\mu(s)$ accounts for the fixed effects from the predictor variables and represents a general trend; $\omega(s)$ accounts for the spatial random effect and $\xi(s)$ represents the spatially uncorrelated fine-scale process with mean 0, variance σ_ξ^2.

For large spatial data, this method can be computationally intensive in the process of covariance matrix (Σ) inversion for spatial random process $\omega(s)$. One way to resolve the issue is by reducing the rank of the spatial random effect covariance matrix. More details will be discussed in Section 2.

2. Methods for Predictions on Large Spatial Data

2.1. FRK

The Fixed Rank Kriging (FRK) mitigates the computational issue by representing the spatial random effect $\omega(s)$ with a linear combination of a total of K basis functions and K coefficients, with K smaller than the total number of observations N (Cressie and Johannesson, 2008).

FRK does more than adopting the basis functions in the modeling process, it allows basis functions to be specified for observed locations points extracted from multiple

resolutions throughout the spatial domain. This is typically helpful to relieve the spatial non-stationarity problem for large spatial data.

Using notations from Heaton et al. (2018), the random effect term $\omega(s)$ in equation (7) can be written as:

$$\omega(s) = \sum_{r=1}^{R} \sum_{k=1}^{K_r} h_{rk}(s)\theta_{rk}, \;\; s \in D \subset \mathbb{R}^d, \tag{8}$$

where there are in total R different resolutions, $h_{rk}(s)$ is the k^{th} spatial basis function at the r^{th} resolution with the corresponding random coefficient θ_{rk}, K_r is the number of basis functions at the r^{th} resolution and $K = \sum_{r=1}^{R} K_r$ is the total number of basis functions (or corresponding coefficients). The basis functions employed by FRK are the bisquare functions (Heaton et al., 2018). According to Cressie and Johannesson (2008), a bisquare function is written as:

$$S_{j(l)}(u) = \begin{cases} \{1 - (||u - v_{j(l)}||/r_l)^2\}^2, & ||u - v_{j(l)}|| \leq r_l, \\ 0, & \text{otherwise}, \end{cases} \tag{9}$$

where u represents any locations, $v_{j(l)}$ is one of the center points at lth resolution and r_l is 1.5 times the shortest great arc distance between center points of the lth resolution; center points correspond to the center locations extracted from the defined grids throughout spatial domain at each resolution.

Note that in equation (7), the fixed effect term $\mu(s)$ could be written as $\mu(s) = \mathbf{x}'(s)\boldsymbol{\beta}$ where $\mathbf{x}(s)$ is a vector of covariates values at location s and $\boldsymbol{\beta}$ is a vector of unknown coefficients. These coefficients are estimated through ordinary least squares (OLS) (Cressie and Katzfuss, 2011).

According to Heaton et al. (2018), at each resolution, the covariance among basis function coefficients θ_{rk} are determined by the distance between the corresponding two locations and a common spatial correlation parameter. Basis function coefficients across different resolutions are independent. The vector of basis function coefficients $\boldsymbol{\theta}$ is assumed to be a K-dimensional Gaussian vector with mean 0 and a K-by-K positive definite covariance matrix M with single unknown parameter ϕ. From Zammit-Mangion and Cressie (2018), the spatial dependence parameters: basis-function matrix M as well as the unknown σ_ξ^2, are estimated using the maximum likelihood (ML) through expectation-maximization (EM) algorithm. Apart from the EM method, the method of moment (MM) can derive the estimates as well (Cressie and Katzfuss, 2011). The measurement error variance σ_ε^2 is obtained by first computing semivariogram estimates through the robust variogram estimator and then fitting a straight line with weighted least squares. The resulting intercept would be the estimated measurement error variance (Cressie and Katzfuss, 2011).

2.2. LatticeKrig

According to Nychka et al. (2015), LatticeKrig model is similar to FRK with respect to its general hierarchical modeling structure and support of multiresolutional basis functions but with the fine-scale process, as the term $\xi(s)$ shown in equation (7), left out. So the LatticeKrig modeling structure for each spatial observation i is:

$$Z(s_i) = \mathbf{x}'\boldsymbol{\beta} + g(s_i) + \varepsilon_i, \tag{10}$$

where $\mathbf{x}$ is a vector of covariates values, $\boldsymbol{\beta}$ represents a vector of unknown linear coefficients, $g(\cdot)$ is the spatial random effect term and is also assumed to be a smooth Gaussian process, ε as the measurement error with zero mean (Nychka et al., 2015).

One major goal of this method is still spatial interpolation which models the spatial random process term $g(\cdot)$ so that to accurately predict the field values for unobserved locations as well as quantify the prediction uncertainty (Nychka et al., 2015; Wikle, Zammit-Mangion and Cressie, 2019).

As with FRK, LatticeKrig can employ the lower rank structure for the spatial random process and the multiresolutional setup. However, it chooses the radial basis functions instead of the bisquare ones and adds constraints for normalization (Heaton et al., 2018).

Eventually, the spatial random effect term could be written as a linear combination of some fixed basis functions with random basis coefficients:

$$g(s) = \sum_{r=1}^{R} \sum_{k=1}^{m} c_{rk} h_{rk}(s), \tag{11}$$

where $h_{rk}(s)$ denotes the basis function around each location and c_{rk} is the stochastic coefficients. For each resolution r, the marginal variance of $c_{rk}h_{rk}(s)$ is constrained to be $\sigma_g^2 \alpha_r$, where $\sigma_g^2 > 0$, $\alpha_r > 0$ and $\sum_{r=1}^{R} \alpha_r = 1$ (Heaton et al., 2018; Nychka et al., 2015).

The basis at each location is constructed by a function of a scaled distance (Heaton et al., 2018):

$$h_{rk}(s) = \psi(\parallel s - u_{rk} \parallel / \theta_r), \tag{12}$$

where u_{rk} represents the regular grid point for the spatial domain, θ_r is a scale parameter and is defined to be 2^{-r}. The basis function $\psi(\cdot)$ is derived from Wendland polynomial correlation function with compact support (Heaton et al., 2018):

$$\psi(d) \propto \begin{cases} \frac{1}{3}(1-d)^6(35d^2 + 18d + 3) & \text{if } d \leq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Those zero function values result from greater distances between the location points and predefined grid points contribute to the compactness of the support.

The fixed effect terms $\boldsymbol{\beta}$ as well as the covariance parameter σ_g^2 are estimated by maximizing the log-likelihood value. The basis coefficient vector for each resolution, $\boldsymbol{c}_r$, follows the multivariate Gaussian distribution with zero mean and a covariance matrix $\sigma_g^2 Q_r^{-1}$ (Nychka et al., 2015). Q_r^{-1} is parameterized by a single unknown parameter ϕ_r (Heaton et al., 2018). The authors of the LatticeKrig also assume that basis coefficients at each resolution follow the Gaussian Markov Random Field, in particular, with a spatial autoregressive correlation structure. By defining a block diagonal autoregression matrix $\boldsymbol{B}$, along with the previously specified constraints on the marginal variance for the spatial random process and the estimated covariance parameter $\hat{\sigma}_g^2$, it is straightforward to estimate the sparse precision matrix $\boldsymbol{Q}$ based on equation (Nychka et al., 2015):

$$\boldsymbol{Q} = \frac{1}{\sigma_g^2} \boldsymbol{B}'\boldsymbol{B}. \tag{14}$$

Furthermore, Heaton et al. (2018) commented in their paper that due to the sparsity of the precision matrix, LatticeKrig allows a more flexible spatial random process with greater number of basis functions as well as basis coefficients, meanwhile, not at the expense of sacrificing its computational efficiency.

2.3. Predictive Processes

This approach conducts full Bayesian inference on the geostatistical data and it applies Markov Chain Monte Carlo (MCMC) for parameter estimates (Banerjee et al., 2008).

In Predictive Processes, the observed process $\boldsymbol{Z}(\cdot)$ also inherits the two-stage hierarchical structure (Finley et al., 2015):

$$\boldsymbol{Z}(s) = \boldsymbol{\mu}(s) + \tilde{\boldsymbol{\omega}}(s) + \boldsymbol{\epsilon}(s), \ s \in D. \tag{15}$$

The general trend term $\boldsymbol{\mu}(s) = \boldsymbol{X}(s)\boldsymbol{\beta}$ contains unknown coefficients $\boldsymbol{\beta}$. And the spatial random effect term $\tilde{\boldsymbol{\omega}}(s)$ relies on the distribution properties of K pre-specified knot locations over the spatial domain (K $<$ total number of location points N). The knots are usually selected based on some prediction-driven design criterion, such as minimizing spatially averaged prediction variance over all the observed locations (Finely et al., 2009). Note that this $\tilde{\boldsymbol{\omega}}(s)$ could also be expressed in the form of linear combination of basis functions ($\mathbb{C}'(s, s^*)\Sigma_{\boldsymbol{\omega}^*}^{-1}$) and basis coefficients ($\boldsymbol{\omega}^*$) as with FRK (Heaton et al., 2018):

$$\tilde{\boldsymbol{\omega}}(s) = \mathbb{C}'(s, s^*)\Sigma_{\boldsymbol{\omega}^*}^{-1}\boldsymbol{\omega}^*, \tag{16}$$

where $\mathbb{C}(s, s^*) = \sigma^2\rho(s, s^*)$ is a product of an unknown parameter σ^2 and the correlation function $\rho(\cdot)$; s represents locations with observed response values and s^* represents knot locations. An exponential correlation structure was chosen when applying this approach to the study of Heaton et al. (2018), which can be expressed as:

$$\rho(s, s^*; \phi) = \exp(-\phi||s - s^*||), \tag{17}$$

where ϕ is the unknown parameter. The K terms from the basis coefficient vector $\boldsymbol{\omega}^*$ are assumed to be multivariate Gaussian/normal distributed $\mathcal{N}(0, \Sigma_{\boldsymbol{\omega}^*})$, where $\Sigma_{\boldsymbol{\omega}^*}$ is a K-by-K covariance matrix for knot locations with each entry calculated by $\mathbb{C}(s_i^*, s_j^*)$, $i = 1, \ldots, K$, $j = 1, \ldots, K$. The model also assumes that measurement error terms $\boldsymbol{\epsilon}(s)$ are independent and identically normal distributed $\mathcal{N}(0, \tau^2)$, where τ^2 is another unknown parameter (Finley et al., 2015).

After obtaining the posterior samples for unknown parameters $\boldsymbol{\beta}$, σ^2, ϕ, τ^2 as well as random effect coefficients at knot locations $\boldsymbol{\omega}^*$, one posterior prediction for a new locations s_0 can be sampled from a normal distribution with mean $\boldsymbol{X}\hat{\boldsymbol{\beta}} + \mathbb{C}'_{nk}(s_0, s^*; \hat{\sigma}^2, \hat{\phi})\Sigma^{-1}_{\boldsymbol{\omega}^*}(s^*, s^*; \hat{\sigma}^2, \hat{\phi})\hat{\boldsymbol{\omega}}^*$ and variance $\hat{\tau}^2 I$, where $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$, $\hat{\phi}$, $\hat{\tau}^2$ and $\hat{\boldsymbol{\omega}}^*$ are one set of posterior parameter samples, $\mathbb{C}'_{nk}(\cdot)$ is the n-by-K (assume to make predictions at n locations) covariance matrix among the to-be-predicted locations and knot locations, and $\Sigma_{\boldsymbol{\omega}^*}$ is the K-by-K covaraince matrix for knot locations themselves (Finley et al., 2015).

Finley et al. (2015) have also proposed an improvement to the Predictive Processes by taking the variance that is not captured by the low-rank spatial random effect approximation into account. Now the model structure for observed process should look like:

$$\boldsymbol{Z}(s) = \boldsymbol{\mu}(s) + \tilde{\boldsymbol{\omega}}(s) + \boldsymbol{\xi}(s) + \boldsymbol{\epsilon}(s), \ s \in D, \tag{18}$$

where the $\boldsymbol{\xi}(s)$ is the fine-scale process with normal distribution $\mathcal{N}(0, \sigma^2 - \mathbb{C}'(s, s^*)\Sigma^{-1}_{\boldsymbol{\omega}^*}\mathbb{C}'(s, s^*))$ (Finley et al., 2015; Heaton et al., 2018).

One advantage of predictive processing is that it can accommodate the complex modeling structure through the choice of the covariance function. One disadvantage of this method is that there is no closed-form expression for the basis functions, they are derived from iterative calculations and thus usually require greater computational resources and time (Heaton et al., 2018).

2.4. Spatial Partitioning

In spatial partitioning, it first of all partitions the overall spatial domain into non-overlapping subregions. There are various partitioning approaches available for spatial data, as for the study of Heaton et al. (2018), they chose to partition the spatial domain into equal areas. For each subregion, a mixed effect model can be fitted for the underlying spatial process $\boldsymbol{Y}_d = \{Y(s_i) : s_i \in \mathcal{D}_d\}$, just like the models we discussed previously (Heaton et al., 2018):

$$\boldsymbol{Y}_d|\boldsymbol{\theta} \overset{ind}{\sim} \mathcal{N}(\boldsymbol{X}_d\boldsymbol{\beta} + \boldsymbol{H}_d\boldsymbol{\theta}, \Sigma(\phi_d)), \tag{19}$$

where $d = 1, \ldots, D$ and each $\mathcal{D}_d$ represents one of the equal area subregions partitioned from the original spatial domain $\mathcal{D}$; $\boldsymbol{X}_d$ is the design matrix with the given predictor variables and $\boldsymbol{\beta}$ is the general trend unknown coefficients; $\boldsymbol{H}_d$ is the basis expansion and $\boldsymbol{\theta}$ represents the random basis coefficients; $\Sigma(\phi_d)$ denotes the covariance matrix with a covariance parameter ϕ_d. Coefficient estimates are derived from maximizing the likelihood.

As is shown from the equation above, the spatial random effect can be again represented by a product of a matrix of basis functions and a vector of basis coefficients. One key point for this method is that location points within one subregion are dependent (with shared fixed effect parameters and random effect parameters) while location points across different subregions are independent (Heaton et al., 2018). This cross-subregion independence helps reduce the computational cost by enabling the simultaneous estimation of the coefficients in all subregions. The simultaneous estimation was conducted by parallel computing.

2.5. Covariance Tapering

Covariance tapering (Kaufman, Schervish and Nychka, 2008) is a method for approximating maximum likelihood estimators (MLEs) under the context of spatially correlated Gaussian process assumption. The covariance tapering elementwisely multiplies the original covariance matrix with another correlation matrix of the same dimension but with a compact support (Kaufman, Schervish and Nychka, 2008; Heaton et al., 2018). The resulting matrix remains positive definite and is sparse (Kaufman, Schervish and Nychka, 2008). By replacing the original covariance matrix with this tapered covariance matrix, the computational difficulty would be relieved.

When estimating the unknown parameters in modeling process (e.g. kriging), one-taper approximation and two taper approximation are the common approaches (Kaufman, Schervish and Nychka, 2008). The likelihood of a spatial Gaussian process can be expressed as (Heaton et al., 2018):

$$f_{\boldsymbol{Y}}(\boldsymbol{y}) = (\frac{1}{\sqrt{2\pi}})^N |\boldsymbol{\Sigma}|^{-1/2} \exp(\text{trace}(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})(\boldsymbol{y}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1})). \tag{20}$$

In one-taper approximation, covariance matrix $\boldsymbol{\Sigma}$ is substituted by $\boldsymbol{\Sigma} \odot \boldsymbol{T}$ where $\odot$ represents elementwise product and $\boldsymbol{T}$ is the tapering covariance matrix. In two-taper approximation, covariance $\boldsymbol{\Sigma}$ is substituted by $\boldsymbol{\Sigma} \odot \boldsymbol{T}$ and the empirical covariance $(\boldsymbol{y}-\boldsymbol{\mu})(\boldsymbol{y}-\boldsymbol{\mu})'$ is substituted by $(\boldsymbol{y}-\boldsymbol{\mu})(\boldsymbol{y}-\boldsymbol{\mu})' \odot \boldsymbol{T}$ (Heaton et al., 2018). Also note that the resulting tapered covariance matrix should keep the same marginal variances for the process, which could serve as a constraint condition on the choice of tapering covariance matrix $\boldsymbol{T}$ (Kaufman, Schervish and Nychka, 2008).

Moreover, the one-tapering approach results in biased parameter estimates while the two-tapering approach gives unbiased estimates. However, the two-tapering approach

has much greater computational cost than the one-tapering approach. So there is a trade-off between accuracy and computational cost.

In the study of Heaton et al. (2018), to further reduce the computational burden, they estimated the unknown parameters by minimizing the squared difference between the parameterized covariance function and the empirical covariance value instead of through maximizing the likelihood.

2.6. Multiresolution Approximation (MRA)

Similar to the FRK approach, the Multiresolution Approximation (MRA) uses the linear combination of basis functions and basis coefficients to represent the spatial random effect. One main idea of this MRA method is to approximate the true spatial random effect Gaussian processes by increasing the number of basis functions while decreasing the support of each function when the resolution increases (Katzfuss and Gong, 2017). Thus it could generate a covariance matrix for the basis coefficients/weights that is sparse so that to reduce computational costs. The decreased support can be achieved through MRA-taper or MRA-block (Katzfuss and Gong, 2017). Another key feature of this MRA method is that it applies Predictive Processes (as introduced previously) to approximate the generic Gaussian process which does not have any restrictions on the covariance function choice as long as the resulting covariance matrix is positive definite (Katzfuss and Gong, 2017).

Heaton et al. (2018) chose to use MRA-block for their case study. Typically, the MRA-block iteratively partitions the spatial domain at each resolution into finer subregions, assuming the random process between subregions are independent (Katzfuss and Gong, 2017). The number of recursive partition (or resolution level) M, the number of subregions J partitioned from the previous spatial domain as well as the number of basis functions r in each partition can be customized. Also, note that for each part or subpart, there is r number of knots being selected and they are at locations where the basis functions reach their maximums. Using the notation from Katzfuss (2017), the set of knot locations in each resolution level m is denoted by $\mathcal{Q}^{(m)}$, $m = 0, \ldots, M$.

To understand the mechanism of MRA-block, we need to first take a look at the recursive partitioning. Using the study of Heaton et al. (2018) as an example, they specified 9 different resolution levels ($M = 9$), 2 parts being partitioned from each region or subregion in the previous resolution ($J = 2$), and 64 basis functions as well as knots in each partition ($r = 64$). More specifically, in the first resolution level, there is not any partitioning or subparts, but it has 64 basis functions to approximate the process. When moving to the second resolution level, the original spatial domain $\mathcal{D}$ is partitioned into 2 subregions and in each subregion, there are again 64 basis functions to approximate the corresponding process. For the next resolution level, each subpart in the previous resolution level is further partitioned into 2 parts and there are always 64 basis functions and 64 knots in each part.

According to Katzfuss (2017), the predictive processes employed in this approach approximates the actual Gaussian process by recursively summing up approximated

'remainder terms' in each resolution level m. To illustrate the approximated Gaussian process at the highest resolution level M (should be the best approximation among all levels) with equations, and $\omega_M(\cdot)$ represents the spatial random Gaussian process of interest (Sang, Jun and Huang, 2011; Katzfuss, 2017):

$$\omega_M(\cdot) = \tau_0(\cdot) + \tau_1(\cdot) + \cdots + \tau_{M-1}(\cdot) + \delta_M(\cdot), \tag{21}$$

where $\tau_m(s) = E(\delta_m(s)|\delta_m(\mathcal{Q}^{(m)}))$, $s \in \mathcal{D}$ denotes the approximated Gaussian process from Predictive Processes at current resolution m; $\delta_m(\cdot) = [\delta_{m-1}(\cdot) - \tau_{m-1}(\cdot)]_{[m]} \sim \text{GP}(0, \upsilon_m)$ denotes the remainder term from the previous resolution $m-1$; $\tau_0(s) = E(\omega_0(s)|\omega_0(\mathcal{Q}^{(0)}))$, $s \in \mathcal{D}$, and $\delta_1(\cdot) = [\omega_0(\cdot) - \tau_0(\cdot)]_{[1]}$; υ_m denotes the covariance matrix at resolution m for the corresponding locations.

In MRA, parallel computing under distributed-memory systems could greatly reduce the computational time and complexity involved in the inferences and predictions (Katzfuss, 2017).

2.7. Nearest Neighbor Gaussian Processes (NNGP)

The Nearest Neighbor Gaussian Process (NNGP) approach was firstly proposed by Datta et al. (2016) which is a sparsity-inducing method for inferences and predictions of large geostatistical data. The key steps of this approach include the Bayesian inference and the recovery of latent spatial random effect $\boldsymbol{\omega}(s)$. To illustrate how NNGP takes effect in the Bayesian inference procedure, let's first look at the Bayesian hierarchical models formulation (Finley et al., 2017):

$$p(\boldsymbol{\beta}, \boldsymbol{\theta}, \sigma_\epsilon^2) \times \mathcal{N}(\boldsymbol{\omega}|\mathbf{0}, \mathbf{C}) \times \mathcal{N}(\mathbf{y}|\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\omega}, \sigma_\epsilon^2\mathbf{I}), \tag{22}$$

where $\boldsymbol{\beta}$ represents the unknown general trend coefficients, $\boldsymbol{\omega}$ is the spatial random effect term, $\boldsymbol{\theta}$ are parameters for the Gaussian Process covariance $\mathbf{C}$, σ_ϵ^2 is the variance of measurement error, $\mathbf{y}$ is a vector of responses and $\boldsymbol{X}$ is the design matrix with covariates information.

With NNGP, the prior distribution of the spatial random effect $\boldsymbol{\omega}$ could be expressed by (Finley et al., 2017):

$$p(\boldsymbol{\omega}) = p(\omega_1) \prod_{i=2}^{n} p(\omega_i|\text{Pa}[i]), \tag{23}$$

where $\omega_i \equiv \omega(s_i)$ and $\text{Pa}[i] = \{\omega_1, \omega_2, \ldots, \omega_m\}$ which could be defined as a set of m (a small number relative to the total number of locations n) nearest neighboring (distance-based) locations of s_i.

With the assumption of Gaussian process, the spatial random effects for different locations $\omega(s_i)$, $i = 1, \ldots, N$ follow a multivariate normal distribution. The conditional

distribution for each $\omega(s_i)$ can be expressed by the random effects of the rest of location points. The compact form expression for the spatial random effect term is (Finley et al., 2017; Heaton et al., 2018):

$$\boldsymbol{\omega} = \boldsymbol{A}\boldsymbol{\omega} + \boldsymbol{\eta}, \tag{24}$$

and $\mathbf{A}$ is a sparse lower triangular matrix with diagonal values all zeros and at most m ($m < \mathrm{N}$) non-zero entires in each row. This m refers to the m distance-based nearest neighbors of a location, as is illustrated in $\mathrm{Pa}[i]$ in equation (23). This nearest neighbor constraint helps further reduce the size of matrix $\mathbf{A}$ so that can facilitate computation. And $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{D})$ with $\mathbf{D}$ as the diagonal matrix with diagonal entries $\{\mathrm{Var}(\omega_1), \mathrm{Var}(\omega_2|\omega_1), \mathrm{Var}(\omega_3|\omega_2, \omega_1), \ldots, \mathrm{Var}(\omega_n|\omega_{n-1}, \omega_{n-2}, \ldots, \omega_1)\}$.

Finley et al. (2017) further implemented and extended the concept of NNGP into collapsed NNGP (using low dimensional MCMC chain for full recovery of the spatial random effect $\omega(s)$), response NNGP (which skips the step of recovering $\omega(s)$ and instead, predict the response at new locations directly) and conjugate NNGP (which allows MCMC-free inferences) for practical uses. In the study of Heaton et al. (2018), they implemented both the response NNGP and conjugate NNGP to predict the missing values from the real satellite image and the simulated dataset.

2.8. Stochastic Partial Differential Equations (Stochastic PDEs)

The approach of Stochastic Partial Differential Equations (SPDEs) aims to use the approximate solution of partial differential equations as an explicit link between Gaussian fields (GFs) and the Gaussian Markov random fields (GMRFs), which could ensure both prediction accuracy and scalable computation (Lindgren, Rue and Lindstrom, 2011). It specifies the GMRF representation as the SPDE whose solutions are GFs with Matern covariance function. The latent Gaussian process modeling is based on the properties and covariance structure of the GFs while the computational efficiency is improved by the sparse precision matrix resulting from the Markov properties of GMRFs. Furthermore, the GMRFs enable the method to conduct approximate Bayesian inferences by integrated nested Laplace approximation (INLA) which has the advantage of generating precise estimates, providing tools for error assessment as well as short computational time (Rue, Martino and Chopin, 2009).

The GMRF representation or the SPDE weak solution could be expressed by the familiar spatial random effect term $\boldsymbol{\omega}(\boldsymbol{s})$:

$$\boldsymbol{\omega}(\boldsymbol{s}) = \sum_{k=1}^{n} h(\boldsymbol{s}) w_k, \tag{25}$$

where n is the total number of vertices of a defined triangulated lattice, $h(\cdot)$ denotes the basis functions and w_k, $k = 1, \ldots, n$ are the basis coefficients (or weights) from Gaussian distribution (Lindgren, Rue and Lindstrom, 2011). The triangulation is based

on maximizing the minimum interior triangle angle as well as the constraint that one triangle vertex should be placed at each observed location within the spatial domain (Lindgren, Rue and Lindstrom, 2011).

According to Heaton et al. (2018), the choice of triangulation could determine the sparse precision matrix (inversion of the corresponding covariance matrix) parameters: $1/\phi_\sigma^2$ (precision parameter) as well as $1/\phi_r$ (inverse range parameter) and thus the structure of the sparse precision matrix $\boldsymbol{Q}_\theta(\boldsymbol{\phi})$. This precision matrix $\boldsymbol{Q}_\theta(\boldsymbol{\phi})$ along with the pre-specified prior $\boldsymbol{Q}_\beta = \boldsymbol{I} \cdot 10^{-8}$ for the unknown coefficients $\boldsymbol{\beta}$ collective form the block-diagonal precision matrix $\boldsymbol{Q_y} = \text{diag}(\boldsymbol{Q_\theta}, \boldsymbol{Q_\beta})$, where the distribution of spatial latent random process $\boldsymbol{y}$ is represented by the joint Gaussian distribution of $(\boldsymbol{\theta}, \boldsymbol{\beta})$. Thus, the observed process could be represented by $\boldsymbol{Z} = \boldsymbol{X\beta} + \boldsymbol{H\theta} + \boldsymbol{\varepsilon} = \boldsymbol{Ay} + \boldsymbol{\varepsilon}$ where $\boldsymbol{y} = (\boldsymbol{\theta}\ \boldsymbol{\beta})'$ and $\boldsymbol{A}$ is the design matrix $(\boldsymbol{H}, \boldsymbol{X})$, $\boldsymbol{H}$ refers to a sparse basis evaluation matrix, and $\boldsymbol{\varepsilon}$ is a noise vector with mean zero and precision matrix $\boldsymbol{Q}_\varepsilon = \boldsymbol{I}/\sigma_\varepsilon^2$.

With $\boldsymbol{Q}_y$, $\boldsymbol{A}$ and $\boldsymbol{Q}_\varepsilon$ well defined, the conditional precision matrix for the process $\boldsymbol{y}$ given $\boldsymbol{Z}$ is (Heaton et al., 2018):

$$\boldsymbol{Q}_{y|z} = \boldsymbol{Q}_y + \boldsymbol{A}'\boldsymbol{Q}_\varepsilon\boldsymbol{A}. \tag{26}$$

Using Takahashi recursions on the Cholesky factor of $\boldsymbol{Q}_{y|z}$, the conditional covariance matrix $\boldsymbol{Q}_{y|z}^{-1}$ could be derived (Rue, Martino and Chopin, 2009). Thus the conditional expectation of the underlying process given the observed values would be (Heaton et al., 2018):

$$\boldsymbol{\mu}_{y|z} = \boldsymbol{Q}_{y|z}^{-1}\boldsymbol{A}'\boldsymbol{Q}_\varepsilon\boldsymbol{Z}. \tag{27}$$

2.9. Periodic Embedding

The Periodic Embedding approach conducted in the case study of Heaton et al. (2018) was initially proposed by Guinness (2017) as an iterative semiparametric spectral approach designed for analyzing regular gridded data with stationary process assumption. As is introduced by Heaton et al. (2018), the spectral method application in geostatistical data faces the problem of biased spectrum estimation as well as missing values. Guinness (2017) resolved the issues by iteratively imputing missing data with periodic conditional simulations on a small-domain expanded lattice as well as a parametric filtering method to further reduce the smoothing bias from each iteration.

The small domain expansion is conducted by specifying an expansion factor τ and multiplying it with the original grid dimensions $\boldsymbol{N}$, so the total number of locations $m = \tau\boldsymbol{N}$ involved in the analysis would be greater than $\boldsymbol{N}$ (Heaton et al., 2018).

The computation involved in this approach is scalable because of the implementation of discrete Fourier transformation (DFT) through fast Fourier transformation (FFT) (Guinness, 2017; Heaton et al., 2018). Due to the lossless property of Fourier transformation, the field value variations throughout the spatial domain could be inspected on

the frequency domain. In other words, the covariance function could be expressed in terms of the spectral density f and complex sine waves.

The main task of this approach is to iteratively obtain a better estimate of the spectral density f (Guinness, 2017). This requires the calculation of a vector of complex Fourier coefficients through DFT. For each spatial location $\boldsymbol{s}$, we could find a corresponding Fourier coefficient $J(\boldsymbol{\omega})$ at a specific frequency $\boldsymbol{\omega}$ in the frequency domain. This Fourier coefficient $J(\boldsymbol{\omega})$ is computed by (Guinness, 2017; Heaton et al., 2018):

$$J(\boldsymbol{\omega}) = \frac{1}{\sqrt{m}} \sum_{\boldsymbol{s}} Y(\boldsymbol{s}) \exp(-i\boldsymbol{\omega}'\boldsymbol{s}), \tag{28}$$

where $Y(\cdot) = (U'(\cdot), V'(\cdot))$ contains both the observed (U) and unobserved (or missing, denoted by V) field values, m is the total number of locations of interest after grid expansion and $i = \sqrt{-1}$. Also, recall the Euler's formula, the exponentiated term could be expressed as:

$$\exp(ix) = \cos(x) + i\sin(x), \ x \in \mathbb{R}. \tag{29}$$

And the update of the spectral density f at iteration k could be expressed by (Guinness, 2017; Heaton et al., 2018):

$$f_{k+1}(\boldsymbol{\omega}) = \sum_{\boldsymbol{\nu} \in \mathbb{F}_y} E_k(|J(\boldsymbol{\nu})|^2 | U) \alpha(\boldsymbol{\omega} - \boldsymbol{\nu}), \tag{30}$$

where $\mathbb{F}_y$ is a set of Fourier frequencies related to the grid for all y_s, α is a smoothing kernel function, U as the vector of observed values and E_k is the conditional expectation of the periodogram $|J(\boldsymbol{\nu})|^2$ (square of the amplitude or power) under the current zero-mean multivariate Gaussian distributed spectral density f_k, whose periodic approximation to the true covariance function could be written as (Guinness, 2017; Heaton et al., 2018):

$$R_k(h) = \frac{1}{m} \sum_{\boldsymbol{\omega} \in \mathbb{F}_y} f_k(\boldsymbol{\omega}) \exp(i2\pi\boldsymbol{\omega}h), \tag{31}$$

where h represents the distance between any two locations. Note that in the case study of Heaton et al. (2018), the conditional expectation $E_k(\cdot|\cdot)$ is replaced by $|J(\boldsymbol{\nu})|^2$ calculated with the conditional simulation of missing values given the observed values under the current approximated covariance function R_k.

Moreover, to reduce the bias caused by the smoothing effect in the iterative spectral density estimates, a parametric filtering method was introduced. With this filtering method, the parametric spectral density $f_\theta(\cdot)$ is estimated by maximizing the Whittle likelihood $l(\theta)$ (Whittle, 1954; Guinness, 2017; Heaton et al., 2018):

$$l(\theta) = -\frac{m}{2}\log 2\pi - \frac{1}{2}\sum_{\boldsymbol{\omega}\in\mathbb{F}_y}(\log f_\theta(\boldsymbol{\omega}) + \frac{\tilde{E}_k(|J(\boldsymbol{\omega})|^2|U)}{f_\theta(\boldsymbol{\omega})}). \tag{32}$$

And the resulting updated spectral density would be (Guinness, 2017):

$$f_{k+1}(\boldsymbol{\omega}) = f_{\theta_k}(\boldsymbol{\omega})\sum_{\boldsymbol{\nu}\in\mathbb{F}_y}\frac{\tilde{E}_k(|J(\boldsymbol{\nu})|^2|U))}{f_{\theta_k}(\boldsymbol{\nu})}\alpha(\boldsymbol{\omega}-\boldsymbol{\nu}), \tag{33}$$

where θ_k is the maximizer of $l(\theta)$.

2.10. Metakriging

According to Guhaniyogi and Banerjee (2018), Metakriging provides a scalable approximation framework for the full Bayesian inference of the unknown parameters involved in the prediction and recovering of the latent spatial Gaussian process under the context of a massive amount of data. First, it partitions the dataset into K subsets, then it conducts inference independently on each subset and derives the so-called 'subset posteriors'. Eventually, it will combine all the information from 'subset posteriors' into a single 'meta-posterior' for the unknown parameters of interest (Guhaniyogi and Banerjee, 2018).

To be more specific, the goal of the analysis is to make approximate Bayesian inference on $\boldsymbol{\Omega} = \{\boldsymbol{\beta}, \boldsymbol{\phi}\}$, where $\boldsymbol{\beta}$ as the unknown random coefficients, $\boldsymbol{\phi}$ as the covariance parameters (Heaton et al., 2018). Eventually, it is possible to derive one approximate posterior predictive density $p(y(s_0)|\boldsymbol{y})$ as well as one approximate posterior density for the spatial process $p(\omega(s_0)|\boldsymbol{y})$ by integrating the information from subset posteriors $p(\boldsymbol{\Omega}|\boldsymbol{y}_k)$. The subset posteriors are computed based on the Bayesian inference (Guhaniyogi and Banerjee, 2018):

$$p(\boldsymbol{\Omega}|\boldsymbol{y}_k) = p(\boldsymbol{\beta}, \boldsymbol{\phi}|\boldsymbol{y}_k) = p(\boldsymbol{\phi}|\boldsymbol{y}_k) \times p(\boldsymbol{\beta}|\boldsymbol{\phi}, \boldsymbol{y}_k), \tag{34}$$

where $\boldsymbol{\beta}$ is assigned with a Gaussian prior distribution, $\boldsymbol{\phi}$ is assigned with an inverse Gamma prior distribution.

From the Metakriging application in the case study of Heaton et al. (2018), the original dataset was randomly partitioned into $K = 30$ subsets. From each subset, there were in total $M = 2000$ sets of $\boldsymbol{\Omega}$ samples generated by Markov Chain Monte Carlo (MCMC) methods (Guhaniyogi and Banerjee, 2018; Heaton et al., 2018).

The approximation of the meta-posterior predictive distribution $p(y(s_0)|\boldsymbol{y})$ is based on (Guhaniyogi and Banerjee, 2018):

$$
\begin{aligned}
p(y(s_0)|\boldsymbol{y}) &\approx \sum_{k=1}^{K} \alpha_{\rho,k}(\boldsymbol{y}) p(y(s_0)|\boldsymbol{y}_k) \\
&= \sum_{k=1}^{K} \alpha_{\rho,k}(\boldsymbol{y}) \int p(y(s_0)|\boldsymbol{\Omega}, \boldsymbol{y}_k) p(\boldsymbol{\Omega}|\boldsymbol{y}_k) d\boldsymbol{\Omega},
\end{aligned}
\tag{35}
$$

and the empirical version of this meta-posterior predictive distribution is:

$$
\sum_{k=1}^{K} \sum_{j=1}^{M} \frac{\alpha_{\rho,k}(\boldsymbol{y})}{M} 1_{y(s_0)^{(j,k)}}. \tag{36}
$$

Similarly, the empirical meta-posterior density for the spatial random effect term could be expressed as (Guhaniyogi and Banerjee, 2018):

$$
\sum_{k=1}^{K} \sum_{j=1}^{M} \frac{\alpha_{\rho,k}(\boldsymbol{y})}{M} 1_{\omega(s_0)^{(j,k)}}. \tag{37}
$$

Observed that the key term in the equations above is the $\alpha_{\rho,k}(\cdot)$, which is a function of ρ (the resulting value computed from a positive-definite kernel function) and the observed $\boldsymbol{y}$ (Guhaniyogi and Banerjee, 2018; Heaton et al., 2018). And it is estimated by the Weiszfeld iterative algorithm using the iteratively updated Geometric Median (GM) value (Minsker et al., 2014; Guhaniyogi and Banerjee, 2018).

According to Guhaniyogi and Banerjee (2018) and Heaton et al. (2018), the geometric median π^* can be expressed as

$$
\pi^* = \sum_{k=1}^{K} \alpha_{\rho,k}(\boldsymbol{y}) p_k, \tag{38}
$$

where $\sum_{k=1}^{K} \alpha_{\rho,k}(\boldsymbol{y}) = 1$, $p_k = p(\boldsymbol{\Omega}|\boldsymbol{y}_k)$ represents the individual posterior density of data subset k and

$$
\alpha_{\rho,k}(\boldsymbol{y}) = \frac{||p_k - \pi^*||_\rho^{-1}}{\sum_{k=1}^{K} ||p_k - \pi^*||_\rho^{-1}}. \tag{39}
$$

One advantage of Metakriging is its robustness to outliers which is resulting from the data splitting (Minsker et al., 2014). The computational efficiency of this approach is realized by parallel implementation over different cores (Guhaniyogi and Banerjee, 2018).

2.11. Gapfill

Gapfill, a novel spatio-temporal predictive method proposed by Florian et al. (2018), independently predicts each missing field value in the spatial domain by the observed values at its neighboring locations. The novelty of this method comes from its purely algorithmic as well as distribution-free nature (Heaton et al., 2018).

Recall the goal of this method is to predict each spatio-temporal missing value. Note that each spatio-temporal missing value corresponds to one spatial location, at one given time point. Therefore, both spatial and temporal extents need to be specified.

The overall 'gap-filling' procedure for each missing value mainly has four steps (Florian et al., 2018):

1. Iteratively obtain a subset of data (spatially) around the target missing value at all given time points (as the method is designed for spatio-temporal analysis). This subset of data is further divided into subimages by different time points. The finally selected data subset should contain an adequate number of subimages that are not empty and an adequate number of non-missing values in each subimage.

2. Give a score and a corresponding rank to each subimage in the selected subset of data. The score for each subimage is determined by the proportion of observed values in that subimage that are greater than values observed at the same locations in all other subimages. Rank the subimages by the increasing order of the score, e.g. the subimage with the lowest score ends up getting a rank value of 1.

3. Estimate an empirical quantile for the target missing value relative to the observed values in the same subimage. This estimated quantile level is calculated by taking the average of the empirical quantile levels in all the other subimages that correspond to a non-missing value at the same spatial location.

4. Fit a quantile regression line to all the observed values at the estimated empirical quantile level $\hat{\tau}_{s_t,a_t}$ (from step 3) in the selected subset of data over different ranks, and obtain the coefficient estimates $\hat{\beta}_0(\hat{\tau}_{s_t,a_t})$ and $\hat{\beta}_1(\hat{\tau}_{s_t,a_t})$. Then, use the derived coefficient estimates as well as the rank value that matches the subimage where the target missing value is at to predict the target field value. See the regression formula and prediction equation below. Note that $Q(\cdot)$ represents the response value; x and y denote the spatial extent; s and a denote the temporal extent (e.g. s represents season and a represents year) which define the subimages; T and t means target; r is the rank:

$$Q(\hat{\tau}_{s_t,a_t}|r) = \hat{\beta}_0(\hat{\tau}_{s_t,a_t}) + \hat{\beta}_1(\hat{\tau}_{s_t,a_t})r, \tag{40}$$

$$\hat{Z}[x_T, y_T, s_T, a_T] = \hat{\beta}_0(\hat{\tau}_{s_t,a_t}) + \hat{\beta}_1(\hat{\tau}_{s_t,a_t})r_t. \tag{41}$$

The major advantage of this Gapfill method pointed out by its authors was the robustness to outliers as it inherits the properties of quantile regression. Also, since the prediction

of one missing value is independent of the prediction of another missing value, it allows the predictions of a greater amount of spatio-temporal missing values through parallel computing (Florian et al., 2018).

2.12. Local Approximate Gaussian Processes (laGP)

According to Gramacy and Apley (2015), the local approximate Gaussian Process (laGP) predicts the field value at each unobserved location s by training a Gaussian process predictor on its m observed neighbors. There are multiple ways for identifying the reasonable neighboring locations around each s which include the commonly used distance-based nearest neighbors (NNs) search. However, knowing that the NNs might not provide the best data subset for the prediction purpose, Gramacy and Apley (2015) have implemented two greedy searches to find better neighbors. One is based on minimizing the mean-square prediction error (MSPE), the other one is based on maximizing the reduction in predictive variance at s (so-called Active Learning Cohn, ALC criterion).

The inferences on the unknown parameters could be conducted based on maximizing likelihood as well as profile likelihood using Newton-like methods (Gramacy and Apley, 2015). Let D represents data and $D = (X_{N\times P}, Y_{N\times 1})$, where X denotes the design matrix and Y denotes the observed process which has a multivariate Gaussian likelihood with mean zero, covariance $\tau^2 K$. K is an $N \times N$ correlation matrix and τ^2 is an unknown variance parameter. After specifying the prior distribution of τ^2 to be inverse Gamma, based on empirical Bayes inference, the conditional probability distribution of the process Y given K is (Gramacy and Apley, 2015):

$$p(Y|K) = \frac{\Gamma[N/2]}{(2\pi)^{N/2}|K|^{1/2}} \times (\frac{\psi}{2})^{-\frac{N}{2}},\ \psi = Y'K^{-1}Y. \tag{42}$$

And the predictive distribution of the process conditioned on the observed data and correlation matrix K follows Student-t distribution with N degrees of freedom. Its mean and scale are expressed as:

$$\begin{aligned} \mu(s|D,K) &= k'(s)K^{-1}Y, \\ \sigma^2(s|D,K) &= \frac{\psi[K(s,s) - k'(s)K^{-1}k(s)]}{N}, \end{aligned} \tag{43}$$

where $k'(s)$ is a vector with length N whose jth component is $K(s, s_j)$ (Gramacy and Apley, 2015).

The pure local approximate Gaussian Process (laGP) method does not intend to relieve the computational burden of large spatial data analysis by reducing the rank of the covariance matrix, as with FRK, or try to come up with any sparse covariance matrices. There is a great amount of computation involved with this local approximate Gaussian

process but it can be conducted properly through 'multi-core parallelization' and 'multi-node automation' since the prediction procedure at one location point is independent of the prediction procedure at another location point (Gramacy and Apley, 2015; Heaton et al., 2018).

3. Replication Study on Simulated Dataset

3.1. The Simulated Dataset

Heaton et al. (2018) have compared the predictive performances of twelve spatial interpolation methods on a simulated dataset. They claimed that the simulated dataset represents the scenario where the covariance function was correctly specified. That is to say the data are simulated from a specified true latent spatial process model plus some sampling schemes that best approximate real-world scenarios (Wikle, Zammit-Mangion and Cressie, 2019). The 150,000 simulated observations were based on 2,500 observed daytime land surface temperatures from MODIS satellite image (Level-3 data) on August 4, 2016 (Heaton et al., 2018). A Gaussian process model with constant mean, exponential covariance function and a measurement error term (or nugget effect) is fitted as (Heaton et al., 2018):

$$\begin{aligned} \boldsymbol{Y}(\boldsymbol{s}) &= \boldsymbol{\mu}(\boldsymbol{s}) + \boldsymbol{\omega}(\boldsymbol{s}) + \boldsymbol{\varepsilon}(\boldsymbol{s}) \\ &= \boldsymbol{X}(\boldsymbol{s})\boldsymbol{\beta} + \boldsymbol{\omega}(\boldsymbol{s}) + \boldsymbol{\varepsilon}(\boldsymbol{s}), \end{aligned} \tag{44}$$

where spatial random effect term $\boldsymbol{\omega}(\boldsymbol{s})$ is assumed to have an exponential covariance structure (which is also equivalent to a Matern covariance structure setting the smoothing parameter to $\frac{1}{2}$), $\boldsymbol{\mu}(\boldsymbol{s}) = \boldsymbol{X}(\boldsymbol{s})\boldsymbol{\beta}$ represents the general trend and $\boldsymbol{\varepsilon}(\boldsymbol{s})$ is the measurement error with Gaussian white noise structure $\text{GP}(\boldsymbol{0}, \sigma^2_{\boldsymbol{\varepsilon}}\boldsymbol{I})$. The unknown parameters involved in this assumed Gaussian process are estimated by maximizing likelihood (Heaton et al., 2018).

Given the information that the data is simulated partly from the true spatial process, it allows us to evaluate how effective different prediction methods can approximate the true underlying spatial process. This evaluation is done by comparing the predicted values with the simulated ones. In the study of Heaton et al. (2018), five numerical scoring metrics were computed based on the simulated true temperatures as well as the predicted temperatures. These scoring metrics will be discussed in the next subsection.

3.2. Numerical Scoring Metrics for Predictive Performance Comparison

To assess and compare the twelve methods' predictive performances, Heaton et al. (2018) have calculated the following numerical scoring metrics: mean absolute error (MAE), root mean squared error (RMSE), continuous rank probability score (CRPS), interval score (INT) and prediction interval coverage (CVG).

3.2.1. MAE and RMSE

The mean absolute error (MAE) is calculated by:

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|\hat{\boldsymbol{z}}(\boldsymbol{s}_{0_i}) - \boldsymbol{z}(\boldsymbol{s}_{0_i})|, \tag{45}$$

where $\boldsymbol{s}_0$ denotes the m locations where predictions would take place, $\hat{\boldsymbol{z}}(\cdot)$ represents the predicted attribute values (temperatures) that can reflect the complex process modeled by the selected method and $\boldsymbol{z}(\cdot)$ represents the true attribute values (temperatures) from simulated dataset.

Using the same notations for the to-be-predicted locations, predicted temperatures as well as the true temperatures, the root mean sqaured error (RMSE) is formulated as:

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{\boldsymbol{z}}(\boldsymbol{s}_{0_i}) - \boldsymbol{z}(\boldsymbol{s}_{0_i}))^2}. \tag{46}$$

MAE and RMSE are the two metrics chosen by Heaton et al. (2018) for assessing the prediction accuracy. The smaller the calculated metric value, the more accurate the predictions are made. And RMSE seems to give greater weights to large discrepancy between predicted and actual temperatures. According to Willmott and Matsuura (2006), however, MAE is more of a natural measure than RMSE for computing the average spatial interpolation error. They claim that RMSE not only describes the average error, but also other elements such as the variance of the error frequency distribution.

3.2.2. CRPS, INT and CVG

The continuous rank probability score (CRPS), interval score (INT) and prediction interval coverage (CVG) are used for quantifying the prediction uncertainty (Heaton et al., 2018). In other words, these scores' major task is to evaluate the variability of the predicted values if we are going to make such predictions repeatedly. The score values can shed light on questions such as: how likely are we going to produce another prediction value that is far off the current one? how likely the prediction intervals will cover the true values?

According to Gneiting and Raftery (2007), the continuous ranking probability score (CRPS) for predictions with a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ can be calculated by:

$$\text{CRPS}(\mathcal{N}(\mu, \sigma^2), x) = \sigma[\frac{1}{\sqrt{\pi}} - 2\varphi(\frac{x-\mu}{\sigma}) - \frac{x-\mu}{\sigma}(2\Phi(\frac{x-\mu}{\sigma}) - 1)], \tag{47}$$

where x denotes the true observed value being held out, φ represents the probability density function (pdf) and Φ represents the cumulative distribution function (cdf) of a standard Gaussian distribution. Heaton et al. (2018) computed the negative oriented

CRPS in their study and replaced the true probability distribution parameters μ and σ^2 with the estimated mean $\hat{\mu}$ and estimated variance $\hat{\sigma}^2$ from each of the predictive methods, which was simply $-\text{CRPS}(\mathcal{N}(\hat{\mu}, \hat{\sigma}^2), x)$. Then they summarized the $-$CRPS scores for all locations by taking the average. The negatively oriented version of CRPS indicates how close the probabilistic forecasts and the deterministic forecasts are (Gneiting and Raftery, 2007). The smaller the $-$CRPS, the more precise the predictions are being made.

The interval score (INT), as implied by its name, is closely related to the characteristics of prediction interval. It can be calculated by (Gneiting and Raftery, 2007):

$$\text{INT}(l, u; x) = (u - l) + \frac{2}{\alpha}(l - x)\mathbb{1}\{x < l\} + \frac{2}{\alpha}(x - u)\mathbb{1}\{x > u\}, \tag{48}$$

where l is the lower bound of the prediction interval, u is the upper bound of the prediction interval, thus $u - l$ denotes the width of the prediction interval. Note that, a prespecified significance level α, the population mean μ and population variance σ^2 are involved in the calculation of l and u. x, again is the true value being held out; $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ represents the specified predictive quantiles; and $\mathbb{1}\{\cdot\}$ denotes the indicator function. By holding a fixed α for all predictive/interpolation methods, the method with a narrower prediction interval will obtain a smaller interval score. Thus, smaller INT indicates smaller variance from the predicted values. Heaton et al. (2018) have chosen α to be 0.05 and the estimated mean and variance, $\hat{\mu}$ and $\hat{\sigma}^2$, are plugged into the equation above.

The calculation of prediction interval coverage (CVG), compared with the previous two scores, is more straightforward. A CVG score is simply computed by the proportion of the prediction intervals that have covered the true values (Heaton et al., 2018), which can be expressed by:

$$\text{CVG}(l, u; x) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{x > l \text{ and } x < u\}, \tag{49}$$

again x is the true value being held out, l and u denote the lower and upper bounds of the prediction interval respectively, m is the total number of observations being predicted and $\mathbb{1}\{\cdot\}$ is the indicator function. If the empirical probability distribution perfectly reflects the assumed probability distribution from the model, the CVG value will be very close to $1 - \alpha$, where α is the prespecified significance level. Heaton et al. (2018) have chosen α to be 0.05 and the estimated prediction probability distribution parameters, $\hat{\mu}$ and $\hat{\sigma}^2$, are used in the computation of l and u.

3.3. Predictive Performance Comparison

The tables below show the MAE, RMSE, CRPS, INT and CVG scores for each of the predictive methods implemented in the case study of Heaton et al. (2018) on the simulated data. Heaton et al. (2018) have published their simulated dataset

and the reproducible code for twelve methods here: https://github.com/finnlindgren/heatoncomparison. The scores from Table 1 were generated from the replication study. The scores from Table 2 were from the case study competition (Heaton et al., 2018). Note that for Metakriging, the replicated results were filled with NAs as it failed to finish the predictions for all of the 44,431 missing values even after running for over 125 hours. One possible cause of this endless running time might be the removal of the latest version 'Mposterior' R package, which was critical in the computation, from CRAN repository. For NNGP Response, the predictions were conducted by functions from 'spNNGP' R package in the replication study, instead of the direct compilation of C++ program provided by Heaton et al. (2018), plus some post-MCMC steps to compute the predicted values. Furthermore, for Predictive Processes, the MCMC parameter samples were derived by compiling the given C++ program and the posterior predictive samples were computed in R based on equations introduced in Finley et al. (2015).

Method	**MAE**	**RMSE**	**CRPS**	**INT**	**CVG**
FRK	1.02	1.33	0.75	8.37	0.84
Gapfill	0.73	1.00	0.64	18.01	0.44
Lattice Krig	0.63	0.87	0.45	4.04	0.97
LAGP	0.77	1.08	0.56	5.44	0.92
Metakriging	NA	NA	NA	NA	NA
MRA	0.61	0.83	0.43	3.64	0.93
NNGP Conjugate	0.65	0.88	0.46	3.80	0.96
NNGP Response	0.65	0.88	0.46	3.84	0.96
Partition	0.64	0.86	0.47	5.05	0.86
Pred. Proc	1.08	1.56	0.79	8.96	0.89
SPDE	0.62	0.86	0.45	4.05	0.97
Tapering	0.72	1.01	0.58	6.83	1.00
Periodic Embed.	0.64	0.90	0.46	4.04	0.96

Table 1: Numerical scores for each method replicated on the simulated data.

Method	**MAE**	**RMSE**	**CRPS**	**INT**	**CVG**
FRK	1.03	1.31	0.74	8.35	0.84
Gapfill	0.73	1.00	0.64	18.01	0.44
Lattice Krig	0.63	0.87	0.45	4.04	0.97
LAGP	0.79	1.11	0.57	5.71	0.90
Metakriging	0.74	0.97	0.53	4.69	0.99
MRA	0.61	0.83	0.43	3.64	0.93
NNGP Conjugate	0.65	0.88	0.46	3.79	0.96
NNGP Response	0.65	0.88	0.46	3.81	0.96
Partition	0.64	0.86	0.47	5.05	0.86
Pred. Proc	0.89	1.21	0.79	12.75	0.77
SPDE	0.62	0.86	0.59	7.81	1.00
Tapering	0.69	0.97	0.55	6.39	1.00
Periodic Embed.	0.65	0.91	0.47	4.16	0.97

Table 2: Numerical scores for each method on the simulated data given by Heaton et al. (2018).

In general, the reproduced predictive performances are similar to the ones from Heaton

et al. (2018). Given the information that the data were simulated from a Gaussian Process with an exponential covariance function that only depends on the Euclidean distance between locations, the prediction accuracy of each method, excluding the pure-algorithmic Gapfill (Heaton et al., 2018), was naturally good as the correct covariance structure was specified. Among those negative oriented numerical scoring metrics, MRA has hit the most of the lowest scores and NNGPs have the prediction interval coverage closest to 0.95. Predictive Process method does show a relatively large discrepancy between the original and reproduced results. This might be caused by slightly different ways of conducting the post-MCMC predictions. The reproduced results were generated from unmodified The Predictive Processes, while as pointed out by Heaton et al. (2018), there is also a modified version of Predictive Processes which adds the finer scale error term to avoid underestimating the marginal variance of the original process. Although the code for this post-MCMC prediction under the Predictive Processes approach was not provided by Heaton et al. (2018), it is a reasonable guess that they employed the modified Predictive Processes as we can see their INT score is 12.75, larger than 8.96 from the replication study. The key parts of the code for reproducing the results in Table 1 are enclosed in Appendix A.

Table 3 and Table 4 summarize the implementation options as well as the required computational resources, in terms of run time and number cores used, from both the Heaton et al. (2018) case study competition and replication study.

Method	**Implemented By**
Gapfill	R package 'gapfill'
NNGP Conjugate	R package 'spNNGP'
FRK	R packages 'FRK', 'INLA'
LAGP	R packages 'LatticeKrig', 'tgp', 'laGP', 'plgp'
Periodic Embedding	R package 'npspec'
MRA	MATLAB M-Files
Lattice Krig	R packages 'LatticeKrig'
NNGP Response	C++ program, R functions
Partition	R functions, R package 'LatticeKrig'
SPDE	R package 'INLA'
Covariance Tapering	R functions, R packages 'fields', 'gapfill'
Predictive Processes	C++ program, R functions; or R package 'spBayes'
Metakriging	R packages 'spBayes', 'Mposterior'

Table 3: Implementation option(s) for each method. For methods that have more than one R packages involved, only the key R packages are listed in this table. For more information about the implementation details, please refer to Appendix A.

Recall that for NNGP response, Heaton et al. (2018) compiled the C++ program for MCMC procedure and wrote R functions to further generate posterior predictive samples. However, in the replication study, both inference and prediction steps were completed by functions from the R package 'spNNGP'. Thus, the number of cores used in this method, see Table 4, are not very comparable between the original and replication study. Moreover, in the replication study, methods that required a higher level of parallelization that needed more than 12 logical processors, larger RAM or simply longer running

Method	Time1 (Min)	Time2 (Min)	Cores Used1	Cores Used2
Gapfill	0.63	1.43	40	12
NNGP Conj.	1.99	2.14	10	10
FRK	2.18	1.15	1	1
LAGP	2.28	5.70	40	40
Embedding	13.31	14.04	1	1
MRA	13.57	24.35	1	1
LatticeKrig	25.58	53.42	1	1
NNGP Resp.	45.06	61.17	10	12
Partition	77.56	542.12	55	39
SPDE	138.34	120.83	2	2
Tapering	188.36	136.37	1	1
Pred. Proc.	639.23	761.27	1	1
Metakriging	2888.89	NA	30	30

Table 4: Run time and level of parallelization (in terms of the number of cores used) comparison between Heaton et al. (2018) case study competition and replication study. Time1 and Cores Used1 are from case study competition; Time2 and Cores Used2 are from replication study.

time were implemented using CLA Compute Cluster from LATIS Research, University of Minnesota. These methods include: LAGP, NNGP response, Spatial Partitioning, SPDE, Covariance Tapering, Predictive Processes, and Metakriging. M-files from MRA were run using the MATLAB online. The rest of methods were implemented using an individual laptop with Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, 2208 Mhz, with 6 physical Core(s), 12 Logical Processor(s). As is shown in Table 3 and Table 4, most methods are replication friendly in the sense that they have relevant R packages available and a medium level of parallelization which requires no more than 10 cores (in the case study competition) or 12 cores (in the replication study) to control the running time to be below 13 hours, or 780 minutes. However, Metakriging needs some extra attention for its implementation issues.

4. Applications on Sea Level Pressure Dataset

4.1. Sea Level Pressure (SLP) Dataset

This section will describe the applications of two spatial interpolation methods from Heaton et al. (2018) on the NOAA Extended Reconstructed Sea Level Pressure (SLP) dataset (Smith and Reynolds, 2004). The dataset can be downloaded from https://www.esrl.noaa.gov/psd/gcos_wgsp/Gridded/data.noaa.erslp.html. The SLP dataset contains 16,020 locations on a 180×89 grid with latitude ranging from -88 to 88 and longitude ranging from 0 to 358, which generally covers the extent of the entire world. For each location point in the dataset, it has monthly mean SLP values (in millibar) recorded for 1,728 months from Jan, 1854 to Dec 1997. In fact, at each timestamp, 4,946 out of the 16,020 location points (approximately 31% of them) do not have their SLP values missing. These missing values all come from land areas. The raw data was in netCDF (network Common Data Form) and was transformed into a data frame in R by the package 'ncdf4'. The code for SLP data pre-processing is attached in Appendix B.

4.2. Sea Level Pressure Interpolations for Land Areas

The goal of this analysis is to predict the missing SLP values for land areas based on the observed SLP values for all 1,728 months using two of the spatial interpolation methods introduced above. As we are mainly interested in the spatial interpolation performance in this study, we further assume that different timestamps are independent with each other.

The selected methods for spatial interpolation on SLP dataset are Nearest Neighbor Gaussian Process Conjugate (NNGP Conjugate) as well as Fixed Rank Kriging (FRK). The main reason for choosing these methods is that they are faster to implement. Also, according to the results from the case study of Heaton et al. (2018), NNGP had relatively good prediction performances whether a correct covariance structure was specified or not. Thus, the NNGP approach might be worth another try on the real-world SLP dataset with unknown covariance structure for its underlying spatial process.

Although an exact covariance structure for the SLP dataset is not known, it is still possible to get a grasp of it by summarizing the observed data. One common way is to create empirical semivariogram (or variogram) and compare it with the fitted theoretical semivariograms, see Figure 1. Note that since we have in total 1,728 months of SLP data for one spatial domain of interest, it is not practical to check the semivariograms for all 1,728 months. Also, with the independence assumption along the time aspect, one timestamp is deemed to be adequate to check the spatial autocorrelation structure. Therefore, the empirical semivariogram as well as the fitted variogram models in Figure 1 were based on one randomly chosen month. According to Schabenberger and Gotway (2005), the semivariogram is defined as

$$\begin{aligned}\gamma(\mathbf{s},\mathbf{h}) &= \frac{1}{2}\mathrm{Var}[Z(\mathbf{s}) - Z(\mathbf{s}+\mathbf{h})] \\ &= \frac{1}{2}\{\mathrm{Var}[Z(\mathbf{s})] + \mathrm{Var}[Z(\mathbf{s}+\mathbf{h})] - 2\mathrm{Cov}[Z(\mathbf{s}), Z(\mathbf{s}+\mathbf{h})]\}.\end{aligned} \tag{50}$$

When the spatial process is second-order stationary, the impact of the absolute coordinates will be eliminated. The covariance function can then be expressed by $\mathrm{Cov}[Z(\mathbf{s}), Z(\mathbf{s}+\mathbf{h})] = C(\mathbf{h})$ (Schabenberger and Gotway, 2005). Examples of second-order stationary covariance functions include Matern covariance function, exponential covariance function, Gaussian covariance function as well as spherical covariance function. With the assumption of second-order stationary, the semivariogram can be rewritten as a function that only depends on the distances between any two locations in the spatial domain, namely (Schabenberger and Gotway, 2005):

$$\gamma(\mathbf{h}) = \frac{1}{2}\{2\sigma^2 - 2C(\mathbf{h})\} = C(\mathbf{0}) - C(\mathbf{h}). \tag{51}$$

In the equation above, $C(\mathbf{h})$ will gradually decrease as the lag distance $\mathbf{h}$ between any two location points increases. $C(\mathbf{0}) = \sigma^2$ is called the sill, the shortest lag distance h' at which $C(\mathrm{h}') = 0$ is called the range and the discrepancy between $\gamma(\mathbf{0})$ calculated from

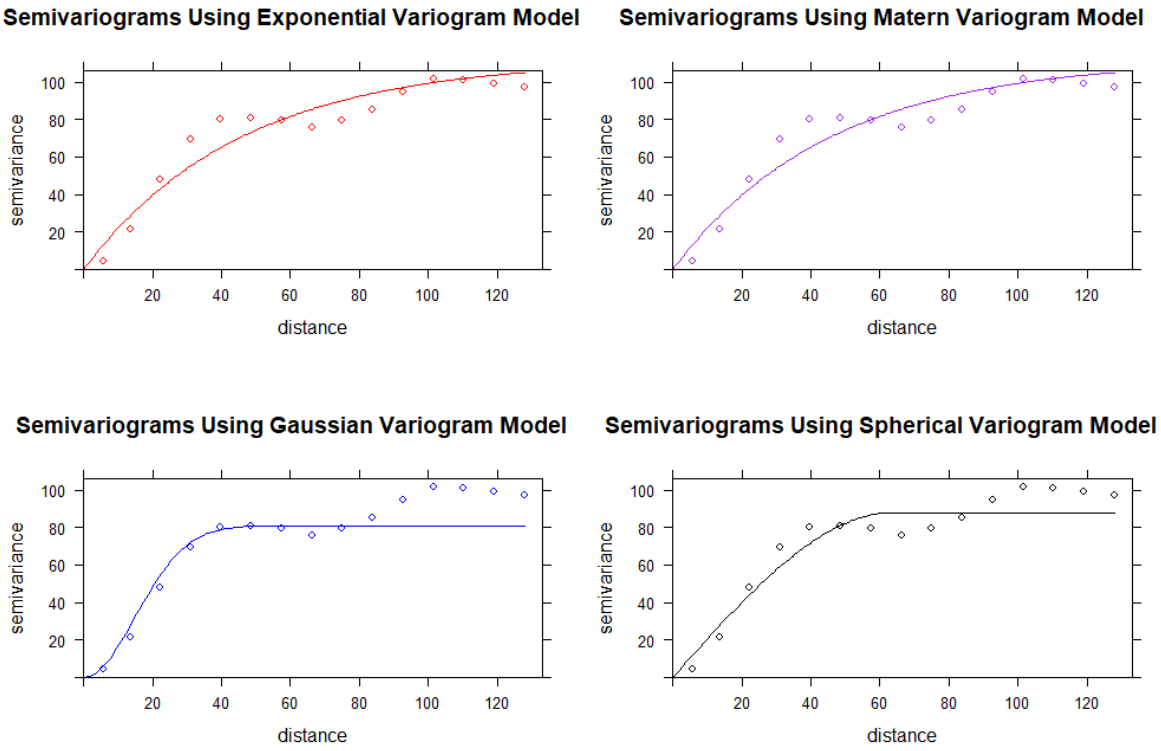

Figure 1: Empirical and theoretical semivariograms. Dotted lines represent the empirical semivariograms and the solid lines represent the fitted theoretical semivariograms. Note that the smoothness parameter from the fitted Matern variogram is 0.5 which implies this Matern variogram model is equivalent to an exponential variogram model.

the actual data and 0 is termed as the nugget effect (Schabenberger and Gotway, 2005). The nugget effect is usually composed of the latent fine-scale process as well as the measurement error (Schabenberger and Gotway, 2005). The sill, range and nugget also contribute to the parameters set of the variogram models. For example, in Figure 1, all of the variogram models have nugget effects approximately 0; both of the exponential and the Matern variogram models have sill = 112 and range = 46; Gaussian variogram model has sill = 81 and range = 21; spherical variogram model has sill = 88, range = 64. From Figure 1, none of the proposed covariance structure looks very inappropriate on the empirical data. The best-fitted variogram model selected from the 'gstat' R package was the spherical one.

For each of the NNGP Conjugate and FRK methods, an intercept-only model without any other covariates, a model that using longitude and latitude as its covariates and a model with month as a covariate were fitted. Note that for the models using longitude and latitude as covariates, though they cannot be regarded strictly as the trend surface models (Chorley and Haggett, 1965), the name of 'trend surface' was borrowed to represent relevant models using coordinates as predictor variables. Since it is possible to explicitly specify the covariance structure using 'spNNGP' package in R, both the

exponential and spherical covariance functions were tried in the NNGP Conjugate models. Recall that the SLP dataset has about 31% of its locations with missing SLP values. When applying the automatic basis function placement function from 'FRK' R package, it is inevitable to have basis functions placed in the continental regions with missing SLP values which might cause problems for the model fit (Zammit-Mangion, 2018). Thus, part of the basis functions landed in the continental regions were removed manually.

Note that the dataset for fitting the 'month' models was different from the commonly used SLP dataset (with wide format) for the other models. One natural data frame structure for 'month' models would be the long-format data and it can be easily reshaped from the wide-format one. However, the resulting dataset will have $n = 16,020 \times 1,728 = 27,682,560$ observations in total and can drastically increase the computation costs no matter what methods to implement. Also, the duplicate coordinates from the long format data will trigger errors when running functions from 'spNNGP' package. Recall the independence assumption among the timestamps (1,728 months), a temporary dataset with a dimension of $16,020 \times 4$ was thus generated 1,728 times iteratively. The four variables are: lon (longitude), lat (latitude), slp (SLP value) and month (from 1 to 12). During each iteration, for each of the 16,020 unique location points, a number was randomly sampled from 1 to 1,728 and the corresponding SLP value was assigned to the variable slp. Since the time range of the original dataset is from Jan 1854 to Dec 1997, it is reasonable to let 1 denote the first month (Jan 1854) and 1,728 denote the last month (Dec 1997). Then, the exact month could be inferred from the remainder of the sampled number divided by 12. For instance, if the remainder is 0, then the month should be December.

Figure 2 demonstrates the prediction results from the fitted models. Note that for each model, the summarized predicted values of each location point for all 1,728 months or 1,728 composite data frames (for the case of 'month' models) were aggregated by taking the average over 1,728 predicted monthly mean SLPs. The resulting value for each location can be regarded as the predicted SLP annual mean. Based on the results shown in Figure 2, the predicted annual mean SLPs generally look similar among all the models fitted from NNGP Conjugate and FRK, except that FRK approach gives much higher values for the Antarctica region that have longitudes near 0 degree. Both methods have shown some local abrupt changes in the summarized predicted SLP values, with more of them occur in Antarctica for FRK and central areas of any continents for NNGP Conjugate. Figure 3 has mapped out the pooled prediction standard errors (square root of the average prediction variance) summarized from all the fitted models. The intercept-only and trend surface models fitted using NNGP Conjugate have almost all of their pooled prediction standard errors below 5 mb, while the intercept-only and trend surface models fitted using FRK have generally higher pooled prediction standard errors plus some very high cases appear in Antarctica. In addition, no distinction between exponential and spherical covariance structures is reflected in the summarized predicted values as well as pooled prediction standard errors from the corresponding NNGP Conjugate models.

As for the models using month as a covariate, their predicted annual mean SLP values do not visually look different from the ones obtained from other models. However, their pooled prediction standard errors are much higher, see Figure 3. One possible explanation of this result might be the greater nugget effect appear in the composite dataset from which the models are fitted. According to Cressie and Katzfuss (2011), the resulting kriging standard errors can be influenced by the nugget which in FRK is estimated by a variogram model. Whether the nugget effect is estimated by the variogram model or not in our selected interpolation methods, it might still be valuable to check the impact of new data structure on the empirical semivariogram. Comparing the empirical semivariograms for Jan, 1854 from the original dataset and a composite dataset, see Figure 4, the composite dataset does show a larger nugget value. Besides, in the models using month as a covariate, the number of large pooled prediction standard errors from NNGP Conjugate model tends to exceed the one from FRK model.

Note that the model fits and predictions were conducted using the CLA Compute Cluster from LATIS Research, University of Minnesota. The relevant code is enclosed in Appendix B.

For this geostatistical predictive analysis, the missing patterns for all 1,728 timestamps are the same and the missingness is determined by the surface type. All the observed SLP values come from the ocean while all the unobserved/missing SLP values are from the land (Smith and Reynolds, 2004). Thus, it is not possible to specify a different missing pattern for land area SLPs since all the locations on the land are unsampled. As a result, there is no direct way to assess the resulting predicted values. However, the Climate Prediction Center from NOAA National Weather Service does publish images for interpolated annual mean SLP values, monthly mean SLP values and pentad mean SLP values summarized from 1979 to 1995. They can be found here: https://www.cpc.ncep.noaa.gov/products/precip/CWlink/clim.table.html. Thus, one further step of the analysis would be summarizing and visualizing the predicted SLP values only for the months from 1979 to 1995.

Recall that we have a complete set of observations from ocean areas, it is thus possible to hold out part of the truely observed samples for predictive outcome evaluation. The next subsection will describe a predictive analysis for ocean area SLP values as well as a post-prediction evaluation.

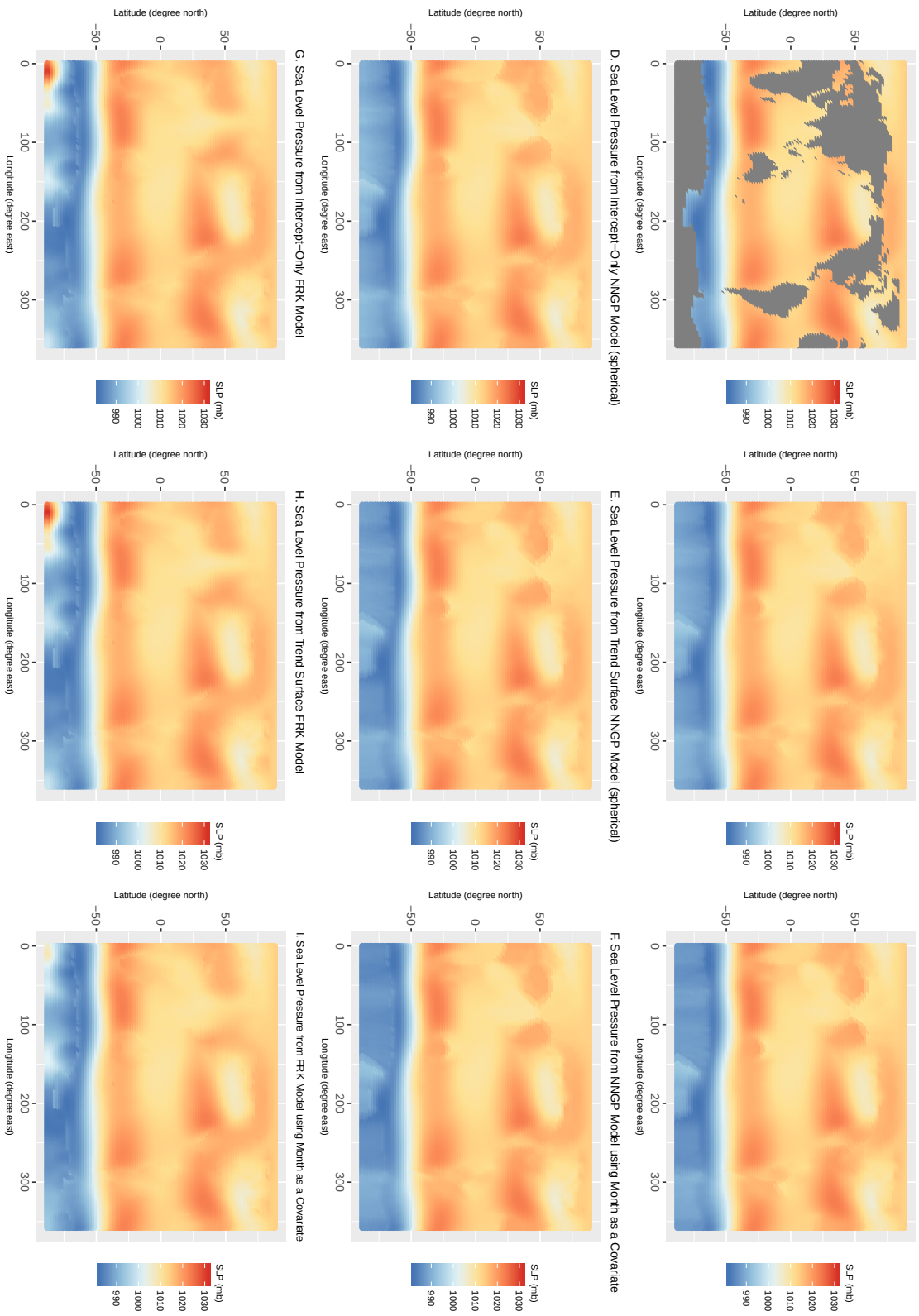

Figure 2: Visualizations of the observed SLPs from the original dataset and the predicted annual mean land SLPs from the models fitted using NNGP Conjugate and FRK methods.

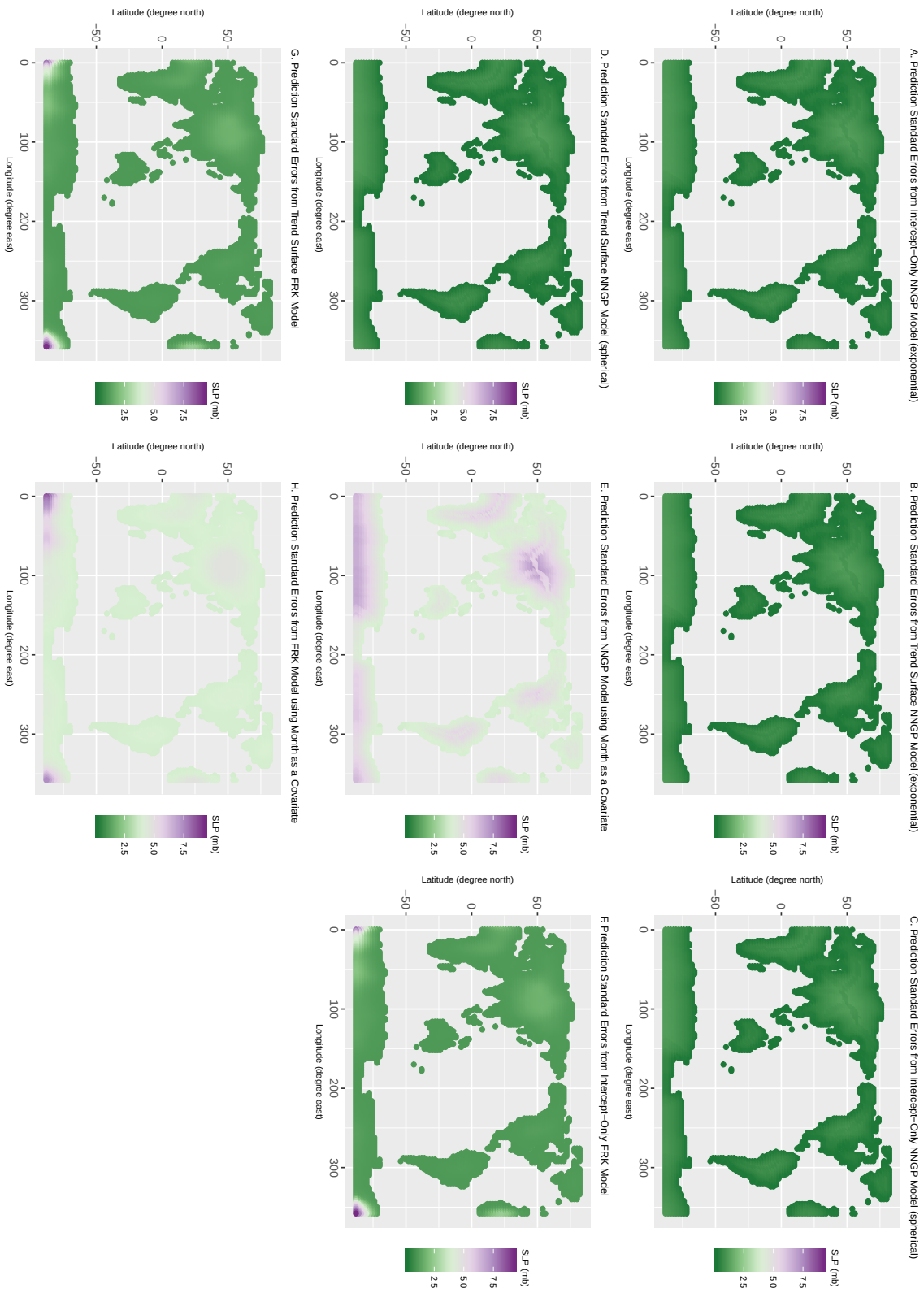

Figure 3: Visualizations of pooled prediction standard errors of the predicted annual mean land SLPs from the models fitted using NNGP Conjugate and FRK methods.

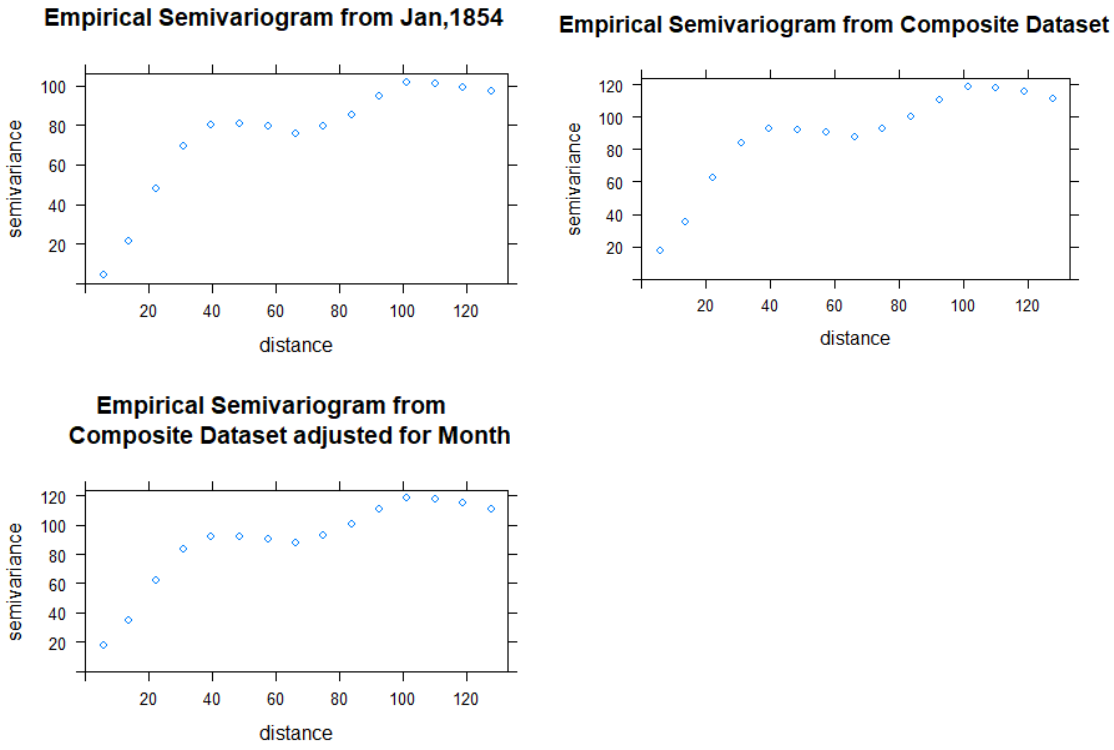

Figure 4: Empirical semivariograms from Jan, 1854 and a composite data with month as a covariate. Note that no big change in the empirical semivariogram of composite dataset after accounting for month effect.

4.3. Sea Level Pressure Interpolations for Ocean Areas

This analysis aimed to predict the annual mean ocean SLP values aggregated by the monthly mean data (from the original dataset) at randomly chosen hold-out locations from the year 1990 to 1997 and evaluate the predictive performances by computing the numerical scores given by Heaton et al. (2018). Since we were only interested in the ocean SLP values, all of the land area related location points were dropped from the dataset at the beginning. Also, note that the independence assumption in the time dimension remained throughout the whole study, this analysis ignored the correlations among different months. The NNGP Conjugate approach was chosen for this analysis because of its accessible implementation from R package 'spNNGP', low computational burden and superb predictive performances shown in the case study competition of Heaton et al. (2018). Two models were fitted, one intercept-only model and one trend surface model using longitude and latitude as covariate variables. Both the models utilized the spherical covariance function. The training data was set to the monthly mean ocean SLP values at those non-hold-out locations from the year 1854 to 1980.

As the goal of our analysis is to make predictions on the annual mean ocean SLP values for the year 1990 to 1997 using the given monthly means from NOAA (Smith and Reynolds, 2004), some level of data aggregation is inevitable, either before or after the model fits. This analysis has explored both cases.

In the case of data aggregation occurring before any model fits, the training data became the annual mean SLP values calculated by taking the average of the monthly mean SLP values at non-hold-out locations from the year 1854 to 1980. Then two model fits (intercept-only and trend surface models) using NNGP Conjugate were conducted. The predicted annual mean ocean SLPs were produced in seconds, see Table 5. These predicted values were then compared with the true annual mean ocean SLPs aggregated by monthly means from the year 1990 to 1997. According to Table 5, the numerical scores for both intercept-only NNGP and trend surface NNGP models appeared to be very close. They have shown very good prediction accuracy if we compare their MAEs (approximately 0.4) and RMSEs (approximately 0.5) with 40.72, the range of true 1990-1997 annual mean ocean SLPs. The CRPSs (approximately 0.3) and INTs (approximately 5.0) are small and CVGs (approximately 0.7) seem to be quite off the target 0.95. Since we ignored the temporal dependence in this analysis, the resulting prediction standard errors as well as the numerical scores calculated based on them, such as CRPS, INT and CVG, will not be trustworthy.

Model Type	MAE	RMSE	CRPS	INT	CVG	Run Time (Sec)
Intercept-only NNGP Conj.	0.397	0.520	0.308	5.013	0.707	3.26
Trend surface NNGP Conj.	0.397	0.519	0.307	4.942	0.716	3.43

Table 5: Numerical scores and run time (in second) for each type of NNGP Conjugate models fitted by annual mean ocean SLP values from 1854 to 1980. The predicted values were compared with the true annual mean ocean SLPs aggregated by the monthly mean ocean SLPs from 1990 to 1997. Note that for each type of NNGP Conjugate model, 10 cores were used.

Figure 5 visually compares the predicted 1990-1997 annual mean ocean SLPs with true values. The predictions from the two models look almost the same and are very close to their actual values. Figure 6 compares the prediction standard errors for the predicted 1990-1997 annual mean ocean SLPs. Most of the prediction standard errors are shown to be smaller than 0.35. Again, one needs to be cautious about these standard errors when interpreting the analysis results as the analysis did not consider the temporal dependency.

In the case of aggregation occurring after the model fits, the training data remained to be the 1854-1980 monthly mean ocean SLPs at the non-hold-out locations. For each type of the NNGP Conjugate models (intercept-only or trend surface one), 1,524 models were fitted as there were 1,524 months from the year 1854 to 1980. After each model fit, a set of predictions at the hold-out locations were generated for each of the 96 months from 1990-1997. To effectively store the predicted results, after each model fit, the predicted monthly means were aggregated by taking the average over 96 months and their corresponding prediction standard errors were pooled by taking the square root of the average prediction variance. Eventually, the 1,524 sets of aggregated predicted ocean SLPs were further aggregated by taking the average over 1,524 months; the 1,524 sets of prediction standard errors were pooled again by the pooling method just mentioned. Then, the prediction accuracy and uncertainty were assessed again by computing the five numerical scores, see Table 6.

Model Type	MAE	RMSE	CRPS	INT	CVG	Run Time (Min)
Intercept-only NNGP Conj.	0.398	0.523	0.301	4.170	0.772	211.23
Trend surface NNGP Conj.	0.397	0.519	0.300	4.131	0.772	176.17

Table 6: Numerical scores and run time (in minute) for each type of NNGP Conjugate model fitted by monthly ocean SLP data from 1854 to 1980 (127 years). For each hold-out location, the predicted monthly ocean SLPs were aggregated by taking the mean values across the 1,524 months (127 year times 12 months) where 1,524 models were fitted. The aggregated predicted values were then regarded as the predicted 1990-1997 annual mean ocean SLPs. These resulting predicted annual means were then compared with the true annual mean ocean SLP values aggregated by the monthly mean ocean SLPs from 1990 to 1997. Note that for each type of the NNGP Conjugate models listed, 10 cores were used.

By comparing Table 6 with Table 5, the results from the case of aggregation after model fits have demonstrated similar prediction accuracy in terms of the MAEs and RMSEs. Their CRPSs and INTs are slightly lower. The CVGs have been increased from 0.71 to 0.77. In general, predictive performances for the annual mean SLPs have been improved a little bit and the details about the predicted monthly means have been kept. However, these results took hours to complete all the model fits and predictions, see Table 6. The visualizations for this case are not expected to be very different from Figure 5 and Figure 6, thus they will not be shown here.

Note that the analysis for the case of aggregation before model fits was conducted on the individual laptop mentioned in Section 3. The analysis for the other case was conducted using CLA Computer Cluster from LATIS Research, University of Minnesota. The code

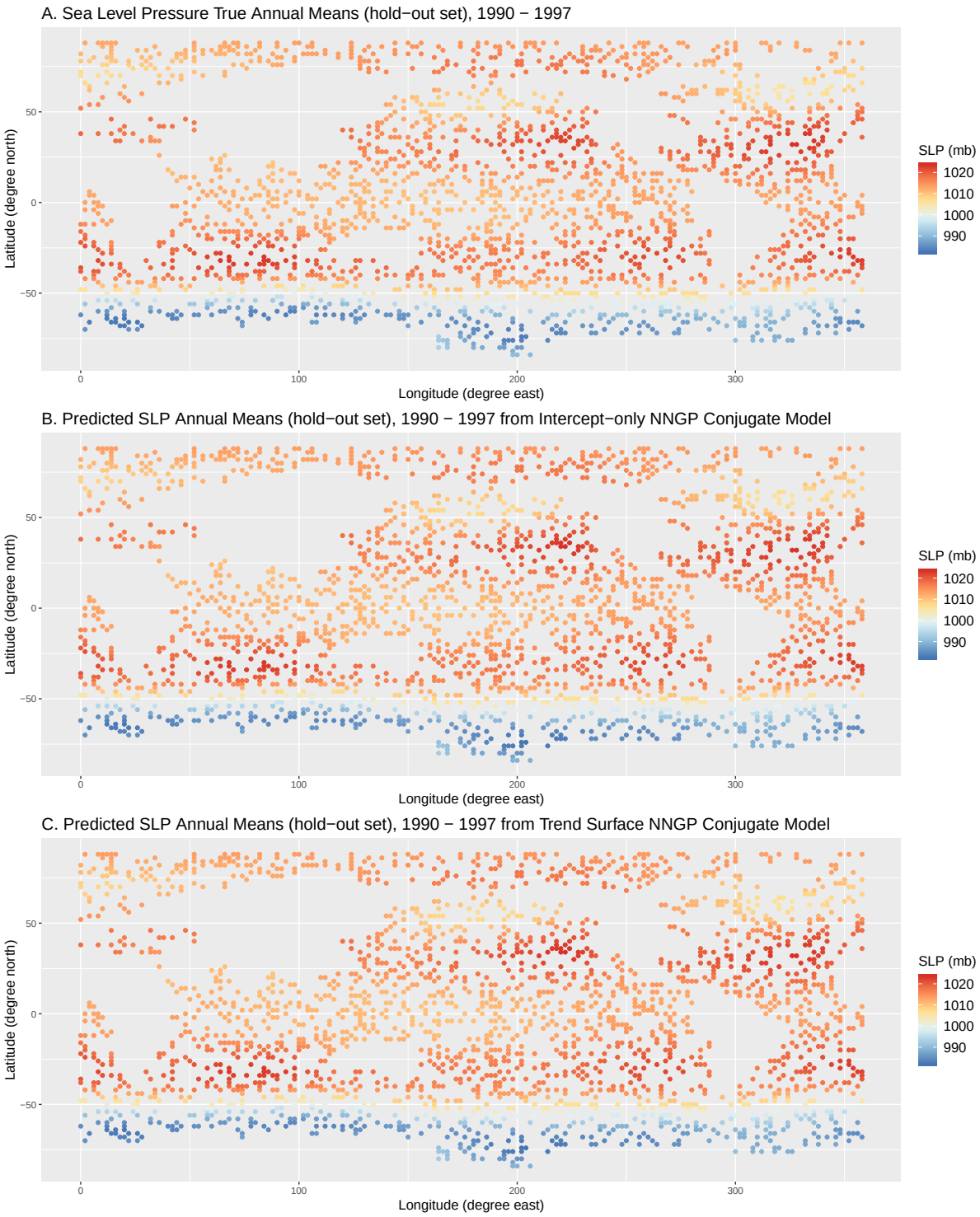

Figure 5: Visualizations of the true 1990-1997 annual mean ocean SLPs at hold-out location and the predicted values from the two models fitted using NNGP Conjugate based on 1985-1980 annual mean ocean SLPs.

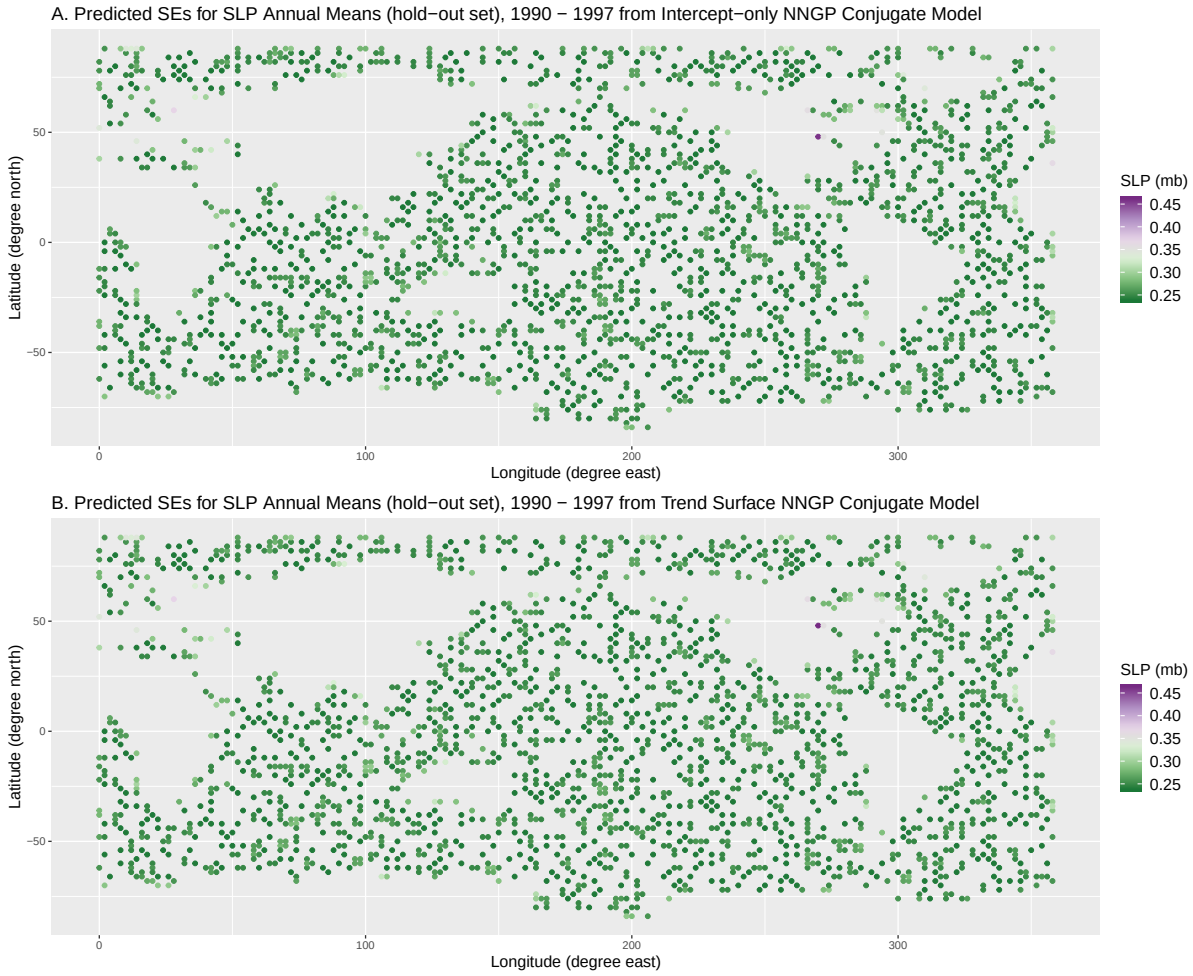

Figure 6: Visualizations of the prediction standard errors of predicted 1990-1997 annual mean ocean SLPs from the two models fitted using NNGP Conjugate based on 1854-1980 annual mean ocean SLPs .

for the entire analysis is provided in Appendix B.

5. Discussion and Conclusions

This paper intended to: (1) review some basic concepts in geostatistics and twelve predictive methods for spatial interpolations on large geostatistical data that have participated in the case study competition of Heaton et al. (2018), (2) reproduce the predictive performance scores for simulated MODIS data based on the code provided by Heaton et al. (2018), (3) implement two of the faster methods, NNGP Conjugate and FRK, on NOAA SLP dataset to predict the missing SLP values for land areas, and (4) further evaluate the predictive performances of NNGP Conjugate models using NOAA SLP ocean-location data subset.

As for the reproducibility of the twelve methods, with limited computation resources, some Bayesian related approaches seem to be too slow to generate the full posterior predictive samples by using functions from the corresponding R packages. Due to the constraint in walltime, the predictive results of Metakriging could not be replicated successfully. Most of the reproduced predictive performance scores on the simulated dataset appear to be consistent with the ones listed in Heaton et al. (2018), except for the Predictive Processes as there might exist some minor differences in the prediction procedures chosen by the research group from Heaton et al. (2018) case study competition and this replication study.

NNGP Conjugate and FRK approaches have accessible R packages for implementation and fairly low computational costs. From the case study competition of Heaton et al. (2018), NNGP Conjugate performed consistently well for both simulated and real datasets. And FRK with multiresolutional basis functions is robust to non-stationarity (which is a feature carried by most real-world large spatial data) by nature (Cressie and Johannesson, 2008). Thus for the applications in a real-world context, these methods were selected for interpolating the missing values for land area locations in the SLP dataset. The analysis predicted the mean SLPs month by month throughout the timestamps from the year 1854 to 1997 and the overall prediction results were aggregated into annual means in the end. Regardless of the model specifications, their predicted annual mean land SLPs look similar and some local abrupt shifts among the predicted land SLP values can be captured by the visualizations. NNGP Conjugate models generally have lower pooled prediction standard errors except for the one using month as a covariate. What made 'month' models special was the different structure of dataset from which the models were fitted. One so-called composite dataset was constructed by keeping one set of unique coordinates of the locations and sampling one SLP value from the 1,728 months for each location. A model fit would occur after each composite dataset was constructed. This dataset reconstruction was mainly for relieving the technical issues that occurred when implementing functions from the corresponding R package, for instance, the 'spNNGP' package. For FRK approach, there is still more to explore with respect to the SLP dataset interpolations. Due to the time limit of the study, the FRK models fitted in this data analysis project only employed the

bisquare basis functions. According to 'FRK' R package, other basis function options such as Gaussian, exponential and Matern are available. Furthermore, the nature of the original SLP dataset covers both the spatial and temporal aspects. This study has only focused on the spatial interpolation methods and their applications, thus ignore the correlations among the time points in the SLP dataset. Some of the methods studied by Heaton et al. (2018), such as FRK, Predictive Processes and Gapfill, allow for spatio-temporal modeling as well. Improvements on the prediction precisions may be achieved by conducting spatio-temporal analysis using these developed methods.

One more noteworthy finding from the analyses of NOAA SLP dataset was: although the temporal dependence nature was ignored, the predicted performances about the 1990-1997 annual mean ocean SLPs at hold-out locations appeared to be fairly accurate by using the NNGP Conjugate approach on the 1854-1980 monthly mean ocean SLPs from non-hold-out locations. Two cases of data aggregation concerning whether or not to aggregate the training data monthly means into annual means before any model fits were investigated. Both cases have shown good and similar predictive accuracy according to their MAE and RMSE values, but aggregation before model fits can largely reduce the computing time though at some costs of information loss. Also, the resulting prediction standard errors from the analysis should be treated with caution as ignoring the temporal dependence will lead to greater standard errors. Nevertheless, the analysis of ocean SLP values has again justified the superb geostatistical predictive performance of the NNGP Conjugate approach with respect to prediction accuracy.

References

Arbia, Giuseppe, and Michele Di Marcantonio. 2015. "Forecasting Interest Rates Using Geostatistical Techniques." *Econometrics* 3 (4): 1–28. https://www.mdpi.com/2225-1146/3/4/733/pdf.

Bakka, Haakon, Håvard Rue, Geir-Arne Fuglstad, Andrea I. Riebler, David Bolin, Janine Illian, Elias Krainski, Daniel P. Simpson, and Finn K. Lindgren. 2018. "Spatial Modelling with INLA: A Review." *WIRES (Invited Extended Review)*, no. Feb. http://arxiv.org/abs/1802.06350.

Banerjee, Sudipto, Alan E. Gelfand, Andrew O. Finley, and Huiyan Sang. 2008. "Gaussian Predictive Process Models for Large Spatial Data Sets." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70 (4): 825–48. https://doi.org/10.1111/j.1467-9868.2008.00663.x.

Berliner, L. Mark. 1996. "Hierarchical Bayesian Time Series Models." In *Maximum Entropy and Bayesian Methods*, edited by Kenneth M. Hanson and Richard N. Silver, 15–22. Dordrecht: Springer Netherlands.

Chorley, R. J., and P. Haggett. 1965. "Trend-Surface Mapping in Geographical Research." *Transactions of the Institute of British Geographers*, no. 37. Royal Geographical Society (with the Institute of British Geographers), Wiley: 47–67. http://www.jstor.org/stable/621689.

Corporation, Microsoft, and Steve Weston. 2018. *DoParallel: Foreach Parallel Adaptor for the 'Parallel' Package*. https://CRAN.R-project.org/package=doParallel.

Cressie, N. 1993. *Statistics for Spatial Data. Revised Ed.* New York: John Wiley & Sons.

Cressie, Noel, and Gardar Johannesson. 2008. "Fixed Rank Kriging for Very Large Data Sets." *Journal of the Royal Statistical Society Series B* 70 (February): 209–26. https://doi.org/10.1111/j.1467-9868.2007.00633.x.

Datta, Abhirup, Sudipto Banerjee, Andrew O. Finley, and Alan E. Gelfand. 2016a. "Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets." *Journal of the American Statistical Association* 111 (514). Taylor & Francis: 800–812. https://doi.org/10.1080/01621459.2015.1044091.

———. 2016b. "On Nearest-Neighbor Gaussian Process Models for Massive Spatial Data." *Wiley Interdisciplinary Reviews: Computational Statistics* 8 (5): 162–71. https://doi.org/10.1002/wics.1383.

De Coninck, Arne, Bernard De Baets, Drosos Kourounis, Fabio Verbosio, Olaf Schenk, Steven Maenhout, and Jan Fostier. 2016. "Needles: Toward Large-Scale Genomic Prediction with Marker-by-Environment Interaction." *Genetics* 203 (1): 543–55. https://doi.org/10.1534/genetics.115.179887.

Douglas Nychka, Reinhard Furrer, John Paige, and Stephan Sain. 2017. "Fields: Tools

for Spatial Data." Boulder, CO, USA: University Corporation for Atmospheric Research. https://doi.org/10.5065/D6W957CT.

Finley, Andrew, Sudipto Banerjee, and Øyvind Hjelle. 2017. *MBA: Multilevel B-Spline Approximation*. https://CRAN.R-project.org/package=MBA.

Finley, Andrew, Abhirup Datta, and Sudipto Banerjee. 2017. *SpNNGP: Spatial Regression Models for Large Datasets Using Nearest Neighbor Gaussian Processes*. https://CRAN.R-project.org/package=spNNGP.

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

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

Finley, Andrew O., Huiyan Sang, Sudipto Banerjee, and Alan E. Gelfand. 2009. "Improving the Performance of Predictive Process Modeling for Large Datasets." *Computational Statistics & Data Analysis* 53 8: 2873–84. https://doi.org/10.1016/j.csda.2008.09.008.

Gerber, F., R. de Jong, M. E. Schaepman, G. Schaepman-Strub, and R. Furrer. 2018. "Predicting Missing Values in Spatio-Temporal Remote Sensing Data." *IEEE Transactions on Geoscience and Remote Sensing* 56 (5): 2841–53. https://doi.org/10.1109/TGRS.2017.2785240.

Gerber, Florian, Rogier de Jong, Micheal E. Schaepman, Gabriela Schaepman-Strub, and Reinhard Furrer. 2018. "Predicting Missing Values in Spatio-Temporal Remote Sensing Data." *IEEE Transactions on Geoscience and Remote Sensing*, 1–13. https://doi.org/10.1109/TGRS.2017.2785240.

Gneiting, Tilmann, and Adrian E Raftery. 2007. "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association* 102 (477). Taylor & Francis: 359–78. https://doi.org/10.1198/016214506000001437.

Gramacy, Robert B. 2016. "laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R." *Journal of Statistical Software* 72 (1): 1–46. https://doi.org/10.18637/jss.v072.i01.

Gramacy, Robert B., and Daniel W. Apley. 2015. "Local Gaussian Process Approximation for Large Computer Experiments." *Journal of Computational and Graphical Statistics* 24 (2). Taylor & Francis: 561–78. https://doi.org/10.1080/10618600.2014.914442.

Gräler, Benedikt, Edzer Pebesma, and Gerard Heuvelink. 2016. "Spatio-Temporal Interpolation Using Gstat." *The R Journal* 8 (1): 204–18. https://journal.r-project.org/archive/2016-1/na-pebesma-heuvelink.pdf.

Guhaniyogi, Rajarshi, and Sudipto Banerjee. 2018. "Meta-Kriging: Scalable Bayesian Modeling and Inference for Massive Spatial Datasets." *Technometrics* 60 (4). Taylor &

Francis: 430–44. https://doi.org/10.1080/00401706.2018.1437474.

Guinness, Joseph. 2017. "Spectral Density Estimation for Random Fields via Periodic Embeddings." *arXiv E-Prints*, October. https://arxiv.org/pdf/1710.08978.pdf.

Heaton, Matthew J., Abhirup Datta, Andrew O. Finley, Reinhard Furrer, Joseph Guinness, Rajarshi Guhaniyogi, Florian Gerber, et al. 2018. "A Case Study Competition Among Methods for Analyzing Large Spatial Data." *Journal of Agricultural, Biological and Environmental Statistics*, December. https://doi.org/10.1007/s13253-018-00348-w.

Hefley, Trevor J., Kristin M. Broms, Brian M. Brost, Frances E. Buderman, Shannon L. Kay, Henry R. Scharf, John R. Tipton, Perry J. Williams, and Mevin B. Hooten. 2017. "The Basis Function Approach for Modeling Autocorrelation in Ecological Data." *Ecology* 98 (3): 632–46. https://doi.org/10.1002/ecy.1674.

Katzfuss, Matthias. 2017. "A Multi-Resolution Approximation for Massive Spatial Datasets." *Journal of the American Statistical Association* 112 (517). Taylor & Francis: 201–14. https://doi.org/10.1080/01621459.2015.1123632.

Katzfuss, Matthias, and Noel Cressie. 2011. "Tutorial on Fixed Rank Kriging (Frk) of Co2 Data." https://niasra.uow.edu.au/content/groups/public/@web/@inf/@math/documents/mm/uow175999.pdf.

Katzfuss, Matthias, and Wenlong Gong. 2017. "A class of multi-resolution approximations for large spatial datasets." *arXiv E-Prints*, October. https://arxiv.org/pdf/1710.08976.pdf.

Kaufman, Cari G., Mark J. Schervish, and Douglas W. Nychka. 2008. "Covariance Tapering for Likelihood-Based Estimation in Large Spatial Data Sets." *Journal of the American Statistical Association* 103 (484). Taylor & Francis: 1545–55. https://doi.org/10.1198/016214508000000959.

Kourounis, D., A. Fuchs, and O. Schenk. 2018. "Towards the Next Generation of Multiperiod Optimal Power Flow Solvers." *IEEE Transactions on Power Systems* PP (99): 1–10. https://doi.org/10.1109/TPWRS.2017.2789187.

Krige, D.G. 1951. "A Statistical Approaches to Some Basic Mine Valuation Problems on the Witwatersrand." *Journal of the Chemical, Metallurgical and Mining Society of South Africa* 52: 119–39.

Kuntz, Michael, and Marco Helbich. 2014. "Geostatistical Mapping of Real Estate Prices: An Empirical Comparison of Kriging and Cokriging." *International Journal of Geographical Information Science* 28 (9). Taylor & Francis: 1904–21. https://doi.org/10.1080/13658816.2014.906041.

Lindgren, Finn, and Håvard Rue. 2015. "Bayesian Spatial Modelling with R-INLA." *Journal of Statistical Software* 63 (19): 1–25. http://www.jstatsoft.org/v63/i19/.

Lindgren, Finn, Håvard Rue, and Johan Lindström. 2011. "An Explicit Link Between Gaussian Fields and Gaussian Markov Random Fields: The Stochastic Partial Differential

Equation Approach." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73 (4): 423–98. https://doi.org/10.1111/j.1467-9868.2011.00777.x.

Lindgren, Finn, Håvard Rue, and Johan Lindström. 2011. "An Explicit Link Between Gaussian Fields and Gaussian Markov Random Fields: The Stochastic Partial Differential Equation Approach (with Discussion)." *Journal of the Royal Statistical Society B* 73 (4): 423–98.

Martins, Thiago G., Daniel Simpson, Finn Lindgren, and Håvard Rue. 2013. "Bayesian Computing with Inla: New Features." *Computational Statistics & Data Analysis* 67: 68–83. http://www.sciencedirect.com/science/article/pii/S0167947313001552.

Microsoft, and Steve Weston. 2017. *Foreach: Provides Foreach Looping Construct for R*. https://CRAN.R-project.org/package=foreach.

Minsker, Stansilav, Sanvesh Srivastava, Lizhen Lin, and David B. Dunson. 2014. "Robust and Scalable Bayes via a Median of Subset Posterior Measures." *arXiv*. http://arxiv.org/abs/1403.2660.

Nychka, Douglas, Soutir Bandyopadhyay, Dorit Hammerling, Finn Lindgren, and Stephan Sain. 2015. "A Multiresolution Gaussian Process Model for the Analysis of Large Spatial Datasets." *Journal of Computational and Graphical Statistics* 24 (2). Taylor & Francis: 579–99. https://doi.org/10.1080/10618600.2014.914946.

Nychka, Douglas, Dorit Hammerling, Stephan Sain, and Nathan Lenssen. 2016. "LatticeKrig: Multiresolution Kriging Based on Markov Random Fields." Boulder, CO, USA: University Corporation for Atmospheric Research. https://doi.org/10.5065/D6HD7T1R.

O. Finley, Andrew, Abhirup Datta, Bruce C. Cook, Douglas Morton, Hans E. Andersen, and Sudipto Banerjee. 2017. "Applying Nearest Neighbor Gaussian Processes to Massive Spatial Data Sets: Forest Canopy Height Prediction Across Tanana Valley Alaska," February.

Oliver, M. 2010. *Geostatistical Applications for Precision Agriculture*. https://doi.org/10.1007/978-90-481-9133-8.

Pebesma, Edzer J. 2004. "Multivariable Geostatistics in S: The Gstat Package." *Computers & Geosciences* 30: 683–91.

Pierce, David. 2019. *Ncdf4: Interface to Unidata netCDF (Version 4 or Earlier) Format Data Files*. https://CRAN.R-project.org/package=ncdf4.

R Core Team. 2018. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rue, Håvard, Sara Martino, and Nicholas Chopin. 2009. "Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations (with Discussion)." *Journal of the Royal Statistical Society B* 71: 319–92.

Rue, Håvard, Sara Martino, and Nicolas Chopin. 2009. "Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations."

Journal of the Royal Statistical Society: Series B (Statistical Methodology) 71 (2): 319–92. https://doi.org/10.1111/j.1467-9868.2008.00700.x.

Rue, Håvard, Andrea I. Riebler, Sigrunn H. Sørbye, Janine B. Illian, Daniel P. Simpson, and Finn K. Lindgren. 2017. "Bayesian Computing with INLA: A Review." *Annual Reviews of Statistics and Its Applications* 4 (March): 395–421. http://arxiv.org/abs/1604.00860.

Sang, Huiyan, Mikyoung Jun, and Jianhua Z. Huang. 2011. "Covariance Approximation for Large Multivariate Spatial Data Sets with an Application to Multiple Climate Model Errors." *Ann. Appl. Stat.* 5 (4). The Institute of Mathematical Statistics: 2519–48. https://doi.org/10.1214/11-AOAS478.

Schabenberger, O., and C.A. Gotway. 2005. *Statistical Methods for Spatial Data Analysis*. Boca Raton, FL: Chapman & Hall/CRC. https://www.crcpress.com/Statistical-Methods-for-Spatial-Data-Analysis/Schabenberger-Gotway/p/book/9781584883227.

Slocum, Terry A, Robert M McMaster, Fritz C Kessler, Hugh H Howard, and Robert B Mc Master. 2008. *Thematic Cartography and Geographic Visualization*. 3rd ed. Prentice Hall.

Smith, T.M., and R.W. Reynolds. 2004. "Reconstruction of Monthly Mean Oceanic Sea Level Pressure Based on COADS and Station Data (1854-1997)." J. Oceanic Atmos. Tech., 21, 1272-1282. https://www.esrl.noaa.gov/psd/gcos_wgsp/Gridded/data.noaa.erslp.html.

Stevens, S. S. 1946. "On the Theory of Scales of Measurement." *Science* 103 (2684). American Association for the Advancement of Science: 677–80. https://doi.org/10.1126/science.103.2684.677.

Tobler, W. R. 1970. "A Computer Movie Simulating Urban Growth in the Detroit Region." *Economic Geography* 46. [Clark University, Wiley]: 234–40. http://www.jstor.org/stable/143141.

Verbosio, Fabio, Arne De Coninck, Drosos Kourounis, and Olaf Schenk. 2017. "Enhancing the Scalability of Selected Inversion Factorization Algorithms in Genomic Prediction." *Journal of Computational Science* 22 (Supplement C): 99–108. https://doi.org/10.1016/j.jocs.2017.08.013.

WHITTLE, P. 1954. "ON STATIONARY PROCESSES IN THE PLANE." *Biometrika* 41 (3-4): 434–49. https://doi.org/10.1093/biomet/41.3-4.434.

Wikle, Christopher K., and Mevin B. Hooten. 2010. "A General Science-Based Framework for Dynamical Spatio-Temporal Models." *TEST* 19 (3): 417–51. https://doi.org/10.1007/s11749-010-0209-z.

Wikle, C.K., A. Zammit-Mangion, and N. Cressie. 2019. *Spatio-Temporal Statistics with R*. Boca Raton, FL: Chapman & Hall/CRC. https://spacetimewithr.org/book.

Willmott, C. J., and K. Matsuura. 2006. "On the Use of Dimensioned Measures of Error to Evaluate the Performance of Spatial Interpolators." *International Journal of Geographical Information Science* 20 (1). Taylor & Francis: 89–102. https://doi.org/10.1080/13658810500286976.

Zammit-Mangion, Andrew. 2018. *FRK: Fixed Rank Kriging*. https://CRAN.R-project.org/package=FRK.

Zammit-Mangion, Andrew, and Noel Cressie. 2018. "Fixed Rank Kriging: The R Package." https://cran.r-project.org/web/packages/FRK/vignettes/FRK_intro.pdf.

Appendix A

The code in this section can be download from Heaton et al. (2018) case study competition GitHub repository: https://github.com/finnlindgren/heatoncomparison/tree/master/Code.

Fixed Rank Kriging

```
###################################
## Author: Andrew Zammit-Mangion ##
###################################

## Load packages
library(FRK)
library(sp)
library(ggplot2)
library(gridExtra)
library(INLA)
library(splancs)

## runFRK: Main program function
runFRK <- function(df, f = Temp ~ Lon + Lat, tol = 0.01) {

  t1 <- proc.time()                       # start timer

  ## Make BAUs as SpatialPixels
  BAUs <- df           # assign BAU (basic area unit)s, df = data
  BAUs$Missing <- is.na(BAUs$Temp) # mark missing data
  BAUs$Temp <- NULL                # remove data from BAUs
  BAUs$fs <- 1                     # set fs variation to unity
  coordinates(BAUs)  <- ~Lon+Lat   # to SpatialPointsDataFrame
  gridded(BAUs) <- TRUE            # to SpatialPixelsDataFrame

  ## Make Data as SpatialPoints
  dat <- subset(df,!is.na(Temp))       # no missing data
  coordinates(dat)  <- ~Lon+Lat        # to SpatialPointsDataFrame

  basis <- auto_basis(plane(),         # on the plane
              data = dat,      # data around which to make basis
              regular = 0,     # irregular basis
              nres = 3,        # 3 resolutions
              scale_aperture = 1)  # aperture scaling

  ## Remove basis functions in problematic region
```

```
  if(nrow(df) == 150000) {
    basis_df <- data.frame(basis)
    rmidx <- which(basis_df$loc2 > 36.5 &
                     basis_df$loc1 > -94.5 &
                     basis_df$res == 3)
    basis <- remove_basis(basis,rmidx)
  }

  ## Estimate using Maximum Likelihood (ML)
  S <- FRK(f = f,                          # formula for SRE model
           data = dat,                     # data
           basis = basis,                  # Basis
           BAUs = BAUs,                    # BAUs
           tol = tol)                      # EM iterations


  ## Predict
  BAUs_pred <- SRE.predict(S)             # predict over all BAUs
  BAUs_pred_df <- data.frame(BAUs_pred) # convert to data frame

  ## Compute variance of predicted observations and conf. intervals
  BAUs_pred_df$sd_obs <- sqrt(BAUs_pred_df$sd^2 + S@Ve[1,1])
  BAUs_pred_df$conflo <- BAUs_pred_df$mu - 1.96*BAUs_pred_df$sd_obs
  BAUs_pred_df$confhi <- BAUs_pred_df$mu + 1.96*BAUs_pred_df$sd_obs

  ## Only return missing data and relevant variables
  Res <- subset(BAUs_pred_df,Missing==TRUE)
  Res$Missing <- Res$fs <- Res$var <- NULL

  t2 <- proc.time()                       # stop timer

## Simulated Data experiment
load("SimulatedTemps.RData")
set.seed(25)
FRKsim <- runFRK(sim.data,f <- Temp ~ 1, tol=0.1)
save(file='./FRKSimResults.RData',list="FRKsim")
```

LatticeKrig

```
# LatticeKrig model for big N challenge fits LatticeKrig
# model for different data sets and predicts at missing locations

######## Specify settings  #############
```

```
dataCase<-"simulated"  # "test","simulated" or "SatelliteTemps"
runMode <-"sequential"  # "sequential" and "parallel"(for cluster)
M <- 100  # Number of conditional draws (used for standard errors)
nCores <- 55 # Number of cores (for parallel runMode only)
plotting <- "yes" # Choices are "yes" or "no"

######## Load files and libraries  #####
library( LatticeKrig)
library(spam64)
source("functions.R")

if( runMode=="parallel"){
  library(doParallel)
  library(foreach)
  source("LKrig.sim.conditional.foreach.R")
}

#### Load data and specify model parameters ####
# switch among the three cases
if( dataCase=="test"){
  load("../../Data/SmallTestData.RData")
  lon <- code.test$Lon
  lat <- code.test$Lat
  temps <- code.test$MaskedData
  NC<- 16
  nlevel<- 2
  a.wght<- 4.4
  nu<- .5
}

if( dataCase=="simulated"){
  load("../../Data/SimulatedTemps.RData")
  lon <- sim.data$Lon
  lat <- sim.data$Lat
  temps <- sim.data$Temp
  NC<- 30
  nlevel<-4
  a.wght<-  4.4
  nu<- .1
}

if( dataCase=="SatelliteTemps"){
  load("../../Data/SatelliteTemps.RData")
  lon <- sat.temps$Lon
```

```
  lat <- sat.temps$Lat
  temps <- sat.temps$Temp
  NC<- 40
  nlevel<- 4
  a.wght<-  10.25
  nu<- .1
 }

#### Fit the model ####
# create locations and responses (y)
dataObject<- makeData(lon,lat, temps)

# setup LKrig object
LKinfoFinal<- LKrigSetup( dataObject$x,
                          NC = NC,
                          nlevel = nlevel,
                          a.wght = a.wght,
                              nu = nu)
# determine fit
  comp.time <- system.time(fitFinal<- LatticeKrig( dataObject$x,
                             dataObject$y, LKinfo= LKinfoFinal))
  if( plotting=="yes"){
  print( fitFinal) }# Display of fit

#### Conditional simulation to determine prediction SE ####
# M is the number of independent draws from conditional
  set.seed(234)

  if( runMode=="parallel"){
    comp.time <- comp.time +
      system.time(outputSim<- LKrig.sim.conditional.foreach(
        fitFinal, x.grid = dataObject$xMissing,M = M,
        nCores=nCores))}
  if( runMode=="sequential"){
    comp.time <- comp.time +
      system.time(outputSim<- LKrig.sim.conditional(fitFinal,
                  x.grid = dataObject$xMissing,
                  M = M))}

# NOTE: prediction SE needs to include uncertainty due to nugget
# random component ( estimated as fitFinal$sigma.MLE )

comp.time <- comp.time + system.time(standardError<- sqrt(
      apply( outputSim$g.draw, 1, "var") +
```

```
      fitFinal$sigma.MLE^2))


yHat<- outputSim$ghat
CI95Lower<- yHat - 1.96* standardError
CI95Upper<- yHat + 1.96* standardError

####  Save everything important to an R binary file ####
finalResults<- list(x=dataObject$xMissing,
                    yHat=yHat,
                    standError=standardError)

save(finalResults, fitFinal, comp.time,
      file=paste0(dataCase,"NC",NC,"nlevel",
                  nlevel,"finalResults.rda"))
```

Predictive Processes

The parameters as well as random effect samples were generated using Markov Chain Monte Carlo (MCMC) method by running the c++ code from https://github.com/finnlindgren/heatoncomparison/tree/master/Code/PredProc (Heaton et al., 2018). The code chunk below shows the post-MCMC posterior predictive sampling procedure.

```
rm(list=ls())
library(spBayes)
library(fields)
# read-in the results from .cpp
# knots coordinates
knots = scan('knots')
knot.coords = matrix(knots, nrow = 196, ncol = 2, byrow = TRUE)
# beta samples from predictive process
p.beta.samples = matrix(scan('sim-beta'), nrow = 25000, ncol = 1)
# theta sampels from predictive process
p.theta.samples = matrix(scan('sim-theta'), nrow = 25000, ncol = 3)
# random coefficients from predictive process
p.wStr.samples = matrix(scan('sim-wStr'), nrow = 25000, ncol = 196)

# read-in data
load("SimulatedTemps.RData")
data = sim.data
nas <- which(is.na(data$Temp))
n.samples = 25000

# training data
```

```
coords.tr = as.matrix(data[,1:2][-nas,,drop = FALSE])
X.tr = as.matrix(cbind(1, rnorm(dim(coords.tr)[1])))
y.tr = as.matrix(data[,3][-nas,drop=FALSE])

# test data
coords.ho = as.matrix(data[,1:2][nas,,drop = FALSE])
X.ho = as.matrix(cbind(1, rnorm(dim(coords.ho)[1])))
y.ho = as.matrix(data[,3][nas,drop = FALSE])

# other useful values
knotsPredD = iDist(coords.ho,knot.coords)
knotsD = iDist(knot.coords)

burn_in = 0.5 * n.samples
p.beta.samples = p.beta.samples[-(1:burn_in),]
p.theta.samples = p.theta.samples[-(1:burn_in),]
p.wStr.samples = p.wStr.samples[-(1:burn_in),]

# a matrix for predicted results
y.pred.mat = matrix(0, nrow = n.samples-burn_in,
                    ncol = length(y.ho))

for (i in 1:(n.samples-burn_in)){

  # C' is nxr matrix whose entires are the covariance
  # between w(si) and w(sj*)
  C.knotsPred = p.theta.samples[i,1]*exp(-knotsPredD*
                                           p.theta.samples[i,3])
  # c*^(-1) is rxr covariance matrix of w(si*)
  C.knots = p.theta.samples[i,1]*exp(-knotsD*
                                       p.theta.samples[i,3])

  # Z
  Z = C.knotsPred%*%solve(C.knots)

  # mu
  mu = X.ho[,2] * p.beta.samples[i] +
    Z%*%as.matrix(p.wStr.samples[i,])
  # D = tau.sq
  D = p.theta.samples[i,2]
  sr.D = sqrt(D)

  y.pred.mat[i,] = rnorm(dim(X.ho)[1], mu, sr.D)
}
```

```r
# aggregate the prediction samples
y.pred = apply(y.pred.mat, 2, mean) # mean
y.pred.sd = apply(y.pred.mat, 2, sd) # standard deviation

# save the outputs in a list
pp_results = list(y.pred,y.pred.sd)
save(pp_results, file = 'pp_results.Rdata')
```

Spatial Partitioning

```r
##########################################################
## Author: Matt Heaton
## Date: 10 May 2017
## Description: Code to fit partitioned spatial model
## Instructions: Please set folder to current directory
## ########################################################


################################################
## Load Libraries and Source Clust Functions ##
################################################
rm(list=ls())
library(LatticeKrig)
library(plyr)
library(parallel)
n.cores <- as.integer(Sys.getenv('PBS_NUM_PPN'))-1

######################
## Read in the data ##
######################
data.set <- "sim"
if(data.set=="sat"){
  read.path <- '../../Data/SatelliteTemps.RData'
  write.path <- './SatelliteResults.RData'
  load(read.path)
  spatdat <- sat.temps
  nrows <- 500
  rm(list="sat.temps")
} else if(data.set=="sim"){
  read.path <- 'SimulatedTemps.RData'
  write.path <- './SimulationResults.RData'
  load(read.path)
  spatdat <- sim.data
```

```
  nrows <- 500
  rm(list="sim.data")
} else if(data.set=="test"){
  read.path <- '../../Data/SmallTestData.RData'
  write.path <- './SmallTestResults.RData'
  load(read.path)
  spatdat <- code.test
  names(spatdat) <- c("Lon","Lat","Temp","TrueObs")
  nrows <- 100
} else {
  stop('data.set must be either sat, sim or test')
}

################################################
## Define a few variables for ease up front ##
################################################
lon <- matrix(spatdat$Lon,nrow=nrows)
lat <- matrix(spatdat$Lat,nrow=nrows)
u.lon <- unique(spatdat$Lon)
u.lat <- unique(spatdat$Lat)
minDist <- min(abs(u.lon[1]-u.lon[2]),abs(u.lat[1]-u.lat[2]))
maxDist <- sqrt((u.lon[1]-u.lon[length(u.lon)])^2+(u.lat[1]-
                                        u.lat[length(u.lat)])^2)
obs <- which(!is.na(spatdat$Temp))
N <- length(obs)

###################################
## Define Global Basis Functions ##
###################################
n.knots <- 4 ## n.knots^2 actually
xlim <- range(spatdat$Lon)
ylim <- range(spatdat$Lat)
xlocs <- seq(xlim[1],xlim[2],length=n.knots+1)[-1]
xlocs <- xlocs-0.5*mean(diff(xlocs))
ylocs <- seq(ylim[1],ylim[2],length=n.knots+1)[-1]
ylocs <- ylocs-0.5*mean(diff(ylocs))
knot.locs <- expand.grid(xlocs,ylocs)
B <- rdist(cbind(spatdat$Lon,spatdat$Lat),knot.locs)
rl <- 2*min(rdist(knot.locs)[lower.tri(rdist(knot.locs))])
B <- ((1-(B/rl)^2)^2)*(B<=rl)
# plot(spatdat$Lon,spatdat$Lat,pch=19,cex=.2)
# points(knot.locs[,1],knot.locs[,2],pch=19,col="red")
# image.plot(lon,lat,matrix(B[,2],nrow=nrows))
```

```
##############
## X-matrix ##
##############
if(data.set=="sat"){
  X <- cbind(1,spatdat$Lon,spatdat$Lat,B)
} else if(data.set=="sim") {
  X <- cbind(1,B)
} else {
  X <- cbind(1,B)
}
P <- ncol(X)

########################################
## Cluster Residuals via Agg clustering ##
########################################
n.grps <- 5
n.clust <- n.grps^2
xlocs <- seq(xlim[1],xlim[2],length=n.grps+1)[-1]
xlocs <- xlocs-0.5*mean(diff(xlocs))
ylocs <- seq(ylim[1],ylim[2],length=n.grps+1)[-1]
ylocs <- ylocs-0.5*mean(diff(ylocs))
part.centroids <- expand.grid(xlocs,ylocs)
D <- rdist(cbind(spatdat$Lon,spatdat$Lat),part.centroids)
clust <- apply(D,1,which.min)
spatdat$cluster <- clust
# plot(spatdat$Lon,spatdat$Lat,pch=19,cex=.2)
# points(part.centroids[,1],part.centroids[,2],pch=19,col="red")
# image.plot(lon,lat,matrix(spatdat$cluster,nrow=nrows))
range(table(spatdat$cluster[obs]))
summary(lm(Temp~X+as.factor(cluster),data=spatdat))$r.sq

#########################################################
## Grid search for omega (nugget) and alpha (decay) ##
#########################################################
n.om <- 20
n.alpha <- 10
om.grid <- seq(0,0.99,length=n.om)
alpha.grid <- 3/seq(minDist,maxDist,length=n.alpha)
oa.grid <- expand.grid(om.grid,alpha.grid)

##############################################
## Function to Evaluate Likelihood pieces ##
##############################################
setup.MLE <- function(xlist){
```

```
  ## Set up variables
  nc <- nrow(yc)
  D <- rdist(sc)
  Rc <- xlist$omega*exp(-xlist$alpha*D)+(1-xlist$omega)*diag(nc)
  if(xlist$omega==0){ #if omega=0 then no spatial correlation, Rc=I
    Rc.chol <- Rc
    Rc.inv <- Rc
  } else {
    Rc.chol <- chol(Rc)
    Rc.inv <- chol2inv(Rc.chol)
  }

  ## Calculate critical quantities
  log.det <- 2*sum(diag(Rc.chol))
  Xp.Rcinv.y <- t(Xc)%*%Rc.inv%*%yc
  Xp.Rcinv.X <- t(Xc)%*%Rc.inv%*%Xc
  yp.Rcinv.y <- t(yc)%*%Rc.inv%*%yc

  ## Add to Sum
  xlist$Xp.Rcinv.y.sum <- xlist$Xp.Rcinv.y.sum+Xp.Rcinv.y
  xlist$Xp.Rcinv.X.sum <- xlist$Xp.Rcinv.X.sum+Xp.Rcinv.X
  xlist$yp.Rcinv.y.sum <- xlist$yp.Rcinv.y.sum+yp.Rcinv.y
  xlist$logdet.sum <- xlist$logdet.sum + log.det

  ## Return new xlist
  return(xlist)

}

##############################################
## Function to Evaluate MLEs and log-like ##
##############################################
get.MLEs <- function(xlist){

  ## Calculate beta.hat
  beta.hat <- chol2inv(chol(xlist$Xp.Rcinv.X.sum))%*%
                                    xlist$Xp.Rcinv.y.sum

  ## Calculate sigma2.hat
  ss <- (xlist$yp.Rcinv.y.sum-2*t(beta.hat)%*%xlist$Xp.Rcinv.y.sum+
           t(beta.hat)%*%xlist$Xp.Rcinv.X.sum%*%beta.hat)
  sig2.hat <- ss/(N-P)
```

```
  ## Calculate log-like
  loglike <- -(N/2)*log(sig2.hat)-0.5*xlist$logdet.sum -
                                      0.5*(ss/sig2.hat)

  ## Return info
  return(list(beta.hat=beta.hat,sig2.hat=sig2.hat,
              omega.hat=xlist$omega,alpha.hat=xlist$alpha,
              loglike=loglike))

}

####################################
## Create lists w/unique om/alpha ##
####################################
oa.list <- vector("list",length=nrow(oa.grid))
for(i in 1:length(oa.list)){
  oa.list[[i]]$omega <- oa.grid[i,1]
  oa.list[[i]]$alpha <- oa.grid[i,2]
  oa.list[[i]]$Xp.Rcinv.y.sum <- matrix(0,P,1)
  oa.list[[i]]$Xp.Rcinv.X.sum <- matrix(0,P,P)
  oa.list[[i]]$yp.Rcinv.y.sum <- 0
  oa.list[[i]]$logdet.sum <- 0
}

##########################################
## Find the MLEs via parallel processing ##
##########################################

## Calculate critical quantities in each cluster
#chk.list <- oa.list[[34]]
comp.time <- system.time({
for(i in 1:n.clust){
  clust.obs <- which(spatdat$cluster==i & !is.na(spatdat$Temp))
  yc <- matrix(spatdat$Temp[clust.obs],ncol=1)
  Xc <- matrix(X[clust.obs,],ncol=P)
  sc <- cbind(spatdat$Lon[clust.obs],spatdat$Lat[clust.obs])
  oa.list <- mclapply(oa.list,setup.MLE,mc.cores=n.cores)
  #chk.list <- setup.MLE(chk.list)
  print(paste0(100*round(i/n.clust,2)," Percent Complete"))
}
})
tot.time <- comp.time[3]
comp.time <- comp.time[3]/(nrow(oa.grid))
#chk.MLEs <- get.MLEs(chk.list)
```

```
#chk.lm <- summary(lm(Temp~X-1,data=spatdat))

## Find MLEs and log.like
all.mles <- lapply(oa.list,get.MLEs)

## Select the model that is the MLE
the.mle <- which.max(sapply(all.mles,function(xlist)
                            {return(xlist$loglike)}))
fitted.model <- all.mles[[the.mle]]

#######################################################
## Function to calculate predictions in parallel ##
#######################################################
get.preds <- function(xlist){

  ## Setup Variables
  sc.obs <- which(!is.na(xlist$yc))
  D <- rdist(xlist$sc)
  R <- fitted.model$omega.hat*exp(-fitted.model$alpha.hat*D)+
    (1-fitted.model$omega.hat)*diag(nrow(D))
  mn <- xlist$Xc%*%fitted.model$beta.hat

  ## Calculate Predictive Distribution
  R12.R22inv <- R[-sc.obs,sc.obs]%*%chol2inv(chol(R[sc.obs,sc.obs]))
  pred.mn <- mn[-sc.obs,]+R12.R22inv%*%(xlist$yc[sc.obs,]
                                        -mn[sc.obs,])
  pred.var <- diag(R[-sc.obs,-sc.obs]-R12.R22inv%*%R[sc.obs,
                                                     -sc.obs])
  pred.var <- fitted.model$sig2.hat*pred.var

  ## Return predmn and variances
  all.pred.mns <- xlist$yc
  all.pred.mns[-sc.obs] <- pred.mn
  all.pred.vars <- matrix(0,nrow=nrow(xlist$yc),ncol=1)
  all.pred.vars[-sc.obs] <- pred.var
  return(data.frame(pred.mn=all.pred.mns,
                    pred.sd=sqrt(all.pred.vars)))

}

##############################
## Predictions in Parallel ##
##############################
clustdata <- vector("list",length=n.clust)
```

```
for(i in 1:n.clust){
  clust.obs <- which(spatdat$cluster==i)
  clustdata[[i]]$yc <- matrix(spatdat$Temp[clust.obs],ncol=1)
  clustdata[[i]]$Xc <- X[clust.obs,]
  clustdata[[i]]$sc <- cbind(spatdat$Lon[clust.obs],
                             spatdat$Lat[clust.obs])
}
pred.time <- system.time({
  preds <- mclapply(clustdata,get.preds,mc.cores=n.cores)
})
tot.time <- tot.time+pred.time[3]

#######################################
## Divide out Predictions to Dataset ##
#######################################
spatdat$pred <- spatdat$Temp
spatdat$pred.se <- 0
for(i in 1:n.clust){
  spatdat$pred[spatdat$cluster==i] <- preds[[i]]$pred.mn
  spatdat$pred.se[spatdat$cluster==i] <- preds[[i]]$pred.sd
}
spatdat$lowlim <- spatdat$pred-1.96*spatdat$pred.se
spatdat$uplim <- spatdat$pred+1.96*spatdat$pred.se
# image.plot(lon,lat,matrix(spatdat$pred,nrow=nrows))
# image.plot(lon,lat,matrix(spatdat$Temp,nrow=nrows))

###############################
## Check Predictive Accuracy ##
###############################
if(data.set=="sat"){
  load('../../Data/AllSatelliteTemps.RData')
  tst.set <- which(is.na(all.sat.temps$MaskTemp) & !is.na(
                                  all.sat.temps$TrueTemp))
  tst.obs <- all.sat.temps$TrueTemp[tst.set]
} else if(data.set=='sim'){
  load('AllSimulatedTemps.RData')
  tst.set <- which(is.na(all.sim.data$MaskTemp))
  tst.obs <- all.sim.data$TrueTemp[tst.set]
} else {
  tst.set <- which(is.na(spatdat$Temp))
  tst.obs <- spatdat$TrueObs[tst.set]
}
ll <- spatdat$lowlim[tst.set]
pred <- spatdat$pred[tst.set]
```

```
ul <- spatdat$uplim[tst.set]
print(mean(ll<=tst.obs & ul>=tst.obs)) #CVG
print(mean(pred-tst.obs)) #Bias
print(sqrt(mean((pred-tst.obs)^2))) #RMSE

############################
## Save your predictions ##
############################
if(data.set=="sat"){
  sat.results <- spatdat
  save(file=write.path,list=c("sat.results","tot.time",
                              "comp.time","pred.time"))
} else if(data.set=="sim"){
  sim.results <- spatdat
  save(file=write.path,list=c("sim.results","tot.time",
                              "comp.time","pred.time"))
} else {
  test.results <- spatdat
  save(file=write.path,list=c("test.results","tot.time",
                              "comp.time","pred.time"))
}
```

Covariance Tapering

```
plotImage <- function(x, all=TRUE, xlab='Lon', ylab='Lat', ...) {
    # Plotting the data
    if(!is.matrix(x))
        if (length( x)==n) {
            datamat[] <- x
        } else if (length( x)==nisNA) {
            datamat[isNA] <- x
        } else  if (length( x)==nnotNA) {
            datamat[notNA] <- x
        }  else   image.plot( x, xlab='Lon', ylab='Lat', ...)
    if (!all)     datamat[notNA] <- NA
    gapfill::Image(t(datamat), xlab=xlab, ylab=ylab, ...)
}

vgMatrix <- function( x, xlag=1:50, ylag=1:50){
    # Rudimentary estimation of the variogram. Result is a matrix.
    dimx <- dim(x)[1]
    dimy <- dim(x)[2]
    nx <- length(xlag)
```

```
    ny <- length(ylag)
    vg <- matrix(0, nx, ny)
    for( i in 1:nx) {
      for( j in 1:ny) {
        xind <- 1:(dimx-xlag[i])
        yind <- 1:(dimy-ylag[j])
        # Here is the actual estimation (Materons's estimator),
        # very rudimentary...
        vg[i,j] <- mean( (x[ xind, yind] - x[ xind+xlag[i],
                                  yind+ylag[j]])^2, na.rm=T)
      }
    }
    return(vg)
}

extractDirVg <- function(vgmat, dlon, dlat) {
    # From a 'varigram' matrix, we extract individual
    # directional variograms.
    dvg <- 1:(2*floor( min( dim(vgmat)-1)/2))
    dvg2 <- (1:floor( min( dim(vgmat)-1)/2))
    h <- emp <- list()
    h[[1]] <- dlon[dvg]
    emp[[1]] <- vgmat[dvg,1]
    h[[2]] <- sqrt( dlon[2*dvg2+1]^2+dlat[dvg2+1]^2)
    emp[[2]] <- vgmat[cbind(2*dvg2+1,dvg2+1)]
    h[[3]] <- sqrt( dlon[dvg]^2+dlat[dvg]^2)
    emp[[3]] <- vgmat[cbind(dvg,dvg)]
    h[[4]] <- sqrt( dlon[dvg2+1]^2+dlat[2*dvg2+1]^2)
    emp[[4]] <- vgmat[cbind(dvg2+1,2*dvg2+1)]
    h[[5]] <- dlat[dvg]
    emp[[5]] <- vgmat[1,dvg]
    return( list( lag=h, emp=emp))
}


vg.exp <- function( h, theta)
    # Exponential variogram, analogue to cov.exp, simplified here...
    theta[3] + theta[2] * (1- exp(-h/theta[1]))


fit <- function(theta) {
    # Calculate (unweighted) sums of squares for 'optim'.
    # theta=(range, sill, nugget, anisotropy angle, anisotropy ratio)
    ss <- sum( (emp[[1]]-vg.exp(lag[[1]], theta[1:3]))^2)
```

```
    for( i in 2:length(emp))
        ss <- ss + sum( (emp[[i]]-vg.exp(lag[[i]], theta[1:3]) )^2)
    return(ss)
    }


args <- commandArgs(TRUE)

if (length(args)==0) {

    rm(list = ls())
    delta <- 0.2     # taper range
    verbose <- list(print=TRUE, plot=FALSE)

} else {

    #delta <- as.numeric( args[ length(args)] )
    verbose <- list(print=TRUE, plot=FALSE)
    for(i in 1:length(args)){
      eval(parse(text=args[[i]]))
    }

}
cat( '\n\n\nTaper range: ',delta,'\n\n')


timing <- list(`Setting up structure and load data` = system.time({

    source('tapering_functions.R')

    # load CRAN packages and possibly other elements
    library("spam")
    library('spam64')
    library("fields")
    library("gapfill")      # for visualization:

    set.seed( 14)
    if (delta==0.2) # taper range theta=0.2 implies the following...
                    # hard coded here
        taperpar <- c(63210027,22072781,19352752,379374314,897958295)
                    # taper range theta=0.2
    if( delta==0.25)
        taperpar <- c(96499273,36032278,28577473,493298457,1209425271)
                    # taper range theta=0.25
```

```
    if( delta==0.3)
        taperpar <- c(136438362,53926803,39139221,646039160,1658695619)
                    # taper range theta=0.3

    if(verbose$print) {
        cat( '\n\n\nTaper range: ',delta,'\n\n')
        print( args <- commandArgs())
        print(taperpar)
        cat( '\n\n\n')
    }

    load("SimulatedTemps.RData")  # Load data and prepare variables.

    Lat <- unique(sim.data$Lat)
    Lon <- unique(sim.data$Lon)
    nx <- length( Lon)
    ny <- length( Lat)
    n <- nx*ny

    # Arrange in matrix form for better display
    datamat <- array(sim.data$Temp, c(nx, ny))  # dim(datamat):500 300
    notNA <- !is.na(sim.data[,3])
    isNA <- !notNA
    nnotNA <- sum(notNA)
    nisNA <- n-nnotNA

    xobs <- sim.data[ notNA, 1:2]
    xpred <- sim.data[ isNA, 1:2]
    Y <- sim.data[ notNA, 3]

})[1]  )

timing$`Estimation of trend`= system.time({

  # Simple model for the mean. Easy to extend with `poly` and
  # `lm` construction:
    X <- as.data.frame( poly( scale( sim.data[,1:2]),
                              degree=2, raw=TRUE) )
    drift <- lm( Y ~ `1.0`+`0.1`, data=cbind(Y, X[ notNA,]))
    mupred <- predict( drift, X)

    if(verbose$print)
        summary( drift)
```

```
    if (verbose$plot) {
        plotImage( datamat)
        quilt.plot(sim.data[ notNA,-4], nx=nx, ny=ny)
        plotImage( mupred)
    }

})[1]

#############################################################
# estimation
timing$`Estimation of covariance structure`= system.time({

    residmat <- datamat
    residmat[notNA] <- drift$resid

    vg <- vgMatrix(residmat,xlag=1:160,ylag=1:161)

    if(verbose$plot)  image.plot(vg)
#

    dlat <- as.vector( nearest.dist( cbind(Lon[1], Lat),
             cbind(Lon[1], Lat[1]), miles=FALSE, delta=10))[-1]
# length 300
    dlon <- as.vector( nearest.dist( cbind(Lon, Lat[1]),
             cbind(Lon[1], Lat[1]), miles=FALSE, delta=10))[-1]
# length 500

# We assume isotropy:
    empirical <- extractDirVg( vg, dlon, dlat)
    emp <- empirical$emp
    lag <- empirical$lag

    res <- optim( c(.4,8,.1), fit, method = "L-BFGS-B",
                 lower=c(.05, 7, 0, 0, 1 ),
                 upper=c(.5,  14, 2, pi,2), hessian=TRUE)

   if(verbose$print) {
        cat('\n\n\n')
        print(res)
        cat('\n\n\n')
    }

    if (verbose$plot) {
        plot( lag[[1]], emp[[1]],
```

```
             ylim=c(0,max(unlist(emp))), type='l')
        for (i in 2:5)   lines( lag[[i]], emp[[i]], col=i)

        h <- seq(0,to=1.7,l=100)
        lines( h, vg.exp( h, res$par[1:3]), col=2, lwd=2)
    }

} )[1]

#####################################################################

timing$`Construction of Sigma` <- system.time({
    spam.options(nearestdistnnz=c(taperpar[1], 400))
    hobs <- nearest.dist( xobs, miles=FALSE, delta=delta)

    if( verbose$print) {
        cat('\nDistance matrix (upper only):')
        shobs <- summary(hobs)
        print( c( log(shobs$nnz, 2)))
    }

    hobs <- hobs + t(hobs)
    Cobsobs <- cov.exp( hobs, theta=c( res$par[1:3])) *
      cov.wend1( hobs, theta=c( delta,1,0))

})[1]

timing$`Cholesky decomposion` <- system.time({

    cholCobsobs <- chol( Cobsobs, memory=list(nnzR=taperpar[4]))
    if( verbose$print) {
        cat('\nCholesky factor:')
        scholCobs <- summary(cholCobsobs)
        print( c( log(scholCobs$nnzR,2)))
    }

})[1]


timing$`Prediction` <- system.time({
    spam.options(nearestdistnnz=c(taperpar[2], 400))
    hpredobs <- nearest.dist( xpred, xobs, method='eucli',
                              miles=FALSE, delta=delta)
    if( verbose$print) {
```

```
        cat('\nDistance matrix (pred-obs):')
        shpredobs <- summary(hpredobs)
    }

    Cpredobs <-  cov.exp( hpredobs, theta=c( res$par[1:3])) *
      cov.wend1( hpredobs, theta=c( delta,1,0))
    Ypred <- c( Cpredobs %*% solve.spam( cholCobsobs, drift$resid) +
                  mupred[ isNA])

})[1]

timing$`Uncertainty calculation` <- system.time({

    M <- 250  # some large number

    spam.options(nearestdistnnz=c(taperpar[3], 400))
    hpred <- nearest.dist( xpred, method='eucli',
                           miles=FALSE, delta=delta)
    if( verbose$print) {
        cat('\nDistance matrix (pred; upper only):')
        shpred <- summary(hpred)
    }
    hpred <- hpred + t(hpred)

    Cpredpred <- cov.exp( hpred, theta=c( res$par[1:3])) *
      cov.wend1( hpred, theta=c( delta,1,0))

    Call <- rbind( cbind(Cobsobs, t(Cpredobs)),
                   cbind( Cpredobs, Cpredpred))

    cholCall <- chol( Call, memory=list(nnzR=taperpar[5]))

    if( verbose$print){
        cat('\nCholesky factor:')
        scholCall <- summary( cholCall)
        print( c( log(scholCall$nnzR,2)))
    }

    samples <- t( rmvnorm.spam( M, Sigma=Call, Rstruct=cholCall))

    Yconditional <- samples[(nnotNA+1):n,] - Cpredobs %*%
      solve.spam( cholCobsobs, samples[1:nnotNA,])
    # + mupred[ isNA]   # last is not necessary here.
```

```
    preduncertainty <- apply( Yconditional, 1, sd)

    predCIlower <- Ypred + apply( Yconditional, 1,
                                  quantile, probs=0.025)
    predCIupper <- Ypred + apply( Yconditional, 1,
                                  quantile, probs=0.975)


})[1]

if (verbose$print){
    cat( '\n\n\n')
    print(timing)
    cat( '\n\n\nTotal time: ', sum( unlist(timing)),'s, ',
        sum( unlist(timing))/3600,'h.\n\n\n')
}

if (verbose$plot) {
    plotImage( Ypred, all=F)

    # quilt.plot( rbind(xobs,xpred),apply( samples,1,mean))
    # quilt.plot( rbind(xobs,xpred),apply( samples,1,sd))
    quilt.plot( xpred, preduncertainty)

    plotImage(apply( Yconditional, 1, mean) )
    plotImage(predCIlower, all=FALSE )
    plotImage(predCIupper, all=FALSE )
    plotImage(predCIupper-predCIlower, all=FALSE )
    plotImage((predCIupper-predCIlower)/preduncertainty, all=FALSE )

}

## save results to file
save(Ypred, preduncertainty, res, timing, delta, verbose,
     Yconditional, predCIlower, predCIlower,
     shobs, shpred, shpredobs, scholCobs, scholCall,
     file=paste0("./SimulatedTemps_tapering_",delta,".RData"))

sessionInfo()
```

MRA

The following code should be running with Matlab. The 'subroutines' sources files could be found in https://github.com/finnlindgren/heatoncomparison/tree/master/

Code/MRA/subroutines.

```
%% MRA main script for Heaton project
% Executes the MRA for the simulated and satellite data. Calculation
% options include prediction, parameter optimization and likelihood
% only evaluation.

%% Preliminary settings
clear all; addpath('subroutines');

%% Options to specify [Options to change here.]
% Choose dataType 'satellite' or 'simulated'
dataType='simulated';
% Choose calculationType 'prediction', 'optimize', 'likelihood'
calculationType='optimize';
plotting=1; % Only use '1' for testing.
%% Setup options [Do NOT change.]
M=9; % Total number of levels
J=2; % Number of partitions by level
r=64; % number of knots per partition
% offset percentage from partition boundaries
offsetPercentage = 0.01;

%% Load data
[data, regressionModel, domainBoundaries, predVec, theta,
  varEps] = load_data(dataType);

%% Build hierarchical grid structure
[knots, ~,nRegions, outputdata, predlocs] = build_structure( M,J,r,
      domainBoundaries, offsetPercentage,data(:,1:3),predVec );

%% Potential optimization
switch calculationType
    case {'optimize', 'prediction'}
        switch calculationType
            case 'optimize'
              fun=@(thetaOpt)MRA( [thetaOpt(1) thetaOpt(2)],
                  outputdata, knots, M, J, nRegions,thetaOpt(3));
              % Dummy values required by optimization routine
              A = [];b = [];Aeq = [];    beq = [];
              % Limits and initial values for parameter search
              lb = [0,0,0]; ub = [10,1,5]; x0 = [5,0.3,0.1];

              tic; x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub);
              elapsedTimeOptimization=toc;
```

```
            % Assign values from optimization to theta and varEps
            theta=[x(1) x(2)];
            varEps=x(3);
        end
        % Prediction
        tic
        [sum_loglik,preds] = MRA( theta, outputdata, knots, M,
                                  J, nRegions,varEps, predlocs);
        elapsedTimePrediction=toc;
        % Reformat data for plotting
        pred = cell2mat(preds);predloc=cell2mat(predlocs);
        predVariance=pred(:,2);
        % Add the prediction from the regression
        predRegression=predict(regressionModel,predloc);
        predMean=pred(:,1)+predRegression;
        save(['ResultsMra',dataType],'predloc','predMean',
             'predVariance');
    case 'likelihood'
        % Prediction
        tic
        [sum_loglik] = MRA( theta, outputdata, knots, M, J,
                            nRegions,varEps);
        elapsedTimeLikelihood=toc;
    otherwise
        warning('Unexpected calculationType. Code is not executed.')
end

%% plots
if plotting==1
figure
scatter(data(:,1),data(:,2),5,data(:,4));
colormap(parula)
colorbar
[cmin,cmax] = caxis;
caxis([cmin,cmax])
title('Ground truth')
xlabel('x locations')
ylabel('y locations')
figure
scatter(predloc(:,1),predloc(:,2),5,predMean);
colormap(parula)
colorbar
caxis([cmin,cmax])
```

```
title('Predicted values')
xlabel('x locations')
ylabel('y locations')
figure
scatter(predloc(:,1),predloc(:,2),5,predVariance);
colormap(flip(autumn))
colorbar
title('Prediction variance')
xlabel('x locations')
ylabel('y locations')
end

%% Notes:

% calculationType specifies what is calculated.
% Option 'prediction' uses a default values for
% the parameters and just conducts the prediction.
% Option 'optimize' optimizes over the range, variance
% and measurement error  and then predicts using the
% values obtained from the optimization.
% Option 'likelihood' only calculates the likelihood.
```

NNGP-Conjugate

```
rm(list=ls())
library(spNNGP)
library(MBA)
library(fields)

# simulated temperatures dataset read-in
load("SimulatedTemps.RData")
data = sim.data
nas <- which(is.na(data$Temp)) # find NAs

# training data coordinates
coords = as.matrix(data[,1:2][-nas,,drop = FALSE])
# training data lat and lon
X = as.matrix(cbind(1, rnorm(dim(coords)[1])))
# training data response (temp)
y = as.matrix(data[,3][-nas,drop=FALSE])

# test data coordinates
coords.ho = as.matrix(data[,1:2][nas,,drop = FALSE])
```

```
# test data lat and lon
X.ho = as.matrix(cbind(1, rnorm(dim(coords.ho)[1])))
# test data response
y.ho = as.matrix(data[,3][nas,drop = FALSE])

# useful arguments
cov.model <- "exponential"
sigma.sq <- 8.5
sigma.sq.IG <- c(2, sigma.sq)
g <- 5

theta.alpha <- as.matrix(expand.grid(seq(0.5, 5, length.out=g),
               seq(0.01/sigma.sq, 0.1/sigma.sq, length.out=g)))
colnames(theta.alpha) <- c("phi", "alpha")

# NNGP-conjugate Bayesian regression model fit
m.c <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 15,
                  k.fold = 5, score.rule = "crps",
                  n.omp.threads = 3,
                  theta.alpha = theta.alpha,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)
m.c$run.time[3]/60

m.c$sigma.sq.hat
tau.sq <- m.c$theta.alpha[2]*m.c$sigma.sq.hat
tau.sq

crps.surf <- mba.surf(m.c$k.fold.scores[,c("phi","alpha","crps")],
             no.X=100, no.Y=100)$xyz.est
image.plot(crps.surf, xlab="phi", ylab="alpha=tau^2/sigma^2",
           main="CRPS (lower is better)")
points(m.c$theta.alpha, col="white", pch=2)

# prediction
theta.alpha <- as.vector(m.c$theta.alpha)
names(theta.alpha) <- c("phi", "alpha")

m.p <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 15,
                  X.0=X.ho, coords.0=coords.ho,
                  n.omp.threads = 10,
                  theta.alpha = theta.alpha,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)
```

```
m.p$run.time[3]/60

# total runtime for x-validation and prediction
total.time <- m.c$run.time[3]/60+m.p$run.time[3]/60
total.time

# estimates
round(m.p$sigma.sq.hat,2)
m.p$theta.alpha
tau.sq <- m.p$theta.alpha[2]*m.p$sigma.sq.hat
tau.sq
round(m.p$beta.ha,2)

# confidence interval:
# (y(s)|Y - hat(y(s)) / sqrt((a-1)/a*Varhat(y(s))
# ~ t-distributions with 2a degrees of freedom
# So, the alpha^th quantile for y(s)|Y will be
# given by hat(y(s)) + sqrt((a-1)/a*Varhat(y(s)))
# * t_{2a,alpha}. Then 95% CI will be
# (2.5 qntl, 97.5 qntl)
a <- m.p$ab[2]
t <- qt(0.975, 2*a)
# margin of error
me <- sqrt((a-1)/a*m.p$y.0.hat.var)*t

# predicted temperatures and 95%CI
pred <- cbind(m.p$y.0.hat, m.p$y.0.hat-me, m.p$y.0.hat+me)
colnames(pred) <- c("50%","2.5%","97.5%")
# standard deviation
nngp_conj_stdev = sqrt(m.p$y.0.hat.var)
# save the output to .RData
save(file='./NNGP_CONJ_SimResults.RData',
     list=c("pred",'nngp_conj_stdev'))
```

NNGP-Response

```
# try spNNGP package for post process prediction
library(spNNGP)
library(FNN)
library(fields)

# read-in data
load("SimulatedTemps.RData")
```

```
data = sim.data
nas <- which(is.na(data$Temp))

# training data
coords.tr = as.matrix(data[,1:2][-nas,,drop = FALSE])
X.tr = as.matrix(cbind(1, rnorm(dim(coords.tr)[1])))
y.tr = as.matrix(data[,3][-nas,drop=FALSE])

# test data
coords.ho = as.matrix(data[,1:2][nas,,drop = FALSE])
X.ho = as.matrix(cbind(1, rnorm(dim(coords.ho)[1])))
y.ho = as.matrix(data[,3][nas,drop = FALSE])

# useful arguments
cov.model <- "exponential"
n.samples <- 10000

starting <- list("phi"=1.21, "sigma.sq"=10, "tau.sq"=0.05)
tuning <- list("phi"=0.05, "sigma.sq"=0.01, "tau.sq"=0.01)
priors <- list("phi.Unif"=c(0.6, 30), "sigma.sq.IG"=c(2, 10),
               "tau.sq.IG"=c(2, 0.01))

# fit NNGP Bayesian spatial regression model
m.r <- spNNGP(y.tr~X.tr-1, coords=coords.tr, starting=starting,
              method="response", n.neighbors=15, tuning=tuning,
              priors=priors, cov.model=cov.model,
              n.samples=n.samples, n.omp.threads=12)
# make prediction
p.r <- spPredict(m.r, X.0 = X.ho, coords.0 = coords.ho,
                 n.omp.threads=12)

# aggregate the posterior prediction samples
y_pred_nngpresp = apply(p.r$p.y.0, 1, mean) # mean
y_sd_nngpresp = apply(p.r$p.y.0, 1, sd) # standard deviation

# save the final outputs
save(file = 'nngp_postproc_result.RData',
     list = c('y_pred_nngpresp','y_sd_nngpresp'))
```

Stochastic PDEs

```
## Modified from
```

```
## import library
library(INLA)

## Input:
data.path <- "/home/wang6718/Documents/SPDE" ## Path to data files
dataset <- 3
## 1: SatelliteTemps.RData
## 2: SimulatedTemps.RData
## 3: SmallTestData.RData

## Model settings:
spde.alpha <- 1.5
## 1.5 gives an approximation of an exponential covariance

## inla() options:
num.threads = 2
## Limit memory usage by limiting the number of openmp threads.
openmp.strategy = "large"
## "large" turns off nested parallelism

## Output:
##
## Model 1, "Trend": intercept + longitude + latitude
## Model 2, "TrendField": intercept + longitude + latitude +
##                        gmrf/spde
##
## Posterior reconstructions for all lattice points, in the
## same order as the input data:
##   reconstructionsX: data.frame(Lon, Lat, Temp, SD)
##   predX: data.frame(mean, sd), where mean=Temp and sd=SD:
##          special objects used when calling the scoring
##          methods from assessment.R
##   tempsX: list(mean, sd), same information as predX,
##           but in matrix format for plotting
##   temps, temps0, temps00: Data in matrix format,
#                             tempsX, but only list(mean)
##
## resultX: inla model outputs
## times: Overall running times for each method
## scores.observed: Assessment scores evaluated for the
## observed data;Only to be used for sanity checking.
## scores.unobserved: Assessment scores evaluated for
##                    the holdout(unbserved) data
##                    for dataset 3.
```

```
##                     Set Temp0 below to the full data for
##                     dataset 2 for comparisons.
##
## The posterior predictive distributions F_i=N(m_i, s_i^2)
## include the observation noise variance, so are appropriate
## to evaluate on "new" observations y.
##
## Mean scores: S({F_i}, {y_i}) := 1/N \sum_{i=1}^N S(F_i, y_i)
##
## All the scores are negatively oriented, i.e. small values are
## better, and all are proper, i.e. the expected score under y~G
## cannot be made smaller than when F=G (the predictive
## distribution is the same as the truth).
## For more scoring details, see Gneiting & Raftery, JASA, 2007.

## q is the 1-alpha/2 quantile of N(0,1), and alpha=0.05
##
## Remark: (y-m)^2 + s^2 is not a proper score, and should not be
## used. It's frighteningly common to see it used in the wild.
## Resist the temptation. (See G&R above for more details.)

## Unify data storage to allow the same code to
## run for all three data sets.
## Temp: properly observed observations
## Temp0: all observations, including holdout data
## Temp00: all observations not in Temp
## (Temp00 is only used when plotting results)
if (dataset == 1) {
    load(file.path(data.path, "SatelliteTemps.RData"))
    n.row <- 500
    n.col <- 300
    indata <- sat.temps
    indata$Temp0 <- indata$Temp
    extend <- -0.5
    max.edge <- 360/38820*10
} else if (dataset == 2) {
    load(file.path(data.path, "SimulatedTemps.RData"))
    n.row <- 500
    n.col <- 300
    indata <- sim.data
     ## Change to the full dataset for comparison
    indata$Temp0 <- indata$Temp
    extend <- -0.5
    max.edge <- 360/38820*10
```

```
} else {
    load(file.path(data.path, "SmallTestData.RData"))
    n.row <- 100
    n.col <- 100
    indata <- code.test
    indata$Temp <- indata$MaskedData
    indata$Temp0 <- indata$FullData
    extend <- -0.5
    max.edge <- 1/100*10
}

## Construct prior parameter distributions heuristically:
## Medium range, relative half-spread factor
prior.range <- c(max(c(diff(range(indata$Lon)),
              diff(unique(indata$Lat)))) * 1/5, 5)
## Medium sd, relative half-spread factor
prior.sigma <- c(sd(indata$Temp, na.rm=TRUE) / 2, 4)

## Extract available but "not observed" values.
indata$Temp00 <- indata$Temp0
indata$Temp00[!is.na(indata$Temp)] <- NA

## Construct centred covariate versions
LonCentre <- mean(range(indata$Lon))
LatCentre <- mean(range(indata$Lat))
indata$LonC <- indata$Lon - LonCentre
indata$LatC <- indata$Lat - LatCentre

lon <- matrix(indata$Lon,nrow=n.row)
lat <- matrix(indata$Lat,nrow=n.row)
temps <- list(mean=matrix(indata$Temp,nrow=n.row))
temps0 <- list(mean=matrix(indata$Temp0,nrow=n.row))
temps00 <- list(mean=matrix(indata$Temp00,nrow=n.row))

## Run a linear trend model

time1 <- system.time({
    formula1 <- Temp ~ 1 + LonC + LatC
    result1 <- inla(formula1, data=indata, family="gaussian",
        control.predictor=list(compute=TRUE),
        control.inla=list(strategy="gaussian", int.strategy="eb"),
        verbose=TRUE, num.threads=num.threads,
        control.compute=list(openmp.strategy=openmp.strategy))
```

```
    reconstruction1 <- data.frame(
        Lon=indata$Lon,
        Lat=indata$Lat,
        Temp=result1$summary.linear.predictor[,"mean"],
        SD=sqrt(result1$summary.linear.predictor[,"sd"]^2
                + 1/result1$summary.hyperpar[1,"0.5quant"]))

    temps1 <- list(mean=matrix(reconstruction1$Temp,nrow=n.row),
                   sd=matrix(reconstruction1$SD,nrow=n.row))
})


## Run the full model.
time2 <- system.time({
time2.prep <- system.time({
    loc2 <- cbind(indata$Lon, indata$Lat)
    mesh2 <- inla.mesh.create(loc=loc2,
                              extend=list(offset=extend),
                              refine=list(min.angle=30))
    proj2 <- inla.mesh.projector(mesh2, loc2)
    col.idx <- apply(proj2$proj$bary, 1, which.max)
    idx2 <- mesh2$graph$tv[proj2$proj$t +
           (col.idx-1)*nrow(mesh2$graph$tv)]

    ## lognormal prior
    ## Centre the parameterisation at range=1, sigma=1
    nu <- spde.alpha - 1
    kappa.zero <- sqrt(8*nu) / 1
    tau.zero <- (gamma(nu) / (gamma(spde.alpha) * 4*
                         pi * kappa.zero^(2*nu)) )^0.5 / 1
    ## sigma^2 = Gamma(0.5)/( Gamma(1.5) (4\pi)^(dim/2)
    ##             * kappa^(2*0.5) * tau^2 )
    ## tau = [ Gamma(0.5)/( Gamma(1.5) (4\pi)^(dim/2)
    ##             * kappa^(2*0.5) ) ]^0.5 / sigma
    ## kappa = sqrt(8*0.5) / range
    ## tau = [ Gamma(0.5)/( Gamma(1.5) (4\pi)^(dim/2)
    ##             * (sqrt(8*0.5)/range)^(2*0.5) ) ]^0.5 / sigma
    B.tau   <- cbind(log(tau.zero), nu, -1)
    B.kappa <- cbind(log(kappa.zero), -1, 0)

    theta.prior.mean <- log(c(prior.range[1], prior.sigma[2]))
    ## Half-width of prior prediction interval, on log-scale:
    ## 2*sd = log(rel)
    theta.prior.prec <- 4 / log(c(prior.range[2], prior.sigma[2]))^2
```

```
    spde2 <- inla.spde2.matern(mesh2, alpha=spde.alpha,
                    B.tau = B.tau, B.kappa = B.kappa,
                    theta.prior.mean = theta.prior.mean,
                    theta.prior.prec = theta.prior.prec,
                    constr=FALSE)

    hyper.range.initial <- log(prior.range[1])
    hyper.sigma.initial <- log(prior.sigma[1])
    hyper.family.initial <- 2

    spde2$f$hyper.default$theta1$initial <- hyper.range.initial
    spde2$f$hyper.default$theta2$initial <- hyper.sigma.initial

    formula2 <- Temp ~ 1 + LonC + LatC + f(field, model=spde)

    ok <- !is.na(indata$Temp) | TRUE
    data2 <- list(Temp=indata$Temp[ok], LonC=indata$LonC[ok],
                  LatC=indata$LatC[ok], field=idx2[ok], spde=spde2)
})
  time2.run <- system.time({
    result2 <- inla(formula2, data=data2, family="gaussian",
                    control.family=list(hyper=list(theta=list(
                      initial=hyper.family.initial))),
                    control.predictor=list(compute=TRUE),
                    control.inla=list(strategy="gaussian",
                                      int.strategy="eb",
                                      force.diagonal=TRUE,
                                      stupid.search=FALSE),
                    verbose=TRUE, num.threads=num.threads,
                    control.compute=list(openmp.strategy=
                                           openmp.strategy))
  })
    times2.post <- system.time({
    reconstruction2 <- data.frame(Lon=indata$Lon,
                                  Lat=indata$Lat,
                                  Temp=indata$Temp*NA,
                                  SD=indata$Temp*NA)
    reconstruction2$Temp[ok] <- result2$summary.
                                linear.predictor[,"mean"]
    reconstruction2$SD[ok] <-
        sqrt(result2$summary.linear.predictor[,"sd"]^2
             + 1/result2$summary.hyperpar[1,"0.5quant"])

    temps2 <- list(mean=matrix(reconstruction2$Temp,nrow=n.row),
```

```
                sd=matrix(reconstruction2$SD,nrow=n.row))

## Convert model parameters to humanly readable form
result2.field <- inla.spde.result(result2, "field", spde=spde2)
param2 <- rbind("nugget.sd"=vapply(2:7, function(x) {
  if (x==1) {
    1
  } else if (x==8) {
    0
  } else if (x==2) {
     inla.emarginal(function(x) x^(-0.5),
                    result2$marginals.hyperpar[[1]])
                } else if (x==3) {
                  m <- inla.emarginal(function(x) x^(-0.5),
                       result2$marginals.hyperpar[[1]])
                  inla.emarginal(function(x) (x^(-0.5) - m)^2,
                       result2$marginals.hyperpar[[1]])^0.5
                } else {
                  kk <- c(5,4,3,6)[x-3]
                  result2$summary.hyperpar[1,kk]^(-0.5)
                }
              }, 1.0))
param2 <- rbind(param2, "field.range"=vapply(2:7, function(k) {
  if (k==1 | k==8) {
    result2.field$summary.log.range.nominal[1, k]
  } else if (k==2) {
    inla.emarginal(function(x) x,
        result2.field$marginals.range.nominal[[1]])
  } else if (k==3) {
    m <- inla.emarginal(function(x) x,
        result2.field$marginals.range.nominal[[1]])
    inla.emarginal(function(x) (x - m)^2,
        result2.field$marginals.range.nominal[[1]])^0.5
  } else {
    exp(result2.field$summary.log.range.nominal[,k])
  }
}, 1.0))
param2 <- rbind(param2,
          "field.sd"=vapply(2:7, function(k) {
            if (k==1 | k==8) {
              result2.field$summary.log.variance.nominal[1, k]
            } else if (k==2) {
              inla.emarginal(function(x) x^0.5,
              result2.field$marginals.variance.nominal[[1]])
```

```
                } else if (k==3) {
                  m <- inla.emarginal(function(x) x^0.5,
                    result2.field$marginals.variance.nominal[[1]])
                    inla.emarginal(function(x) (x^0.5 - m)^2,
                    result2.field$marginals.
                    variance.nominal[[1]])^0.5
                } else {
                    exp(result2.field$summary.
                        log.variance.nominal[,k] / 2)
                }
                }, 1.0))
    colnames(param2) <- colnames(result2.field$summary.
                                 log.range.nominal)[-c(1,8)]
    param2 <- as.data.frame(param2)
    })
})

## Get a rough kriging&variance time estimate,
## and sanity check that the posterior
## median paramters give a similar result
## to the Empirical Bayes estimate.
time3 <- system.time({
  time3.setup <- system.time({
    if (TRUE) {
      ### lognormal prior
      Qx <- inla.spde2.precision(spde2,
                  theta=(c(result2$summary.hyperpar[2,"0.5quant"],
                  result2$summary.hyperpar[3,"0.5quant"])))
    } else {
      ### pcprior
      Qx <- inla.spde2.precision(spde2,
            theta=log(c(result2$summary.hyperpar[2,"0.5quant"],
            result2$summary.hyperpar[3,"0.5quant"])))
    }
    ok3 <- !is.na(indata$Temp)
    covar <- cbind(1,
                   mesh2$loc[idx2[ok3],1,drop=FALSE] - LonCentre,
                   mesh2$loc[idx2[ok3],2,drop=FALSE] - LatCentre)
    Qcovar <- Diagonal(3, c(0, 1e-8, 1e-8))
    Qprior <- rBind(cBind(Qx, Matrix(0, nrow(Qx), ncol(covar))),
                    cBind(Matrix(0, ncol(covar), nrow(Qx)), Qcovar))
    Afield <- sparseMatrix(i=1:sum(ok3), j=idx2[ok3], x=1,
                           dims=c(sum(ok3), spde2$n.spde))
    Aobs <- cBind(Afield, covar)
```

```
  Qobs <- Diagonal(sum(ok3),
                   result2$summary.hyperpar[1,"0.5quant"])
  Qpost <- Qprior + t(Aobs) %*% Qobs %*% Aobs

  Apred <- cBind(inla.spde.make.A(mesh2, loc2),
                 1,
                 loc2[,1] - LonCentre,
                 loc2[,2] - LatCentre)
})

time3.mean <- system.time({
  x3.mean <- Apred %*% inla.qsolve(Qpost, as.matrix(t(Aobs) %*%
                                   (Qobs %*% indata$Temp[ok3])))
})
time3.sd <- system.time({
  ## Split into spare (s) and dense (d) parts:
  ## A S A' = [As Ad] [Sss Ssd; Sds Sdd] [As'; Ad']
  ##        = As Sss As' + As Ssd Ad' + Ad Sds As' + Ad Sdd Ad'
  ## (A S A')_kk = \sum_{i=s,j=s} As_ki Sss_ij As_kj
  ##               +\sum_{i=s,j=d} As_ki Ssd_ij Ad_kj
  ##               +\sum_{i=d,j=s} Ad_ki Sds_ij As_kj
  ##               +\sum_{i=d,j=d} Ad_ki Sdd_ij Ad_kj
  idx.sparse <- 1:mesh2$n
  idx.dense <- mesh2$n + 1:ncol(covar)
  ## Make sure the non-zero pattern covers A' A:
  Spost <- inla.qinv(Qpost + (t(Apred) %*% Apred) * 0)
  ## Calculate (A inverse(Q) A')_kk :
  As <- Apred[,idx.sparse,drop=FALSE]
  Ad <- Apred[,idx.dense,drop=FALSE]
  AsSs <- As %*% Spost[idx.sparse, idx.sparse, drop=FALSE]
  AsSd <- As %*% Spost[idx.sparse, idx.dense, drop=FALSE]
  AdSd <- Ad %*% Spost[idx.dense, idx.dense, drop=FALSE]
  x3.sd <- (rowSums(AsSs * As) + 2*rowSums(AsSd * Ad) +
              rowSums(AdSd * Ad))^0.5
})

reconstruction3 <- data.frame(Lon=indata$Lon,
                              Lat=indata$Lat,
                              Temp=indata$Temp*NA,
                              SD=indata$Temp*NA)
reconstruction3$Temp <- x3.mean
reconstruction3$SD <-
  sqrt(x3.sd^2 + 1/result2$summary.hyperpar[1,"0.5quant"])
```

```
  temps3 <- list(mean=matrix(reconstruction3$Temp,nrow=n.row),
                 sd=matrix(reconstruction3$SD,nrow=n.row))
})

## Save Output ##
 set.names <- c("SatTemps","SimTemps","TestData")
 write.name <- paste0("./",set.names[dataset],'Fitted.RData')
 save(file=write.name,list=c("reconstruction1","reconstruction2",
                             "reconstruction3","indata",
                             "time1","time3","result1","result2"))
```

Periodic Embedding

```
# install.packages("../../npspec_0.1.0.tar.gz",
# repos = NULL, type = "source" )
library("npspec")

# read in land surface temperature data and put in a matrix
load("SimulatedTemps.RData")
tmpr <- matrix( sim.data$Temp, 500, 300 )

# get grid size
n1 <- nrow(tmpr)
n2 <- ncol(tmpr)
nvec_obs <- c(n1,n2)

# get pattern of missing values
y <- tmpr[1:nvec_obs[1],1:nvec_obs[2]]
observed <- !is.na(y)
nobs <- sum(observed)

# define locations and covariates
locs <- as.matrix( expand.grid( 1:nvec_obs[1], 1:nvec_obs[2] ) )
X <- array(NA, c(nvec_obs,3))
X[,,1] <- 1
X[,,2] <- array( locs[,1], nvec_obs)
X[,,3] <- array( locs[,2], nvec_obs)

# fit the model
t1 <- proc.time()
fit <- iterate_spec(y, observed, X = X, burn_iters = 30,
                    par_spec_fun = spec_AR1, embed_fac = 1.2,
                    precond_method = "Vecchia", m = 10,
```

```
                    silent = TRUE, ncondsim = 50)
(proc.time() - t1)/60

# predictions
pred_mat <- fit$condexp
pred_vec <- c(pred_mat)

# calculate the prediction variances based on the
# conditional simulations
cond_diff <- array(NA, dim(fit$condsim) )
for(j in 1:dim(fit$condsim)[3]) cond_diff[,,j] <- fit$condsim[,,j]
  - fit$condexp
meansq <- function(x) 1/length(x)*sum(x^2)
predvar_mat <- apply( cond_diff, c(1,2), meansq )
predvar_vec <- c(predvar_mat)

# compute the standard error of the prediction values
pe_stdev = sqrt(predvar_vec)

# compute upper and lower bounds
pe_upper <- pred_vec + 1.96*pe_stdev
pe_lower <- pred_vec - 1.96*pe_stdev

# save the results in an .RData file
save(file="./SimPeriodicEmbedding.RData",
     list=c("pred_vec","pe_stdev","pe_upper","pe_lower"))

# plots
par(mfrow=c(1,2))
fields::image.plot(pred_mat)
fields::image.plot(predvar_mat)
```

Metakriging

```
rm(list=ls())
## Libraries
library(mcmc)
library(MASS)
library(KernSmooth)
library(fields)
library(pscl)
library(mvtnorm)
library(spBayes)
```

```r
library(MCMCpack)
library(Mposterior)
library(parallel)
library(doParallel)
library(foreach)

## Simulated Data
load("SimulatedTemps.RData")
na.obs <- which(is.na(sim.data$Temp))
## Training Data
dat.nomiss <- cbind(sim.data$Lon[-na.obs],
                    sim.data$Lat[-na.obs],
                    sim.data$Temp[-na.obs])
## Test Data
dat.miss <- cbind(sim.data$Lon[na.obs],
                  sim.data$Lat[na.obs],
                  sim.data$Temp[na.obs])

## Useful Quantities
## Total number of training samples
n.sample <- nrow(dat.nomiss)
## Total number of test samples
n.test <- nrow(dat.miss)
## Number of cores or subsets
n.core <- 30
per.core <- floor(nrow(dat.nomiss)/n.core)
## Number of observations in different subsets
n.part <- c(rep(per.core,n.core-1),
            n.sample-per.core*(n.core-1))
## This is same as n.core
n.split <- length(n.part)
## Number of MCMC iterations
mcmc.sample <- 2000
## Burn in
n.burn <- 0.5*mcmc.sample
## Divide predicted data and predict them
## independently in each subset
p.sub <- c(0,seq(1000,44000,by=1000),n.test)

## GP regression on data subsets

a <- 1:n.sample
sample.loc <- dat.nomiss[,1:2]
y <- dat.nomiss[,3]
```

```
## Training response
Y.train <- y
## Training coordinates
coords.train <- sample.loc
## Test coordinates
coords.pred <- dat.miss[,1:2]
## Test sample size
n.test <- nrow(dat.miss)
index.part <- list()
X.part <- list()
Y.part <- list()
coords.part <- list()

for(i in 1:n.split){
  beg<-Sys.time()
  index.part[[i]] <- sample(a,n.part[i],replace=FALSE)
  ## Response in i th subset
  Y.part[[i]] <- Y.train[index.part[[i]]]
  ## Predictor in i th subset
  X.part[[i]] <- rep(1,n.part[i])
  ## Coordinates in i th subset
  coords.part[[i]] <- coords.train[index.part[[i]],]
  a <- setdiff(a,index.part[[i]])
}

##########################################
##Partitioned GP function
##Works with subset i, for i=1,...,n.core
##########################################

partitioned_GP <- function(i){
  ## Model fitting

  library(spBayes)
  starting <- list("phi"=3, "sigma.sq"=5, "tau.sq"=1)
  tuning <- list("phi"=0.01, "sigma.sq"=0.01, "tau.sq"=0.01)
  priors.1 <- list("beta.Norm"=list(0, 1000),
              "phi.Unif"=c(3/10, 3/0.1), "sigma.sq.IG"=c(2, 2),
              "tau.sq.IG"=c(2, 0.1))
  cov.model <- "exponential"
  ## Response in subset i
  ZZ <- Y.part[[i]]
  ## Predictor in subset i
  XX <- X.part[[i]]
```

```
  ## Coordinates in subset i
  CC <- coords.part[[i]]

  ## GP computation in each subset
  m.1 <- spLM(ZZ~XX-1, coords=CC, starting=starting,
            tuning=tuning, priors=priors.1, cov.model=cov.model,
            n.samples=mcmc.sample,verbose=FALSE)
  ## Recover all MCMC samples in each subset
  m.1.samp <- spRecover(m.1, start=n.burn+1, verbose=FALSE)
  ## List of MCMC iterates from subset i
  subAtom <- cbind(m.1.samp$p.beta.recover.samples,
                   m.1.samp$p.theta.recover.samples)
  ## Delete this quantity from the memory
  rm(m.1.samp)
  ## Garbage cleaning
  gc()

  ## Prediction
  Y.pred <- t(spPredict(m.1,
              pred.covars=as.matrix(rep(1,p.sub[2]-p.sub[1])),
              pred.coords=coords.pred[(p.sub[1]+1):p.sub[2],],
              start=0.5*mcmc.sample)$p.y.predictive.samples)

  for(l in 2:(length(p.sub)-1)){
     m.1.pred <- spPredict(m.1,
                 pred.covars=as.matrix(rep(1,p.sub[l+1]-p.sub[l])),
                 pred.coords=coords.pred[(p.sub[l]+1):p.sub[l+1],],
                 start=0.5*mcmc.sample)$p.y.predictive.samples
     Y.pred <- cbind(Y.pred,t(m.1.pred))
  }
  ## MCMC samples for both parameter estimates
  ## and prediction after burn-in
  hh <- list(subAtom,Y.pred)
  names(hh) <- c("atoms","predictions")
  return(hh)
}

####### Parallelization ########
## Number of clusters for parallel implementation
cl<-makeCluster(n.core)
registerDoParallel(cl)

## Start time
strt<-Sys.time()
```

```r
## Parallelized subset computation of GP in different cores
obj <- foreach(i=1:n.core) %dopar% partitioned_GP(i)
## Total time for parallelized inference
final.time <- Sys.time()-strt
stopCluster(cl)


####### Combine ##################

subAtomList <- list()
for(i in 1:length(n.part)){
  ## MCMC samples to run Weiszfeld algorithm
  subAtomList[[i]] <- obj[[i]]$atoms[seq(1,mcmc.sample-n.burn,10),]
}
beg.rec <- Sys.time()
## Combination using Weiszfeld algorithm
medPosterior <- findWeiszfeldMedian(subAtomList, sigma = 0.1,
                                    maxit = 100, tol = 1e-5)
## Posterior wts. using Weiszfeld's algorithm
wts <- medPosterior$weiszfeldWts
## Time for combining subset posteriors
recomb.time <- Sys.time()-beg.rec

print('Finish combining data!')

####### Prediction ################
Y.med <- numeric()
Y.lower_qnt <- numeric()
Y.upper_qnt <- numeric()
beg <- Sys.time()
## Number of posterior predictive samples
n.pred.sample <- nrow(obj[[1]]$predictions)

for(j in 1:ncol(obj[[1]]$predictions)){
  ## Posterior predictive samples for j th predicted point
  ## from first subsample
  a.sub.i<- obj[[1]]$predictions[,j]
  a.wts <- wts[1]*rep(1/n.pred.sample,n.pred.sample)
  for(i in 2:n.split){
     ## Posterior predictive samples for j th predicted point
     ## from i th subsample
     a.sub.i <- c(a.sub.i,obj[[i]]$predictions[,j])
     ## Corresponding empirical weights
     a.wts <- c(a.wts, wts[i]*rep(1/n.pred.sample,n.pred.sample))
```

```
  }
  new.atom <- sort(a.sub.i,decreasing=F,index.return=T)$ix
  ## MCMC samples for the meta posterior
  atoms <- a.sub.i[new.atom]
  ## Corresponding weights
  prob <- a.wts[new.atom]
  id11 <- min(which(cumsum(prob)>=0.025))
  id12 <- min(which(cumsum(prob)>=0.975))
  id13 <- min(which(cumsum(prob)>=0.5))
  ## Posterior 2.5% quantile for j th predicted point
  Y.lower_qnt[j] <- atoms[id11]
  ## Posterior 97.5% quantile for j th predicted point
  Y.upper_qnt[j] <- atoms[id12]
  ## Posterior median for j th predicted point
  Y.med[j] <- atoms[id13]
}

## Time for calculating quantiles
quantile.calculation.time <- Sys.time()-beg

## Total time for subset running, combining subset posteriors
## and calculating quantiles
Time <- final.time+recomb.time+quantile.calculation.time

## Save Results
res.df <- data.frame(Y.med=Y.med,
                     Y.lower_qnt=Y.lower_qnt,
                     Y.upper_qnt=Y.upper_qnt)
write.table(x=res.df,file="./MetaKrigingSimResults.txt",
            quote=FALSE,row.names=FALSE)
save(file="MetaSimTime.RData",list=c("Time"))
```

Gapfill

```
rm(list = ls())
## all packages are available on CRAN
library("gapfill")
library("doParallel");
registerDoParallel(12) # run 40 tasks in parallel
library("abind")

## load data
load("SimulatedTemps.RData")
```

```
## rearrange data as matrix
data <- with(sim.data,
             array(Temp,
                   c(length(unique(Lon)),
                     length(unique(Lat)))))
dim <- dim(data)
nx <- dim[1]; ny <- dim[2]

## display data
Image(data)

## augment data: since the gapfill method is designed for
## spatio-temproal data, we artificially create 9 additional
## and similar images by shifting the given image
data_augmented <- abind(data,
                    data[c(1,1:(nx-1)),],
                    data[,c(1,1:(ny-1))],
                    data[c(2:nx,nx),],
                    data[,c(2:ny,ny)],

                    data[c(2:nx,nx),c(2:ny,ny)],
                    data[c(2:nx,nx),c(1,1:(ny-1))],
                    data[c(1,1:(nx-1)),c(2:ny,ny)],
                    data[c(1,1:(nx-1)),c(1,1:(ny-1))],

                    data[c(3:nx,nx,nx),],
                    data[,c(3:ny,ny,ny)],
                    data[c(1,1,1:(nx-2)),],
                    data[,c(1,1,1:(ny-2))],
                    along = 3)
dim(data_augmented) <- c(nx, ny, dim(data_augmented)[3], 1)


## predict missing values
out <- Gapfill(data_augmented,
       ## only predict missing values in first (original) image
       subset = which(is.na(data_augmented[,,1,1])),
       ## use parallel processing via R package foreach
       dopar = TRUE,
       ## tuning parameters of the algorithm
       initialSize = c(2L, 2L, 100L, 100L),
       nTargetImage = 2,
       nQuant = 3,
```

```
       ## restrict values to the following range
       clipRange = range(data_augmented[,,1,1],
                         na.rm=TRUE),
       ## return prediction interval
       nPredict = 3, predictionInterval = TRUE
)

## extract prediction and prediction interval
prediction <- out$fill[,,1,1,1]
ciLo <- out$fill[,,1,1,2]
ciUp <- out$fill[,,1,1,3]

## save results to file
save(prediction, ciLo, ciUp,
     file = "SimulatedTemps_gapfill.RData")
```

laGP

```
## Submission to Matthew Heaton's "Large N Competition"
## by Robert B. Gramacy and Furong Sun, Virginia Tech

## change this to 2x the number of cores on the node
## (assumes hyper-threading; only one node is needed)
nth <- 40

## Here is what we need
library(LatticeKrig)
library(tgp)
library(laGP)
library(plgp)

## simulated problem
load("SimulatedTemps.RData")
pdf("temps_sim.pdf")
data <- sim.data
nr <- 500; nc <- 300 # number of rows and number of columns
Lon <- matrix(data$Lon, nrow=nr, ncol=nc)
Lat <- matrix(data$Lat, nrow=nr, ncol=nc)
Temp <- matrix(data$Temp, nrow=nr, ncol=nc)

## extract the training/testing problem
load(nas <- which(is.na(data$Temp))
X <- cbind(data$Lon, data$Lat)[-nas,] # training input
```

```
N <- nrow(X)
y <- data$Temp[-nas] # training resposne
XX <- cbind(data$Lon, data$Lat)[nas,] # test input
NN <- nrow(XX)

## code inputs and predictive grid to unit cube
maxX <- apply(rbind(X, XX), 2, max)
minX <- apply(rbind(X, XX), 2, min)
for (j in 1:ncol(X)){
  X[,j] <- X[,j] - minX[j]
  X[,j] <- X[,j]/(maxX[j]-minX[j])
  XX[,j] <- XX[,j] - minX[j]
  XX[,j] <- XX[,j]/(maxX[j]-minX[j])
}

##
## time for fitting
##

## macro-scale analysis on a maximum entropy
# sub-design of size n=100
n <- 100
# fi returns 100 new design locations as indices
sub <- dopt.gp(100, Xcand=X)

## priors for the global (subset) GP
da <- darg(list(mle=TRUE, max=10), X) # X-training input
ga <- garg(list(mle=TRUE, max=10), y) # y-training response

## fit the global GP
gpsepi <- newGPsep(sub$XX, y[sub$fi],
                   d=da$start, g=ga$start, dK=TRUE)
that <- mleGPsep(gpsepi, param="both", tmin=c(da$min, ga$min),
                 tmax=c(da$max, ga$max), ab=c(da$ab, ga$ab),
                 maxit=200)

## predictions from the global GP on the test set
psub <- predGPsep(gpsepi, XX, lite=TRUE)

## calculation of residuals at the full set of input locations
pX <- predGPsep(gpsepi, X, lite=TRUE)
yresid <- y - pX$mean

## done with the psub GP, clean up
```

```
deleteGPsep(gpsepi)

## Now for laGP on residuals from the (subset) global GP

## scale the inputs according to the macro-analysis lengthscales
scale <- sqrt(that$theta[1:2])
Xs <- X; XXs <- XX
for(j in 1:ncol(Xs)){
  Xs[,j] <- Xs[,j] / scale[j]
  XXs[,j] <- XXs[,j] / scale[j]
}

## local analysis on residuals (and scaled inputs)
## from local analysis
out <- aGPsep(Xs, yresid, XXs, d=list(start=1, max=20),
              g=that$theta[3],
              omp.threads=nth, verb=0)

##
## Calulations are basically done, just have to
## process some of the outputs for intervals and visualization
##

## 2x2 grid for visualization
par(mfrow=c(2,2))

## original data
image.plot(Lon, Lat, Temp, zlim=range(data$Temp, na.rm=TRUE),
           xlab="Longitude", ylab="Latitude",
           main="Original Data with Subset")
## with space-filling sub-design for global GP
points(data$Lon[-nas][sub$fi], data$Lat[-nas][sub$fi], pch=19)

## visualization of global GP predictive surface
p <- data$Temp
p[nas] <- psub$mean
PTemp <- matrix(p, nrow=nr, ncol=nc)
image.plot(Lon, Lat, PTemp, zlim=range(data$Temp, na.rm=TRUE),
           xlab="Longitude", ylab="Latitude",
           main="Global GP on Subset")

## visualiation of the global-local hybrid
w
```

```
##
## Assessing predictive uncertainty for the global-local hybrid
## requires combining uncertainty from two models: uncertainty
## in the global gp mean used to define residuals, and the
## full predictive uncertainty from the laGP on the residuals.
##

## Deriving the uncertainty in the global GP mean requires
## removing the nugget from psub variance.
library(plgp)
K <- covar.sep(sub$XX, d=that$theta[1:2], g=that$theta[3])
psi <- drop(t(y[sub$fi]) %*% solve(K) %*% y[sub$fi])
s2.f <- psub$s2 * nrow(sub$XX) / psi
eps <- sqrt(.Machine$double.eps)
s2.f <- s2.f - that$theta[3] + eps
s2.f <- s2.f * psi / nrow(sub$XX)

## combine variances from psub (from residuals) and local GP
stdev <- sqrt(s2.f + out$var)
p3 <- data$Temp
p3[nas] <- stdev
p3[-nas] <- NA
PTemp3 <- matrix(p3, nrow=nr, ncol=nc)
image.plot(Lon, Lat, PTemp3, xlab="Longitude", ylab="Latitude",
           main="Predictive SD based on Global-Local laGP")

## outputs for missing values in terms of means
## and 95% interval
pmean <- out$mean + psub$mean
pupper <- pmean + 1.96*stdev
plower <- pmean - 1.96*stdev

save(file="./SimLAGP.RData",list=c("pmean","pupper","plower"))

##
## That's all folks
##
dev.off()
```

Code for Getting the Results

This code is modified from the one provided by Heaton et al. (2018) in https://github.com/finnlindgren/heatoncomparison/blob/master/Code/GetResults/GetResults.R.

```
library(R.matlab)

########################################
## Load Simulated and Satellite Data ##
########################################
load('AllSatelliteTemps.RData')
Lon <- matrix(all.sat.temps$Lon,nrow=500)
Lat <- matrix(all.sat.temps$Lat,nrow=500)
load('AllSimulatedTemps.RData')
#Sat.true <- matrix(all.sat.temps$TrueTemp,nrow=500)
Sim.true <- matrix(all.sim.data$TrueTemp,nrow=500)
sim.miss <- which(is.na(all.sim.data$MaskTemp))
#sat.miss <- which(is.na(all.sat.temps$MaskTemp))
sim.preds <- 1:length(sim.miss)
#sat.preds <- 1:length(sat.miss)

######################
## Get NNGP Results ##
######################
load('NNGP_CONJ_SimResults.RData')
nngp_conj_pred = pred[,1]
load('nngp_postproc_result.RData')

##########################
## Get Tapering Results ##
##########################
load('SimulatedTemps_tapering_0.2.RData')
taper.sim <- list(pred=Ypred,
                  unc=preduncertainty,
                  timing=timing)
rm(list=c("delta","predCIlower","scholCall","res",
          "scholCobs","shobs","shpred","shpredobs",
          "verbose","Yconditional",
          "Ypred","preduncertainty"))

########################
## Get Gapfill Results ##
########################
load('SimulatedTemps_gapfill.RData')
gapfill.sim <- data.frame(pred=c(prediction),
                          low=c(ciLo),up=c(ciUp))
gapfill.sim$pse <-
  (gapfill.sim$up-gapfill.sim$low)/(2*qnorm(0.975))
```

```
###################################
## Get Spatial Partition Results ##
###################################
load("SimulationResults.RData")
partition.sim <- sim.results
partition.sim.tottime <- tot.time
partition.sim.liketime <- comp.time
partition.sim.predtime <- pred.time
rm(list=c("sim.results","tot.time",
          "comp.time","pred.time"))

######################
## Get FRK Results ##
######################
load("FRKSimResults.RData")

############################
## Get LatticeKrig Results ##
############################
load('simulatedNC30nlevel4finalResults.rda')
LK.sim <- finalResults

########################
## Get SPDEs Results ##
########################
load('SimTempsFitted.RData')
SPDE.sim <- reconstruction3
rm(list=c("reconstruction3","reconstruction1",
          "reconstruction2","indata"))


####################
## Get MRA Results ##
####################
mra.sim <- readMat("ResultsMrasimulated.mat")
mra.sim$predMat <- matrix(0,nrow=nrow(Lon),
                          ncol=ncol(Lon))
mra.sim$sdMat <- mra.sim$predMat
for(rw in 1:nrow(Lon)){
  ## prediction locations from MRA are in different
  ## order so need to reorder
  the.lon <- unique(Lon[rw,])
  mra.vals <- mra.sim$predMean[mra.sim$predloc[,1]==the.lon]
  mra.sd <- sqrt(mra.sim$predVariance[mra.sim$predloc[,1]==the.lon])
```

```
  mra.latvals <- mra.sim$predloc[mra.sim$predloc[,1]==the.lon,2]
  mra.vals <- mra.vals[order(mra.latvals,decreasing=TRUE)]
  mra.sd <- mra.sd[order(mra.latvals,decreasing=TRUE)]
  mra.sim$predMat[rw,] <- mra.vals
  mra.sim$sdMat[rw,] <- mra.sd
}

############################
## Get Pred Proc. Results ##
############################
load('pp_results.Rdata')

######################
## Get LAGP Results ##
######################
load("SimLAGP_results.RData")
sim.lagp <- list(mean=pmean,lower=plower,upper=pupper)
sim.lagp$sd <-
  (sim.lagp$upper-sim.lagp$lower)/(2*qnorm(0.975))

##############################
## Get Circ. Embed. Results ##
##############################
load("SimPeriodicEmbedding.RData")

####################################
############ Functions##############
####################################
crps <- function(predlist,trueobs) {
  z <- as.numeric((trueobs - predlist$mean) / predlist$sd)
  scores <- predlist$sd * (z *(2 * pnorm(z, 0, 1) - 1) +
                             2 * dnorm(z, 0, 1) - 1/sqrt(pi))
  return(scores)
}

intscore <- function(x, y, alpha=0.05) {
  hw <- -qnorm(alpha/2) * x$sd
  scores <- 2 * hw +
    (2/alpha) * (((x$mean - hw) - y) * (y < x$mean - hw) +
                   (y - (x$mean + hw)) * (y > x$mean + hw))
  return(scores)
}

cvg <- function(x, y, alpha=0.05) {
```

```
  hw <- -qnorm(alpha/2) * x$sd
  scores <- y >= (x$mean - hw) & y <= (x$mean + hw)
  return(scores)
}


##########################################
## Numerical Results for Simulated Data ##
##########################################
simres.df <- data.frame(Method=c("NNGPr","NNGPc",
               "Taper","GapFill","Partition","FRK",
        "LK","SPDE","MRA", "Pred.Proc","LAGP","CE"))

## MAE
simres.df$MAE <- c(mean(abs(y_pred_nngpresp[sim.preds]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(nngp_conj_pred[sim.preds]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(taper.sim$pred[sim.preds]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(gapfill.sim$pred[sim.miss]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(partition.sim$pred[sim.miss]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(FRKsim$results$mu[sim.preds]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(LK.sim$yHat[sim.preds]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(SPDE.sim$Temp[sim.miss]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(mra.sim$predMat[sim.miss]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(pp_results[[1]]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(sim.lagp$mean[sim.preds]-
                      Sim.true[sim.miss]),na.rm=TRUE),
                   mean(abs(pred_vec[sim.miss]-
                      Sim.true[sim.miss]),na.rm=TRUE)
)

## RMSE
simres.df$RMSE <- c(sqrt(mean((y_pred_nngpresp[sim.preds]-
                           Sim.true[sim.miss])^2,na.rm=TRUE)),
                    sqrt(mean((nngp_conj_pred[sim.preds]-
```

```
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((taper.sim$pred[sim.preds]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((gapfill.sim$pred[sim.miss]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((partition.sim$pred[sim.miss]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((FRKsim$results$mu[sim.preds]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((LK.sim$yHat[sim.preds]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((SPDE.sim$Temp[sim.miss]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((mra.sim$predMat[sim.miss]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((pp_results[[1]]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((sim.lagp$mean[sim.preds]-
                        Sim.true[sim.miss])^2,na.rm=TRUE)),
                  sqrt(mean((pred_vec[sim.miss]-
                        Sim.true[sim.miss])^2,na.rm=TRUE))
)

## CRPS
simres.df$CRPS <- c(mean(crps(list(mean=y_pred_nngpresp[sim.preds],
            sd=y_sd_nngpresp[sim.preds]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=nngp_conj_pred[sim.preds],
            sd=nngp_conj_stdev[sim.preds]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=taper.sim$pred[sim.preds],
            sd=taper.sim$unc[sim.preds]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=gapfill.sim$pred[sim.miss],
            sd=gapfill.sim$pse[sim.miss]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=partition.sim$pred[sim.miss],
            sd=partition.sim$pred.se[sim.miss]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=FRKsim$results$mu[sim.preds],
            sd=FRKsim$results$sd[sim.preds]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=LK.sim$yHat[sim.preds],
            sd=LK.sim$standError[sim.preds]),Sim.true[sim.miss]),
```

```
            na.rm=TRUE),
          mean(crps(list(mean=SPDE.sim$Temp[sim.miss],
            sd=SPDE.sim$SD[sim.miss]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=mra.sim$predMat[sim.miss],
            sd=mra.sim$sdMat[sim.miss]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=pp_results[[1]],
            sd=pp_results[[2]]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=sim.lagp$mean[sim.preds],
            sd=sim.lagp$sd[sim.preds]),Sim.true[sim.miss]),
            na.rm=TRUE),
          mean(crps(list(mean=pred_vec[sim.miss],
            sd=pe_stdev[sim.miss]),Sim.true[sim.miss]),
            na.rm=TRUE)
)

## Interval Score
simres.df$intscore <-
  c(mean(intscore(list(mean=y_pred_nngpresp[sim.preds],
                       sd=y_sd_nngpresp[sim.preds]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=nngp_conj_pred[sim.preds],
                       sd=nngp_conj_stdev[sim.preds]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=taper.sim$pred[sim.preds],
                       sd=taper.sim$unc[sim.preds]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=gapfill.sim$pred[sim.miss],
                       sd=gapfill.sim$pse[sim.miss]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=partition.sim$pred[sim.miss],
                       sd=partition.sim$pred.se[sim.miss]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=FRKsim$results$mu[sim.preds],
                       sd=FRKsim$results$sd[sim.preds]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=LK.sim$yHat[sim.preds],
                       sd=LK.sim$standError[sim.preds]),
                       Sim.true[sim.miss]),na.rm=TRUE),
            mean(intscore(list(mean=SPDE.sim$Temp[sim.miss],
                       sd=SPDE.sim$SD[sim.miss]),
                       Sim.true[sim.miss]),na.rm=TRUE),
```

```
              mean(intscore(list(mean=mra.sim$predMat[sim.miss],
                        sd=mra.sim$sdMat[sim.miss]),
                        Sim.true[sim.miss]),na.rm=TRUE),
              mean(intscore(list(mean=pp_results[[1]],
                        sd=pp_results[[2]]),
                        Sim.true[sim.miss]),na.rm=TRUE),
              mean(intscore(list(mean=sim.lagp$mean[sim.preds],
                        sd=sim.lagp$sd[sim.preds]),
                        Sim.true[sim.miss]),na.rm=TRUE),
              mean(intscore(list(mean=pred_vec[sim.miss],
                        sd=pe_stdev[sim.miss]),
                        Sim.true[sim.miss]),na.rm=TRUE)
)

## Coverage
simres.df$CVG <- c(mean(cvg(list(mean=y_pred_nngpresp[sim.preds],
                     sd=y_sd_nngpresp[sim.preds]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=nngp_conj_pred[sim.preds],
                     sd=nngp_conj_stdev[sim.preds]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=taper.sim$pred[sim.preds],
                     sd=taper.sim$unc[sim.preds]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=gapfill.sim$pred[sim.miss],
                     sd=gapfill.sim$pse[sim.miss]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=partition.sim$pred[sim.miss],
                     sd=partition.sim$pred.se[sim.miss]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=FRKsim$results$mu[sim.preds],
                     sd=FRKsim$results$sd[sim.preds]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=LK.sim$yHat[sim.preds],
                     sd=LK.sim$standError[sim.preds]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=SPDE.sim$Temp[sim.miss],
                     sd=SPDE.sim$SD[sim.miss]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=mra.sim$predMat[sim.miss],
                     sd=mra.sim$sdMat[sim.miss]),
                     Sim.true[sim.miss]),na.rm=TRUE),
                    mean(cvg(list(mean=pp_results[[1]],
                     sd=pp_results[[2]]),
```

```
                    Sim.true[sim.miss]),na.rm=TRUE),
                  mean(cvg(list(mean=sim.lagp$mean[sim.preds],
                    sd=sim.lagp$sd[sim.preds]),
                    Sim.true[sim.miss]),na.rm=TRUE),
                  mean(cvg(list(mean=pred_vec[sim.miss],
                    sd=pe_stdev[sim.miss]),
                    Sim.true[sim.miss]),na.rm=TRUE)
)
```

Appendix B

SLP Data Preprocessing

The code below demonstrates how to read in a netCDF file and reshape it into a data frame in R. The code is modified from http://geog.uoregon.edu/bartlein/courses/geog490/week04-netCDF.html.

```
# package load-in
library(ncdf4)

# open the .nc file
nc_in = nc_open('slp.mnmean.nc')
# print out metadata
print(nc_in)

# get longitude and latitude
lon <- ncvar_get(nc_in,"lon")
nlon <- dim(lon)
lat <- ncvar_get(nc_in,"lat")
nlat <- dim(lat)
# print out spatial dimension
print(c(nlon,nlat))

# get time
time <- ncvar_get(nc_in,"time")
nt=dim(time)
nt
tunits <- ncatt_get(nc_in,"time","units")
tunits

# get Sea Level Pressure (SLP) values

# attribute name
attname = 'slp' # sea level pressure

# get sea level pressures
slp_array <- ncvar_get(nc_in,attname)
dlname <- ncatt_get(nc_in,attname,"long_name")
dunits <- ncatt_get(nc_in,attname,"units")
fillvalue <- ncatt_get(nc_in,attname,"_FillValue")

# check the dimension of the array:
dim(slp_array) # 180 longitudes, 89 latitudes, 1728 months
```

```r
# reshape the netCDF 'bricks' to data frame
# load some packages
library(chron)

# convert time -- split the time units string into fields
tustr <- strsplit(tunits$value, " ")
tdstr <- strsplit(unlist(tustr)[3], "-")
tmonth <- as.integer(unlist(tdstr)[2])
tday <- as.integer(unlist(tdstr)[3])
tyear <- as.integer(unlist(tdstr)[1])
chron(time,origin=c(tmonth, tday, tyear))

# replace netCDF fill values with NA's
slp_array[slp_array==fillvalue$value] <- NA

# reshape the array into vector
slp_vec_long <- as.vector(slp_array)
length(slp_vec_long)

# reshape the vector into a matrix
slp_mat <- matrix(slp_vec_long, nrow=nlon*nlat, ncol=nt)
dim(slp_mat)

# create a wide format data frame
# create dataframe -- reshape data
# matrix (nlon*nlat rows by 2 cols) of lons and lats
lonlat <- as.matrix(expand.grid(lon,lat))
dim(lonlat)

# reshape
slp_df <- data.frame(cbind(lonlat,slp_mat))
# rename the columns of the data frame
names(slp_df) <- c("lon","lat",paste0("Month ", 1:1728))

# create a long format data frame
# lonlat2 = do.call(rbind, replicate(1728, lonlat, simplify=FALSE))
# month = rep(rep(1:12, each = nlon*nlat),144)
# slp_df2 = data.frame(cbind(lonlat2,
#                            slp = slp_vec_long,
#                            month=month))
# names(slp_df2)[1:2] = c('lon','lat')

# save the resulting data frame
save(file = 'slp_data.Rdata', list = 'slp_df')
```

```
# save(file = 'slp_data_long.RData', list = 'slp_df2')
```

NNGP Conjugate Models for SLPs from Land Areas

```
rm(list=ls())
# import library
library(spNNGP)

# load-in the SLP data
load('slp_data.Rdata')
#########################
## data pre-processing ##
#########################
data = slp_df
rm(slp_df)
# find out the locations with NAs
# same for all time stamps
nas <- which(is.na(data[,3]))

# training data coordinates
coords = as.matrix(data[,1:2][-nas,,drop = FALSE])
# training data, design matrix X for trend surface model
X =  as.matrix(cbind(1,coords))
# training data, response (temp)
y = as.matrix(data[,850][-nas,drop=FALSE])

# test data coordinates
coords.ho = as.matrix(data[,1:2][nas,,drop = FALSE])
# test data, covariates for trend surface model
X.ho = as.matrix(cbind(1,coords.ho))

############################
##  Intercept-only model ##
############################

# useful arguments
cov.model <- "exponential" # also tried 'spherical'
sigma.sq <- 5
sigma.sq.IG <- c(2, sigma.sq)
 g <- 5
 theta.alpha0 <- as.matrix(expand.grid(seq(0.0001, 0.02,
                                           length.out=g),
                     seq(0.001/sigma.sq, 0.01/sigma.sq,
```

```
                                          length.out=g)))
 colnames(theta.alpha0) <- c("phi", "alpha")

# create the matrix to store the predicted values
pred.mat = matrix(0, nrow = dim(coords.ho)[1],
                  ncol = dim(data)[2]-2)
std.mat = matrix(0, nrow = dim(coords.ho)[1],
                 ncol = dim(data)[2]-2)

# a loop to populate the pred.mat

for (i in 1:(dim(data)[2]-2)){
  # update the response values based on time stamps
  y = as.matrix(data[,i+2][-nas,drop=FALSE])

  # model fit and select the theta.alpha best estimate
  m.c <- spConjNNGP(y~1, coords=coords, n.neighbors = 10,
              k.fold = 5, score.rule = "crps",
              n.omp.threads = 3,
              theta.alpha = theta.alpha0,
              sigma.sq.IG = sigma.sq.IG,
              cov.model = cov.model)
  # store the selected theta.alpha
  theta.alpha = as.vector(m.c$theta.alpha)
  names(theta.alpha) <- c("phi", "alpha")

  # used the selected theta.alpha to predict
  m.p <- spConjNNGP(y~1, coords=coords, n.neighbors = 10,
                    X.0=as.matrix(rep(1,dim(coords.ho)[1])),
                    coords.0=coords.ho,
                    n.omp.threads = 10,
                    theta.alpha = theta.alpha,
                    sigma.sq.IG = sigma.sq.IG,
                    cov.model = cov.model)

  # predicted sea level pressure
  pred.mat[,i] <- m.p$y.0.hat
  std.mat[,i] <- sqrt(m.p$y.0.hat.var)
}

# save the prediction output matrices
save(file = 'SLP_NNGP_intercept.RData',
     list = c('pred.mat','std.mat'))
```

```
##########################
## Trend Surface Model ##
##########################
# create the matrix to store the predicted values
pred.mat2 = matrix(0, nrow = dim(coords.ho)[1],
                   ncol = dim(data)[2]-2)
std.mat2 = matrix(0, nrow = dim(coords.ho)[1],
                  ncol = dim(data)[2]-2)

# a loop to populate the pred.mat

for (i in 1:(dim(data)[2]-2)){
  # update the response values based on time stamps
  y = as.matrix(data[,i+2][-nas,drop=FALSE])

  # model fit and select the theta.alpha best estimate
  m.c2 <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                     k.fold = 5, score.rule = "crps",
                     n.omp.threads = 3,
                     theta.alpha = theta.alpha0,
                     sigma.sq.IG = sigma.sq.IG,
                     cov.model = cov.model)
  # store the selected theta.alpha
  theta.alpha2 = as.vector(m.c2$theta.alpha)
  names(theta.alpha2) <- c("phi", "alpha")

  # prediction
  m.p2 <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                      X.0=X.ho, coords.0=coords.ho,
                      n.omp.threads = 10,
                      theta.alpha = theta.alpha2,
                      sigma.sq.IG = sigma.sq.IG,
                      cov.model = cov.model)

  # predicted sea level pressure
  pred.mat2[,i] <- m.p2$y.0.hat
  std.mat2[,i] <- sqrt(m.p$y.0.hat.var)
}

# save the prediction output matrices
save(file = 'SLP_NNGP_trend.RData',
     list = c('pred.mat2','std.mat2'))
```

```
##################################
## Adding month as a covariate ##
##################################
rm(list=ls())
# import the libraries
library(spNNGP)

# load-in the SLP data
load('slp_data.Rdata')
############################
## data pre-processing ###
############################
nrow = dim(slp_df)[1]
nmonth = dim(slp_df)[2] - 2
nas <- which(is.na(slp_df[,3]))

full.pred.mat = matrix(0, nrow = length(nas), ncol = nmonth)
full.sd.mat = matrix(0, nrow = length(nas), ncol = nmonth)

# create composite datset iteratively and
# fit FRK model
for (i in 1:nmonth){

  data = slp_df[,c(1,2)]
  data$month = numeric(nrow)
  data$slp = numeric(nrow)

  which.month = sample(1:nmonth, nrow, replace = TRUE)
  data$month = which.month %% 12
  month12.index = which(data$month == 0)
  data$month[month12.index] = 12
  data$month = as.factor(data$month)

  for (j in 1:nrow){
    data$slp[j] = slp_df[j, which.month[j]+2]
  }

  # find out the locations with NAs
  # same for all time stamps
  nas <- which(is.na(data$slp))

  # training data coordinates
  coords = as.matrix(data[,1:2][-nas,,drop = FALSE])
  # training data, design matrix X for trend surface model
```

```
  X =  as.matrix(cbind(1,data$month[-nas]))
  # training data, response (temp)
  y = as.matrix(data$slp[-nas,drop=FALSE])

  # test data coordinates
  coords.ho = as.matrix(data[,1:2][nas,,drop = FALSE])
  # test data, covariates for trend surface model
  X.ho = as.matrix(cbind(1,data$month[nas]))

  # useful arguments
  cov.model <- "spherical"
  sigma.sq <- 5
  sigma.sq.IG <- c(2, sigma.sq)
  g <- 5
  theta.alpha0 <- as.matrix(expand.grid(seq(0.0001, 0.02,
                                            length.out=g),
                        seq(0.001/sigma.sq, 0.01/sigma.sq,
                            length.out=g)))
  colnames(theta.alpha0) <- c("phi", "alpha")

  # model fit and select the theta.alpha best estimate
  m.c3 <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                  k.fold = 5, score.rule = "crps",
                  n.omp.threads = 4,
                  theta.alpha = theta.alpha0,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

  theta.alpha3 = as.vector(m.c3$theta.alpha)
  names(theta.alpha3) <- c("phi", "alpha")

  # prediction
  m.p3 <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                   X.0=X.ho, coords.0=coords.ho,
                   n.omp.threads = 10,
                   theta.alpha = theta.alpha3,
                   sigma.sq.IG = sigma.sq.IG,
                   cov.model = cov.model)

  full.pred.mat[,i] <- m.p3$y.0.hat
  full.sd.mat[,i] <- sqrt(m.p3$y.0.hat.var)
}

save(file = 'nngp_conj_month.RData',
```

```
    list = c('full.pred.mat','full.sd.mat'))
```

FRK Models for SLPs from Land Areas

```
####################
## Intercept-Only ##
####################

## Load packages
library(FRK)
library(sp)
#library(ggplot2)
#library(gridExtra)
library(INLA)
#library(splancs)

load('slp_data.Rdata')

runFRK <- function(df, f = slp ~ lon + lat, tol = 0.02) {

  ## Make BAUs as SpatialPixels
  BAUs <- df
  BAUs$Missing <- is.na(BAUs$slp)
  BAUs$slp <- NULL
  BAUs$fs <- 1     # set fs variation to unity
  coordinates(BAUs)  <- ~lon+lat # to SpatialPointsDataFrame
  gridded(BAUs) <- TRUE     # to SpatialPixelsDataFrame

  ## Make Data as SpatialPoints
  dat <- subset(df,!is.na(slp)) # no missing data in data frame
  coordinates(dat)  <- ~lon+lat # Convert to SpatialPointsDataFrame

  basis <- auto_basis(plane(),      # we are on the plane
                      data = dat,   # around which to make basis
                      regular = 0,  # irregular basis
                      nres = 3,     # 3 resolutions
                      scale_aperture = 1
                )

  nas = which(is.na(df$slp))
  ## Remove basis functions in problematic region

  if(nrow(df) == 16020) {
```

```
  basis_df <- data.frame(basis)
  rmidx <- which( (((basis_df$loc2 < -73 ) &
          (basis_df$loc1 > 0 & basis_df$loc1 < 358 )) |
          ((basis_df$loc2 > -25 & basis_df$loc2 < 25 ) &
          (basis_df$loc1 > 10 & basis_df$loc1 < 40)) |
          ((basis_df$loc2 > 25 & basis_df$loc2 < 70 ) &
          (basis_df$loc1 > 60 & basis_df$loc1 < 120)) |
          ((basis_df$loc2 > 30 & basis_df$loc2 < 70 ) &
          (basis_df$loc1 > 230 & basis_df$loc1 < 270)) |
          ((basis_df$loc2 > -40 & basis_df$loc2 < 5 ) &
          (basis_df$loc1 > 290 & basis_df$loc1 < 320)) |
          ((basis_df$loc2 > 12.5 & basis_df$loc2 < 45 ) &
          (basis_df$loc1 > 338 & basis_df$loc1 < 358))) &
          (basis_df$res == 1 | basis_df$res == 2 |
                                      basis_df$res == 3) )
  basis <- remove_basis(basis,rmidx)
}

## Estimate using Maximum Likelihood (ML)
S <- FRK(f = f,             # formula for SRE model
         data = dat,        # data
         basis = basis,     # Basis
         BAUs = BAUs,       # BAUs
         tol = tol)         # EM iterations


## Predict
BAUs_pred <- predict(S)  # predict over all BAUs
BAUs_pred_df <- data.frame(BAUs_pred) # to dataframe

## Compute variance of predicted observations
## and conf. intervals
BAUs_pred_df$sd_obs <- sqrt(BAUs_pred_df$sd^2 +
                                      S@Ve[1,1])
BAUs_pred_df$conflo <- BAUs_pred_df$mu -
                        1.96*BAUs_pred_df$sd_obs
BAUs_pred_df$confhi <- BAUs_pred_df$mu +
                       1.96*BAUs_pred_df$sd_obs

## Only return missing data and relevant variables
Res <- subset(BAUs_pred_df, Missing==TRUE)
Res$Missing <- Res$fs <- Res$var <- NULL

## Return in a list
```

```
  list(results = Res,
       SREmodel = S)
}

# find out locations with missing values
nas = which(is.na(slp_df[,3]))

mu.mat.frk =  matrix(0, nrow = length(nas),
                     ncol = dim(slp_df)[2]-2)
sd.mat.frk =  matrix(0, nrow = length(nas),
                     ncol = dim(slp_df)[2]-2)

for (i in 1:(dim(slp_df)[2]-2))
  {
    # subset the data
    data_in = slp_df[,c(1,2,i+2)]
    colnames(data_in)[3] = 'slp'
    out = runFRK(data_in, f = slp ~ 1)
    mu.mat.frk[,i] = out$results$mu
    sd.mat.frk[,i] = out$results$sd_obs
  }

save(file = 'slp_frk_intercept.RData',
     list = c('mu.mat.frk','sd.mat.frk'))

####################
## Trend Surface ##
####################

## Load packages
library(FRK)
library(sp)
#library(ggplot2)
#library(gridExtra)
library(INLA)
#library(splancs)

load('slp_data.Rdata')

runFRK <- function(df, f = slp ~ lon + lat, tol = 0.02) {

  ## Make BAUs as SpatialPixels
  BAUs <- df
  BAUs$Missing <- is.na(BAUs$slp)
```

```
BAUs$slp <- NULL
BAUs$fs <- 1       # set fs variation to unity
coordinates(BAUs)  <- ~lon+lat # to SpatialPointsDataFrame
gridded(BAUs) <- TRUE  # to SpatialPixelsDataFrame

## Make Data as SpatialPoints
dat <- subset(df,!is.na(slp))
coordinates(dat)  <- ~lon+lat # to SpatialPointsDataFrame

basis <- auto_basis(plane(),
                    data = dat,
                    regular = 0,
                    nres = 3,
                    scale_aperture = 1
)

nas = which(is.na(df$slp))
## Remove basis functions in problematic region

if(nrow(df) == 16020) {
  basis_df <- data.frame(basis)
  rmidx <- which( (((basis_df$loc2 < -73 ) &
            (basis_df$loc1 > 0 & basis_df$loc1 < 358 )) |
            ((basis_df$loc2 > -25 & basis_df$loc2 < 25 ) &
            (basis_df$loc1 > 10 & basis_df$loc1 < 40)) |
            ((basis_df$loc2 > 25 & basis_df$loc2 < 70 ) &
            (basis_df$loc1 > 60 & basis_df$loc1 < 120)) |
            ((basis_df$loc2 > 30 & basis_df$loc2 < 70 ) &
            (basis_df$loc1 > 230 & basis_df$loc1 < 270)) |
            ((basis_df$loc2 > -40 & basis_df$loc2 < 5 ) &
            (basis_df$loc1 > 290 & basis_df$loc1 < 320)) |
            ((basis_df$loc2 > 12.5 & basis_df$loc2 < 45 ) &
            (basis_df$loc1 > 338 & basis_df$loc1 < 358))) &
            (basis_df$res == 1 | basis_df$res == 2 |
                              basis_df$res == 3) )
  basis <- remove_basis(basis,rmidx)
}

## Estimate using Maximum Likelihood (ML)
S <- FRK(f = f,             # formula for SRE model
         data = dat,        # data
         basis = basis,     # Basis
         BAUs = BAUs,       # BAUs
         tol = tol)         # EM iterations
```

```
  ## Predict
  BAUs_pred <- predict(S)
  BAUs_pred_df <- data.frame(BAUs_pred) # to dataframe

  ## Compute variance of predicted observations and
  #  conf. intervals
  BAUs_pred_df$sd_obs <- sqrt(BAUs_pred_df$sd^2 +
                                      S@Ve[1,1])
  BAUs_pred_df$conflo <- BAUs_pred_df$mu -
                         1.96*BAUs_pred_df$sd_obs
  BAUs_pred_df$confhi <- BAUs_pred_df$mu +
                         1.96*BAUs_pred_df$sd_obs

  ## Only return missing data and relevant variables
  Res <- subset(BAUs_pred_df, Missing==TRUE)
  Res$Missing <- Res$fs <- Res$var <- NULL

  ## Return in a list
  list(results = Res,
       SREmodel = S)
}

mu.mat.frk2 =  matrix(0, nrow = length(nas),
                      ncol = dim(slp_df)[2]-2)
sd.mat.frk2 =  matrix(0, nrow = length(nas),
                      ncol = dim(slp_df)[2]-2)

for (i in 1:(dim(slp_df)[2]-2))
{
  # subset the data
  data_in = slp_df[,c(1,2,i+2)]
  colnames(data_in)[3] = 'slp'
  out = runFRK(data_in)
  mu.mat.frk2[,i] = out$results$mu
  sd.mat.frk2[,i] = out$results$sd_obs
}

save(file = 'slp_frk_trend.RData',
     list = c('mu.mat.frk2','sd.mat.frk2'))

########################
## Month as Covariate ##
```

```
########################

## Load packages
library(FRK)
library(sp)
#library(ggplot2)
#library(gridExtra)
library(INLA)
#library(splancs)


# define the runFRK() function
runFRK <- function(df, f = slp ~ month, tol = 0.02) {

  ## Make BAUs as SpatialPixels
  BAUs <- df
  BAUs$Missing <- is.na(BAUs$slp)
  BAUs$slp <- NULL
  BAUs$fs <- 1
  coordinates(BAUs)  <- ~lon+lat
  gridded(BAUs) <- TRUE

  df$month <- NULL

  ## Make Data as SpatialPoints
  dat <- subset(df,!is.na(slp))
  coordinates(dat)  <- ~lon+lat

  basis <- auto_basis(plane(),
                      data = dat,
                      regular = 0,
                      nres = 3,
                      scale_aperture = 1
  )


  ## Remove basis functions in problematic region

  if(nrow(df) == 16020) {
    basis_df <- data.frame(basis)
    rmidx <- which( (((basis_df$loc2 < -73 ) &
              (basis_df$loc1 > 0 & basis_df$loc1 < 358 )) |
              ((basis_df$loc2 > -25 & basis_df$loc2 < 25 ) &
              (basis_df$loc1 > 10 & basis_df$loc1 < 40)) |
```

```
            ((basis_df$loc2 > 25 & basis_df$loc2 < 70 ) &
            (basis_df$loc1 > 60 & basis_df$loc1 < 120)) |
            ((basis_df$loc2 > 30 & basis_df$loc2 < 70 ) &
            (basis_df$loc1 > 230 & basis_df$loc1 < 270)) |
            ((basis_df$loc2 > -40 & basis_df$loc2 < 5 ) &
            (basis_df$loc1 > 290 & basis_df$loc1 < 320)) |
            ((basis_df$loc2 > 12.5 & basis_df$loc2 < 45 ) &
            (basis_df$loc1 > 338 & basis_df$loc1 < 358))) &
            (basis_df$res == 1 | basis_df$res == 2 |
                                  basis_df$res == 3) )
    basis <- remove_basis(basis,rmidx)
  }

  ## Estimate using Maximum Likelihood (ML)
  S <- FRK(f = f,
           data = dat,
           basis = basis,
           BAUs = BAUs,
           tol = tol)


  ## Predict
  BAUs_pred <- predict(S)
  BAUs_pred_df <- data.frame(BAUs_pred)

  ## Compute variance of predicted observations and
  #  conf. intervals
  BAUs_pred_df$sd_obs <- sqrt(BAUs_pred_df$sd^2 +
                                S@Ve[1,1])
  BAUs_pred_df$conflo <- BAUs_pred_df$mu -
                      1.96*BAUs_pred_df$sd_obs
  BAUs_pred_df$confhi <- BAUs_pred_df$mu +
                      1.96*BAUs_pred_df$sd_obs

  ## Only return missing data and relevant variables
  Res <- subset(BAUs_pred_df, Missing==TRUE)
  Res$Missing <- Res$fs <- Res$var <- NULL

  ## Return in a list
  list(results = Res,
       SREmodel = S)
}

# load-in the SLP data
```

```
load('slp_data.Rdata')

nrow = dim(slp_df)[1]
nmonth = dim(slp_df)[2] - 2
nas = which(is.na(slp_df[,3]))

mu.mat.frk3 =  matrix(0, nrow = length(nas),
                      ncol = nmonth)
sd.mat.frk3 =  matrix(0, nrow = length(nas),
                      ncol = nmonth)

# create composite datset iteratively and
# fit FRK model
for (i in 1:nmonth){

  data = slp_df[,c(1,2)]
  data$month = numeric(nrow)
  data$slp = numeric(nrow)

  which.month = sample(1:nmonth, nrow,
                       replace = TRUE)
  data$month = which.month %% 12
  month12.index = which(data$month == 0)
  data$month[month12.index] = 12
  data$month = as.factor(data$month)

  for (j in 1:nrow){
    data$slp[j] = slp_df[j, which.month[j]+2]
  }

  out = runFRK(data)
  mu.mat.frk3[,i] = out$results$mu
  sd.mat.frk3[,i] = out$results$sd_obs

}

save(file = 'slp_frk_month.RData',
     list = c('mu.mat.frk3','sd.mat.frk3'))
```

Visualizations for SLP Interpolations on Land Areas

```
##############################
## Semivariograms  Figure 1 ##
```

```
##############################
library(gstat)
load('slp_data.Rdata')

# remove the rows with NA temperature
observed = slp_df[!is.na(slp_df[,3]),]

# obtain the empirical semi-variogram
# using one of the months as a representative
empirical <- variogram(observed[,800]~1,
                loc= ~lon+lat, data=observed)

# select a variogram model from
# exponential, Matern, Gaussian or
# Spherical structures
fit.variogram(empirical, vgm(c("Exp", "Mat",
                               "Gau", "Sph")))
# Spherical structure was selected

# fit different variogram models
fit.variogram(empirical,vgm("Exp"))
fit.variogram(empirical,vgm("Mat"))
fit.variogram(empirical,vgm("Gau"))
fit.variogram(empirical,vgm("Sph"))

# create variograms based on the
# previously estimated parameters
mod.exp = vgm(psill=112, model="Exp",
              nugget=0.0001, range=46)
mod.mat = vgm(psill=112, model="Mat",
              nugget=0.0001, range=46, kappa = 0.5)
mod.gau = vgm(psill=81, model="Gau",
              nugget=0.0001, range=21)
mod.sph = vgm(psill=88, model="Sph",
              nugget=0.0001, range=64)

par(mfrow=c(2,2))
plot(empirical, model = mod.exp, col = 'red',
     main = 'Semivariograms Using Exponential
     Variogram Model')
plot(empirical, model = mod.mat, col = 'purple',
     main = 'Semivariograms Using Matern
     Variogram Model')
plot(empirical, model = mod.gau, col = 'blue',
```

```
     main = 'Semivariograms Using Gaussian
     Variogram Model')
plot(empirical, model = mod.sph, col = 'black',
     main = 'Semivariograms Using Spherical
     Variogram Model')


##############################
## Semivariograms Figure 4 ##
##############################
load('slp_data.Rdata')
# composite one dataset with month
data = slp_df[,c(1,2)]
  data$month = numeric(nrow)
  data$slp = numeric(nrow)

  which.month = sample(1:nmonth, nrow,
                       replace = TRUE)
  data$month = which.month %% 12
  month12.index = which(data$month == 0)
  data$month[month12.index] = 12
  data$month = as.factor(data$month)

  for (j in 1:nrow){
    data$slp[j] = slp_df[j, which.month[j]+2]
  }

# remove the rows with NA temperature
observed = slp_df[!is.na(slp_df[,3]),]
observed_comp = data[-nas,]

# for Jan 1854
empirical = variogram(observed[,3]~1,
                loc= ~lon+lat, data=observed)
# for composite dataset
empirical_comp = variogram(observed_comp$slp~1,
          locations = ~lon+lat, data=observed_comp)
empirical_comp2 = variogram(observed_comp$slp~month,
          locations = ~lon+lat, data=observed_comp)

plot(empirical, main='Empirical Semivariogram
     from Jan,1854')
plot(empirical_comp, main='Empirical Semivariogram
     from Composite Dataset')
```

```
plot(empirical_comp2, main='Empirical Semivariogram
     from Composite Dataset adjusted for Month')

##############
## Figure 2 ##
##############

library(ggplot2)
library(ggpubr)

# load the raw data and prediction outputs
load('slp_data.Rdata')
load('SLP_NNGP_intercept.RData')
load('SLP_NNGP_trend.RData')
load('SLP_NNGP_intercept2.RData')
load('SLP_NNGP_trend2.RData')
load('nngp_conj_month.RData')

load('slp_frk_intercept.RData')
load('slp_frk_trend.RData')
load('slp_frk_month.RData')

# mark the indices of NAs
nas <- which(is.na(slp_df[,3]))
# Aggregate the raw data by mean over 1728 months
data0 = cbind(slp_df[,c(1,2)],
        slp = apply(slp_df[,-c(1,2)],1,mean))

########################################
## prepare the data for visualization ##
########################################

# get a copy of the original dataset for
# intercept-only NNGP model (exponential)
data_nngp1 = slp_df
# impute the missing values with the predicted values
data_nngp1[nas, -c(1,2)] = pred.mat
# aggregate the data by mean
data_nngp1 = cbind(data_nngp1[,c(1,2)],
              slp = apply(data_nngp1[,-c(1,2)],1,mean))

# get a copy of the original dataset for
# trend surface NNGP model (exponential)
data_nngp2 = slp_df
```

```
# impute the missing values with the predicted values
data_nngp2[nas, -c(1,2)] = pred.mat2
# aggregate the data by mean
data_nngp2 = cbind(data_nngp2[,c(1,2)],
              slp = apply(data_nngp2[,-c(1,2)],1,mean))

# get a copy of the original dataset for
# intercept-only NNGP model (spherical)
data_nngp3 = slp_df
# impute the missing values with the predicted values
data_nngp3[nas, -c(1,2)] = pred.mat3
# aggregate the data by mean
data_nngp3 = cbind(data_nngp3[,c(1,2)],
              slp = apply(data_nngp3[,-c(1,2)],1,mean))

# get a copy of the original dataset for surface
# trend NNGP model (spherical)
data_nngp4 = slp_df
# impute the missing values with the predicted values
data_nngp4[nas, -c(1,2)] = pred.mat4
# aggregate the data by mean
data_nngp4 = cbind(data_nngp4[,c(1,2)],
              slp = apply(data_nngp4[,-c(1,2)],1,mean))

# get a copy of the original dataset for
# intercept-only model (spherical)
data_nngp5 = slp_df
# impute the missing values with the predicted values
data_nngp5[nas, -c(1,2)] = full.pred.mat
# aggregate the data by mean
data_nngp5 = cbind(data_nngp5[,c(1,2)],
              slp = apply(data_nngp5[,-c(1,2)],1,mean))

# get a copy of the original dataset for intercept
# only FRK model
data_frk1 = slp_df
# impute the missing values with the predicted values
data_frk1[nas,-c(1,2)] = mu.mat.frk
# aggregate the data by mean
data_frk1 = cbind(data_frk1[,c(1,2)],
            slp = apply(data_frk1[,-c(1,2)],1,mean))

# get a copy of the original dataset for trend
# surface FRK model
```

```
data_frk2 = slp_df
# impute the missing values with the predicted values
data_frk2[nas,-c(1,2)] = mu.mat.frk2
# aggregate the data by mean
data_frk2 = cbind(data_frk2[,c(1,2)],
           slp = apply(data_frk2[,-c(1,2)],1,mean))

# get a copy of the original dataset for FRK
# model with month as a covariate
data_frk3 = slp_df
# impute the missing values with the predicted values
data_frk3[nas,-c(1,2)] = mu.mat.frk3
# aggregate the data by mean
data_frk3 = cbind(data_frk3[,c(1,2)],
            slp = apply(data_frk3[,-c(1,2)],1,mean))


# unify the range of the color ramp

zlimits = range(c(data0$slp[-nas], data_nngp1$slp,
                  data_nngp2$slp, data_nngp3$slp,
                  data_nngp4$slp, data_nngp5$slp,
                  data_frk1$slp, data_frk2$slp,
                  data_frk3$slp))

# 1. The original graph with NA for monthly slp,
plot0 <- ggplot(data0) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("A. Sea Level Pressure Data")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# 2. intercept only nngp (exponential)

plot_nngp1 <- ggplot(data_nngp1) +
```

```
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("B. Sea Level Pressure from
          Intercept-Only NNGP Model (exponential)")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# trend surface nngp (exponential)

plot_nngp2 <- ggplot(data_nngp2) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("C. Sea Level Pressure from
          Trend Surface NNGP Model (exponential)")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))


# intercept only nngp (spherical)

plot_nngp3 <- ggplot(data_nngp3) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
```

```
  ggtitle("D. Sea Level Pressure from
          Intercept-Only NNGP Model (spherical)")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# trend surface nngp (spherical)

plot_nngp4 <- ggplot(data_nngp4) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("E. Sea Level Pressure from Trend
          Surface NNGP Model (spherical)")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# month nngp (spherical)

plot_nngp5 <- ggplot(data_nngp5) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("F. Sea Level Pressure from NNGP Model
          using Month as a Covariate")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))
```

```r
# intercept only frk

plot_frk1 <- ggplot(data_frk1) +
  geom_point(aes(x=lon, y=lat,
                 colour = slp)) +
  scale_color_distiller(palette = "RdYlBu",
                        name = 'SLP (mb)',
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("G. Sea Level Pressure from
          Intercept-Only FRK Model")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# trend surface frk
plot_frk2 <- ggplot(data_frk2) +
  geom_point(aes(x=lon, y=lat,
                 colour = slp)) +
  scale_color_distiller(palette = "RdYlBu",
                        name = 'SLP (mb)',
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("H. Sea Level Pressure from Trend
          Surface FRK Model")+
  theme(plot.title = element_text(size = 9),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# month frk
plot_frk3 <- ggplot(data_frk3) +
  geom_point(aes(x=lon, y=lat,
                 colour = slp)) +
  scale_color_distiller(palette = "RdYlBu",
                        name = 'SLP (mb)',
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
```

```
  ggtitle("I. Sea Level Pressure from FRK Model
          using Month as a Covariate")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# align the plots together for comparison
ggarrange(plot0, plot_nngp1, plot_nngp2,
          plot_nngp3, plot_nngp4, plot_nngp5,
          plot_frk1, plot_frk2, plot_frk3,
          nrow = 3, ncol = 3)

##############
## Figure 3 ##
##############

# mark the indices of NAs
nas <- which(is.na(slp_df[,3]))
# Aggregate the raw data by mean over 1728 months
data0 = cbind(slp_df[,c(1,2)],
              slp = apply(slp_df[,-c(1,2)],1,mean))

###################################
## Prediction sd visualizations ##
###################################

# prediction sd for intercept-only NNGP model (exponential)
sd_nngp1 = cbind(data0[nas,c(1,2)],
                 sd = apply(std.mat, 1, mean))

# prediction sd for trend surface NNGP model (exponential)
sd_nngp2 = cbind(data0[nas,c(1,2)],
                 sd = apply(std.mat2, 1, mean))

# prediction sd for intercept-only NNGP model (spherical)
sd_nngp3 = cbind(data0[nas,c(1,2)],
                 sd = apply(std.mat3, 1, mean))

# prediction sd for trend surface NNGP model (spherical)
sd_nngp4 =cbind(data0[nas,c(1,2)],
                sd = apply(std.mat4, 1, mean))
```

```
# prediction sd for NNGP model with month as covariate
sd_nngp5 = cbind(data0[nas,c(1,2)],
                 sd = apply(full.sd.mat, 1, mean))

# prediction sd for intercept-only FRK model
sd_frk1 = cbind(data0[nas,c(1,2)],
                sd = apply(sd.mat.frk, 1, mean))

# prediction sd for trend surface FRK model
sd_frk2 = cbind(data0[nas,c(1,2)],
                sd = apply(sd.mat.frk2, 1, mean))

# prediction sd for FRK model with month as covariate
sd_frk3 = cbind(data0[nas,c(1,2)],
                sd = apply(sd.mat.frk3, 1, mean))

# unify the range of the color ramp

zlimits = range(c(sd_nngp1$sd, sd_nngp2$sd,
                  sd_nngp3$sd, sd_nngp4$sd,
                  sd_nngp5$sd, sd_frk1$sd,
                  sd_frk2$sd))

plot_nngp1_sd <- ggplot(sd_nngp1) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("A. Prediction Standard Errors from
          Intercept-Only NNGP Model (exponential)")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_nngp2_sd <- ggplot(sd_nngp2) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
```

```
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("B. Prediction Standard Errors from
          Trend Surface NNGP Model (exponential)")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_nngp3_sd <- ggplot(sd_nngp3) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("C. Prediction Standard Errors from
          Intercept-Only NNGP Model (spherical)")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_nngp4_sd <- ggplot(sd_nngp4) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("D. Prediction Standard Errors from
          Trend Surface NNGP Model (spherical)")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_nngp5_sd <- ggplot(sd_nngp5) +
```

```
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("E. Prediction Standard Errors from
          NNGP Model using Month as a Covariate")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_frk1_sd <- ggplot(sd_frk1) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("F. Prediction Standard Errors from
          Intercept-Only FRK Model")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_frk2_sd <- ggplot(sd_frk2) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("G. Prediction Standard Errors from
          Trend Surface FRK Model")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
```

```
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

plot_frk3_sd <- ggplot(sd_frk3) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)",
                        limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("H. Prediction Standard Errors from FRK
          Model using Month as a Covariate")+
  theme(plot.title = element_text(size = 8),
        axis.title = element_text(size=7),
        legend.text = element_text(size=7),
        legend.title = element_text(size=7),
        legend.key = element_rect(size=0.5))

# align the plots together for comparison
ggarrange(plot_nngp1_sd, plot_nngp2_sd, plot_nngp3_sd,
          plot_nngp4_sd, plot_nngp5_sd,
          plot_frk1_sd, plot_frk2_sd, plot_frk3_sd,
          nrow = 3, ncol = 3)
```

NNGP Conjugate Models for SLPs from Ocean Areas

The code chunk below shows the case of aggregation before model fits.

```
library(spNNGP)
library(ggplot2)
library(ggpubr)

####################################
## Prepare SLP dataset for Ocean ##
####################################

load('slp_data.Rdata')
# find the rows correspond to land areas
# (with NAs in the original dataset)
land_index = which(is.na(slp_df[ ,3]))
# remove the land related rows
slp_ocean = slp_df[-land_index,]
```

```r
# randomly choose about 20% of
# the observations to be held out
set.seed(2019)
hold_out = sample(1:dim(slp_ocean)[1],
                  round(dim(slp_ocean)[1]*0.2))

# prepare for the training dataset
# the annual mean SLP values from
# year 1854 to year 1980 (127 years)
train = slp_ocean[ ,1:(127*12+2)]
# remove the hold-out set
train = train[-hold_out, ]
# aggregate the training dataset
# annual mean
train_agg = cbind(train[,c(1,2)],
            slp = apply(train[,-c(1,2)],1,mean))

# prepare for the testing datset
# the annual mean SLP values from
# year 1990 to year 1997 (8 years)
# skipping 1981-1989, 9 years
test = slp_ocean[ ,-(3:((127+9)*12+2))]
# subset the hold-out set
test = test[hold_out, ]
true_slp = test[,3:ncol(test)]
true_slp_agg = apply(true_slp, 1, mean)
test[,3:ncol(test)] = NA

# useful arguments
# training data coordinates (fixed)
coords = as.matrix(train[,1:2])
# training data, design matrix X for trend surface model
X =  as.matrix(cbind(1,coords))
y = train_agg$slp

# test data coordinates
coords.ho = as.matrix(test[,1:2])
# test data, covariates for trend surface model
X.ho = as.matrix(cbind(1,coords.ho))

# other useful arguments
cov.model <- "spherical"
sigma.sq <- 5
sigma.sq.IG <- c(2, sigma.sq)
```

```
g <- 5
theta.alpha0 <- as.matrix(expand.grid(seq(0.0001, 0.02,
                                          length.out=g),
                  seq(0.001/sigma.sq, 0.01/sigma.sq,
                                            length.out=g)))
colnames(theta.alpha0) <- c("phi", "alpha")

############################
## Intercept-only Model ##
############################
system.time({
m.c <- spConjNNGP(y~1, coords=coords, n.neighbors = 10,
                  k.fold = 5, score.rule = "crps",
                  n.omp.threads = 3,
                  theta.alpha = theta.alpha0,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

# store the selected theta.alpha
theta.alpha = as.vector(m.c$theta.alpha)
names(theta.alpha) <- c("phi", "alpha")

# used the selected theta.alpha to predict
m.p <- spConjNNGP(y~1, coords=coords, n.neighbors = 10,
                  X.0=as.matrix(rep(1,dim(coords.ho)[1])),
                  coords.0=coords.ho,
                  n.omp.threads = 10,
                  theta.alpha = theta.alpha,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

pred.ocean3 = m.p$y.0.hat
std.ocean3 = sqrt(m.p$y.0.hat.var)

})

############################
## Trend Surface Model ##
############################

system.time({
m.c1 <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                  k.fold = 5, score.rule = "crps",
                  n.omp.threads = 3,
```

```
                  theta.alpha = theta.alpha0,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

# store the selected theta.alpha
theta.alpha = as.vector(m.c1$theta.alpha)
names(theta.alpha) <- c("phi", "alpha")

# used the selected theta.alpha to predict
m.p1 <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                  X.0=X.ho,
                  coords.0=coords.ho,
                  n.omp.threads = 10,
                  theta.alpha = theta.alpha,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

pred.ocean4 = m.p1$y.0.hat
std.ocean4 = sqrt(m.p1$y.0.hat.var)

})

##############################
## Compute Numerical scores ##
## to evaluate predictive   ##
## performances             ##
##############################

########################################
## Functions from Heaton et al. (2018) ##
########################################
crps <- function(predlist,trueobs) {
  z <- as.numeric((trueobs - predlist$mean) / predlist$sd)
  scores <- predlist$sd * (z *(2 * pnorm(z, 0, 1) - 1) +
                           2 * dnorm(z, 0, 1) - 1/sqrt(pi))
  return(scores)
}

intscore <- function(x, y, alpha=0.05) {
  hw <- -qnorm(alpha/2) * x$sd
  scores <- 2 * hw + (2/alpha) * (((x$mean - hw) - y) *
                                    (y < x$mean - hw) +
                                    (y - (x$mean + hw)) *
                                    (y > x$mean + hw))
```

```
  return(scores)
}

cvg <- function(x, y, alpha=0.05) {
  hw <- -qnorm(alpha/2) * x$sd
  scores <- y >= (x$mean - hw) & y <= (x$mean + hw)
  return(scores)
}

#####################################
## Scores from Intercept-only NNGP ##
#####################################

mae1 = mean(abs(pred.ocean3 - true_slp_agg))
rmse1 = sqrt(mean((pred.ocean3 - true_slp_agg)^2))
crps1 = mean(crps(list(mean=pred.ocean3,
                       sd=std.ocean3),true_slp_agg))
int1 = mean(intscore(list(mean=pred.ocean3,
                          sd=std.ocean3),true_slp_agg))
cvg1 = mean(cvg(list(mean=pred.ocean3,
                     sd=std.ocean3),true_slp_agg))

#####################################
## Scores from Trend Surface NNGP ##
#####################################

mae2 = mean(abs(pred.ocean4 - true_slp_agg))
rmse2 = sqrt(mean((pred.ocean4 - true_slp_agg)^2))
crps2 = mean(crps(list(mean=pred.ocean4,
                       sd=std.ocean4),true_slp_agg))
int2 = mean(intscore(list(mean=pred.ocean4,
                          sd=std.ocean4),true_slp_agg))
cvg2 = mean(cvg(list(mean=pred.ocean4,
                     sd=std.ocean4),true_slp_agg))
```

The code chunk below shows the case of aggregation after the model fits.

```
rm(list=ls())
library(spNNGP)

#####################################
## Prepare SLP dataset for Ocean ##
#####################################
```

```r
load('slp_data.Rdata')
# find the rows correspond to land areas
# (with NAs in the original dataset)
land_index = which(is.na(slp_df[ ,3]))

# remove the land related rows
slp_ocean = slp_df[-land_index,]

# randomly choose about 20% of
# the observations to be held out
set.seed(2019)
hold_out = sample(1:dim(slp_ocean)[1],
                  round(dim(slp_ocean)[1]*0.2))

# prepare for the training dataset
# the annual mean SLP values from
# year 1854 to year 1980 (127 years)
train = slp_ocean[ ,1:(127*12+2)]
# remove the hold-out set
train = train[-hold_out, ]

# prepare for the testing datset
# the annual mean SLP values from
# year 1990 to year 1997 (8 years)
# skipping 1981-1989, 9 years
test = slp_ocean[ ,-(3:((127+9)*12+2))]
# subset the hold-out set
test = test[hold_out, ]
true_slp = test[,3:ncol(test)]
test[,3:ncol(test)] = NA

#####################################
## prepare useful argument values ##
#####################################

# training data coordinates (fixed)
coords = as.matrix(train[,1:2])
# training data, design matrix X for trend surface model
X =  as.matrix(cbind(1,coords))

# test data coordinates
coords.ho = as.matrix(test[,1:2])
# test data, covariates for trend surface model
X.ho = as.matrix(cbind(1,coords.ho))
```

```
# other useful arguments
cov.model <- "spherical"
sigma.sq <- 5
sigma.sq.IG <- c(2, sigma.sq)
g <- 5
theta.alpha0 <- as.matrix(expand.grid(seq(0.0001, 0.02,
                                          length.out=g),
                      seq(0.001/sigma.sq, 0.01/sigma.sq,
                                         length.out=g)))
colnames(theta.alpha0) <- c("phi", "alpha")

############################
## Intercept-only Models ##
############################

# create the matrix to store the predicted annual
# mean of 1990-1997 for each model trained used
# one of the timestamp from 1854-1980
pred.ocean = matrix(0, nrow = dim(test)[1],
                    ncol = dim(train)[2]-2)
std.ocean = matrix(0, nrow = dim(test)[1],
                   ncol = dim(train)[2]-2)

# try to fit a model and make predictions for 98 times

total_time = 0

for (i in 1: (dim(pred.ocean)[2])){

# assign the response variable
y = as.matrix(train[,i+2])

m.c <- spConjNNGP(y~1, coords=coords, n.neighbors = 10,
                  k.fold = 5, score.rule = "crps",
                  n.omp.threads = 3,
                  theta.alpha = theta.alpha0,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

# store the selected theta.alpha
theta.alpha = as.vector(m.c$theta.alpha)
names(theta.alpha) <- c("phi", "alpha")
```

```
# used the selected theta.alpha to predict
m.p <- spConjNNGP(y~1, coords=coords, n.neighbors = 10,
                  X.0=as.matrix(rep(1,dim(coords.ho)[1])),
                  coords.0=coords.ho,
                  n.omp.threads = 10,
                  theta.alpha = theta.alpha,
                  sigma.sq.IG = sigma.sq.IG,
                  cov.model = cov.model)

# if to make predictions at hold-out locations
# for 98 timestamps (months from 1990-1997),
# the predicted values will be the same
# for all 98 months, and the pooled
# prediction variance will be the same too

# populate the output matrices
pred.ocean[,i] = m.p$y.0.hat
std.ocean[,i] = sqrt(m.p$y.0.hat.var)

# update the total time in (Min)
total_time = total_time + m.c$run.time[3]/60 +
                            m.p$run.time[3]/60
}


# save the output matrices
save(file = 'SLP_NNGP_ocean_intercept.RData',
     list = c('pred.ocean','std.ocean','total_time'))

############################
## Trend Surface Models ##
############################

# create the matrix to store the predicted annual
# mean of 1990-1997 for each model trained used
# one of the timestamp from 1854-1980
pred.ocean2 = matrix(0, nrow = dim(test)[1],
                     ncol = dim(train)[2]-2)
std.ocean2 = matrix(0, nrow = dim(test)[1],
                    ncol = dim(train)[2]-2)

# try to fit a model and make predictions for 98 times

total_time2 = 0
```

```
for (i in 1:(dim(pred.ocean2)[2])){

  y = as.matrix(train[,i+2])

  m.c <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                    k.fold = 5, score.rule = "crps",
                    n.omp.threads = 3,
                    theta.alpha = theta.alpha0,
                    sigma.sq.IG = sigma.sq.IG,
                    cov.model = cov.model)

  # store the selected theta.alpha
  theta.alpha = as.vector(m.c$theta.alpha)
  names(theta.alpha) <- c("phi", "alpha")

  # used the selected theta.alpha to predict
  m.p <- spConjNNGP(y~X-1, coords=coords, n.neighbors = 10,
                    X.0=X.ho,
                    coords.0=coords.ho,
                    n.omp.threads = 10,
                    theta.alpha = theta.alpha,
                    sigma.sq.IG = sigma.sq.IG,
                    cov.model = cov.model)

  # if to make predictions at hold-out locations
  # for 98 timestamps (months from 1990-1997),
  # the predictions and prediction SEs will be the same
  # for all 98 months

  # If aggregation by taking the average across time,
  # nothing will change. So skip the aggregation steps here.
  # Note for prediction SEs, this way of aggregation
  # ignores the correlation between

  # populate the output matrices
  pred.ocean2[,i] = m.p$y.0.hat
  std.ocean2[,i] = sqrt(m.p$y.0.hat.var)

  # update the total time in (Min)
  total_time2 = total_time2 + m.c$run.time[3]/60 +
                              m.p$run.time[3]/60
}
```

```r
# save the output matrices
save(file = 'SLP_NNGP_ocean_trend.RData',
     list = c('pred.ocean2','std.ocean2','total_time2'))


##########################################
## Code for numerical score calculations ##
##########################################

rm(list=ls())
load('SLP_NNGP_ocean_intercept.RData')
load('SLP_NNGP_ocean_trend.RData')

load('slp_data.Rdata')
# find the rows correspond to land areas
# (with NAs in the original dataset)
land_index = which(is.na(slp_df[ ,3]))

# remove the land related rows
slp_ocean = slp_df[-land_index,]

# randomly choose about 20% of
# the observations to be held out
set.seed(2019)
hold_out = sample(1:dim(slp_ocean)[1],
                  round(dim(slp_ocean)[1]*0.2))
test = slp_ocean[ ,-(3:((127+9)*12+2))]
# subset the hold-out set
test = test[hold_out, ]
true_slp = test[,3:ncol(test)]

##################
## Annual means ##
##################

# summarize the true SLPs for the hold-out set
# compute the annual mean by taking the average over time
true_slp_agg = apply(true_slp, 1, mean)

# summarize the predicted values
# and pooled prediction standard errors
pred_ocean_agg = apply(pred.ocean, 1, mean)
std_ocean_agg = sqrt(apply(std.ocean^2, 1, mean))
```

```
pred_ocean_agg2 = apply(pred.ocean2, 1, mean)
std_ocean_agg2 = sqrt(apply(std.ocean2^2, 1, mean))

# compute the numerical scores to evaluate the
# predictive performances for these two NNGP models

##############################################
## Run functions from Heaton et al. (2018) ##
## for CRPS, INT and CVG                    ##
##############################################

####################################
## scores for intercept-only model ##
####################################

mae1 = mean(abs(pred_ocean_agg - true_slp_agg))
rmse1 = sqrt(mean((pred_ocean_agg - true_slp_agg)^2))
crps1 = mean(crps(list(mean=pred_ocean_agg,
                       sd=std_ocean_agg),true_slp_agg))
int1 = mean(intscore(list(mean=pred_ocean_agg,
                     sd=std_ocean_agg),true_slp_agg))
cvg1 = mean(cvg(list(mean=pred_ocean_agg,
                sd=std_ocean_agg),true_slp_agg))


####################################
## scores for surface-trend model ##
####################################

mae2 = mean(abs(pred_ocean_agg2 - true_slp_agg))
rmse2 = sqrt(mean((pred_ocean_agg2 - true_slp_agg)^2))
crps2 = mean(crps(list(mean=pred_ocean_agg2,
                       sd=std_ocean_agg2),true_slp_agg))
int2 = mean(intscore(list(mean=pred_ocean_agg2,
                          sd=std_ocean_agg2),true_slp_agg))
cvg2 = mean(cvg(list(mean=pred_ocean_agg2,
                     sd=std_ocean_agg2),true_slp_agg))
```

Visualizations for SLP Interpolations on Ocean Areas

```
#################################################
## Visualizations for Predicted SLP Annual Means ##
## Figure 5                                     ##
```

```
#####################################################

zlimits = range(c(true_slp_agg, pred.ocean3, pred.ocean4))

# original true data
data0 = cbind(test[,1:2], slp = true_slp_agg)
plot0 <- ggplot(data0) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)", limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("A. Sea Level Pressure True Annual Means (hold-out set),
          1990 - 1997")+
  theme(plot.title = element_text(size = 16),
        axis.title = element_text(size=13),
        legend.text = element_text(size=13),
        legend.title = element_text(size=13),
        legend.key = element_rect(size=0.9))

# From intercept-only NNGP Conjugate
data1 = cbind(test[,1:2], slp = pred.ocean3)
plot1 <- ggplot(data1) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
                        name = "SLP (mb)", limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("B. Predicted SLP Annual Means (hold-out set),
          1990 - 1997 from Intercept-only NNGP Conjugate Model")+
  theme(plot.title = element_text(size = 16),
        axis.title = element_text(size=13),
        legend.text = element_text(size=13),
        legend.title = element_text(size=13),
        legend.key = element_rect(size=0.9))

# From intercept-only NNGP Conjugate
data2 = cbind(test[,1:2], slp = pred.ocean4)
plot2 <- ggplot(data2) +
  geom_point(aes(x = lon,y = lat,
                 colour = slp))+
  scale_color_distiller(palette = "RdYlBu",
```

```
                        name = "SLP (mb)", limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("C. Predicted SLP Annual Means (hold-out set),
          1990 - 1997 from Trend Surface NNGP Conjugate Model")+
  theme(plot.title = element_text(size = 16),
        axis.title = element_text(size=13),
        legend.text = element_text(size=13),
        legend.title = element_text(size=13),
        legend.key = element_rect(size=0.9))

ggarrange(plot0, plot1, plot2,
          nrow = 3, ncol = 1)

########################################
## Visualizations for Prediction SEs ##
## Figure 6                           ##
########################################

zlimits = range(c(std.ocean3, std.ocean4))

sd_data1 = cbind(test[,1:2], sd = std.ocean3)
plot3 <- ggplot(sd_data1) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)", limits=zlimits) +
  xlab("Longitude (degree east)") +
  ylab("Latitude (degree north)") +
  ggtitle("A. Predicted SEs for SLP Annual Means (hold-out set),
          1990 - 1997 from Intercept-only NNGP Conjugate Model")+
  theme(plot.title = element_text(size = 15),
        axis.title = element_text(size=13),
        legend.text = element_text(size=13),
        legend.title = element_text(size=13),
        legend.key = element_rect(size=0.9))

sd_data2 = cbind(test[,1:2], sd = std.ocean4)
plot4 <- ggplot(sd_data2) +
  geom_point(aes(x = lon,y = lat,
                 colour = sd))+
  scale_color_distiller(palette = "PRGn",
                        name = "SLP (mb)", limits=zlimits) +
  xlab("Longitude (degree east)") +
```

```
  ylab("Latitude (degree north)") +
  ggtitle("B. Predicted SEs for SLP Annual Means (hold-out set),
          1990 - 1997 from Trend Surface NNGP Conjugate Model")+
  theme(plot.title = element_text(size = 15),
        axis.title = element_text(size=13),
        legend.text = element_text(size=13),
        legend.title = element_text(size=13),
        legend.key = element_rect(size=0.9))

ggarrange(plot3, plot4,
          nrow = 2, ncol = 1)
```